Neural Network Basics

Daniel Hug

August and September 2023

1 Neural Net Basics

Sources:

- http://neuralnetworksanddeeplearning.com
- https://brilliant.org/wiki/feedforward-neural-networks/#formal-definition
- https://brilliant.org/wiki/backpropagation/

1.1 Neural Net Definitions

Here we look into a neural network based on perceptron cells with an activation function. A single perception can be regarded as a linear classifier, i.e. it is able to output which of two linearly separable data sets a given point belongs to. In order to classify more complex data sets, layers of perceptrons with non-linear activation functions are required.

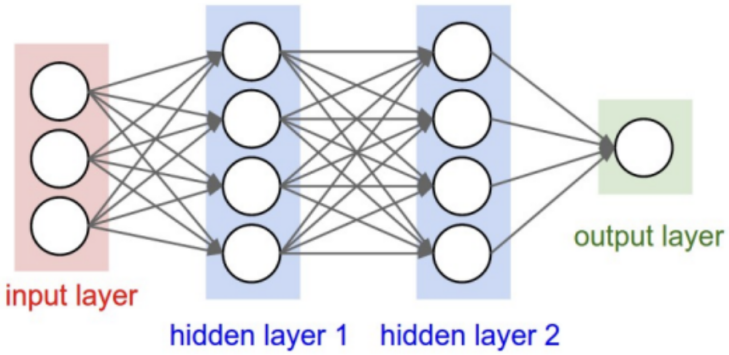

Figure 1: A fully connected multi-layer neural network with three inputs, two hidden layers, each with four perceptron nodes and an output layer with a single output node. (Source: https://towardsdatascience.com)

A neural net consists of L layers with n^l nodes in each layer $l = [0, L-1]$. For the input layer ($l = 0$) the nodes are simple input nodes, while for the remaining layers consist of perceptron nodes (see figure 2).

A perceptron with an arbitrary activation function looks like this:

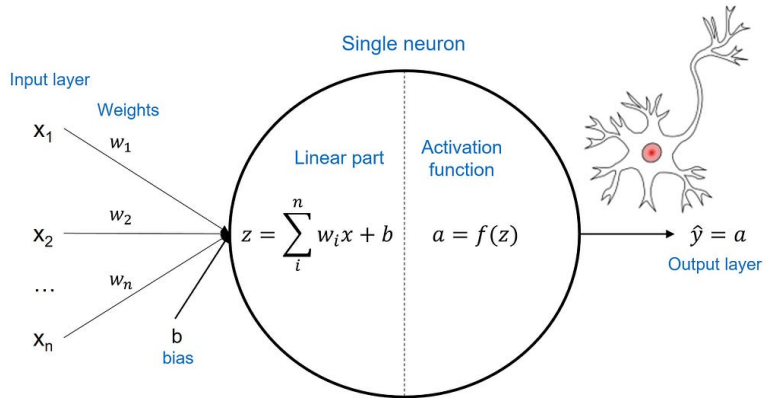

Figure 2: A node of a neural network with inputs, bias and activiation function. (Source: https://towardsai.net)

Definitions:

1. We use a training dataset X consisting of input-output pairs $(\vec{x}_i, \vec{y}_i)$, where $\vec{x}_i$ is the input and $\vec{y}_i$ is the desired or target output of the network on the input $\vec{x}_i$ ($\vec{y}_i$ is a label). The set of input-output training pairs of size N is denoted $X = \{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_N, \vec{y}_N)\}$.

2. A neural network has learning parameters, weights and biases, that are collectivly denoted θ. The nodes between different layers are fully connected, while there are no connections between nodes within the same layer. For backpropagation the parameters of primary interest are the weights w^l_{tf}, defining the weight[1] between the node with index t in layer l and the node with index f in layer $l-1$, and b^l_t, defining the bias of the node with index t in layer l.

3. A loss function $L(X, \theta)$, which defines a quantitative measure for the error between the desired output $\vec{y}_i$ and the calculated actual output $\vec{a}^l_i$ of the neural network on the input $\vec{x}_i$ for a given input pair $(\vec{x}_i, \vec{y}_i) \in X$ and a particular value of the learning parameters θ.

Following values are used to define the network:

- x_t, the input at index t of an input vector $\vec{x}_i$ of the training set in the input layer ($x_t = a^{l=0}_t$ after activation of the input layer in the implementation, see below for definition of a^l_t).

- w^l_{tf}, the weight between the node with index t ($\underline{t}o$) in layer l and the node with index f ($\underline{f}rom$) in layer $l-1$.

- b^l_t, the bias of the node with index t in layer l.

- $z^l_t = \sum_f w^l_{tf} a^{l-1}_f + b^l_t$ is the total input the node gets from the activated nodes of the previous layer and it's bias.

- Activation functions f_h for the $\underline{h}$idden layers and the $\underline{o}$utput layer f_o.

- The activation a^l_t at index t in layer l: $a^l_t = f_h(z^l_t)$ in hidden layers or $a^{L-1}_t = f_o(z^{L-1}_t)$ for the output layer. The activation function for the input layer is the identity function in the implementation.

- a^{L-1}_t is the output component at the index t in the output layer at the given input $\vec{x}_i$. The intended output is defined by $\vec{y}_i$ as second part of the training pair $(\vec{x}_i, \vec{y}_i)$, where y_t is the component t of $\vec{y}_i$.

[1] In this notation the index t stands for *to*, whereas f stands for *from*. Index t refers to nodes in layer l, while index f refers to nodes in layer $l-1$ for the forward pass.

The goal of training is to minimize the loss function and thus to minimize the remaining error for a given set of training data. A typical example of a loss function is the quadratic cost function (Mean Squared Error (MSE) with additional factor of $\frac{1}{2}$ for easy calculation of its derivative). The total loss is an average of the N partial losses L_i for each training pair $(\vec{x}_i, \vec{y}_i)$ as shown in equation (1), where the partial loss itself is a sum over the contributions of each component t of the difference between the actual output vector of the network $\vec{a}_i^{L-1}$ and the intended output vector $\vec{y}_i$ for a given training pair i.

$$L(X,\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|\vec{a}_i^{L-1} - \vec{y}_i\|^2 = \frac{1}{N}\sum_{i=1}^{N}\underbrace{\sum_t \frac{1}{2}(a_t^{L-1} - y_t)^2}_{L_i} \tag{1}$$

We need to understand how the deviation of the actual output to the desired output depends on the choice of learning parameters θ, i.e. the weights w and the bias values b. In order to minimize the loss function, and thus the deviation between actual and desired output, gradient descent is used. This means that the training parameters have to be changed in a direction against the gradient of the function L. This requires to calculate all partial derivatives of the loss function with respect to the weights $\frac{\partial L}{\partial w}$ and to the biases $\frac{\partial L}{\partial b}$ in order to update them with η as learn rate according to

$$w^{new} = w^{old} - \eta\frac{\partial L}{\partial w} \tag{2a}$$

$$b^{new} = b^{old} - \eta\frac{\partial L}{\partial b} \tag{2b}$$

To understand why this is the case, let's consider an arbitrary cost function with three parameters $C(w_1, w_2, b)$ as an example. For small changes we get:

$$\Delta C \approx \frac{\partial C}{\partial w_1}\Delta w_1 + \frac{\partial C}{\partial w_2}\Delta w_2 + \frac{\partial C}{\partial b}\Delta b \tag{3}$$

So $\Delta C \approx \vec{\nabla} C \cdot \Delta\vec{v}$ with $\vec{\nabla} C = (\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \frac{\partial C}{\partial b})^T$ and $\Delta\vec{v} = (\Delta w_1, \Delta w_2, \Delta b)^T$.

To minimize C, we need to assure that $\Delta C < 0$.

If we specifically choose $\Delta\vec{v} = -\eta\vec{\nabla} C$, i.e. select the change of $\Delta\vec{v}$ against the direction of the gradient vector, we get

$$\Delta C \approx \vec{\nabla} C \cdot (-\eta\vec{\nabla} C) = -\eta\vec{\nabla} C \cdot \vec{\nabla} C = -\eta\|\vec{\nabla} C\|^2 \tag{4}$$

which assures a negative ΔC, because $\|\vec{\nabla} C\|^2$ is a positive value.

In practise we can either calculate the gradient for the full training set, or rather approximate the gradient by selecting a random subset of the training samples (called a mini-batch), or directly calculate the gradient we get for each individual training sample pair (called stochastic gradient descent). The resulting gradient is applied according to equations (2a) and (2b) to update the weights and biases. The approaches of gradient approximation are mainly used to reduce the computational effort.

It is called an epoch, when the neural net has seen all available training data once during a training cycle. The training typically covers several hundred epochs for minimizing the loss function.

The available data is normally split into training data, validation data, test data in a ratio of 70:20:10. The training data set is used to learn the training parameters. The validation data is used for initially optimizing the meta parameters (e.g. learning rate, batch sizes, activation functions, etc.). The test data set is used to calculate the quality of the training as a result (precision, recall, accuracy). All subsets must of course be large enough to be representative for the full available data set or we run into trouble due to biased data in the subsets.

The learning or training cylce for a neural network looks like this:

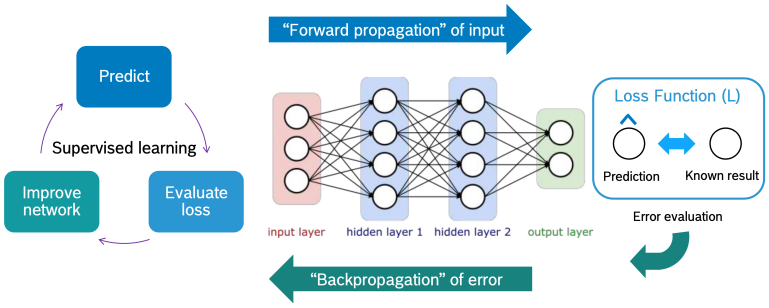

Figure 3: Training or learning cycle of a neural network. (Source: Bosch AIMEX training)

TODO: *fill in prodedure to compute accuracy, precision, etc. here*

1.2 Neural Net Computations

Initialize the network

- Set up the network structure by defining:
  - total number of layers L
  - nodes per layer
  - activation functions for hidden and output layers
- Initialize the weights w and biases b with random values.
- Read the training data and the corresponding labels.

Forward pass

1. Make a_t^0 available for every t in the input layer ($l = 0$) for a given training input $\vec{x}_i$.
2. Compute the activation $z_t^l = \sum_f w_{tf}^l a_f^{l-1} + b_t^l$ for every node in a forward pass for each t in each layer $l = [1, L-1]$ starting from $l = 1$.
3. Make the output for each layer available by computing $a_t^l = f_h(z_t^l)$ for the hidden layers and $a_t^l = f_o(z_t^l)$ for the output layer.
4. Calculate the partial loss L_i for the given training pair $(\vec{x}_i, \vec{y}_i)$ and the calculated output from the forward pass.

Loss function derivatives

The total loss $L = \frac{1}{N} \sum_i^N L_i$ is an average of the partial losses L_i for the given training pairs $(\vec{x}_i, \vec{y}_i)$. To calculate how the loss function depends on the chosen learning parameters, we use the chain rule:

$$\frac{\partial L_i}{\partial w_{tf}^l} = \underbrace{\frac{\partial L_i}{\partial z_t^l}}_{\equiv \delta_t^l} \frac{\partial z_t^l}{\partial w_{tf}^l} = \delta_t^l \frac{\partial z_t^l}{\partial w_{tf}^l} \tag{5a}$$

$$\frac{\partial L_i}{\partial b_t^l} = \underbrace{\frac{\partial L_i}{\partial z_t^l}}_{\equiv \delta_t^l} \frac{\partial z_t^l}{\partial b_t^l} = \delta_t^l \frac{\partial z_t^l}{\partial b_t^l} \tag{5b}$$

$\delta_t^l = \frac{\partial L_i}{\partial z_t^l}$ is called the "error term" due to the fact that it is directly linked to the deviation between actual and computed output of the neural net in the forward pass.

Using the definition of $z_t^l = \sum_f w_{tf}^l a_f^{l-1} + b_t^l$ from the previous section we get

$$\frac{\partial z_t^l}{\partial w_{tf}^l} = \frac{\partial}{\partial w_{tf}^l}\left(\sum_f w_{tf}^l a_f^{l-1} + b_t^l\right) = a_f^{l-1} \tag{6a}$$

$$\frac{\partial z_t^l}{\partial b_t^l} = \frac{\partial}{\partial b_t^l}\left(\sum_f w_{tf}^l a_f^{l-1} + b_t^l\right) = 1 \tag{6b}$$

where only one term remains of the sum for specific values for t and f in (6a).

Combining the previous equations we get as a result

$$\frac{\partial L_i}{\partial w_{tf}^l} = \delta_t^l \cdot a_f^{l-1} \tag{7a}$$

$$\frac{\partial L_i}{\partial b_t^l} = \delta_t^l \tag{7b}$$

Backpropagation - output layer ($l = L - 1$):

With

$$L_i = L_i(\vec{x}_i, \vec{y}_i) = \sum_t L_i(a_t^l, y_t) = \sum_t L_i\left(f_o(z_t^l), y_t\right) \tag{8}$$

and using the definition of the error term δ_t^l from above we get

$$\delta_t^l \equiv \frac{\partial L_i}{\partial z_t^l} = \sum_t \frac{\partial L_i}{\partial f_o}\left(f_o(z_t^l), y_t\right) \cdot \frac{\partial f_o}{\partial z_t^l} = \sum_t L_i'\left(f_o(z_t^l), y_t\right) \cdot f_o'(z_t^l) \tag{9}$$

which, when inserting into equations (7a) and (7b), yields

$$\frac{\partial L_i}{\partial w_{tf}^l} = \delta_t^l \cdot a_f^{l-1} = \sum_t L_i'\left(f_o(z_t^l), y_t\right) \cdot f_o'(z_t^l) \cdot a_f^{l-1} \tag{10a}$$

$$\frac{\partial L_i}{\partial b_t^l} = \delta_t^l = \sum_t L_i'\left(f_o(z_t^l), y_t\right) \cdot f_o'(z_t^l) \tag{10b}$$

These formulas will be directly coded in during the backwards pass during the training cycle to calculate the gradients for learning.

Backpropagation - hidden layers ($1 \leq l \leq L-2$):

For hidden layers we try to link the local contribution to the loss in layer l with the contribution to the loss in a layer $l+1$. Like in forward pass we have to refer to different layers with in the formulas to link them[2].

Looking at the error term and using the chain rule again we can find a link between different layers:

$$\delta_f^l = \frac{\partial L_i}{\partial z_f^l} = \sum_t \underbrace{\frac{\partial L_i}{\partial z_t^{l+1}}}_{=\delta_t^{l+1}} \frac{\partial z_t^{l+1}}{\partial z_f^l} = \sum_t \delta_t^{l+1} \frac{\partial z_t^{l+1}}{\partial z_f^l} \tag{11}$$

Using the definition of z_t^{l+1} we get

$$z_t^{l+1} = \sum_f \left(w_{ft}^{l+1} a_f^l\right) + b_t^l = \sum_f \left(w_{ft}^{l+1} f_h(z_f^l)\right) + b_t^l \tag{12}$$

This leads to

$$\frac{\partial z_t^{l+1}}{\partial z_f^l} = \frac{\partial}{\partial z_f^l}\left(\sum_f \left(w_{ft}^{l+1} f_h(z_f^l)\right) + b_t^l\right) = w_{ft}^{l+1} \frac{\partial f_h(z_f^l)}{\partial z_f^l} = w_{ft}^{l+1} f_h'(z_f^l) \tag{13}$$

because there only remains the one term related to the specific index we derive for of the sum.

Inserting equation (13) into equation (11) we get a formula for the error term in a hidden layer l

$$\delta_f^l = \sum_t \delta_t^{l+1} w_{ft}^{l+1} f_h'(z_f^l) = f_h'(z_f^l) \sum_t \delta_t^{l+1} w_{ft}^{l+1} \tag{14}$$

which will be used to update the error term in the hidden layers during the backward pass. This is called backpropagation, because the error contributions are spread layer by layer thoughout the neural network beginning from the output layer $l = L-1$ down to the layer $l = 1$.

Looking at eqations (7a) and (14) we can see how the network layers $l-1$, l and $l+1$ are linked to each other.

[2] In the notation we use here, again the index t stands for *to*, whereas f stands for *from*. However, index t here refers to nodes in layer $l+1$, while index f refers to nodes in layer l for the backward pass. This is conceptually the same use as for the forward pass, but with reference to other layers.