

Université de Montréal

**Sequence to Sequence Learning and Its Speech Applications**

**par Ying Zhang**

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

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

April, 2018

© Ying Zhang, 2018.



# Résumé

Les réseaux neuronaux récurrents (RNN) ont été dominants dans le domaine de la parole au cours des dernières décennies, étant donné leurs propriétés attrayantes de modélisation de séquences. Les réseaux neuronaux convolutionnels (CNN) ont été présentés comme une alternative pour la modélisation de séquences en raison de leur capacité à réduire les variations spectrales et à modéliser les corrélations spectrales dans les caractéristiques acoustiques pour la reconnaissance automatique de la parole (ASR). Des travaux récents suggèrent que les nombres complexes pourraient être utilisés comme une représentation de caractéristique plus riche que le spectre et qui pourraient donc être bénéfique pour les tâches liées à la parole.

Dans la thèse, nous abordons d'abord les concepts de base de l'apprentissage automatique, les blocs de construction de l'apprentissage profond et discutons des méthodes populaires capables de faire des modélisations séquentielles, en particulier des réseaux de neurones convolutionnels, célèbres en tant que réseaux feed-forward. Nous présentons ensuite deux travaux de recherche liés à la modélisation séquence-séquence sur la parole. Premièrement, nous introduisons une nouvelle approche pour adresser la reconnaissance de la parole avec des réseaux de neurones convolutionnels qui montre des performances comparables avec leur homologue des réseaux neuronaux récurrents. Deuxièmement, nous présentons un nouveau modèle, tirant parti de la représentation dans le domaine complexe, et définissons des circonvolutions complexes, des stratégies complexes de normalisation par lots et d'initialisation de poids complexes. Le modèle a atteint l'état de l'art de la tâche de prédiction du spectre de la parole dans un cadre récurrent convolutionnel.

**Mots clés:** réseaux de neurones, apprentissage automatique, apprentissage profond, réseaux de neurones convolutionnels, modélisation de séquences, reconnaissance de la parole, représentation complexe



# Summary

Recurrent Neural Networks (RNNs), which has the attractive properties of modelling sequences, has been dominant in speech field in the recent decades. Convolutional Neural Networks (CNNs) has been shown as an alternative to model sequences because of its capacity of reducing spectral variations and modeling spectral correlations in acoustic features for automatic speech recognition (ASR). Recent work suggests that complex numbers could be used as a richer feature representation than spectrum which may benefit the speech related tasks.

In the thesis, we first cover the basic concepts in machine learning, building blocks of deep learning and discuss the popular methods that are capable of doing sequence-to-sequence modelling, specially convolutional neural networks, which is famous as a class of feed-forward nets. We then present two research work related to sequence-to-sequence modelling on speech. We introduce a new approach to address speech recognition with convolutional neural networks which shows the comparable results with their recurrent neural networks counterpart. In addition, we present a new model taking advantage of the representation in the complex domain and define complex convolutions, complex batch-normalization, complex weight initialization strategies. The new model results in state-of-the-art of speech spectrum prediction in a convolutional recurrent setting.

**Keywords:** neural networks, machine learning, deep learning, convolutional neural networks, sequence modelling, speech recognition, complex representation



# Contents

<b>Résumé</b> . . . . .	<b>ii</b>
<b>Summary</b> . . . . .	<b>iii</b>
<b>Contents</b> . . . . .	<b>iv</b>
<b>List of Figures</b> . . . . .	<b>vi</b>
<b>List of Tables</b> . . . . .	<b>vii</b>
<b>List of Abbreviations</b> . . . . .	<b>ix</b>
<b>Acknowledgments</b> . . . . .	<b>x</b>
<b>1 Introduction to Deep Learning</b> . . . . .	<b>1</b>
1.1 Machine Learning Basics . . . . .	1
1.1.1 Definition . . . . .	1
1.1.2 Tasks Categorization . . . . .	1
1.1.3 Empirical Risk . . . . .	2
1.1.4 Generalization . . . . .	3
1.2 Artificial Neural Networks . . . . .	4
1.2.1 Perceptron . . . . .	4
1.2.2 Multi-Layer Perceptrons . . . . .	5
1.2.3 Activation Function . . . . .	5
1.2.4 Cost Functions . . . . .	7
1.2.5 Optimization . . . . .	8
<b>2 Sequence to Sequence Learning</b> . . . . .	<b>12</b>
2.1 Concept . . . . .	12
2.2 Recurrent Neural Networks . . . . .	12
2.3 Long Short Term Memory . . . . .	14
2.4 Bidirectional Structure . . . . .	15
2.5 Convolutional Neural Networks . . . . .	15
2.5.1 Convolution . . . . .	16
2.5.2 Pooling . . . . .	17


---

2.5.3	Fully Connected Layer . . . . .	18
2.6	Sequence Loss Function . . . . .	18
2.6.1	Connectionist Temporal Classification . . . . .	18
<b>3</b>	<b>Prologue to First Article . . . . .</b>	<b>21</b>
<b>4</b>	<b>Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks . . . . .</b>	<b>22</b>
4.1	Introduction . . . . .	22
4.2	Experiments . . . . .	25
4.2.1	Data . . . . .	25
4.2.2	Models . . . . .	25
4.2.3	Training and Evaluation . . . . .	25
4.2.4	Results . . . . .	26
4.3	Discussion . . . . .	26
<b>5</b>	<b>Prologue to Second Article . . . . .</b>	<b>28</b>
<b>6</b>	<b>Deep Complex Networks . . . . .</b>	<b>29</b>
6.1	Introduction . . . . .	29
6.2	Complex Building Blocks . . . . .	31
6.2.1	Representation of Complex Numbers . . . . .	31
6.2.2	Complex Convolution . . . . .	31
6.2.3	Complex Differentiability . . . . .	32
6.2.4	Complex-Valued Activations . . . . .	33
6.2.5	Complex Batch Normalization . . . . .	34
6.2.6	Complex Weight Initialization . . . . .	36
6.2.7	Complex Convolutional Residual Network . . . . .	38
6.3	Experiments . . . . .	40
6.3.1	Image Recognition . . . . .	40
6.3.2	Automatic Music Transcription . . . . .	42
6.3.3	Speech Spectrum Prediction . . . . .	44
<b>7</b>	<b>Conclusion . . . . .</b>	<b>46</b>
	<b>Bibliography . . . . .</b>	<b>47</b>



# List of Figures

1.1	The curve between training / generalization error and model capacity. X-axis denotes model capacity and Y-axis denotes the error. . .	3
1.2	A graphical illustration of perceptron algorithm . . . . .	4
1.3	A graphical illustration of multi-layer perceptron . . . . .	5
1.4	A graphical depiction of the difference between BGD and SGD. SGD offers noiser gradient in each iteration. . . . .	9
1.5	The detailed Adam algorithm showed in Kingma and Ba (2014) . .	10
2.1	A recurrent neural network unrolled in time (Figure adapted from Yoshua Bengio’s slides) . . . . .	13
2.2	The difference between a bidirectional RNN and an unidirectional RNN. (Figure adapted from wikipedia) . . . . .	15
2.3	<i>The convolution layer and max-pooling layer applied upon input features. . . . .</i>	16
2.4	<i>Dilated convolution on 2D data. Figure is adapted from Yu and Koltun (2015) . . . . .</i>	17
4.1	<i>Network structure for phoneme recognition on the TIMIT dataset. The model consists of 10 convolutional layers followed by 3 fully-connected layers on the top. All convolutional layers have the filter size of <math>3 \times 5</math> and we use max-pooling with size of <math>3 \times 1</math> only after the first convolutional layer. First and second numbers correspond to frequency and time axes respectively. . . . .</i>	24
6.1	Complex convolution and residual network implementation details. .	32
6.2	<i>Learning curve for speech spectrum prediction from dev set. . . . .</i>	45



# List of Tables

4.1	Phoneme Error Rate (PER) on TIMIT. 'NP' is the number of parameters. 'BiLSTM-3L-250H' denotes the model has 3 bidirectional LSTM layers with 250 units in each direction. In the CNN model, (3, 5) is the filter size. Results suggest that deeper architecture and larger filter sizes leads to better performance. The best performing model on Development set, has a test PER of 18.2 % . . . . .	27
6.1	Classification error on CIFAR-10, CIFAR-100 and SVHN* using different complex activations functions ( $z$ ReLU, modReLU and CReLU). WS, DN and IB stand for the wide and shallow, deep and narrow and in-between models respectively. The prefixes R & C refer to the real and complex valued networks respectively. Performance differences between the real network and the complex network using CReLU are reported between their respective best models. All models are constructed to have roughly 1.7M parameters except the modReLU models which have roughly 2.5M parameters. modReLU and $z$ ReLU were largely outperformed by CReLU in the reported experiments. Due to limited resources, we haven't performed all possible experiments as the conducted ones are already conclusive. A "-" is filled in front of an unperformed experiment. . . . .	38
6.2	Classification error on CIFAR-10, CIFAR-100 and SVHN* using different normalization strategies. NCBN, CBN and BN stand for a Naive variant of the complex batch-normalization, complex batch-normalization and regular batch normalization respectively. (R) & (C) refer to the use of the real- and complex-valued convolution respectively. The complex models use CReLU as activation. All models are constructed to have roughly 1.7M parameters. 5 out of 6 experiments using the naive variant of the complex batch normalization failed with the apparition of NaNs during training. As these experiments are already conclusive and due to limited resources, we haven't conducted other experiments for the NCBN model. A "-" is filled in front of an unperformed experiment. . . . .	39

---

6.3	MusicNet experiments. $FS$ is the sampling rate. $Params$ is the total number of parameters. We report the average precision (AP) metric that is the area under the precision-recall curve. . . . .	43
6.4	Speech Spectrum Prediction on TIMIT test set. $CConv-LSTM$ denotes the Complex Convolutional LSTM. . . . .	44





# List of Abbreviations

ANN	Artificial Neural Network
BGD	Batch Gradient Descent
CNN	Convolutional Neural Networks
CTC	Connectionist Temporal Classification
GD	Gradient Descent
I.I.D	Independent and Identically Distributed
LSTM	Long Short Term Memory
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NLL	Negative Log-Likelihood
MLP	Multilayer perceptron
PReLU	Parametric Rectifier Linear Unit
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent



# Acknowledgments

I would like to express my thanks to my colleagues in MILA for their support and help on my thesis. Specially, I want to thank my supervisor Aaron Courville for his mentorship, guidance and encouragement throughout my research.

I would like to extend my thanks to my co-authors Mohammad Pezeshiki, Philémon Brakel, Saizheng Zhang, César Laurent, Dmitry Serdyuk, Chiheb Trabelsi, thank you all for your support, hard-work and insights!

Last but not least, the work presented in the thesis could not have been done without the great help from Theano team.

# 1

# Introduction to Deep Learning

In this chapter we cover the basic concepts of machine learning and several fundamental components in deep learning family.

---

## 1.1 Machine Learning Basics

### 1.1.1 Definition

Machine Learning algorithms are described as a set of algorithms equipped with the ability to learn from data. Different from the traditional algorithms with hand-crafted features, such algorithms could learn to make data-driven predictions or decisions without being explicitly designed. Machine learning has been widely used in computer vision, speech recognition and natural language processing (NLP). Related applications include object detection (Ren et al., 2015; He et al., 2017, 2016a; Redmon et al., 2016), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014; Sak et al., 2014), language modelling (Mikolov et al., 2010; Sundermeyer et al., 2012), etc.

### 1.1.2 Tasks Categorization

Machine learning tasks mainly fall into supervised learning, semi-supervised learning, unsupervised learning, active learning and reinforcement learning, which depend on the kind of data provided to the learning algorithm.

**Supervised Learning:** Provided with a set of inputs  $\mathbf{x} \in X$  and targets  $\mathbf{y} \in Y$ , the algorithm is to learn some parameterized functions  $F : X \rightarrow Y$  that map inputs to outputs:

$$\mathbf{y} = F(\mathbf{x}; \theta), \tag{1.1}$$

where  $\theta$  is a set of learnable parameters.

---

**Semi-Supervised Learning:** Given a set of inputs  $\mathbf{x} \in X$  with partial targets  $\mathbf{y} \in Y$ . The task is to learn some parameterized functions  $F : X \rightarrow Y$  that map inputs to outputs.

**Active Learning:** a special case of semi-supervised learning. The algorithm is capable of interactively querying the users (or some other information source) to label the outputs with new inputs.

**Reinforcement Learning:** The algorithm is to learn the optimal policy for the agent in a dynamic environment by taking actions and optimizing its future reward.

**Unsupervised Learning:** Provided a set of inputs  $\mathbf{x} \in X$  with no targets, the algorithm is to learn the underlying data structure from the unlabeled inputs:

$$\mathbf{h} = F(\mathbf{x}; \theta), \tag{1.2}$$

where  $F$  is parameterized function,  $\theta$  is a set of learnable parameters and  $\mathbf{h}$  is the underlying representation.

### 1.1.3 Empirical Risk

To learn the supervised tasks, cost function needs be provided to measure how close the predictions are to their corresponding targets and optimize the algorithm based on its cost. More formally, we assume a cost function in the form of  $L(\hat{\mathbf{y}}, \mathbf{y})$ , where  $\hat{\mathbf{y}}$  is the prediction and  $\mathbf{y}$  is the target. The **risk** is then defined as:

$$R(F) = \mathbf{E}[L(F(\mathbf{x}, \theta), \mathbf{y})] = \int L(F(\mathbf{x}, \theta), \mathbf{y}) dP(\mathbf{x}, \mathbf{y}) \tag{1.3}$$

where  $P(\mathbf{x}, \mathbf{y})$  is joint probability distribution over space  $X$  and space  $Y$  and  $\mathbf{E}$  is the expectation.

The goal could be formed as finding the optimal function  $F(\mathbf{x}, \theta)$  among a family of parameterized functions for which **risk** is minimal. In most cases, we are unable to access the complete data space and the joint probability distribution  $P(\mathbf{x}, \mathbf{y})$  is unknown to us. Alternatively, we use **empirical risk** to approximate the true **risk**:

$$\tilde{R}(F) = \frac{1}{n} \sum_{i=1}^n L(F(\mathbf{x}_i, \theta), \mathbf{y}_i) \tag{1.4}$$

---

where  $n$  is the total number of training data we can access.

The learning goal is then transformed into finding the optimal function  $F(\mathbf{x}, \theta)$  for which **empirical risk** is minimal, which is also known as **empirical risk minimization**.

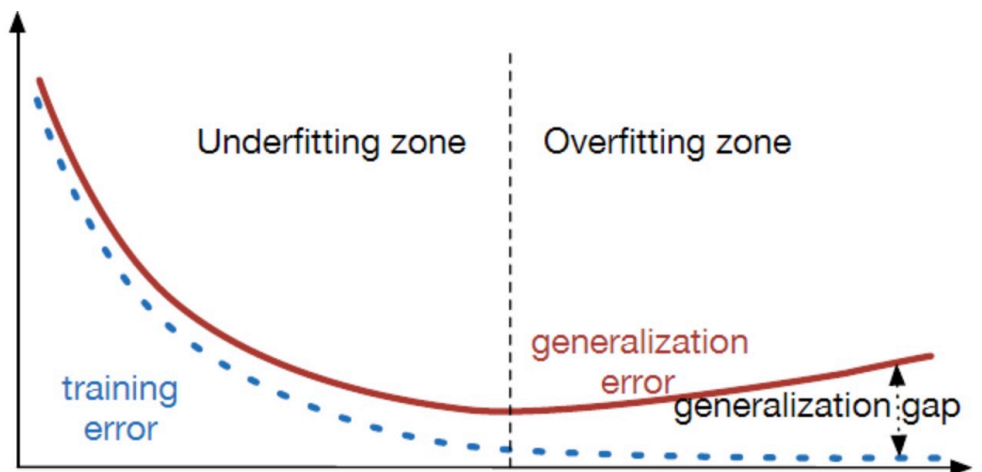
### 1.1.4 Generalization

Generalization denotes how well a algorithm with learned parameters perform on unseen and independent and identically distributed (i.i.d) data compared to the training set, which is a crucial ability of the algorithm learning. The generalization error is defined as the expected loss on any new input.

To evaluate how good a model is, two factors are considered:

1. if the training error is small;
2. if the gap between training error and generalization error is small.

These two factors result in two important concepts in machine learning: **underfitting** and **overfitting**, which are highly related to the model capacity. As shown in Figure 1.1, underfitting occurs when the model does not have enough capacity to obtain a small training error while overfitting occurs when the model has enough capacity but the gap between training error and generalization error is large.



**Figure 1.1** – The curve between training / generalization error and model capacity. X-axis denotes model capacity and Y-axis denotes the error.

---

## 1.2 Artificial Neural Networks

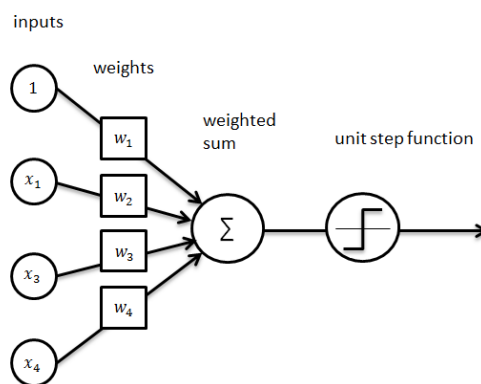
### 1.2.1 Perceptron

Inspired by the biological neural networks that build animal brains, Artificial Neural Networks (ANNs) are based on a collection of connected units called artificial neurons, where each connection between neurons can transmit a signal to another neuron. One famous algorithm among ANNs dating back to early 60's is Perceptron (Rosenblatt, 1958), which was created by Rosenblatt.

Perceptron is designed to tackle binary classification problem in supervise learning. The algorithm is to learn to decide if an input belongs to a negative class or a positive class based on the dot product over a set of parameters and the input values. Formally, consider a set of inputs  $\mathbf{x} = \{x_1, x_2, \dots, x_d\}$  containing  $d$  input scalars, a set of  $d$  scalar weights  $\mathbf{w} = \{w_1, w_2, \dots, w_d\}$  and a single scalar bias term  $b$ , the perceptron algorithm is defined as follows:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1.5)$$

The bias term shifts the boundary from the origin and is independent of inputs. Figure 1.2 shows a schematic diagram of the algorithm.



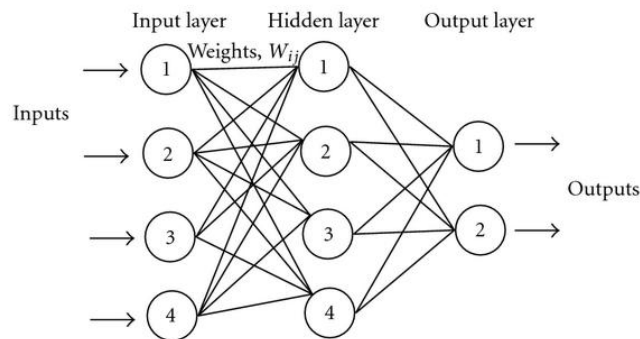
**Figure 1.2** – A graphical illustration of perceptron algorithm

The perceptron algorithm is also termed as single-layer perceptron, which is distinguished from multi-layer perceptron.

---

## 1.2.2 Multi-Layer Perceptrons

Multi-layer perceptron (MLP) is the extension of a single-layer perceptron, which consists of at least three layers (including an input layer and an output layer). Different from the unit step function used in perceptron algorithm, a non-linear activation function (which will be discussed in details in the following section) is applied to the dot product value.



**Figure 1.3** – A graphical illustration of multi-layer perceptron

As shown in Figure 1.3, MLPs are fully connected. Each node in one layer is connected with a set of weights  $\mathbf{w}_{ij}$  to every node in the following layer. Cybenko's theorem (Cybenko, 1989) shows that MLPs are universal function approximators, which proves that a 3-layer perceptron can approximate continuous functions on compact subsets of  $\mathbf{R}^n$  under certain assumptions. The property suggests that such artificial neural networks could model various functions with appropriate weights and activation function. MLPs has been widely used in image classification (LeCun et al., 1998; Cireřan et al., 2012) and speech recognition (Lopes and Perdigao, 2011; Boulard and Morgan, 1990) since 1980's.

## 1.2.3 Activation Function

The activation function is a crucial component in neural networks. It takes as input the dot product value with a bias term and output a value decides whether the node should be "fired" or not. Here, we introduce several popular activation functions. In all the following functions,  $x$ ,  $w$ ,  $b$  represent an input random variable, a scalar weight and a scalar bias term, respectively.

---

• **Unit step function** is a discontinuous function that is used in perception algorithm. It is defined as:

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1.6)$$

• **Sigmoid function** is inspired from probability theory and its output ranges from 0 to 1 in an "S" shape, which is commonly used in multi-layer perceptron. It is defined as:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1.7)$$

• **Hyperbolic tangent function (Tanh)** is a rescaling of the sigmoid function in which output ranges from  $-1$  to  $1$ :

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (1.8)$$

• **Rectifier Linear Unit (ReLU)** is inspired from biological motivations. The advantages of ReLU include efficient computation, sparse activation, etc. It is favored for deep neural networks:

$$f(x) = \max(0, x) \quad (1.9)$$

• **Parametric Rectifier Linear Unit (PReLU)** is an extension of the ReLU in which the output of the function in the regions that input is a linear function of the input with a slope of  $\alpha$ :

$$f(x) = \begin{cases} f(x), & \text{if } f(x) > 0 \\ \alpha f(x), & \text{otherwise} \end{cases} \quad (1.10)$$

the extra parameter  $\alpha$  is usually initialized to 0.1 and can be trained.

• **Maxout** takes the maximum output from  $n$  piece-wise linear functions:

$$f(x) = \max(f'(x), f''(x)) \quad (1.11)$$



---

where for  $f'(x)$  and  $f''(x)$  we have:

$$f'(x) = w' \times x + b', \dots \quad f^n(x) = w^n \times x + b'', \quad (1.12)$$

• **Softmax function** is also named normalized exponential function. It normalizes  $K$ -dimensional vector  $\mathbf{x}$  to a vector ranging from 0 to 1 that adds up to 1. It commonly used for represent a probability distribution for  $K$  possible outputs. Thus, softmax function has been highly used in classification tasks. It is defined as:

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^K e^{x_k}} \quad (1.13)$$

## 1.2.4 Cost Functions

Following what we discussed about empirical risk in 1.1.3, it is necessary to define the concrete formulation of the cost function (also known as the objective function)  $L(\hat{\mathbf{y}}, \mathbf{y})$ , where  $\hat{\mathbf{y}}$  is the prediction and  $\mathbf{y}$  is the target. Typically, the output of the cost function is a measurement of how far the predictions are from the targets. Various cost functions could be chosen for specific tasks. Here, we introduce 3 popular cost functions:

**Cross-Entropy:** Cross-Entropy defines a loss between the categorical output and the target in a supervised classification model. The categorical output is usually interpreted as a probability value for its category from the softmax output:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_c 1_{\mathbf{y}=c} \log \hat{\mathbf{y}}_c \quad (1.14)$$

where  $c$  denotes the target. The loss increases when the prediction diverges from the target and goes to zero when the prediction assigns 100% probability to the target class.

**Mean Squared Error (MSE):** Mean squared error has been widely use in regression tasks. Different from classification, regression models output continuous values:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \quad (1.15)$$

where  $n$  denotes  $n$  training examples we could access.

**L1 loss:** L1 loss measures the absolute distance between the prediction and the

---

target:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n |\hat{\mathbf{y}}_i - \mathbf{y}_i| \quad (1.16)$$

L1 loss function is robust and less vulnerable by outliers while MSE is very sensitive to outliers.

## 1.2.5 Optimization

In the deep learning context, optimization is defined as finding the optimal weights of a model that could minimize a given cost function. Optimization of ANNs is crucial and difficult. A set of approaches have been developed to solve the task and all popular optimizers methods are gradient-based.

One popular technique in gradient-based optimization family is gradient descent, which finds a local minimum of the given cost function based on the derivatives in a iterative way.

Suppose we have a model to learn a mapping function from  $\mathbf{x} \in X$  to  $\mathbf{y} \in Y$ ,

$$\mathbf{y} = F(\mathbf{x}; \theta), \quad (1.17)$$

where  $\theta$  is a set of learnable parameters. The derivative of the function in terms of parameters is defined as  $F'_\theta(\mathbf{x}; \theta)$ , which in 1-D, is the slope of  $F(\mathbf{x}; \theta)$  with  $\theta$  at the point  $\mathbf{x}$ . It tells us how to scale  $\theta$  to make an improvement over  $\mathbf{y}$ . More specifically, we could reduce the value of loss function by changing the parameters as follows:

$$\theta_{t+1} = \theta_t - \epsilon F'_\theta(\mathbf{x}; \theta_t) \quad (1.18)$$

where  $t$  indicates the  $t$ -th iteration and  $\epsilon$  denotes a positive scalar determining the size of the step. The iteration would stop when  $\mathbf{y}$  is lower than all the neighboring points.

To train the ANN parameters, we need to use gradient descent for backward propagation of errors (backpropagation) through different layers. The method computes the gradient of the cost function with respect to the parameters. In order to complete backpropagation, we need to use the chain rule. Suppose we have two

simple functions,

$$f(q, z) = q \times z \tag{1.19}$$

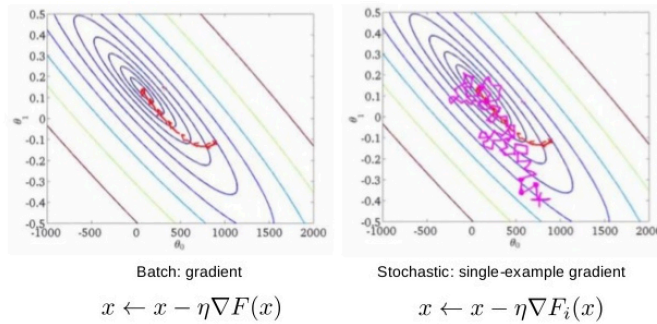
$$q(x, y) = x + y \tag{1.20}$$

we have known how to compute the gradient of  $f$  with respect to  $q$  and the gradient of  $q$  with respect to  $x$  from the aforementioned method. To compute the gradient of  $f$  with respect to  $x$ , we perform,

$$f'_x(q, z) = f'_q(q, z) \times q'_x(x, y) \tag{1.21}$$

In this case, the gradient is simply obtained by multiplication of two gradient in each function.

Typically, a training dataset would be provided when we train ANNs. There are two trends to apply gradient descent: batch gradient descent (BGD) and stochastic gradient descent (SGD). Batch gradient descent calculates the gradient using the whole training dataset. This method could be expensive and inefficient when the dataset is in the large-scale. Stochastic gradient descent computes the gradient using a single example in the dataset, many applications of SGD actually use a mini-batch including several examples. The gradient computed in this case is noisier but it turns out working well. A graphical depiction of the difference between BGD and SGD is shown in Figure 1.4.



**Figure 1.4** – A graphical depiction of the difference between BGD and SGD. SGD offers noisier gradient in each iteration.

One could imagine that standard SGD may oscillate across the ravine instead of going down to the optimum if the loss surface has the form of a long shallow ravine leading to the optimum and steep walls on the sides. In this case, standard SGD

may lead to slower convergence. Momentum (Sutskever et al., 2013) was proposed to alleviate this problem. The momentum update follows:

$$v_{t+1} = \alpha v_t + \epsilon F'_\theta(\mathbf{x}; \theta_t) \quad (1.22)$$

$$\theta_{t+1} = \theta_t - v_{t+1} \quad (1.23)$$

where  $v_{t+1}$  denotes the velocity at  $t + 1$ -th iteration,  $\alpha$  determines for how many iterations from the previous gradients are involved into the current update.

SGD or SGD with momentum could be categorized as the first-order iterative optimization algorithm. Another popular first-order optimization algorithm is Adaptive Moment Estimation (Adam) proposed by Kingma and Ba in 2014 (Kingma and Ba, 2014) which computes the adaptive learning rates for each parameter. It takes advantage of per-parameter learning rates that helps to improve the performance of the problems with sparse gradients and that are adapted based on the average of recent magnitudes of the gradients for the parameters. In addition, Adam also benefit from the average of the second moments of the gradients. Basically, the algorithm computes an exponential moving average of the gradient and the squared gradient. There are two scalar hyper-parameters that control the decay rates of these moving averages. The detailed algorithm is described in Figure 1.5.

---

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $g_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

---

**Require:**  $\alpha$ : Stepsize  
**Require:**  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates  
**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$   
**Require:**  $\theta_0$ : Initial parameter vector  
 $m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)  
 $v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)  
 $t \leftarrow 0$  (Initialize timestep)  
**while**  $\theta_t$  not converged **do**  
     $t \leftarrow t + 1$   
     $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )  
     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)  
     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)  
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)  
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)  
     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)  
**end while**  
**return**  $\theta_t$  (Resulting parameters)

---

Figure 1.5 – The detailed Adam algorithm showed in Kingma and Ba (2014)

---

In addition to first-order optimization algorithms, second-order optimization has been explored as well. One simple instantiation is Newton's method. Newton's method is based on a second-order Taylor series expansion to approximate  $f(x)$  near some point  $x$ . Newton's method could be helpful when the nearby point is a local minimum. In this case, it performs faster than gradient descent since it uses information about the second derivative which makes the convergence in fewer steps than gradient descent.

As we mentioned, optimization of ANNs can be difficult. One scenario is when the parameters get stuck at points that are neither maximum or minimum, we call these points as saddle points. A saddle point is the point where the derivatives become zero but not a local extremum on both axes. Research on how to escape saddle points when training ANNs has been developed in recent years (Dauphin et al., 2014; Jin et al., 2017).

# 2

# Sequence to Sequence Learning

In this chapter we will introduce several building blocks of sequence to sequence learning in deep learning scenarios – recurrent neural networks (RNNs), convolutional neural networks (CNNs) based sequence modelling and their variants.

---

## 2.1 Concept

Sequence to sequence learning typically denotes the methods to map the input sequence to a variable length output sequence. More specifically, sequence to sequence learning includes synced sequence input to sequence output (labels are assigned to each unit of sequence input) and unsynced sequence input to sequence output (labels are not aligned with input). They has been successfully used in the domain of speech recognition (Graves et al., 2013; Graves and Jaitly, 2014; Sak et al., 2014), machine translation (Luong et al., 2015; Bahdanau et al., 2014) and optical character recognition (OCR) (Lee and Osindero, 2016; He et al., 2016d). The length of input for sequence to sequence learning is usually unknown as a *priori* which would make MLPs fail easily. MLPs are powerful for the tasks whose inputs and outputs are of a fixed size, however, many problems have sequential properties. In the next section, we will present recurrent neural networks (RNNs), a class of ANNs whose connections between units are cyclic. The nature of RNNs allows them to model the temporal relations inside the sequential input.

---

## 2.2 Recurrent Neural Networks

Reccurent Neural Networks (RNNs) are sequential extension of feedforward neural networks. Given an input sequence of vectors  $(\mathbf{x}_1, \dots, \mathbf{x}_T)$ , the model

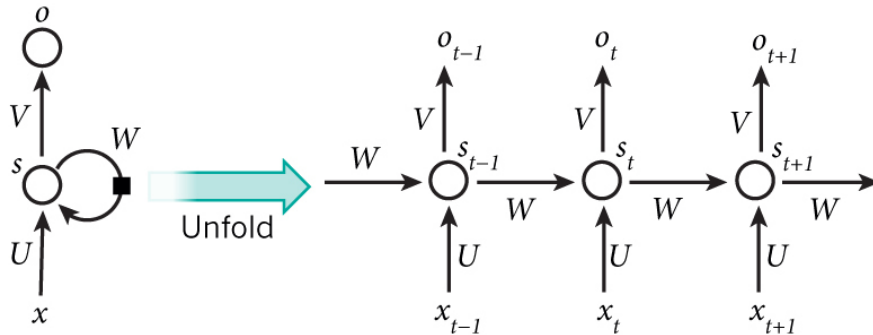
produces a sequence of hidden states ( $\mathbf{s}_1, \dots, \mathbf{s}_T$ ) and a sequence of outputs ( $\mathbf{y}_1, \dots, \mathbf{y}_T$ ), which are computed at time step  $t$  as follows

$$\mathbf{s}_t = \varphi(\mathbf{W}\mathbf{s}_{t-1} + \mathbf{U}\mathbf{x}_t) \quad (2.1)$$

$$\mathbf{o}_t = \mathbf{V}\mathbf{s}_t \quad (2.2)$$

where  $\mathbf{W}$  is the recurrent weight matrix,  $\mathbf{U}$  is the input-to-hidden weight matrix,  $\mathbf{V}$  is the hidden-to-output weight matrix and  $\varphi$  is an arbitrary activation function (usually a logistic sigmoid function or tanh function). The parameters are shared by all time steps in the network.

The RNNs could map sequences to sequences with known alignment between the inputs and outputs. We show an unrolled recurrent neural network in Figure 2.1.



**Figure 2.1** – A recurrent neural network unrolled in time (Figure adapted from Yoshua Bengio’s slides)

Training a recurrent neural network is similar to training MLPs. However, the gradient at each output depends not only on the current time step, but also the previous time steps, we call it backpropagation through time (BPTT). Training such networks via BPTT is known to be particularly difficult due to what is called the vanishing and exploding gradient problem, which hinders the models from learning long-term dependencies. The gradient  $\frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_{t-n}}$  can be computed as follows,

$$\frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_{t-n}} = \prod_{k=t-n+1}^t \mathbf{U}^T \text{diag}(\varphi'_k), \quad (2.3)$$

where  $\text{diag}(\varphi'_k) = \varphi'(\mathbf{W}\mathbf{s}_{t-1} + \mathbf{U}\mathbf{x}_t)$ . The equation above shows that the gradient

---

flow through time heavily depends on the hidden-to-hidden matrix  $\mathbf{U}$ , but  $\mathbf{W}$  and  $\mathbf{x}_t$  appear to play a limited role: they only come in the derivative of  $\varphi'$  mixed with  $\mathbf{U}$ . An improved architecture that efficiently utilize the information from different resources is proposed by Wu, et al in (Wu et al., 2016).

---

## 2.3 Long Short Term Memory

Long Short Term Memory (LSTM) is a special kind of recurrent structure which was proposed by Sepp Hochreiter and Jürgen Schmidhuber in 1997 (Hochreiter and Schmidhuber, 1997). It addresses the vanishing gradient problem commonly found in RNNs by incorporating gating functions into its state dynamics. The main components include a memory cell to store the state for the time step up to now, an input gate to modulate the extent to which a new input at time step  $t$  flows into the memory cell, an output gate to control the extent to which the information in the cell flows out, a forget gate to determine how much history would be preserved in the cell. More specifically, we define the computation at time step  $t$  as follows:

$$\mathbf{i}_t = \varphi(\mathbf{W}_{si}\mathbf{s}_{t-1} + \mathbf{U}_{xi}\mathbf{x}_t) \quad (2.4)$$

$$\mathbf{f}_t = \varphi(\mathbf{W}_{sf}\mathbf{s}_{t-1} + \mathbf{U}_{xf}\mathbf{x}_t) \quad (2.5)$$

$$\mathbf{o}_t = \varphi(\mathbf{W}_{so}\mathbf{s}_{t-1} + \mathbf{U}_{xo}\mathbf{x}_t) \quad (2.6)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot (\mathbf{W}_s\mathbf{s}_{t-1} + \mathbf{U}_{xc}\mathbf{x}_t) \quad (2.7)$$

$$\mathbf{s}_t = \mathbf{o}_t \odot \varphi(\mathbf{c}_t) \quad (2.8)$$

where  $\mathbf{i}_t$ ,  $\mathbf{f}_t$ ,  $\mathbf{o}_t$ ,  $\mathbf{c}_t$ ,  $\mathbf{s}_t$  denotes input gate, forget gate, output gate, memory cell state and output of the LSTM unit, respectively.  $\mathbf{W}_s$  and  $\mathbf{U}_x$  are weight matrices and  $\odot$  denotes Hadamard product (element-wise product).

Variants to LSTM includes peephole LSTM whose gates are computed based on the memory cell state  $\mathbf{c}_{t-1}$  and the input  $\mathbf{x}_t$ , convolutional LSTM which incorporates convolution operator that we will introduce in the following section. Gated recurrent unit (GRU) (Cho et al., 2014) is another gating mechanism which is similar to LSTM while there is only single gating unit determines the forgetting factor



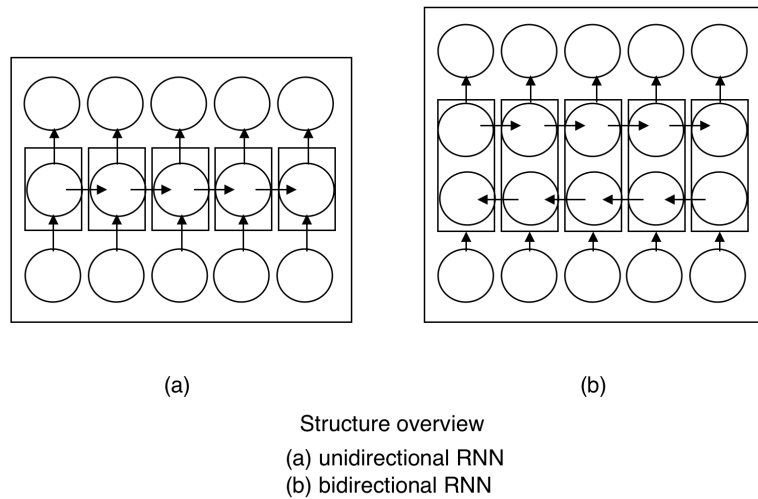
---

and the extent to update the state. Empirical comparisons between LSTM and GRU can be found in (Chung et al., 2014).

---

## 2.4 Bidirectional Structure

Bidirectional RNNs were proposed by Schuster and Paliwal in 1997 (Schuster and Paliwal, 1997). They use information not only from the past states but also from the future states, which can access long-range context in both directions. Bidirectional structures has been shown to be superior over unidirectional structure in Graves and Schmidhuber (2005). A diagram depicting the difference between two structures is shown in Figure 2.2.

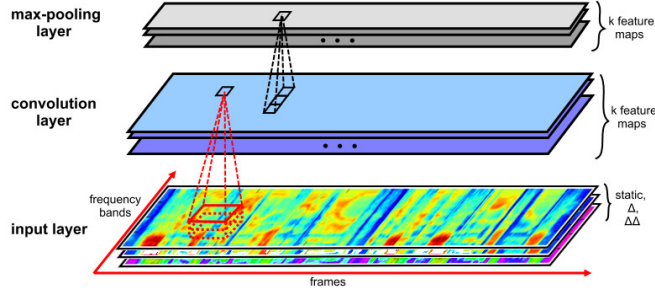


**Figure 2.2** – The difference between a bidirectional RNN and an unidirectional RNN. (Figure adapted from wikipedia)

---

## 2.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) is known as a class of feed-forward ANNs which were inspired by biological processes. They are widely used in the vision community for image classification, video recognition and other applications. CNNs



**Figure 2.3** – *The convolution layer and max-pooling layer applied upon input features.*

have been popular for sequence modelling in recent years (Venugopalan et al., 2015; Gehring et al., 2017) because of their computation efficiency and capacity to model long-range dependencies by stacking layers. A complete CNN is typically composed of stacked convolutional layers and pooling layers and at the top of which are multiple fully-connected layers. We introduce the building blocks of CNNs by taking acoustic features as input.

### 2.5.1 Convolution

As shown in Figure 2.3, given a sequence of input feature values  $\mathbf{X} \in \mathbb{R}^{c \times b \times f}$  with number of channels  $c$ , frequency bandwidth  $b$ , and time length  $f$ , the convolutional layer convolves  $\mathbf{X}$  with  $k$  filters  $\{\mathbf{W}_i\}_k$  where each  $\mathbf{W}_i \in \mathbb{R}^{c \times m \times n}$  is a 3D tensor with its width along the frequency axis equal to  $m$  and its length along frame axis equal to  $n$ . The resulting  $k$  pre-activation feature maps consist of a 3D tensor  $\mathbf{H} \in \mathbb{R}^{k \times b_H \times f_H}$ , in which each feature map  $\mathbf{H}_i$  is computed as follows:

$$\mathbf{H}_i = \mathbf{W}_i * \mathbf{X} + b_i, \quad i = 1, \dots, k. \quad (2.9)$$

The symbol  $*$  denotes the convolution operation and  $b_i$  is a bias parameter.

Convolution helps to extract features from input feature without losing spatial relations between units of the input. The size of the resulting feature maps is determined by the number of filters, the number of units by which the filters are slid (also known as stride) and the amount of padding around the border of the input (usually zero-padding). Each unit in the feature map is related to a local region of the input is known as receptive field. Another critical concept in CNNs

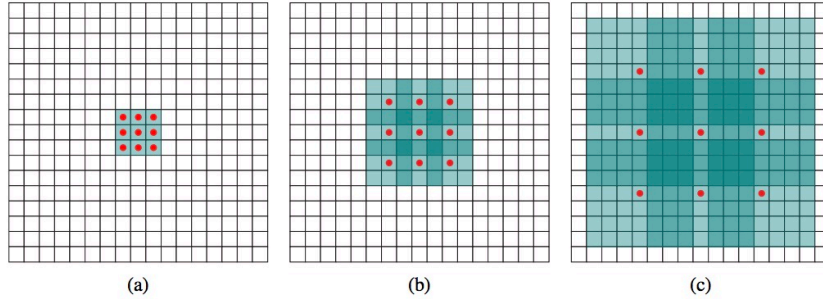


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a)  $F_1$  is produced from  $F_0$  by a 1-dilated convolution; each element in  $F_1$  has a receptive field of  $3 \times 3$ . (b)  $F_2$  is produced from  $F_1$  by a 2-dilated convolution; each element in  $F_2$  has a receptive field of  $7 \times 7$ . (c)  $F_3$  is produced from  $F_2$  by a 4-dilated convolution; each element in  $F_3$  has a receptive field of  $15 \times 15$ . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

**Figure 2.4** – Dilated convolution on 2D data. Figure is adapted from Yu and Koltun (2015)

is parameter sharing, which is used to control the number of parameters and to capture local features that could lie anywhere in the input.

Convolution has evolved into different variations in recent years. One popular variation is dilated convolution (Yu and Koltun, 2015), which is a convolution with a dilated filter (Diagram is shown in Figure 2.4). Dilation factor could be customized depending on the task. Dilated convolution has been widely applied to semantic segmentation and audio generation (Yu and Koltun, 2015; Oord et al., 2016).

## 2.5.2 Pooling

After the element-wise non-linearities are applied to the pre-activation feature maps, the features will pass through a pooling layer which outputs the value from  $p$  adjacent units. The pooling operation reduces the dimensionality of the input while retaining the important local features. Pooling operations fall into various categories, such as mean pooling (mean value is computed over a region), max pooling (max value is computed over a region) and etc. In the case of max pooling, suppose that the  $i$  th feature map before and after pooling are  $\tilde{\mathbf{H}}_i$  and  $\hat{\mathbf{H}}_i$ , then  $[\hat{\mathbf{H}}_i]_{r,t}$  at position  $(r, t)$  is computed by:

$$[\hat{\mathbf{H}}_i]_{r,t} = \max_{j=1}^p \{[\tilde{\mathbf{H}}_i]_{r \times s+j,t}\}, \quad (2.10)$$

---

where  $s$  is the step size and  $p$  is the pooling size, and all the  $[\tilde{\mathbf{H}}_i]_{r \times s+j, t}$  values inside the *max* have the same time index  $t$ . Pooling operation results in translation invariant features, which means the same (pooled) feature will be active even when the image undergoes (small) translations. This property is desirable when the tasks are position independent, for example, object recognition.

### 2.5.3 Fully Connected Layer

A fully connected layer (FCL) is typically a set of MLP layers with a softmax function at the top. The purpose of FCL is to combine the local features from convolutional layers and pooling layers in a non-linear fashion. FCL is important when the global feature is needed for the task, for example, image classification. In sequence-to-sequence learning tasks, FCL is usually replaced by  $1 \times 1$  convolution to keep the sequential nature. We refer readers to (Gehring et al., 2017) for more details.

---

## 2.6 Sequence Loss Function

Loss function design in unsynced sequence to sequence learning (labels are not aligned with input) is nontrivial due to its many-to-one, one-to-many or many-to-many mapping possibilities. We introduce a novel loss function, Connectionist Temporal Classification (CTC), which is proposed by Graves, et al (Graves et al., 2006) that solves the many-to-one mapping problem in unsegmented sequence data.

### 2.6.1 Connectionist Temporal Classification

Consider any sequence to sequence mapping task in which  $\mathbf{X} = \{X_1, \dots, X_T\}$  is the input sequence and  $\mathbf{Z} = \{Z_1, \dots, Z_L\}$  is the target sequence. In the case of speech recognition,  $\mathbf{X}$  is the acoustic signal and  $\mathbf{Z}$  is a sequence of symbols. In order to train the neural acoustic model,  $Pr(\mathbf{Z}|\mathbf{X})$  must be maximized for each input-output pair.

One way to provide a distribution over variable length output sequences given some much longer input sequence, is to introduce a many-to-one mapping of latent

---

variable sequences  $\mathbf{O} = \{O_1, \dots, O_T\}$  to shorter sequences that serve as the final predictions. The probability of some sequence  $\mathbf{Z}$  can then be defined to be the sum of the probabilities of all the latent sequences that map to that sequence. Connectionist Temporal Classification (CTC) (Graves et al., 2006) specifies a distribution over latent sequences by applying a softmax function to the output of the network for every time step, which provides a probability for emitting each label from the alphabet of output symbols at that time step  $Pr(O_t|\mathbf{X})$ . An extra *blank* output class ‘-’ is introduced to the alphabet for the latent sequences to represent the probability of not outputting a symbol at a particular time step. Each latent sequence sampled from this distribution can now be transformed into an output sequence using the many-to-one mapping function  $\sigma(\cdot)$  which first merges the repetitions of consecutive non-blank labels to a single label and subsequently removes the blank labels as shown in Equation 2.11:

$$\left. \begin{array}{l} \sigma(a, b, c, -, -) \\ \sigma(a, b, -, c, c) \\ \sigma(a, a, b, b, c) \\ \sigma(-, a, -, b, c) \\ \vdots \\ \sigma(-, -, a, b, c) \end{array} \right\} = (a, b, c). \quad (2.11)$$

Therefore, the final output sequence probability is a summation over all possible sequences  $\pi$  that yield to  $\mathbf{Z}$  after applying the function  $\sigma$ :

$$Pr(\mathbf{Z}|\mathbf{X}) = \sum_{\mathbf{o} \in \sigma^{-1}(\mathbf{z})} Pr(\mathbf{O}|\mathbf{X}). \quad (2.12)$$

A dynamic programming algorithm similar to the forward algorithm for HMMs Graves (2012b) is used to compute the sum in Equation 2.12 in an efficient way. The intermediate values of this dynamic programming can also be used to compute the gradient of  $\ln Pr(\mathbf{Z}|\mathbf{X})$  with respect to the neural network outputs efficiently.

To generate predictions from a trained model using CTC, we use the *best path decoding* algorithm. Since the model assumes that the latent symbols are independent given the network outputs in the framewise case, the latent sequence with the highest probability is simply obtained by emitting the most probable label at each time-step. The predicted sequence is then given by applying  $\sigma(\cdot)$  to that latent

---

sequence prediction:

$$\mathbf{L} \approx \sigma(\pi^*), \tag{2.13}$$

in which  $\pi^*$  is the concatenation of the most probable output and is formalized by  $\pi^* = \text{Argmax}_{\pi} Pr(\pi|\mathbf{X})$ . Note that this is not necessarily the output sequence with the highest probability. Finding this sequence is generally not tractable and requires some approximate search procedure like a beam-search.

In the following chapters, we present two new approaches of sequence-to-sequence modelling on speech. The first one is based on CNNs and the other is built on RNNs.

# 3

## Prologue to First Article

**Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks.** Ying Zhang, Mohammad Pezeshki, Philémon Brakel, Saizheng Zhang, César Laurent Yoshua Bengio, Aaron Courville. Interspeech, 2016.

*Personal Contribution.* The underlying idea of combining convolutional neural networks with connectionist temporal classification was mine. Mohammad Pezeshki, Philémon Brakel and I discussed the structure variations based on the model. I conducted the experiments on TIMIT and Mohammad worked on Wall Street Journal. I implemented the majority of the code based on our private speech repository in MILA. I contributed to the writing of the paper while Mohammad Pezeshki, Philémon Brakel, Saizheng Zhang and César Laurent helped to revise the paper, with valuable comments from my supervisors Aaron Courville and Yoshua Bengio.

# 4

# Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks

---

## 4.1 Introduction

Recently, Convolutional Neural Networks (CNNs) (LeCun et al., 1998) have achieved great success in acoustic modeling (Abdel-Hamid et al., 2012; Sainath et al., 2013b,a). In the context of Automatic Speech Recognition, CNNs are usually combined with HMMs/GMMs (Mohamed et al., 2012; Hinton et al., 2012), like regular Deep Neural Networks (DNNs), which results in a hybrid system (Abdel-Hamid et al., 2012; Sainath et al., 2013b,a). In the typical hybrid system, the neural net is trained to predict frame-level targets obtained from a forced alignment generated by an HMM/GMM system. The temporal modeling and decoding operations are still handled by an HMM but the posterior state predictions are generated using the neural network.

This hybrid approach is problematic in that training the different modules separately with different criteria may not be optimal for solving the final task. As a consequence, it often requires additional hyperparameter tuning for each training stage which can be laborious and time consuming. Furthermore, these issues have motivated a recent surge of interests in training ‘end-to-end’ systems (Hannun et al., 2014; Bahdanau et al., 2016; Miao et al., 2015). End-to-end neural systems for speech recognition typically replace the HMM with a neural network that provides a distribution over sequences directly. Two popular neural network sequence models are Connectionist Temporal Classification (CTC) (Graves et al., 2006) and recurrent models for sequence generation (Bahdanau et al., 2016; Chorowski et al., 2015).

To the best of our knowledge, all end-to-end neural speech recognition systems employ recurrent neural networks in at least some part of the processing pipeline. The most successful recurrent neural network architecture used in this context is the Long Short-Term Memory (LSTM) (Graves et al., 2013; Graves, 2012a; Hochreiter



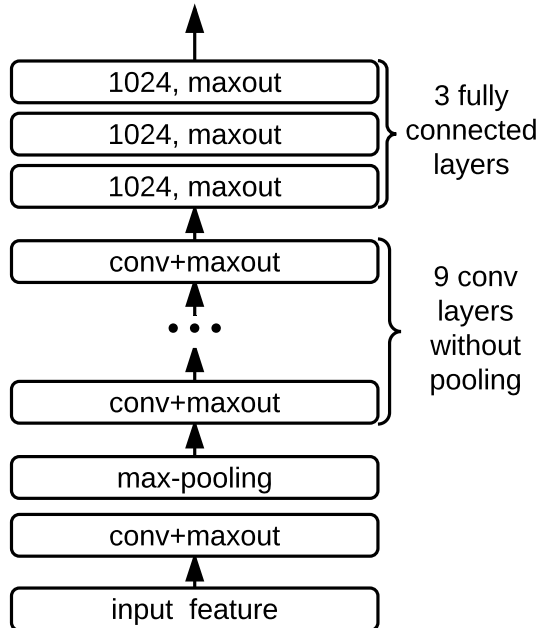
---

and Schmidhuber, 1997; Vinyals et al., 2012). For example, a model with multiple layers of bi-directional LSTMs and CTC on top which is pre-trained with the transducer networks (Graves et al., 2013; Graves, 2012a) obtained the state-of-the-art on the TIMIT dataset. After these successes on phoneme recognition, similar systems have been proposed in which multiple layers of RNNs were combined with CTC to perform large vocabulary continuous speech recognition (Hannun et al., 2014; Miao et al., 2013). It seems that RNNs have become somewhat of a default method for end-to-end models while hybrid systems still tend to rely on feed-forward architectures.

While the results of these RNN-based end-to-end systems are impressive, there are two important downsides to using RNNs/LSTMs: (1) The training speed can be very slow due to the iterative multiplications over time when the input sequence is very long; (2) The training process is sometimes tricky due to the well-known problem of gradient vanishing/exploding (Hochreiter, 1991; Bengio et al., 1994). Although various approaches have been proposed to address these issues, such as data/model parallelization across multiple GPUs (Hannun et al., 2014; Sutskever et al., 2014) and careful initializations for recurrent connections (Le et al., 2015), those models still suffer from computationally intensive and otherwise demanding training procedures.

Inspired by the strengths of both CNNs and CTC, we propose an end-to-end speech framework in which we combine CNNs with CTC without intermediate recurrent layers. We present experiments on the TIMIT dataset and show that such a system is able to obtain results that are comparable to those obtained with multiple layers of LSTMs. The only previous attempt to combine CNNs with CTC that we know about (Song and Cai), led to results that were far from the state-of-the-art. It is not straightforward to incorporate CNN into an end-to-end manner since the task may require the model to incorporate long-term dependencies. While RNNs can learn these kind of dependencies and have been combined with CTC for this very reason, it was not known whether CNNs were able to learn the required temporal relationships.

In this paper, we argue that in a CNN of sufficient depth, the higher-layer features are capable of capturing temporal dependencies with suitable context information. Using small filter sizes along the spectrogram frequency axis, the model is able to learn fine-grained localized features, while multiple stacked convolutional



**Figure 4.1** – Network structure for phoneme recognition on the TIMIT dataset. The model consists of 10 convolutional layers followed by 3 fully-connected layers on the top. All convolutional layers have the filter size of  $3 \times 5$  and we use max-pooling with size of  $3 \times 1$  only after the first convolutional layer. First and second numbers correspond to frequency and time axes respectively.

layers help to learn diverse features on different time/frequency scales and provide the required non-linear modeling capabilities.

Unlike the time windows applied in DNN systems (Abdel-Hamid et al., 2012; Sainath et al., 2013b,a), the temporal modeling is deployed within convolutional layers, where we perform a 2D convolution similar to vision tasks, and multiple convolutional layers are stacked to provide a relatively large context window for each output prediction of the highest layer. The convolutional layers are followed by multiple fully connected layers and, finally, CTC is added on the top of the model. Following the suggestion from (Sainath et al., 2013a), we only perform pooling along the frequency band on the first convolutional layer. Specifically, we evaluate our model on phoneme recognition for the TIMIT dataset.

---

## 4.2 Experiments

In this section, we evaluate the proposed model on phoneme recognition for the TIMIT dataset. The model architecture is shown in Figure 4.1.

### 4.2.1 Data

We evaluate our models on the TIMIT (Garofolo et al., 1993) corpus where we use the standard 462-speaker training set with all SA records removed. The 50-speaker development set is used for early stopping. The evaluation is performed on the core test set (including 192 sentences). The raw audio is transformed into 40-dimensional log mel-filter-bank (plus energy term) coefficients with deltas and delta-deltas, which results in 123 dimensional features. Each dimension is normalized to have zero mean and unit variance over the training set. We use 61 phone labels plus a blank label for training and then the output is mapped to 39 phonemes for scoring.

### 4.2.2 Models

Our best model consists of 10 convolutional layers and 3 fully-connected hidden layers. Unlike the other layers, the first convolutional layer is followed by a pooling layer, which is described in section 2. The pooling size is  $3 \times 1$ , which means we only pool over the frequency axis. The filter size is  $3 \times 5$  across the layers. The model has 128 feature maps in the first four convolutional layers and 256 feature maps in the remaining six convolutional layers. Each fully-connected layer has 1024 units. Maxout with 2 piece-wise linear functions is used as the activation function. Some other architectures are also evaluated for comparison, see section 4.4 for more details.

### 4.2.3 Training and Evaluation

To optimize the model, we use Adam (Kingma and Ba, 2014) with learning rate  $10^{-4}$ . Stochastic gradient descent with learning rate  $10^{-5}$  is then used for fine-tuning. Batch size 20 is used during training. The initial weight values were drawn uniformly from the interval  $[-0.05, 0.05]$ . Dropout (Srivastava et al., 2014) with a probability of 0.3 is added across the layers except for the input and output

---

layers. L2 norm with coefficient  $1e-5$  is applied at fine-tuning stage. At test time, simple best path decoding (at the CTC frame level) is used to get the predicted sequences.

#### 4.2.4 Results

Our model achieves 18.2% phoneme error rate on the core test set, which is slightly better than the LSTM baseline model and the transducer model with an explicit RNN language model. The details are presented in Table 4.1. Notice that the CNN model could take much less time to train in comparison with the LSTM model when keeping roughly the same number of parameters. In our setup on TIMIT, we get  $2.5\times$  faster training speed by using the CNN model without deliberately optimizing the implementation. We suppose that the gain of the computation efficiency might be more dramatic with a larger dataset.

To further investigate the different structural aspects of our model, we disentangle the analysis into three sub-experiments considering the number of convolutional layers, the filter sizes and the activation functions, as shown in table 4.1. It turns out that the model may benefit from (1) more layers, which results in more nonlinearities and larger input receptive fields for units in the top layers; (2) reasonably large context windows, which help the model to capture the spatial/temporal relations of input sequences in reasonable time-scales; (3) the Maxout unit, which has more functional freedoms comparing to ReLU and parametric ReLU.

---

### 4.3 Discussion

Our results showed that convolutional architectures with CTC cost can achieve results comparable to the state-of-the-art by adopting the following methodology: (1) Using a significantly deeper architecture that results in a more non-linear function and also wider receptive fields along both frequency and temporal axes; (2) Using maxout non-linearities in order to make the optimization easier; and (3) Careful model regularization that yields better generalization in test time, especially for small datasets such as TIMIT, where over-fitting happens easily.

**Table 4.1** – Phoneme Error Rate (PER) on TIMIT. ‘NP’ is the number of parameters. ‘BiLSTM-3L-250H’ denotes the model has 3 bidirectional LSTM layers with 250 units in each direction. In the CNN model, (3, 5) is the filter size. Results suggest that deeper architecture and larger filter sizes leads to better performance. The best performing model on Development set, has a test PER of 18.2 %

Model	NP	Dev PER	Test PER
BiLSTM-3L-250H Graves et al. (2013)	3.8M	-	18.6%
BiLSTM-5L-250H Graves et al. (2013)	6.8M	-	18.4%
TRANS-3L-250H Graves et al. (2013)	4.3M	-	18.3%
CNN-(3,5)-10L-ReLU	4.3M	17.4%	19.3%
CNN-(3,5)-10L-PReLU	4.3M	17.2%	18.9%
CNN-(3,5)-6L-maxout	4.3M	18.7%	21.2%
CNN-(3,5)-8L-maxout	4.3M	17.7%	19.8%
CNN-(3,3)-10L-maxout	4.3M	18.4%	19.9%
CNN-(3,5)-10L-maxout	4.3M	<b>16.7%</b>	<b>18.2%</b>

We conjecture that the convolutional CTC model might be easier to train on phoneme-level sequences rather than the character-level. Our intuition is that the local structures within the phonemes are more robust and can easily be captured by the model. Additionally, phoneme-level training might not require the modeling of many long-term dependencies in comparison with character-level training. As a result, for a convolutional model, learning the phonemes structure seems to be easier, but empirical research needs to be done to investigate if this is indeed the case.

Finally, an important point that favors convolutional over recurrent architectures is the training speed. In a CNN, the training time can be rendered virtually independent of the length of the input sequence due to the parallel nature of convolutional models and the highly optimized CNN libraries available (Abadi et al., 2016). Computations in a recurrent model are sequential and cannot be easily parallelized. The training time for RNNs increases at least linearly with the length of the input sequence.

# 5

## Prologue to Second Article

**Deep Complex Networks.** Chiheb Trabelsi <sup>1</sup>, Olexa Bilaniuk <sup>1</sup>, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, Christopher J.Pal. Sixth International Conference on Learning Representations (ICLR), 2018.

*Personal Contribution.* Chiheb Trabelsi and I proposed the idea of recurrent complex networks. I prepared the code for recurrent models with audio data, conducted the spectrum experiments and wrote the paper on that section. The rest of the authors worked on the feed-foward nets.

---

1. Equal first author

# 6

# Deep Complex Networks

---

## 6.1 Introduction

Recent research advances have made significant progress in addressing the difficulties involved in learning deep neural network architectures. Key innovations include normalization techniques (Ioffe and Szegedy, 2015; Salimans and Kingma, 2016) and the emergence of gating-based feed-forward neural networks like Highway Networks (Srivastava et al., 2015). Residual networks (He et al., 2016b,c) have emerged as one of the most popular and effective strategies for training very deep convolutional neural networks (CNNs). Both highway networks and residual networks facilitate the training of deep networks by providing shortcut paths for easy gradient flow to lower network layers thereby diminishing the effects of vanishing gradients (Hochreiter, 1991). He et al. (2016c) show that learning explicit residuals of layers helps in avoiding the vanishing gradient problem and provides the network with an easier optimization problem. Batch normalization (Ioffe and Szegedy, 2015) demonstrates that standardizing the activations of intermediate layers in a network across a minibatch acts as a powerful regularizer as well as providing faster training and better convergence properties. Further, such techniques that standardize layer outputs become critical in deep architectures due to the vanishing and exploding gradient problems.

The role of representations based on complex numbers has started to receive increased attention, due to their potential to enable easier optimization (Nitta, 2002), better generalization characteristics (Hirose and Yoshida, 2012), faster learning (Arjovsky et al., 2016; Danihelka et al., 2016; Wisdom et al., 2016) and to allow for noise-robust memory mechanisms (Danihelka et al., 2016). Wisdom et al. (2016) and Arjovsky et al. (2016) show that using complex numbers in recurrent neural networks (RNNs) allows the network to have a richer representational capacity. Danihelka et al. (2016) present an LSTM (Hochreiter and Schmidhuber,

---

1997) architecture augmented with associative memory with complex-valued internal representations. Their work highlights the advantages of using complex-valued representations with respect to retrieval and insertion into an associative memory. In residual networks, the output of each block is added to the output history accumulated by summation until that point. An efficient retrieval mechanism could help to extract useful information and process it within the block.

In order to exploit the advantages offered by complex representations, we present a general formulation for the building components of complex-valued deep neural networks and apply it to the context of feed-forward convolutional networks and convolutional LSTMs. Our contributions in this paper are as follows:

1. A formulation of complex batch normalization, which is described in Section 6.2.5;
2. Complex weight initialization, which is presented in Section 6.2.6;
3. A comparison of different complex-valued ReLU-based activation functions presented in Section 6.3.1;
4. A state of the art result on the MusicNet multi-instrument music transcription dataset, presented in Section 6.3.2;
5. A state of the art result in the Speech Spectrum Prediction task on the TIMIT dataset, presented in Section 6.3.3.

We perform a sanity check of our deep complex network and demonstrate its effectiveness on standard image classification benchmarks, specifically, CIFAR-10, CIFAR-100. We also use a reduced-training set of SVHN that we call SVHN\*. For audio-related tasks, we perform a music transcription task on the MusicNet dataset and a Speech Spectrum prediction task on TIMIT. The results obtained for vision classification tasks show that learning complex-valued representations results in performance that is competitive with the respective real-valued architectures. Our promising results in music transcription and speech spectrum prediction underscore the potential of deep complex-valued neural networks applied to acoustic related tasks<sup>1</sup> – We continue this paper with discussion of motivation for using complex operations and related work.

---

1. The source code is located at [http://github.com/ChihebTrabelsi/deep\\_complex\\_networks](http://github.com/ChihebTrabelsi/deep_complex_networks)



---

## 6.2 Complex Building Blocks

In this section, we present the core of our work, laying down the mathematical framework for implementing complex-valued building blocks of a deep neural network.

### 6.2.1 Representation of Complex Numbers

We start by outlining the way in which complex numbers are represented in our framework. A complex number  $z = a + ib$  has a real component  $a$  and an imaginary component  $b$ . We represent the real part  $a$  and the imaginary part  $b$  of a complex number as logically distinct real valued entities and simulate complex arithmetic using real-valued arithmetic internally. Consider a typical real-valued 2D convolution layer that has  $N$  feature maps such that  $N$  is divisible by 2; to represent these as complex numbers, we allocate the *first*  $N/2$  feature maps to represent the real components and the remaining  $N/2$  to represent the imaginary ones. Thus, for a four dimensional weight tensor  $W$  that links  $N_{in}$  input feature maps to  $N_{out}$  output feature maps and whose kernel size is  $m \times m$  we would have a weight tensor of size  $(N_{out} \times N_{in} \times m \times m) / 2$  complex weights.

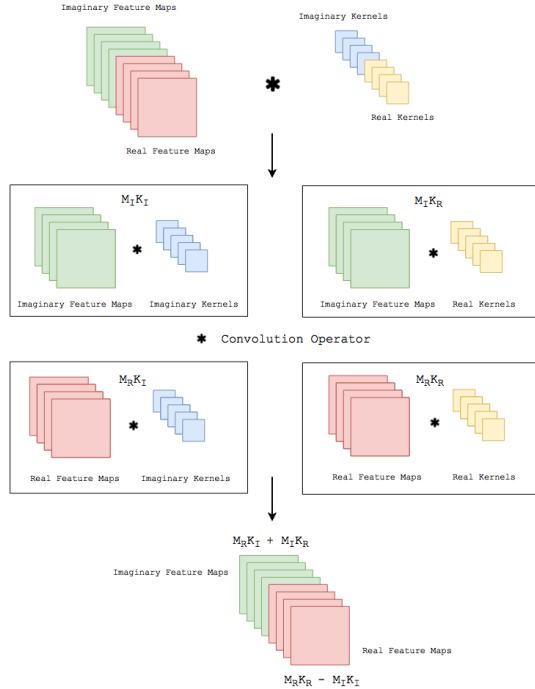
### 6.2.2 Complex Convolution

In order to perform the equivalent of a traditional real-valued 2D convolution in the complex domain, we convolve a complex filter matrix  $W = A + iB$  by a complex vector  $h = x + iy$  where  $A$  and  $B$  are real matrices and  $x$  and  $y$  are real vectors since we are simulating complex arithmetic using real-valued entities. As the convolution operator is distributive, convolving the vector  $h$  by the filter  $W$  we obtain:

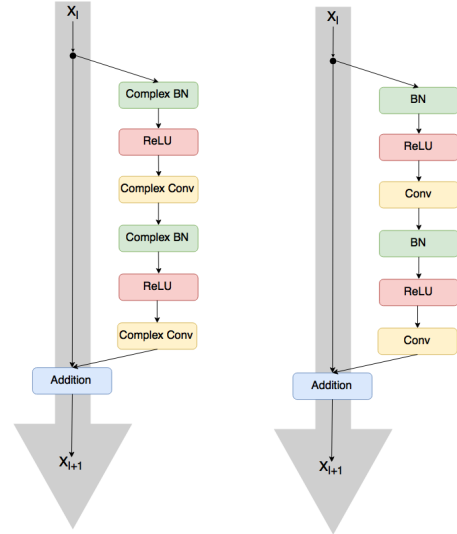
$$W * h = (A * x - B * y) + i(B * x + A * y). \quad (6.1)$$

As illustrated in Figure 6.1a, if we use matrix notation to represent real and imaginary parts of the convolution operation we have:

$$\begin{bmatrix} \Re(W * h) \\ \Im(W * h) \end{bmatrix} = \begin{bmatrix} A & -B \\ B & A \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}. \quad (6.2)$$



(a) An illustration of the complex convolution operator.



(b) A complex convolutional residual network (left) and an equivalent real-valued residual network (right).

Figure 6.1 – Complex convolution and residual network implementation details.

### 6.2.3 Complex Differentiability

In order to perform backpropagation in a complex-valued neural network, a sufficient condition is to have a cost function and activations that are *differentiable* with respect to the real and imaginary parts of each complex parameter in the network.

By constraining activation functions to be *complex differentiable* or *holomorphic*, we restrict the use of possible activation functions for a complex valued neural networks. Hirose and Yoshida (2012) shows that it is unnecessarily restrictive to limit oneself only to holomorphic activation functions; Those functions that are differentiable with respect to the real part and the imaginary part of each parameter are also compatible with backpropagation. (Arjovsky et al., 2016; Wisdom et al., 2016; Danihelka et al., 2016) have used non-holomorphic activation functions and optimized the network using regular, real-valued backpropagation to compute partial derivatives of the cost with respect to the real and imaginary parts.

---

Even though their use greatly restricts the set of potential activations, it is worth mentioning that holomorphic functions can be leveraged for computational efficiency purposes. As pointed out in Sarroff et al. (2015), using holomorphic functions allows one to share gradient values. So, instead of computing and back-propagating 4 different gradients, only 2 are required.

## 6.2.4 Complex-Valued Activations

### ModReLU

Numerous activation functions have been proposed in the literature in order to deal with complex-valued representations. (Arjovsky et al., 2016) have proposed modReLU, which is defined as follows:

$$\text{modReLU}(z) = \text{ReLU}(|z| + b) e^{i\theta_z} = \begin{cases} (|z| + b) \frac{z}{|z|} & \text{if } |z| + b \geq 0, \\ 0 & \text{otherwise,} \end{cases} \quad (6.3)$$

where  $z \in \mathbb{C}$ ,  $\theta_z$  is the phase of  $z$ , and  $b \in \mathbb{R}$  is a learnable parameter. As  $|z|$  is always positive, a bias  $b$  is introduced in order to create a “*dead zone*” of radius  $b$  around the origin 0 where the neuron is inactive, and outside of which it is active. The authors have used modReLU in the context of unitary RNNs. Their design of modReLU is motivated by the fact that applying separate ReLUs on both real and imaginary parts of a neuron performs poorly on toy tasks. The intuition behind the design of modReLU is to preserve the pre-activated phase  $\theta_z$ , as altering it with an activation function severely impacts the complex-valued representation. modReLU does not satisfy the Cauchy-Riemann equations, and thus is not holomorphic. We have tested modReLU in deep feed-forward complex networks and the results are given in Table 6.1.

### ℂReLU and $z$ ReLU

We call Complex ReLU (or ℂReLU) the complex activation that applies separate ReLUs on both of the real and the imaginary part of a neuron, i.e:

$$\mathbb{C}\text{ReLU}(z) = \text{ReLU}(\Re(z)) + i \text{ReLU}(\Im(z)). \quad (6.4)$$

---

$\mathbb{C}\text{ReLU}$  satisfies the Cauchy-Riemann equations when both the real and imaginary parts are at the same time either strictly positive or strictly negative. This means that  $\mathbb{C}\text{ReLU}$  satisfies the Cauchy-Riemann equations when  $\theta_z \in ]0, \pi/2[$  or  $\theta_z \in ]\pi, 3\pi/2[$ . We have tested  $\mathbb{C}\text{ReLU}$  in deep feed-forward neural networks and the results are given in Table 6.1.

It is also worthwhile to mention the work done by Guberman (2016) where a ReLU-based complex activation which satisfies the Cauchy-Riemann equations everywhere except for the set of points  $\{\Re(z) > 0, \Im(z) = 0\} \cup \{\Re(z) = 0, \Im(z) > 0\}$  is used. The activation function has similarities to  $\mathbb{C}\text{ReLU}$ . We call Guberman (2016) activation as  $z\text{ReLU}$  and is defined as follows:

$$z\text{ReLU}(z) = \begin{cases} z & \text{if } \theta_z \in [0, \pi/2], \\ 0 & \text{otherwise,} \end{cases} \quad (6.5)$$

We have tested  $z\text{ReLU}$  in deep feed-forward complex networks and the results are given in Table 6.1.

### 6.2.5 Complex Batch Normalization

Deep networks generally rely upon Batch Normalization (Ioffe and Szegedy, 2015) to accelerate learning. In some cases batch normalization is essential to optimize the model. The standard formulation of Batch Normalization applies only to real values. In this section, we propose a batch normalization formulation that can be applied for complex values.

To standardize an array of complex numbers to the standard normal complex distribution, it is not sufficient to translate and scale them such that their mean is 0 and their variance 1. This type of normalization does not ensure equal variance in both the real and imaginary components, and the resulting distribution is not guaranteed to be circular; It will be elliptical, potentially with high eccentricity.

We instead choose to treat this problem as one of whitening 2D vectors, which implies scaling the data by the square root of their variances along each of the two principal components. This can be done by multiplying the  $\mathbf{0}$ -centered data  $(\mathbf{x} - \mathbb{E}[\mathbf{x}])$  by the inverse square root of the  $2 \times 2$  covariance matrix  $\mathbf{V}$ :

$$\tilde{\mathbf{x}} = (\mathbf{V})^{-\frac{1}{2}} (\mathbf{x} - \mathbb{E}[\mathbf{x}]),$$

where the covariance matrix  $\mathbf{V}$  is

$$\mathbf{V} = \begin{pmatrix} V_{rr} & V_{ri} \\ V_{ir} & V_{ii} \end{pmatrix} = \begin{pmatrix} \text{Cov}(\Re\{\mathbf{x}\}, \Re\{\mathbf{x}\}) & \text{Cov}(\Re\{\mathbf{x}\}, \Im\{\mathbf{x}\}) \\ \text{Cov}(\Im\{\mathbf{x}\}, \Re\{\mathbf{x}\}) & \text{Cov}(\Im\{\mathbf{x}\}, \Im\{\mathbf{x}\}) \end{pmatrix}.$$

The square root and inverse of  $2 \times 2$  matrices has an inexpensive, analytical solution, and its existence is guaranteed by the positive (semi-)definiteness of  $\mathbf{V}$ . Positive definiteness of  $\mathbf{V}$  is ensured by the addition of  $\epsilon \mathbf{I}$  to  $\mathbf{V}$  (Tikhonov regularization). The mean subtraction and multiplication by the inverse square root of the variance ensures that  $\tilde{\mathbf{x}}$  has standard complex distribution with mean  $\mu = 0$ , covariance  $\Gamma = 1$  and pseudo-covariance (also called relation)  $C = 0$ . The mean, the covariance and the pseudo-covariance are given by:

$$\begin{aligned} \mu &= \mathbb{E}[\tilde{\mathbf{x}}] \\ \Gamma &= \mathbb{E}[(\tilde{\mathbf{x}} - \mu)(\tilde{\mathbf{x}} - \mu)^*] = V_{rr} + V_{ii} + i(V_{ir} - V_{ri}) \\ C &= \mathbb{E}[(\tilde{\mathbf{x}} - \mu)(\tilde{\mathbf{x}} - \mu)] = V_{rr} - V_{ii} + i(V_{ir} + V_{ri}). \end{aligned} \tag{6.6}$$

The normalization procedure allows one to decorrelate the imaginary and real parts of a unit. This has the advantage of avoiding co-adaptation between the two components which reduces the risk of overfitting (Cogswell et al., 2015; Srivastava et al., 2014).

Analogously to the real-valued batch normalization algorithm, we use two parameters,  $\beta$  and  $\gamma$ . The shift parameter  $\beta$  is a complex parameter with two learnable components (the real and imaginary means). The scaling parameter  $\gamma$  is a  $2 \times 2$  positive semi-definite matrix with only three degrees of freedom, and thus only three learnable components. In much the same way that the matrix  $(\mathbf{V})^{-\frac{1}{2}}$  normalized the variance of the input to 1 along both of its original principal components, so does  $\gamma$  scale the input along desired new principal components to achieve a desired variance. The scaling parameter  $\gamma$  is given by:

$$\gamma = \begin{pmatrix} \gamma_{rr} & \gamma_{ri} \\ \gamma_{ri} & \gamma_{ii} \end{pmatrix}.$$

As the normalized input  $\tilde{\mathbf{x}}$  has real and imaginary variance 1, we initialize both  $\gamma_{rr}$  and  $\gamma_{ii}$  to  $1/\sqrt{2}$  in order to obtain a modulus of 1 for the variance of the normalized value.  $\gamma_{ri}$ ,  $\Re\{\beta\}$  and  $\Im\{\beta\}$  are initialized to 0. The complex batch

---

normalization is defined as:

$$\text{BN}(\tilde{\mathbf{x}}) = \gamma \tilde{\mathbf{x}} + \boldsymbol{\beta}. \quad (6.7)$$

We use running averages with momentum to maintain an estimate of the complex batch normalization statistics during training and testing. The moving averages of  $V_{ri}$  and  $\boldsymbol{\beta}$  are initialized to 0. The moving averages of  $V_{rr}$  and  $V_{ii}$  are initialized to  $1/\sqrt{2}$ . The momentum for the moving averages is set to 0.9.

### 6.2.6 Complex Weight Initialization

In a general case, particularly when batch normalization is not performed, proper initialization is critical in reducing the risks of vanishing or exploding gradients. To do this, we follow the same steps as in [Glorot and Bengio \(2010\)](#) and [He et al. \(2015\)](#) to derive the variance of the complex weight parameters.

A complex weight has a polar form as well as a rectangular form

$$W = |W|e^{i\theta} = \Re\{W\} + i \Im\{W\}, \quad (6.8)$$

where  $\theta$  and  $|W|$  are respectively the argument (phase) and magnitude of  $W$ .

Variance is the difference between the *expectation of the squared magnitude* and the *square of the expectation*:

$$\text{Var}(W) = \mathbb{E}[WW^*] - (\mathbb{E}[W])^2 = \mathbb{E}[|W|^2] - (\mathbb{E}[W])^2,$$

which reduces, in the case of  $W$  symmetrically distributed around 0, to  $\mathbb{E}[|W|^2]$ . We do not know yet the value of  $\text{Var}(W) = \mathbb{E}[|W|^2]$ . However, we do know a related quantity,  $\text{Var}(|W|)$ , because the magnitude of complex normal values,  $|W|$ , follows the Rayleigh distribution (Chi-distributed with two degrees of freedom (DOFs)). This quantity is

$$\text{Var}(|W|) = \mathbb{E}[|W||W|^*] - (\mathbb{E}[|W|])^2 = \mathbb{E}[|W|^2] - (\mathbb{E}[|W|])^2. \quad (6.9)$$

Putting them together:

$$\text{Var}(|W|) = \text{Var}(W) - (\mathbb{E}[|W|])^2, \text{ and } \text{Var}(W) = \text{Var}(|W|) + (\mathbb{E}[|W|])^2.$$

---

We now have a formulation for the variance of  $W$  in terms of the variance and expectation of its magnitude, both properties analytically computable from the Rayleigh distribution’s single parameter,  $\sigma$ , indicating the *mode*. These are:

$$\mathbb{E}[|W|] = \sigma\sqrt{\frac{\pi}{2}}, \quad \text{Var}(|W|) = \frac{4 - \pi}{2}\sigma^2.$$

The variance of  $W$  can thus be expressed in terms of its generating Rayleigh distribution’s single parameter,  $\sigma$ , thus:

$$\text{Var}(W) = \frac{4 - \pi}{2}\sigma^2 + \left(\sigma\sqrt{\frac{\pi}{2}}\right)^2 = 2\sigma^2. \quad (6.10)$$

If we want to respect the [Glorot and Bengio \(2010\)](#) criterion which ensures that the variances of the input, the output and their gradients are the same, then we would have  $\text{Var}(W) = 2/(n_{in} + n_{out})$ , where  $n_{in}$  and  $n_{out}$  are the number of input and output units respectively. In such case,  $\sigma = 1/\sqrt{n_{in} + n_{out}}$ . If we want to respect the [He et al. \(2015\)](#) initialization that presents an initialization criterion that is specific to ReLUs, then  $\text{Var}(W) = 2/n_{in}$  which  $\sigma = 1/\sqrt{n_{in}}$ .

The magnitude of the complex parameter  $W$  is then initialized using the Rayleigh distribution with the appropriate mode  $\sigma$ . We can see from equation 6.10, that the variance of  $W$  depends on its magnitude and not on its phase. We then initialize the phase using the uniform distribution between  $-\pi$  and  $\pi$ . By performing the multiplication of the magnitude by the phasor as is detailed in equation 6.8, we perform the complete initialization of the complex parameter.

In all the experiments that we report, we use variant of this initialization which leverages the independence property of unitary matrices. As it is stated in [Cogswell et al. \(2015\)](#), [Srivastava et al. \(2014\)](#), and [Tompson et al. \(2015\)](#), learning decorrelated features is beneficial for learning as it allows to perform better generalization and faster learning. This motivates us to achieve initialization by considering a (semi-)unitary matrix which is reshaped to the size of the weight tensor. Once this is done, the weight tensor is multiplied by  $\sqrt{He_{var}/\text{Var}(W)}$  or  $\sqrt{Glorot_{var}/\text{Var}(W)}$  where  $Glorot_{var}$  and  $He_{var}$  are respectively equal to  $2/(n_{in} + n_{out})$  and  $2/n_{in}$ . In such a way we allow kernels to be independent from each other as much as possible while respecting the desired criterion. Note that we perform the analogous initialization for real-valued models by leveraging the independence property of orthogonal matrices in order to build kernels that are as much independent from each other as possible while respecting a given criterion.

**Table 6.1** – Classification error on CIFAR-10, CIFAR-100 and SVHN\* using different complex activations functions ( $z$ ReLU, modReLU and  $\mathbb{C}$ ReLU). WS, DN and IB stand for the wide and shallow, deep and narrow and in-between models respectively. The prefixes R & C refer to the real and complex valued networks respectively. Performance differences between the real network and the complex network using  $\mathbb{C}$ ReLU are reported between their respective best models. All models are constructed to have roughly 1.7M parameters except the modReLU models which have roughly 2.5M parameters. modReLU and  $z$ ReLU were largely outperformed by  $\mathbb{C}$ ReLU in the reported experiments. Due to limited resources, we haven’t performed all possible experiments as the conducted ones are already conclusive. A ”-” is filled in front of an unperformed experiment.

ARCH	CIFAR-10			CIFAR-100			SVHN*		
	$z$ ReLU	MODReLU	$\mathbb{C}$ ReLU	$z$ ReLU	MODReLU	$\mathbb{C}$ ReLU	$z$ ReLU	MODReLU	$\mathbb{C}$ ReLU
CWS	11.71	23.42	6.17	-	50.38	<b>26.36</b>	80.41	7.43	3.70
CDN	9.50	22.49	6.73	-	50.64	28.22	80.41	-	3.72
CIB	11.36	23.63	5.59	-	48.10	28.64	4.98	-	3.62
	ReLU			ReLU			ReLU		
RWS	<b>5.42</b>			27.22			<b>3.42</b>		
RDN	6.29			27.84			3.52		
RIB	6.07			27.71			4.30		
DIFF	-0.17			+0.86			-0.20		

## 6.2.7 Complex Convolutional Residual Network

A deep convolutional residual network of the nature presented in He et al. (2016b,c) consists of 3 *stages* within which feature maps maintain the same shape. At the end of a stage, the feature maps are downsampled by a factor of 2 and the number of convolution filters are doubled. The sizes of the convolution kernels are always set to 3 x 3. Within a stage, there are several *residual blocks* which comprise 2 convolution layers each.

In the complex valued setting, the majority of the architecture remains identical to the one presented in He et al. (2016c) with a few subtle differences. Since all datasets that we work with have real-valued inputs, we present a way to learn their imaginary components to let the rest of the network operate in the complex plane. We learn the initial imaginary component of our input by performing the operations present within a single real-valued residual block

$$BN \rightarrow ReLU \rightarrow Conv \rightarrow BN \rightarrow ReLU \rightarrow Conv$$



**Table 6.2** – Classification error on CIFAR-10, CIFAR-100 and SVHN\* using different normalization strategies. NCBN, CBN and BN stand for a Naive variant of the complex batch-normalization, complex batch-normalization and regular batch normalization respectively. (R) & (C) refer to the use of the real- and complex-valued convolution respectively. The complex models use CReLU as activation. All models are constructed to have roughly 1.7M parameters. 5 out of 6 experiments using the naive variant of the complex batch normalization failed with the apparition of NaNs during training. As these experiments are already conclusive and due to limited resources, we haven’t conducted other experiments for the NCBN model. A ”-” is filled in front of an unperformed experiment.

ARCH	CIFAR-10			CIFAR-100			SVHN*		
	NCBN(C)	CBN(R)	BN(C)	NCBN(C)	CBN(R)	BN(C)	NCBN(C)	CBN(R)	BN(C)
WS	-	<b>5.47</b>	6.32	27.29	<b>26.63</b>	27.89	NAN	3.80	<b>3.52</b>
DN	-	5.89	6.71	NAN	27.13	28.83	NAN	3.54	3.58
IB	-	5.66	6.83	NAN	26.99	29.89	NAN	3.74	3.56

Using this learning block yielded better empirical results than assuming that the input image has a null imaginary part. The parameters of this real-valued residual block are trained by backpropagating errors from the task specific loss function. Secondly, we perform a  $Conv \rightarrow BN \rightarrow Activation$  operation on the obtained complex input before feeding it to the first residual block. We also perform the same operation on the real-valued network input instead of  $Conv \rightarrow Maxpooling$  as in He et al. (2016c). Inside, residual blocks, we subtly alter the way in which we perform a projection at the end of a stage in our network. We concatenate the output of the last residual block with the output of a 1x1 convolution applied on it with the same number of filters used throughout the stage and subsample by a factor of 2. In contrast, He et al. (2016c) perform a similar 1x1 convolution with twice the number of feature filters in the current stage to both downsample the feature maps spatially and double them in number.

In this section, we present empirical results from using our model to perform image, music classification and spectrum prediction. First, we present our model’s architecture followed by the results we obtained on CIFAR-10, CIFAR-100, and SVHN\* as well as the results on automatic music transcription on the MusicNet benchmark and speech spectrum prediction on TIMIT.

---

## 6.3 Experiments

We conduct the experiments on different tasks with feed-forward complex networks and recurrent complex networks as follows.

### 6.3.1 Image Recognition

We adopt an architecture inspired by [He et al. \(2016c\)](#). The latter will also serve as a baseline to compare against. We train comparable real-valued Neural Networks using the standard ReLU activation function. We have tested our complex models with the CReLU,  $z$ ReLU and modRelu activation functions. We use a cross entropy loss for both real and complex models. A global average pooling layer followed by a single fully connected layer with a softmax function is used to classify the input as belonging to one of 10 classes in the CIFAR-10 and SVHN datasets and 100 classes for CIFAR-100.

We consider architectures that trade-off model depth (number of residual blocks per stage) and width (number of convolutional filters in each layer) given a fixed parameter budget. Specifically, we build three different models - wide and shallow (WS), deep and narrow (DN) and in-between (IB). In a model that has roughly 1.7 million parameters, our WS architecture for a complex network starts with 12 complex filters (24 real filters) per convolution layer in the initial stage and 16 residual blocks per stage. The DN architecture starts with 10 complex filters and 23 blocks per stage while the IB variant starts with 11 complex filters and 19 blocks per stage. The real-valued counterpart has also 1.7 million parameters. Its WS architecture starts with 18 real filters per convolutional layer and 14 blocks per stage. The DN architecture starts with 14 real filters and 23 blocks per stage and the IB architecture starts with 16 real filters and 18 blocks per stage.

All models (real and complex) were trained using the backpropagation algorithm with Stochastic Gradient Descent with Nesterov momentum ([Nesterov, 1983](#)) set at 0.9. We also clip the norm of our gradients to 1. We tweaked the learning rate schedule used in [He et al. \(2016c\)](#) in both the real and complex residual networks to extract small performance improvements in both. We start our learning rate at 0.01 for the first 10 epochs to warm up the training and then set it at 0.1 from epoch 10-100 and then anneal the learning rates by a factor of 10 at epochs 120 and 150. We end the training at epoch 200.

---

Table 6.1 presents our results on performing image classification on CIFAR-10, CIFAR-100. In addition, we also consider a truncated version of the Street View House Numbers (SVHN) dataset which we call SVHN\*. For computational reasons, we use the required 73,257 training images of Street View House Numbers (SVHN). We still test on all 26,032 images. For all the tasks and for both the real- and complex-valued models, The WS architecture has yielded the best performances. This is in concordance with Zagoruyko and Komodakis (2016) who observed that wider and shallower residual networks perform better than their deeper and narrower counterpart. On CIFAR-10 and SVHN\*, the real-valued representation performs slightly better than its complex counterpart. On CIFAR-100, the complex representation outperforms the real one. In general, the obtained results for both representation are quite comparable. To understand the effect of using either real or complex representation for a given task, we designed hybrid models that combine both. Table 6.2 contains the results for hybrid models. We can observe in the Table 6.2 that in cases where complex representation outperformed the real one (wide and shallow on CIFAR-100), combining a real-valued convolutional filter with a complex batch normalization improves the accuracy of the real-valued convolutional model. However, the complex-valued one is still outperforming it. In cases, where real-valued representation outperformed the complex one (wide and shallow on CIFAR-10 and SVHN\*), replacing a complex batch normalization by a regular one increased the accuracy of the complex convolutional model. Despite that replacement, the real-valued model performs better in terms of accuracy for such tasks. In general, these experiments show that the difference in efficiency between the real and complex models varies according to the dataset, to the task and to the architecture.

Ablation studies were performed in order to investigate the importance of the 2D whitening operation that occurs in the complex batch normalization. We replaced the complex batch normalization layers with a naive variant (NCBN) which, instead of left multiplying the centred unit by the inverse square root of its covariance matrix, just divides it by its complex variance. Here, this naive variant of CBN is Mimicking the regular BN by not taking into account correlation between the elements in the complex unit. The Naive variant of the Complex Batch Normalization performed very poorly; In 5 out of 6 experiments, training failed with the appearance of NaNs . By way of contrast, all 6 complex-valued Batch Normal-

---

ization experiments converged. Results are given in Table 6.2.

Another ablation study was undertaken to compare CReLU, modReLU and zReLU. Again the differences were stark: All CReLU experiments converged and outperformed both modReLU and zReLU, both which variously failed to converge or fared substantially worse. We think that modRelu didn't perform as well as CReLU due to the fact that consecutive layers in a feed-forward net do not represent time-sequential patterns, and so, they might need to drop some phase information. Results are reported in Table 6.1.

### 6.3.2 Automatic Music Transcription

In this section we present results for the automatic music transcription (AMT) task. The nature of an audio signal allows one to exploit complex operations as presented earlier in the paper. The experiments were performed on the MusicNet dataset (Thickstun et al., 2016). For computational efficiency we resampled the original input from 44.1kHz to 11kHz using the algorithm described in Smith (2002). This sampling rate is sufficient to recognize frequencies presented in the dataset while reducing computational cost dramatically. We modeled each of the 84 notes that are present in the dataset with independent sigmoids (due to the fact that notes can fire simultaneously). We initialized the bias of the last layer to the value of -5 to reflect the distribution of silent/non-silent notes. As in the baseline, we performed experiments on the raw signal and the frequency spectrum. For complex experiments with the raw signal, we considered its imaginary part equal to zero. When using the spectrum input we used its complex representation (instead of only the magnitudes, as usual for AMT) for both real and complex models. For the real model, we considered the real and imaginary components of the spectrum as separate channels. The model we used for raw signals is a shallow convolutional network similar to the model used in the baseline, with the size reduced by a factor of 4 (corresponding to the reduction of the sampling rate). The filter size was 512 samples (about 12ms) with a stride of 16. The model for the spectral input is similar to the VGG model (Simonyan and Zisserman, 2015). The first layer has filter with size of 7 and is followed by 5 convolutional layers with filters of size 3. The final convolution block is followed by a fully connected layer with 2048 units. The latter is followed, in its turn, by another fully connected layer with 84 sigmoidal

**Table 6.3** – MusicNet experiments. *FS* is the sampling rate. *Params* is the total number of parameters. We report the average precision (AP) metric that is the area under the precision-recall curve.

ARCHITECTURE	FS	PARAMS	AP, %
SHALLOW, REAL	11kHz		66.1
SHALLOW, COMPLEX	11kHz		66.0
SHALLOW, THICKSTUN ET AL. (2016)	44.1kHz	-	67.8
DEEP, REAL	11kHz	10.0M	69.6
DEEP, COMPLEX	11kHz	8.8M	<b>72.9</b>

units. In all of our experiments we use an input window of 4096 samples or its corresponding FFT (which corresponds to the 16,384 window used in the baseline) and predicted notes in the center of the window. All networks were optimized with Adam. We start our learning rate at  $10^{-3}$  for the first 10 epochs and then anneal it by a factor of 10 at each of the epochs 100, 120 and 150. We end the training at epoch 200. For the real-valued models, we have used ReLU as activation. CReLU has been used as activation for the complex-valued models.

The complex network was initialized using the unitary initialization scheme respecting the He criterion as described in Section 6.2.6. For the real-valued network, we have used the analogue initialization of the weight tensor. It consists of performing an orthogonal initialization with a gain of  $\sqrt{2}$ . The complex batch normalization was applied according to Section 6.2.5. Following Thickstun et al. (2016) we used recordings with ids '2303', '2382', '1819' as the test subset and additionally we created a validation subset using recording ids '2131', '2384', '1792', '2514', '2567', '1876' (randomly chosen from the training set). The validation subset was used for model selection and early stopping. The remaining 321 files were used for training. The results are summarized on Table 6.3. We achieve a performance comparable to the baseline with the shallow convolutional network. our VGG-based deep real-valued model reaches 69.6% average precision on the downsampled data. With significantly fewer parameters than its real counterpart, the VGG-based deep complex model, achieves 72.9% average precision which is the state of the art to the best of our knowledge.

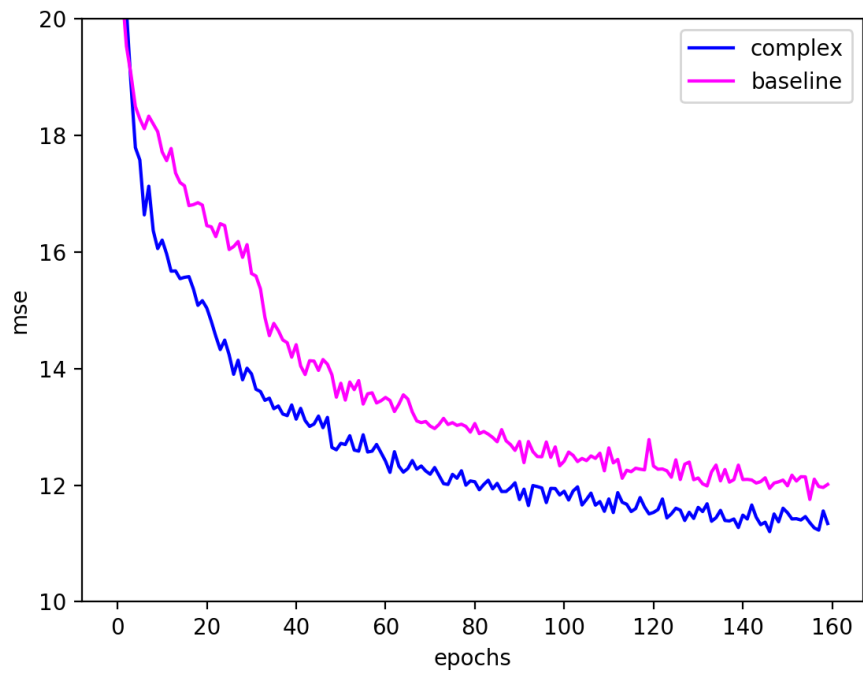
**Table 6.4** – Speech Spectrum Prediction on TIMIT test set. *CConv-LSTM* denotes the Complex Convolutional LSTM.

MODEL	MSE(VALENATION)	MSE(TESE)
LSTM WISDOM ET AL. (2016)	16.59	16.98
FULL-CAPACITY URNN WISDOM ET AL. (2016)	14.56	14.66
CONV-LSTM (OUR BASELINE)	11.10	12.18
CCONV-LSTM (OURS)	<b>10.78</b>	<b>11.90</b>

### 6.3.3 Speech Spectrum Prediction

We apply both a real Convolutional LSTM Xingjian et al. (2015) and a complex Convolutional LSTM on speech spectrum prediction task. In this task, the model predicts the magnitude spectrum. It implicitly infers the real and imaginary components of the spectrum at time  $t + 1$ , given all the spectrum (imaginary part and real components) up to time  $t$ . This is slightly different from (Wisdom et al., 2016). The real and imaginary components are considered as separate channels in both model. We evaluate the model with mean-square-error (MSE) on log-magnitude to compare with the others Wisdom et al. (2016). The experiments are conducted on a downsampled (8kHz) version of the TIMIT dataset. By following the steps in Wisdom et al. (2016), raw audio waves are transformed into frequency domain via short-time Fourier transform (STFT) with a Hann analysis window of 256 samples and a window hop of 128 samples (50% overlap). We use a training set with 3690 utterances, a validation set with 400 utterances and a standard test set with 192 utterance.

To match the number of parameters for both model, the Convolutional LSTM has 84 feature maps while the complex model has 60 complex feature maps (120 feature maps in total). Adam Kingma and Ba (2014) with a fixed learning rate of  $1e-4$  is used in both experiments. We initialize the complex model with the unitary initialization scheme and the real model with orthogonal initialization respecting the Glorot criterion. The result is shown in Table 6.4 and the learning curve is shown in Figure 6.2. Our baseline model has achieved the state of the art and the complex convolutional LSTM model performs better over the baseline in terms of MSE and convergence.



**Figure 6.2** – *Learning curve for speech spectrum prediction from dev set.*

# 7

## Conclusion

This thesis presented two research paper related to sequence-to-sequence modelling on speech. The first paper introduced a new approach to address speech recognition with convolutional neural networks and the second one presented a new model taking advantage of the representation in the complex domain, which could benefit the speech tasks where the phase is critical. We summarize the work in the following:

- In the first paper, we present a CNN-based end-to-end speech recognition framework without recurrent neural networks which are widely used in speech recognition tasks. We show promising results on the TIMIT dataset and conclude that the model has the capacity to learn the temporal relations that are required for it to be integrated with CTC. We already observed a gain in computational efficiency on the TIMIT dataset and training the model on large vocabulary datasets and integrate with the language model would be a part of our further study. Another interesting direction is to apply Batch Normalization [Ioffe and Szegedy \(2015\)](#) to the current model.
- In the second paper, We present key building blocks required to train complex valued neural networks, such as complex batch normalization and complex weight initialization. We have also explored a wide variety of complex convolutional network architectures, including some yielding competitive results for image classification and state of the art results for a music transcription task and speech spectrum prediction. We hope that our work will stimulate further investigation of complex valued networks for deep learning models and their application to more challenging tasks such as generative models for audio and images.

We hope to combine the convolutional sequence modelling and complex representation in speech recognition and speech synthesis domain as a future work.





# Bibliography

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.
- Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., and Penn, G. (2012). Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280. IEEE.
- Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128.
- Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4945–4949. IEEE.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.
- Bourlard, H. and Morgan, N. (1990). A continuous speech recognition system embedding mlp into hmm. In *Advances in neural information processing systems*, pages 186–193.
- Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference*

- 
- on *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585.
- Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.
- Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L., and Batra, D. (2015). Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.
- Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N., and Graves, A. (2016). Associative long short-term memory. *arXiv preprint arXiv:1602.03032*.
- Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941.
- Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., and Pallett, D. S. (1993). Darpa timit acoustic-phonetic continuous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon Technical Report N*, 93.
- Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

- 
- Graves, A. (2012a). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.
- Graves, A. (2012b). Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer.
- Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.
- Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772.
- Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 IEEE international conference on*, pages 6645–6649. IEEE.
- Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610.
- Guberman, N. (2016). On complex valued convolutional neural networks. *arXiv preprint arXiv:1602.09046*.
- Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.
- He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

- 
- He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016c). Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer.
- He, P., Huang, W., Qiao, Y., Loy, C. C., and Tang, X. (2016d). Reading scene text in deep convolutional sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3501–3508.
- Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97.
- Hirose, A. and Yoshida, S. (2012). Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence. *IEEE Transactions on Neural Networks and learning systems*, 23(4):541–551.
- Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Master’s thesis, Institut fur Informatik, Technische Universitat, Munchen*.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. (2017). How to escape saddle points efficiently. *arXiv preprint arXiv:1703.00887*.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

- 
- Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Lee, C.-Y. and Osindero, S. (2016). Recursive recurrent nets with attention modeling for ocr in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2231–2239.
- Lopes, C. and Perdigao, F. (2011). Phoneme recognition on the timit database. In *Speech Technologies*. InTech.
- Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.
- Miao, Y., Gowayyed, M., and Metze, F. (2015). Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 167–174. IEEE.
- Miao, Y., Metze, F., and Rawat, S. (2013). Deep maxout networks for low-resource speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 398–403. IEEE.
- Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.
- Mohamed, A.-r., Dahl, G. E., and Hinton, G. (2012). Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14–22.
- Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ . In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547.
- Nitta, T. (2002). On the critical points of the complex-valued neural network. In *Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on*, volume 3, pages 1099–1103. IEEE.

- 
- Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.
- Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.
- Sainath, T. N., Kingsbury, B., Mohamed, A.-r., Dahl, G. E., Saon, G., Soltau, H., Beran, T., Aravkin, A. Y., and Ramabhadran, B. (2013a). Improvements to deep convolutional neural networks for lvcsr. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 315–320. IEEE.
- Sainath, T. N., Mohamed, A.-r., Kingsbury, B., and Ramabhadran, B. (2013b). Deep convolutional neural networks for lvcsr. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8614–8618. IEEE.
- Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.
- Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.
- Saroff, A. M., Shepardson, V., and Casey, M. A. (2015). Learning representations using complex-valued nets. *arXiv preprint arXiv:1511.06351*.
- Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

- 
- Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*.
- Smith, J. O. (2002). Digital audio resampling. Online <http://www-ccrma.stanford.edu/~jos/resample>.
- Song, W. and Cai, J. End-to-end deep neural network for automatic speech recognition. *Technical Report*.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.
- Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385.
- Sundermeyer, M., Schlüter, R., and Ney, H. (2012). Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.
- Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.
- Thickstun, J., Harchaoui, Z., and Kakade, S. (2016). Learning features of music from scratch. In *Proceedings of the International Conference on Learning Representations*.
- Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2015). Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656.

- 
- Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015). Sequence to sequence-video to text. In *Proceedings of the IEEE international conference on computer vision*, pages 4534–4542.
- Vinyals, O., Ravuri, S. V., and Povey, D. (2012). Revisiting recurrent neural networks for robust asr. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4085–4088. IEEE.
- Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. (2016). Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888.
- Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. R. (2016). On multiplicative integration with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 2856–2864.
- Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810.
- Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. In *Proceedings of the International Conference on Learning Representations*.
- Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *Proceedings of the British Machine Vision Conference*.