Project 3: Neural Network

Due on 11/08/2019 @ 3:30pm

CS 491 – Machine Learning

Eric Duong
eduong@nevada.unr.edu

Kripash Shrestha
kripashs@gmail.com

Contents

1 Neural Network

1.1 `build_model(X, y, nn_hdim, num_passes=20000, print_loss=False)`

1.1.1 Setting up

This function is going to return a model that is implemented as dictionary of the following structure:

`model = {'W1':weight1, 'b1':bias1, 'W2':weight2, 'b2':bias2}`

All of the parameters are initialized between 0 and 1. We treat number of passes as number of epochs. Each batch contains the entire sample/training dataset. We iterate through the entire dataset and update 20,000 times.

1.1.2 Forward pass

Before be can update our weights and biases, we must calculate the forward passing variables and predictions. Let M be the number of hidden nodes in the hidden layer. Our first weight $\mathbf{W}_1$ allows us to map 2 features into M hidden nodes. We also have the first bias $\vec{b_1}$ which has dimension equals to M. Similarly, our second weight $\mathbf{W}_2$ allows us to map number of hidden nodes M back to 2 dimensions as output, and the second bias $\vec{b_2}$ which has dimension equals to 2. Let $\vec{x}$ be a sample from the training dataset $\mathbf{X}$, where $\mathbf{X}$ has N number of samples. We can forward pass and predict a label from the input features a single sample:

$$\underset{1\times M}{\vec{a}} = \underset{1\times 2}{\vec{x}}\,\underset{2\times M}{\mathbf{W}_1} + \underset{1\times M}{\vec{b}_1}$$

$$\underset{1\times M}{\vec{h}} = \tanh(\vec{a})$$

$$\underset{1\times 2}{\vec{z}} = \underset{1\times M}{\vec{h}}\,\underset{M\times 2}{\mathbf{W}_2} + \underset{1\times 2}{\vec{b}_2}$$

$$\underset{1\times 2}{\vec{\hat{y}}} = \text{softmax}(\vec{z})$$

However, instead of predicting the label for each sample $\vec{x}$, we predict the labels for the entire sample set $\mathbf{X}$ simultaneously, with number of batches equal to N number of samples (the entire sample set). Our forward pass transforms into:

$$\underset{N\times M}{a} = \underset{N\times 2}{\mathbf{X}}\,\underset{2\times M}{\mathbf{W}_1} + \underset{N\times M}{b_1}$$

$$\underset{N\times M}{h} = \tanh(a)$$

$$\underset{N\times 2}{z} = \underset{N\times M}{h}\,\underset{M\times 2}{\mathbf{W}_2} + \underset{N\times 2}{b_2}$$

$$\underset{N\times 2}{\hat{y}} = \text{softmax}(z)$$

Note: `numpy` applies broadcasting to $\vec{b}_1$ and $\vec{b}_2$ and turn them into b_1 and b_2 with $N \times M$ dimensions.

1.1.3 Compute gradient and back-propagate

Before we can back-propagate, we must transform labels y into classification vector. With label `0` we have `[1, 0]`, and with label `1` we have `[0, 1]`. We transform y into $N \times 2$ matrix. After computing the values for the gradients, we have the following gradient with certain dimensions:

$$\underset{N\times 2}{\frac{\partial L}{\partial \hat{y}}} \quad \underset{N\times M}{\frac{\partial L}{\partial a}} \quad \underset{M\times 2}{\frac{\partial L}{\partial \mathbf{W}_2}} \quad \underset{N\times 2}{\frac{\partial L}{\partial b_2}} \quad \underset{2\times M}{\frac{\partial L}{\partial \mathbf{W}_1}} \quad \underset{N\times M}{\frac{\partial L}{\partial b_1}}$$

With matching dimensions, we can update the weights and biases with some learning rate η as such:

$$\mathbf{W}_1 \leftarrow \mathbf{W}_1 - \eta \frac{\partial L}{\partial \mathbf{W}_1}$$
$$b_1 \leftarrow b_1 - \eta \frac{\partial L}{\partial b_1}$$
$$\mathbf{W}_2 \leftarrow \mathbf{W}_2 - \eta \frac{\partial L}{\partial \mathbf{W}_2}$$
$$b_2 \leftarrow b_2 - \eta \frac{\partial L}{\partial b_2}$$

Algorithm 1: `build_model`

Data: $\mathbf{X}$ contains features of the entire sample set. y contains a list of labels of the entire sample set.

Result: `model =` {`'W1':`$\mathbf{W}_1$`, 'b1':`b_1`, 'W2':`$\mathbf{W}_2$`, 'b2':`b_2}

begin

- convert y into 2D classification vectors.
- initialize $\mathbf{W}_1$, b_1, $\mathbf{W}_2$ and b_2 between 0 and 1.
- **for** number of passes **do**
  - $\hat{y} \leftarrow$ forward pass with the entire sample set $\mathbf{X}$.
  - compute gradients of the weights and biases.
  - update weights and biases with gradients multiplied with η
- **return** {`'W1':`$\mathbf{W}_1$`, 'b1':`b_1`, 'W2':`$\mathbf{W}_2$`, 'b2':`b_2}

1.2 `calculate_loss(model, X, Y)`

1.2.1 Setting up

This function is going to return a loss value that represents the loss over the total number of samples:
We will call each label in Y, y_truth for our iteration.

1.2.2 Implementation

The function takes the model as shown below:
`model = {'W1':weight1, 'b1':bias1, 'W2':weight2, 'b2':bias2}`

The function will then calculate the labels to classifications vectors of either [0,1] or [1,0] based on if the label represents either a 1 or 0 (respectively).

For each sample:

1. The function will calculate the hidden layer activation using the function:

$$a = w1 * sample + b1 \tag{1}$$

2. The function will take the calculated activation and take the tanh of the function. We will call this value h.

3. The function will calculate the output of the model by using the following function:

$$a = w2 * h + b2 \tag{2}$$

4. We then calculate the prediction vector for the sample and apply the softmax to get the output classification for the sample. We will call this y_predict.

5. We then calculate the loss for the current sample by using the function:

$$loss = Y\_truth * log(y\_predict) \tag{3}$$

   After calculating the loss for the sample, we add it to an overall loss for the model over the given samples

After calculating the loss for all of the samples and getting the total loss. The function will return the loss over the total number of samples.

1.3 `predict(model, X)`

1.3.1 Setting up

This function is going to return the index of the softmax array (prediction array) that has the highest classification value:

1.3.2 Implementation

`model = {'W1':weight1, 'b1':bias1, 'W2':weight2, 'b2':bias2}`

We first calculate the hidden layer activation by taking our input x and dotting it with the weight W1 from the model and then adding the bias1 from the model. The equation that represents this is:

$$a = w1 * x + b1 \tag{4}$$

We then take the calculated activation and take the *tanh* of the activation; we will call this value h. We take the calculated value from the tanh function and use that as our input into the next hidden layer. We calculate the activation with:

$$z = w2 * h + b2 \tag{5}$$

We will take the softmax of the values in the array and take the index with the high classification value. This will represent our prediction for the sample input.

2 Output

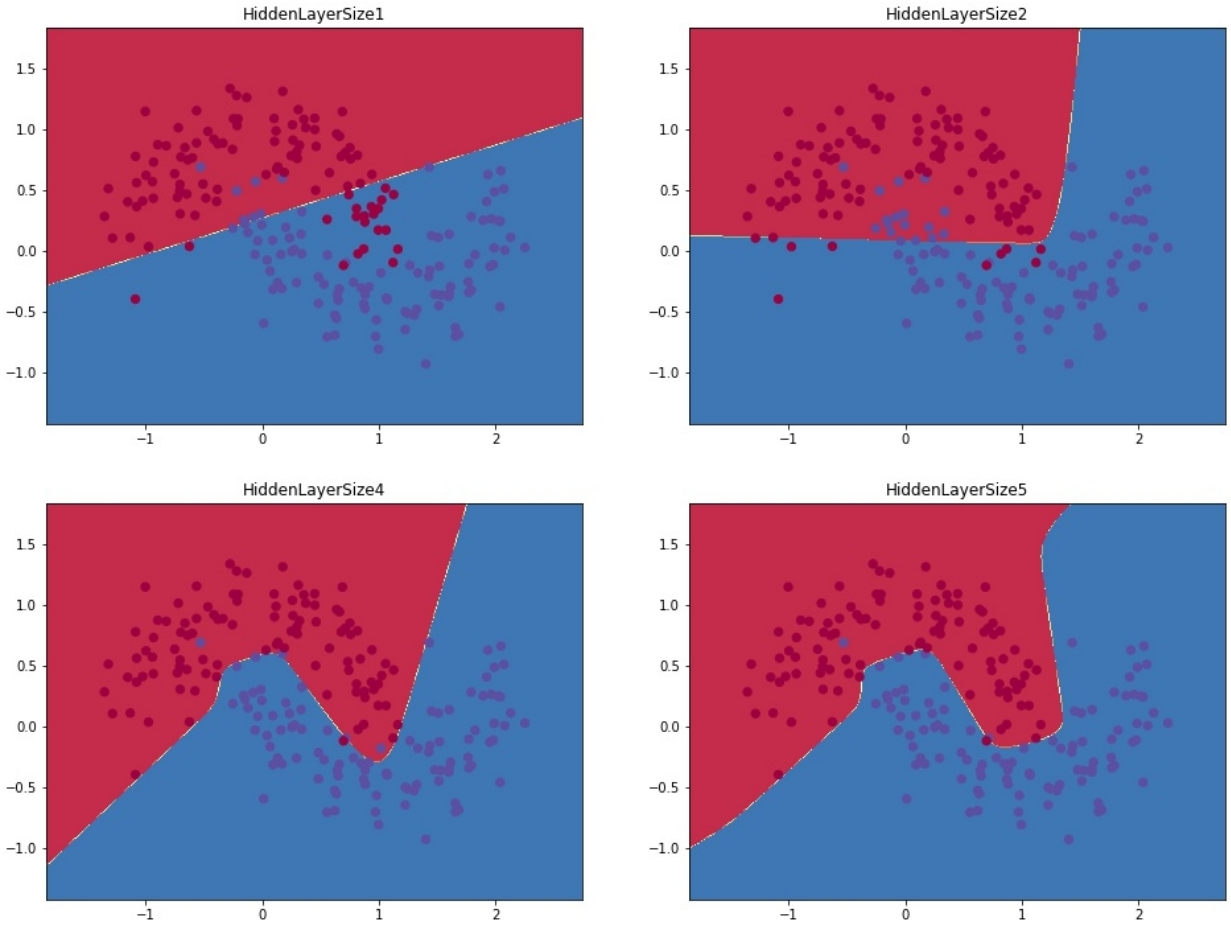

Figure 1: Decision boundary for 1, 2, 4, and 5 hidden nodes. Last training loss: 0.03