Université de Montréal

# Balancing Signals for Semi-Supervised Sequence Learning

par

**Olga(Ge Ya) Xu**

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures et postdoctorales

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en Informatique

février 2020

**Université de Montréal**
Faculté des études supérieures et postdoctorales

Ce mémoire intitulé

# Balancing Signals for Semi-Supervised Sequence Learning

présenté par

**Olga(Ge Ya) Xu**

a été évalué par un jury composé des personnes suivantes :

*Yoshua Bengio*
(président-rapporteur)

*Christopher Pal*
(directeur de recherche)

*Aaron Courville*
(membre du jury)

Mémoire accepté le :
*August 30, 2019*

# Sommaire

Les réseaux neuronaux récurrents (RNN) sont des modèles puissants qui ont obtenu des réalisations exceptionnelles dans de nombreuses tâches d'apprentissage séquentiel. Malgré leurs réalisations, les modèles RNN souffrent encore de longues séquences pendant l'entraînement. C'est parce que l'erreur se propage en arrière de la sortie vers les couches d'entrée transportant des signaux de gradient, et avec une longue séquence d'entrée, des problèmes comme la disparition et l'explosion des gradients peuvent survenir. Cette thèse passe en revue de nombreuses études actuelles et architectures existantes conçues pour contourner les problèmes de dépendance à long terme de la rétropropagation dans le temps (BPTT).

Nous nous concentrons principalement sur la méthode proposée par  cite Trinh2018 qui utilise une méthode d'apprentissage semi-supervisée pour atténuer les problèmes de dépendance à long terme dans BPTT. Malgré les bons résultats obtenus avec le modèle de cite Trinh2018, nous suggérons que le modèle peut être encore amélioré avec une manière plus systématique d'équilibrer les signaux auxiliaires. Dans cette thèse, nous présentons notre article -  emph RNNs with Private and Shared Representations for Semi-Supervised Learning - qui est actuellement en cours de révision pour AAAI-2019. Nous proposons une architecture RNN semi-supervisée avec des représentations privées et partagées explicitement conçues qui régule le flux de gradient de la tâche auxiliaire à la tâche principale.

**Keywords:** Apprentissage Automatique; Apprentissage Semi-supervisé; Formation Auxiliaire; Sequence Learning

# Summary

Recurrent Neural Networks(RNNs) are powerful models that have obtained outstanding achievements in many sequence learning tasks. Despite their accomplishments, RNN models still suffer with long sequences during training. It is because error propagate backwards from output to input layers carrying gradient signals, and with long input sequence, issues like vanishing and exploding gradients can arise. This thesis reviews many current studies and existing architectures designed to circumvent the long-term dependency problems in backpropagation through time (BPTT).

Mainly, we focus on the method proposed by Trinh et al. (2018) which uses semi-supervised learning method to alleviate the long-term dependency problems in BPTT. Despite the good results Trinh et al. (2018)'s model achieved, we suggest that the model can be further improved with a more systematic way of balancing auxiliary signals. In this thesis, we present our paper – *RNNs with Private and Shared Representations for Semi-Supervised Learning* – which is currently under review for AAAI-2019. We propose a semi-supervised RNN architecture with explicitly designed private and shared representations that regulates the gradient flow from auxiliary task to main task.

**Keywords:** Apprentissage Automatique; Apprentissage Semi-supervisé; Formation Auxiliaire; Sequence Learning

# Contents

# List of Tables

# List of Figures

# Acknowledgement

Our unpublished work is presented in this thesis – *RNNs with Private and Shared Representations for Semi-Supervised Learning.* The completion of this undertaking would not have been possible without the participation and support of my friends – **Jie Fu**, **Pengfei Liu** and **Zhihao Luo**, and my supervisor – **Prof. Chris Pal**. Their contributions are fully appreciated and gratefully acknowledged. We also thank **Xiang Zhang** for helping us with our experimental data.

Special thanks to my advisor **Prof. Chris Pal** for his invaluable guidance and endless support. His enthusiasm and dedication for research have always been the biggest motivation of mine. I am very thankful for the excellent example he has provided as an affable, hardworking researcher and mentor.

# Chapter 1

# Introduction

## 1.1. Background

A sequence is an ordered collection of objects. Sequential data occur everywhere in our daily life. Some examples of sequential data include gene sequencing, written text, human speech and stock prices. Sequence modelling can help us to identify genetic diseases, understand literature, generate speeches and recognize patterns in stock prices. Thus, learning sequences has immense value and great real-life applications.

Traditionally, Hidden Markov Models (HMMs) are commonly used to process sequential data. HMMs make several strong assumptions (Rabiner and Juang, 1986):

(1) The observation space, $\mathcal{V}$, is known and finite. Let $\mathcal{O} = \{o_1 \ldots o_T\}$ be the observation sequence. Every observation comes from the observation space, i.e. $\forall o_i \in O, o_i \in \mathcal{V}$.

(2) The hidden state space, $\mathcal{Q}$, is known and finite. Let $\mathcal{H} = \{h_1 \ldots h_T\}$ be the hidden state sequence. Every hidden state comes from the hidden state space, i.e. $\forall h_i \in \mathcal{H}, h_i \in \mathcal{Q}$.

(3) **Markov Assumption**. The future states depend only on the current state:

$$P(h_i = a | h_1, \ldots, h_{i-1}) = P(h_i = a | h_{i-1}) \tag{1.1.1}$$

(4) **Independence Assumption**. The probability of an output observation, $o_i$, only depends on the immediate hidden state $h_t$:

$$P(o_t | o_1, \ldots, o_T; h_1, \ldots, h_T) = P(o_t | h_t) \tag{1.1.2}$$

However, these assumption leads to several limitations of Hidden Markov Models on sequence learning:

(1) Memory of HMM is represented in a discrete $|\mathcal{Q}|$-state multinomial. The hidden state of an HMM can only transfer $\log(|\mathcal{Q}|)$ bits of information about what it generates so far (Brown and Hinton, 2001).

(2) HMM struggles to capture long range dependencies (Bengio and Frasconi, 1995; Brown and Hinton, 2001).

Unlike HMMs, recurrent neural networks (RNNs) use distributed hidden states which allow information about the past to be stored efficiently (Hinton, 2013). Several modifications on RNNs are also made to improve their long-term memories (Hochreiter and Schmidhuber, 1997; Cho et al., 2014). In recent years, RNNs have shown great promises in many sequence learning tasks and applications (OpenAI et al., 2018; Baccouche et al., 2011; Graves et al., 2013; Malhotra1 et al., 2015).

In general, a recurrent neural network processes an input sequence $\boldsymbol{x}$ by feeding its old state vector and current input vector into a recurrence function at each time step:

$$\boldsymbol{z}_t, \boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{1.1.3}$$

here $\boldsymbol{x}_t$ is the input vector at time $t$, $\boldsymbol{h}_t$ is the hidden state vector at time $t$ and $\boldsymbol{z}_t$ is the estimated output vector at time $t$. A vanilla RNN produces outputs by the following equations:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_h) = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h) \tag{1.1.4}$$

*How to train an RNN?* Suppose we have an RNN model as depicted in Figure-1.1 (Chen, 2016):

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h) \tag{1.1.5}$$

$$\boldsymbol{z}_t = \mathrm{softmax}(\boldsymbol{W}_{hz}\boldsymbol{h}_t + \boldsymbol{b}_z) \tag{1.1.6}$$

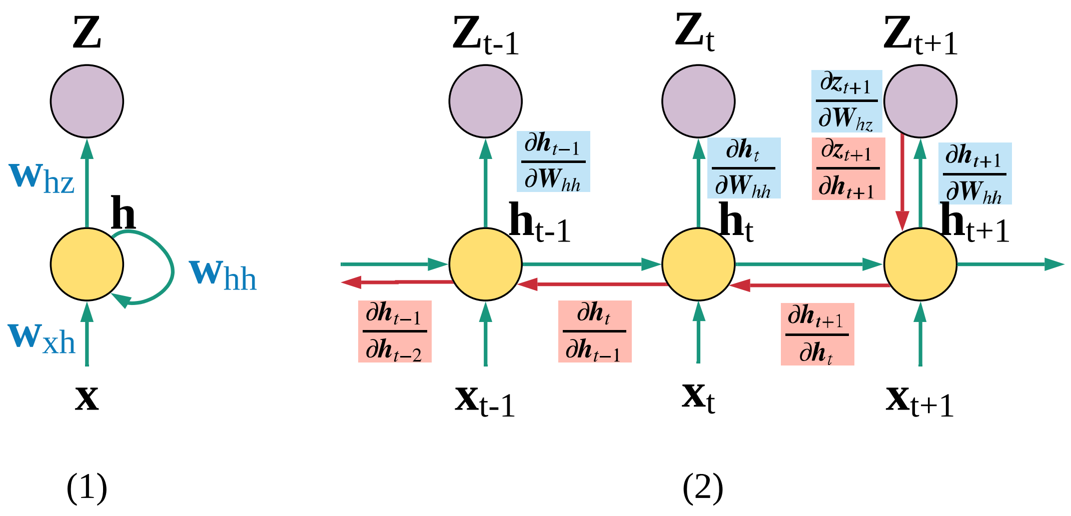
Z
$W_{hz}$
h
$W_{hh}$
$W_{xh}$
x
$Z_{t-1}$
$Z_t$
$Z_{t+1}$
$h_{t-1}$
$h_t$
$h_{t+1}$
$x_{t-1}$
$x_t$
$x_{t+1}$
$\frac{\partial h_{t-1}}{\partial W_{hh}}$
$\frac{\partial h_t}{\partial W_{hh}}$
$\frac{\partial z_{t+1}}{\partial W_{hz}}$
$\frac{\partial z_{t+1}}{\partial h_{t+1}}$
$\frac{\partial h_{t+1}}{\partial W_{hh}}$
$\frac{\partial h_{t-1}}{\partial h_{t-2}}$
$\frac{\partial h_t}{\partial h_{t-1}}$
$\frac{\partial h_{t+1}}{\partial h_t}$
(1)
(2)

**Figure 1.1.** This is an example of RNN: (1) On the left is the rolled up, recursive description of the RNN. (2) On the right is the unrolled version of the RNN; the red arrow in the diagram shows how the loss signal flows from $\boldsymbol{z}_{t+1}$ using backpropagation through time.

Furthermore, let's suppose the final goal of the RNN is to map input sequence $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_T\}$ to its corresponding target sequence $\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_T\}$. The objective for this model is to maximize the likelihood of the data over; namely, we want to find a set of parameters that minimize the negative log likelihood:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \arg\min_{\theta} - \sum_{t}^{T} \boldsymbol{y}_t \log \boldsymbol{z}_t \tag{1.1.7}$$

The derivative of $\mathcal{L}$ with respect to $\boldsymbol{z}_t$ is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t} = -(\boldsymbol{y}_t - \boldsymbol{z}_t) \tag{1.1.8}$$

Tracing the gradient flow in Figure-1.1. and using chain rule, we can also derive the following derivatives (check Chen (2016)'s note for detailed derivation):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{hz}} = \sum_{t}^{T} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{W}_{hz}} \tag{1.1.9}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{hh}} = \sum_{t}^{T} \sum_{i}^{t} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_i} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_{hh}} \tag{1.1.10}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{xh}} = \sum_{t}^{T} \sum_{i}^{t} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_t} \frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_i} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_{xh}} \tag{1.1.11}$$

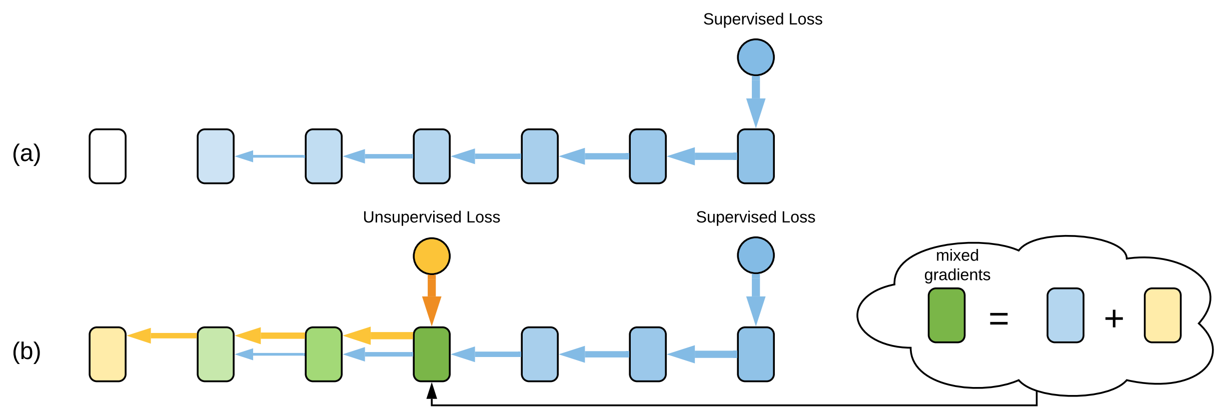

**Figure 1.2.** Schematic about gradient flows of different architectures during the backward pass. Each cube indicates a set of hidden states at a time-step. Shading means the signal strength of gradients – a darker color is associated with more reliable gradients. Arrows indicate gradient flows, and those thicker ones represent more reliable gradients. Blue filling indicates gradients w.r.t. the supervised loss. Yellow filling indicates gradients w.r.t. the unsupervised loss. Green filling represents a mixture of the supervised and unsupervised gradients. White filling means there is no gradient flowing through the hidden states. (a) Gradient flows of a typical RNN with a supervised loss. (b) Gradient flows of a semi-supervised RNN (Trinh et al., 2018).

The technique presented, unfolding an RNN through time and using backpropagation algorithm to find its gradient, is called backpropagation through time (BPTT)(Rumelhart et al., 1986; Werbos, 1990; Chen, 2016).

Note that the term $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_i}$ in Equation-1.1.10 and Equation-1.1.11 represents the chain rule over the sequence $\{\boldsymbol{h}_i, \ldots, \boldsymbol{h}_t\}$. As the gap between $i$ and $t$ becomes larger, the term $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_i}$ would become smaller (each partial derivative term within the chain rule is a matrix with small values). Thus, gradient value would shrink over time and eventually vanish as shown in Figure 1.2-a. This causes the states that are far away from current state has little or no contribution to the parameters' update. This phenomenon is known as the vanishing gradient problem in BPTT (Bengio et al., 1994; Hochreiter et al., 2001). A way to mitigate the vanishing gradient problem is to use activation functions that condone larger derivatives. However, this will encourage the exploding gradient problem (Pascanu et al., 2012).

Long short-term memory model (LSTM) and gated recurrent unit (GRU) have been proposed to alleviate the vanishing gradient problem and the exploding gradient problem

(Hochreiter and Schmidhuber, 1997; Cho et al., 2014). These architectures use gates to regulate the flow of information in and out of the cell, and this allows gradient to flow more effectively over long sequences.

LSTM (Hochreiter and Schmidhuber, 1997) explicitly introduced cells which are responsible to decide when to read, write and modify information in memory. Specifically, LSTMs use a gating mechanism to manipulate information stored in states. Gates are like valves and learn to keep relevant information and forget irrelevant ones. Gates utilize the sigmoid activation function ($\sigma(x) = \frac{1}{1+e^{-x}}$) which outputs numbers between 0 and 1 to describe the percentage of information should be let through. There are three types of gates in LSTM: forget gate, input gate and output gate. The forget gate, as its name suggests, throws away the old memory if the gate is closed. The gate's status depends on the current input and previous hidden state:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f). \tag{1.1.12}$$

To update the cell state, we also have the input gate. The input gate decides what to update depending on the current input and previous hidden states:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i). \tag{1.1.13}$$

Meanwhile, we use the *tanh* function to generate a new memory candidate, $\tilde{C_t}$:

$$\tilde{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C). \tag{1.1.14}$$

The input gate's output will be element-wise multiplied by the new memory candidate and will be added to the old memory to form a new memory:

$$C_t = i_t \odot \tilde{C_t} + f_t \odot C_{t-1}. \tag{1.1.15}$$

Lastly, the output gate is used to generate the cell's output:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \tag{1.1.16}$$

$$h_t = o_t \odot \tanh(C_t). \tag{1.1.17}$$

Overall, the memory cells of LSTM can maintain information in memory for long periods of time. Thus, this architecture is more effective in learning long-term dependencies than standard RNN.

Unlike LSTM, GRU (Cho et al., 2014) uses only the hidden states to transfer information. It is composed of two gates: update gate and reset gate. Similar to the three gates in LSTM, these two gates decide what information to keep and what information to forget:

$$u_t = \sigma(W_i \odot [h_{t-1}, x_t]), \tag{1.1.18}$$

$$r_t = \sigma(W_r \odot [h_{t-1}, x_t]), \tag{1.1.19}$$

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tanh(W \odot [r_t \odot h_{t-1}, x_t]). \tag{1.1.20}$$

In general, the GRU has less parameters than LSTM making it more computationally efficient.

Other variants of RNN, including iRNN and uRNN, also promote long term dependencies of RNN. iRNN (Le et al., 2015) suggests that RNN composed of Rectified Linear Unit (ReLU) with the correct initialization trick is good at capturing long-term dependencies. The essence of the model is in the initialization trick. The recurrent weight matrix is initialized as identity matrix with zero bias. This implies that the new hidden state vector is identical to the sum of previous hidden state vector and positive input values. For this reason, during BPTT, the error derivatives for the hidden units remain constant and no extra error-derivatives are added. In contrast, unitary evolution recurrent neural network (Arjovsky et al., 2015), also known as uRNN, learns a unitary matrix with eigenvalues of 1 to encourage the learning of long-term dependencies. It is assessed that learning becomes difficult and vanishing/exploding gradient problems start to occur when the eigenvalues of the weight matrix deviate from 1. uRNN enforces the hidden to hidden matrix to: (1) be unitary which can be decomposed into a set of structured unitary transform matrices with number of parameters $\mathcal{O}(n)$; (2) have eigenvalues of absolute value 1. Overall, uRNN offers an elegant solution to reduce memory usage while enhancing long-term dependencies.

Despite recent advances in RNN architectures, capturing long-term dependencies on long sequences using BPTT remains a fundamental challenge. RNNs with deep transition functions are hard to train, even with modern architectures like LSTM, GRU, iRNN and uRNN.

Moreover, Sejnowski and Rosenberg (1987); Bai et al. (2018); Kalchbrenner et al. (2014) and many others have proposed the idea of using convolutional neural network (CNN) to tackle sequence modeling tasks. In a more recent work, Bai et al. (2018) have reported that a simple generic temporal convolutional network (TCN) can have effective long-term memory and outperform some RNN variants on several commonly used sequence modeling tasks. TCN combines 1D fully convolutional network (FCN) with causal convolution to produce output for sequence modeling. It also uses modern convolutional architecture's techniques – dilated convolution (van den Oord et al., 2016) and residual connections (He et al., 2015) – to reduce size of the network and enhance the network's memory. However, TCNs exhibit several disadvantages comparing to regular RNNs: (1) RNNs require less data storage compared to TCNs because RNNs can discard the observed sequence and store the entire history in a fixed length vector $h_t$ (Bai et al., 2018); (2) TCNs may perform poorly transferring a model from one domain to another (Bai et al., 2018); (3) Trinh et al. (2018)'s work shows that RNNs with an auxiliary unsupervised loss outperform TCNs in terms of accuracy on long sequence learning tasks; (4) RNNs can model, in principle, infinitely long dependencies with a finite number of parameters.

Lately, some strong evidences in Trinh et al. (2018)'s work suggest that using semi-supervised learning techniques can enhance RNN's ability to capture long-term dependencies.

In semi-supervised learning, unsupervised tasks are often used to catalyze and boost the learning of the supervised tasks. Oftentimes, the unsupervised share of semi-supervised learning is introduced in a pre-training fashion. For instance, Radford (2018) shows that generative language model pretrained on a large unlabeled corpus followed by discriminating fine-tuning leads to a significant improvement on natural language understanding tasks. An additional example is to fix the unsupervised pretrained weights of a model and add additional task-specific capacity to the model making it possible to leverage large, rich and universal representations for downstream tasks (Peters et al., 2018).

On the other hand, Trinh et al. (2018) suggest that training supervised and unsupervised tasks concurrently can improve RNN's memory. Trinh et al. (2018) propose the idea of adding local auxiliary loss to force RNNs to reconstruct or predict sub-sequences and obtain local signals. Adding local unsupervised auxiliary losses can bring benefit to sequence learning: unsupervised loss induces regularization to the model, allowing the model to generalize very long sequences. Future works, which elaborated on Trinh et al. (2018)'s idea, conspicuously shared a common semi-supervised architecture which consists two distinct RNNs (Peters et al., 2018): a primary RNN for supervised task and a secondary RNN for auxiliary unsupervised task. More specifically, the secondary RNN is injected locally along the sequence wherever an anchor point is placed. Figure-1.2-b illustrates a simplified version of Trinh et al. (2018)'s architecture and shows how gradients flow in this architecture.

## 1.2. Motivation

Sequence learning has many applications in broad areas of research and conceivably has great resemblance to how human learns and make decisions(Robertson and Takács, 2018). Recurrent neural network has been designed to work with sequential data and is still been widely used to tackle sequential learning problems today. Furthermore, traditional RNNs perform poorly for long sequences. Later, LSTM and GRU are explicitly designed to avoid long-term dependency issues. As mentioned above, in spite of the fact that vanishing gradient problem has been partly addressed by LSTM and GRU, very long-term dependencies are yet very difficult to capture and learning such type of dependencies is still an activate area of research. Many variations of RNN have been introduced to attenuate the long-term dependency problems: Dilated RNN (Chang et al., 2017), hierarchical RNNs (El Hihi and Bengio, 1995), IndRNN (Li et al., 2018), SkipRNN (Campos et al., 2018), semi-supervised RNN with auxiliary loss (Trinh et al., 2018), etc.

Most of these previously mentioned methods endeavor to learn long-range dependencies by modifying RNN architecture. One noteworthy method is Trinh et al. (2018)'s semi-supervised learning approach which proposed to insert unsupervised auxiliary loss at random positions along the input sequence to enhance long-term dependency. It is not uncommon to apply auxiliary loss to help with optimizing the learning process. There are many examples of using auxiliary loss to promote gradient propagation in shallow layers of deep neural networks: GoogLeNet (Szegedy et al., 2014) applies auxiliary classifiers to increase gradient signals in intermediate layers that get propagate back; in PSPNet, auxiliary loss is added to the `res4b22` residual block to assist gradient flow (Zhao et al., 2016); in Zhang et al. (2016)'s work, auxiliary autoencoder is suggested to help classification network to obtain better local solution. Additionally, Trinh et al. (2018)'s method, a semi-supervised learning approach that inserts unsupervised auxiliary at random positions along the input sequence to improve long-term dependencies, is orthogonal to the other RNN variants (Chang et al., 2017; Koutník et al., 2014; Li et al., 2018; Campos et al., 2018; Le et al., 2015; Arjovsky et al., 2015) that were mentioned earlier. Hence, it can be used hand in hand with other models to improve RNN's performance. For this reason, we focus on examining the method proposed by Trinh et al. (2018) in order to make this method more effective.

We can motivate semi-supervised sequence learning from different perspectives:

(1) ***From a neurobiology point of view***, the knowledge in a neural network is stored in weights that connect artificial neurons which has vague resemblance to the synaptic weights of neuron connections in brain. R. Storrs and Kriegeskorte (2019) suggests that the recent advances in neural networks provide us insights into the frameworks of cognitive and neural processing in humans. Furthermore, R. Storrs and Kriegeskorte (2019) states that the unsupervised signal encourages the model to exploit information more efficiently because it favors the model to learn all regularities within data. He then argues that human brains are far more capable of learning from data than machines without explicit supervision. Thus, learning supervised and unsupervised tasks simultaneously is a good starting point, and this kind of learning procedure could closely resembles how perception and cognition

might be performed in our brains.

(2) ***From a pedagogy point of view***, semi-supervised sequence learning with auxiliary loss imitates how human learns in real life. Local reconstruction loss pushes the model to remember previously seen sequence. Kember (1996)'s research in cognitive science shows that memorization and understanding contribute to each other. His research also manifests the fact that memorization can strengthen one's ability to recall information. There are many other researches in the fields of cognitive science and pedagogy show that memorization and understanding can reinforce each other (Klemm, 2007; Entwistle and Au, 2001). The phenomena of "memorization improves understanding; understanding contributes to better learning" may occur in machine learning as well. If so, this could provide an authentic explanation to why unsupervised auxiliary tasks could assist the main task in learning.

On the other hand, it is believed that the ability of making prediction plays a significant role in how our brains perceive and learn (Hawkins and Blakeslee, 2004). R. Storrs and Kriegeskorte (2019) affirm that if a model is arguably accurate on estimating a certain cognitive function, then such model has the power to predict future sequence while extracting information for the main task.

Overall, existing RNN models tend to only address a single modality and task exclusively. However, our brains extensively interact with different modalities while fulfilling cognitive computation. From pedagogy and cognitive science's perspectives, a good comprehensive model should not only tackle the task of interest, but also memorize seen information and predict upcoming events.

(3) ***From a deep learning point of view***, unsupervised signal is extremely useful, and it enables the network exploit meaningful information. Unsupervised learning requires no additional data and label; thus, unsupervised auxiliary loss can be very profitable and economic. Unsupervised task benefits to both optimization and generalization process of models. Erhan et al. (2010) claim that the beneficial

generalization phenomenon of semi-supervised learning occurs because of the non-convexity of the objective function (multi-modality).

As we mentioned that the unsupervised auxiliary task can be jointly implemented with many new innovated RNN architectures, it makes this approach accessible for many RNN applications.

Nevertheless, local auxiliary loss can promote gradient propagation through long sequences. It is an effective method to encourage long-term dependency for BPTT. Ultimately, inserting auxiliary task improves generalization and BPTT of sequence learning, and it is a handy tool which can be used in a wide range of applications.

Although the aforementioned semi-supervised structures alleviate the long-term BPTT problem, the auxiliary objective of these approaches differs from the main task's objective and this can impair the training of the main task by contaminating the entire representation space with unsupervised loss gradient. Properly coordinating the supervised task and auxiliary task turns into an important yet interesting challenge in this type of semi-supervised learning tasks.

## 1.3. Thesis Statement

This thesis explores the idea of how to coordinate between supervised and unsupervised tasks for semi-supervised sequence learning. To be more specific, this thesis explores two of these methods in detail:

(1) **Weighted loss for semi-supervised learning**: multiplying unsupervised loss with a coefficient to balance the learning signals of the auxiliary task with corresponding primary task.

(2) **Private and shared representations for semi-supervised learning**: control information flow by separating the representation space. Specifically, the RNN hidden space is divided into private and shared sub-spaces where the private space is exclusive for the supervised task and the shared space is available for both tasks.

## 1.4. Thesis Outline

In Chapter-2, we explore techniques found in other literature to balance auxiliary signals. Almost all of these techniques use weight coefficient to balance auxiliary tasks and main tasks, although the specific procedure of using weight coefficient varies.

In Article-1, we present our method of balancing and regulating auxiliary gradient flow by introducing the concept of private and shared representation space. We not only describe the theoretical advantages of our method in Section-1 of the article, but also display the empirical asset of our method in Section-4 and Section-5 of the article.

# Bibliography

M. Arjovsky, A. Shah, et Y. Bengio, “Unitary evolution recurrent neural networks”, *CoRR*, vol. abs/1511.06464, 2015. En ligne: `http://arxiv.org/abs/1511.06464`

M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, et A. Baskurt, “Sequential deep learning for human action recognition”, *2nd International Workshop on Human Behavior Understanding (HBU)*, vol. A.A. Salah, B. Lepri ed., pp. 29–39, 2011.

S. Bai, J. Z. Kolter, et V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”, *arXiv:1803.01271*, 2018.

Y. Bengio, P. Simard, et P. Frasconi, “Learning long-term dependencies with gradient descent is difficult”, *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

Y. Bengio et P. Frasconi, “Diffusion of context and credit information in markovian models”, *CoRR*, vol. abs/cs/9510101, 1995. En ligne: `http://arxiv.org/abs/cs/9510101`

A. D. Brown et G. E. Hinton, “Products of hidden markovmodels”, *AISTATS*, 2001. En ligne: `http://www.cs.toronto.edu/~fritz/absps/aistats_2001.pdf`

V. Campos, B. Jou, X. G. i Nieto, J. Torres, et S.-F. Chang, “Skip RNN: Learning to skip state updates in recurrent neural networks”, dans *International Conference on Learning Representations*, 2018. En ligne: `https://openreview.net/forum?id=HkwVAXyCW`

S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. J. Witbrock, M. Hasegawa-Johnson, et T. S. Huang, “Dilated recurrent neural networks”, *CoRR*, vol. abs/1710.02224, 2017. En ligne: `http://arxiv.org/abs/1710.02224`

G. Chen, “A gentle tutorial of recurrent neural network with error backpropagation”, 2016. En ligne: `https://pdfs.semanticscholar.org/50df/ee143524751aafdb2a65a7acf50618c6a5a6.pdf`

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, et Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, *CoRR*, vol. abs/1406.1078, 2014. En ligne: `http://arxiv.org/abs/1406.1078`

S. El Hihi et Y. Bengio, “Hierarchical recurrent neural networks for long-term dependencies”, dans *Proceedings of the 8th International Conference on Neural Information Processing Systems*, série NIPS’95. Cambridge, MA, USA: MIT Press, 1995, pp. 493–499. En ligne: `http://dl.acm.org/citation.cfm?id=2998828.2998898`

N. Entwistle et C. Au, “‘memorisation with understanding’ in approaches to studying: cultural variant or response to assessment demands?” 01 2001.

D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, et S. Bengio, “Why does unsupervised pre-training help deep learning?” *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Mars 2010. En ligne: `http://dl.acm.org/citation.cfm?id=1756006.1756025`

A. Graves, A. Mohamed, et G. E. Hinton, “Speech recognition with deep recurrent neural networks”, *CoRR*, vol. abs/1303.5778, 2013. En ligne: `http://arxiv.org/abs/1303.5778`

J. Hawkins et S. Blakeslee, *On Intelligence.* New York, NY, USA: Times Books, 2004.

K. He, X. Zhang, S. Ren, et J. Sun, “Deep residual learning for image recognition”, *CoRR*, vol. abs/1512.03385, 2015. En ligne: `http://arxiv.org/abs/1512.03385`

G. E. Hinton, “Recurrent neural networks”, Lecture slides, 2013. En ligne: `https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf`

S. Hochreiter et J. Schmidhuber, “Long short-term memory”, *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: `10.1162/neco.1997.9.8.1735`. En ligne: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`

S. Hochreiter, Y. Bengio, P. Frasconi, et J. Schmidhuber, “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”, 2001.

N. Kalchbrenner, E. Grefenstette, et P. Blunsom, “A convolutional neural network for modelling sentences”, dans *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, Juin 2014, pp. 655–665. DOI: `10.3115/v1/P14-1062`. En ligne: `https://www.aclweb.org/anthology/P14-1062`

D. Kember, “The intention to both memorise and understand: Another approach to learning?” *Higher Education*, vol. 31, no. 3, pp. 341–354, 1996. En ligne: `http://www.jstor.org/stable/3447651`

W. R. Klemm, “What good is learning if you don’t remember it?” *Journal of Effective Teaching*, vol. 7, no. EJ1055665, pp. 61–73, 2007. En ligne: `https://eric.ed.gov/?id=EJ1055665`

J. Koutník, K. Greff, F. J. Gomez, et J. Schmidhuber, “A clockwork RNN”, *CoRR*, vol. abs/1402.3511, 2014. En ligne: `http://arxiv.org/abs/1402.3511`

Q. V. Le, N. Jaitly, et G. E. Hinton, “A simple way to initialize recurrent networks of rectified linear units”, *CoRR*, vol. abs/1504.00941, 2015. En ligne: `http://arxiv.org/abs/1504.00941`

S. Li, W. Li, C. Cook, C. Zhu, et Y. Gao, “Independently recurrent neural network (indrnn): Building A longer and deeper RNN”, *CoRR*, vol. abs/1803.04831, 2018. En ligne: `http://arxiv.org/abs/1803.04831`

P. Malhotra1, L. Vig2, G. Shroff, et P. Agarwal, “Long short term memory networks for anomaly detection in time series”, *ESANN*, 2015. En ligne: `https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf`

OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, et W. Zaremba, “Learning dexterous in-hand manipulation”, *CoRR*, 2018. En ligne: `http://arxiv.org/abs/1808.00177`

R. Pascanu, T. Mikolov, et Y. Bengio, “Understanding the exploding gradient problem”, *CoRR*, vol. abs/1211.5063, 2012. En ligne: `http://arxiv.org/abs/1211.5063`

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, et L. Zettlemoyer, “Deep contextualized word representations”, *arXiv preprint arXiv:1802.05365*, 2018.

K. R. Storrs et N. Kriegeskorte, “Deep learning for cognitive neuroscience”, 03 2019.

L. R. Rabiner et B. H. Juang, “An introduction to hidden markov models”, *IEEE ASSP Magazine*, vol. 3, pp. 4–16, 1986.

A. Radford, “Improving language understanding by generative pre-training”, 2018.

E. Robertson et  Takács, “Motor sequence learning”, *Scholarpedia*, vol. 13, no. 5, p. 12319, 2018, revision #186421. DOI: `10.4249/scholarpedia.12319`

D. E. Rumelhart, G. E. Hinton, et R. J. Williams, “Learning Representations by Back-propagating Errors”, *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: `10.1038/323533a0`. En ligne: `http://www.nature.com/articles/323533a0`

T. J. Sejnowski et C. R. Rosenberg, “Parallel networks that learn to pronounce english text”, *Complex Systems*, vol. 1, 1987.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, et A. Rabinovich, “Going deeper with convolutions”, *CoRR*, vol. abs/1409.4842, 2014. En ligne: `http://arxiv.org/abs/1409.4842`

T. H. Trinh, A. M. Dai, T. Luong, et Q. V. Le, “Learning longer-term dependencies in rnns with auxiliary losses”, *CoRR*, vol. abs/1803.00144, 2018. En ligne: `http://arxiv.org/abs/1803.00144`

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, et K. Kavukcuoglu, “Wavenet: A generative model for raw audio”, *CoRR*, vol. abs/1609.03499, 2016. En ligne: `http://arxiv.org/abs/1609.03499`

P. J. Werbos, “Backpropagation through time: what it does and how to do it”, *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990. DOI: `10.1109/5.58337`

Y. Zhang, K. Lee, et H. Lee, “Augmenting supervised neural networks with unsupervised objectives for large-scale image classification”, *CoRR*, vol. abs/1606.06582, 2016. En ligne: `http://arxiv.org/abs/1606.06582`

H. Zhao, J. Shi, X. Qi, X. Wang, et J. Jia, “Pyramid scene parsing network”, *CoRR*, vol. abs/1612.01105, 2016. En ligne: `http://arxiv.org/abs/1612.01105`

# Chapter 2

## Balancing Signals in Auxiliary Learning

In this chapter, we present and discuss some interesting existing methods for balancing auxiliary signals.

## 2.1. Introduction

The technique of using auxiliary tasks during learning have been popularized in the field of machine learning due to its great performance in practice: it can be used to increase gradient signals in shallow layers of very deep neural networks (DNNs) (Szegedy et al., 2014; Zhao et al., 2016; Zhang et al., 2016); it can be used to stimulate denser gradient signals that aid in task-relevant representation learning in reinforcement learning (Mirowski et al., 2016); it can be used to encourage gradient flow over long sequences for sequence learning (Trinh et al., 2018)... Let $\mathcal{T}_{\text{aux}}$ and $\mathcal{T}_{\text{main}}$ be the auxiliary and main tasks with corresponding networks $f_{\text{aux}}$ and $f_{\text{main}}$ and corresponding losses $\mathcal{L}_{\text{aux}}$ and $\mathcal{L}_{\text{main}}$. The networks $f_{\text{aux}}$ and $f_{\text{main}}$ share a subset of parameters $\theta$ (i.e. $y_{\text{aux}} = f_{\text{aux}}(\gamma_{\text{aux}}, \theta; x_{\text{aux}})$ and $y_{\text{main}} = f_{\text{main}}(\gamma_{\text{main}}, \theta; x_{\text{main}})$). The main loss, $\mathcal{L}_{\text{main}}$, is the only loss we truly care for. The intuition behind this auxiliary technique is that the shared parameters $\theta$ would be changed when we minimize $\mathcal{L}_{\text{aux}}$ and hopefully it would reduce main task's loss, $\mathcal{L}_{\text{main}}$. However, performance of the auxiliary tasks depends on how well they align with the main task.

In semi-supervised learning with auxiliary unsupervised task, the auxiliary task would normally have different objective from the main supervised task. For example, the main task is to build an AI agent to navigate in complex environment and the auxiliary task is to compute the depth map of the environment (Mirowski et al., 2016). Differences in objective functions can cause the auxiliary signals to hurt the learning of the main task. In another word, auxiliary tasks are not always helpful for building useful representations.

Balancing supervised and unsupervised task is not a trivial job. It is difficult to evaluate the degree of helpfulness of the auxiliary task and where the auxiliary task starts to hurt the main task. We want to find a proper way to balance semi-supervised signals in sequence learning. It is unclear how Trinh et al. (2018) balance semi-supervised signals during training. In the next section, we will delve into several related literature and examine their method of balancing auxiliary and main tasks' signals.

## 2.2. Related Work

In this section, we will discuss about three common techniques used in literature to balance auxiliary signals.

### 2.2.1. Balancing Signals using Constant Coefficients

As mentioned previously, auxiliary networks are not only used in sequential learning scenario to help gradient propagation. It is often used in DNN for its effectiveness in regularization and usefulness in combating the vanishing gradient problem. Regularly, the auxiliary signals are balanced using constant coefficients.

GoogLeNet (Szegedy et al., 2014) inserts auxiliary classifiers to intermediate layers. These classifiers are small CNNs applied on the Inception `4a` and Inception `4b`'s outputs. Their losses are added to the total loss of the network with a discount weight of 0.3. PSPNet (Zhao et al., 2016) also uses auxiliary loss in residual block `res4b22` (shallow layer of the network). This loss is been discounted by a factor of 0.4 when it is added to the network's total loss.

The practice of using discounted weight to balance signals can be formulated as following:

$$\hat{\gamma_{\text{aux}}}, \hat{\theta}, \hat{\gamma_{\text{main}}} := \underset{\gamma_{\text{aux}}, \theta, \gamma_{\text{main}}}{\arg\min}\ \lambda \mathcal{L}_{\text{aux}}(\,\cdot\,; \gamma_{\text{aux}}, \theta) + \mathcal{L}_{\text{main}}(\,\cdot\,; \gamma_{\text{main}}, \theta) \tag{2.2.1}$$

where $\lambda$ is the discounted weight coefficient and it is a hyper-parameter. $\gamma_{\text{main}}$ is a set of parameters used only for the main task and $\gamma_{\text{aux}}$ is a set of parameters used only for the auxiliary tasks. Using weight coefficient to balance auxiliary signals is very simple and intuitive; thus, it is the most common approach.

Averaging the semi-supervised losses is also commonly used for balancing semi-supervised tasks, and it is a special scenario of using weight coefficient to balance signals. For example, Toshniwal et al. (2017) introduce two auxiliary losses(phoneme decoder loss($\mathcal{L}_{\text{p}}$) and state-level loss($\mathcal{L}_{\text{s}}$)) to attention-enabled encoder-decoder RNNs for conversational speech recognition. They balance their total loss by averaging these losses: $\mathcal{L} = \frac{\mathcal{L}_{\text{o}}+\mathcal{L}_{\text{p}}+\mathcal{L}_{\text{s}}}{3}$ where $\mathcal{L}_{\text{o}}$ is the original loss.

### 2.2.2. Balancing Losses using Learnable Coefficients

The coefficient to balance different tasks can be added to the network's learnable parameters (Liebel and Körner, 2018). To be specific, Liebel and Körner (2018) tackle both of the single-image depth estimation (SIDE) and semantic segmentation problems by introducing multiple auxiliary tasks to them. They set up this challenge in multi-task learning setting and use the following equation to combine all the losses:

$$\mathcal{L} = \sum_{\tau \in \mathcal{T}} \lambda_\tau \mathcal{L}_\tau(\,\cdot\,; \omega_\tau) \tag{2.2.2}$$

where $\mathcal{T}$ is a set of tasks(including both main and auxiliary tasks) and $\omega_\tau = \{\theta_\tau\}$.

Instead of manually tuning $\lambda_\tau$'s, they can be added to the set of learnable parameters $\omega_\tau$ (i.e. $\omega_\tau = \{\theta_\tau, \lambda_\tau\}$). Regularization terms are also introduced to the final loss to avoid trivial solution. Liebel and Körner (2018) use a special regularization term $R(\lambda_\tau) = \ln(1 + {\lambda_\tau}^2)$ to avoid negative regularization values. By putting everything together, the final total loss is:

$$\mathcal{L} = \sum_{\tau \in \mathcal{T}} \frac{1}{2{\lambda_\tau}^2} \mathcal{L}(\,\cdot\,; \omega_\tau) + \ln(1 + {\lambda_\tau}^2) \tag{2.2.3}$$

### 2.2.3. Adapting Auxiliary Losses Using Gradient Similarity

Du et al. (2019) propose an idea which refines the balancing signals using weight coefficient methods. Du et al. (2019) suggest to regulate the weight coefficient $\lambda$ at each iteration by monitoring the helpfulness of $\mathcal{T}_{\text{aux}}$ to $\mathcal{T}_{\text{main}}$ given $\theta^{(t)}$, $\gamma_{\text{aux}}^{(t)}$ and $\gamma_{\text{main}}^{(t)}$. That is, following the same setting as Equation-2.2.1, the value of $\lambda^{(t)}$ is determined by the following at each iteration:

$$\lambda^{(t+1)} := \arg\min_{\lambda} \mathcal{L}_{\text{main}}\Big(\gamma_{\text{main}}^{(t)} - \alpha\nabla_{\gamma_{\text{main}}}\mathcal{L}_{\text{main}}, \theta^{(t)} - \alpha\nabla_{\theta}\big(\lambda\mathcal{L}_{\text{aux}} + \mathcal{L}_{\text{main}}\big), y_{\text{main}}\Big) \tag{2.2.4}$$

where $\alpha$ is the learning rate.

Du et al. (2019) point out that solving Equation-2.2.4 is expensive and offer a heuristic solution to approximate $\lambda^{(t)}$. They propose to use cosine similarity between gradients of tasks as an estimate of the weight coefficient. The update rules for the parameters would be the following:

$$\gamma_{\text{aux}}^{(t+1)} := \gamma_{\text{aux}}^{(t)} - \alpha\nabla_{\gamma_{\text{aux}}}\mathcal{L}_{\text{aux}} \tag{2.2.5}$$

$$\gamma_{\text{main}}^{(t+1)} := \gamma_{\text{main}}^{(t)} - \alpha\nabla_{\gamma_{\text{main}}}\mathcal{L}_{\text{main}} \tag{2.2.6}$$

$$\theta^{(t+1)} := \theta^{(t)} - \alpha\Big(\nabla_{\theta}\mathcal{L}_{\text{main}} + \max\big(0, \cos(\nabla_{\theta}\mathcal{L}_{\text{main}}, \nabla_{\theta}\mathcal{L}_{\text{aux}})\big)\nabla_{\theta}\mathcal{L}_{\text{aux}}\Big) \tag{2.2.7}$$

There are some resemblances between Du et al. (2019)'s technique and gradient-based meta-learning methods.

Recently, Valatka (2019) attempts to make the auxiliary loss adaptive by using Du et al. (2019)'s method. However, he does not find the approach effective and claims that it slows down the gradient calculation process significantly.

## 2.3. Discussion

Balancing auxiliary signal is a challenging yet important job. In this chapter, we discuss about the problems with auxiliary tasks and why it is important to know how to accurately balance the learning tasks. Gradients of the auxiliary task may contain worthless information for the task of interest; in the worst case scenario, they may harm or slow down the learning of the primary task.

Using weight coefficient to balance auxiliary signals is a common scheme due to its flexible nature. The three methods presented in Section-2.2 choose to use weight coefficient to balance auxiliary signals in various ways:

- **Multiplying auxiliary loss by a constant discount coefficient:** This is arguably the simplest method to balance auxiliary signal. By introducing a hyperparameter, the importance of the auxiliary task can be controlled by tuning this hypermeter up or down. The value of the hyperparameter is often determined using search algorithms such as grid search or random search. This method works well in the setting where the auxiliary task has the same interest as the primary task (i.e. adding an auxiliary classifier to shallow layers of a deep classification neural network).

- **Multiplying auxiliary loss by a learnable coefficient:** The benefits of using a learnable coefficient over a constant coefficient are: (1) the value of the coefficient can be changed during learning; (2) the network can automatically determine the value of the coefficient(no need to set up manually). The importance of the auxiliary task can change as learning advances. Thus, it is wiser to use a dynamic coefficient than a constant coefficient.

- **Using gradient similarity as coefficient:** Despite its debatable efficacy, this is still a notable method. It is the only method that the value of the coefficient is determined by the correlation of the tasks. However, it is time consuming using this technique since it needs to access and modify the gradient information every time-step during training.

There are many drawbacks of using weight coefficient to balance signals between the tasks. Using weight coefficient cannot determine what information learnt in the auxiliary task is helpful for the main task and what information is harmful. Even using the techniques from Liebel and Körner (2018) and Du et al. (2019), we cannot update using only useful information nor prevent unhelpful signal damaging the main task. Methods like them have limited control of the amount of signal passing from auxiliary task to main task, and they do not have control of the type of information that is passing.

Sadly, not much literature touches on the topic of balancing auxiliary signals in semi-supervised sequence learning setting. In the next chapter, we propose our method for coordinating semi-supervised learning tasks in RNNs. Although we focus our method only in the domain of sequence learning, our method can be easily modified to adapt in other kinds of applications of auxiliary learning.

# Bibliography

Y. Du, W. M. Czarnecki, S. M. Jayakumar, R. Pascanu, et B. Lakshminarayanan, “Adapting auxiliary losses using gradient similarity”, 2019. En ligne: `https://openreview.net/forum?id=r1gl7hC5Km`

L. Liebel et M. Körner, “Auxiliary tasks in multi-task learning”, *CoRR*, vol. abs/1805.06334, 2018. En ligne: `http://arxiv.org/abs/1805.06334`

P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, et R. Hadsell, “Learning to navigate in complex environments”, *CoRR*, vol. abs/1611.03673, 2016. En ligne: `http://arxiv.org/abs/1611.03673`

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, et A. Rabinovich, “Going deeper with convolutions”, *CoRR*, vol. abs/1409.4842, 2014. En ligne: `http://arxiv.org/abs/1409.4842`

S. Toshniwal, H. Tang, L. Lu, et K. Livescu, “Multitask learning with low-level auxiliary tasks for encoder-decoder based speech recognition”, *CoRR*, vol. abs/1704.01631, 2017. En ligne: `http://arxiv.org/abs/1704.01631`

T. H. Trinh, A. M. Dai, T. Luong, et Q. V. Le, “Learning longer-term dependencies in rnns with auxiliary losses”, *CoRR*, vol. abs/1803.00144, 2018. En ligne: `http://arxiv.org/abs/1803.00144`

L. Valatka, “Planet algorithm in a multi-agent environment”, 06 2019.

Y. Zhang, K. Lee, et H. Lee, “Augmenting supervised neural networks with unsupervised objectives for large-scale image classification”, *CoRR*, vol. abs/1606.06582, 2016. En ligne:

http://arxiv.org/abs/1606.06582

H. Zhao, J. Shi, X. Qi, X. Wang, et J. Jia, “Pyramid scene parsing network”, *CoRR*, vol. abs/1612.01105, 2016. En ligne: http://arxiv.org/abs/1612.01105

## Premier article.

# RNNs with Private and Shared Representations for Semi-Supervised Learning

par

Olga (Ge Ya) Xu[1], Jie Fu[2], Pengfei Liu[3], Zhi Hao Luo[4] et Chris Pal[5]

([1]) 2900 Edouard Montpetit Blvd, Montreal
Université de Montréal

([2]) 2900 Edouard Montpetit Blvd, Montreal
Polytechnique Montréal

([3]) 220 Handan Rd, Shanghai
Fudan University

([4]) 2900 Edouard Montpetit Blvd, Montreal
Polytechnique Montréal

([5]) 2900 Edouard Montpetit Blvd, Montreal
Polytechnique Montréal

Cet article a été soumis à la revue under review for AAAI-2020.

Les principales contributions de Olga(Ge Ya) Xu à cet article sont présentées.

- Suggested the presented idea;

- Built the model in PyTorch;
- Designed the experiments;
- Fine-tuned the model hyper-parameters and conducted the experiments.

Jie Fu(co-author 2) and I conceived the presented idea together.

Pengfei Liu(co-author 3) and I designed the experiments together.

Zhi Hao Luo(co-author 4) and I fine-tuned the model parameters for MNIST and CIFAR-10 experiments together.

Abstract. Training recurrent neural networks (RNNs) on long sequences using backpropagation through time (BPTT) remains a fundamental challenge. It has been shown that adding a local unsupervised loss term into the optimization objective makes the training of RNNs on long sequences more effective. While the importance of an unsupervised task can in principle be controlled by a coefficient in the objective function, the gradients with respect to the unsupervised loss term still influence all the hidden state dimensions, which might cause important information about the supervised task to be degraded or erased. Compared to existing semi-supervised sequence learning methods, this thesis focuses upon a traditionally overlooked mechanism – an architecture with explicitly designed *private and shared hidden units* designed to mitigate the detrimental influence of the auxiliary unsupervised loss over the main supervised task. We achieve this by dividing the RNN hidden space into a private space for the supervised task or a shared space for both the supervised and unsupervised tasks. We present extensive experiments with the proposed framework on several long sequence modeling benchmark datasets. Results indicate that the proposed framework can yield performance gains in RNN models where long term dependencies are notoriously challenging to deal with.

**Keywords:** Semi-supervised Learning; Sequence Learning; Auxiliary Learning

## 1. Introduction

Some semi-supervised learning techniques are utilized to improve the learning of long-term dependencies in RNNs as mentioned in Section-1.1. Simultaneously training an auxiliary task in additional to the main supervised task helps the model by inducing regularization and allowing it to generate enhanced generalization of very long sequences

(Trinh et al., 2018).

Despite the ability of these new semi-supervised architectures to mitigate the problem of long-distance BPTT, these approaches risk impairing the training of the main task by contaminating the entire representation space with the unsupervised loss gradients.

The challenge we address here is how to properly coordinate supervised and unsupervised tasks. In Section-2.2, we have briefly discussed about three approaches of using weighted coefficients to balance supervised and unsupervised losses. One of these methods keeps the weighted coefficients constant during the entire training process; one of these methods turns the weight coefficient into a learnable parameter; one of these methods adapts the weight coefficient using a cosine similarity metric to measure the resemblance between the unsupervised and supervised tasks. However, all of these methods cannot radically address the aforementioned problem since representations for supervised and unsupervised learning are still entangled in the same space. It is true that the contribution of the unsupervised task can in principle be controlled by a coefficient in the objective function, but the gradients with respect to the unsupervised loss term still influence all the hidden state dimensions, which might cause important information about the supervised task to be erased accidentally. Another approach is to coordinate these two types of learning by specifying a training order and separating them into different learning phases. This approach usually pre-trains a model under unsupervised setting first, then use the model for supervised learning (Radford, 2018).

While these methods can provide rich auxiliary knowledge which are potentially useful for the main task, there is no guarantee that this asynchronous learning fashion could allow the main task to utilize the auxiliary information well. Hence, long-term dependencies are still difficult to capture. Thus, it is crucial to ask: how to cooperate the auxiliary unsupervised tasks to best serve the main supervised learning task for long sequence learning?

RNN's representation space can be divided into various groups/subspaces, and different groups/subspaces can be employed to perform different tasks. This concept can be useful for modeling long-term dependencies. An example is the hierarchical RNNs (El Hihi and Bengio, 1995; Koutník et al., 2014) where each group is responsible for a subset of hidden states and each processes input at different clock speeds. It is also possible to let each layer represent a group, and each group may run at different time scales (Schmidhuber, 1992; Chung et al., 2016). Yet another inspiration comes from the shared-private feature spaces introduced in (Liu et al., 2017). However, it is only studied in the context of supervised learning tasks.

With the above analysis in mind, we propose to solve the long-term dependency problem by enabling the two RNNs to have a shared feature space for both supervised and unsupervised tasks and a private space dedicated to the supervised task. The key insight is to associate different time-scale updating operations of distinct RNNs with different representational spaces. The RNNs can only exchange information through the shared feature space, and the private space is kept privately without any interference from the unsupervised task. As a byproduct, the proposed variant of RNNs trains and evaluates slightly faster than Trinh et al. (2018)'s since the architecture by design introduced an inductive bias that the modules for auxiliary tasks should have less parameters. Figure-3.1-(c) shows how the gradients flow through the hidden states during the backward pass of BPTT for the proposed architecture. It is clear that the lower (blue) space is not allowed to receive gradients from the unsupervised task.

## 2. Related Work

Trinh et al. (2018) propose RNN-AE (RNN AutoEncoder) to form an auxiliary unsupervised task to aid RNNs in handling long sequences, i.e. r-RNN (reconstruction) and p-RNN (prediction). The r-RNN approach tries to reconstruct the original input from the internal representation. On the other hand, the p-RNN approach tries to predict the future input from the internal representation.

Skim-RNN (Seo et al., 2017), inspired by the principles of speed reading, dynamically decides on whether to "skim" or "full-read" an input at each time-step by determining the

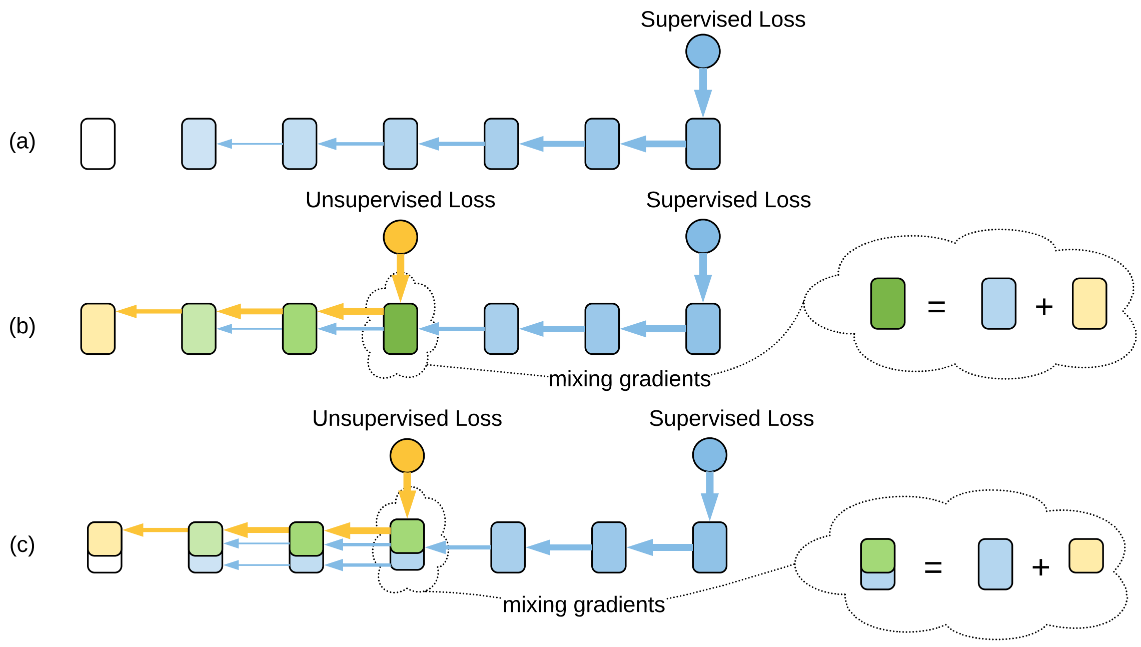

**Figure 3.1.** Similar as Figure-1.2. Schematic about gradient flows of different architectures during the backward pass. Each cube indicates a set of hidden states at a time-step. Shading means the signal strength of gradients – a darker color is associated with more reliable gradients. Arrows indicate gradient flows, and those thicker ones represent more reliable gradients. Blue filling indicates gradients w.r.t. the supervised loss. Yellow filling indicates gradients w.r.t. the unsupervised loss. Green filling represents a mixture of the supervised and unsupervised gradients. White filling means there is no gradient flowing through the hidden states. (a) Gradient flows of a typical RNN with a supervised loss. (b) Gradient flows of a semi-supervised RNN Trinh et al. (2018). (c) Gradient flows of our proposed method, which consists of a shared and private spaces.

significance of that input. Particularly, Skim-RNN(Seo et al., 2017) only updates the entire hidden state with the default RNN cell if the decision on current input token is "full-read" and updates partial hidden state with a smaller RNN if the decision on current input token is "skim", whereas regular RNN sequentially "reads" input token and update the entire hidden state at each time-step. Skim-RNN(Seo et al., 2017) aims at accelerating the inference speed and models only for supervised tasks. In contrast, our method is specifically designed for long sequence learning problems with an unsupervised loss. Furthermore, a skim-RNN only uses one RNN at each time-step which is determined by a reinforcement learning agent, whereas ours always use multiple RNNs. As a side-effect of not relying on reinforcement learning algorithms, our method is easier to train in practice.

The hidden state of our proposed RNN also has multiple subsets, but they run at the same clock speed. Even more importantly, we introduce an inductive bias that different hidden sub-spaces should be responsible for different tasks.

## 3. Methodology

We will discuss these methods in detail. We first briefly explain our version of the RNN-AE and its key differences with that introduced by Trinh et al. (2018); then, we dive into our method of inducing a private-and-shared structure.

### 3.1. RNN Autoencoder

Given an input sequence $[x_1,x_2,\dots,x_n]$ and its corresponding state vectors $[\boldsymbol{h}_1,\boldsymbol{h}_2,\dots,\boldsymbol{h}_n]$ generated by running the sequence through an RNN, we define an unsupervised loss in addition to the supervised loss.

As shown in Figure 3.1-(b) and 3.1-(c), we add the unsupervised tasks of local sequence reconstruction and prediction at various anchor points within each input sequence . We sample $m$ anchors at locations $a_1, a_2, ..., a_m$ along each input sequence, and at each anchor $a_i$, we perform local reconstruction and/or prediction within a neighbourhood of $x_{a_i}$.

*How do we select the anchor positions?*

We sample anchor locations differently from Trinh et al. (2018)'s approach. Instead of randomly sampling the anchor locations over the entire sequence, we evenly divide the input into $m$ sub-sequences, where $m$ is the number of anchors, and only sample each anchor within its corresponding region.

Let's suppose we would like to have $m$ anchor positions for reconstruction/prediction and the reconstruction/prediction's window size is $l$. We first split the input sequence into

$m$ equal-sized sub-sequences from time-step $l$ to time-step $n-l$ (i.e. the $i$-th sub-sequence is $[x_{\lfloor (i-1)\frac{n-2l}{m} \rfloor + l}, \ldots, x_{\lfloor i\frac{n-2l}{m}+l \rfloor}]$: the first sub-sequence is $[x_l, \ldots, x_{\lfloor \frac{n-2l}{m}+l \rfloor}]$ and the last($m$-th) sub-sequence is $[x_{\lfloor (m-1)\frac{n-2l}{m} \rfloor + l}, \ldots, x_{n-l}]$). Later, we sample one anchor point from each sub-sequence. After all, we would obtain $m$ anchor positions.

This is to ensure that reconstruction spans most if not all of the input sequence; this way, gradient flows back to a higher percentage of the input.

*How do we implement local reconstruction?*

This is done by using an auxiliary reconstruction RNN with GRUs (a decoder network) at each anchor. Specifically, an auxiliary RNN at anchor $a_i$ is initialized by $\boldsymbol{h}_{a_i}$, and asked to reconstruct the sequence of tokens prior to $x_{a_i}$. Here, $\boldsymbol{h}_{a_i}$ is the state vector at time step $a_i$. The reconstruction step can be described as following:

$$\hat{x}_{a_i} = x_{a_i} \tag{3.1}$$

$$\hat{x}_{t-1}, \hat{\boldsymbol{h}}_{t-1} = \mathrm{RNN}_{\mathrm{rec}}(\hat{x}_t, \hat{\boldsymbol{h}}_t) \tag{3.2}$$

The reconstruction RNN is denoted as $\mathrm{RNN}_{\mathrm{rec}}$. $\hat{x}_t$ is the reconstructed input at time $t$ and $\hat{\boldsymbol{h}}_t$ is the shared hidden representation space between the unsupervised and supervised tasks at time $t$. We will discuss more about the shared and private space in Section-3.2.

Rather than reconstructing the sequence in a forward manner as in Trinh et al. (2018)'s approach, we find that backwards reconstruction works better in practice. Thus, we reconstruct local sequences backward during unsupervised reconstruction.

Lastly, we compute the reconstruction auxiliary loss using L2-norm for each anchor $i$:

$$\mathcal{L}^i_{\mathrm{rec}} = \frac{\sum_{t=a_i-1}^{a_i-l} \left(x_t - \hat{x}_t\right)^2}{l} \tag{3.3}$$

$$\mathcal{L}_{\mathrm{rec}} = \sum_i^m \mathcal{L}^i_{\mathrm{rec}} \tag{3.4}$$

Equation-3.4 generates a total unsupervised loss as the sum of auxiliary loss at each anchor location, and its gradient flows back to each anchor neighbourhood to improve long-term dependency.

*How do we implement local prediction?*

At each anchor, the reconstruction task uses the sequence prior to this anchor as the target while the prediction tasks uses the sequence after this anchor as target. In this sense, the reconstruction tasks reconstruct what has already been seen and the prediction tasks predicts what is to come.

Specifically, at anchor $a_i$, the prediction task's target is the upcoming sequence ($x_{a_i}$ to $x_{a_i+l}$) of some length l. For unsupervised prediction, we follow the similar approach listed in Equation-3.1 and Equation-3.2. We initialize $\hat{x_{a_i}}$ to be $x_{a_i}$ and define a prediction RNN that replaces the reconstruction RNN in Equation-3.2. The prediction RNN would output a prediction for the target sequence $x_{a_i+1}$ to $x_{a_i+l}$:

$$\hat{x}_{a_i} = x_{a_i} \tag{3.5}$$

$$\hat{x}_{t+1}, \hat{\boldsymbol{h}}_{t+1} = \mathrm{RNN}_{\mathrm{pred}}(\hat{x}_t, \hat{\boldsymbol{h}}_t) \tag{3.6}$$

The prediction RNN is denoted as $\mathrm{RNN}_{\mathrm{rec}}$. Similar to equations (Equation-3.3 and Equation-3.4), we use the following equations to compute the prediction loss:

$$\mathcal{L}^i_{\mathrm{pred}} = \frac{\sum_{t=a_i+1}^{a_i+l} \left(x_t - \hat{x}_t\right)^2}{l} \tag{3.7}$$

$$\mathcal{L}_{\mathrm{pred}} = \sum_i^m \mathcal{L}^i_{\mathrm{pred}} \tag{3.8}$$

Prediction loss is considered to be analogous to language model loss. The main objective for language modelling is to learn the structure of natural language through distributed representation of words. Language modelling is often used to predict the upcoming word. Instead of applying a language model over the whole input sequence, we predict local sub-sequences only.

There are two details that our method differs from Trinh et al. (2018)'s metioned previously: (1) anchor sampling and (2) backward reconstruction. Furthermore, in various task settings, we include both unsupervised prediction and reconstruction instead of using only one. and most importantly, rather than using the entire hidden state of the anchor to do reconstruction and prediction, we only use a part of the state vector for these unsupervised tasks; this is the point we expand on next.

## 3.2. Private and Shared Structure

We propose to divide the state vectors of the main RNN into task-specific chunks. To achieve this, we send only part of the hidden vector to the auxiliary RNN for reconstruction. More concretely, we take the $d$-dimension state vector $\boldsymbol{h} = [h_1,h_2,...,h_d]^\intercal$ and split it into $\boldsymbol{h}^s$ and $\boldsymbol{h}^p$ (shared and private state) where $\boldsymbol{h}^s = [h_1, h_2, ..., h_r]^\intercal$ and $\boldsymbol{h}^p = [h_{r+1}, ..., h_d]^\intercal$; i.e. $\boldsymbol{h} = [\boldsymbol{h}^{s\intercal}||\boldsymbol{h}^{p\intercal}]^\intercal$ where $[\boldsymbol{a}||\boldsymbol{b}]$ denotes the concatenation of $\boldsymbol{a}$ and $\boldsymbol{b}$.

At each anchor step, only $\boldsymbol{h}^s$ is used to initialize the auxiliary RNN for the reconstruction task, and $\boldsymbol{h}^p$ remains untouched. More precisely, we reconstruct sub-sequence $[x_{a_i-1}, x_{a_i-2}, \ldots, x_{a_i-l}]$ or predict sub-sequence $[x_{a_i+l}, x_{a_i+2}, \ldots, x_{a_i+l}]$ by initializing $\hat{\boldsymbol{h}}_{a_i}$ to be:

$$\hat{\boldsymbol{h}}_{a_i} = \boldsymbol{h}^s_{a_i} \tag{3.9}$$

Afterwards, we either use Equation-3.1 and Equation-3.2 to reconstruction, use Equation-3.5 and Equation-3.6 to predict, or both. Lastly, we use Equation-3.3, Equation-3.4, Equaiton-3.7 and Equation-3.8 to calculate total auxiliary loss.

At the final time step, we use the entire state vector $\boldsymbol{h}$ for the supervised task. In this sense, $\boldsymbol{h}^s$ is the section of state vector that is shared by the reconstruction task and the final supervised task; $\boldsymbol{h}^p$ is private in the sense that it is the section of state vector that is reserved for the supervised task.

Some internal representations learnt are more applicable for both unsupervised and supervised tasks than others. The intuition behind our approach is that we want to partition the feature space to reduce the impact of the unsupervised tasks on the supervised task. After a few iterations of training, naturally, the network learns to store features that are relevant for the unsupervised tasks in the shared space because the unsupervised tasks can only retrieve information from the shared features of current time step, $\boldsymbol{h}_t^s$. In our case, the unsupervised tasks are reconstruction and prediction. Since the selection of anchor points is arbitrary, the unsupervised RNNs might take action at random, the shared hidden states should be ready to pass information that is useful for these tasks to the unsupervised RNNs at any point in time. Consequently, the shared region learns to capture more specific features for the unsupervised tasks than the private region.

This way, we overcome the negative side effects of RNN-AE while retaining its ability to introduce gradient at all time steps. Thus we are able to facilitate the learning of long-term dependency without hindering the model's ability to perform supervised task.

## 4. Experiments

The experiments are designed to answer a key question: Since we divide hidden states into sub-spaces, compared with RNNs with a holistic hidden space, is our proposed method indeed more effective?

We code our model using PyTorch (Paszke et al., 2017) and the repository of our code can found in the Appendix-B.1. We evaluate our proposed methods on two benchmark tasks: image classification and sentiment analysis. For image classification, we used pixel-level sequential MNIST, Fashion-MNIST and CIFAR-10 as our datasets. For sentiment analysis, we use the IMDB movie reviews corpus and DBPedia ontology dataset. For the IMDB movie reviews, our tasks are done on the character level using 1-hot vectors and for DBPedia tasks we learn character level embeddings. Detailed descriptions

[1]We use one-hot character-level embedding and remove the inputs with length less than 500.

| **Dataset** | **Description** |
|---|---|
| Seq. MNIST (LeCun and Cortes, 2010) | The MNIST database contains 60K grey-scale images of handwritten digits for training. It has 10 classes in total and each image has size of $28 \times 28$. In this thesis, images in MNIST are fed into the network one pixel at a time (sequential MNIST). The network receives a 1-dimensional input vector at each time step. |
| Seq. CIFAR10 (Krizhevsky et al., 2010) | The CIFAR-10 database contains 60K $32 \times 32$ colour images of objects in 10 classes. In this thesis, images in CIFAR10 are fed into the network one pixel at a time (sequential CIFAR). The network receives a 3-dimensional input vector at each time step. |
| Seq. Fashion-MNIST (Xiao et al., 2017) | The Fashion-MNIST database contains 60K $28 \times 28$ grey-scale images of fashion items in 10 classes. In this thesis, images in Fashion-MNIST are fed into the network one pixel at a time (sequential Fashion-MNIST). The network receives a 1-dimensional input vector at each time step. |
| IMDB(Maas et al., 2011) | The IMDB database contains 50K of movie reviews. These reviews are separated into 2 classes: positive or negative review. The dataset is been preprocessed using one-hot character embedding(60d). |
| DBPedia Ontology(Zhang et al., 2015) | The DBPedia dataset contains large carefully selected Wikipeida of 14 classes. The dataset has been preprocessed in character level(100d). |

**Table 3.1.** Description of the datasets used in this thesis.

| **Dataset** | **Mean Length** | **# Classes** | **Train Set Size** | **Valid. Set Size** |
|---|---|---|---|---|
| Seq. MNIST | 784 | 10 | 50K | 10K |
| Seq. CIFAR10 | 1024 | 10 | 45K | 15K |
| Seq. Fashion-MNIST | 784 | 10 | 50K | 10K |
| IMDB | 1430[1] | 2 | 20K | 2.2K |
| DBPedia Ontology | 300 | 14 | 560K | 70K |

**Table 3.2.** Key statistics of the datasets used in this thesis.

about the datasets are given in Table-3.1 and statistics of the datasets are given in Table-3.2.

In order to compare with state-of-the-art results fairly, we re-implement and re-run all the baseline methods. For all of our experiments, including the baseline and re-implementations, we grid search our hyperparameters using a validation set to find the optimal values. For faster convergence, we adopt SGDR (Loshchilov and Hutter, 2016) as the optimizer throughout this paper. SGDR is a stochastic gradient descent optimizer with learning rate scheduler. It periodically performs restarts of SGD: during each period of restart, the learning rate is set to some value and is scheduled to decrease. We incorporate early stopping with patience of 50 epochs to avoid overfitting, which is also based on a validation set.

As shown in Table-3.4, our proposed method achieves better outcome on both MNIST and CIFAR-10 than previous competitive results. Table-3.3 provides a more comprehensive list of the experiments along with the hyperparameters used for each one. The top row(s) of each sections, the ones without parameters, are baseline results run with GRU. As the shared proportion parameter reaches 100%, our model reduce to the RNN-AE model introduced in Trinh et al. (2018), thus the rows with "100%" as the value for the "shared" hyperparameter are results for Trinh et al. (2018). As previously mentioned, the hyperparameters for these cases are also grid searched to ensure fair comparison.

In our experiments, we are able to achieve better accuracy with either less or comparable number of parameters. In the case of MNIST, we are able to do so with less than one-tenth of the parameters compared to Trinh et al. (2018).

Overall, from these tables, we can study the effect of key hyperparameters such as private-vs-shared proportion, number of anchors and reconstruction length. we see how entirely sharing the hidden space is not optimal, which leads us to ask the follow up question: how much of the space should be shared? In the following Section-5.1, we describe our analysis on this very inquiry.

| Tasks | Accuracy | Unsup. Type |  | Sharing Schema |  | Sampling Opts. |  | #Param. |
|---|---|---|---|---|---|---|---|---|
|  |  | Pred. | Rec. | Shared | Private | #Anchor | Rec Len. |  |
| **MNIST** | 98.5% |  |  | - | - | - | - | 90K |
|  | 98.8% |  | ✓ | 100% | 0% | 1 | 50 | 1M |
|  | 98.5% |  | ✓ | 100% | 0% | 20 | 30 | 1M |
|  | 98.8% |  | ✓ | 30% | 70% | 1 | 50 | 76K |
|  | **98.9%** | ✓ | ✓ | 30% | 70% | 40 | 20 | 84K |
| **F-MNIST** | 90.5% |  |  | - | - | - | - | 11K |
|  | 90.4% |  | ✓ | 100% | 0% | 5 | 32 | 12K |
|  | **90.8%** |  | ✓ | 60% | 40% | 20 | 30 | 12K |
|  | 90.4% | ✓ | ✓ | 50% | 50% | 5 | 30 | 11K |
| **CIFAR10** | 70.8% |  |  | - | - | - | - | 3.1M |
|  | 68.9% |  |  | - | - | - | - | 5.4M |
|  | 73.5% |  | ✓ | 100% | 0% | 20 | 30 | 5.4M |
|  | 75.1% |  | ✓ | 40% | 60% | 20 | 30 | 3.9M |
|  | **76.2%** | ✓ | ✓ | 60% | 40% | 10 | 64 | 4.7M |
| **IMDB** | 84.4% |  |  | - | - | - | - | 95K |
|  | 84.8% |  | ✓ | 100% | 0% | 1 | 50 | 69K |
|  | 85.1% |  | ✓ | 60% | 40% | 1 | 64 | 59K |
|  | **85.7%** |  | ✓ | 60% | 40% | 5 | 50 | 75K |
| **DBPedia** | 83.1% |  |  | - | - | - | - | 3.5M |
|  | 95.4% |  | ✓ | 100% | 0% | 20 | 15 | 6.9M |
|  | 97.1% |  | ✓ | 50% | 50% | 20 | 15 | 4.5M |
|  | **97.2%** |  | ✓ | 40% | 60% | 20 | 15 | 4.1M |

**Table 3.3.** Performances of our models on MNIST, fashion-MNIST, CIFAR-10, and IMDB datasets. Note that when the shared proportion reaches 100%, it reduces to the model proposed in Trinh et al. (2018).

## 5. Analysis

In this section, we present two crucial analyses on the private-shared structure. First, we examine how shared proportion affects our model outcomes. Then, we visualize both the private and shared hidden space learned by our model to highlight distinct features of the two.

|  | Unpermuted Sequential MNIST | Sequential CIFAR10 |
|---|---|---|
| iRNN (Le et al., 2015) | 97% | N/A |
| uRNN (Arjovsky et al., 2015) | 95.1% | N/A |
| RNN with Aux. Loss (Trinh et al., 2018) | 98.4% | 72.2% |
| RNN with private-shared space | **98.9%** | **76.2%** |

**Table 3.4.** Comparing test accuracy on unpermuted sequential MNIST and sequential CIFAR10. iRNN(Le et al., 2015) is a RNN variant that is composed with Rectified Linear Units(ReLU) and its weight matrices are initialized with identity matrix. uRNN(Arjovsky et al., 2015) is a RNN variant whose hidden to hidden weight matrix is unitary matrix with eignvalues of absolute value exactly 1.

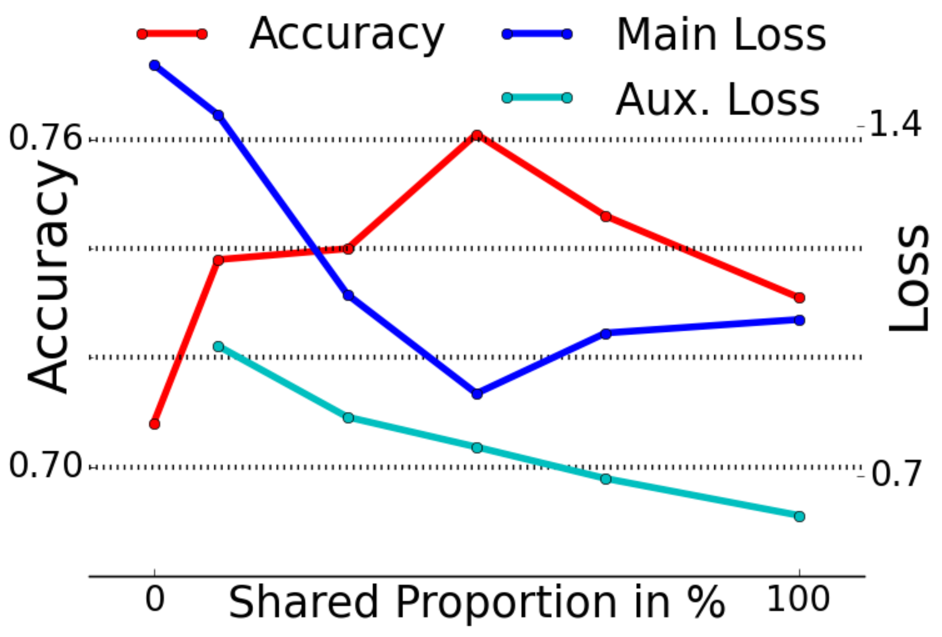

**Figure 3.2.** Accuracy (left axis) and Loss (right axis) vs Shared Proportion. We visualize the results from Table-255.6

### 5.1. Effect of Shared Proportion

To observe exactly what effect different sharing proportions have on classification tasks, we run our model on CIFAR-10 with varying share percentages from 0% to 100% as listed in Table-255.6. The results are given in Figure-3.2.

| Acc. | L Main | L Aux. | S.% | # Param. |
|---|---|---|---|---|
| 70.8 | 1.523 | - | 0% | 3.1M |
| 73.8 | 1.422 | 0.960 | 10% | 3.3M |
| 74.0 | 1.063 | 0.818 | 30% | 4M |
| **76.1** | **0.865** | 0.758 | 50% | 5.4M |
| 74.6 | 0.986 | 0.695 | 70% | 7.3M |
| 73.1 | 1.013 | **0.621** | 100% | 11M |

**Table 255.6.** Experiments on shared proportions (S.%) performed on CIFAR. Accuracy is reported in percentage. L Main and L Aux. are the main loss and auxiliary loss respectively.

With 0% shared, the model is essentially an RNN without unsupervised loss as no part of the hidden state is sent to the auxiliary RNN. In this setting, our model achieves the lowest accuracy of the group. As we slowly increase the shared percentage, the accuracy rises. From this, we confirm that the addition of auxiliary loss is indeed helpful in modeling long-term dependencies.

Remarkably, there is a pronounced peak at the 50% mark – where half of the state is shared by the auxiliary RNN and half is reserved for the classification task – after which, model performance, in terms of both cross entropy loss and classification accuracy begins to worsen. One may suspect that this is caused by overfitting since the number of parameters steadily increases. However, in this set of experiments, the capacity of the main RNN is fixed as the dimension of the hidden state remains the same. As we include more of the hidden state for unsupervised task, the auxiliary RNN increases in capacity, thus causing the overall parameter number to increase. Furthermore, the auxiliary test loss lowers progressively, suggesting that there is no overfitting in the auxiliary RNN either. In this case, the change in performance should attributed to the difference in shared proportion. This finding agrees with what we posited earlier: that the learned knowledge from the unsupervised task is not all helpful towards the supervised task, and that this information may actually hurt performance by overwriting knowledge that are important to the main task. What is somewhat surprising is that at 100% shared, the model's classification result is barely comparable to the 10% version. We did not expect such big drop in classification rate ($\sim$3%)

at the maximum shared level with comparable model capacity. However, this further corroborates the importance of disentangling the hidden space through a private-shared framework.

## 5.2. Shared and Private Spaces

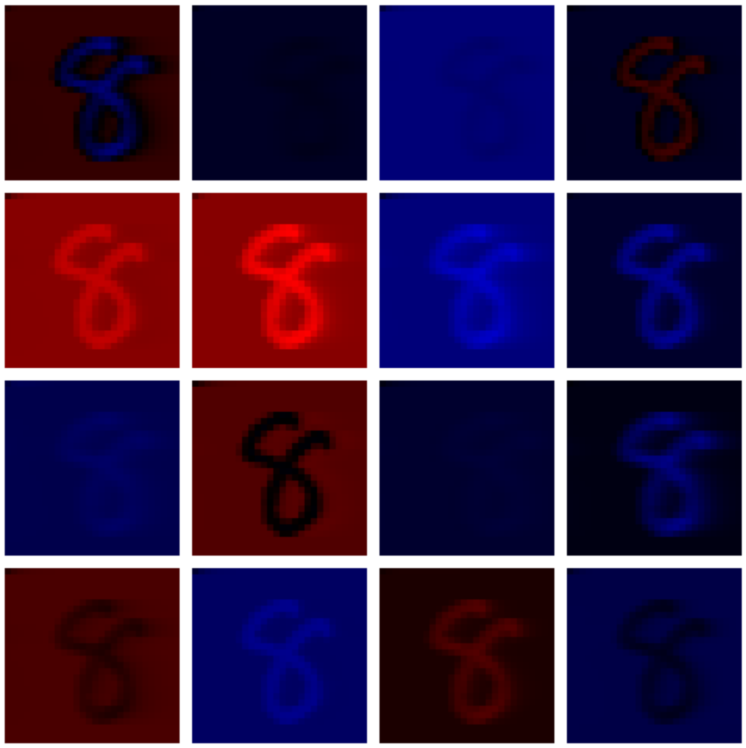

**Figure 3.3.** Hidden state response to an "8" at beginning of training

To better understand how the use of our framework influences representation learning, we analyze the hidden vectors of the main RNN. For this analysis, we retrain our MNIST model with 50% shared proportion and a hidden space size of only 16. In other words, the first 8 dimensions are shared, and the last 8 are private. We hypothesize that the small representation-space would force the RNN to learn important features; it would also allow us to see how knowledge from the two tasks might contend with each other if there existed any competition.

To visualize the hidden space, we collect the RNN's hidden state vectors $\{\boldsymbol{h}_t \in \mathbb{R}^{d\times 1} | \ 1 \leq t \leq 784\}$ after it receives an MNIST image (784 pixels) as input. We concatenate them horizontally to obtain $\boldsymbol{H} = [\boldsymbol{h}_1 || \boldsymbol{h}_2 || ... || \boldsymbol{h}_{784}] \in \mathbb{R}^{d\times 784}$. Then, we look

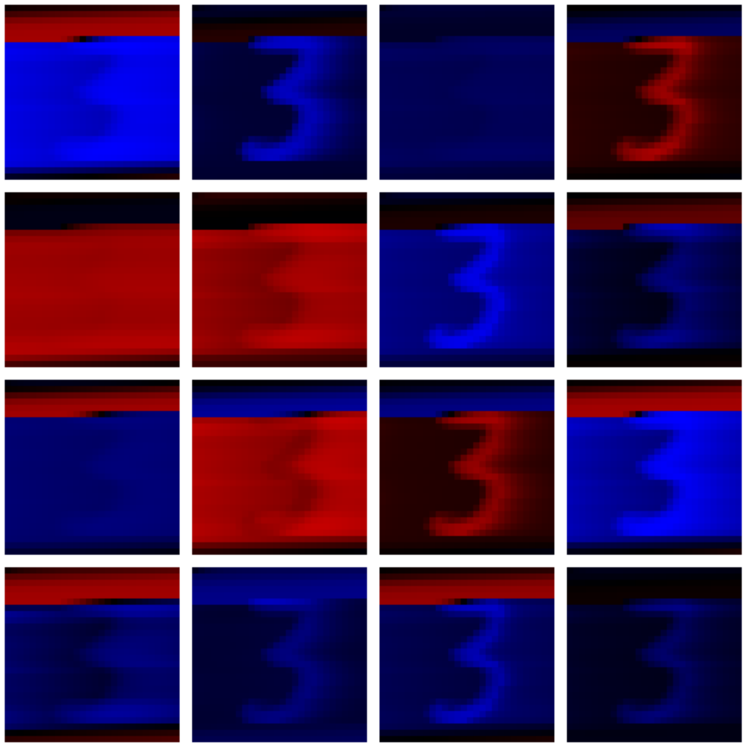

**Figure 3.4.** Hidden state response to a "3" after 5 epochs

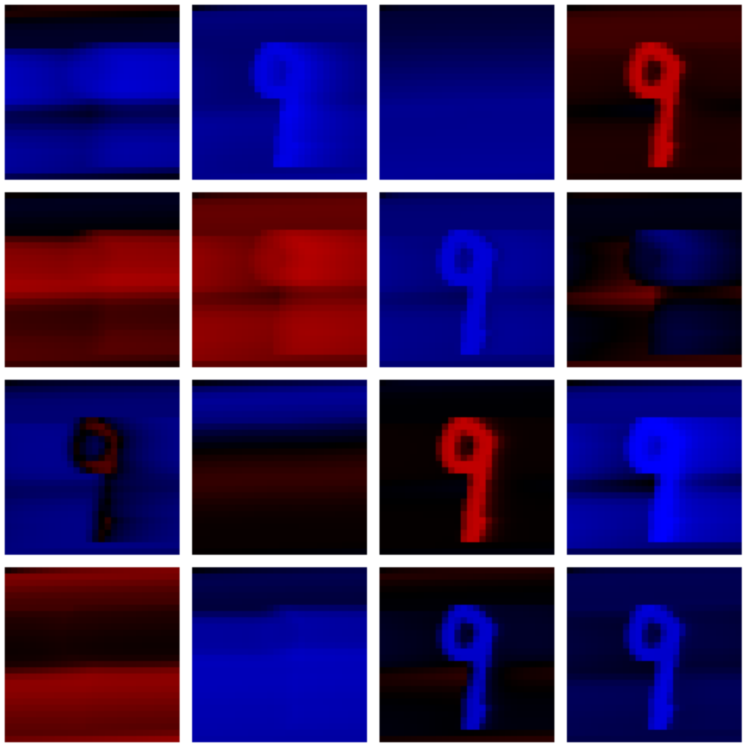

**Figure 3.5.** Hidden state response to a "9" after 25 epochs

at how the $n^{th}$ dimension of the hidden state changes across time by picking out the $n^{th}$ row of $\boldsymbol{H}$. We reshape each row into a $28 \times 28$ image to better compare the state elements.

Figure-3.3, Figure-3.4 and Figure-3.5 show three instances of visualization of the hidden state vector across time, which are generated using the same network and different input images at different stages of training. Respectively from left to right, the images are generated when training begins, progresses, and converges. Each small square in the image corresponds to a single dimension of the hidden state from $t = 1$ to $t = 784$. In this setting, the left 2 columns of each subplot contain the 8 shared dimensions and the right 2 columns corresponds to the 8 private dimensions.

Noticeably, as training progresses the hidden state changes how it responds to input. Moreover, there are dimensions in the shared region of the hidden state that originally display a response similar to those in the private region, only to be replaced later. For example, in the beginning of training, there are several similar responses in multiple dimensions of the state vector. In both the designated shared and private region, some dimensions seem to have a high correlation with the input image thus resulting in a readable number. As the model refines itself, the aforementioned response begins to fade from the shared representation; however, it still exists in the private representation. In the end, we see the a complete absence of such response in the shared representation, replaced by more abstract features.

One possible explanation would be that this type of response is more important to the supervised task than to the unsupervised one. By generating a large non-zero value when the input is non-zero, a dimension of the hidden state allows the RNN to propagate this information along to later times steps. This could be very helpful for digit classification, as we might need to know what is written in the beginning to decide which digit it is. However, it is not required for the auxiliary task which concerned with only local pixels and has less need to propagate a strong signal when a particular input is given.

# 6. Conclusion

In this thesis, we have presented a semi-supervised RNN architecture with explicitly designed private and shared hidden representations. This architecture allows information transfer between the supervised and unsupervised tasks in a hitherto unexplored way. Compared with other similar semi-supervised RNN techniques, our experiments on widely used and competitive benchmark data sets suggest that our formulation indeed yields performance gains. We conjecture that these gains come from the desirable properties of both gradient and information flow in architectures with shared and private representations. As a side-product, our proposed architecture trains and evaluates faster than the related alternatives that we have explored since the architecture introduces an inductive bias that the modules for auxiliary tasks should have fewer parameters.

# Bibliography

M. Arjovsky, A. Shah, et Y. Bengio, “Unitary evolution recurrent neural networks”, *CoRR*, vol. abs/1511.06464, 2015. En ligne: `http://arxiv.org/abs/1511.06464`

J. Chung, S. Ahn, et Y. Bengio, “Hierarchical multiscale recurrent neural networks”, *arXiv preprint arXiv:1609.01704*, 2016.

S. El Hihi et Y. Bengio, “Hierarchical recurrent neural networks for long-term dependencies”, dans *Proceedings of the 8th International Conference on Neural Information Processing Systems*, série NIPS’95. Cambridge, MA, USA: MIT Press, 1995, pp. 493–499. En ligne: `http://dl.acm.org/citation.cfm?id=2998828.2998898`

J. Koutník, K. Greff, F. J. Gomez, et J. Schmidhuber, “A clockwork RNN”, *CoRR*, vol. abs/1402.3511, 2014. En ligne: `http://arxiv.org/abs/1402.3511`

A. Krizhevsky, V. Nair, et G. Hinton, “Cifar-10 (canadian institute for advanced research)”, *University of Toronto*, 2010. En ligne: `http://www.cs.toronto.edu/~kriz/cifar.html`

Q. V. Le, N. Jaitly, et G. E. Hinton, “A simple way to initialize recurrent networks of rectified linear units”, *CoRR*, vol. abs/1504.00941, 2015. En ligne: `http://arxiv.org/abs/1504.00941`

Y. LeCun et C. Cortes, “MNIST handwritten digit database”, 2010. En ligne: `http://yann.lecun.com/exdb/mnist/`

P. Liu, X. Qiu, et X. Huang, “Adversarial multi-task learning for text classification”, *arXiv preprint arXiv:1704.05742*, 2017.

I. Loshchilov et F. Hutter, “Sgdr: stochastic gradient descent with restarts”, *arXiv preprint arXiv:1608.03983*, 2016.

A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, et C. Potts, “Learning word vectors for sentiment analysis”, dans *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies.* Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. En ligne: http://www.aclweb.org/anthology/P11-1015

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, et A. Lerer, “Automatic differentiation in PyTorch”, dans *NIPS Autodiff Workshop*, 2017.

A. Radford, “Improving language understanding by generative pre-training”, 2018.

J. Schmidhuber, “Learning complex, extended sequences using the principle of history compression”, *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992.

M. Seo, S. Min, A. Farhadi, et H. Hajishirzi, “Neural speed reading via skim-rnn”, *arXiv preprint arXiv:1711.02085*, 2017.

T. H. Trinh, A. M. Dai, T. Luong, et Q. V. Le, “Learning longer-term dependencies in rnns with auxiliary losses”, *CoRR*, vol. abs/1803.00144, 2018. En ligne: http://arxiv.org/abs/1803.00144

H. Xiao, K. Rasul, et R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”, *CoRR*, vol. abs/1708.07747, 2017. En ligne: http://arxiv.org/abs/1708.07747

X. Zhang, J. J. Zhao, et Y. LeCun, “Character-level convolutional networks for text classification”, *CoRR*, vol. abs/1509.01626, 2015. En ligne: http://arxiv.org/abs/1509.01626

# Appendix

## B.1. Code

The code for this thesis can be found here `https://tinyurl.com/yxw53h7y`.