Université de Montréal

Deep Learning for Video Modelling

par
Olivier Mastropietro

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Août, 2017

Résumé

Ce mémoire de maîtrise présente une exploration des modèles génératifs dans le contexte de la vidéo. Ceci a demandé une étude approfondie des problèmes encourus par les chercheurs dans cette branche de la vision par ordinateur. Ce mémoire établi deux axes problématiques, celui venant des données et celui des modèles.

Concernant les données, les méthodes accomplissant l'état-de-l'art dans ce domaine sont appliqués sur des bases de données qui potentiellement sous représentent les défis existant dans les vidéos de tous les jours. Ainsi, il est possible que l'innovation évolue ultimement vers des cul-de-sacs et une nouvelle bases de données est suggérées afin de résoudre ce problème.

Quant aux modèles, la génération de vidéos est à la frontière des applications des procéssus génératifs. C'est un champs de recherche encore très ouvert aux découvertes de tailles car non seulement est-il devant des obstacles d'ingénieries, tant aux niveaux logiciels que physiques, mais il se trouve à être un véritable casse-tête. En apprentissage profond, la modélisation d'images statiques entre présentement dans une phase plus mature, mais qu'en est-il pour des séquences d'images et de leurs générations? De très récents modèles ont réussi d'impressionnantes générations image par image et exhibent de longues séquences sans dégradation rapide de la qualité visuelle. En analysant ceux-ci, ce mémoire propose le modèle *feature flow* comme un choix raisonnable à considérer pour cette tâche et espère convaincre pourquoi.

La génération comme sujet d'étude elle-même a fait également l'objet d'une attention particulière à travers ce mémoire. Il augmente le déjà populaire *generative adversarial networks* avec un mécanisme d'inférence, *adversarially learned inference.* Cette version améliorée excelle aux mêmes tâches que son prédécesseur tout en offrant une représentation abstraite des données grâce au mécanisme d'inférence. Il y a espoir lors de travaux futures d'exhiber tout son potentiel, l'élevant comme un choix de modèle important.

Mots clés: réseaux de neurones, apprentissage machine, apprentissage profond, intelligence artificielle, vision par ordinateur, vidéos, modèles génératifs.

Abstract

This thesis presents an exploration of generative models in the context of video generation. It focuses on an investigation of the problems faced by researchers when working on this branch of computer vision. It is argued throughout this thesis that video suffers from two main issues, namely on the data side and on the model side.

Data-wise, current state-of-the-art models in this field are applied on datasets that can potentially misrepresent the true challenges with real videos and pushes model innovations in corners that could be dead ends on this task. A new dataset is proposed in light of this situation that tries to fix these problems.

Model-wise, video generation is on the very frontier of generative applications. It represents an area still very open for breakthrough since not only is it faced with engineering, hardware and software obstacles, it also offers a real puzzle for models. If deep learning modelling for static images is entering a more mature phase, how does one transition to a sequence of images and moreover generate them? Very recent models have yielded impressive next frame generations and are able to show long sequences of frames that do not rapidly degrade. This thesis proposes the feature flow model as a natural choice to consider when doing this task and hope to reasonably argue as to why.

Generation as an object of study itself has also been given attention throughout this thesis. It augments the already popular *generative adversarial networks* with an inference mechanism, *adversarially learned inference.* This upgraded version excels at the same tasks than its predecessor while offering an abstract representation of its data through the inference procedure. There is hope for a display of its full potential in future works setting it as a strong model choice.

Keywords: neural networks, machine learning, deep learning, artificial intelligence, computer vision, videos, generative models.

Acknowledgements

I would like to thank many people who have made this thesis, and more broadly the master's degree, a real attainable goal.

First, my parents for without their continual support and their value in education I would not be here.

Next, the Montreal institute for learning algorithms where I had the chance to do this master's degree. A particular thanks to Vincent Dumoulin from whom I discovered the field of artificial intelligence and deep learning and my supervisor Aaron Courville who gave a chance to this random physicist showing up at his door.

A special thanks goes to the Theano staff that made possible the implementations of all the models that somehow always had *scan* doing something weird. Pascal Lamblin has especially been patient from my never really programmed start to RNN tweaking transition.

I have met many people at MILA, but four people really made this trip worthwhile. Faruk Ahmed for helping me bother the old main lab, César Laurent for the many 1pm coffee break, Ishmael Belghazi for taking my mind off coding reminding me of the wonders of mathematics and finally, my mentor Nicolas Ballas whom had to put up with my countless questions.

Lastly, to my trout fisherman friends. Even if the universe is deterministic or stochastic, if my free will is but a comfortable illusion and moral values somehow always end up being a human invention, they make life colourful and my motivation to go forward only grows from our endless discussions and interactions.

List of Figures

List of Tables

List of Abbreviations

The list of abbreviations is ordered as they appear in the text.

MILA	Montreal institute for learning algorithms
CNN	Convolutional neural network
Conv3D	3D convolutional neural network
RNN	Recurrent neural network
LSTM	Long-short term memory network
ConvLSTM	Convolutional long-short term memory network
PGM	Probabilistic graphical models
NADE	Neural autoregressive distribution estimation
PixelRNN	Pixel recurrent neural network
PixelVAE	Pixel variational autoencoder
MCMC	Markov chain monte carlo
KL	Kullback-Leibler divergence
VAE	Variational autoencoder
GAN	Generative adversarial networks
JSD	Jensen-Shannon divergence
SOTA	State-of-the-art
ALI	Adversarially learned inference
SVHN	Street view house numbers
CelebA	Large-scale CelebFaces Attributes
InfoGAN	Information maximizing GAN
BiGAN	Bidirectional GAN
SVM	Support vector machine
KTH	Kungliga Tekniska Högskolan dataset
UCF101	University of Central Florida 101 action categories dataset
HMDB	Human motion database
ILSVRC	ImageNet large scale visual recognition challenge
MNIST	Modified national institute of standards and technology dataset
OF	Optical flow
iDT	Improved dense trajectory
TDD	Trajectory-pooled deep-convolutional descriptor
TSN	Temporal segment networks

KVMF	Key volume mining deep framework
GRU-RCN	Gated recurrent unit recurrent convolution networks
LTC	Long-term temporal convolutions
DrNet	Disentangled representations network
CDNA	Convolutional dynamic neural advection
RALI	Recurrent adversarially learned inference
FF	Feature flow

Contents

Chapter 1

Introduction

This thesis will present the results of a master's degree worth of research into the area of deep learning, a sub-field of machine learning itself a sub-field of artificial intelligence. The focus of this research is the study of video modelling as a broad topic. There is a dip in using deep learning methods for the task of activity recognition, but the strong emphasis is on using generative type of models. There is also an inquiry on the nature of these types of models themselves.

This document is divided as follow: It begins with a short introduction to set the stage with the many parts of deep learning that were used throughout the thesis. It proceeds to the generative modelling landscape presenting its main inhabitants and their various strong and weak attributes. From this field, it then presents a conference paper in which I am a co-author addressing some of these points. It finishes with two chapters discussing my foray into videos, first on the discriminative side and lastly on the generative side. Both sides are accompanied first with a review of existing work and followed next with proposed solutions to remaining unsolved problems.

This thesis assumes as a starting point a certain knowledge in machine learning with its core concepts and their mathematical backgrounds.

All graphs, figures and tables were done by myself unless mentioned otherwise where clear attributions are provided.

It is written in the "companionate we" where it will return only briefly on a few occasions into first-person.

Chapter 2

Deep Learning

This chapter will introduce the building blocks of deep learning used throughout the thesis. It is not meant to be a comprehensive detailed tour of the different core pieces used in the various models presented later. For in-depth explanation of the theory behind the field, we kindly refer the avid reader to the *Deep Learning* book [Goodfellow, Bengio, and Courville, 2016].

In order not to overwhelm the presentation with citations and since all materials ahead is inspired by the textbook, we will not cite all the original sources of the following work, unless not referenced in the book.

2.1 Introduction

Deep learning at the core is the act of stacking layers of neurons that propagate successive function's output as input for a function above, all the way up from data as initial input to an arbitrary final output. An equation simply written as:

$$\mathbf{y} = f_I(f_{I-1}(...f_2(f_1(\mathbf{x}))...)) \tag{2.1}$$

summarizes surprisingly well the application of a deep neural network for output tensor $\mathbf{y}$, a depth of I different f_i functions and input tensor $\mathbf{x}$.

2.1.1 Simple network

The most simple network is the fully connected one shown in Figure 2.1. Such graphs are useful for a mental picture of what is going on. Each circle is a neuron, except for the very first layer where they are better understood as values for each dimension of the input. Each arrow represents a learnable parameter of the network also known as a weight w.

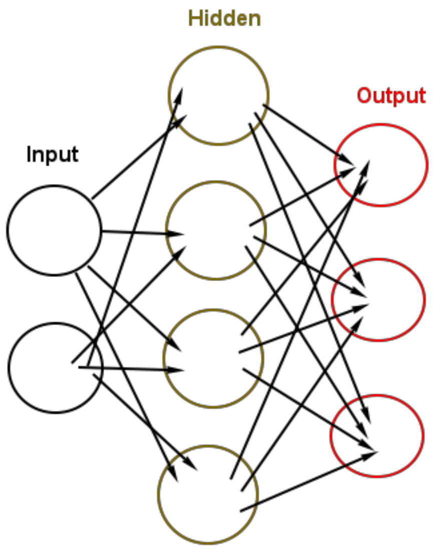


Figure 2.1: Fully connected network. This picture shows a neural net with an input vector of 2 dimensions, 4 neurons in the hidden layer and 3 neurons as output. This illustrates a very simple case where the hidden neurons could be doing a *sigmoid* activation and the final neurons could be a *softmax* function that outputs a probability distribution for example. Note that this cannot be categorized as deep, it would need at least another hidden layer!

A neuron, in this very artificial scenario, is simply an elementwise application of a function to the incoming values. There is usually a bias term b that is added at each neuron. All the weights can be easily condensed in a matrix $\mathbf{W}$ where each column is for all parameters leading to a neuron. More formally, assuming an input vector $\mathbf{x}$, a weight matrix $\mathbf{W}$, a bias vector $\mathbf{b}$ and an elementwise function's application f (called activation function) at depth i, a single layer's processing can be written as

$$\mathbf{h} = f_i(\mathbf{W}^\top \mathbf{x} + \mathbf{b}) \tag{2.2}$$

So *deep* is achieved by using multiple layers and the most simple deep architecture is called a multilayer perceptron. It can be represented simply by stacking multiple hidden layers between input and output layers. The final output neurons are doing the trick here. Indeed, it is used to represent multiple different mathematical objects depending on the task. Using *sigmoid* output function gives a probability distribution, *softmax* for multiclassification, *relu* for a positive only output... so on and so forth.

A fully connected layer is backed by a powerful property called the *universal* *function approximator theorem.* This theorem states, under mild assumptions regarding the activation function, that any continuous function on compact subsets of $\mathbb{R}^n$ can be approximated given the appropriate parameters. The main drawback is that it doesn't offer guidance onto what values these parameters should take or on how to algorthmicaly learn them.

2.1.2 Black boxes

We have seen deep but what is learning? *Learning* is done by the algorithm known as *backpropagation.* Backpropagation means nothing more than the application of the chain rule of derivatives on an error function (loss) $\mathcal{L}(\mathbf{y})$ on Equation 2.1. The exact form of $\mathcal{L}$ will be defined by a task on a dataset. Each layer f_i is parametrized by a set of K parameters $\boldsymbol{\theta}^i$ and each parameter θ^i_k is learned by differentiating the loss with respect to its value and updating it accordingly

$$\hat{\theta}^i_k \leftarrow \theta^i_k - \eta\ \partial\mathcal{L}(\mathbf{y})/\partial\theta^i_k \tag{2.3}$$

for new value $\hat{\theta}^i_k$, current value θ^i_k and scalar η.

The problem of learning then becomes one of minimization of an error function by gradient descent. Optimization is a very active field of research and has a wealth of literature that will not be dwelt upon, it will be used as a black box throughout this thesis.

A second very important black box that will also not be covered in detail in this text is *regularization.* Regularization is, to quote section 5.2.2 from the *Deep Learning* book, "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." In other words, it can be thought of as modifying the belief in a model so it does not depend only on the training data. This will be done by applying some prior or constraints on it for the sake of its ability to generalize. Many different strategies exist to achieve this and as optimization, it is another bubbling area of research.

We will go over one example so it is easier to understand the concept. An early regularization technique, before the rise of deep learning, consisted in parameters constraint. It is done by extending the training objective:

$$\hat{\mathcal{L}}(\mathbf{y}, \boldsymbol{\theta}) = \mathcal{L}(\mathbf{y}) + \alpha\Omega(\boldsymbol{\theta}) \tag{2.4}$$

for scalar α controlling the strength of the regularization term Ω. The two constraints usually used are L^2: $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{W}\|_2^2$ and L^1: $\Omega(\boldsymbol{\theta}) = \|\mathbf{W}\|_1 = \sum_i |w_i|$ commonly known as *weight decay.* Note that this is done only on the weights $\mathbf{W}$

and not on the whole parameter set $\boldsymbol{\theta}$. Weight decay has many interpretations. L^2 can be seen on an optimization point of view as keeping the parameters contributing the most to the gradients while reducing the value of others. L^1 will tend to create a network with more sparse activation which can be a desirable property. Both, if the weight decay pushes the values down to zero, can also act as keeping the activations in their linear regime and prevent vanishing gradient (with a *sigmoid* for example).

2.2 Convolutional neural network

Even though the all-mighty universal function approximation theorem backs all neural networks, the theorem never specifies the amount of neurons needed to express the data-generating function one is trying to model. Without infinite data and infinite training time, most of the game is played by imposing priors or inductive biases on the neural network architectures or how it processes information from the data. One great success of this design incentive was the invention of convolutional neural network (CNN).

2.2.1 Convolution

CNN changes Equation 2.2 for

$$\mathbf{h} = f_i(\mathbf{W}^\top * \mathbf{x} + \mathbf{b}) \tag{2.5}$$

where $*$ is the *convolution* operator. The convolution operator is mathematically defined for continuous real values t and kernel w as:

$$\begin{aligned} s(t) &= (x * w)(t) \\ &= \int x(a)w(t-a)da \\ &= \sum_{a=-\infty}^{\infty} x(a)w(t-a) \\ &= (w * x)(t) = \sum_{a=-\infty}^{\infty} x(t-a)w(a) \end{aligned}$$

The third line is the discretization of the second line and the fourth is the commutative property. When working with images, the last line can be expanded to its most widely used implementation and application in standard neural network software library:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.6}$$

There are two finite sums because we now apply a discrete convolution on a two-dimensional input, namely an image of finite pixels height and width. The learnable parameters linking Equation 2.6 with $\mathbf{W}$ of Equation 2.5 are the kernel values $K(m,n)$. Notice the $+$ instead of $-$, this subtle sign switch is referred as kernel flipping. The function of Equation 2.6 is technically called *cross-correlation*, but since K is learned, it has no impact whether we flip the kernel or not and by historical convention we shall keep writing the word convolution.

The output $S(i,j)$ is called the *feature map* and there are usually multiple kernels of small size applied to the input to create a stack of feature maps. Important technical terms that will keep showing up are: *strides*, represent the overlap in the kernel application or sliding pattern, *padding*, arbitrary border values that are added to input this way controlling the output size and *channels*, an additional input's dimension along height and width[1]. See Figure 2.2 for a 1D example.

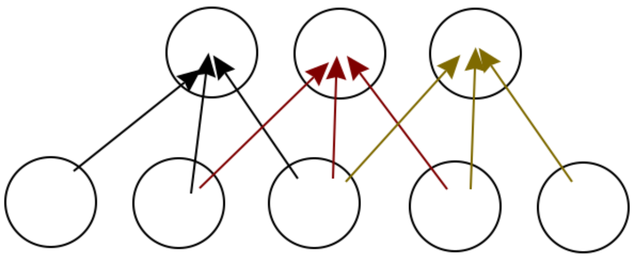


Figure 2.2: 1 dimensional convolutional network. Its graph looks very much like a fully connected layer but with only local interactions and parameters sharing in the output. This is illustrated with different arrow colours, each triplet of arrows are the *same* parameters being applied successively. In practice, the terminology of this CNN would be: 1 channel, 1 feature map, kernel size of 3, strides of 1 and padding of 0. More kernels could be slid on the same input to increase the number of feature maps.

2.2.2 Pooling

A convolutional neural network typically uses the convolution operator in conjunction with a pooling operation. Pooling is a function which returns a statistic of its input. Many different statistics can be chosen, but the two most common are max

[1]Channel is easily understood with an example. For a colour image, the colour values are specified in the 3 channels (RGB).

and average pooling. The key is to apply pooling in small windows on the input just the kernels are in convolution. This enforces the learned representation to be approximately invariant to small translations. This is an interesting property, as invariance to local translation forces the model to care whether some feature is present rather than have a pixel perfect location of where it is.

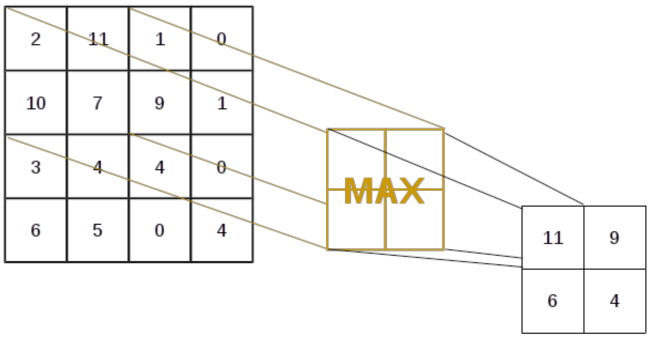


Figure 2.3: 2 × 2 maximum pooling window with 2 × 2 strides and zero padding. The resulting output is downsampled by 4.

A convolutional network with convolution and/or pooling also has the effect of downsampling the input. Downsampling is an important mechanism that eases computation in practice. First, it makes it possible to treat a variable sized input, which a fully connected layer is unable to cope with. Second, it reduces computational complexity because the next layer receives a smaller input. Not only does it hold these practical benefits, it also displays a desirable modelling property. We can view downsampling as compressing information (not in the strict information theory definition) and forcing the model to discard non-relevant bits.

Finally, convolution and pooling operations can be trivially generalized to more dimensions, 3D convolutional neural networks (Conv3D) will be particularly of interests in chapter 5 and chapter 6.

2.2.3 Inductive bias

The power behind CNN can be explained with the concept of prior probability distribution over the parameters of a network. Applying such a prior over the weights of a network constrains the values of the weights to what we think they should be before even training on any data. Any kind of neural network can be viewed as a fully connected one and any straying from this basic architecture can be interpreted as using some kind of prior over its parameters. In that sense, a convolutional architecture introduces a prior over the weights of a layer. It enforces all the connections of a neuron to be zero except for a small region around it.

Overall, this induces a bias in the architecture to distribute the information processing in a tight, locally spatially, organized structure. It works amazingly well when working with natural images where a reasonable assumption behind them is that every pixels are only correlated to the ones in their spatial neighbourhood. A crucial insight that has been shown is that using an architecture of successive convolution and pooling layers results in a network that learns more and more abstract features when going up in the hierarchy. Layers very close to the pixels will become edges detectors where the ones near the last classification layers will encode more abstract concepts such as *face* or *tree*.

2.3 Recurrent neural network

Recurrent neural networks (RNN) are another form of architecture, fine tuned for sequence modelling. RNN extend Equation 2.2 to:

$$\mathbf{h}_t = f_i(\mathbf{W}^\top \mathbf{x}_t + \mathbf{U}^\top \mathbf{h}_{t-1} + \mathbf{b}) \tag{2.7}$$

where the input is now part of a sequence $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ of T tensors. The parameter set $\boldsymbol{\theta}$ now contains the weight matrix $\mathbf{U}$ which learns the relationship with previously computed state $\mathbf{h}_{t-1}$. In most cases there is also an additional learned parameter $\mathbf{h}_0$ to complete the recurrence. See Figure 2.4 for an illustration.

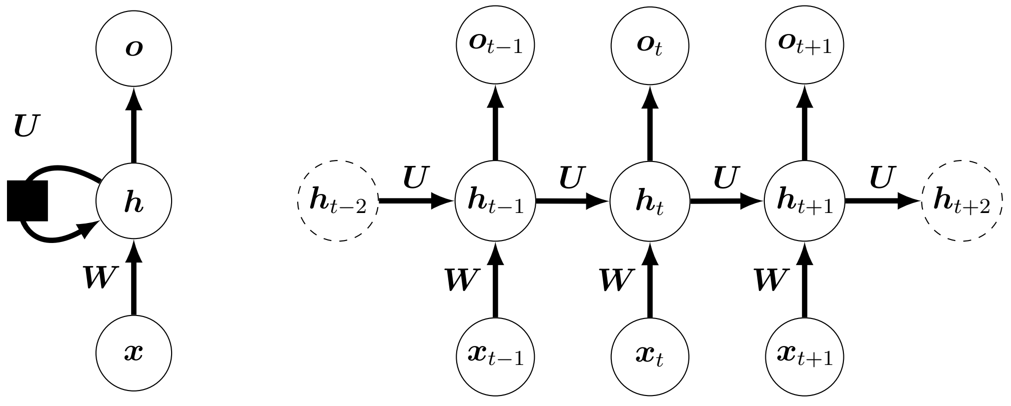


Figure 2.4: Compact graph of a recurrent neural network (*left*) and unfolded (*right*). The compact version uses a loop with a square to indicate a recurrence connection. Note that arrows here are **not** individual parameters as was the case in previous graphs. They actually display the full matrix multiplication (and optional bias vector addition), ex.: the arrow from $\mathbf{x}_t$ to $\mathbf{h}_t$ represents $\mathbf{h}_t = f(\mathbf{W}^\top \mathbf{x}_t)$ where f is the activation function. Such a graph is usually called a computation graph to distinguish them from ones displaying the insides of a specific neural network.

Observe that parameter sharing is happening across the sequence, $\mathbf{W}$ and $\mathbf{U}$ are reused every time. This assures the model doesn't concentrate its efforts on learning each timestep separately and instead finds the correct functions through time.

Recurrent neural networks add another property to their arsenal as they are proven to be Turing complete, i. e. they can represent any function computed by a Turing machine. Their parameters are trained using *backpropagation through time*, a three word concept which again only means using the chain rule of calculus. Every step is dependant of the previous one and the derivatives from the loss to the very first input can all be deterministically and unambiguously computed. Note that it would be a whole different story if a circular dependence was involved in Figure 2.4 (*right*).

RNN exhibit a very large freedom into their architectural design's choice. Three major patterns that will be used in chapter 5 and beyond are to be kept in mind:

1. Producing an output at each time step, as in Figure 2.4
2. Producing an output at each time step and having it feeding back as the recurrence instead of the hidden to hidden connection.
3. Producing a single output after processing the entire sequence (optionally doing 2) as well).

Pattern 2 is useful when RNNs are asked to process their own output as a sequence. Next frame generation investigated in chapter 6 will do exactly that. Let's consider a simplified version:

The network generates a video starting from one frame of the data, generating the next one and then using it as input for the next generation until a chosen amount of steps is done. Be wary that this example in practice will be deep and will actually be using a mixture of 1 and 2.

Teacher forcing may be a valuable procedure for training when recurrent connections from the output feed back as input. This technique is to simply use the true input instead of the output of the network as the feedback connection. In our example, the RNN would generate the next frame again by starting from one in the data. But instead of giving this generated frame as input for the next step, it takes the next true frame from the data. You can then train every output with a loss on every true target, as well reusing the true input which in effect is bootstrapping the network so it doesn't stray into hopeless parameter landscape early in training. The major disadvantage of this method is if the RNN will later be used in an open-loop fashion. This mode is when it stops using the ground truth but reuses its output as input. With time, this can lead to very different input the network was used to see. Its output will stray more and more from the data and can reinforce the accumulation of errors. This problem can in practice be reduced by training the network in both teaching forcing and open-loop modes.

Classification is an example when one needs the design change proposed by 3). The recurrent neural network then computes a summary of the whole sequence which is the final single output. This is usually taken by a classification layer to produce the class prediction.

There are a myriad of other architectural specifications corresponding to various different needs, whether RNN are used for input/output sequences of same length, variable lengths, fixed but unequal length, unknown length, etc. They all have solutions making RNN a very solid and general tool for sequence modelling.

2.3.1 Long short-term memory network

There is, however, one great challenge facing recurrent neural networks, namely learning long-term dependencies. Here we will essentially rehash the analysis done in the textbook, it is a short one that provides the reader with a simplified but accurate view of the problem.

Stripping away all terms not involving recurrence of Equation 2.7 yields:

$$\mathbf{h}_t = \mathbf{U}^\top \mathbf{h}_{t-1} \tag{2.8}$$

$$\mathbf{h}_t = (\mathbf{U}^t)^\top \mathbf{h}_0 \tag{2.9}$$

Assuming $\mathbf{U}$ admits an eigendecomposition of the form

$$\mathbf{U} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top \tag{2.10}$$

and further requiring $\mathbf{Q}$ to be orthogonal simplifies the recurrence to

$$\mathbf{h}_t = \mathbf{Q}^\top \boldsymbol{\Lambda}^t \mathbf{Q}\mathbf{h}_0 \tag{2.11}$$

Notice the eigenvalues $\boldsymbol{\Lambda}$ being raised to the power of t. This will induce an explosion from eigenvalues of magnitude greater than one or, on the opposite, a decay from ones of magnitude less than one. The vanishing and exploding gradient problem is a constant issue clouding over RNN and has been a case for major studies ever since its discovery in the early 90s. One key element in the recent rise of deep learning and its success can be traced back to breakthroughs on how to make RNN properly work. There have been many attempts at solutions and as of today the most effective practical solution has been to use gated RNN based on gated recurrent unit. The long short-term memory (LSTM) network is one of the variants most wildly employed and is the one that shall be discussed here.

First, we need to cover the concept of leaky and gated units. Leaky units are based around the idea of having paths on which the product of derivatives is close to one. This can be achieved by having units with linear self-connections and weightings close to one on these connections. An example is accumulating a running average a of value v such as $a^t \leftarrow \alpha a^{t-1} + (1-\alpha)v^t$, here the α parameter act as a linear self-connection from a to its previous values. Depending on α, the running average can keep or discard information through time. Leaky units are units with self-connection displaying such behaviour and gated units are a generalization of this idea.

A gated unit is a self-connection that is learned at each timestep, it grants the network the ability to decide whether it wants to accumulate or forget information depending on where it is reading the input sequence. With this in mind, the dynamical system captured by the LSTM is illustrated on Figure 2.5 and described by the following set of equations:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f^\top \mathbf{x}_t + \mathbf{U}_f^\top \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{2.12}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i^\top \mathbf{x}_t + \mathbf{U}_i^\top \mathbf{h}_{t-1} + \mathbf{b}_i) \tag{2.13}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^\top \mathbf{x}_t + \mathbf{U}_o^\top \mathbf{h}_{t-1} + \mathbf{b}_o) \tag{2.14}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c^\top \mathbf{x}_t + \mathbf{U}_c^\top \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{2.15}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.16}$$

$\mathbf{f}$ is the forget gate, $\mathbf{i}$ the input gate, $\mathbf{o}$ output gate and the couple $\mathbf{c}$ and $\mathbf{h}$ are the cell and hidden states respectively (all these gates are vectors). σ is the sigmoid activation function and $\odot$ denotes the elementwise product. The forget gate controls what to remember in the self-loop of cell state $\mathbf{c}$, the input gate takes care of gating the input to the self-loop while the output gate does the same at the output of the cell state self-loop. They are all conditioned on the current input $\mathbf{x}_t$ and past hidden state $\mathbf{h}_{t-1}$. Together, these gates control the flow of information modulating the hidden state $\mathbf{h}_t$.

2.3.2 Convolutional LSTM

Next, we shall make a slight modification to LSTM creating the convolutional version of the long short-term memory network [Shi et al., 2015] (ConvLSTM). As was the case in going from the fully connected to the CNN, the changes in the equations are only in going from the matrix multiplication to the convolution operator.

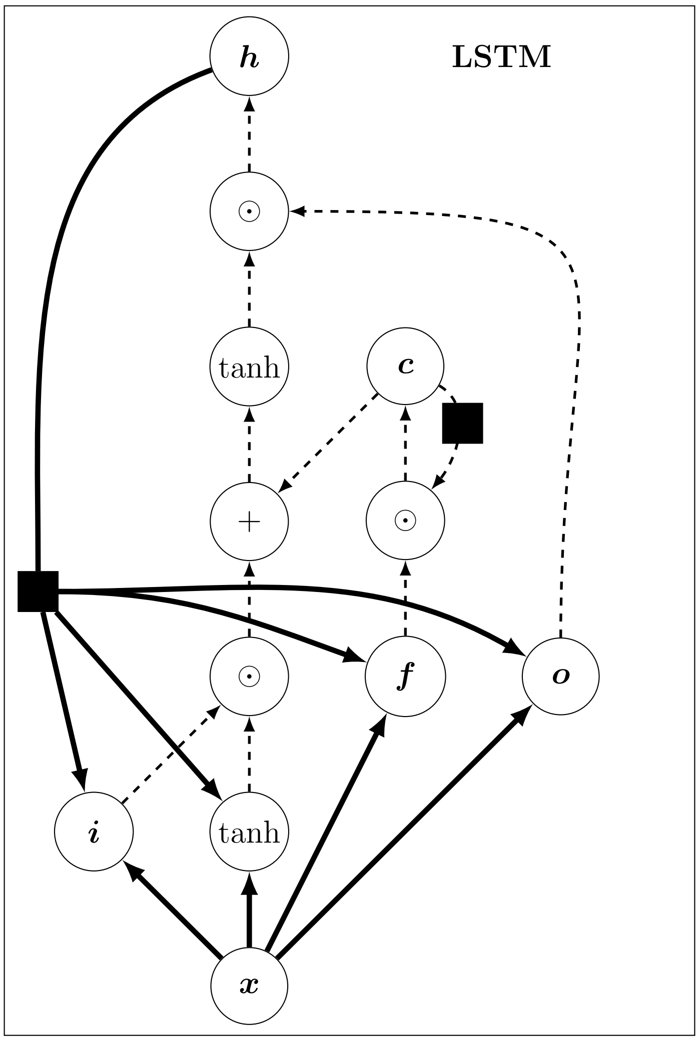


Figure 2.5: Computation graph of the long short-term memory cell. Black boxes indicate recurrences, bold arrows full parameters applications (see Figure 2.4) and dashed arrows computation flow. The vector **h** is the final output.

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f^\top * \mathbf{x}_t + \mathbf{U}_f^\top * \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i^\top * \mathbf{x}_t + \mathbf{U}_i^\top * \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o^\top * \mathbf{x}_t + \mathbf{U}_o^\top * \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c^\top * \mathbf{x}_t + \mathbf{U}_c^\top * \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

Where now everything besides the parameters are tensors (2 dimensional if used with images). Most of what can be said about CNN and RNN can be ported to a ConvLSTM making it an interesting choice when dealing with sequences of images.

2.4 Autoencoders

The final deep learning building block to be presented is a little bit more exotic and quite important for generative models, it is the autoencoder.

Autoencoders are usually decomposed into two sub-models, an encoder which maps the input to a code $\mathbf{z} = e(\mathbf{x})$ and a decoder that tries to reconstruct the input from the code $\hat{\mathbf{x}} = d(\mathbf{z})$. An important aspect in this kind of setting is to **not** have *perfect* reconstructions in the sense that $\mathbf{x} = \hat{\mathbf{x}}$ for all $\mathbf{x}$ in the dataset. Basically, if the autoencoder learns to simply copy the data perfectly it will not learn anything insightful about it. This is why innovation with regards to this class of models has concentrated on this aspect and is being driven by finding ways to constrain their ability in copying the input.

Typically they are trained with a loss of the form $\mathcal{L}(\mathbf{x}, d(e(\mathbf{x})))$ like a mean squared error between the input and the reconstruction for example. However, given enough capacity this doesn't prevent the autoencoder to learn an identity mapping. Therefore, strategies to prevent pure copying will focus on different losses:

$$\mathcal{L}(\mathbf{x}, d(e(\mathbf{x}))) + \Omega(\mathbf{z}) \tag{2.17}$$

$$\mathcal{L}(\mathbf{x}, d(e(\mathbf{x}))) + \Omega(\mathbf{z}, \mathbf{x}) \tag{2.18}$$

$$\mathcal{L}(\mathbf{x}, d(e(\tilde{\mathbf{x}}))) \tag{2.19}$$

The first one is adding a regularizer to the latent code $\mathbf{z}$, forcing it to be sparse for example. The second line is extending the penalty to the inputs such as penalizing the derivatives of $\mathbf{h}$ with respect to $\mathbf{x}$. Contractive autoencoder implement this regularizing term. The last line belongs to the class of denoising autoencoder. $\tilde{\mathbf{x}}$ means a corrupted version of the input, the task then for the autoencoder is to try to remove the corruption in the input.

Autoencoders are not restricted to deterministic mappings. They can be generalized to stochastic functions $p_e(\mathbf{z}|\mathbf{x})$ and $p_d(\mathbf{x}|\mathbf{z})$.

Doing so, we can now think in term of log-likelihood. The same way the classical classification setup is to learn the probability distribution $p(\mathbf{y}|\mathbf{x})$ where $\mathbf{y}$ is the vector of targets, we can train the decoder to optimize on $\log p_d(\mathbf{x}|\mathbf{z})$ where $\mathbf{x}$ from the dataset are taken as the targets. The encoder can be optimized on $\log p_e(\mathbf{z}|\mathbf{x})$ with a prior on the probability distribution governing the latent code $p(\mathbf{z})$.

2.4.1 Upsampling

Autoencoders have been made convolutional as well granting them the same bonuses that bring convolutional networks. This raises a fundamental issue in computer vision, *upsampling*. Upsampling is characterized by going from lower resolutions to higher and many techniques exist in order to fill the new pixels being created in the process. Since this projects the input in a higher space, it is a non trivial problem to devise solutions that will be efficient and preserve the visual appeal and consistency of the image.

Autoencoders make heavy use of this procedure when upsampling from the latent space and going back into the data space. We would like to be able to learn in a deep learning way on how to proceed. The case of a fully connected layer is very simple. Recall Equation 2.2, there can be any relative difference in the number of dimensions between $\mathbf{x}$ and $\mathbf{h}$. Every neuron in the input has a connection to every neuron in the output. This simplicity does not carry over with CNN because we would like to keep their property of locally structuring computation. For this purpose, transposed convolutions, or deconvolution [Zeiler et al., 2010], have been proposed as a general way to do so. The idea is to treat the process as a convolution, but *expand* the input with zeroes. See Figure 2.6 for a 1D visualization.

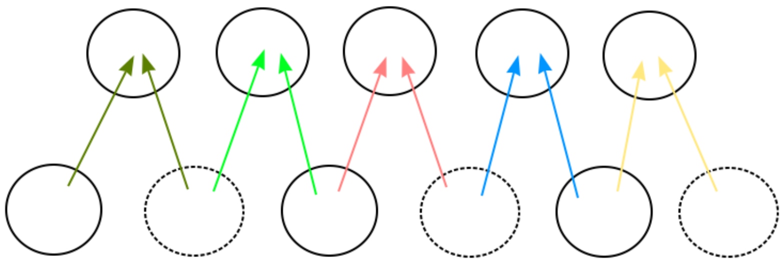


Figure 2.6: 1D deconvolution for input vector of 3 dimensions, output vector of 5, and kernel of size 2. The dashed circles are zeroes inserted between the values of the input. Like a regular convolution, each couple of arrows is to represent the same parameters being applied everytime. The net result is a higher dimensional output.

Some modern architectures will favor using *bilinear upsampling* instead of deconvolution. They insert convolution layers tuned so they do not alter the dimensions between these upsampling layers in order to perform learning. Wojna et al. [2017] is a recent overview of the different possibilities for the decoder and they experimentally show that this choice can have an important consequence depending on the task.

Chapter 3

Generative Models

There are two broad classes of models gathering presently most of the deep learning fuss. We already have implicitly discussed the discriminative class in chapter 2 through classification examples. These models are ones that deal with the challenge of learning relevant features of the data in order to identify, classify and/or discriminate on them. In recent years another class, generative models, have distinguished themselves as a strong choice for learning structures in the data through the process of generation. The reasoning is quite simple: If a model can generate indistinguishable data from the real one it is studying then it must have understood something meaningful about its distribution. Generative models operate in the unsupervised learning regime most of the time which is a clear advantage over discriminative models whom are usually relying on supervised learning and additional information.

Deep learning and generative models taps greatly into the field of probabilistic graphical model. Without stating it, we have already bumped into graph theory in the last chapter and we will not go more in depth in that very large topic. Koller and Friedman [2009] is a solid textbook on the matter and we refer it for this chapter and for readers interested in working out the inner gears of PGM.

In this chapter we will cover three types of generative models that have risen among the ranks: autoregressive networks, variational autoencoders and generative adversarial networks. The last two will give birth to chapter 4 and the very last one is the backbone of chapter 6.

3.1 Autoregressive networks

The goal we seek is for a sampling mechanism correctly modelling the data distribution $p(\mathbf{x})$. We have already brushed up the concept of having a latent space $\mathbf{z}$ which makes this wish possible in section 2.4 by ways of the conditional distribution

$p(\mathbf{x}|\mathbf{z})$. When doing this, an immediate issue arises in computing the marginal $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$ which is more often than not intractable without restrictive assumptions on the latent's code prior $p(\mathbf{z})$ and the conditional. These problems will be investigated in the next two sections.

Autoregressive networks exploit instead the fundamental factorial property of any D-dimensional probability distribution:

$$p(\mathbf{x}) = \prod_{d=1}^{D} p(x_{o_d}|\mathbf{x}_{o_{<d}}) \tag{3.1}$$

where the index o is to underline that the exact order of the $\mathbf{x}$ factorization doesn't matter. $\mathbf{x}_{o_{<d}}$ describes the subvector along the $d-1$ dimensions of ordering o. The clever observation to be distilled from Equation 3.1 is that it can be treated in a sequential fashion starting from $p(x_{o_1}|\mathbf{x}_{o_{<1}})$ working your way up to $p(x_{o_d}|\mathbf{x}_{o_{<d-1}})$ hence yielding the complete $p(\mathbf{x})$. Neural autoregressive distribution estimation [Uria et al., 2016] (NADE) were among the first ones to use neural networks on Equation 3.1 with success at generation (compared to before neural methods that were employed like restricted boltzman machines). NADE and autoregressive networks's training is by the standard minibatch stochastic gradient descent on the negative log-likelihood,

$$\frac{1}{N}\sum_{n=1}^{N} -\log p(\mathbf{x}^n) = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{d} -\log p(x^n_{o_d}|\mathbf{x}^n_{o_{<d}}) \tag{3.2}$$

for N data points in the minibatch.

Recall that we presented a tailor made neural network when playing with sequences. Naturally then, pixel recurrent neural network [Oord et al., 2016] (PixelRNN) proposed to model the sequence of conditionals using LSTM. This way, PixelRNN achieved the most breathtaking image generation of its time. Unfortunately, this was accomplished by paying high computational price, indeed sampling from autoregressive networks is slow by definition. Casting the problem in this manner requires sequential sampling making PixelRNN's sampling *pixel by pixel* in order to generate an image. Nevertheless, such impressive results secured a spot for autoregressive networks as competitive models for ones not minding the hardware burden.

Efforts have been made to bridge this framework with latent space models and try to profit from both potentials such as the PixelVAE [Gulrajani et al., 2017b]. We shall now need some theoretical background in order to make some sense of this latent business.

3.2 Variational autoencoders

Let's come back to the intractable data marginal likelihood when switching to a latent code model $p_{\text{model}}(\mathbf{x}, \mathbf{z})$ such as autoencoders:

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

We first need to introduce some notation: We shall write true distributions, like the data $p(\mathbf{x})$, the latent code prior $p(\mathbf{z})$, the true data posterior $p(\mathbf{x}|\mathbf{z})$ and the true code posterior $p(\mathbf{z}|\mathbf{x})$ without subscript. The distributions we will approximate them with and try to learn will be written with their parameters subscript, $p_\theta(\mathbf{x}|\mathbf{z})$ and $q_\phi(\mathbf{z}|\mathbf{x})$.

This integral does not generally admits a close form. It could be possible to get one using $p(\mathbf{z}|\mathbf{x}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})/p(\mathbf{x})$ if we had very particular forms of $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$, but $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$ are usually taken for granted to be intractable and we are interested in solutions free from further assumptions. We could also draw samples from $p(\mathbf{z}|\mathbf{x})$ by skipping the normalization factor $p(\mathbf{x})$ and then approximate the integral by sample averages by ways of MCMC. This suffers from scalability problems with large dataset where deep learning has the potential to shine. This is the motivation behind going for a variational approach and using approximate distributions p_θ and q_ϕ.

3.2.1 Kullback-Leibler divergence

What makes distributions good approximation of one another? Answering this question begs the need for a notion of similarity. Thus our story begins with the *Kullback-Leibler* divergence.

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx$$

The KL is a measure of the relative entropy of probability distribution P with respect to Q. Another interpretation from coding theory tells us that this will extract the expected number of extra bits needed to code the members of P using the code incoming from Q.

We can now try to approximate the true $p(\mathbf{z}|\mathbf{x})$ with q_ϕ using the KL:

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{x},\mathbf{z})} d\mathbf{z} \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x},\mathbf{z})] + \log p(\mathbf{x})
\end{aligned}
\tag{3.3}
$$

Isolating the log-likelihood of the data,

$$
\log p(\mathbf{x}) = D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log q_\phi(\mathbf{z}|\mathbf{x}) + \log p(\mathbf{x},\mathbf{z})]
$$

But the KL cannot be negative $D_{KL} \geq 0$,

$$
\log p(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] = L[q_\phi] \tag{3.4}
$$

Equation 3.4 is known as the evidence lower bound. This inequality allows us to bypass the unknown normalization $p(\mathbf{x})$. Manipulating it a little bit more by factorizing the joint with our data posterior approximation $p(\mathbf{x},\mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, we get

$$
\begin{aligned}
L[q_\phi] &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})}] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))
\end{aligned}
\tag{3.5}
$$

This last equation is the final piece of the puzzle assembling our second generative model, the variational autoencoder [Kingma and Welling, 2013] (VAE).

3.2.2 Learning a variational bayes latent space

In the previous subsection we used a variational bayes method. Mathematically, $L[q_\phi]$ is a functional (a function mapping a function to a real scalar) and we did what is called variational inference in order to find the approximating posterior (bayes).

Let's throw neural networks in the mix. We will take our classical stochastic autoencoder with an encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and a decoder $p_\theta(\mathbf{x}|\mathbf{z})$ parametrizing the two distributions. Remember that we want to be able to sample from our model fairly easily so it can be used as generation, for this we shall now make the required assumptions and approximate everything as Gaussian. It might appear odd because it was argued since the beginning that we wanted a constraints free solution. Sadly Equation 3.5 did not make any jump to tractability, it was solved this way in essence for easier optimization as we shall soon appreciate. The details of our approximations are as follow:

- prior: $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$
- encoder: $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}^2_\phi(\mathbf{x})\mathbf{I})$
- decoder: $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_\theta(\mathbf{z}), \mathbf{I})$

Where $\mathcal{N}$ is the Gaussian distribution with average $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$. Neural networks parametrized by $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ will fully describe both Gaussian. There is one technicality in our way, how can backpropagation work and train these networks, both imply stochastic processes? There is a trick known as the *reparametrization trick* [Kingma, 2013, Bengio et al., 2014, 2013] that keeps everything differentiable through stochastic units in neural networks. They propose to reparametrize $\mathbf{z}$ such that:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{3.6}$$

This way, the encoder can deterministically parametrized its Gaussian with $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\sigma}_\phi$ which can be learned through gradient descent and stochasticity is relegated to the injected noise $\boldsymbol{\epsilon}$. The prior's assumption served two purposes. The obvious one is that sampling from this simple Gaussian is trivial, the second one is that the KL can be solved analytically. From appendix B of the VAE paper we find that for two Gaussian:

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

where the sum along J is for the dimensions of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

As for the decoder we do not need to apply Equation 3.6 because it is implicitly embedded in the loss. Indeed, the decoder's objective is $\mathbb{E}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ and it can be shown that training this with mean squared error is equivalent to optimize on a log-likelihood of a Gaussian with unit standard deviation.

Grouping these terms will give us the final loss for training a VAE:

$$\mathcal{L}(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2) + ||\hat{\mathbf{x}} - \mathbf{x}||_2 \tag{3.7}$$

where $\hat{\mathbf{x}}$ is a sample from $p_\theta(\mathbf{x}|\mathbf{z})$.

Variational autoencoders' wonders are many, they are somewhat easy to train and provide us with efficient inference and sampling mechanisms. By inference we mean the path $\mathbf{x} \rightarrow \mathbf{z}$ which can be taught, if learned right, of a powerful way to abstract the data and manipulate it. The drawback is that their samples tend to be blurry and of less quality compared with both models from past and next subsections. It has been suggested that this is explained by the issue of maximum likelihood training paradigm combined with conditional independence assumption on the output given the latent variables [Theis et al., 2016]. The last model of this chapter will take a departure from this paradigm.

3.3 Generative adversarial networks

Neural networks have been shown to be very unreliable against adversarial examples [Szegedy et al., 2014]. Adversarial examples stem from the concept of robustness to noise which neural networks are not very good at. A striking example can be seen on Figure 3.1.

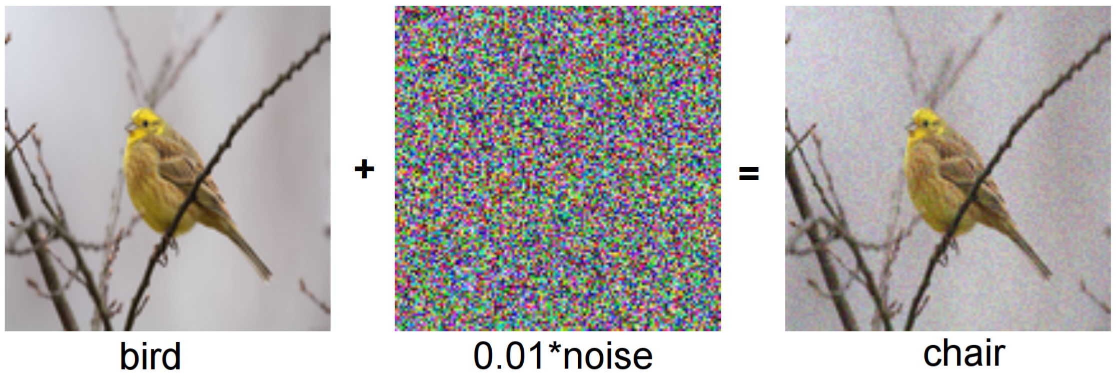


Figure 3.1: Hypothetical adversarial example. Uniform noise is added to the image completely throwing off the class label. This example would actually still work even if the corruption to the image was smaller and imperceptible to human eyes.

The good news is that this can be steered to our gain and generative adversarial networks [Goodfellow et al., 2014] (GAN) have been proposed as an impressive type of generative models based on the adversarial idea. The GAN training is conceptually simple: a network called the discriminator is asked to classify whether it receives a true data point $\mathbf{x}$ or a fake one $\tilde{\mathbf{x}}$. This would be trivially easy if not for the fact that $\tilde{\mathbf{x}}$ is produced by another neural network, the generator, with mission to fool the discriminator. Both network play a game, the generator constantly on the toes of the discriminator with net result on strengthening their respective tasks.

3.3.1 Adversarial game

Formally, for a data distribution $p(\mathbf{x})$, prior probability distribution $p(\mathbf{z})$, discriminator's function $D(\mathbf{x})$ and generator's function $G(\mathbf{z})$, GAN implements a two-player minmax game with value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{3.8}$$

It is easy to see that at the early iterations of the game it won't be hard for the discriminator to tell its inputs apart since the generator will most likely produce garbage. $\log(1 - D(G(\mathbf{z})))$ will saturate and the gradient for G will be too weak. For this reason, Equation 3.8 is modified to optimize

$$\max_G \max_D V'(G, D) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(D(G(\mathbf{z})))] \tag{3.9}$$

instead. Goodfellow et al. [2014] have shown that at optimality (and the usual "enough" parameters assumption) the GAN algorithm will converge to $p_{\text{data}}(\mathbf{x}) = p_{\text{g}}(\mathbf{x})$ where p_{g} is the implicit probability distribution over the data obtained with G. Therefore, the generator will at optimality of the game model perfectly the data distribution.

By careful inspection (see paper for mathematical proof) it is not too hard to realize that the optimal discriminator happening when $p_{\text{data}}(\mathbf{x}) = p_{\text{g}}(\mathbf{x})$ can only output 0.5, $D^*_G(\mathbf{x}) = \frac{1}{2}$ where D^*_G denotes optimality for a fixed G. We can convince ourselves that this is the case with this light argument: The two distributions it receives in input are equal, and nothing can be learned to distinguish them so the discriminator will only at best flip a coin when asked to classify them apart. At this point, it can be shown then that the cost value for G is

$$C(G) = -\log(4) + 2 \cdot JSD(p_{\text{data}}||p_{\text{g}}) \tag{3.10}$$

where JSD is the Jensen-Shannon divergence, the symmetrical version of the KL: $JSD(Q||P) = \frac{1}{2}KL(Q||A) + \frac{1}{2}KL(P||A)$ for $A = \frac{1}{2}(Q + P)$.

3.3.2 Advantages and disadvantages

On one hand, generative adversarial networks do not require the hardware powerhouse of autoregressive networks to generate impressive samples outmatching the visual quality of variational autoencoders when both are applied to images in a convolutional architectural fashion [Radford et al., 2016, Larsen et al., 2016]. As was experimentally conjectured in Theis et al. [2016], it could be an attribute of the shift from the maximum likelihood paradigm in training the generator with Equation 3.10 instead.

This can be glimpsed by inspection of the KL properties since maximizing the likelihood between our unknown data distribution and generated one is equivalent to minimizing $KL(p_{\text{data}}||p_{\text{g}})$. If $p_{\text{data}}(x) > p_{\text{g}}(x)$ then x has higher probability coming from the data and the generator "drops" probability mass of these points. In this case, if $p_{\text{g}}(x) \to 0$ then $KL \to \infty$ and the generator pays an extremely high cost for dropping part of the data. At the opposite, if $p_{\text{data}}(x) < p_{\text{g}}(x)$ then x has high probability to come from the generator's distribution and this happens when the generator outputs samples that do not look real. Here when $p_{\text{data}}(x) \to 0$ then $KL \to 0$ and the generator is not penalized for generating unreal samples. Therefore, one can imagine that since the GAN's generator is optimized on the JSD of Equation 3.10 then it is acting on a sort of optimal middle ground of the two cost regimes of the KL described above.

They are also as straightforward to sample from as VAE since the sampling procedure is similarly done with a deterministic function on a prior $\tilde{\mathbf{x}} = G(p(\mathbf{z}))$ normally taken to be Gaussian or other simple distributions.

Another interesting result is that GAN have shed light on a surprisingly efficient path for semi-supervised learning when Salimans et al. [2016] grabbed this task' state-of-the-art (SOTA) at the time on many popular deep learning datasets. No hard explanation currently exists, but one could conjecture[2] that this is an undercover form of transfer learning. The discriminator starts by learning on an easier problem, whether two points are from the same distribution, and then reuse this knowledge on the harder problem of discriminating among classes.

On the other hand, the game behind generative adversarial networks is notoriously unstable and demands constant babysitting from the user. That is because the core reason behind failure modes in training, relative capacity between the discriminator and the generator (and to some extent the discriminator's ability to model the data as well), is very hard to estimate other than by trial and error. If one neural network overpower the other, training will usually end in failure. Arjovsky and Bottou [2017] is a recent effort towards explaining this instability. They have proven what was originally experienced by Goodfellow et al. [2014], the

[2]Conjecture attributed to Ishmael Belghazi following discussions on semi-supervised learning and yet unpublished work.

original cost function Equation 3.8 will give vanishing gradients to the generator. This can be seen in corollary 2.1 of the paper:

$$\lim_{||D-D^*||} \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G_{\boldsymbol{\theta}}(\mathbf{z})))] = 0$$

They have also derived in theorem 2.5 what does the expectation of the default practical version of the value function V' of Equation 3.9 gradient is:

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[-\nabla_{\boldsymbol{\theta}} \log D^*(G_{\boldsymbol{\theta}}(\mathbf{z}))|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}] = \nabla_{\boldsymbol{\theta}}[KL(p_{\text{g}}||p_{\text{data}}) - 2 \cdot JSD(p_{\text{g}}||p_{\text{data}})]|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$$

They argue that first because JSD is in the opposite sign, it will push for the distributions to be different. Second, the KL here is the reversed (in term of its arguments' positions) of the maximum likelihood. It will do in this form the opposite of what was written above, it will pay an extremely high cost for generating out of distribution samples and a low one for dropping part of the data distribution, which is what is observed in practice.

Furthermore, the value function's scalar of Equation 3.8 has no reasonable interpretation. Unlike in a classification scenario where the user needs only to take care of the training error vs the validation one, monitoring its quantity is of almost no importance with regards to outputting data matching samples and the solution is to periodically assess their quality by eye.

Model-wise, GAN lack an inference mechanism and cannot manipulate the data on an easier abstract space which is one of the most interesting property of latent space model like VAE and many efforts have aimed to bridge the gap recently [Larsen et al., 2016, Lamb et al., 2016, Dosovitskiy and Brox, 2016]. This particular point will be the motivation behind next chapter.

In any case, generative adversarial networks have become immensely popular due to the quality of their results they can generate if trained correctly. Among the explosion of papers [Mao et al., 2017, Arjovsky et al., 2017, Gulrajani et al., 2017a], to name a few, have offered insight on fixes to the practical issues mentioned above. Finding a metric that when optimized correlates with sample quality is particularly a golden graal.

Chapter 4

Adversarially learned inference

4.0 Prologue to the paper

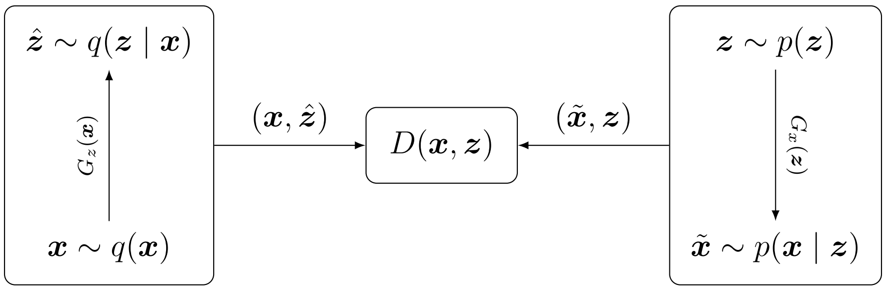
This chapter presents joint work with Vincent Dumoulin[3], Ishmael Belghazi[3], Ben Poole[4], Alex Lamb[3], Martin Arjovsky[5] and Aaron Courville[36] published at the 5th International Conference on Learning Representations (ICLR 2017) [Dumoulin et al., 2017].

It has been integrated to this thesis subject to few modifications with permission from the authors. The modifications are mostly to accommodate the paper into this document. Doing so, the abstract and appendix have been taken off and the introduction has been trimmed down since it contained materials already presented in chapter 3. Everything else is as is.

Note that keeping with the spirit of minimal changes has the effect of a slight notation desynchronization with the rest of the thesis. The other chapters do actually take notation conventions close to this paper so it is not dramatic enough but it is good to keep this in mind. The formatting and general visual appeal also changed due to the whims of LaTeX's compilation in another environment.

My contribution in this publication are concentrated in section 4.4 where I have done part of the CelebA experiments and mostly subsection 4.4.3.

[3]MILA, Université de Montréal
[4]Neural Dynamics and Computation Lab, Stanford
[5]New York University
[6]CIFAR fellow



$\hat{\boldsymbol{z}} \sim q(\boldsymbol{z} \mid \boldsymbol{x})$
$G_z(\boldsymbol{x})$
$\boldsymbol{x} \sim q(\boldsymbol{x})$
$(\boldsymbol{x}, \hat{\boldsymbol{z}})$
$D(\boldsymbol{x}, \boldsymbol{z})$
$(\tilde{\boldsymbol{x}}, \boldsymbol{z})$
$\boldsymbol{z} \sim p(\boldsymbol{z})$
$G_x(\boldsymbol{z})$
$\tilde{\boldsymbol{x}} \sim p(\boldsymbol{x} \mid \boldsymbol{z})$

Figure 4.1: The adversarially learned inference (ALI) game.

4.1 Introduction

In this paper, we propose a novel approach to integrate efficient inference within the GAN framework. Our approach, called Adversarially Learned Inference (ALI), casts the learning of both an inference machine (or encoder) and a deep directed generative model (or decoder) in an GAN-like adversarial framework. A discriminator is trained to discriminate joint samples of the data and the corresponding latent variable from the encoder (or approximate posterior) from joint samples from the decoder while in opposition, the encoder and the decoder are trained together to fool the discriminator. Not only are we asking the discriminator to distinguish synthetic samples from real data, but we are requiring it to distinguish between two joint distributions over the data space and the latent variables.

With experiments on the Street View House Numbers (SVHN) dataset [Netzer et al., 2011], the CIFAR-10 object recognition dataset [Krizhevsky and Hinton, 2009], the CelebA face dataset [Liu et al., 2015] and a downsampled version of the ImageNet dataset [Russakovsky et al., 2015], we show qualitatively that we maintain the high sample fidelity associated with the GAN framework, while gaining the ability to perform efficient inference. We show that the learned representation is useful for auxiliary tasks by achieving results competitive with the state-of-the-art on the semi-supervised SVHN and CIFAR10 tasks.

4.2 Adversarially learned inference

Consider the two following probability distributions over $\boldsymbol{x}$ and $\boldsymbol{z}$:

- the *encoder* joint distribution $q(\boldsymbol{x}, \boldsymbol{z}) = q(\boldsymbol{x})q(\boldsymbol{z} \mid \boldsymbol{x})$,
- the *decoder* joint distribution $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z})$.

These two distributions have marginals that are known to us: the encoder marginal $q(\boldsymbol{x})$ is the empirical data distribution and the decoder marginal $p(\boldsymbol{z})$ is usually defined to be a simple, factorized distribution, such as the standard Normal distribution $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. As such, the generative process between $q(\boldsymbol{x}, \boldsymbol{z})$ and $p(\boldsymbol{x}, \boldsymbol{z})$ is reversed.

ALI's objective is to match the two joint distributions. If this is achieved, then we are ensured that all marginals match and all conditional distributions also match. In particular, we are assured that the conditional $q(\boldsymbol{z} \mid \boldsymbol{x})$ matches the posterior $p(\boldsymbol{z} \mid \boldsymbol{x})$.

In order to match the joint distributions, an adversarial game is played. Joint pairs $(\boldsymbol{x}, \boldsymbol{z})$ are drawn either from $q(\boldsymbol{x}, \boldsymbol{z})$ or $p(\boldsymbol{x}, \boldsymbol{z})$, and a discriminator network learns to discriminate between the two, while the encoder and decoder networks are trained to fool the discriminator.

The value function describing the game is given by:

$$\begin{aligned}\min_G \max_D V(D, G) &= \mathbb{E}_{q(\boldsymbol{x})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(1 - D(G_x(\boldsymbol{z}), \boldsymbol{z}))] \\ &= \iint q(\boldsymbol{x}) q(\boldsymbol{z} \mid \boldsymbol{x}) \log(D(\boldsymbol{x}, \boldsymbol{z})) d\boldsymbol{x} d\boldsymbol{z} \\ &+ \iint p(\boldsymbol{z}) p(\boldsymbol{x} \mid \boldsymbol{z}) \log(1 - D(\boldsymbol{x}, \boldsymbol{z})) d\boldsymbol{x} d\boldsymbol{z}.\end{aligned} \tag{4.1}$$

An attractive property of adversarial approaches is that they do not require that the conditional densities can be computed; they only require that they can be sampled from in a way that allows gradient backpropagation. In the case of ALI, this means that gradients should propagate from the discriminator network to the encoder and decoder networks.

This can be done using the the reparametrization trick [Kingma, 2013, Bengio et al., 2014, 2013]. Instead of sampling directly from the desired distribution, the random variable is computed as a deterministic transformation of some noise such that its distribution is the desired distribution. For instance, if $q(z \mid x) = \mathcal{N}(\mu(x), \sigma^2(x) I)$, one can draw samples by computing

$$z = \mu(x) + \sigma(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{4.2}$$

More generally, one can employ a change of variable of the form

$$v = f(u, \epsilon) \tag{4.3}$$

where ϵ is some random source of noise.

The discriminator is trained to distinguish between samples from the encoder $(\boldsymbol{x}, \hat{\boldsymbol{z}}) \sim q(\boldsymbol{x}, \boldsymbol{z})$ and samples from the decoder $(\tilde{\boldsymbol{x}}, \boldsymbol{z}) \sim p(\boldsymbol{x}, \boldsymbol{z})$. The generator is trained to fool the discriminator, i.e., to generate $\boldsymbol{x}, \boldsymbol{z}$ pairs from $q(\boldsymbol{x}, \boldsymbol{z})$ or $p(\boldsymbol{x}, \boldsymbol{z})$

Algorithm 1 The ALI training procedure.

```
$\theta_g, \theta_d \leftarrow$ initialize network parameters
repeat
    $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(M)} \sim q(\boldsymbol{x})$    ▷ Draw $M$ samples from the dataset and the prior
    $\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(M)} \sim p(\boldsymbol{z})$
    $\hat{\boldsymbol{z}}^{(i)} \sim q(\boldsymbol{z} \mid \boldsymbol{x} = \boldsymbol{x}^{(i)}), \quad i = 1, \dots, M$    ▷ Sample from the conditionals
    $\tilde{\boldsymbol{x}}^{(j)} \sim p(\boldsymbol{x} \mid \boldsymbol{z} = \boldsymbol{z}^{(j)}), \quad j = 1, \dots, M$
    $\rho_q^{(i)} \leftarrow D(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{z}}^{(i)}), \quad i = 1, \dots, M$    ▷ Compute discriminator predictions
    $\rho_p^{(j)} \leftarrow D(\tilde{\boldsymbol{x}}^{(j)}, \boldsymbol{z}^{(j)}), \quad j = 1, \dots, M$
    $\mathcal{L}_d \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log(\rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M} log(1 - \rho_p^{(j)})$    ▷ Compute discriminator loss
    $\mathcal{L}_g \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log(1 - \rho_q^{(i)}) - \frac{1}{M}\sum_{j=1}^{M} \log(\rho_p^{(j)})$    ▷ Compute generator loss
    $\theta_d \leftarrow \theta_d - \nabla_{\theta_d}\mathcal{L}_d$    ▷ Gradient update on discriminator network
    $\theta_g \leftarrow \theta_g - \nabla_{\theta_g}\mathcal{L}_g$    ▷ Gradient update on generator networks
until convergence
```

that are indistinguishable one from another. See Figure 4.1 for a diagram of the adversarial game and Algorithm 1 for an algorithmic description of the procedure.

In such a setting, and under the assumption of an optimal discriminator, the generator minimizes the Jensen-Shannon divergence [Lin, 1991] between $q(\boldsymbol{x}, \boldsymbol{z})$ and $p(\boldsymbol{x}, \boldsymbol{z})$. This can be shown using the same proof sketch as in the original GAN paper [Goodfellow et al., 2014].

4.2.1 Relation to GAN

ALI bears close resemblance to GAN, but it differs from it in the two following ways:

- The generator has two components: the encoder, $G_z(\boldsymbol{x})$, which maps data samples $\boldsymbol{x}$ to $\boldsymbol{z}$-space, and the decoder $G_x(\boldsymbol{z})$, which maps samples from the prior $p(\boldsymbol{z})$ (a source of noise) to the input space.
- The discriminator is trained to distinguish between joint pairs $(\boldsymbol{x}, \hat{\boldsymbol{z}} = G_x(\boldsymbol{x}))$ and $(\tilde{\boldsymbol{x}} = G_x(\boldsymbol{z}), \boldsymbol{z})$, as opposed to marginal samples $\boldsymbol{x} \sim q(\boldsymbol{x})$ and $\tilde{\boldsymbol{x}} \sim p(\boldsymbol{x})$.

4.2.2 Alternative approaches to feedforward inference in GANs

The ALI training procedure is not the only way one could learn a feedforward inference network in a GAN setting.

In recent work, Chen et al. [2016] introduce a model called InfoGAN which minimizes the mutual information between a subset $\boldsymbol{c}$ of the latent code and $\boldsymbol{x}$ through the use of an auxiliary distribution $Q(\boldsymbol{c} \mid \boldsymbol{x})$. However, this does not correspond to full inference on $\boldsymbol{z}$, as only the value for $\boldsymbol{c}$ is inferred. Additionally, InfoGAN requires that $Q(\mathbf{c} \mid \mathbf{x})$ is a tractable approximate posterior that can be sampled from and evaluated. ALI only requires that inference networks can be sampled from, allowing it to represent arbitrarily complex posterior distributions.

One could learn the inverse mapping from GAN samples: this corresponds to learning an encoder to reconstruct $\boldsymbol{z}$, i.e. finding an encoder such that $\mathbb{E}_{z\sim p(z)}[\|z - G_z(G_x(z))\|_2^2] \approx 0$. We are not aware of any work that reports results for this approach. This resembles the InfoGAN learning procedure but with a fixed generative model and a factorial Gaussian posterior with a fixed diagonal variance.

Alternatively, one could decompose training into two phases. In the first phase, a GAN is trained normally. In the second phase, the GAN's decoder is frozen and an encoder is trained following the ALI procedure (i.e., a discriminator taking *both* $\boldsymbol{x}$ and $\boldsymbol{z}$ as input is introduced). We call this *post-hoc learned inference.* In this setting, the encoder and the decoder cannot interact together during training and the encoder must work with whatever the decoder has learned during GAN training. Post-hoc learned inference may be suboptimal if this interaction is beneficial to modelling the data distribution.

4.2.3 Generator value function

As with GANs, when ALI's discriminator gets too far ahead, its generator may have a hard time minimizing the value function in Equation 4.1. If the discriminator's output is sigmoidal, then the gradient of the value function with respect to the discriminator's output vanishes to zero as the output saturates.

As a workaround, the generator is trained to maximize

$$V'(D, G) = \mathbb{E}_{q(\boldsymbol{x})}[\log(1 - D(\boldsymbol{x}, G_{\boldsymbol{z}}(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(D(G_{\boldsymbol{x}}(\boldsymbol{z}), \boldsymbol{z}))] \tag{4.4}$$

which has the same fixed points but whose gradient is stronger when the discriminator's output saturates.

The adversarial game does not require an analytical expression for the joint distributions. This means we can introduce variable changes without having to know the explicit distribution over the new variable. For instance, sampling from $p(z)$ could be done by sampling $\epsilon \sim \mathcal{N}(0, I)$ and passing it through an arbitrary differentiable function $z = f(\epsilon)$.

However, gradient propagation into the encoder and decoder networks relies on the reparametrization trick, which means that ALI is not directly applicable to either applications with discrete data or to models with discrete latent variables.

4.2.4 Discriminator optimality

Proposition 1. *Given a fixed generator G, the optimal discriminator is given by*

$$D^*(x,z) = \frac{q(x,z)}{q(x,z)+p(x,z)}. \tag{4.5}$$

Proof. For a fixed generator G, the complete data value function is

$$V(D,G) = \mathbb{E}_{x,z\sim q(x,z)}[\log(D(x,z))] + \mathbb{E}_{x,z\sim p(x,z)}[\log(1-D(x,z))]. \tag{4.6}$$

The result follows by the concavity of the log and the simplified Euler-Lagrange equation first order conditions on $(x,z)\to D(x,z)$. □

4.2.5 Relationship with the Jensen-Shannon divergence

Proposition 2. *Under an optimal discriminator D^*, the generator minimizes the Jensen-Shanon divergence which attains its minimum if and only if $q(\boldsymbol{x},\boldsymbol{z}) = p(\boldsymbol{x},\boldsymbol{z})$.*

Proof. The proof is a straightforward extension of the proof in Goodfellow et al. [2014]. □

4.2.6 Invertibility

Proposition 3. *Assuming optimal discriminator D and generator G. If the encoder $G_{\boldsymbol{x}}$ is deterministic, then $G_{\boldsymbol{x}} = G_{\boldsymbol{z}}^{-1}$ and $G_{\boldsymbol{z}} = G_{\boldsymbol{x}}^{-1}$ almost everywhere.*

Sketch of proof. Consider the event $R_\epsilon = \{\boldsymbol{x} : \|x - (G_{\boldsymbol{x}} \circ G_{\boldsymbol{z}})(\boldsymbol{x}))\| > \epsilon\}$ for some positive ϵ. This set can be seen as a section of the $(\boldsymbol{x},\boldsymbol{z})$ space over the elements $\boldsymbol{z}$ such that $\boldsymbol{z} = G_{\boldsymbol{z}}(x)$. The generator being optimal, the probabilities of R_ϵ under $p(\boldsymbol{x},\boldsymbol{z})$ and $q(\boldsymbol{x},\boldsymbol{z})$ are equal. Now $p(\boldsymbol{x} \mid \boldsymbol{z}) = \delta_{x-G_x(z)}$, where δ is the Dirac delta distribution. This is enough to show that there are no x satisfying the event R_ϵ and thus $G_{\boldsymbol{x}} = G_z^{-1}$ almost everywhere. By symmetry, the same argument can be applied to show that $G_{\boldsymbol{z}} = G_{\boldsymbol{x}}^{-1}$.
The complete proof is given in [Donahue et al., 2017], in which the authors independently examine the same model structure under the name Bidirectional GAN (BiGAN). □

4.3 Related Work

Other recent papers explore hybrid approaches to generative modelling. One such approach is to relax the probabilistic interpretation of the VAE model by replacing either the KL-divergence term or the reconstruction term with variants that have better properties. The adversarial autoencoder model [Makhzani et al., 2016] replaces the KL-divergence term with a discriminator that is trained to distinguish between approximate posterior and prior samples, which provides a more flexible approach to matching the marginal $q(\boldsymbol{z})$ and the prior. Other papers explore replacing the reconstruction term with either GANs or auxiliary networks. Larsen et al. [2016] collapse the decoder of a VAE and the generator of a GAN into one network in order to supplement the reconstruction loss with a learned similarity metric. Lamb et al. [2016] use the hidden layers of a pre-trained classifier as auxiliary reconstruction losses to help the VAE focus on higher-level details when reconstructing. Dosovitskiy and Brox [2016] combine both ideas into a unified loss function.

ALI's approach is also reminiscent of the adversarial autoencoder model, which employs a GAN to distinguish between samples from the approximate posterior distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ and prior samples. However, unlike adversarial autoencoders, no explicit reconstruction loss is being optimized in ALI, and the discriminator receives joint pairs of samples $(\boldsymbol{x}, \boldsymbol{z})$ rather than marginal $\boldsymbol{z}$ samples.

Independent work by Donahue et al. [2017] proposes the same model under the name Bidirectional GAN (BiGAN), in which the authors emphasize the learned features' usefulness for auxiliary supervised and semi-supervised tasks. The main difference in terms of experimental setting is that they use a deterministic $q(\boldsymbol{z} \mid \boldsymbol{x})$ network, whereas we use a stochastic network. In our experience, this does not make a big difference when $\boldsymbol{x}$ is a deterministic function of $\boldsymbol{z}$ as the stochastic inference networks tend to become deterministic as training progresses. When using stochastic mappings from $\boldsymbol{z}$ to $\boldsymbol{x}$, the additional flexibility of stochastic posteriors is critical.

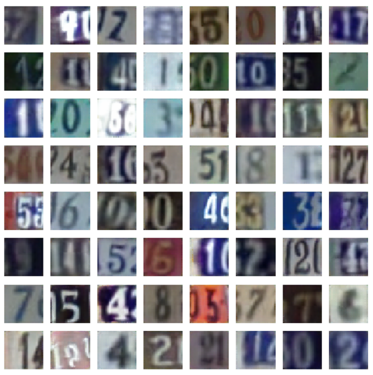
4.4 Experimental results

We applied ALI to four different datasets, namely CIFAR10 [Krizhevsky and Hinton, 2009], SVHN [Netzer et al., 2011], CelebA [Liu et al., 2015] and a center-cropped, 64×64 version of the ImageNet dataset [Russakovsky et al., 2015].[7]

Transposed convolutions are used in $G_x(\boldsymbol{z})$. This operation corresponds to

[7]The code for all experiments can be found at `https://github.com/IshmaelBelghazi/ALI`. Readers can also consult the accompanying website at `https://ishmaelbelghazi.github.io/ALI`.

the transpose of the matrix representation of a convolution, i.e., the gradient of the convolution with respect to its inputs. For more details about transposed convolutions and related operations, see Dumoulin and Visin [2016], Shi et al. [2016], Odena et al. [2016].


(a) SVHN samples.

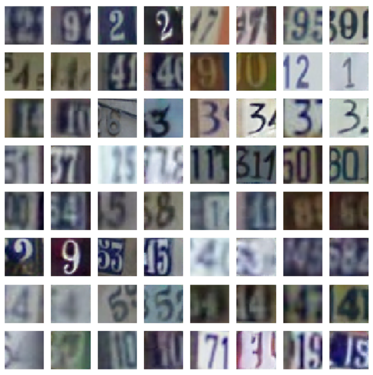
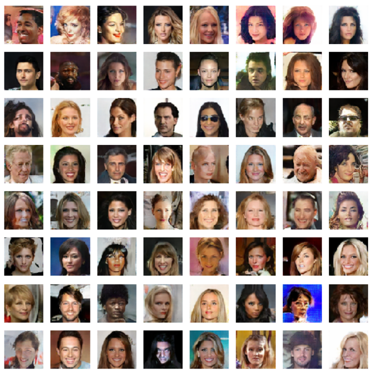

(b) SVHN reconstructions.

Figure 4.2: Samples and reconstructions on the SVHN dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions (e.g., second column contains reconstructions of the first column's validation set samples).


(a) CelebA samples.

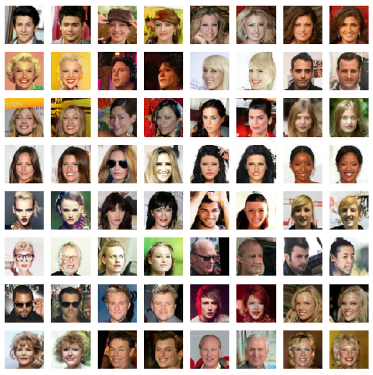
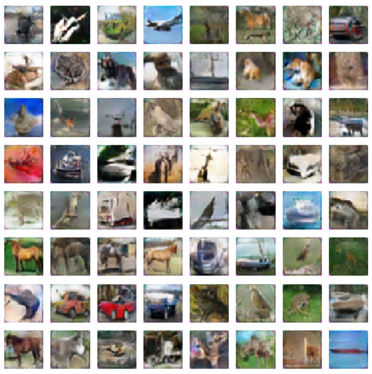

(b) CelebA reconstructions.

Figure 4.3: Samples and reconstructions on the CelebA dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions.

4.4.1 Samples and Reconstructions

For each dataset, samples are presented (Figures 4.2a, 4.3a 4.4a and 4.5a). They exhibit the same image fidelity as samples from other adversarially-trained models.

We also qualitatively evaluate the fit between the conditional distribution $q(\boldsymbol{z} \mid \boldsymbol{x})$ and the posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$ by sampling $\hat{\boldsymbol{z}} \sim q(\boldsymbol{z} \mid \boldsymbol{x})$ and


(a) CIFAR10 samples.

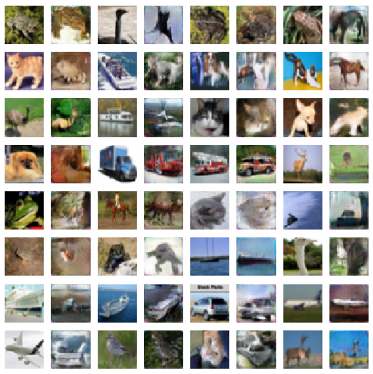
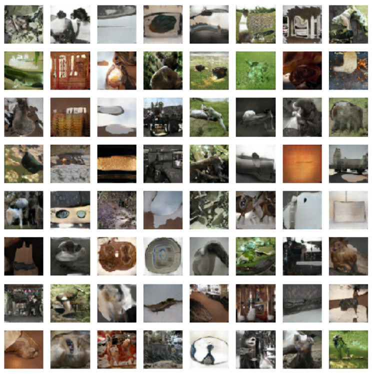

(b) CIFAR10 reconstructions.

Figure 4.4: Samples and reconstructions on the CIFAR10 dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions.


(a) Tiny ImageNet samples.

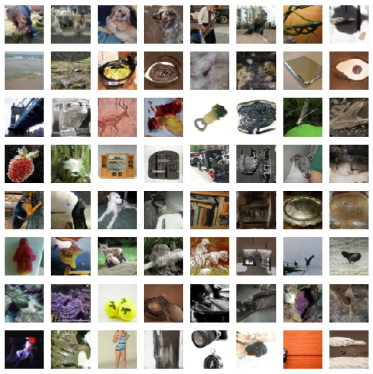

(b) Tiny ImageNet reconstructions.

Figure 4.5: Samples and reconstructions on the Tiny ImageNet dataset. For the reconstructions, odd columns are original samples from the validation set and even columns are corresponding reconstructions.

$\hat{\boldsymbol{x}} \sim p(\boldsymbol{x} \mid \boldsymbol{z} = \hat{\boldsymbol{z}})$ (Figures 4.2b, 4.3b, 4.4b and 4.5b). This corresponds to reconstructing the input in a VAE setting. Note that the ALI training objective does *not* involve an explicit reconstruction loss.

We observe that reconstructions are not always faithful reproductions of the inputs. They retain the same crispness and quality characteristic to adversarially-trained models, but oftentimes make mistakes in capturing exact object placement, colour, style and (in extreme cases) object identity. The extent to which reconstructions deviate from the inputs varies between datasets: on CIFAR10, which arguably constitutes a more complex input distribution, the model exhibits less faithful reconstructions. This leads us to believe that poor reconstructions are a sign of underfitting.

This failure mode represents an interesting departure from the blurriness char-

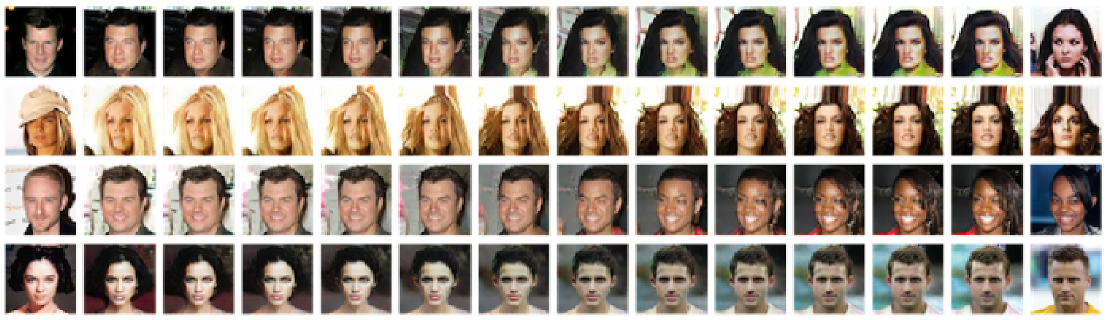


Figure 4.6: Latent space interpolations on the CelebA validation set. Left and right columns correspond to the original pairs $\mathbf{x}_1$ and $\mathbf{x}_2$, and the columns in between correspond to the decoding of latent representations interpolated linearly from $\mathbf{z}_1$ to $\mathbf{z}_2$. Unlike other adversarial approaches like DCGAN [Radford et al., 2016], ALI allows one to interpolate between actual data points.

acteristic to the typical VAE setup. We conjecture that in the underfitting regime, the latent variable representation learned by ALI is potentially more invariant to less interesting factors of variation in the input and do not devote model capacity to capturing these factors.

4.4.2 Latent space interpolations

As a sanity check for overfitting, we look at latent space interpolations between validation set examples (Figure 4.6). We sample pairs of validation set examples $\mathbf{x}_1$ and $\mathbf{x}_2$ and project them into $\mathbf{z}_1$ and $\mathbf{z}_2$ by sampling from the encoder. We then linearly interpolate between $\mathbf{z}_1$ and $\mathbf{z}_2$ and pass the intermediary points through the decoder to plot the input-space interpolations.

We observe smooth transitions between pairs of examples, and intermediary images remain believable. This is an indicator that ALI is not concentrating its probability mass exclusively around training examples, but rather has learned latent features that generalize well.

4.4.3 Semi-supervised learning

We investigate the usefulness of the latent representation learned by ALI through semi-supervised benchmarks on SVHN and CIFAR10.

We first compare with GAN on SVHN by following the procedure outlined in Radford et al. [2016]. We train an L2-SVM on the learned representations of a model trained on SVHN. The last three hidden layers of the encoder as well as its output are concatenated to form a 8960-dimensional feature vector. A 10,000 example held-out validation set is taken from the training set and is used for model selection. The SVM is trained on 1000 examples taken at random from the remainder of the training set. The test error rate is measured for 100 different SVMs trained on different random 1000-example training sets, and the average error rate is measured along with its standard deviation.

Using ALI's inference network as opposed to the discriminator to extract features, we achieve a missclassification rate that is roughly $3.00 \pm 0.50\%$ lower than reported in Radford et al. [2016] (Table 4.1), which suggests that ALI's inference mechanism is beneficial to the semi-supervised learning task.

We then investigate ALI's performance when label information is taken into account during training. We adapt the discriminative model proposed in Salimans et al. [2016]. The discriminator takes x and z as input and outputs a distribution over $K + 1$ classes, where K is the number of categories. When label information is available for $q(x, z)$ samples, the discriminator is expected to predict the label. When no label information is available, the discriminator is expected to predict $K + 1$ for $p(x, z)$ samples and $k \in \{1, \ldots, K\}$ for $q(x, z)$ samples.

Interestingly, Salimans et al. [2016] found that they required an alternative training strategy for the generator where it tries to match first-order statistics in the discriminator's intermediate activations with respect to the data distribution (they refer to this as *feature matching*). We found that ALI did not require feature matching to obtain comparable results. We achieve results competitive with the state-of-the-art, as shown in Tables 4.1 and 4.2. Table 4.2 shows that ALI offers a modest improvement over Salimans et al. [2016], more specifically for 1000 and 2000 labelled examples.

Table 4.1: SVHN test set missclassification rate

Model	Missclassification rate
VAE (M1 + M2) [Kingma et al., 2014]	36.02
SWWAE with dropout [Zhao et al., 2016]	23.56
DCGAN + L2-SVM [Radford et al., 2016]	22.18
SDGM [Maaløe et al., 2016]	16.61
GAN (feature matching) [Salimans et al., 2016]	$\mathbf{8.11 \pm 1.3}$
ALI (ours, L2-SVM)	19.14 ± 0.50
ALI (ours, no feature matching)	$\mathbf{7.42 \pm 0.65}$

We are still investigating the differences between ALI and GAN with respect to feature matching, but we conjecture that the latent representation learned by ALI is better untangled with respect to the classification task and that it generalizes better.

Table 4.2: CIFAR10 test set missclassification rate for semi-supervised learning using different numbers of trained labelled examples. For ALI, error bars correspond to 3 times the standard deviation.

Number of labelled examples	1000	2000	4000	8000
Model		Missclassification rate		
Ladder network [Rasmus et al., 2015]			20.40	
CatGAN [Springenberg, 2016]			19.58	
GAN (feature matching) [Salimans et al., 2016]	$\mathbf{21.83 \pm 2.01}$	$\mathbf{19.61 \pm 2.09}$	$\mathbf{18.63 \pm 2.32}$	$\mathbf{17.72 \pm 1.82}$
ALI (ours, no feature matching)	$\mathbf{19.98 \pm 0.89}$	$\mathbf{19.09 \pm 0.44}$	$\mathbf{17.99 \pm 1.62}$	$\mathbf{17.05 \pm 1.49}$

4.4.4 Conditional Generation

We extend ALI to match a conditional distribution. Let $\boldsymbol{y}$ represent a fully observed conditioning variable. In this setting, the value function reads

$$\begin{aligned}\min_G \max_D V(D,G) = \mathbb{E}_{q(\boldsymbol{x})\,p(\boldsymbol{y})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x},\boldsymbol{y}),\boldsymbol{y}))] \\ + \mathbb{E}_{p(\boldsymbol{z})\,p(\boldsymbol{y})}[\log(1 - D(G_x(\boldsymbol{z},\boldsymbol{y}),\boldsymbol{z},\boldsymbol{y}))]\end{aligned} \tag{4.7}$$

We apply the conditional version of ALI to CelebA using the dataset's 40 binary attributes. The attributes are linearly embedded in the encoder, decoder and discriminator. We observe how a single element of the latent space z changes with respect to variations in the attributes vector y. Conditional samples are shown in Figure 4.7.

4.4.5 Importance of learning inference jointly with generation

To highlight the role of the inference network during learning, we performed an experiment on a toy dataset for which $q(\boldsymbol{x})$ is a 2D Gaussian mixture with 25 mixture components laid out on a grid. The covariance matrices and centroids have been chosen such that the distribution exhibits lots of modes separated by large low-probability regions, which makes it a decently hard task despite the 2D nature of the dataset.

We trained ALI and GAN on 100,000 $q(\boldsymbol{x})$ samples. The decoder and discriminator architectures are identical between ALI and GAN (except for the input of the discriminator, which receives the concatenation of $\mathbf{x}$ and $\mathbf{z}$ in the ALI case). Each model was trained 10 times using Adam [Kingma and Ba, 2014] with random learning rate and β_1 values, and the weights were initialized by drawing from a Gaussian distribution with a random standard deviation.

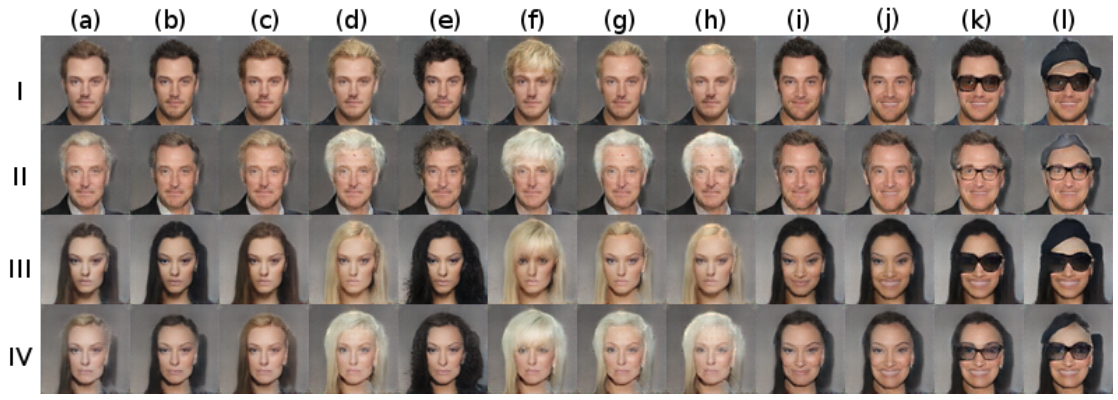


Figure 4.7: Conditional generation sequence. We sample a single fixed latent code z. Each row has a subset of attributes that are held constant across columns. The attributes are male, attractive, young for row I; male, attractive, older for row II; female, attractive, young for row III; female, attractive, older for Row IV. Attributes are then varied uniformly over rows across all columns in the following sequence: (b) black hair; (c) brown hair; (d) blond hair; (e) black hair, wavy hair; (f) blond hair, bangs; (g) blond hair, receding hairline; (h) blond hair, balding; (i) black hair, smiling; (j) black hair, smiling, mouth slightly open; (k) black hair, smiling, mouth slightly open, eyeglasses; (l) black hair, smiling, mouth slightly open, eyeglasses, wearing hat.

We measured the extent to which the trained models covered all 25 modes by drawing 10,000 samples from their $p(\boldsymbol{x})$ distribution and assigning each sample to a $q(\boldsymbol{x})$ mixture component according to the mixture responsibilities. We defined a dropped mode as one that wasnt assigned to *any* sample. Using this definition, we found that ALI models covered 13.4 ± 5.8 modes on average (min: 8, max: 25) while GAN models covered 10.4 ± 9.2 modes on average (min: 1, max: 22).

We then selected the best-covering ALI and GAN models, and the GAN model was augmented with an encoder using the *learned inverse mapping* and *post-hoc learned inference* procedures outlined in subsection 4.2.2. The encoders learned for GAN inference have the same architecture as ALIs encoder. We also trained a VAE with the same encoder-decoder architecture as ALI to outline the qualitative differences between ALI and VAE models.

We then compared each models inference capabilities by reconstructing 10,000 held-out samples from $q(\boldsymbol{x})$. Figure 4.8 summarizes the experiment. We observe the following:

- The ALI encoder models a marginal distribution $q(\boldsymbol{z})$ that matches $\boldsymbol{p(z)}$ fairly well (row 2, column a). The learned representation does a decent job at clustering and organizing the different mixture components.
- The GAN generator (row 5, columns b-c) has more trouble reaching all the modes than the ALI generator (row 5, column a), even over 10 runs of

hyperparameter search.

- Learning an inverse mapping from GAN samples does not work very well: the encoder has trouble covering the prior marginally and the way it clusters mixture components is not very well organized (row 2, column b). As discussed in subsection 4.2.2, reconstructions suffer from the generator dropping modes.
- Learning inference post-hoc doesn't work as well as training the encoder and the decoder jointly. As had been hinted at in subsection 4.2.2, it appears that adversarial training benefits from learning inference at training time in terms of mode coverage. This also negatively impacts how the latent space is organized (row 2, column c). However, it appears to be better at matching $q(\boldsymbol{z})$ and $p(\boldsymbol{z})$ than when inference is learned through inverse mapping from GAN samples.
- Due to the nature of the loss function being optimized, the VAE model covers all modes easily (row 5, column d) and excels at reconstructing data samples (row 3, column d). However, they have a much more pronounced tendency to smear out their probability density (row 5, column d) and leave "holes" in $\mathbf{q}(\mathbf{z})$ (row 2, column d). Note however that recent approaches such as Inverse Autoregressive Flow [Kingma et al., 2016] may be used to improve on this, at the cost of a more complex mathematical framework.

In summary, this experiment provides evidence that adversarial training benefits from learning an inference mechanism jointly with the decoder. Furthermore, it shows that our proposed approach for learning inference in an adversarial setting is superior to the other approaches investigated.

4.5 Conclusion

We introduced the adversarially learned inference (ALI) model, which jointly learns a generation network and an inference network using an adversarial process. The model learns mutually coherent inference and generation networks, as exhibited by its reconstructions. The induced latent variable mapping is shown to be useful, achieving results competitive with the state-of-the-art on the semi-supervised SVHN and CIFAR10 tasks.

Acknowledgements

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada. We would also like to thank the developers of Theano [Bergstra et al.,

2010, Bastien et al., 2012, Theano Development Team, 2016], Blocks and Fuel [van Merriënboer et al., 2015], which were used extensively for the paper. Finally, we would like to thank Yoshua Bengio, David Warde-Farley, Yaroslav Ganin and Laurent Dinh for their valuable feedback.

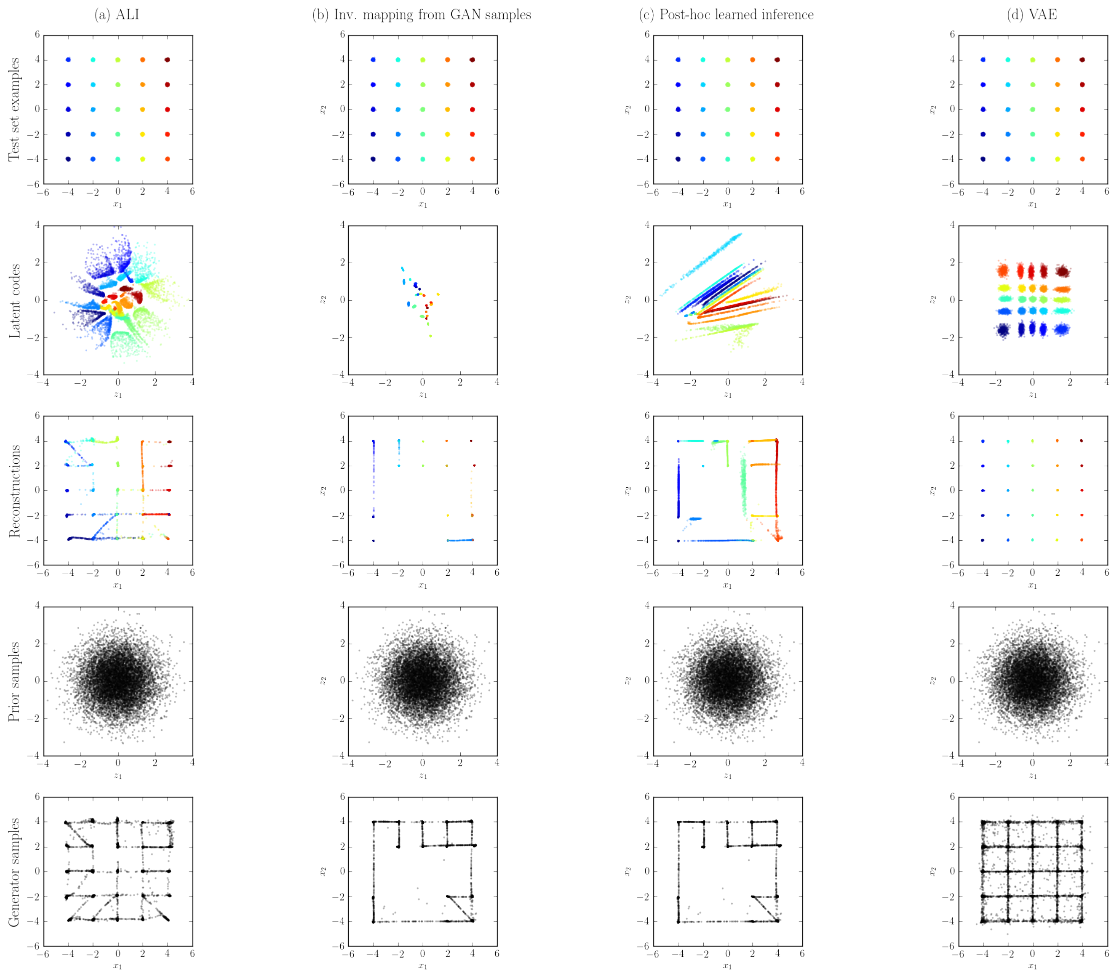


Figure 4.8: Comparison of (a) ALI, (b) GAN with an encoder learned to reconstruct latent samples (c) GAN with an encoder learned through ALI, (d) variational autoencoder (VAE) on a 2D toy dataset. The ALI model in (a) does a much better job of covering the latent space (second row) and producing good samples than the two GAN models (b, c) augmented with an inference mechanism.

Chapter 5

Video Discrimination

The last two chapters will cover my work into the part of computer vision dealing with videos. We will start by looking at discriminative modelling in videos and next chapter will move on to generation.

Note that there are many more tasks than activity recognition that possibly fall into the discrimination category. Video question and answering or video to text are examples of other tasks.

5.1 Activity recognition

Activity recognition is a classification task applied to videos. More specifically, models are asked to recognize human activity by making semantic associations from series of frames. Depending on the datasets used, the activities range from simple *jump*, *walking* or *running* to complex ones such as *soccer* or *playing instrument*. One of the earliest dataset created for this task is the KTH dataset [Schuldt et al., 2004] consisting of 6 different actions and 100 videos per action. Common benchmarks today are usually done on UCF101 [Soomro et al., 2013] and HMDB [Kuehne et al., 2011] datasets.

5.1.1 Problems

Even before going into the temporal domain, computer vision on static images is a hard problem in its own right. This can be observe from the ever increasing accuracies of the ImageNet Large Scale Visual Recognition Challenge. ILSVRC is one of the biggest yearly competition in the computer vision community since 2010. Models compete in image classification and object detection contests on an extremely large dataset. The training set for classification in 2014 contained 1.28 millions images divided in 1000 classes. The best error rates have been steadily decreasing ever since, from 28.2% missclassification in 2010 to 3.57% in 2015. Interestingly, they have discontinued since 2015 the image classification task from ILSVRC, but other challenges have remained. All in all, it shows that models still have room for improvements with regards to image tasks.

This is to be kept in mind while moving on the topic at hand, challenges in activity recognition. These are twofold: models and datasets. First of all, videos burden the learning not only with the same needs as images, but with the addition of a plethora of temporal information. Novel concepts only encountered in videos are for example camera motion, objects coming or leaving the scene or shot transitions. Non-smooth changes like two people talking in a scene and the camera's point of view alternating between the two speakers for example. What's more, throwing vision and time together imposes direct constraints on computational resources like memory and processing time and limits the exploration of models with current hardware.

Second, the current datasets available can add unwanted problems themselves to the task. Some contain too few examples to really leverage deep learning methods, for example KTH with 100 videos per class and 6 classes. Even the popular ones like UCF101, 101 actions categories and 13320 videos and HMDB, 51 classes and 6766 total examples, are not in a well better-off regime.

Moreover, they can be too simple and suffer from the fact that the class can be picked up with very few or no temporal information. UCF101 is particularly afflicted by this condition where more often than not, the classes can be found by a good pretrained vision model on a large image dataset (ImageNet) applied on a single or very few frames.

A fair counter argument could be raised: maybe that is the nature of activity recognition. It is fair to say that there are examples where this is not the case such as identifying different dance styles. They can be categorized regardless of the settings into which they are executed and the various objects the people could be wearing. Even for *soccer* or *swimming*, positively the former one is strongly associated with a soccer ball. Such object will be picked up from a single frame, but soccer could easily be played indoors and it is unclear whether the best models would generalize even to soccer in an indoor gym.

Other datasets, such as Sports1M [Karpathy et al., 2014] or DeepMind's Kinetics [Kay et al., 2017] are big but noisy. Acquiring data and having clean labels is a costly enterprise. These large datasets have been assembling data in semi-automated ways which have yielded somewhat unsatisfying results so far. They can contain many examples which do not or almost not show their label making them hard to process. The video community unfortunately doesn't have access to the equivalent of the MNIST dataset.

5.1.2 Solutions

Recent solutions to activity recognition have mostly focused on how to feed and modify convolutional networks. All results reviewed here are summarized in Table 5.1 at the end of this section.

There is still a strong emphasis on using hand-crafted features such as STIP, Fisher vector and optical flow (OF). Wang and Schmid [2013a] is the model that will set the stage, only relying on hand-crafted features which they name *improved dense trajectory* (iDT). iDT is actually not a hand-crafted feature in itself, it is a name for the process of computing multiple video descriptors (HOG, HOF and/or MBH) along an optical flow trajectory and aggregating them in their particular way. Their method is quite involved and we will not discuss on how to perform training in a purely hand-crafted features setup (one could say it is also against the deep learning spirit). The only thing to know is that after combining all these features they use SVM as the final classifier. Be that as it may, we need at least to introduce optical flow because of its heavy use in almost all models presented below.

Optical flow is easily understood as a set of displacement vector fields $\mathbf{d}_t$ between pairs of consecutive frames t and $t+1$. There are two components, d_t^x, d_t^y, per pixel location for optical flow on an image following each pixel's motion across frames.

The *two-streams* model [Simonyan and Zisserman, 2014] was proposed as a strong choice for activity recognition and was extended with the temporal segment network [Wang et al., 2016] which holds the current SOTA on UCF101 and HMDB. The idea behind two-streams is very simple:

Consider one video example being described by a sequence of K frames $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k\}$ and a sequence of J hand-crafted features $Y = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_j\}$ computed from X. Be wary that J does not necessarily equal K as in the case for optical flow ($J = K-1$). OF will be used as Y for it was found in their paper to give the highest scores with two-streams. We now need two convolutional networks $f_{\boldsymbol{\theta}}$, $g_{\boldsymbol{\phi}}$ parametrized by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ respectively and with both a final softmax layer giving a classification score as output. One stream is on a random static frame $\mathbf{x}_i \sim X$ and the second stream takes a random subset L ($L = 10$ was found sufficient) $\hat{Y} = \{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, ..., \hat{\mathbf{y}}_l\} \sim Y$ as input. The final result is the average classification score from $f_{\boldsymbol{\theta}}(\mathbf{x}_i)$ and $g_{\boldsymbol{\phi}}(\hat{Y})$[8].

As simple as it is, this model grabbed the state-of-the-art in 2014 on par with the pure hand-crafted features and has set its standard ever since. In this paper and all subsequent ones using two-streams, $f_{\boldsymbol{\theta}}(\mathbf{x}_i)$ is called the spatial CNN and $g_{\boldsymbol{\phi}}(\hat{Y})$ is called the temporal CNN. $f_{\boldsymbol{\theta}}$ is usually pretrained on the ILSVRC dataset and

[8]The implementation detail of two-streams for giving a sequence to a CNN is done simply by stacking all the $\hat{Y}$ on the channel axis instead.

gains a massive boost in performance with 72.8% accuracy compared with 52.3% when trained from scratch on UCF101. g_ϕ alone achieves 81.2% and together, they end up with 88.0% classification accuracy. This suggests the presence of the potential problem mentioned earlier: it is not a model which tries to learn explicitly or need much of temporal information (it only uses 10 frames for OF) considering all the pretrained spatial network is doing. This might well be explained by the fact that there is not enough temporal dependencies for classification in UCF101. Another argument along this line is the small 0.6% gain by coupling two-streams with LSTM [Ng et al., 2015].

A year later, trajectory-pooled deep-convolutional descriptor [Wang et al., 2015] (TDD) took the idea even further. They trained a two-streams model and used its convolutional features, any intermediate stack of feature maps by any of the convolutional layers, as input to the feature extraction mechanism of iDT. This approach is a departure from neural network end-to-end learning as it makes heavily use of different modules on top of optical flow to produce an accuracy of 91.5%.

Finally in 2016, the temporal segment networks (TSN) model was proposed and improved the state-of-the-art to 94.2% (best as of writing). It used only the original two-streams idea and removed the non end-to-end features enhancement of TDD. TSN use two-streams multiple times on the same video by sampling various shorter subsets of clips and finally averaging all the results. As it can be observed, none of these models treat time with a fully learnable framework and rely very indirectly to this dimension through hand-crafted features. This idea saw another iteration with key volume mining deep framework [Zhu et al., 2016] (KVMF) where the fundamental differences with TSN appear to be very thin. KVMF propose to sample multiple cubes of 3D spatio-temporal subset of a video and train these inputs in a two-streams fashion. Their method for sampling these 3D cubes seems to be their differentiating factor.

There are other models that do not use a two-streams framework and try a more direct approach to time modelling which are worth mentioning. Unsupervised LSTM [Srivastava et al., 2015], GRU-RCN [Ballas et al., 2016], Convolutional 3D network [Tran et al., 2015] and long-term temporal convolutions [Varol et al., 2017] (LTC).

Unsupervised LSTM is an autoencoder approach using LSTM to first train the encoder/decoder pair to predict the future and reconstruct the past and then transfer the unsupervised knowledge onto activity recognition. They were able to almost match SOTA of their time with a score of 84.3%. More details of this model will be given on subsection 6.1.1.

GRU-RCN, which are *gated recurrent unit*[9] upgraded to use convolution, have shown SOTA matching results (when published) without the framework of two-

[9]GRU have not been presented in chapter 2 but we can think of them as being very close sisters to LSTM.

streams (it does use optical flow and pretraining). They achieved 85.7% accuracy on UCF101. With unsupervised LSTM, they are in the line of hope for sequences RNN-like approaches to this task.

Convolutional 3D networks presents itself as a strong alternative for UCF101, with 85.2% accuracy (using a SVM as a classifier) and 90.4% adding iDT. An interesting aspect of this model is the net result of Conv3D, they compress the time dimension. There is no reason so far indicating that this compressing cannot be done on the time axis like it is done on the spatial one as well. A potential drawback however is if the same interpretation happening in 2D space applies in 3D. There is evidence that lower layers near pixel space in CNN become edges detectors. If this holds in 3D, it implies a great waste of capacity for detection in time since everything not moving will create edges in this dimension. A clear advantage of this method resides in its practicability. Convolutional operators have robust implementation in current hardware offering unbeatable computational performance. LTC is in all theoretical aspects the same as Tran et al. [2015] with longer sequences as input for their Conv3D.

Models	**UCF101**-RGB	**UCF101**	**HMDB**
Two-Streams spatial CNN [Simonyan and Zisserman, 2014]	72.8*	72.8*	40.5
Two-Streams temporal CNN [Simonyan and Zisserman, 2014]	—	81.2*	54.6
Unsupervised LSTM [Srivastava et al., 2015]	75.8	84.3	74
GRU-RCN [Ballas et al., 2016]	80.7*	85.7*	—
iDT‡ [Wang and Schmid, 2013a,b]	85.9	85.9	57.2
Two-Streams [Simonyan and Zisserman, 2014]	—	88.0	59.4
Two-Streams + LSTM† [Ng et al., 2015]	—	88.6	—
Conv3D [Tran et al., 2015]	85.2	90.4	—
TDD† [Wang et al., 2015]	—	91.5	65.9
LTC [Varol et al., 2017]	82.4	92.7	67.2
KVMF† [Zhu et al., 2016]	—	93.1	63.3
TSN† [Wang et al., 2016]	—	**94.2**	**69.4**

Table 5.1: State-of-the-art progress for activity recognition on the two common benchmarks UCF101 (average on 3 splits) and HMDB sorted increasingly for the third column's scores. The accuracies are in %. UCF101's results are split between the first column that displays scores without using additional features such as OF and the second which is using all the heavy artillery the authors could think of. Note that almost *none* of these papers show scores for their models without pretraining, usually done on Sports1M. * is to denote that these scores were computed only using split 1 of UCF101. † is to emphasize models built on top of two-streams. ‡ is to remind that this model is only with hand-crafted feature so there is no differences between its two column's scores.

5.2 Dynamics dataset

The issues discussed in last section motivated the search for a new dataset. This section will present unreleased work that was done with Jakub Sygnowski while he was an intern during winter and spring semesters 2017 at MILA. The contributions are equal with respect to building the dataset while the preliminary results were done by myself.

5.2.1 Design

With the previously discussed problems of activity recognition in mind, we identified the following conditions for the design of a new dataset:

1. Strong temporal dependence
2. Low spatial dependence
3. Easy to process
4. Plentiful

Condition 1 penalizes any solution that does not try to model time directly and gain almost half the accuracy score by looking at a single frame. The second condition will mitigate trained ImageNet model and emphasize even more the first one. This is not arguing that there is something wrong in pretraining. Quite the opposite, almost everything learnable in an image in reality should be transferable to video and models should not learn to *see* every time. The goal here is for a toy task that disentangles learning in video from the appearance and concentrate only in the time relationships. The last two conditions are really there to have deep learning kick in. Past a fuzzy threshold, enough data should make deep learning techniques take off and win over hand-crafted features.

One could argue that the second condition enforces a constraint that is absent from activity recognition. Spotting a soccer ball or a hockey stick is a static image-like feature that instantly yields the class that is being sought. Interestingly, it is an open question as to how much content can be taken off from *soccer* for a human to stop associating the class. This is inherent to the concept of semantics and for more complicated classes raises valid philosophical considerations beyond the scope of this text. Nevertheless, the *low spatial dependence* constraint might indeed create a gap of transferability with activity recognition but remains well grounded in the problem of video.

From these considerations the choice satisfying all four conditions becomes this question: What is a relatively easy task that humans can do which requires them to watch for a few seconds and where only a snapshot has no information in it? Also, what would be a new dataset in video that could emulate the success in research development like MNIST brought to images.

Physical dynamics are one possible options. An interesting fact in its own right is that we have very limited precision when doing this task ourselves. Without pen, paper, mathematics and physics we would have not gone really far compared to what is achieved today. However, it is still possible for humans to output correct predictions in simple physical scenario with a minimum of concentration. We choose this task for our dataset.

We simulated simple physical dynamics using Bullet Physics SDK, a real-time collision detection and multi-physics simulator. In its current state, which is subject to changes in the future, the dataset contains 10 classes made by composition of three different path and speed types plus an additional rotation class:

1. Path type:
   (a) Linear
   (b) Sinusoidal
   (c) Hypocloidal
2. Speed type:
   (a) Constant
   (b) Accelerating
   (c) Bouncing
3. Rotation

The full set of motion classes is given by the set product of the *path* set and *speed* set plus rotation, so technically it can be broken down into 7 simpler classes. The appearance of the objects used for movements are randomized in shape, colour and size. The texture of the background is randomized as well. The dataset is separated as follow: 5000 examples per class in the training set, 500 ex./cl. for validation set and 500 ex./cl. for test set.

5.2.2 Preliminary results

Table Table 5.2 shows the preliminary results of different models that were tried on this task. All models were only trained one or two times and there were no hyperparameter search or fine-tuning. The two-streams model was trained precomputing optical flow from the dataset. The relation network is a model inspired on Santoro et al. [2017] that is explained in subsection 6.3.4. I also took the test on a subset of the validation set containing only 10 examples per class.

The implementation of all models tried to follow the same pattern to make them as comparable as possible. Since they all involve downsampling the image input up to a 1×1 representation for classification, all their convolutions pattern are somewhat the same except for two-streams. Since this one is applied directly on precomputed optical flows of 10 time step, it had more memory to spare. We adapted a VGG-like [Simonyan and Zisserman, 2015] architecture meaning two half padded 3×3 convolutions of stride 1×1 followed by a pooling operation. A fully connected layer sits at the end for classification.

All other models use half padded 3×3 convolution with stride 2×2 resulting in downsampling both image dimensions by 2 after each layer. This could well be too aggressive, but it is the unfortunate compromise with current available hardware.

Conv3D follows the same pattern but with different kernel size for the time dimension. It is bigger at the beginning for bigger compression early on and it decreases with depth. A fully connected layer takes at the end a concatenation of all remaining timesteps there is to produce a classification score.

The ConvLSTM[10] model applies two convolutional LSTM late in the stack (far from pixel space representation where it is very memory intensive). One of them is always at the end and its last timestep outputs what is taken by the fully connected for classification.

Conv3D + ConvLSTM is a merge of these two architectures.

Models	**Train**	**Valid**
Two-Streams	90	88
RelationNetwork	0.2	74
ConvLSTM	0.2	72
Conv3D	0.3	40
Conv3D + ConvLSTM	**0.3**	**36**
Myself (*subset of valid*)		8

Table 5.2: Classification error rate of various models on training and validation set of Dynamics dataset. Errors are in %.

The first striking result from this table is the inability of two-streams to train, it is stuck in an underfitting regime. It is to be noted that this model is actually only the temporal network so there is technically only one stream. We can view this model basically as a CNN applied to an image where all its channels are optical flows (this way of feeding the OF is straight from the paper). For sake of completeness, the real two-streams was tried, with the added spatial net, and it yielded the same score.

[10]In practice deep ConvLSTM network are never *only* composed of ConvLSTM for each layer in the stack, the strain on computational resources builds up too fast. A reasonable network with today's resources (1 computer) for, say 64x64 images, would use 2-3 ConvLSTM depending on the aggressivity of the downsampling policy and they will be closer to the output rather than being near the full sized input. The other layers would normally be regular 2D-CNN treating time as an independent axis.

Since these results are very early, we let the doubt persists as of writing and do not formally cast two-streams as a failure on this task. A more solid stance would require the optical flow to be submitted to a thorough quality check, pretrain it on ILSVRC and a hyperparameter search should be done (as with all other models).

Nevertheless, two things are worth mentioning even in such an early stage. Clearly taking time into account is important by the fact that Conv3D has a 32% lower missclassification. This dataset never displays two static frames since the object is always in motion, this could in effect kill the edge hypothesis weakness discussed earlier and draw all the potential of this model. Also, it is a mystery that the ConvLSTM is not able to generalize much by itself but is actually helping when mixed with Conv3D. This suggests that first a compression of space-time for the inputs in the early layers then followed by a recurrent modelling at the features levels can make for a good combination.

If this is the real winning model for this dataset and if it can transfer to activity recognition or other video tasks is left as future work.

Chapter 6

Video Generation

The final chapter's content is built on all previous chapters. The first section is dedicated to current models in video generation and the following ones present unpublished work done with tremendous help from Nicolas Ballas and Jakub Sygnowski during my master at MILA.

Video generation can be broadly divided into two level of difficulties. The "easy" one, next frame prediction, concerns itself with learning how to output a frame based on the previous ones. Almost all models considered in section 6.1 are in this category. The hard one is to come up with a whole video at once like generating a whole sequence of frames showing *gardening* out of a representation of the word. As we can see, this distinction is about the amount of available information relative to what is asked to be generated. To the best of our knowledge, only Vondrick et al. [2016] presented in subsection 6.1.3 has published somewhat reasonable results on this topic.

One final note before diving into the chapter, as the reader might have noticed, it is getting hard to keep strict formality while drawing the graphs. The models here are huge stacks of usually multiple components. We hope the context will convey enough information to understand the general picture of the models when a graph is shown.

6.1 Current models

6.1.1 Unsupervised LSTM

We have already seen Srivastava et al. [2015] in chapter 5 for activity recognition but the first experiments of their paper were actually on generation. It is worth going a little bit into details since in essence, a lot of models using RNN are built on variants or with modified parts of such architecture. Many models will start by casting their solution in an autoencoder like framework, first an encoder will downsample the video in a recursive fashion and second a decoder-like path will bring it back to its original size. All the design challenges are on how to handle the recurrences. The major motivation behind encoding and then employing a recurrence is because of the high dimensionality of pixel space. It is going to blow out of proportions, regarding parameters and/or computational requirements, when working on a **sequence** of pixels.

Srivastava et al. [2015] proposes to perform generation with a global set of three parameters' set $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3$ where $\boldsymbol{\theta}_k$ is the set of all LSTM parameters. Their method is to use three LSTM, one encoder that will produced only a single output, the learned video representation, and two decoders. One decoder will try to reconstruct the input and the second decoder will try to do future prediction. They call this the Composite LSTM and it is illustrated on Figure 6.1.

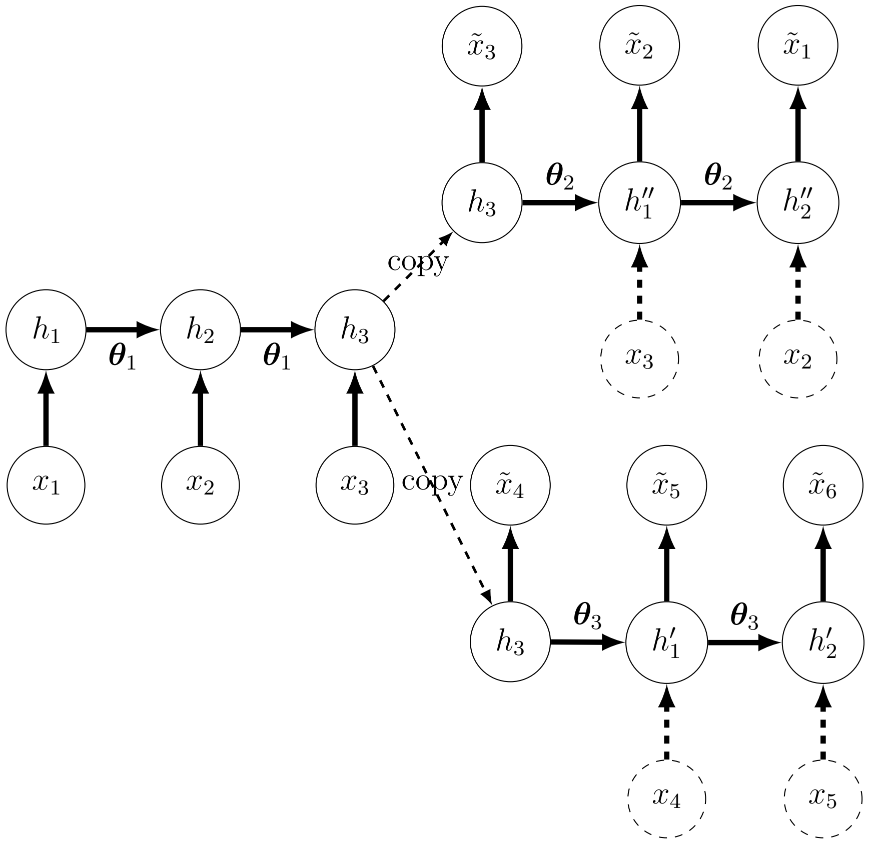


Figure 6.1: The Composite Model for three timesteps, using three LSTM parametrized as the encoder $\boldsymbol{\theta}_1$ with hidden states $\mathbf{h}$, the future decoder $\boldsymbol{\theta}_3$ with hidden states $\mathbf{h}'$ and the past decoder $\boldsymbol{\theta}_2$ with hidden states $\mathbf{h}''$. True inputs are noted from $\mathbf{x}$ and predicted/reconstructed ones from $\tilde{\mathbf{x}}$. This also shows the optional conditioning represented as the dashed circles. The first hidden states of both decoders is the copied latent representation learned by the encoder.

They explore the conditional and unconditional modes for the decoder, i. e.

whether giving access to the previous frame to the decoder or only the encoder's representation. There is an interesting trade-off happening between the two modes of training. On one hand, it makes no sense to use it if there is no multiple modes in the target distribution. Such is the case for the reconstruction path which is unimodal in its target space. There is also the case for long-range dependencies. It will be easier to focus on the short-range correlations and the long-range gradient will be small if there is preconditioning on the previous frames. On the other hand, the future prediction path is very multimodal in its target space, not everything needed to predict the future is ever observed in the input.

They obtain reasonable results on moving MNIST (the version of MNIST where multiple digits are moving around the scene) and inconclusive generations for a more complicated dataset. The results suffer from the blurry autoencoder affliction.

The original paper introducing convolutional LSTM in subsection 2.3.2 extended this idea for their experiments. They achieved similar results with the benefit of having less parameters from the convolutional switch.

6.1.2 Multi-scale

Xue et al. [2016] and Mathieu et al. [2016] instead opt for a multi-scale approach. Both papers are harder to compare since they use different datasets to experiment on and different losses, the former is using a VAE-based training while the later a GAN. However they share two common ideas: a departure from a RNN framework for only convolutional networks and to process the video in a multi-scale like way.

The choice to drop RNN seems unconvincing. We find it hard to imagine the CNN holding all in its parameters the static pixels understanding **and** also their time-wise relationships. We suspect this is why they do not show more than 1 or 2 frames prediction in their papers. The observations of chapter 3 though are encountered again, VAE samples tend to be blurry whereas the adversarial loss gives us clear looking predictions.

The multi-scale processing differs among both papers. Mathieu et al. [2016] frames multi-scale this way: Let $\hat{\mathbf{y}}_{t+1}^k$ be the frame to predict at scale k, $X^k = \{\mathbf{x}_1^k, ..., \mathbf{x}_t^k\}$ the video sequence at scale k to use as input and u_k the upscaling function from $k-1$ to k. The final output is defined recursively with network $G_{\boldsymbol{\theta}}$

$$\hat{\mathbf{y}}_{t+1}^k = u_k(\hat{\mathbf{y}}_{t+1}^{k-1}) + G_{\boldsymbol{\theta}}(\mathbf{x}_0^k, ..., \mathbf{x}_t^k, u_k(\hat{\mathbf{y}}_{t+1}^{k-1}))$$

The goal here is to work at many different scales to produce a final prediction which seems like a good idea but unfortunately isn't investigated further. The paper pretrains on Sports1M and do not say if it learns the function u to go from bigger and bigger scale nor how does it downscale the frames. For these reasons, it is quite hard to tell the reach of this form of multi-scaling.

Xue et al. [2016] take a more classical autoencoder approach and their objective is to reproduce the motion. There is an issue already being raised here as they hard code motions to be the difference in pixel space between two frames. L_2 measure between two frames is a poor proxy for motion as would argue optical flow papers from last chapter (think of camera motion). Their method is as follow:

They use multiple encoders at different pixel scale to create a multi-scale stack of feature maps. They put the VAE-like path with motion as input and a very large output. The particular thing here is that they use this output as kernel values for convolving the stack of multi-scale feature maps, they name this *cross convolution*. They finally recombine everything with another decoder neural network to try to match the motion. See Figure 6.2 for a clearer picture of what is going on.

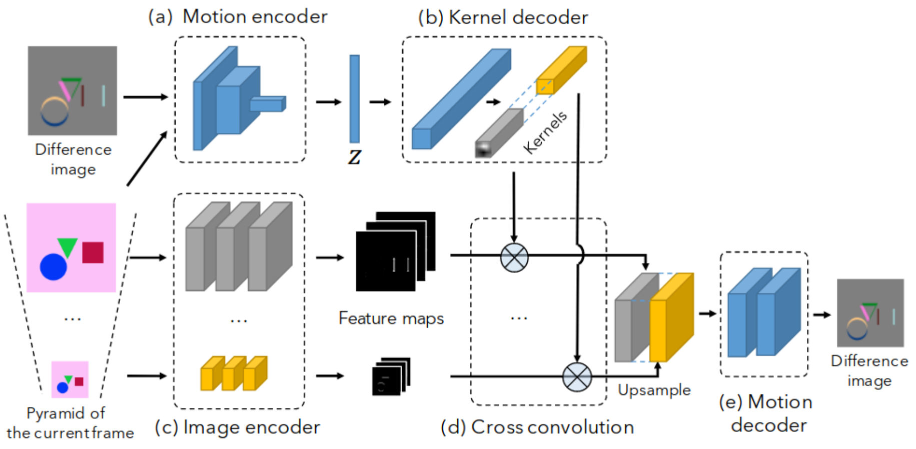


Figure 6.2: Figure 3 of Xue et al. [2016] with permission from the authors.

This idea is interesting as they have a lot of freedom in the network for learning and also they try to understand what is motion. A limitation is inherently in their first assumption of the hard coded description of motion. The generated motion in their resulting experiments evidently get blurred as the byproduct of VAE training.

6.1.3 Foreground vs background

The term *two-streams* resurfaces again with Vondrick et al. [2016] but is not to be confused with the one we've already seen. This time, this paper decides to split the modelling in two, one foreground stream and one background stream all in a GAN-like setup.

Noise sampled from the simple prior climbs two towers, the background CNN tower and the Conv3D foreground tower which splits at some point on the upsampling path into another Conv3D arm called the mask, all together forming generator G:

$$G(\mathbf{z}) = m \odot f + (1 - m) \odot b$$

for low-dimensional latent code $\mathbf{z} \in \mathbb{R}^d$, 2D image b and space-time cuboid m and f. Doing so, object dynamics is learned through the interaction between m and f with the static background b. The obvious limitation of this model is the static background, without modifications this will never be able to model camera motion. Apart from this though, the idea is very promising since the whole object's evolution is learned and more than only the motion. What is meant here is that it tries to model the inherent and one of the biggest challenge to video, a 3D world projected on a 2D screen. For example, this network can potentially hold in its parameters a person turning on itself, hands, arms and legs all rotating and going in and out of the pixel space representation from frame to frame.

The generated videos are definitely not there yet, but they are still impressive with regards of a whole sequence generation out of pure random noise. Lastly, their paper turns their GAN into a conditional GAN with a single image as conditioning. This has the nice property of taking the foreground tower into a next frame prediction machine. The model never really generates the "correct" future, but still shows a certain degree of plausibility.

They are able to bypass the blow-up of working with sequences of pixels since they use Conv3D. This however could be possible culprit in the loss of sample quality. Indeed they need to upsample time and they do so following the exact same logic of upsampling layers in CNN which is unclear if it is a good idea.

6.1.4 Disentangled representations

As of writing, the best KTH next frame generation were achieved by Denton and Birodkar [2017] and by an impressive margin, their model can generate very far ahead in time with few early accumulation of errors. Let's go through their "disentangled representations network" (DrNet).

DrNet describe a non-GAN adversarial training in an encoder/decoder setup. DrNet's main contribution is to enforce a factorization between time-invariant (content) and time-evolving (pose) features and give full learning over this to the network. Those features then can be extracted with the decoder and reconstruct frames. It is best understood by going through the pieces of the proposed training objective $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_{\text{rec}}(E_c, E_p, D) + \alpha\mathcal{L}_{\text{sim}}(E_c) + \beta(\mathcal{L}_{\text{adv}}(E_p) + \mathcal{L}_{\text{adv}}(C))$$

with hyper-parameter α and β and model pieces content encoder E_c, pose encoder E_p, decoder D and scene discriminator network C. The reconstruction loss is pretty standard

$$\mathcal{L}_{\text{rec}}(E_c, E_p, D) = ||D(E_c(\mathbf{x}_t), E_p(\mathbf{x}_{t+k})) - \mathbf{x}_{t+k}||_2^2$$

Notice that E_p takes the future $\mathbf{x}_{t+k}$ from frame $\mathbf{x}_t$ offseted in time by k. The similarity loss will enforce the content to be time-invariant

$$\mathcal{L}_{\text{sim}}(E_c) = ||E_c(\mathbf{x}_t) - E_c(\mathbf{x}_{t+k})||_2^2$$

$\mathcal{L}_{\text{adv}}$ is the bulk of Denton and Birodkar [2017]'s contribution. This loss will impose the assumption that objects's semantics do not change within a video but do between different videos[11]. This adversarial game (differs from a GAN) is played between a discriminator that will try to classify pair of pose features as coming from the same video $E_p(\mathbf{x}_t^i), E_p(\mathbf{x}_{t+k}^i)$ or not $E_p(\mathbf{x}_t^i), E_p(\mathbf{x}_{t+k}^j)$ and the pose encoder that will try to fool said discriminator by wanting to maximize its entropy.

$$\begin{aligned}\mathcal{L}_{\text{adv}}(C) &= \log(C(E_p(\mathbf{x}_t^i), E_p(\mathbf{x}_{t+k}^i))) + \log(1 - C(E_p(\mathbf{x}_t^i), E_p(\mathbf{x}_{t+k}^j)))\\ \mathcal{L}_{\text{adv}}(E_p) &= \frac{1}{2}\log(C(E_p(\mathbf{x}_t^i), E_p(\mathbf{x}_{t+k}^i))) + \frac{1}{2}\log(1 - C(E_p(\mathbf{x}_t^i), E_p(\mathbf{x}_{t+k}^i)))\end{aligned}$$

They conjecture without going further into mathematics that at optimality this will yield the desired separation whereas the pose features won't carry any content information about objects.

They have, as others before, moved training into feature space and a strong point for their model is that they built in time-invariance/variance into their training setup. This gives it full freedom to construct features which will interact well, in line with Vondrick et al. [2016] but without the foreground vs background strict constraint. DrNet's results does show that the content encoder capture semantics, such as colours and objects' type (like a red chair), and the pose encoder can rotate them.

This enables them as well the possibility to do next frame prediction. After training their encoder/decoder, they further train a LSTM taking in input the content and pose features from a single image and then output pose features. They

[11]Ex.: A red chair does not become a blue sky.

recursively use them with the same content features through time to generate the video. They backprop on the LSTM with a L_2 loss between its pose and the pose encoder's (taken as ground truth, remember the encoders were already trained and are fixed here). At test time, they get content and pose features from the LSTM and run it. They generate this way hands-down the best KTH samples with an amazingly simple model, only a traditional convolutional autoencoder coupled with a LSTM.

A weak point against DrNet is that content *can* change through a video. Content will normally appear and disappear, it does not happen in KTH but UCF101 is full of those examples. Also, camera motion is a type of content and pose changing simultaneously and renders such clean factorization invalid.

6.1.5 Forward pixel prediction

The final model we will cover in this section is the convolutional dynamic neural advection (CDNA) from Finn et al. [2016]. Their idea is to predict motion of each pixel from a frame at t as being a local change with respect to this pixel at $t-1$. In other words, a pixel will not teleport itself at a random location. Going from frame to frame, it should move smoothly and locally from its location. They also further make the assumptions that a group of pixels can actually move together. Pixels of the same objects should move as a group. More formally, for transformation kernel $\hat{m}$ of spatial size κ, previous image $\mathbf{x}_{t-1}$ and predicted image $\mathbf{y}_t$ at each pixel (i, j)

$$\mathbf{y}_t[i,j] = \sum_{k\in(-\kappa,\kappa)} \sum_{l\in(-\kappa,\kappa)} \hat{m}[k,l]\mathbf{x}_{t-1}[i-k, j-l] \tag{6.1}$$

Since objects do not all subscribe to the same motion pattern, they cast the problem instead as predicting a discrete distribution of C $\{\hat{m}^c\}$ over each pixel. Doing so, they learn a 3D mask Ξ which is passed through a *softmax* layer on the c axis. They further expands $\mathbf{y}$ to actually be a tensor in three dimensions with an additional channel axis onto which each $\hat{m}^c$ applications are stored. The final prediction is $\hat{\mathbf{y}}_t = \sum_c \mathbf{y}_t[c] \odot \Xi_c$. They also include a background mask where they allow the model to copy each pixel directly from the previous frame and grant it the ability to fill a previously occluded pixel which might need a novel generation.

Finn et al. [2016] argue that their model's benefits are two-fold: first, every predicted motions $\hat{m}^c$ can be reused for multiple pixels and second, it forces the model to hold object centric representations without further supervision. They show in their experiments that indeed, CDNA learn to mask out blobs of pixels, objects, moving in consistent directions. Practically, they use a very heavy architecture of multiple ConvLSTM to produce all the components of CDNA.

6.2 Recurrent adversarially learned inference

In this section, we tried to upgrade ALI of chapter 4 to a recurrent version: recurrent adversarially learned inference (RALI). A reason behind such an enterprise lies in the promise of the abstract latent space. We do not need further convincing that videos are big bulky objects to work with. An encoded abstract space offers the many advantages of being able to compress, manipulate and do inference (a better space to operate on for activity recognition for example). We only tried RALI for the hard problem of generating a whole video.

Many variations of Figure 4.1 can be made when going recurrent and $\mathbf{x}$ is now fixed as a sequence of images. It moves most design choices to be on how to structure $\mathbf{z}$ and of course G's and D's respective architectures. Let's start with the generator.

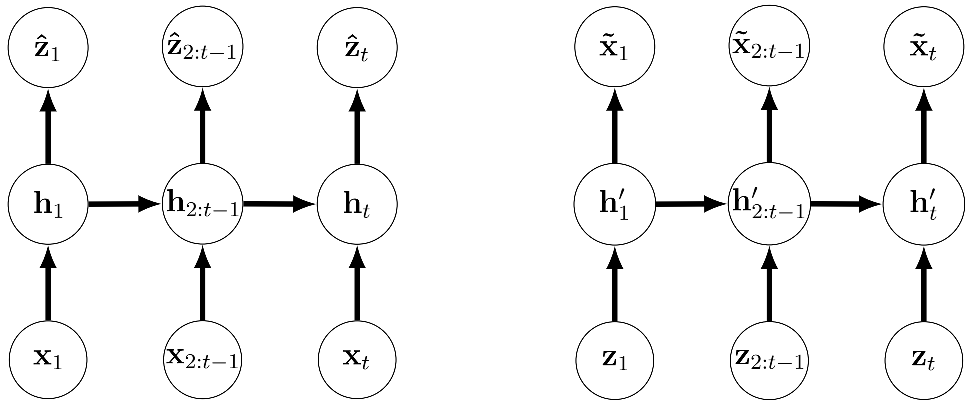


Figure 6.3: Encoder (*left*) and decoder (*right*) for RALI with a sequence of $\mathbf{z}_t$ for a video. The subscript $2 : t-1$ denotes that this graph shows in total T time step. Note that in practice, the network will be deep and have multiple stages of hidden state $\mathbf{h}$ and $\mathbf{h}'$.

Figure 6.3 depicts the scenario when the latent space is taken to be recurrent. Figure 6.4 pictures the case when $\mathbf{z}$ is taken to be one vector. Both generators are deep ConvLSTM network. As for the discriminator, in all cases its design will look closely to the encoder of Figure 6.4 (*left*) but rather with a single scalar output for the class fake/real instead of a vector.

6.2.1 Results

All experiments were made on downsampled KTH (64×64). Most would be qualified as failures except one where we can at least see a resemblance of the dataset, see Figure 6.5.

The setup described by Figure 6.4 fared better at generations rather than having

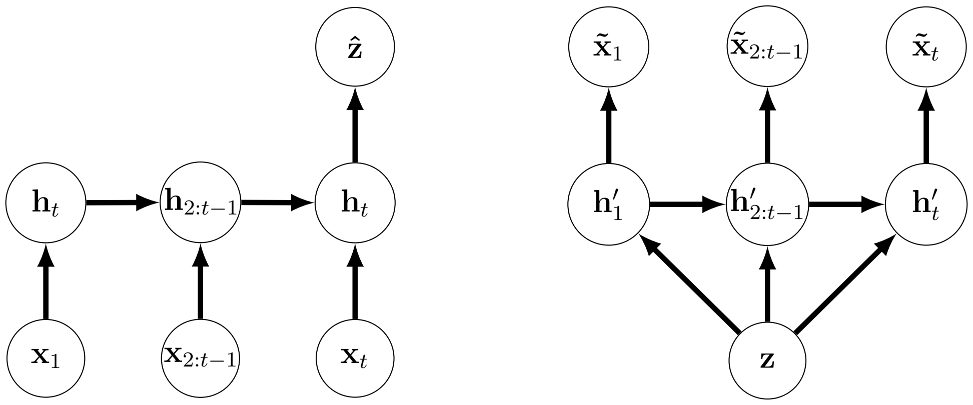


Figure 6.4: Encoder (*left*) and decoder (*right*) for RALI. with a single $\mathbf{z}$ for a video. The subscript $2 : t-1$ denotes that this graph shows in total T time step. The $\mathbf{z}$ on the decoder side can be fed to the first recurrence in many ways. It can be copied for all time step, only drives the first the first $\mathbf{h}'_1$ or be transformed for example. Our best experiment was done by using a MLP that parametrized these connections (shared parameters across time). Note that in practice, the network will be deep and have multiple stages of hidden state $\mathbf{h}$ and $\mathbf{h}'$.

a sequence of $\mathbf{z}$. This level of separation among the global failures is reasonable. Indeed, there is a lot of entropy coming in when trying to inject noise at every timestep. Of course both latent structures can be seen as natural assumptions. The former would take into account that the random changes in every frame is well suited for a sequence of random numbers while the latter will convince us that you can encode all a video as a theme in one single vector. A sequence of $\mathbf{z}$ this way though has the weakness that it fails to capture one strong video observation: much of a frame comes from the past one. That being said, it puts enormous pressure on trying to model new noise every time. For a simple dataset such as KTH, it seems reasonable that a lower entropy model wins. Unfortunately, we did so by using a MLP for the mapping from the latent vector to the whole sequence. This has the regrettable consequence to fix in advance the number of timesteps that will be generated.

There are a few origins for the failure modes of this model:

1. KTH has not enough data
2. GAN instability
3. ALI instability
4. RNN instability
5. All of the above

Obviously, the fifth reason will be the answer of choice. This is not meant to convey that fundamentally RALI is a bad choice for video generation, it is rather a claim for attention to underlying sub-problems. The first is true in the sense that we have empirically observed overfitting on all trials of our discriminator, however, the DrNet model proved us that decent frame generation on KTH is possible with neural networks. It is important to note that the theoretical literature, and the practical solutions it spawned, on GAN addressing the second point were not available at the time (this field *is* fast). Even if they were, ALI's instability is still not to rule out and it has not been investigated as much as GAN. The instability contest does not have a clear winner so far. chapter 2 went over RNN issues, but the real point is that recurrent GAN is still an open question on how to deal with properly. There have been problems trying to make the recurrent extension take off in many other applications and this is an important concern that will follow us in the next sections.

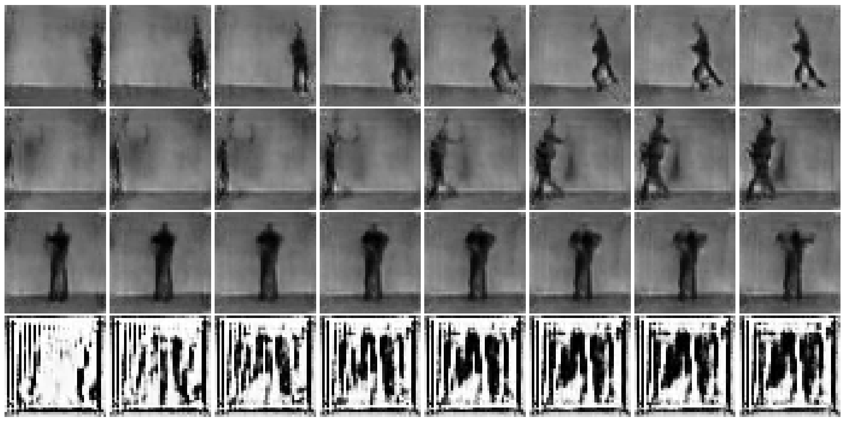


Figure 6.5: RALI samples, time goes forward from left to right, novel sampling is done at each row. The last row is to show that the decoder was unable to structure properly its latent space. This failed generation would happen 10% of the time.

6.3 Feature flow

Finally, we will conclude this thesis with a promising, yet incomplete, model. The reader will have observed that *optical flow* came again and again in the winning models reviewed in chapter 5. Also, section 6.1 should have thought us the benefits of working in feature space instead of trying to ram through pixel space and the robustness of the autoencoder framework. We propose to combine these ideas in what we call the *feature flow* (FF) model.

6.3.1 Inductive bias

Before going through the inner gears of feature flow, we would like to go over a motivation that we believe should be driving research in videos and that should have made itself apparent throughout chapter 5 and chapter 6: An inductive bias seems to be missing in models of spatio-temporal data. Recall how the switch in chapter 2 from fully connected network to convolutional network resulted in an extremely powerful prior induced over the parameters. It granted neural networks in its wake incredible leverage over any locally focused tasks. We argue that this is still lacking in the video network landscape.

Of course, research on images did not come to an end with CNN as was appreciated from the ever increasing ILSVRC classification scores which gave birth to the VGG architecture and its successor Residual Network [He et al., 2016]. Nonetheless, it seems fair to suspect that video is not yet in the phase of building on top of a core piece. ConvLSTM and Conv3D have given some results although they are clearly not the end of the story and as themselves have not proven to be deal breakers (in the sense that they are to be used in every subsequent successful architectures, like CNN are).

An argument against this claim resides in the "absence of proper video dataset" argument as was discussed in chapter 5. This is true and is a plaguing issue of the field. We conjecture that more clean data will only partially solve video.

It is with this in mind that feature flow has been designed, to embed an inductive bias into how a neural network process information from one frame to the next.

6.3.2 Model

Feature flow $\mathcal{F}$ implements a process very close to Equation 6.1 but operates in feature space. The idea is to have a neural network produce a flow field that is local to each fixel[12]. The field describes how fixels move from $t-1$ to t. The generative model will be set up as a GAN. The generator consists of two downscaling ConvLSTM encoders E^1, E^2 and two upscaling decoders D^1, D^2. The encoders will encode $\mathbf{x}_{t-1}$ at two scales, one scale to use as fixels and the other to precondition the flow field. The decoders are two upscaling ConvLSTM that will decode a random latent variable $\mathbf{z}$ to produce the flow field and from feature flow's application decode to generate $\tilde{\mathbf{x}}_t$. The model's graph is shown in Figure 6.7. More formally, given these encoder/decoder functions, feature flow operator $\mathcal{F}$, a latent low-dimensional vector[13] $\mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})$, frame $\tilde{\mathbf{x}}_t$ is predicted from frame $\mathbf{x}_{t-1}$:

$$\tilde{\mathbf{x}}_t = D^2(\mathcal{F}[E^1(\mathbf{x}_{t-1}), D^1(\mathbf{z}_t, E^2(E^1(\mathbf{x}_{t-1})))]) \tag{6.2}$$

Note that this equation was abstracted from the possible hidden states the encoders/decoders could have, it will be important in practice how to manage the

[12]A fixel is the equivalent of pixel in feature space, it is a location (i, j) on a 2D grid.

[13]The subscript t on $\mathbf{z}$ is to emphasize that the latent vector is resampled at every timestep.

recurrences.

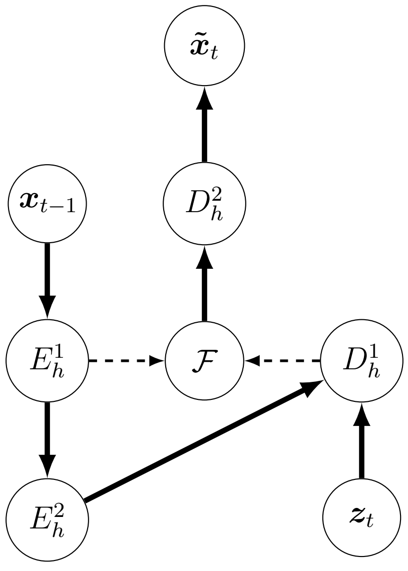


Figure 6.6: Computation graph of feature flow's generator for one frame prediction. The subscript h denotes the hidden state. It is not a requirement to precondition the flow field on the current frame but it has been observed to help. In practice, the recurrences seem to be contributing the most at E_h^1 and D_h^1.

Now we need to define the operation $\mathcal{F}$. For computational and implementation considerations, we assume that all fixels's channels are following the same flow field, meaning we apply the same transformation across the channel axis. At first glance, the equation for one fixel's transformation $E^1_{\mathbf{h}_{t-1}}[i,j]$ looks similar in essence to Equation 6.1:

$$\begin{aligned} &\mathcal{F}(E^1(\mathbf{x}_{t-1}), D^1(\mathbf{z}_t, E^2(E^1(\mathbf{x}_{t-1})))) = \tilde{\mathbf{h}}_t \\ \tilde{\mathbf{h}}_t[i,j] = &\sum_{k\in(-\xi,\xi)} \sum_{l\in(-\xi,\xi)} D^1_{\mathbf{h}_t}[i-k, j-l, (k+\xi)\kappa + l + \xi] E^1_{\mathbf{h}_{t-1}}[i-k, j-l] \end{aligned} \tag{6.3}$$

where $\kappa = 2n+1, n \in \mathbb{N}^+$, $\xi = \lfloor \frac{\kappa}{2} \rfloor$ and square brackets signifies indexing. Note that D_h^1 is indexed as a *volume*. An important contrast with $\hat{m}$ of CDNA is that they learn a distribution for a motion patch $\kappa \times \kappa$ which in their case apply on every neighbour pixels of the past frame *going in* to the predicted pixel of current frame. We do learn a distribution over a patch, but it is for each fixel of the past frame *going out*. Refer to Figure 6.7 for a visual explanation.

Something is worth mentioning with such a model. A fixel's displacement on its feature map does **not** necessarily translate into direct pixel motion. Anything can potentially be encoded in the feature maps depending at which scale it is. Thus, a group of fixels flowing in a local direction in feature space could possibly represent objects that were occluded reappearing in pixel space for example. A weak point

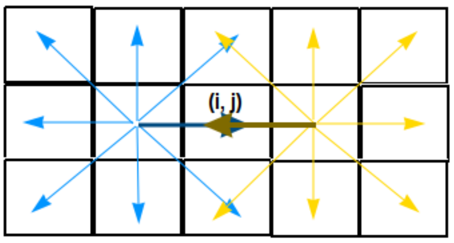


Figure 6.7: Illustration of a $\kappa = 3$ feature flow predictions at location (i, j). For clarity, only two set of patches around (i, j) are shown, there is in reality 9 patches contributing to the predicted fixel for such value of κ. Each colour represents a distribution of how the fixel at the center of each arrow can flow out. For example, this is achieved with $D^1(\mathbf{z})$ computing 9 values at each locations and sending them through a *softmax* . These 9 values are then multiplied by the fixels at frame $t-1$ to give their respective flow patches. This is how, in this two-patches-shown example, that location (i, j) receives additive contribution from dark blue arrow of patch centered at $(i, j-1)$ and dark yellow arrow of patch centered at $(i, j+1)$. In reality with 3×3 patches, there would be 9 arrows of different colours (counting the arrow pointing on itself) contributing to final fixel prediction at location (i, j).

against this model is that this approach will put a lot of pressure in the decoder's parameters to compensate the flow induced changes into an image representation that makes sense.

6.3.3 Samples

All experiments were done on KTH at full 120×120 resolution. All parts of the generator of Figure 6.7 were CNN with aggressive downsampling policy to reduce computation and memory usage. Only two ConvLSTM replaced CNN right before the application of $\mathcal{F}$ on both sides of the graph. So one ConvLSTM produced the fixel grid to be "flowed" by the flow field, output of the other ConvLSTM. This gave more consistent global motion (as perceived by a human inspecting decoded pixel space) in practice than having only CNN everywhere and no recurrence. As per standard GAN-like training, optimization was done with the Adam algorithm [Kingma and Ba, 2014] and since the publication of Salimans et al. [2016], batch normalization [Ioffe and Szegedy, 2015, Cooijmans et al., 2017] on the generator and weight normalization [Salimans and Kingma, 2016] on the discriminator.

The discriminator's architecture is a regular CNN stack with two or three ConvLSTM far from full resolution.

A key aspect of FF is at which downscaled version of the frames $\mathcal{F}$ is applied. We empirically found that 30×30 gave the best results. It is interesting to note the sub-optimality happening at a sort of skewed middle-ground between full 120×120

pixel space and final 1×1 fully abstract space[14]. Too close to pixel space and not only does it lose the computational advantages of working at lower resolution, but it suffers from the many more variability possible as it becomes a real motion prediction mechanism. On the contrary, too close to the most abstract space can result in a too tricky changes of features for the whole decoder upsampling chain to cope with or in trying to move potentially learned time-invariant features.

We trained in a *curriculum* learning fashion meaning we progressively add timesteps as the training goes on. This last trick interacts weirdly with the GAN framework and is inconclusive in its ability to help or harm training.

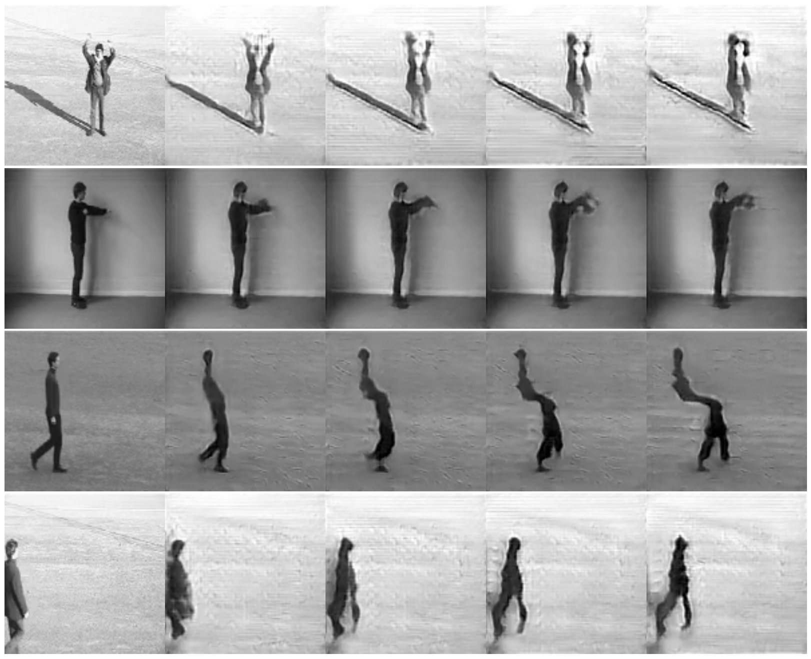


Figure 6.8: Feature flow samples, time goes forward from left to right, novel next frame is done at each row. The first frame is from the dataset and the last four are generations. Motion is correct (except for only half of it for the third row) on all samples.

As we can observe from the frame generation, the motion is correct (in the sense of possible motion in KTH) but the appearance is unimpressive. The pixel space degrade almost on the first generated frame. What is worse, the model has a hard time finding the coherant time-invariant structures and can split the body of a person in half.

It might be too small to see in the samples of Figure 6.8 but there was one appearant thing on all of them, the background was unstable. This is one suspect of the weak point discussed earlier, since all fixels are moving according to a generated

[14]Someone untanned to convolutional network training would raise an eyebrow here, there is nothing left at 1×1! CNN compensate their loss of 2D local grid by expanding more and more the channel axis. A practical example could be a starting RGB image $120 \times 120 \times 3$ going to a final scale $1 \times 1 \times 512$ before classification.

fields, there could be no concept of background or repeated pixels from frame to frame. It has to be learned by the decoders to zero out most of the motion and this put much pressure on them. That being said, we added to the generator the capacity to learn a past frame mask so it could decide to straight up copy pixels from the previous frame if it wished. It is unsatisfaying with regards of reallly learning time-invariant/variant features, it does seem like a hack and is in pixel space, but it did take care of the noisy background.

The best results were obtained by adding a few tricks and taking inspiration from chapter 5. We modified the training to be done in close/open loop mode where instead of only preconditioning on the first frame, we apply L_2 loss for a few generated frames on the generator reusing the grand truth (closed) and then letting it generate based on its past generations (open). On the discriminator's side this implies that it will see more frames and can extract more information from longer sequences.

We modified the discriminator's architecture to be like the best model of the Dynamics dataset results. The discriminator became a merge of Conv3D and ConvLSTM.

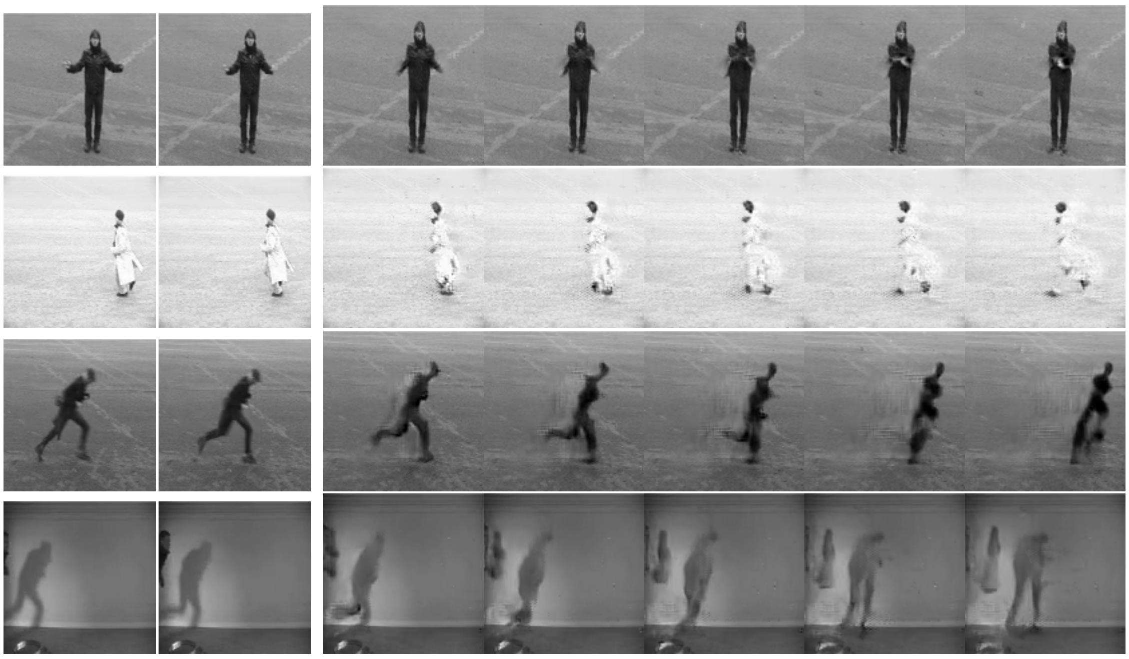


Figure 6.9: Best feature flow (teacher forcing) samples, time goes forward from left to right, novel next frame is done at each row. The first two frames are from the dataset and the last five are generations. In reality, the generator was in closed loop for 5 frames. Notice that the motion in every case is correct. Even so that on the fourth row, it thought the shadow was a person and it made it move alongside the actual profile of the person showing up. We can see from the second and third row that the crisp details of the appearance degrade quickly.

The instability issues that were encountered with RALI can be appreciated on

Figure 6.10. This figure show two common regimes that were happening during training, the *below* figure shows the overfitting of the discriminator and the *above* figure shows instability. Stability is defined in terms of training accuracy and sample quality. Samples would usually be very bad in between the plateaus of hte training accuracies. These were eventually solved with better architectures and hyperparameters with model iterations after the samples from Figure 6.8.

Overfitting is harder to explain than in a pure classification setting. We can expect that over time the features learned by the generator through the gradient of the discriminator would lose some kind of quality. If the discriminator cannot distinguish data it has not been trained on, it is not looking well that it is able to do so on the training set with such a strong margin.

6.3.4 FF for discrimination

We experimented with the idea of having the feature flow concept on a discriminative stage as well. Not only could it help the discriminator, but any discriminative task such as activity recognition could potentially benefit.

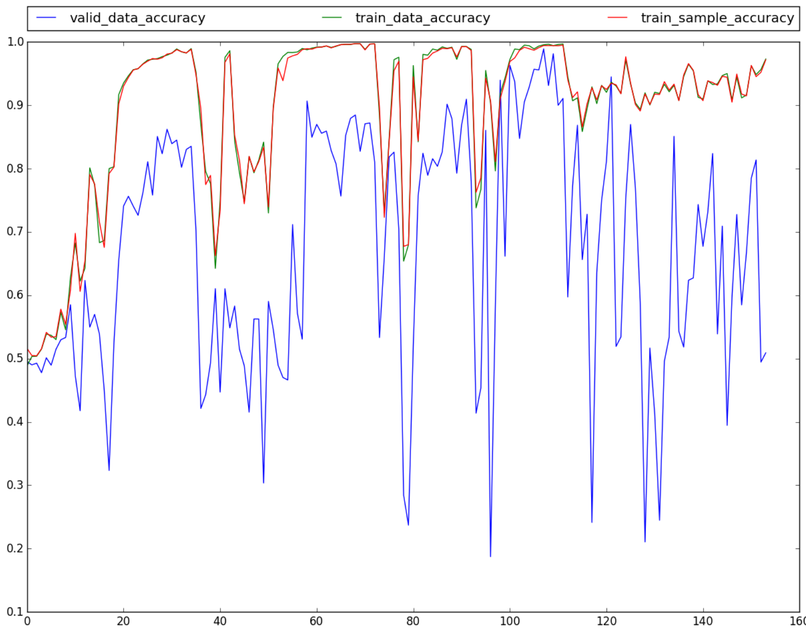
If $\mathcal{F}$ takes as arguments one frame of fixels and one flow field and maps them to the next frame's fixels, then we would like $\mathcal{F}^{-1}$ to give us the flow field between two successive frames.

$$\mathbf{f}_t = \mathcal{F}^{-1}(E(\mathbf{x}_{t-1}), E(\mathbf{x}_t)) \tag{6.4}$$

for flow field $\mathbf{f}_t$ between successive frames $\mathbf{x}_{t-1}, \mathbf{x}_t$ and some encoding function E. If the idea is anything decent, we would hope that these flow fields provide better discriminative features to the discriminator trickling down to better generation.

There are many ways to implement the inverse of feature flow. We investigated two methods, viewing it as cross-correlation or learning it ala Relation Network [Santoro et al., 2017].

Cross-correlation is a signal processing method to estimate the similarity between two signals relative to a translation. We have already seen cross-correlation at Equation 2.6 when discovering CNN, it is indeed the operation they are built on top of. Remember that for a patch of flow field $\kappa \times \kappa$ there are κ^2 values around a single location that needs to be found. Our full flow field $\mathbf{f}$ will again be a 3D volume with length κ^2 for the channel axis. For a choice of encoding function E, κ, location (i, j) and offset $(k, l), k, l \in [-\xi, \xi]$, a single value of the flow field is computed as follow:



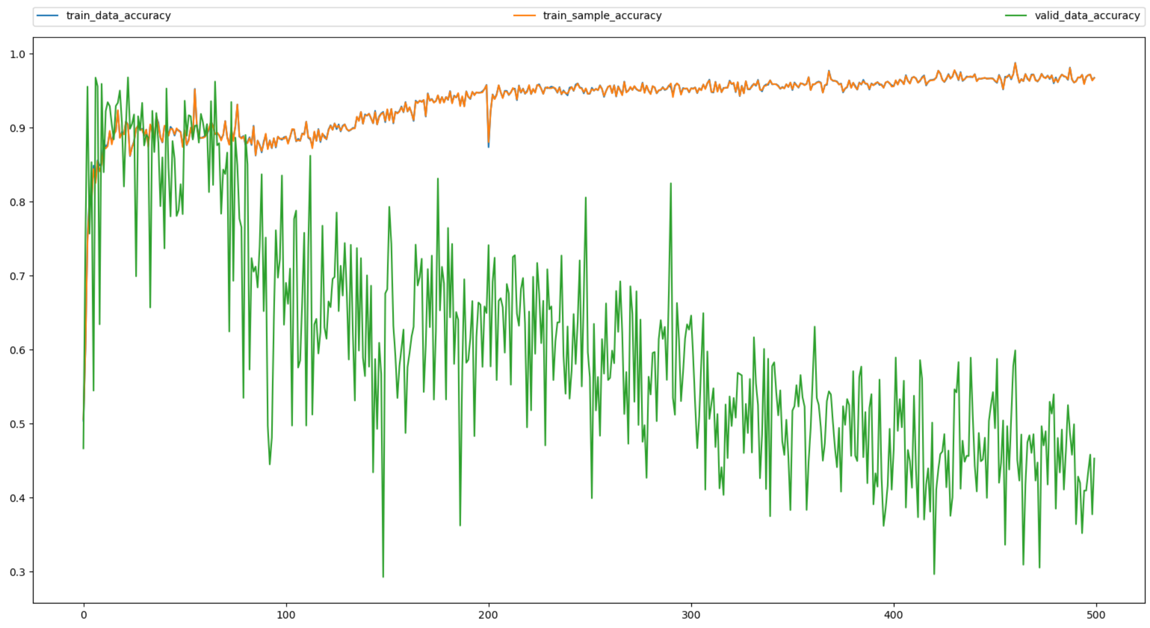


Figure 6.10: Feature flow's training learning curves with accuracies with respect to epochs. *Train/valid data accuracy* are the accuracies of the discriminator on the data from the training/validation set. *Train sample accuracy* is the accuracy of the discriminator on the samples from the generator (there is no concept of training/validation set for generated samples. The unstable regime is shown *above* and the overfitting one *below*.

$$\mathbf{f}_t[i,j,(k+\xi)\kappa+l+\xi] = \sum_{m\in(-\xi,\xi)}\sum_{n\in(-\xi,\xi)} E(\mathbf{x}_{t-1})[i+k+m,j+l+n]E(\mathbf{x}_t)[i+k+m,j+l+n] \tag{6.5}$$

The whole patch of flow field at location (i,j) is computed by rolling over all the possible values of (k,l) and computing a cross-correlation every time. Usually, the downsampled frame will be a volume with a channel axis. To mirror feature flow on the generator's side, we again assume that it is independent across this axis and all flow fields are the same.

The relation network method is to have a neural net actually compute the relation between the fixels of successive frames. With the same formality as Equation 6.5 and a neural network $\mathrm{rn}_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$, a single value of the flow field will then be computed such that

$$\mathbf{f}_t[i,j,(k+\xi)\kappa+l+\xi] = \mathrm{rn}_{\boldsymbol{\theta}}(E(\mathbf{x}_{t-1})[i+k,j+l],E(\mathbf{x}_t)[i+k,j+l])$$

where again rolling over all (k,l) will complete $\mathbf{f}$ at location (i,j). $\mathrm{rn}_{\boldsymbol{\theta}}$ will typically be a small MLP. This approach is difficult to implement in practice without parallelizing the many calls to the MLP. Even then, the generation results were not on par with the simpler cross-correlation and with its lack of success on the Dynamics dataset at Table 5.2, we decided not to push on this idea any further.

We will not show results for this method since no generation were convincingly better. In opposition, there were not strikingly worst and consequently, we believe that in the spirit of having an embedded inductive bias in the network it still requires further exploration.

6.4 Future work

The results of subsection 6.3.3 and subsection 6.3.4 were a good start but have yet to make it to the finish line. What is remarkable is that the motion in image space in most cases is the actual true motion displayed by KTH. What is terribly missing is the finishing touch, the crisp and image-like look that DrNet was able to generate. This seems to suggest that we need a way to factorize even more time-invariant and time evolving features with FF.

Feature flow has still a few directions where it could be enhanced. The channel independent application for example might be untrue. Or sampling at every timestep $\mathbf{z}$ could have given the model too much entropy to deal with. Also, the results tend to point to the fact that the pixel space decoder might have too much pressure on its parameters to reorganized the flows. Either the networks handling the flows are making too much error (we do not have blocks of motions like in CDNA, that is also a possible avenue to improvement) and the decoders need to recover from them, or this is simply asking too much for the decoders.

It is unlikely that these results have been negative enough to undermine the arguments of subsection 6.3.1 and abort the quest for an inductive bias. On the contrary, they are annoyingly close to offer a viable route to generation. It is to note that the potential of FF is many since it is not framework specific, a VAE, or even if ever seeing the light RALI, could use it.

Finally, there is the unmentioned but very present sub-problem in this task, the GAN-RNN interaction. It is hard in reality to have a clear cut marker of what part of the model is the culprit since this issue can be behind every generation. This is a promising and need to be taken road to glory, not only for this feature flow setup, but any GAN-RNN, and by extension RALI, based models.

We did try to use new GAN technologies briefly overviewed in section 3.3 but it did not came with a noticable boost in performances or stability. We suspect the GAN-RNN version may obey different dynamics and could be another reason behind the behaviors observed in Figure 6.10.

Chapter 7

Conclusion

Throughout this thesis, we have mainly treaded on the path of modelling videos with one of the most alive branches of artificial intelligence. We started by assembling the required deep learning knowledge necessary and then made our first stop into generative models. Generative models have been the focus, be it theoretical or application, of this thesis.

On theoretical grounds, we built on generative adversarial networks and variational autoencoders by proposing the adversarially learned inference model. Doing so, we offered a mix of the strongest traits of both frameworks, a latent space and the adversarial training. With adversarial training came the realistic results and with the abstract representation a powerful mechanism to manipulate them. As ALI has shown to be able to do anything that the pieces it is built upon are capable of, we believe future work will further reveal its potential.

On application grounds, we have applied generative models to videos. It is actually unfair to categorize all this work by "application". In truth, as we have argued there are concepts, which could be considered more under a theoretical light, lacking in the understanding of video modelling.

We first took a sidestep on the discriminative side of things. Taking an activity recognition lens, we have looked at a double faced issue coming from the datasets and the models that are being tried on them. For this reason, we designed a new dataset in hopes of narrowing down problems that will give rise to solutions with reach to the whole video field.

But data is not the end of the story and we continued our model investigation through generation. More precisely in the context of next frame generation, this brought us the feature flow model. We introduced the idea of encoding changes from frame to frame into changes in feature space. Generation is done by moving around in this learned abstract space and then decoding back to real human appreciable sequence of images. This model has shown promises and is yet to be fully complete.

Even then though, the video generation landscape, and fairly to say video at large, is still in its infancy and many challenges are looming ahead. There is one point onto which all models have been criticized which is camera motion. It is in our opinion one very interesting and hard phenomena to model and almost nothing presented in this chapter, or in the deep learning computer vision community as a matter of fact, can so far hope to alone solve this problem. Be it live camera feed coming in from a walking robot or random real life videos being uploaded on servers, any decent deep learning model wishing to understand this medium will have to offer a robust solution for this.

Bibliography

Martin Arjovsky and Lon Bottou. Towards principled methods for training generative adversarial networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Martin Arjovsky, Soumith Chintala, and Lon Bottou. Wasserstein GAN. *CoRR*, 2017.

Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *CoRR*, 2012.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, 2013.

Yoshua Bengio, Eric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016.

Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Emily L Denton and vighnesh Birodkar. Unsupervised learning of disentangled representations from video. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4417–4426. Curran Associates, Inc., 2017.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 658–666. Curran Associates, Inc., 2016.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *CoRR*, 2016.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein GANs. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017a.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. PixelVAE: A latent variable model for natural images. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Diederik P Kingma. Fast gradient-based inference with continuous latent variable models in auxiliary form. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 1st International Conference on Learning Representations (ICLR)*, 2013.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27*, pages 3581–3589. Curran Associates, Inc., 2014.

Diederik P Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0-262-01319-3 978-0-262-01319-2.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

Alex Lamb, Vincent Dumoulin, and Aaron Courville. Discriminative regularization for generative models. *CoRR*, 2016.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Jianhua Lin. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. In *ICLR Workshop*, 2016.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, page 4. Curran Associates, Inc., 2011.

Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. http://distill.pub/2016/deconv-checkerboard/, 2016.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

Antti Rasmus, Harri Valpola, Mikko Honkala, Mathias Berglund, and Tapani Raiko. Semi-supervised learning with ladder network. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *CoRR*, 2017.

Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local SVM approach. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ICPR '04, pages 32–36. IEEE Computer Society, 2004. ISBN 0-7695-2128-2. doi: 10.1109/ICPR.2004.747.

Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, and Zehan Wang. Is the deconvolution layer the same as a convolutional layer? *CoRR*, 2016.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In *ICCV Workshop*, 2013.

Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, volume abs/1312.6199, 2014.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, 2016.

Lucas Theis, Aron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

Benigno Uria, Marc-Alexandre Ct, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *CoRR*, 2016.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *CoRR*, 2015.

Gl Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016.

Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *The IEEE International Conference on Computer Vision (ICCV)*, 2013a.

Heng Wang and Cordelia Schmid. Lear-inria submission for the thumos workshop. In *ICCV workshop on action recognition with a large number of classes*, volume 2, page 8, 2013b.

Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4305–4314, 2015.

Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *Proceedings of the European Conference on Computer Vision*, 2016.

Zbigniew Wojna, Vittorio Ferrari, Sergio Guadarrama, Nathan Silberman, Liang-Chieh Chen, Alireza Fathi, and Jasper Uijlings. The devil is in the decoder. *CoRR*, 2017.

Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in Neural Information Processing Systems 29*, pages 91–99. Curran Associates, Inc., 2016.

Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2528–2535, 2010.

Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. In *ICLR Workshop*, 2016.

Wangjiang Zhu, Jie Hu, Gang Sun, Xudong Cao, and Yu Qiao. A key volume mining deep framework for action recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.