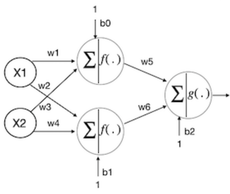
Artificial Neural Network - Basics

Summary

The exercises here are mosly conceptual and aim to deepen your understanding on how artificial neural networks work.

Exercise 1

We have this neural network. Write the expression that computes the output of this ANN.



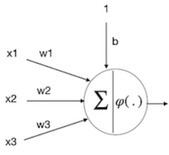
Exercise 2

Consider a neural network with two hidden layers: p = 4 input units, 2 units in the first hidden layer, 3 units in the second hidden layer, and a single output.

- Draw a picture of the network.
- How many parameters are there in your network?
- Plug in some values for the parameters.
- Give some input to the network. Assuming ReLU activation function, write out the output value.

Exercise 3

We must use a non-linear activation function for the neurons of an ANN, otherwise the output of ANN would be just a linear function of the inputs.



Sigmoid function and ReLU are some of the common activtion functions. Sketch these functions.

- What is the effect of bias in the outputs of your artificial neurons?

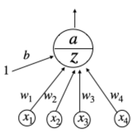
Exercise 4

The TensorFlow Playground (https://playground.tensorflow.org) is a handy neural network simulator built by the TensorFlow team. In this exercise, you will train several binary classifiers in just a few clicks, and tweak the model's architecture and its hyperparameters to gain some intuition on how neural networks work and what their hyperparameters do. Take some time to explore the following:

- The patterns learned by a neural net: Try training the default neural network by clicking the Run button (top left). Notice how it quickly finds a good solution for the classification task. The neurons in the first hidden layer have learned simple patterns, while the neurons in the second hidden layer have learned to combine the simple patterns of the first hidden layer into more complex patterns. In general, the more layers there are, the more complex the patterns can quickly be.
- Activation functions: Try replacing the tanh activation function with a ReLU activation function, and train the network again. Notice that it finds a solution even faster, but this time the boundaries are composed of straight line segments. This is due to the shape of the ReLU function.
- The risk of local minima: Modify the network architecture to have just one hidden layer with three neurons. Train it multiple times (to reset the network weights, click the Reset button next to the Play button). Notice that the training time varies a lot, and sometimes it even gets stuck in a local minimum.
- What happens when neural nets are too small: Remove one neuron to keep just two. Notice that the neural network is now incapable of finding a good solution, even if you try multiple times. The model has too few parameters and systematically underfits the training set (we have a bias problem in the context of bias-variance tradeoff).
- What happens when neural nets are large enough: Set the number of neurons to eight, and train the network several times. Notice that it is now consistently fast and never gets stuck. This highlights an important finding in neural network theory: if neural networks are large enough, they almost never get stuck in local minima, and even when they do, these local optima are almost as good as the global optimum. However, they can still get stuck on long plateaus for a long time.

Exercise 5

Consider the following neuron, where z is the sum of all the inputs to the neuron and a is the output of the activation function.



The activation function is identity function (so this neuron would be just a linear function and not very useful). The current values of the weights are $w_1 = 1$, $w_2 = 2$, $w_3 = 0.5$, $w_4 = 1.5$, $b = -0.2$

Assume we have a regression problem and use a loss function $L(y, a) = \frac{1}{2}(y - a)^2$.

Take a single training datapoint $\mathbf{x} = [0, 0.1, 2, 1]$, with expected output $y = 3$.

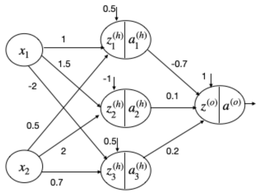
We want to take one gradient descent step and update the weights. Use a learning rate of 0.1.

- First do the forward pass to calculate the current predicted output for the datapoint.
- Use that to calculate the loss.
- Do the backward pass: take the partial derivatives of the loss function with respect to each of the weights (take the derivatives in backward steps using the chain rule).
- Compute the gradinet descent step for each weight to update their value.

What is the new predicted value for your datapoint?

Exercise 6

Consider the following ANN.



Assume that the activation functions for the hidden layer are ReLU, and for the output layer is sigmoid.

- Compute by hand the output of this network for an input vector [1,2].
- Write the steps of the forwrad pass in a ***vectorized form***.
- Write a code that executes the steps (in vectorized form) and computes the output to the input vector [1,2].
- Now use your code and viualize the decision boundary created by the network in a range of [-5,5] for both x1 and x2.
- Change the value of a weight (or a bias) and see if/how the decision boundary changes.