Machine Learning for Natural Language Processing

Neural Natural Language Processing

Lecture 4

Benjamin Muller

INRIA Paris - ALMANACH
benjamin.muller@inria.fr

Course Outline

1. The Why and What of Natural Language Processing
2. Representing text with vectors
3. Task specific Modeling of Text
4. Neural Natural Language Processing
5. Language Modeling
6. Transfer Learning with Neural Modeling for NLP

Lecture 3

Task-specific modelling of textual data

- Preprocessing (encoding, segmentation)
- Modelling sequence tagging tasks (such as POS or NER) with MEMMS and CRF model
- Modelling Sentiment Analysis with a *bag-of-words* model
- Evaluating classification

Lecture Outline

- Deep Learning
  - Feed-Forward Neural Network
  - Recurrent Neural Network
  - Attention Mechanism
  - Embedding layer
- Training Neural Networks
- Differentiable Programming (Pytorch, Tensorflow)

Deep Learning toolkit

What is Deep Learning ?

Definition

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher level features from the raw input[1]

[1] Deng et al. (2014)

This lecture is..

- This lecture **is not** a Deep Learning lecture
- This lecture is a Deep Learning lecture applied to Natural Language Processing

To grasp the overall Deep Learning picture Goodfellow et al. (2016)

Deep Learning and Natural Language Processing

- These last years, Deep Learning has become the main modelling framework for (almost) all NLP tasks

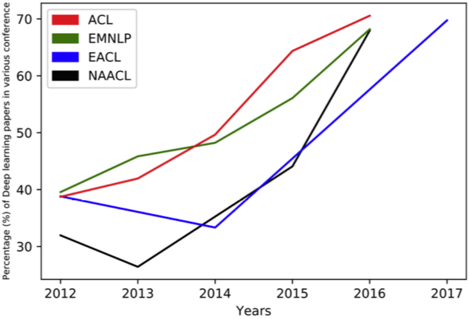

Figure: % of papers published in main NLP conferences that use deep learning

Deep Learning Model

A Deep Learning Model is the combination of these three elements:

- Data
- Architecture
- Training Process (optimization)

Deep Learning Architectures

- Logistic Regression (1 layer Feed-Foward Neural Network)
- Feed-Forward Neural Network
- Recurrent Neural Network
- Attention
- Embedding Layer

Framework

Let $(X_i, .., Y_i)_{i:1..n}$ $X_i \in R^d, Y_i \in R^{d'}$ input and output variables.

Goal: Learn a predictive model of Y_i with X_i.

Logistic Regression (1 layer Neural Network)

Assuming Y is single dimension ($d' =: 1$)

$$Y = \sigma(\sum_j w_j X_j)$$

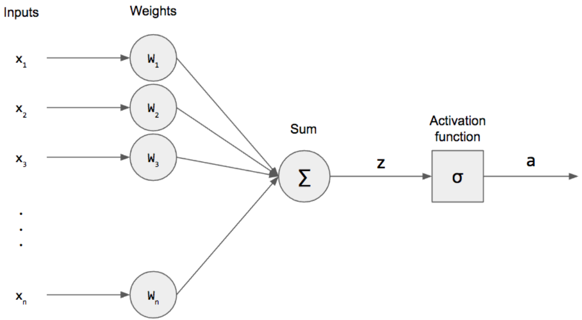

Figure: Representation of 1 layer Neural Network (NN) with sigmoid activation

Feed-Forward Neural Network[2]

We define *a L layers network*, with *activation* function f
W_l weights matrices $W_l \in R^{d_{l-1},d_l}$

$$h_0 = x$$

$$h_l = f(W_l h_{l-1}) \text{ for } l \in 0, .., L$$

$$\hat{y} = h_L$$

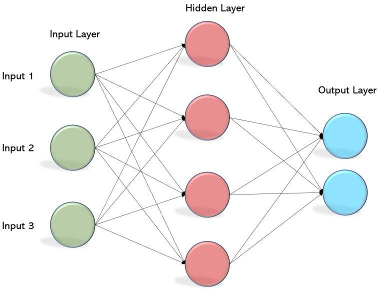

Figure: Representation of 2 layers neural network (1 hidden layer)

NB : h_l is a **hidden vector**.

[2] also called ***Multi-Layer Perceptron (MLP)***

Training Neural Networks: Backpropagation

- We train NNs with Stochastic Gradient Descent (SGD).
- For NNs, computing the gradient with the Chain Rule is very efficient computationally
- We call backpropagation, SGD applied to neural networks.
- All weights are **initialized randomly**

Training Neural Networks: Backpropagation

- We train NNs with Stochastic Gradient Descent (SGD).
- For NNs, computing the gradient with the Chain Rule is very efficient computationally
- We call backpropagation, SGD applied to neural networks.
- All weights are **initialized randomly**

Back-propagation for a Feed-Forward Neural Network (FNN)

Forward Pass for observation x_i

$$h_0 = x_i$$

$$h_l = f(W_l h_{l-1}) \text{ for } l \in 0, .., L$$

$$\hat{y}_i = h_L$$

$l_w(y_i, \hat{y}_i)$ loss function

Backward pass

$$l_w(y_i, \hat{y}_i)$$

$$w_{lk} := w_{lk} - \eta \frac{\partial l_w(y_i, \hat{y}_i)}{\partial w_l k}$$

$\frac{\partial l}{\partial w}$ computed with backprop.

Backpropagation

Backpropagation in a nutshell

- We alternate between forward pass i.e $x_i \rightarrow l_w(y_i, \hat{y}_i)$
- Backward pass during which each weight is updated with regard to the error $l_w(y_i, \hat{y}_i)$, $w \rightarrow w - \eta \frac{l_w(y_i, \hat{y}_i)}{\partial w}$

Intuition

- Iteration after iteration: each weights are updated to minimize the loss function i.e the gap between the prediction and the observed values
- The network weights learn representation of the input that lead to the "best" prediction of the output.

Modelling Sequence with Neural Networks

Let $(X^1, .., X^T)_i, (Y^1, .., Y^T)_i$ sequence of input and output (e.g X^t *1-hot encoded words,* Y^t *POS tags*)

- Feed-Forward Neural Network do not model sequence as such
- Solution: Introducing *recurrent relation* in the network

Recurrent Neural Networks

- Family of neural network architectures in which a recurrent relation is induced by the architecture
- Recurrent Neural Network (RNN) Architectures
    - Vanilla RNN
    - Long-Short-Term Memory RNN

Vanilla RNN

Forward Pass for observation x_i,
activation function f

$$h_0^t = x_i^t$$

$$\text{for } l \in 0, .., L \text{ for } t \in 0, .., T$$

$$h_l^t = f(W_l^f h_{l-1}^t + W_l^r h_l^{t-1} + b_l)$$

$$\hat{y}_i^t = h_L^t$$

$$l_w(y_i, \hat{y}_i) = \sum_{k=1..T} \sum_{k=1..d'} y_{ik}^t . log(\hat{y}_{ik}^t)$$

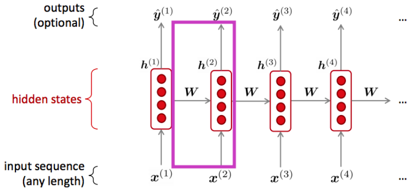

Figure: Recurrent Neural Network

Vanilla RNN schema

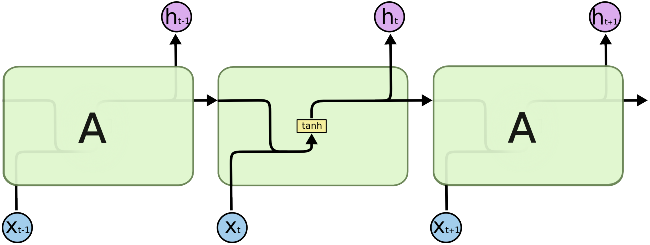

Figure: Recurrent Neural Network[3]

[3] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Training RNN

- RNNs can be trained with some form of Backpropagation through time (BPTT)
- BPTT is the straight application of Backpropagation with recurrent connections[4]

[4] https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

Limits of Vanilla RNN

- Vanishing Gradients problem.
  First elements of the sequence will not get gradients updates. Hence the long range dependencies are poorly captured by the Vanilla RNN.
- Solution: Memory specific weights e.g Long-Short Term Memory (LSTM) cell

LSTM cell

Let $x^t \in R^d$ element of the sequence $(x^1, .., x^T)$ our input
We introduce *memory cell* variable C^t.
The recurrence formula of h^t based on h^{t-1} is given by :

$$\widetilde{C}^t = tanh(W_C[x_t, h_{t-1}] + b_c) \quad \text{candidate cell}$$

$$f^t = \sigma(W_f[x_t, h_{t-1}] + b_f) \quad \text{forget gate}$$

$$i^t = \sigma(W_i[x_t, h_{t-1}] + b_i) \quad \text{input gate}$$

$$o^t = \sigma(W_o[x_t, h_{t-1}] + b_o) \quad \text{ouput gate}$$

$$C^t = i^t \star \widetilde{C}^t + f^t \star C^{t-1} \quad \text{new cell state}$$

$$h^t = o^t \star tanh(C_t) \quad \text{new hidden vector}$$

5

[5] $\star$ *elementwiseproduct*

LSTM cell

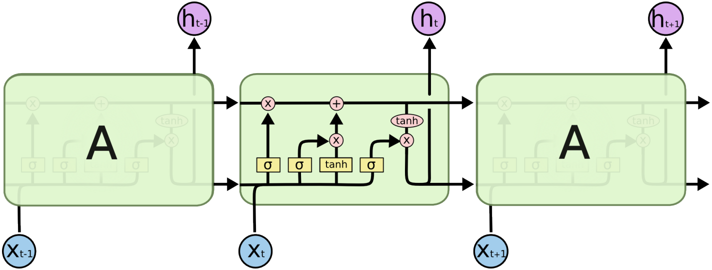

Figure: LSTM [6]

[6] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM intuition

- For each input , a new cell candidate is computed
- Based on the different gates, we compute what information is forgotten, took from the new candidate, and outputed.

- Every weights are trained simultaneously with BPTT
- Based on the input - output, every gates and weights are updated to lead to the "best" prediction

LSTM contextual representation

- Each hidden vector h^t captures information about the left sequence
- Each hidden vector represents the given token t based on its left context.
- h^t is a contextual representation of the token t

Architecturing neural networks

- Feed-Forward Layers, Recurrent Layers (Vanilla, LSTMs) can be used as building blocks for any more complex architectures
- Architectural decisions are task specific
- Best practices and Empirical results should drive what architecture is chosen for a given task

How to model discrete data with Neural Networks ?

- So far, we did not focus on the input data
- Neural Networks work better have been shown to perform better with dense vectors compared discrete vectors
- How to handle discrete tokens tokens efficiently ?

  $\rightarrow$ embedding layer

Embedding Layer

- A **token** is the **basic unit** of discrete data, defined to be an item from a vocabulary indexed by $1, ..., V$.
- We define $E \in R^{Vd}$ an embedding matrix which associates each token to a vector in R^d
- We train those weights simultaneously with backpropagation as any other weights
- We initialize this embedding layer:

Embedding Layer

- A **token** is the **basic unit** of discrete data, defined to be an item from a vocabulary indexed by $1, ..., V$.
- We define $E \in R^{Vd}$ an embedding matrix which associates each token to a vector in R^d
- We train those weights simultaneously with backpropagation as any other weights
- We initialize this embedding layer:
  - randomly as any other weights
  - using a pretrained skip-gram word2vec (or any other word embedding techniques)

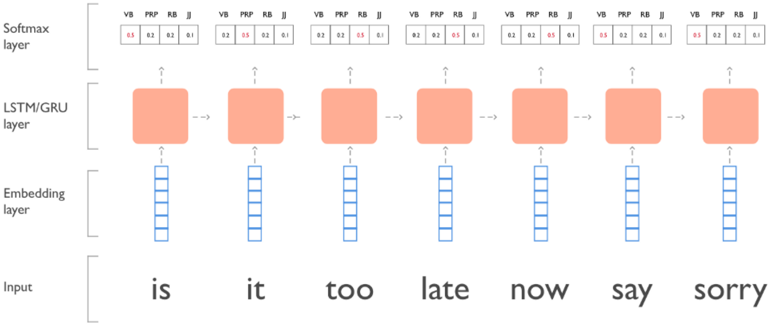

Figure: POS tagging with embedding layer and LSTM recurrent network [7]

[7] GRU is a light variant of LSTM

Sequence Labelling with LSTM: back to POS tagging

We define a sequence labelling RNN with LSTM cell and output feed-forward layer

Let $(x^0, .., x^T)$ sequence of words, $(y^0, .., y^T)$ tags.
All LSTM weights, W_{fo} forward layer and Embedding layer E initialized randomly

For $t \in 0, .., T$

$$h^t, C^t = LSTM_w(x^t, h^{t-1}, C^{t-1})$$

$$s^t = tanh(W_{fo} h^t)$$

$$\hat{y^t} = argmax_k(\hat{p^t})$$

We compute the Cross-Entropy loss based on

$$l_w(y_i, \hat{y}_i) = \sum_{k=1..T} \sum_{k=1..d'} y_{ik}^t . log(\hat{p_{ik}^t})$$

We apply backpropagation.

Bi-Directional LSTMs

- RNNs define a recurrence relation in the left-to-right direction
- What if we need both left-to-right and right-to-left context

  $\rightarrow$ bi-directional LSTM

Bi-Directional LSTMs

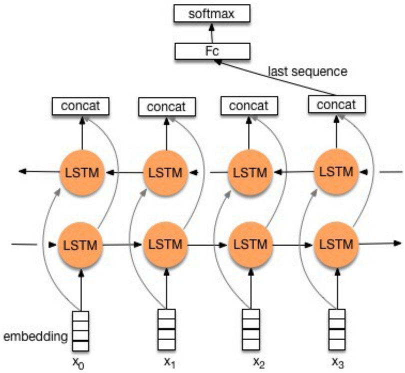

Figure: LSTM

- Define two LSTM cells: one left to right and one right to left
- Concatenate for each step the output hidden vectors

Attention Mechanism

- For input element x^t: the RNN provides a vector representation h^t (contextual)
- For some tasks, some input are more important than others

  $\rightarrow$ attention mechanism

Attention Mechanism

Let $(x^0, .., x^T)$ sequence of words, y label (sequence classification)

We define LSTM+dense layer neural network

$$s^T = f_W(h^T) \text{ with } h^t, C^t = LSTM(h^{t-1}, C^{t-1}, x^t) \text{ for all t}$$

With Attention it becomes

$$s^T = f_W(\widetilde{h^T}) \text{ with } \widetilde{h^T} = \sum_t a_t h^t \text{ and } h^t = LSTM(h^{t-1}, C^{t-1}, x^t)$$

Question: How to compute a_t?

How to compute Attention weights ?

Here is a simpe attention mechanism:

$$u^t = tanh(W_a h^t)$$

$$a^t = softmax_w(u^t)$$

$$\widetilde{h^T} = \sum_t a_t h^t$$

Attention mechanism

- Many different variants of attention mechanisms
- As with all other weights, attention weights are trained end-to-end with backpropagation
- $\widetilde{h^T}$ is also a contextual representation vector of the token x^T

Attention mechanism as an interpretation tool

- Neurals networks are black box models
- Attention mechanisms provide some insights on what information the network is using to make its prediction
- Warning: It does not say how it is using it!

Attention mechanism as an interpretation tool

Using the attention weight we can compute heatmaps in a straightforward way :

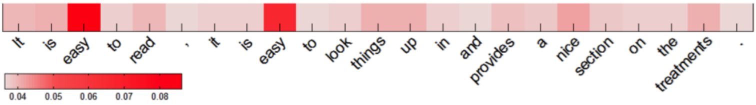

Figure 3: Attention visualization for a review sentence

Zhou et al. (2016)

How do we build/train neural networks in practice ?

- All seen neural networks can be trained using backpropagation [8]
- Implementing backpropagation by hand for all our models is time consuming

[8] if activation function are differential

How do we build/train neural networks in practice ?

- All seen neural networks can be trained using backpropagation [8]
- Implementing backpropagation by hand for all our models is time consuming
- Happily, came differentiable programming and deep learning libraries!

[8] if activation function are differential

Deep Learning Libraries

Definition
Differentiable programming is a programming paradigm in which the programs can be differentiated throughout, usually via automatic differentiation

Deep Learning libraries are implementation of automatic differentiation modules optimized with user-friendly tools for Deep Learning.

e.g: Tensorflow, Keras, Dynet, Pytorch are popular Deep Learning libraries In the lab we will use Pytorch

Deep Learning Libraries

With deep learning libraries, implementing neural networks becomes easy

- Pre-implemented modules like: *Linear* layers, LSTM cells, SGD optimizers
- Automatic Differentiation

In the following lab, we will build a LSTM recurrent neural network for Sentiment Classification using pytorch.

Lecture Summary

- Deep Learning architectures for NLP
  - Feed-Forward Neural Network
  - Recurrent Neural Network (LSTM)
  - Embedding layer
  - Attention Mechanism
- Deep Learning libraries

References I

Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Zhou, X., Wan, X., and Xiao, J. (2016). Attention-based lstm network for cross-lingual sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 247–256.