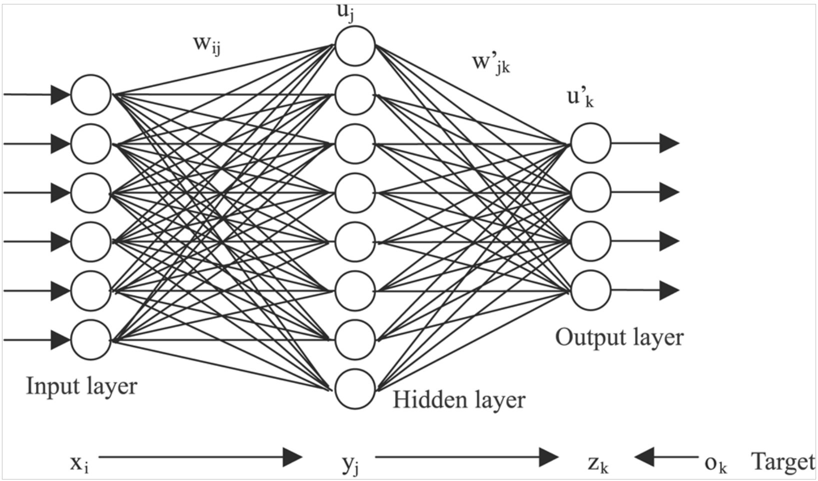
Artificial neural networks and their training algorihm

Tamás Grósz

Artificial Neural Networks



Artificial Neural Networks

Input Layer

The input layer of a neural network is composed of artificial input neurons, and brings the initial data into the system for further processing by subsequent layers of artificial neurons.

Artificial Neural Networks

Input Layer

The input layer of a neural network is composed of artificial input neurons, and brings the initial data into the system for further processing by subsequent layers of artificial neurons.

Hidden Layer(s)

The hidden layers' job is to transform the inputs into something that the output layer can use.

Artificial Neural Networks

Input Layer

The input layer of a neural network is composed of artificial input neurons, and brings the initial data into the system for further processing by subsequent layers of artificial neurons.

Hidden Layer(s)

The hidden layers' job is to transform the inputs into something that the output layer can use.

Output Layer

The output layer the last layer of neurons that produces the outputs/decisions. The output layer neurons may be observed, given that they are the last "actor" nodes in the network.

Multiclass learning

- One neuron can only separate 2 classes
- If we want to separate more than 2 classes we need more neurons in the outpu layer (one for each class)
- The class assosiated with the most active output neuron will be predicted
- The expected output (label) in this case is either the index of the class, or a one-hot vector (4 vs 00001000)

Multiclass learning

- As mentioned before, we have one output neuron for each class
- It would be good if output neurons could predict the $p(c_i|x)$
- To achieve this, we need to change the activation function of the ouptut neurons

Multiclass learning

- As mentioned before, we have one output neuron for each class
- It would be good if output neurons could predict the $p(c_i|x)$
- To achieve this, we need to change the activation function of the ouptut neurons

Softmax activation

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_j(e^{z_j})}$$

Using this function, we can view the output vector as a probability vector.

Cross-entropy loss function

- The MSE loss function is better suited for regression learning
- In case of classification, we need a different loss function that takes into account that we want to learn class probabilities

Cross-entropy loss function

- The MSE loss function is better suited for regression learning
- In case of classification, we need a different loss function that takes into account that we want to learn class probabilities

Cross-entropy

$$CE(\theta, y) = -\frac{1}{N} \sum_{n \leq N} \sum_{d \leq D} y_{nd} ln(o_{nd}),$$

where θ holds the parameters of the network, y is the correct label vector, o is the output vector, N is the number of examples and D is the dimension of the output layer.

The mathematical form of a neural network

- To understand the training algorithm, we need to formulate the neural network
- The whole network could be viewed as a function, applying matrix multiplications and the activation functions:
  - The parameters/weights of the ith layer will be denoted by W_i
  - The input for a hidden layer is either the input (X) or the output of a previous hidden layer
  - F_i denotes the activation function of the ith layer

The mathematical form of a neural network

- To understand the training algorithm, we need to formulate the neural network
- The whole network could be viewed as a function, applying matrix multiplications and the activation functions:
  - The parameters/weights of the ith layer will be denoted by W_i
  - The input for a hidden layer is either the input (X) or the output of a previous hidden layer
  - F_i denotes the activation function of the ith layer

Mathematical form of a NN with two hidden layer

$$NN(X) = F_2(W_2 * F_1(W_1 * F_0(W_0 * X)))$$

Backpropagation algorithm

- We can use the simple gradient descent algorithm to optimize the weights of the network
- Using the cross-entropy loss we can calculate the gradients of the output layer
- What about the hidden layers?

Backpropagation algorithm

- We can use the simple gradient descent algorithm to optimize the weights of the network
- Using the cross-entropy loss we can calculate the gradients of the output layer
- What about the hidden layers?
- By derivating the NN function wrt. some parameters we can calculate the gradients of hidden neurons too (using the chain rule)

Backpropagation algorithm

1. Calculate the output (feedforward phase)
2. Error of the output layer: $o_i - y_i$
3. Backpropagation step: propagate the output error back layer by layer in the NN, each neurons distributes their error to their input neurons proportionately to the weights between them.
4. Calculate the gradients, assuming sigmoid hidden activation:
5. Update the parameters

Stochastic backpropagation algorithm

- Sometimes we have to much data (memory is limited)
- Using only batches of data to update the network is a solution to this problem
- This means that we minimize different loss functions as the input data changes
- In practice this approach works quite well (even better than training on all data at once)
- Stochastic gradient descent (SGD) is the dominant method used to train deep learning models
- If the batchsize==1, then its called online learning, for small batchsizes ($\leq$1000) its called minibatch training

Input data

To help the optimizer it is usually necesery to rescale the input, especcialy if the features have different ranges

Normalization

Simply force all features into a predefined range ([0,1] or [-1,1])

Input data

To help the optimizer it is usually necesery to rescale the input, especcialy if the features have different ranges

Normalization

Simply force all features into a predefined range ([0,1] or [-1,1])
To achieve this we need to calculate the minimal and maximal value of each feature

Input data

To help the optimizer it is usually necesery to rescale the input, especcialy if the features have different ranges

Normalization

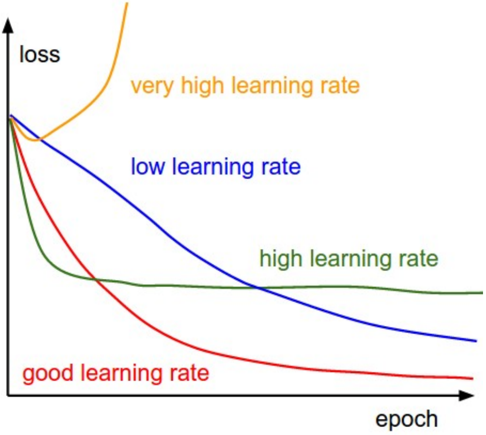
Simply force all features into a predefined range ([0,1] or [-1,1])
To achieve this we need to calculate the minimal and maximal value of each feature

Standardization

Outliers could hurt normalization, in this case we shift and rescale the features to have the same mean and deviation.

The learning rate

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient.



Tips and tricks

- Use a validation/development set to finetune the hyperparameters (learning rate, number of epochs)
- Avoid overfitting and peeking
- More data is allways usefull
- Shuffling the data is helpfull for SGD
- Avoid large networks

Practice

Python tutorial: practice_04.ipynb