Preliminary to ANN, Basic RNN model LSTM and gradient analysis

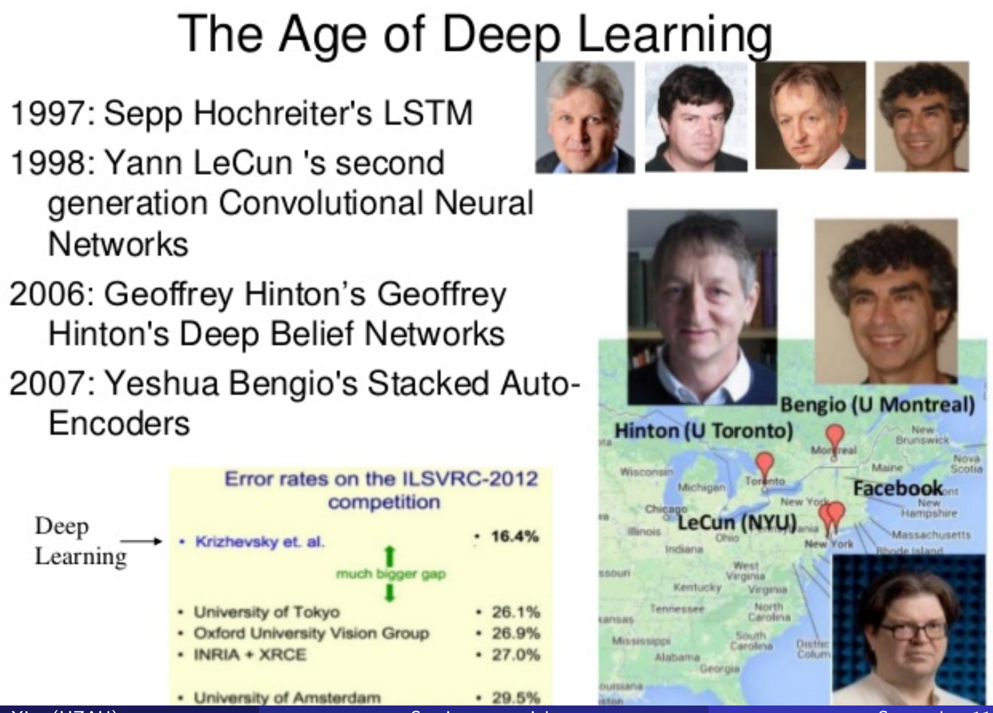
Jingbo Xia

Huazhong Agricultural University

xiajingbo.math@gmail.com

September 11, 2018

Preliminary ANN, Basic RNN model LSTM and gradient analysis



The Age of Deep Learning

1997: Sepp Hochreiter's LSTM

1998: Yann LeCun 's second generation Convolutional Neural Networks

2006: Geoffrey Hinton's Geoffrey Hinton's Deep Belief Networks

2007: Yeshua Bengio's Stacked Auto-Encoders

Error rates on the ILSVRC-2012 competition

Deep Learning → • Krizhevsky et. al. • 16.4%

much bigger gap

• University of Tokyo	• 26.1%
• Oxford University Vision Group	• 26.9%
• INRIA + XRCE	• 27.0%
• University of Amsterdam	• 29.5%

Preliminary ANN

It is strongly recommended to learn CS224d or CS224n, a MOOC offered by Stanford University.
Very often, softmax and cross entropy are used for labeling, so the formulas are list here:
For a given input vector x, softmax function is defined as

$$softmax(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}. \tag{1}$$

For an one-hot vector y and a predicted vector $\bar{y}$, cross entropy is defined as

$$CE(y, \bar{y}) = -\sum_i y_i \log(\bar{y}_i). \tag{2}$$

Understanding Backward propagation (BP) neural network

Backward propagation NN was a classical one, and the advantage of learning BP NN is to observe the error propagation distribution by using basic but fundamental gradient analysis.
Please note that the first step to learn NN is to learn to read NN graph and map the neuron elements back to notations and symbols.
This section will be very fascinated, I promise. (Referred from CS224D: Deep Learning for NLP, Lectrue Notes: Part III.)

Understanding BP neural network — CONT

A single Layer of Neurons

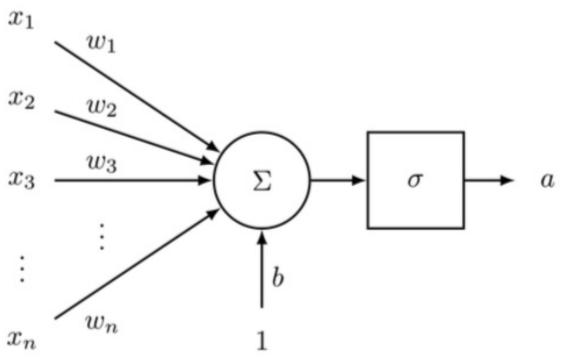


Figure 1: One Neuron Computation. This image captures how in a sigmoid neuron, the input vector x is first scaled, summed, added to a bias unit, and then passed to the squashing sigmoid function.

This unit takes an n-dimensional input vector x and produces the scalar activation (output) a. This neuron is also associated with an n-dimensional weight vector, w, and a bias scalar, b. The output of this neuron is then:

$$a = \frac{1}{1 + \exp(-(w^T x + b))}.$$

Understanding BP neural network — CONT

Feed-forward computation

So far we have seen how an input vector $x \in R^n$ can be fed to a layer of sigmoid units to create activations $a \in R$. But what is the intuition behind doing so?
Let us consider the following NER as an example:

"Museums in Paris are amazing"

Here we want to classify whether or not the center word "Paris" is a named-entity. In this case, we want to capture both the presence of the words in the window of word vectors, and interactions between the words. That intention make the fully connected network meaningful!
Thus, we use matrix $U \in R^{m\times 1}$ to generate an unnormalized score for a classification task from the activations:

$$s = U^T a = U^T f(Wx + b).$$

Understanding BP neural network — CONT

Feed-forward computation

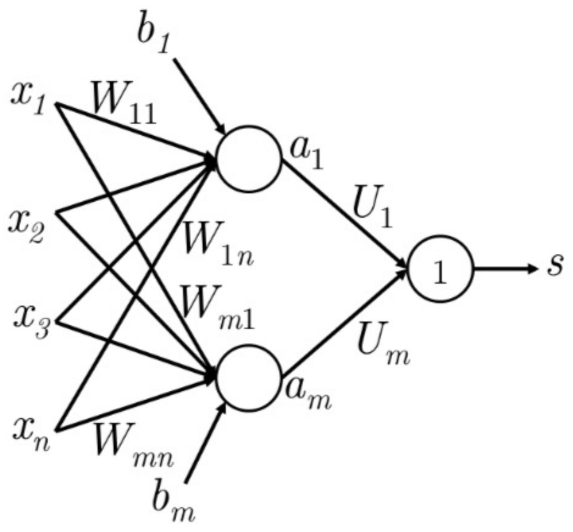


Figure 2: This image captures how a simple feed-forward network might compute its output.

Analysis of Dimensions:
If we represent each word using a 4-dimensional word vector and we use a 5-word window as input (as in the example), then the input $x \in R^{20}$.
If we use 8 sigmoid units in the hidden layer and generate 1 score output from the activations, then $W \in R^{8\times 20}$, a and $b \in R^8$, $U \in R^{8\times 1}$, $s \in R$

Understanding BP neural network — CONT

Maximum Margin Objective Function

Using the previous example, if we call the score computed for the "true" labeled window "Museums in Paris are amazing" as s and the score computed for the "false" labeled window "Not all museums in Paris" as s_c (subscripted as $_c$ to signify that the window is "corrupt").
Then, our objective function would be to maximize $(s - s_c)$ or to minimize $(s_c - s)$. In another word:

$$Score(\text{Positive window, "Museum in Paris is amazing"}) = s, \tag{3}$$

$$Score(\text{Negative window, "Not all museum in Paris"}) = s_c. \tag{4}$$

Thus the optimization objective is:

$$minimize \mathcal{J} = max(s_c - s, 0). \tag{5}$$

In some case, we would want error to be calculated if $(s - s_c < \Delta)$ and not just when $(s - s_c < 0)$. Thus, we modify the optimization objective:

$$minimize \mathcal{J} = max(\Delta + s_c - s, 0). \tag{6}$$

Understanding BP neural network — CONT

Training with Backpropagation — Elemental

We typically need the gradient information for any parameter as required in the update equation:

$$\theta(t+1) = \theta(t) - \alpha \cdot \nabla\theta(t) \cdot \mathcal{J}. \tag{7}$$

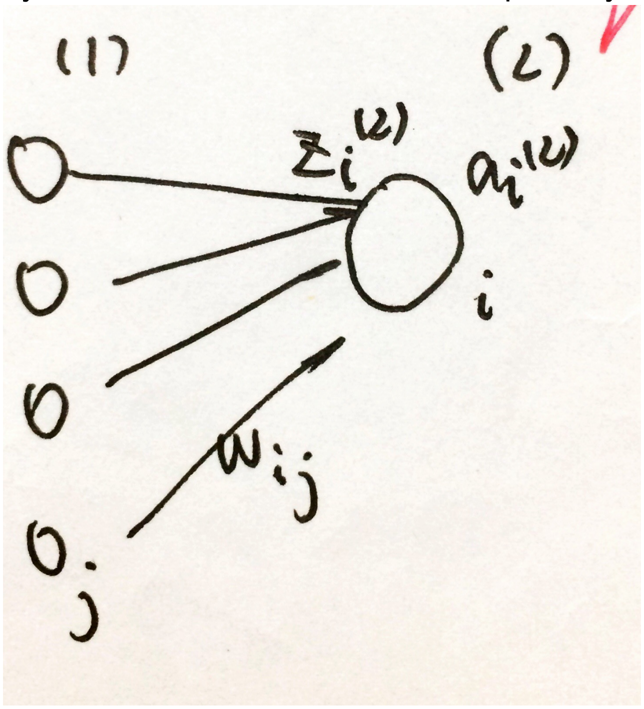
Let's establish some notation that will make it easier to generalize this model:

- x_i is an input to the neural network.
- s is the output of the neural network.
- Each layer (including the input and output layers) has neurons which receive an input and produce an output. The j-th neuron of layer k receives the scalar input $z_j(k)$ and produces the scalar j activation output $a_j^{(k)}$.
- We will call the backpropagated error calculated at $z_j^{(k)}$ as $\delta_j^{(k)}$.
- Layer 1 refers to the input layer and not the first hidden layer. For the input layer, $x_j = z_j^{(1)} = a_j^{(1)}$.
- $W^{(k)}$ is the transfer matrix that maps the output from the k-th layer to the input to the $(k+1)$-th.

Understanding BP neural network — CONT

Training with Backpropagation — Elemental

Please let me remind you of how to trace the subscript of symbols.



Understanding BP neural network — CONT

Training with Backpropagation — Elemental

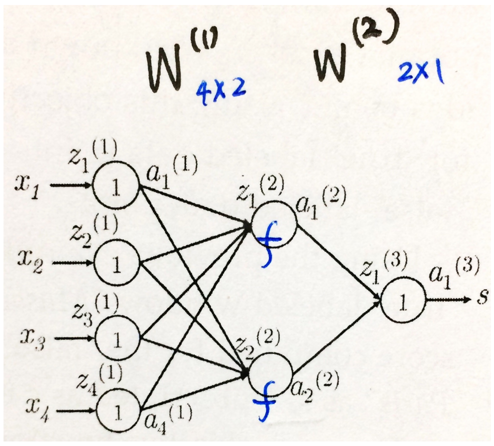


Figure 3: An 4-2-1 NN for backpropagation analysis. Here, neuron j on level (k) receive input $z_j^{(k)}$ and produce activation output $a_j^{(k)}$.

Let us begin: Assume $\Delta = 1$ in equation(6), and suppose the cost $\mathcal{J} = (1 + s_c - s)$ is positive and we want to perform the update of parameter $W_{14}^{(1)}$ (in Figure 3 , we must realize that $W_4^{(1)}$ only contributes to $z_1^{(2)}$ and thus $a_1^{(2)}$. This fact is crucial to understanding backpropagation — backpropagated gradients are only affected by values they contribute to.
We can see from the max-margin loss that:

$$\frac{\partial \mathcal{J}}{\partial s} = -\frac{\partial \mathcal{J}}{\partial s_c} = -1. \qquad (8)$$

Therefore, we will work $\frac{\partial s}{\partial W_{ij}^{(1)}}$ here for simplicity.

Understanding BP neural network — CONT

Training with Backpropagation — Elemental

For $i = 1, 2, 3, 4;\ j = 1, 2,$

$$\begin{aligned}
\frac{\partial s}{\partial W_{ij}^{(1)}} &= \frac{\partial W^{(2)} a^{(2)}}{\partial W_{ij}^{(1)}} \qquad (\because s = W^2 a^{(2)} + b^{(2)} = W_1^2 a_1^{(2)} + W_2^2 a_2^{(2)} + b^{(2)})\\
&= \frac{\partial W_i^{(2)} a_i^{(2)}}{\partial W_{ij}^{(1)}}\\
&= W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial W_{ij}^{(1)}}\\
&= W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial W_{ij}^{(1)}}\\
&= W_i^{(2)} \frac{\partial f(z_i^{(2)})}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial W_{ij}^{(1)}}\\
&= W_i^{(2)} f'(z_i^{(2)}) \frac{\partial z_i^{(2)}}{\partial W_{ij}^{(1)}} \ (\because z_i^{(2)} = b_i^{(1)} + a_1^{(1)} W_{i1}^{(1)} + a_2^{(1)} W_{i2}^{(1)} + a_3^{(1)} W_{i3}^{(1)} + a_4^{(1)} W_{i4}^{(1)}\\
&= W_i^{(2)} f'(z_i^{(2)}) \frac{\partial}{\partial W_{ij}^{(1)}} (b_i^{(1)} + a_1^{(1)} W_{i1}^{(1)} + a_2^{(1)} W_{i2}^{(1)} + a_3^{(1)} W_{i3}^{(1)} + a_4^{(1)} W_{i4}^{(1)})\\
&= W_i^{(2)} f'(z_i^{(2)}) a_j^{(1)}
\end{aligned} \qquad (9)$$

Understanding BP neural network — CONT

Training with Backpropagation — Elemental

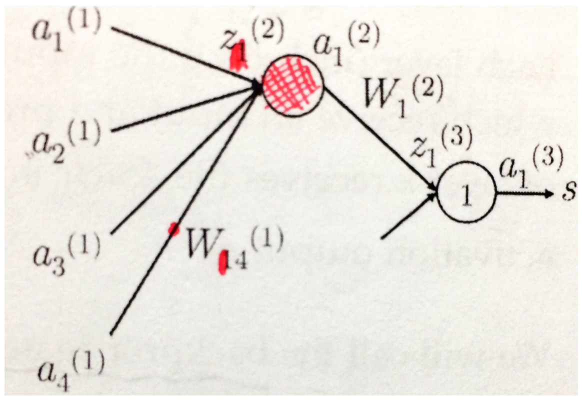


Figure 4: This subnetwork shows the relevant parts of the network required to update $W_{ij}^{(1)}$.

From Equation (8) and (9), we know that the gradient of $W_{ij}^{(1)}$ to the loss function is:

$$\frac{\partial \mathcal{J}}{\partial W_{ij}^{(1)}} = W_i^{(2)} f'(z_i^{(2)}) a_j^{(1)}, \qquad (10)$$

and this value later is shown to essentially be the error propagation backwards from the i-th neuron in layer 2, $\delta_i^{(2)}$:

$$\delta_i^{(2)} := W_i^{(2)} f'(z_i^{(2)}). \qquad (11)$$

To observe the update of $W_{14}^{(1)}$, an easy view is taken from relevant part of the network, see figure 4.

Understanding BP neural network — CONT I

Training with Backpropagation — Elemental

Let us discuss the "error sharing/distribution" interpretation of backpropagation better using Figure 4 as an example. Say we were to update $W_{14}^{(1)}$:

Step 1: Please recall that the backpropagated error calculated at $z_j^{(k)}$ was defined as $\delta_j^{(k)}$.

Step 2: We start with the an error signal of 1 propagating backwards from $a_1^{(3)}$.

Step 3: As $z_1^{(3)} = a_1^{(3)}$ in this case, and the error is still 1. This is now known as $\delta_1^{(3)} = 1$.

Step 4: At this point, the error signal of 1 has reached $z_1^{(3)}$. We now need 1 to distribute the error signal so that the "fair share" of the error reaches to $a_1^{(2)}$.

Step 5: This amount is the (Error signal at $z_1^{(3)} = \delta_1^{(3)}$) $\times W_1^{(2)} = W_1^{(2)}$. Thus, the error at $a_1^{(2)}$ equals to $W_1^{(2)}$.

Understanding BP neural network — CONT II

Training with Backpropagation — Elemental

Step 6: As we did in step 2, we need to move the error across the neuron which maps $z_1^{(2)}$ to $a_1^{(2)}$. We do this by multiplying the error signal at $a_1^{(2)}$ by the local gradient of the neuron which happens to be $f'(z_1^{(2)})$.

Step 7: Thus, the error signal at $z_1^{(2)}$ is $f'(z_1^{(2)})W_1^{(2)}$. This is known as $\delta_1^{(2)}$.

Step 8: Finally, we need to distribute the "fair share" of the error to $W_{14}^{(1)}$ by simply multiplying it by the input it was responsible for forwarding, which happens to be $a_4^{(1)}$.

Step 9: Thus, the gradient of the loss with respect to $W_{14}^{(1)}$ is calculated to be

$$a_4^{(1)} f'(z_1^{(2)}) W_1^{(2)}. \tag{12}$$

Understanding BP neural network — CONT

Training with Backpropagation — Elemental

From the combination of equation(10) and equation (12), we have

$$\frac{\partial \mathcal{J}}{\partial W_{ij}^{(1)}} = \delta_i^{(2)} \times a_j^{(1)}. \tag{13}$$

Conclusion of BACKFORWARD PROPAGATION

This is an amazing result, which shows that the gradient of loss computation equals to the computation of 1 error back forward computation!

" Quote from CS224d: Notice that the result we arrive at using this approach is exactly the same as that we arrived at using explicit differentiation earlier. Thus, we can calculate error gradients with respect to a parameter in the network using either the chain rule of differentiation or using an error sharing and distributed flow approach ? both of these approaches happen to do the exact same thing but it might be helpful to think about them one way or another."

Outline

Understanding LSTM

Well, how to understand a typical RNN model, LSTM? It's pretty well-known!

We will borrow Zheng's example from *Neurocomputing*...

Understanding LSTM

$$
\begin{aligned}
i_t &= \delta(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)\\
f_t &= \delta(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)\\
z_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)\\
c_t &= f_t c_{t-1} + i_t z_t\\
o_t &= \delta(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)\\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\quad (14
$$

The LSTM architecture consists of a set of recurrently connected subnets, known as memory blocks.

x_t: input; h_t: hidden layer; c_t: cell level.

$z_t \in (-1, 1)$: Block gate; $i_t \in (0, 1)$: Input gate; $f_t \in (0, 1)$: Forget gate.

Ref: Zheng, 2017 [1].

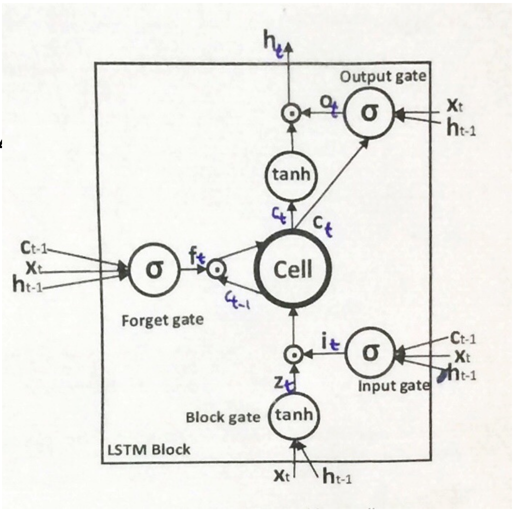


Figure 5: LSTM block.

[1] Zheng S, Hao Y, Lu D, Bao H, Xu J, Hao H, Xu B. Joint entity and relation extraction based on a hybrid neural network. Neurocomputing. 2017 Sep 27;257:59-66.

Understanding LSTM – CONT.

NER: Concatenate the bi-LSTM outputs, $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$, and we have $h_t = [\overleftarrow{h_t}, \overrightarrow{h_t}]$. Let h_t be the input of encoding LSTM, and s_t be the hidden layer of encoding LSTM. Modify equation (**??**) to

$$i_t = \delta(W_{hi}h_t + W_{si}s_{t-1} + W_{Ti}T_{t-1} + b_i),$$

where T_{t-1} is actually the former Tag predicted vector, see figure 6.

$$
\begin{aligned}
T_i &= W_{st}s_t + b_T && \text{—Linear!}\\
y_t &= W_{Ty}T_t + b_y && \text{—Linear!}\\
p_t^i &= \frac{\exp(y_t^j)}{\sum_{j=1}^{nt}\exp(y_t^j)} && \text{—Softmax!}
\end{aligned}
\qquad (15)
$$

nt is the total number of entity tags.

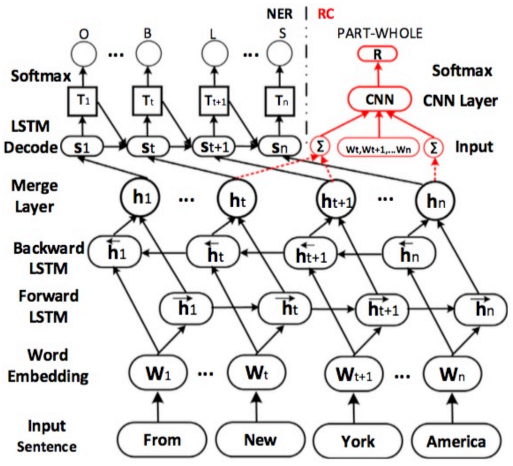
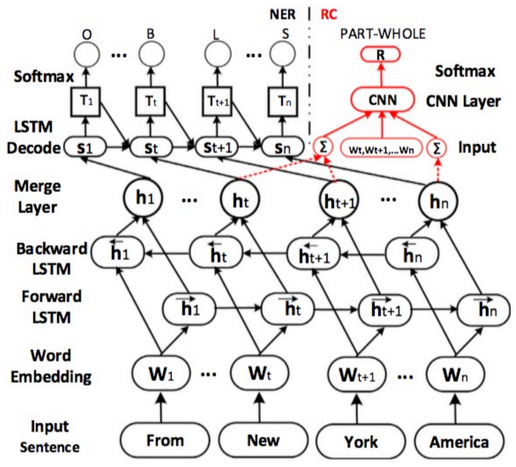


Figure 6: Joint extraction of NER and Relation by Zheng

Understanding LSTM – CONT.

The relation classification model is similar. Just merge Merge Layer h_t into Word Embedding $W_1, W_2, \cdots, W_n$, and then followed with a CNN.



Acknowledgements

Thank you!