Lecture Notes 02: Feed-forward networks

Tudor Berariu
tudor.berariu@gmail.com

October 12, 2016

1 Feed forward networks

Feed forward networks (also called *multi-layer perceptrons*) represent the fundamental neural architecture. Its computational graph is a directed acyclic graph that processes information from some input $\boldsymbol{x}$ to an output $\boldsymbol{y}$. The information flows without any feedback loops.

$$\boldsymbol{y} = f_{\text{net}}(\boldsymbol{x}, \boldsymbol{\theta})$$

Feed forward networks represent a composition of functions that are usually called *layers*.

$$\boldsymbol{y} = f_L(f_{L-1}(\dots f_1(\boldsymbol{x}, \boldsymbol{\theta}_1)\dots, \boldsymbol{\theta}_{L-1}), \boldsymbol{\theta}_L)$$

2 Cost functions

2.1 Regression

In regression tasks the data set comprises of pairs of inputs and correct outputs which are real values. The usual cost function is the Mean Squared Error.

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n}^{N} \left| \boldsymbol{y}\left(\boldsymbol{x}^{(n)}, \boldsymbol{\theta}\right) - \boldsymbol{t}^{(n)} \right|_2^2 = \sum_{n}^{N} \sum_{k}^{K} \left(y_k\left(\boldsymbol{x}^{(n)}, \boldsymbol{\theta}\right) - {t_k}^{(n)} \right)^2$$

The derivative of the cost function w.r.t. $\boldsymbol{y}$ is:

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{y}} = \sum_{n} \boldsymbol{y}\left(\boldsymbol{x}^{(n)}, \boldsymbol{\theta}\right) - \boldsymbol{t}^{(n)}$$

2.2 Classification

For classification tasks the targets are one-hot encoded vectors with the single 1 value corresponding to the correct class.

One way to compute a classification model is to train a neural network to compute a posterior probability distribution over the classes:

$$y_k(\boldsymbol{x}, \boldsymbol{\theta}) \approx P(\mathcal{C}_k | \boldsymbol{x})$$

Given a data set, our goal is to find the parameters $\boldsymbol{\theta}^*$that maximize the probability of the examples being in the correct class.

$$P(\boldsymbol{\theta}) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_k \left(\boldsymbol{x}^{(n)}, \boldsymbol{\theta} \right)^{t_k{}^{(n)}}$$

Maximizing $P(\boldsymbol{\theta})$ is equivalent to minimizing its negative logarithm.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \log(P(\boldsymbol{\theta})) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \sum_{n=1}^{N} \sum_{k=1}^{K} t_k{}^{(n)} \log\left(y_k \left(\boldsymbol{x}^{(n)}, \boldsymbol{\theta} \right) \right)$$

The derivative of the cross-entropy function w.r.t the outputs is given by this formula:

$$\boldsymbol{\delta}_{Lk} = \frac{\partial \mathcal{L}}{\partial y_k} = -\frac{t_k}{y_k} = \begin{cases} 0 & \text{, if } t_k = 0 \\ -1/y_k & \text{, if } t_k = 1 \end{cases} \tag{1}$$

3 Layers

The classic feed-forward networks alternate fully-connected linear layers with non-linear transfer functions (e.g. logisitic, hyperbolic tangent). For regression problems the last layer does not need to go through a squash function. For classification tasks, the last layer is usually a softmax one, forcing the network to approximate a probability distribution.

3.0.1 The fully connected layer

Calculul ieirilor A fully-connected layer computes a projection of an input vector $\boldsymbol{x} \in \mathbb{R}^D$ into an output space $\mathbb{R}^K$. Forumla 2 describes the computation for a single output unit y_k.

$$y_k = \sum_{i=1}^{D} \theta_{ki} x_i + b_k, \qquad \forall k \in \{1, \ldots, K\} \tag{2}$$

Formula 2 can be written in matrix form as in Formula 3.

$$\boldsymbol{y} = \boldsymbol{\Theta} \boldsymbol{x} + \boldsymbol{b} \tag{3}$$

The matrix $\mathbf{x}\Theta \in \mathbb{R}^{K \times D}$ and the vector $\mathbf{b} \in \mathbb{R}^K$ are the parameters of the layer.

Error backpropagation In the error backpropagation phase, the partial derivatives of the loss function with respect to the inputs $\vec{[x]}$ are being computed given $\boldsymbol{\delta}_y = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}$. The computation follows Formula 4 or, the equivalent matrix expression in Formula 5.

$$\frac{\partial \mathcal{L}}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_i} = \sum_{k=1}^{K} \delta_{y_k} \theta_{ki} = \boldsymbol{\delta}_y^{\mathrm{T}} \boldsymbol{\theta}_i \tag{4}$$

$$\frac{\partial E}{\partial \mathbf{x}} = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\delta}_y \tag{5}$$

In a similar fashion the gradient of the loss function with respect to the parameters $\mathbf{x}\Theta$ i $\boldsymbol{b}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Theta}} = \boldsymbol{\delta}_y \boldsymbol{x}^{\mathrm{T}} \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \boldsymbol{\delta}_y \tag{7}$$

3.0.2 Tanh

A `TanH` layer applies the hyperbolic tangent function element wise on an input vector.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (8) \qquad\qquad tanh'(x) = 1 - tanh(x)^2 \quad (9)$$

Computing outputs

$$y_i = tanh(x_i) \qquad \forall i \tag{10}$$

Backpropagating gradients Working on Formula 9, the vector $\boldsymbol{\delta}_x$ can be computed using element-wise multiplication $\odot$:

$$\boldsymbol{\delta}_x = (\mathbf{1} - \boldsymbol{y} \odot \boldsymbol{y}) \odot \boldsymbol{\delta}_y \tag{11}$$

A `TanH` layer has no parameters.

3.0.3 SoftMax

Computing outputs The `SoftMax` layer is used in classification problems where the goal is to compute one-hot encoded output representations. The outputs are interpreted as a posterior distribution over the set of classes. Given some input vector $\boldsymbol{x} \in \mathbb{R}^K$, the output vector $\boldsymbol{y} \in \mathbb{R}^K$ is computed as inFormula 12.

$$y_k = \frac{e^{x_k}}{\sum_{k'=1}^{N} e^{x_{k'}}} \tag{12}$$

Propagarea erorilor Since the `SoftMax` layer has no parameters, in the backward phase only $\boldsymbol{\delta}_x$ needs to be computed as a function of $\boldsymbol{\delta}_y$. Formula 13 describes this relation. For a mathematical proof, go to Section A.

$$\delta_{x\,k} = y_k \left(\delta_{y\,k} - Z\right) \tag{13}$$

where $Z = \sum_{k'} \delta_{y\,k'} y_{k'}$.

4 The forward phase

Algorithm 1 The forward phase

1: **procedure** FORWARD(net, $\mathbf{x}$) ▷ Inputs: a FFN net, and an example $\mathbf{x}$
2: $\quad \mathbf{y}_0 \leftarrow \mathbf{x}$
3: $\quad$ **for** $l \leftarrow 1 \dots L$ **do**
4: $\quad\quad \mathbf{y}_l \leftarrow net.layers[l].forward(\mathbf{y}_{l-1})$ ▷ l^{th} layer's input are $(l-1)^{\text{th}}$ layers's outputs
5: $\quad$ **return** $\mathbf{y}_L$ ▷ Return the last layer's outputs.

5 The backward phase

Algorithm 2 Backpropagation of gradients through the network

1: **procedure** BACKPROPAGATE($net, \mathbf{x}, \boldsymbol{\delta}_L$)
2: $\quad \boldsymbol{y}_0 \leftarrow \mathbf{x}$
3: $\quad$ **for** $l \leftarrow L \dots 1$ **do**
4: $\quad\quad \boldsymbol{\delta}_{l-1} \leftarrow net.layers[l].backward(\mathbf{y}_{l-1}, \boldsymbol{\delta}_l)$ ▷ Gradients are accumulated internally $\frac{\partial E}{\partial \boldsymbol{\theta}_l}$

6 Training the network

Parameters are updated using stochastic gradient descent:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\partial \mathcal{L}\left(\boldsymbol{\theta}^{(t)}\right)}{\partial \boldsymbol{\theta}^{(t)}} \tag{14}$$

Algorithm 3 Stochastic gradient descent

procedure SGD(net, $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{t}^{(n)})\}_{1 \leq n \leq N}, \eta$) ▷ net, the data set, the learning rate
    repeat
        $\mathcal{B} \leftarrow$ a *mini-batch* of size b from $\mathcal{D}$
        for $l \leftarrow 1 \dots L$ **do**
            $y_l \leftarrow net.layers[l].zeroGradients()$ ▷ Set gradients to zero
        for $(\mathbf{x}, \mathbf{t}) \leftarrow \mathcal{B}$ **do** ▷ For each example in the *mini-batch*
            $\mathbf{y} \leftarrow forward(net, \mathbf{x})$ ▷ Compute the net's outputs
            $\boldsymbol{\delta}_L \leftarrow \frac{\partial E}{\partial \mathbf{y}}$ ▷ The gradient of the loss w.r.t. the outputs
            $backpropagate(net, \mathbf{x}, \boldsymbol{\delta}_L)$ ▷ The errors are backpropagated through the network
        for $l \leftarrow 1 \dots L$ **do**
            $net.layers[l].updateParameters(\eta)$ ▷ The parameters are updated
    until convergence

A Backpropagating errors through a `SoftMax` layer

Consider the `SoftMax` function which transforms an input vector $\boldsymbol{x}$ into an output vector $\boldsymbol{y}$ with the same dimension which represents a proability distribution (Formulae 15,16).

$$\mathbf{y} = softmax(\boldsymbol{x}) \tag{15}$$

$$y_k = \frac{e^{x_k}}{\sum_{k'} e^{x_{k'}}} \tag{16}$$

Given the gradient of the loss function $\mathcal{L}$ w.r.t. $\boldsymbol{y}$, we need to find the expression of the gradient w.r.t. the inputs $\boldsymbol{x}$.

$$\boldsymbol{\delta}_y \overset{not.}{=} \frac{\partial E}{\partial \mathbf{y}} \tag{17}$$

$$\boldsymbol{\delta}_x \overset{not.}{=} \frac{\partial E}{\partial \mathbf{x}} \tag{18}$$

Formula 19 describes the computation of a single component of $\boldsymbol{\delta}_x$.

$$\delta_{x_k} = \sum_{k'} \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial x_k} = \sum_{k'} \delta_{y_{k'}} \frac{\partial y_{k'}}{\partial x_k} \tag{19}$$

$$\frac{\partial y_{k'}}{\partial x_k} = \frac{\mathbb{I}(k == k') e^{x_k} \left(\sum_{k''} e^{x_{k''}}\right) - e^{x_{k'}} e^{x_k}}{\left(\sum_{k''} e^{x_{k''}}\right)^2} = \begin{cases} y_k - y_k y_{k'} & \text{, daca } k == k' \\ -y_k y_{k'} & \text{, daca } k \neq k' \end{cases} \tag{20}$$

By using Formula 20 in Formula 19:

$$\delta_{x_k} = \sum_{k'} \delta_{y_{k'}} \frac{\partial y_{k'}}{\partial x_k} = y_k \left(\delta_{y_k} - \sum_{k'} \delta_{y_{k'}} y_{k'} \right) = y_k \left(\delta_{y_k} - Z \right) \tag{21}$$

where $Z = \sum_{k'} \delta_{y_{k'}} y_{k'}$.