1. Why is it generally preferable to use a Logistic Regression classifier rather than a classical Perceptron (i.e., a single layer of linear threshold units trained using the Perceptron training algorithm)? How can you tweak a Perceptron to make it equivalent to a Logistic Regression classifier?

A classical Perceptron will converge only if the dataset is linearly separable, and it won't be able to estimate class probabilities. In contrast, a Logistic Regression classifier will converge to a good solution even if the dataset is not linearly separable, and it will output class probabilities.

A classical perceptron will only converge if the data is linearly seperable. It also cannot compute class probabilities. The logistic regression classifier is able to converge on non-linear data and outputs class probabilities.

If you want to tweak the Perceptron to make it like a logistic regression classifier you can do so by changing the activation function to the softmax or the sigmoid function and train it using gradient descent, or some other optimization algorithm.

2.Why was the logistic activation function a key ingredient in training the first MLPs?

The logistic activation function is smooth at all points. Therefore, its derivative is always non-zero. This allows gradient descent to find smaller values of theta after each iteration. When the activation function is a step function, gradient descent cannot move at all because there is no slope.

3.Name three popular activation functions. Can you draw them?

Popular types of activation functions and when to use them

Binary Step Function. ...

Linear Function. ...

Sigmoid. ...

Tanh. ...

ReLU. ...

Leaky ReLU. ...

Parameterised ReLU. ...

Exponential Linear Unit.

4. Suppose you have an MLP composed of one input layer with 10 passthrough neurons, followed by one hidden layer with 50 artificial neurons, and finally one output layer with 3 artificial neurons. All artificial neurons use the ReLU activation function.

* + What is the shape of the input matrix **X**?
  + What about the shape of the hidden layer’s weight vector **W***h*, and the shape of its bias vector **b***h*?
  + What is the shape of the output layer’s weight vector **W***o*, and its bias vector **b***o*?
  + What is the shape of the network’s output matrix **Y**?
  + Write the equation that computes the network’s output matrix **Y** as a function of **X**, **W***h*, **b***h*, **W***o* and **b***o*.

What is the shape of the input matrix X?

The shape of X will be (m x 10) where m is the batch size. The logic of a passthrough neuron is that it should have no effect at all on the input layer. Therefore, there must be one neuron for each input feature that allows the input feature to pass through to the next layer without altering its value in any way.

What about the shape of the hidden layer’s weight vector Wh, and the shape of its bias vector bh?

Since the hidden layer has 50 neurons and there are 10 features, its weight vector will have the shape of 10 x 50. Because each of the 10 features will need to be multiplied by a weight which is connected to each of the 50 hidden layer neurons.

The bias vector will have a length of 50. This is because one bias unit is added to the output of the passthrough layer. That unit is then multiplied once for each of the 50 artificial neurons.

What is the shape of the output layer’s weight vector Wo, and its bias vector bo?

The output layer has 3 neurons. The input to the output layer is the output of the hidden layer.The output of the hidden layer has 50 neurons, therefore the shape of the output layer's weight vector is 50 x 3.

The bias vector will have a length of 3 because one bias unit is added to the output of the hidden layer. That unit is then multiplied once for each of the 3 neurons in the output layer.

What is the shape of the network’s output matrix Y?

The shape of the output matrix is going to be m x 3, where m is the batch size, and 3 because each example will compute a probability that it belongs to one of three classes.

Write the equation that computes the network’s output matrix Y as a function of X, Wh, bh, Wo and bo.

Y = (X \* Wh + bh) \* (Wo + bo)

When adding a bias vector to a matrix it is added to every single row. This is called broadcasting.

5.How many neurons do you need in the output layer if you want to classify email into spam or ham? What activation function should you use in the output layer? If instead you want to tackle MNIST, how many neurons do you need in the output layer, using what activation function?

Email classification is a binary classification problem, so you would only need one neuron in the output layer. This neuron would indicate the probability that the email is spam or ham. You'd most likely use the sigmoid activation function in the output layer.

For the MNIST problem you would need 10 output neurons in the final layer, one for each digit. You would then replace the logistic function with the softmax function which can output one probability per class per digit.

Predicting housing prices is a linear regression problem. You'd only need one output neuron in the final layer. You wouldn't need to use an activation function at all.

6.What is backpropagation and how does it work? What is the difference between backpropagation and reverse-mode autodiff?

Backpropagation is an algorithm used to train neural networks. It first computes the gradients of the cost function with regards to every model parameter then it performs a gradient descent step using these gradients.

This backpropagation step is performed until the model parameters converge to values that hopefully minimize the cost function.

Backpropagation refers to the whole process of training a neural network. Reverse-mode autodiff is a technique to compute the gradients efficiently. It is used by the backprop algorithm.

7.Can you list all the hyperparameters you can tweak in an MLP? If the MLP overfits the training data, how could you tweak these hyperparameters to try to solve the problem?

In general, the hyperparameters of a neural network you can adjust are the number of hidden layers, the number of neurons in each hidden layer, and the activation function used by each neuron.

For binary classification, use the logistic activation function. For a multi-class problem, use softmax. For a linear regression problem, don't use an activation function.

Some simple ways to try and solve overfitting are reducing the number of hidden layers or the number of neurons.

8.Train a deep MLP on the MNIST dataset and see if you can get over 98% precision. Try adding all the bells and whistles (i.e., save checkpoints, restore the last checkpoint in case of an interruption, add summaries, plot learning curves using TensorBoard, and so on).

What is Deep MLP?

Deep MLP is creating multiple hidden layers in a neural network. Here the depth refers to the number of hidden layers.

10 Steps to train Deep MLP:

Preprocess Data: This is a very important step and more attention should be paid here. Preprocessing helps us to understand the data better and any irrelevant data can be removed or most important fields can be identified.

Weights: Weights need to be initialized before proceeding to the next step. Xavier/ Glorot can be used for Sigmoid/ TanH activations or He for ReLU activation

Activation function: ReLU is the most commonly used activation function now

Batch Normalization: is a process of adding a layer deep in the network to normalize the data. It enables achieve faster convergence.

Optimizer: Most commonly used Optimizer is Adam [Adaptive Model Estimation]

Hyperparameters: Architecture — number of layers, number of neurons; Dropout rate; Optimizer parameters like mean/ variance at time ‘t’

Loss Function: Determine the log function. Commonly used log function is log-loss, squared loss etc

Gradient: Its a good practice to monitor gradients and updates for each epoch, layer and weight. Clipping is a common method for gradient monitoring

Plot: Visualize train loss against epochs. Train loss should be approximately close to test loss for a perfect model

Over-fitting: Take extra care to avoid overfitting. Difference between train and test loss should be small

Load MNIST Data

If you are copying and pasting in the code from this tutorial, start here with these two lines of code which will download and read in the data automatically:

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

Here mnist is a lightweight class which stores the training, validation, and testing sets as NumPy arrays. It also provides a function for iterating through data minibatches, which we will use below.

Start TensorFlow InteractiveSession

TensorFlow relies on a highly efficient C++ backend to do its computation. The connection to this backend is called a session. The common usage for TensorFlow programs is to first create a graph and then launch it in a session.

Here we instead use the convenient InteractiveSession class, which makes TensorFlow more flexible about how you structure your code. It allows you to interleave operations which build a [computation graph](https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.10/tensorflow/g3doc/get_started/basic_usage.md#the-computation-graph) with ones that run the graph. This is particularly convenient when working in interactive contexts like IPython. If you are not using an InteractiveSession, then you should build the entire computation graph before starting a session and [launching the graph](https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.10/tensorflow/g3doc/get_started/basic_usage.md#launching-the-graph-in-a-session).

import tensorflow as tf

sess = tf.InteractiveSession()

Computation Graph

To do efficient numerical computing in Python, we typically use libraries like [NumPy](http://www.numpy.org/) that do expensive operations such as matrix multiplication outside Python, using highly efficient code implemented in another language. Unfortunately, there can still be a lot of overhead from switching back to Python every operation. This overhead is especially bad if you want to run computations on GPUs or in a distributed manner, where there can be a high cost to transferring data.

TensorFlow also does its heavy lifting outside Python, but it takes things a step further to avoid this overhead. Instead of running a single expensive operation independently from Python, TensorFlow lets us describe a graph of interacting operations that run entirely outside Python. This approach is similar to that used in Theano or Torch.

The role of the Python code is therefore to build this external computation graph, and to dictate which parts of the computation graph should be run. See the [Computation Graph](https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.10/tensorflow/g3doc/get_started/basic_usage.md#the-computation-graph) section of [Basic Usage](https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.10/tensorflow/g3doc/get_started/basic_usage.md) for more detail.