Neural Machine Translation

Seminar Report-NLP
Arul, Ehsan
Rheinische Friedrich-Wilhelms-Universität Bonn
July 15, 2017

***Abstract*—Abstract**

I. Introduction

II. Deep Learning - An Introduction

In the recent years, Deep learning has been making a big impact in various fields of computer science. This impact is profound in the perceptual learning task in the fields like computer vision, natural language processing, etc, [10]. Though the idea of training a multi layered neural network to perform function approximation is known to the research community for at least a couple of decades[11], due to the nature of training problem requiring large computational resources, the multi layered neural network remained less practical. With the advent of general purpose graphical processing units(GPGPUs) and the availability of training data, the neural networks are now a practical solution for many perceptual tasks. One of the early success of deep learning was in the field of image classification. In the annual ImageNet classification challenge [3], AlexNet [9] showed a remarkable improvement in the state-of-the-art accuracy. Within a few years, more sophisticated architectures like Google Inception Network [13], Deep Residual Network [4], etc., has improved the accuracy to be comparable with a human in that task. The generality of the neural network made it easily possible to be used for a wide variety of tasks. With more people working on deep learning and the ideas arising in solving on the problems being easily transferable, the deep learning is the state-of-the-art for many learning tasks. In the following section, we will briefly summarize the basics of deep learning.

The basic building block of a Multilayer network, or in more technical term Multilayer perceptron(MLP) is a perceptron. The preceptron 2 closely resembles a neuron in a human brain 1. A preceptron takes a set of input values, computes a weighted sum of the inputs, and outputs a scalar value of a after applying an activation function (mostly non-linear). The weights for computing the weighted sum and the threshold in the activation function are initially set to random values and are learned from the training data. Mathematically, a perceptron with the weight matrix A and the threshold T for an step activation function, for the input vector x, outputs the following,

$$f(x) = \begin{cases} 1, & \text{if } Ax > c \\ 0, & \text{otherwise} \end{cases}$$

A layer has many number of neurons and many layers are stacked one on top of the other to form a MLP. An MLP with very many layers is called as Deep Neural Network(DNN). In the following section we will discuss the properties of DNNs and how they are useful in the context of supervised machine learning.

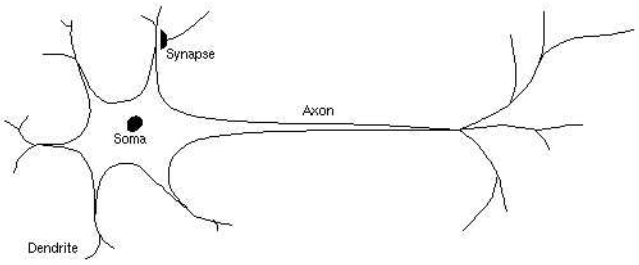


Fig. 1: A biological neuron.

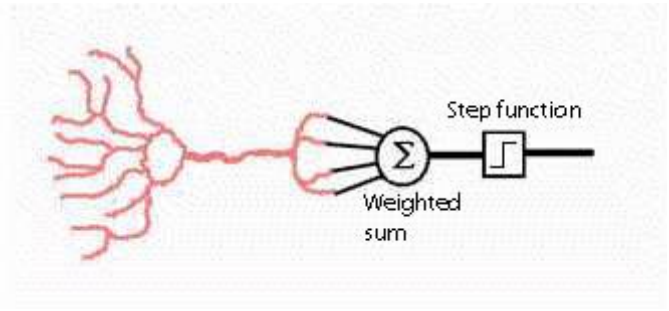


Fig. 2: An artificial neuron (perceptron)

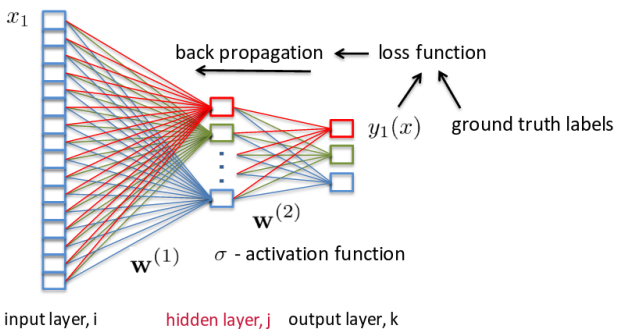


Fig. 3: Layered representation in an MLP

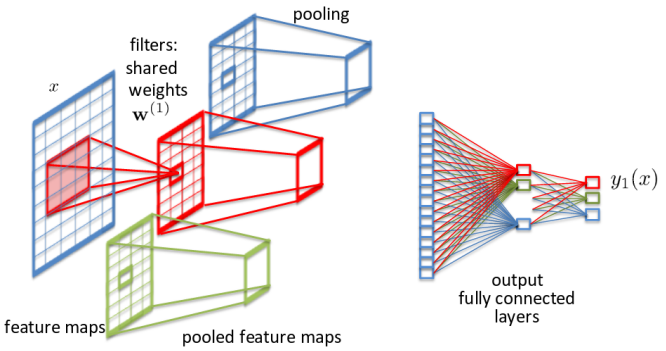


Fig. 4: CNN

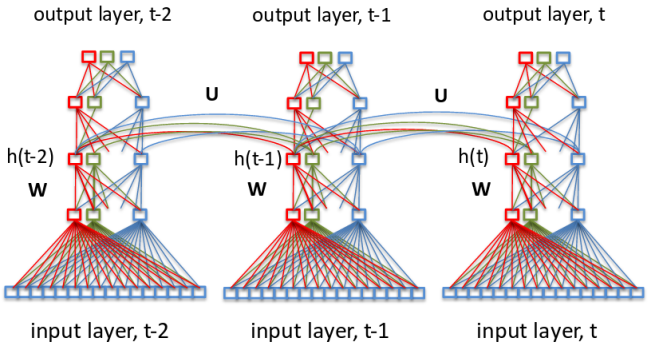


Fig. 5: Recurrent connections in a RNN

A. *Supervised learning*

Supervised learning is the problem of predicting an new output $y\prime$ for a new input $x\prime$, and set of training data that provides a set of input and output $\{x \mapsto y\}$. Most of the supervised learning problems can be formulated as either a classification or regression problem. Classification is the problem of predicting the label for a given $x\prime$ among a set of given labels $\{Y\}$ while regression of predicting a continuous valued output for a given input. The usual way of doing supervised learning is to extract features from the given data and train a model to perform classification or regression.

$$x \xrightarrow[\text{Feature extraction}]{\mathbb{D}(x)} x\prime_{intermediate} \xrightarrow[\text{classifer/regressor}]{\mathbb{F}(x\prime|\theta)} y$$

The power of the DNNs lie in the fact that the models learns the features directly from the given training data and avoids hand engineered features.

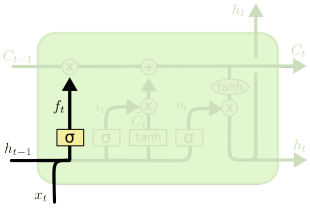
$$x \xrightarrow[\text{hierachical}]{\mathbb{M}(x|\Theta)} y$$

The DNNs with neurons in one layers being connected to all the neurons in the next layer are called Feed Forward Networks. The feed forward networks are harder to train since they have lots of parameters. To make a the DNNs learn useful representation for performing supervised learning, we need special types of connections between the neurons. Two of the most commonly used DNN architectures are Convolutional Neural Networks(CNN) and Recurrent Neural Networks(RNN). In the following section we will discuss these architectures in detail.

1) CNN: todo

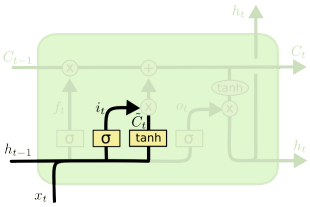
The Convolutional NN are shown in 4

2) RNN: The neurons in the RNNs have recurrent self connections. i.e.The output of the neurons are connected back to the inputs. Thus the output at time step t_1 is computed with taking output at time step t_0 as input. The recurrent connections are shown Fig.5 This enables the RNNs to have an internal memory. RNNs are trained with the a modified version of back propagation called **Back Propagation Through Time(BPTT)** [14].



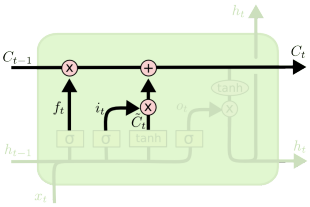
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fig. 6: Forget gate in LSTM



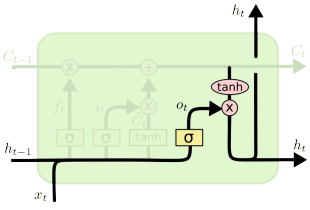
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig. 7: Add gate computing parts to be to modified in LSTM



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 8: Add gate computing data to be added to cell state in LSTM



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 9: Add gate computing output in LSTM

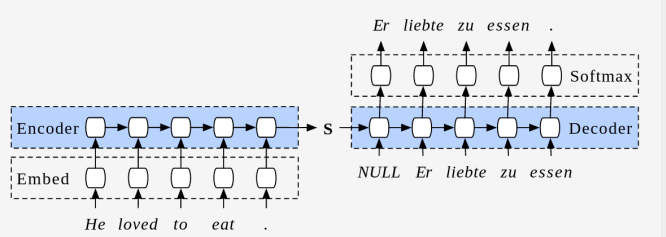


Fig. 10: Simple Encoder-Decoder architecture for machine translation

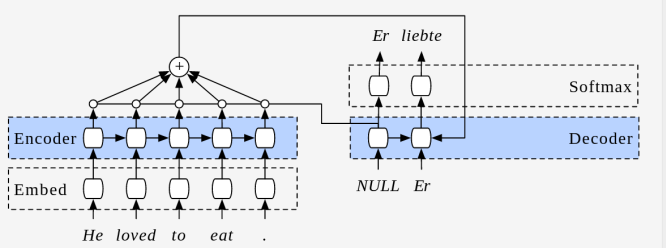


Fig. 11: Context for machine translation

RNNs in general are harder to train and this is particularity evident when the BPTT is done over a larger number of time steps. This is because the gradients simply converges to zero after a few time steps. This problem known as **vanishing gradients problem** is a well studied phenomenon [2].

To alleviate the vanishing gradients problem, a special architecture called **Long Short Term Memory(LSTM)** [6] is used. An LSTM cell is shown in .

LSTM cells has separate internal memory vector and has three gates–forget gate, add gate, and output gate. To understand the Mathematical intuition behind these gates, let us consider a LSTM cell in a hidden state h and internal cell memory vector c_{t-1}. At time t, the cell takes input x_t, previous output h_{t-1} and update the cell memory to c_t, produces the output h_t. The forget computes a vector size c between 0 and 1. This vector determines who information should be preserved and what should be forgotten based on the current input x_t as shown in 6. This vector is later multiplied with the cell memory. If the vector is all zeros, multiplying makes all cell state to forget everything while all ones preserve everything. Then the add gate computes which parts in the cell states to be updates as shown in 7 and the new data to be updated as shown in 8. Finally the output gate generates the output h_t and does not modify the cell state as shown in 9.

B. Related work

PBMT explanation. [8]

C. Neural Machine Translation

Though usage of neural networks did not yield promising results in the early years, the recurrent neural networks started to achieve performance comparable to PBMT as in [7] and [5]. Most of the early architectures were simple encoder-decoder architectures. An simple working of encoder-decoder architecture for machine translation is shown in Fig.10. The architecture has an encoder that takes a sentence in source language and encodes it into a vector S of fixed length. The decoder takes the embedding S as input and generates the sentence in the target language. Some of the major drawbacks with the simple architecture is that

1) The encoder has to encode all the information in the source sentence into fixed size embedding S.
2) The decoder never sees the actual input sentence and has to rely completely on S for generating target sentence.
3) Having fixed size embedding vector makes the architecture less flexible. Smaller size means less information where as using larger vector means for smaller sentences we need zero padding and more computation time.

1) Jointly learning to align and translate: The paper [1] address this issue of having a fixed size embedding by introducing the idea of context 11. The main proposal was to use

1) Encoder outputs a hidden representation for each word in the source sentence F_s.
2) One context vector C in the size of the input sentence that has values between 0 and 1.
3) Each element of the embedding F_s is multiplied with one element in C.

The whole embedding is made available to the decoder and C describes which part of the embedding should be focused to generate the current word based on the previous word.

Encoder RNN, at each input step t, generates hidden state, $h_t = f(x_t, h_{t-1})$.

Unlike in the previous models where only the last hidden state is made available to the decoder, this paper provided all the hidden state and also a context vector that has a weight for each of the hidden vector. The context vector helps the decoder to focus on a part of the sentence. $c = q(\{h_1, \cdots, h_{T_x}\})$.

The decoder is trained to predict the next work y_t given the context vector c and all previously predicted words $\{y_1, \cdots, y_{t-1}\}$

$$p(y) = \prod_{t=1}^{T} p(y_t | \{y_1, \cdots, y_{t-1}\}, c)$$

With RNN, each conditional probability is modeled as,

$$p(y_t | \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

where s_t is the hidden state of the RNN. The context vector for a input sentence i, is computed as a weighted sum of hidden states of the encoder (also known as **annotations**)

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$$

where,

$$e_{ij} = a(s_{i-1}, h_j)$$

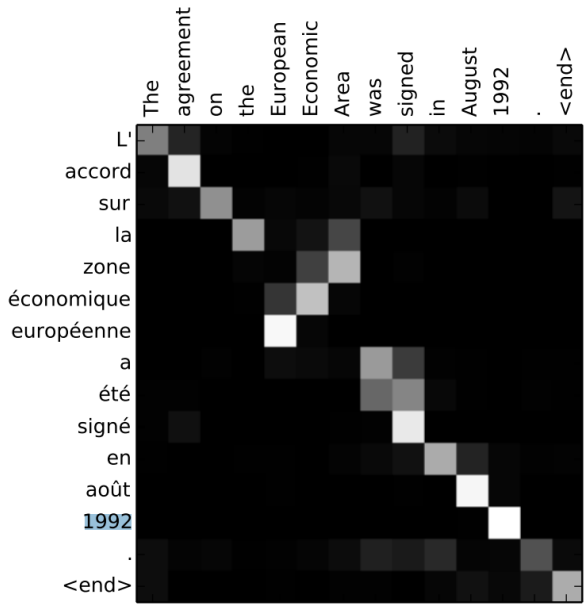


Fig. 12: Visualization of the context in action

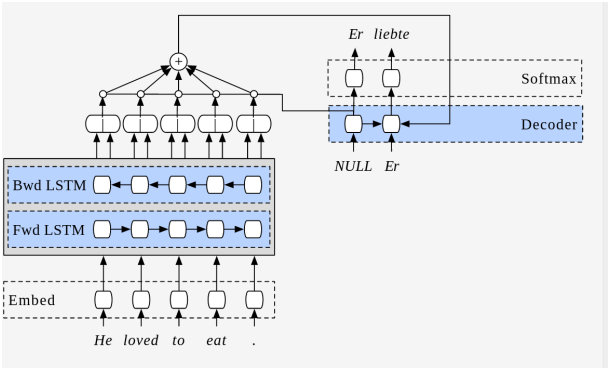


Fig. 13: Bi-directional Encoder

is the alignment model that scores how well the inputs around the j and the output at the position i match.

A feedforward neural network is used as the alignment model and is **jointly trained** with all the NMT system as a whole.

The functionality of the context vector is visualized in the Fig. 12. For example, the French language has the words "European Economic Area"exists in the reverse order as "zone économique européenne". The context learns to focus on the correct order for the decoder.

This paper also made use of the bi-directional LSTM in the layers of the Encoder. Just like an LSTM that has an internal memory to comprehend the past states, the LSTM also comprehends the words that comes next in the sentence. It has two internal state—one for past states and one for the future state. A bi-directional LSTM is shown in 13.

The whole model is trained with standard maximum-likelihood error minimization $\mathbb{O}_{ML}(\Theta) = \sum_{i=1}^{N} logP_{\Theta}(Y^{*(i)}|X^{(i)})$ with stochastic gradient descent(SGD) on a mini-batch of 80 sentences. For the first time, the BLEU score was comparable to phrase based machine translation system. But the fact that there are no individual pieces in the model was a great benefit and the research community started to consider neural systems as an viable alternative phrase based machine translation system.

2) Sequence to Sequence models.: With in an year, more deep and powerful models were computationally possible. Taking the availability of the computational power to advantage a new framework for learning sequence to sequence mapping were proposed. A sequence to sequence model formulated as

$$p(y_1, \cdots, y_T|x_1, \cdots, x_T) = \prod_{t=1}^{T} p(y_t|v, y_1, \cdots, y_{t-1})$$

where v is the internal memory of the RNNs. With more powerful RNNs, this model can be used to learn a variety of tasks like speech recognition, handwritten digits recognition, machine translation, etc as shown in the paper [12]

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[5] Karl Moritz Hermann and Phil Blunsom. Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173*, 2013.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, volume 3, page 413, 2013.

[8] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[11] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[13] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[14] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.