CSCI-UA 9472 Artificial Intelligence

Additional Note on Neural Networks

Augustin Cosse

October 2020

1 A simple neuron

Recall that each unit of a neural network can be seen as an instance of the general model

$$y(\boldsymbol{x}) = \sigma(w_0 + \sum_{j=1}^{D} w_j x_j) \tag{1}$$

A popular approach is to take the activation $\sigma(x)$ to be the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Such a choice is convenient as it makes it possible to interpret the output of the classifier (or neuron) (1) as a proability. Indeed, since the sigmoid maps the real line onto the $[0,1]$ interval, one can decide to use model (1) to represent the probability that a particular example should be classified in class $\mathcal{C}_1$. Consequently, in the binary classification framework, the probability that an example $\boldsymbol{x}^{(i)}$ should be classified in class $\mathcal{C}_0$ will be given by $1 - \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})$. For this particular approach, we thus write

$$p(x^{(i)} \in \mathcal{C}_1) = p(t^{(i)} = 1|\boldsymbol{x}^{(i)}) = \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)}) \tag{2}$$

$$p(x^{(i)} \in \mathcal{C}_0) = p(t^{(i)} = 0|\boldsymbol{x}^{(i)}) = 1 - \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)}) \tag{3}$$

Now that we have chosen a model for our classifier, we are left with "learning" that model. An approach is to look for the weights $\boldsymbol{w}$ that maximize the probability of observing the data sample $\left\{\boldsymbol{x}^{(i)}, t^{(i)}\right\}$ we have. If we assume that our examples have been generated independently and that for $\boldsymbol{x}^{(i)} \in \mathcal{C}_1$, we let $t^{(i)} = 1$, and for $\boldsymbol{x}^{(i)} \in \mathcal{C}_0$ we let $t^{(i)} = 0$, the total probability of observing the set $\left\{\boldsymbol{x}^{(i)}, t^{(i)}\right\}_{i=1}^{N}$ can read as

$$\begin{align}
p(\{t_i\}_{i=1}^N |\{\boldsymbol{x}^{(i)}\}_{i=1}^N) &= p(\left\{t(\boldsymbol{x}^{(i)}) = t^{(i)}\right\}_{i=1}^N |\{\boldsymbol{x}^{(i)}\}_{i=1}^N) \tag{4}\\
&= \prod_{i=1}^{N} p(\boldsymbol{x}^{(i)} \in \mathcal{C}_1)^{t^{(i)}} p(\boldsymbol{x}^{(i)} \in \mathcal{C}_0)^{1-t^{(i)}} \tag{5}\\
&= \prod_{i=1}^{N} \left(\sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})\right)^{t^{(i)}} \left(1 - \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})\right)^{1-t^{(i)}} \tag{6}
\end{align}$$

instead of maximizing the probability (6), a common approach (in order to get rid of the product) is to maximize the log of this probability.

We can do this because the probability is non negative and the log is an increasing function so any x^* such that $p(x^*) \geq p(x)$ for all x also satisfies $\log(p(x^*)) \geq \log(p(x))$ for all x.

When maximizing the log of (6), we get

$$\log \prod_{i=1}^{N} \left(\sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})\right)^{t^{(i)}} \left(1 - \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})\right)^{1-t^{(i)}} \tag{7}$$

$$= \sum_{i=1}^{N} t^{(i)} \log(\sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})) + (1 - t^{(i)}) \log(1 - \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})) \tag{8}$$

$$= \sum_{i=1}^{N} t^{(i)} \log(y(\boldsymbol{x}^{(i)}; \boldsymbol{w})) + (1 - t^{(i)}) \log(1 - y(\boldsymbol{x}^{(i)}; \boldsymbol{w})) \tag{9}$$

Expression (9) is known as the log loss or binary cross entropy loss and it corresponds to the maximum likelihood estimator. It is also common to minimize the opposite of (9) which is known as the negative log likelihood (recall that finding the value $\boldsymbol{x}$ at which a function $f(\boldsymbol{x})$ is maximized is the same as finding the value $\boldsymbol{x}$ at which the function $-f(\boldsymbol{x})$ is minimized). In this case the weights are obtained as

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} - \sum_{i=1}^{N} t^{(i)} \log(\sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})) + (1 - t^{(i)}) \log(1 - \sigma(w_0 + \sum_{j=1}^{D} w_j x_j^{(i)})) \tag{10}$$

Learning can be done through gradient or stochastic gradient descent.

2 Neural Network

In section 1 above, we saw how we could learn the weights of a simple neuron (i.e. simple generalized linear classifier with a sigmoid activation). As we saw in the labs, the simple neuron (1) is limited to linearly separable data. As soon as the data is more complex, we have to turn to more advanced models. We can keep the approach of section (1) but replace the single neuron $\sigma(w_0 + \sum_{j=1}^{D} w_j x_j)$ by a network which will be able to capture more complex patterns from the data. The general form of a one hidden neural network is the following

$$y(\boldsymbol{x}; \boldsymbol{w}) = \sigma\left(w_0^{(2)} + \sum_{j=1}^{N_2} w_j^{(2)} \sigma\left(w_0^{(1)} + \sum_{k=1}^{N_1} w_{jk}^{(1)} x_k\right)\right) \tag{11}$$

If we still work on a classification model, we can still view the more sophisticated model (1) as encoding the probability that a particular example $\boldsymbol{x}^{(i)}$ will belong to class $\mathcal{C}_1$ or $\mathcal{C}_0$. Consequently, we can substitute this model in the log loss (10) and learn the weights through gradient or stochastic gradient descent. Now the difficulty with (1) is that the gradient (especially for a large number of layers) can be considerably more difficult to derive. Fortunately we can rely on a particular efficient approach known as back propagation which derive the derivatives recursively, starting from the output unit. The idea can be summarized as follows.

We can introduce the notation $a_i^{(\ell)}$ to denote the sum

$$a_i^{(\ell)} = \sum_j w_{ij}^{(\ell)} z_j^{(\ell-1)} \tag{12}$$

and use $z_j^{(\ell-1)}$ to denote the output to the j^{th} unit in layer ℓ

$$z_j^{(\ell-1)} = \sigma(a_j^{(\ell-1)}) = \sigma(\sum_k w_{jk}^{(\ell-1)} z_k^{(\ell-2)}) \tag{13}$$

For any weight $w_{ij}^{(\ell)}$ the derivative of the log loss can read as

$$\frac{\partial L}{\partial w_{ij}^{(\ell)}} = \frac{\partial L}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial w_{ij}^{(\ell)}} \tag{14}$$

we then let $\delta_j^{(\ell)}$ to denote the first factor $\delta_j^{(\ell)} = \frac{\partial L}{\partial a_j^{(\ell)}}$. For the second factor, it is easy to see from (12) that

$$\frac{\partial a_i^{(\ell)}}{\partial w_{ij}^{(\ell)}} = z_j^{(\ell-1)} \tag{15}$$

That is to say once we have all $\delta_j^{(\ell)}$ and $z_j^{(\ell)}$, we can compute all gradients. For the log loss, and for a single example $\{\boldsymbol{x}^{(i)}, t^{(i)}\}$ (stochastic gradient framework) one can check that the last δ, $\delta^{\text{out}} = \frac{\partial L}{\partial a^{\text{out}}}$ is given by $\delta^{\text{out}} = y(\boldsymbol{x}^{(i)}) - t^{(i)}$

For all the other delta's, we use the chain rule and write

$$\delta_j^{(\ell-1)} = \sum_k \frac{\partial L}{\partial a_k^{(\ell)}} \frac{\partial a_k^{(\ell)}}{\partial a_j^{(\ell-1)}} \tag{16}$$

The sum comes from the fact that any particular unit from layer $\ell - 1$ appears in all the units of the next layer (see Fig. 1 below).

From (16) we thus get $\delta^{(\ell-1)}$ from the $\delta_k^{(\ell)} = \frac{\partial L}{\partial a_k^{(\ell)}}$, as soon as we can express the second factor $\frac{\partial a_k^{(\ell)}}{\partial a_j^{(\ell-1)}}$. Yet from (12), (13) we can write

$$a_k^{(\ell)} = \sum_{j=1}^{N_\ell} w_{kj}^{(\ell)} \sigma(a_j^{(\ell-1)}) \tag{17}$$

and the second factor in (16) can thus read as

$$\frac{\partial a_k^{(\ell)}}{\partial a_j^{(\ell-1)}} = w_{kj}^{(\ell)} \sigma'(a_j^{(\ell-1)}) \tag{18}$$

Substituting this expression in (16), we then get

$$\delta_j^{(\ell-1)} = \sigma'(a_j^{(\ell-1)}) \sum_k \delta_k^{(\ell)} w_{kj}^{(\ell)} \tag{19}$$

In short, all the gradient can then be computed through the following steps

- Apply an input vector $\boldsymbol{x}^{(i)}$ to the network on compute all the values $a_i^{(\ell)}$ and $z_i^{(\ell)}$ for every layer
- Evaluate the δ_{out} for the output layer
- Backpropagate the δ_k using (19) in order to derive all $\delta_k^{(\ell)}$ for every ℓ
- Evaluate the gradients using (14).

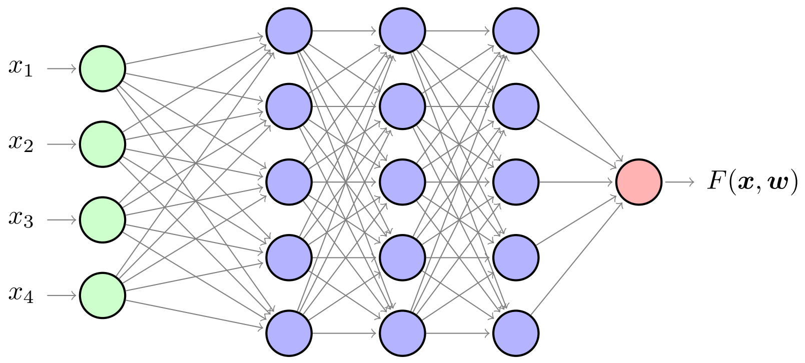

Figure 1: Feedforward Neural Network

Once we have the derivatives, we can apply the gradient step as

$$w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta \frac{\partial L}{\partial w_{ij}^{(\ell)}} \tag{20}$$

$$\vec{\boldsymbol{w}} \leftarrow \vec{\boldsymbol{w}} - \eta \boldsymbol{\nabla} L \tag{21}$$

where the gradient is obtained by stacking the derivatives of the log loss with respect to all the weights

$$\boldsymbol{\nabla} L = \mathrm{grad}_{\boldsymbol{w}} L = \left[\frac{\partial L}{\partial w_{10}^{(1)}}, \frac{\partial L}{\partial w_{11}^{(1)}} \cdots \frac{\partial L}{\partial w_{ij}^{(\ell)}}, \cdots \frac{\partial L}{\partial w_{jk}^{L_{\mathrm{out}}}} \right] \tag{22}$$