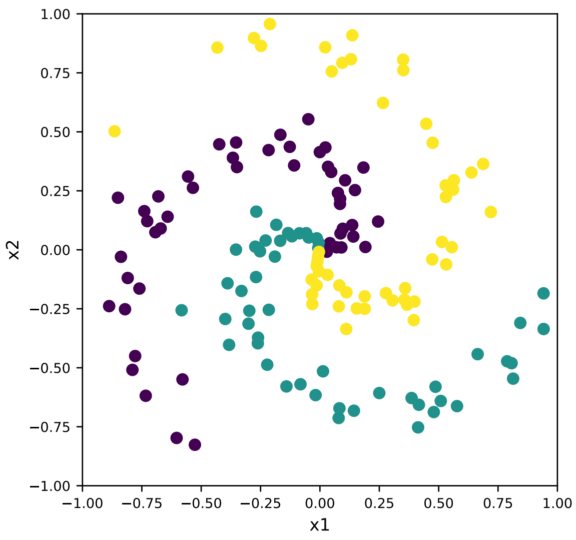
Tutorial 2 Solutions: Feedforward neural networks in TensorFlow

May 28, 2019

The objective of this tutorial is to become comfortable with using the software library TensorFlow to create and train a simple feedforward neural network for supervised learning.

You will use and modify the Python program `tutorial2_spirals.py`. This code starts by generating a random dataset of 2D points with `K` branches. For example, when `K=3` this dataset might look as follows:



For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the branch index such that $y = 0, 1$ or 2 for the example above. Our goal is to implement a neural network capable of classifying the branches.

This network will compare its output with labels in the so-called *one-hot encoding*. For a given label `y=k`, the corresponding one-hot encoding is a `K`-dimensional vector with all entries zero except for the `k`th entry (which has value 1). So when `K=3` the one-hot encodings for the labels are

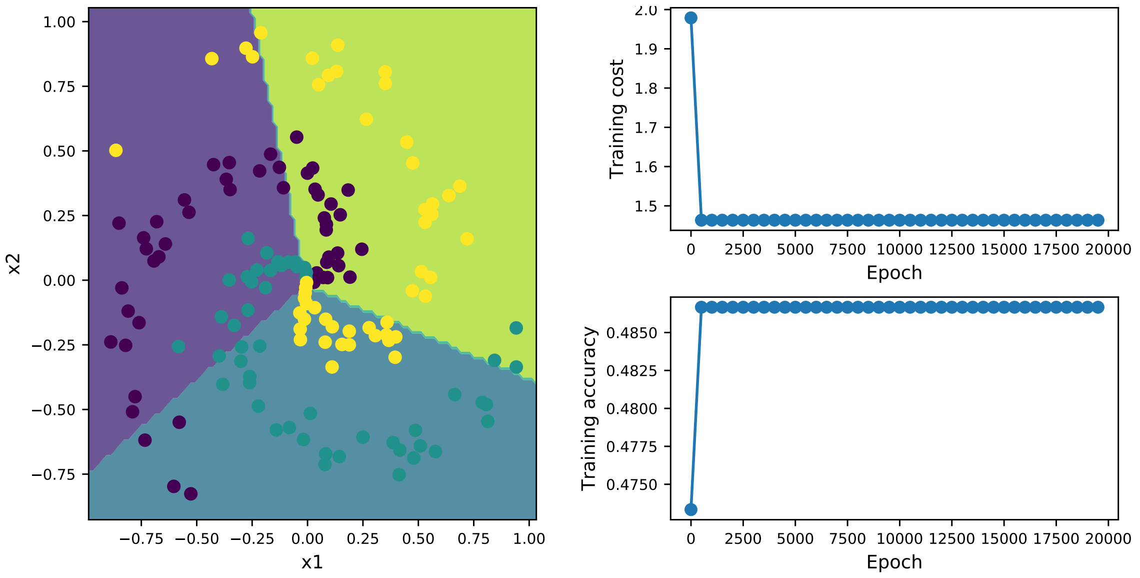
$$0 \rightarrow \begin{bmatrix}1\\0\\0\end{bmatrix}, \qquad 1 \rightarrow \begin{bmatrix}0\\1\\0\end{bmatrix}, \qquad 2 \rightarrow \begin{bmatrix}0\\0\\1\end{bmatrix}.$$

The code first defines the structure of the neural network and then uses the dataset to train this network. The code generates two files: `spiral_data.pdf` (a plot of the dataset such as the one above) and `spiral_results.pdf` (which displays three plots illustrating the results of training).

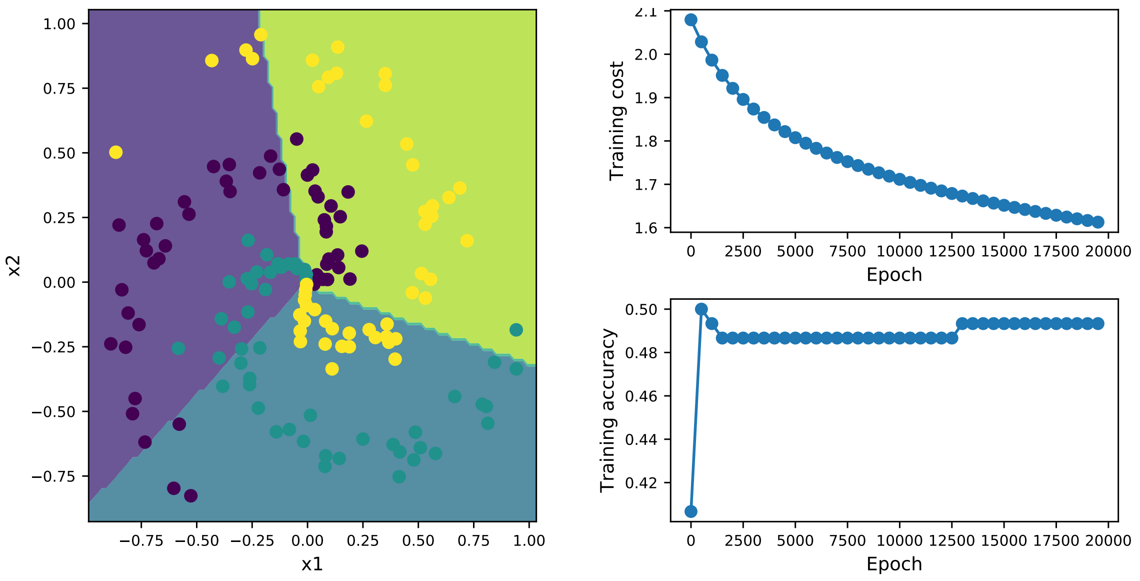
1 Neural network architecture, activation functions, cost functions and learning rate

a) Run the code and look at how it attempts to classify the two-dimensional space. You should find that the resulting classifier separates the two-dimensional space using lines, and thus does a poor job of representing the data.

Solution: Using the parameters provided (with the learning rate set to 1), you should find that the given network classifies the two-dimensional space as:

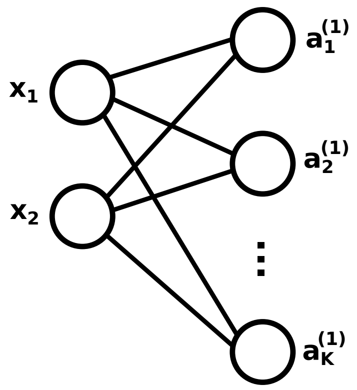


Note: If the learning rate is 0.001, you will be able to observe the training cost decreasing:



b) Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network corresponding to the one in the code for the case of `K` branches. Pay particular attention to the number of neurons in each layer.

Solution: The neural network for the given code is:



where the last layer has K neurons.

c) Add in a hidden layer with 4 neurons and study how this hidden layer changes the output. On paper, draw the neural network in this case.

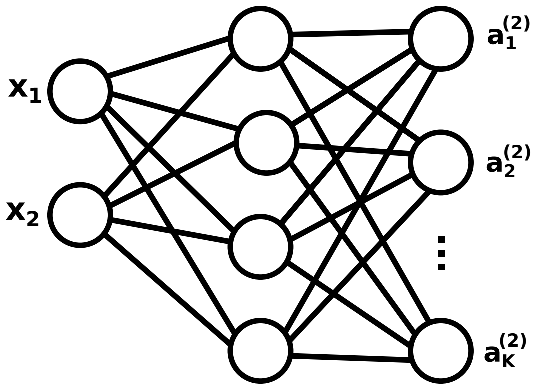
Solution: Replace the `Layer 1` and `Network output` sections with the following code:

```
nH=4
### Layer 1:  ###
W1 = tf.Variable(tf.random_normal([2, nH], mean=0.0, stddev=0.01, dtype=tf.float32))
b1 = tf.Variable( tf.zeros([nH]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )


### Layer 2:  ###
W2 = tf.Variable(tf.random_normal([nH, K], mean=0.0, stddev=0.01, dtype=tf.float32))
b2 = tf.Variable( tf.zeros([K]) )
z2 = tf.matmul(a1, W2) + b2
a2 = tf.nn.sigmoid( z2 )


### Network output:  ###
aL = a2
```

After adding a hidden layer with 4 neurons, the neural network is:



where the last layer has K neurons.

d) Replace the sigmoid activation function on the first layer with a rectified linear unit (ReLU), and study how the choice of activation function changes the output.

Solution: Replace the line `a1 = tf.nn.sigmoid( z1 )` with:

```
a1 = tf.nn.relu( z1 )
```

When using the sigmoid activation function, you might observe that the classifier separates the two-dimensional space into regions that meet roughly at the origin. When using the ReLU activation function on the first layer, the meeting place for these regions tends to jump around more.

e) Change the cost function so that it is computed using the mean-squared error (MSE) instead of the cross-entropy, and study how the choice of cost function changes the output.

Solution: Add the following beneath the line `cost_func = cross_entropy`:

```
mse = tf.reduce_mean( tf.reduce_sum( tf.square(y_onehot - aL) ) )
cost_func = mse
```

f) Study the effects of increasing and decreasing the `learning_rate` hyperparameter. Examine these effects using both the cross-entropy and mean-squared error cost functions.

Solution: When using sigmoid activation functions and the mean-squared error as the cost function, for the given data set you will likely find that the network does not make progress when `learning_rate=1.0`. Decreasing the learning rate to 0.1 enables the network to make progress. For discussion about the affects of changing the learning rate, see Section IVA of `arXiv:1803.08823`.

g) Explain why the K-dimensional one-hot encoding is useful. What do you think would happen if you used a one-dimensional label (such that $\mathtt{y} = 0, 1, \ldots, \mathtt{K} = 1$ or $\mathtt{K}$) instead?

Solution: As an illustrative example, consider the case where there are K=3 branches and the network is uncertain about whether the datapoint $\mathbf{x}$ belongs to branch 0 or branch 2. Specifically, let's say that the network is 50% sure that $\mathbf{x}$ belongs to branch 0 and 50% sure that it belongs to branch 2. This situation is easy to express in terms of the one-hot encoding as

$$\mathbf{a}^{(L)} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix}.$$

Using a one-dimensional encoding, the network would only return a single number between 0 and 2. In this case, it might return the average predicted label such that

$$\mathbf{a}^{(L)} = \left[\tfrac{0+2}{2}\right] = \left[1\right],$$

which is the same as the output it would return if it was certain that $\mathbf{x}$ belongs to branch 1. Thus, we see that a one-dimensional encoding can give network outputs that are hard to interpret when the number of classes K is greater than 2.

2 Group work

For this part of the tutorial, you will work in groups to explore the capabilities of a feedforward neural network with one hidden layer. Each group will plot the accuracy of the neural network as a function of some quantity Q (a property of the network or the data).

For all plots, the accuracy should be measured once you are convinced that the network has converged, which means that you may need to adjust the `N_epochs` parameter in order to run the code longer. In some cases, you may also find it useful to adjust the learning rate as Q is varied (in particular if you find that your network is getting stuck or if training is very slow for certain values of Q).

Each group's task is to plot the accuracy of the neural network as a function of some quantity Q, where

- **Groups 1 & 2:** $Q =$ the number of neurons in the hidden layer
- **Groups 3 & 4:** $Q =$ `mag_noise` (the magnitude of noise in the data)
- **Groups 5 & 6:** $Q =$ `K` (the number of different labels)

Solution: **Check Slack channel tutorial.**