Deep Learning Lecture 3: Multi-Layer Perceptron

Abdelrahman Khaled

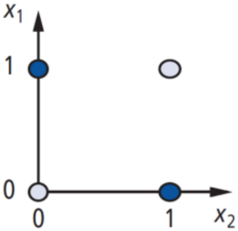
Machine Learning Research Cluster
German University in Cairo

April 8, 2019

Outline

Solving the XOR Problem

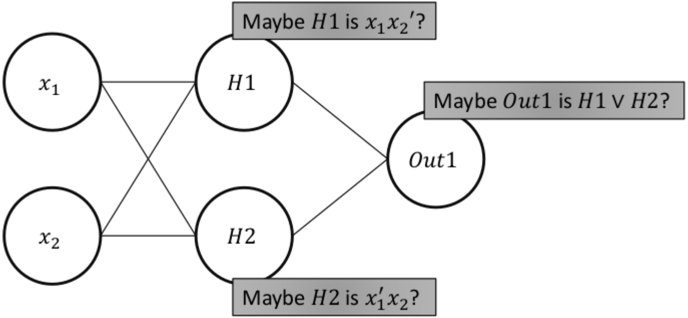
$x_1 \otimes x_2$



$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

$x_1 x_2' + x_1' x_2$

Solving the XOR Problem



Matrix Math

Let $C = A + B$

- A and B must have the same size (same number of rows and columns).
- C is the same size as A and B.
- Every element in C is the result of adding the two elements from the same row and column together.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_{A_{2\times 2}} + \underbrace{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}}_{B_{2\times 2}} = \underbrace{\begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}}_{C_{2\times 2}}$$

Matrix Math

Let $C = AB$.

- Order matters ($AB \neq BA$)
- A must have the same number of columns as B has rows.
- Multiply each element in one row of the first matrix with each corresponding element in one column of the second matrix and add them together.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}}_{A_{2\times 3}} \times \underbrace{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}}_{B_{3\times 2}} =$$

$$\underbrace{\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}}_{C_{2\times 2}}$$

Matrix Math

Let $A^{\mathbf{T}}$ be the transpose of the Matrix A.

- Swap the element from row i column j to row j column i.
- If an element has been previously swapped don't swap it again.

$$\underset{A_{2\times 2}}{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}} \longrightarrow \underset{A^{\mathbf{T}}_{2\times 2}}{\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}}$$

Neuron Activation

We can say that each neuron (node) j at any layer in a neural network stores a number given by:

$$h_j = \sum_{i=1}^{N} w_{ij} x_i$$

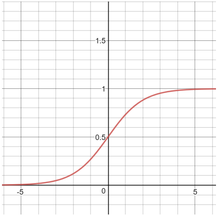
where N is the number of neurons in the previous layer and x_i is the value stored at neuron number i in the previous layer.

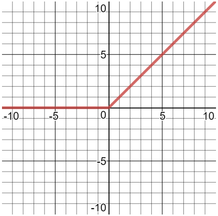
We can use the same thresholding activation function for each neuron, but we'd rather use a continuous function than a discrete one.

Activation Functions

An activation function determines the output of a neuron in a neural network.

- None linear activation functions can produce non-linear mappings between the input and output.
- Different layers can have different activation functions, but it standard that the same layer can only have one activation function.





Feed Forward

- The process by which information travels from the input to the output of a network.
- The weighted sum is calculated for each node in the first layer, and then the activation function is applied to it.
- The result becomes the input for the next layer.
- This continues till the final layer is reached.

Matrix Representation

Let any layer be a column vector X, the set of weights to be a matrix W, the biases be a column vector B and the activation function be a.

The output Y is:

$$Y = a(WX + B)$$

The Error Function

- A function that indicates how well a network is doing.
- The smaller the error the better the performance of the network.

Let E be the error function such that E is the squared difference between all predictions y with actual values t.

$$E = \frac{1}{2} \sum_{i=1}^{N} (t_i - y_i)^2$$

- To train the network, we need to minimize this error.
- Derivatives help us with that.

Gradient Descent

- A derivative calculates the rate of change of a single variable function.
- Gradients are vectors calculated from the rate of change of multi-valued functions
  - We obtain a gradient by partially differentiating a multi-variable function along each dimension.
  - Gradients always point towards the direction of steepest ascent.
  - Consequently, their negative points towards the steepest descent.

Gradient Descent

- Gradient Descent is the process in which we use the gradient of a function to find a local minimum of that function.

$$\nabla E = \langle \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_N} \rangle$$

- To find a local minimum we need to slightly alter all the weights such that they move towards the direction of the negative of the gradient.

$$\Delta w_i \propto -\frac{\partial E}{\partial w_i}$$

Backpropagation

- Shorthand for backward propagation of errors.
- When an error is calculated, it is only calculated at the output layer, so how do we fix the weights of the layers leading up to it?
  - We propagate the error backwards through the network till we reach the weights leaving the input layer.
- We can use the chain rule for all the partial derivatives leading to a weight that needs to be updated.

For simplicity

Let the number of nodes in the input layer be I and each node in that layer is labeled i where $i \in \{1, 2, ..., I\}$.
Let the hidden and output layers be described similarly, but using J, j and K, k respectively.

Backpropagation

For our XOR example, we have two sets of weights.

1. Those going from the input layer to the hidden layer (labeled w_{ij} indicating the weight going from neuron i in the input layer to neuron j in the hidden layer)

$$\frac{\partial E}{\partial w_{ij}} = [\sum_k \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_k} \cdot \frac{\partial h_k}{\partial y_j}] \cdot \frac{\partial y_j}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{ij}}$$

2. Those going from the hidden layer to the output layer (labeled w_{jk} indicating the weight going from neuron j in the hidden layer to neuron k in the output layer)

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_k} \cdot \frac{\partial h_k}{\partial w_{jk}}$$

Gradient Descent

Recall the following formulas:

The error at the output layer

$$E = \frac{1}{2}\sum_{i=1}^{N}(t_i - y_i)^2$$

The activation at a node k in the output layer.

$$y_k = \frac{1}{1 + e^{-h_k}}$$

The weighted sum at a node k in the output layer.

$$h_k = \sum_{j=1}^{J} w_{jk} x_j$$

Gradient Descent

$$
\begin{aligned}
\frac{\partial E}{\partial y_k} &= \frac{\partial(\frac{1}{2}\sum_{i=1}^{N}(t_i - y_i)^2)}{\partial y_k} \\
&= \frac{1}{2}\frac{\partial((t_1 - y_1)^2 + \ldots + (t_k - y_k)^2 + \ldots + (t_N - y_N)^2)}{\partial y_k} \\
&= \frac{1}{2}\frac{\partial((t_k - y_k)^2)}{\partial y_k} \\
&= -(t_k - y_k)
\end{aligned}
$$

Gradient Descent

$$
\begin{aligned}
\frac{\partial y_k}{\partial h_k} &= \frac{\partial(\frac{1}{1+e^{-h_k}})}{\partial h_k} \\
&= -\frac{1}{(1+e^{-h_k})^2} \cdot e^{-h_k} \cdot -1 \\
&= \frac{e^{-h_k}}{(1+e^{-h_k})^2} = \frac{e^{-h_k}+1-1}{(1+e^{-h_k})^2} \\
&= \frac{1+e^{-h_k}}{(1+e^{-h_k})^2} - \frac{1}{(1+e^{-h_k})^2} \\
&= \frac{1}{1+e^{-h_k}}(1 - \frac{1}{1+e^{-h_k}}) \\
&= y_k(1-y_k)
\end{aligned}
$$

Gradient Descent

$$
\begin{aligned}
\frac{\partial h_k}{\partial w_{jk}} &= \frac{\partial(\sum_{j=1}^{J} w_{jk}x_j)}{\partial w_{jk}} \\
&= \frac{\partial\left(w_{1k}x_1 + \ldots + w_{jk}x_j + \ldots + w_{Jk}x_J\right)}{\partial w_{jk}} \\
&= \frac{\partial\left(w_{jk}x_j\right)}{\partial w_{jk}} \\
&= x_j
\end{aligned}
$$

Overfitting

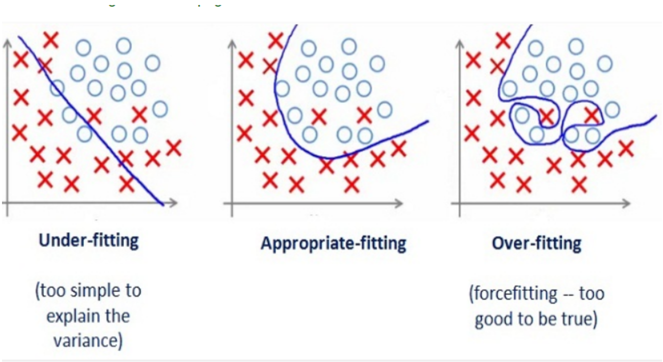


Figure: Image Source

Training VS Testing VS Validation

It is standard practice for any machine learning problem to split the dataset into three separate smaller datasets.

- Training dataset:
  The dataset on which the model trains. The model knows all the targets for all the datapoints and uses that information to better its approximations.
- Testing dataset:
  The dataset with which the model is evaluated. The model does not know any of the targets and its performance is then calculated depending on how wrong or right it was in approximating/classifying the data points in this dataset.

Training VS Testing VS Validation

- Validation dataset:
  This dataset is used in training, though it is not used to edit the weights. We use it to measure the accuracy/performance of our model while training in order to avoid overfitting and allow better regularization.

Programming Task

- Make a Multi-Layer Perceptron that uses a test set and validation set to improve and control training.
  1. Obtain or create a dataset of points to be classified into one or more classes.
  2. Use your model with different configurations on the dataset.
  3. Record your model's performance.
- **Optional**: Generalize the Multi-Layer Perceptron we made to have variable number of hidden layers.

Next Time

1. Neural Network tools: Keras