UNIVERSITY OF NEVADA, RENO

Project 3 - Neural Networks

Sayra Ramirez, Kevin Carlos

November 8, 2019

CONTENTS

1 IMPLEMENTATION

1.1 PREDICT

```
# Helper function to predict an output (0 or 1)
# model is the current version of the model
# x is one sample (without the label)
def predict(model, x):
  W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']

  a = x.dot(W1) + b1
  h = np.tanh(a)
  z = h.dot(W2) + b2
  y_hat = softmax(z)

  prediction = np.argmax(y_hat, axis=1)

  return prediction
```

Predict is a function that takes in the current model at the time of predicting and a single sample, e.g. [1.6434, 3.23256], and performs forward propagation on the model. It then uses softmax to create a probability distribution on the sample. The purpose of using numpy's argmax is to give a general solution to the index of the highest probability. Index 0 or index 1 is the prediction.

1.2 Calculate Loss

```
def calculate_loss(model, X, y):
  loss = 0
  W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
  new_y = encode(y)

  a = X.dot(W1) + b1
  h = np.tanh(a)
  z = h.dot(W2) + b2
  y_hat = softmax(z)

  # Calculate the loss
  for i in range(X.shape[0]):
    for j in range(new_y.shape[1]):
      loss += -(new_y[i][j] * np.log(y_hat[i][j]))

  loss /= X.shape[0]
  return loss
```

Calculate loss takes in three variables: the model, X (the dataset), and y (the labels). We again perform forward propagation. In calculate loss, the y labels are encoded. Meaning a label of 0 becomes [1 0] and a label of 1 becomes [0 1]. We calculate the loss by doing the summation of the incorrectly classified samples multiplied together, as seen in the equation below.

$$L(y-\hat{y}) = -\frac{1}{N}\sum_{n}\sum_{i} y_{n,i} log \hat{y}_{n,i}$$

The last step code-wise is doing the $\frac{1}{N}$ before returning our loss. This function is just a helper function for us to visualize the loss declining over every 1000 iterations.

1.3 BUILD MODEL

```
eta = 0.032
def build_model(X, y, nn_hdim, num_passes=20000, print_loss=False):
  # Randomly initialize input layer weights
  W1 = np.random.rand(2, nn_hdim)
  b1 = np.random.rand(1, nn_hdim)
  W2 = np.random.rand(nn_hdim, 2)
  b2 = np.random.rand(1, 2)

  # Encode the y's
  new_y = encode(y)

  for i in range(num_passes):
    # Forward propagation
    a = X.dot(W1) + b1
    h = np.tanh(a)
    z = h.dot(W2) + b2
    y_hat = softmax(z)

    # Try encoding the y_hats
    new_y_hat = np.zeros((200,2), dtype=int)
    for k in range(y_hat.shape[0]):
      if (np.argmax(y_hat[k]) == 0):
        new_y_hat[k][0] = 1
      else:
        new_y_hat[k][1] = 1
    dLdy_hat = new_y_hat - new_y #0.32
    # dLdy_hat = y_hat - new_y #0.012
    dLda = (1 - pow(h, 2)) * dLdy_hat.dot(W2.T)
    dLdW2 = (h.T).dot(dLdy_hat)
    dLdb2 = np.sum(dLdy_hat)
    dLdW1 = (X.T).dot(dLda)
    dLdb1 = np.sum(dLda)

    # Update weights/biases
    W1 = W1 - (eta * dLdW1)
    W2 = W2 - (eta * dLdW2)
    b1 = b1 - (eta * dLdb1)
    b2 = b2 - (eta * dLdb2)

    model = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}
  return model
```

We define our own eta since it is a hyper parameter that is not passed into the function. The very first step is to randomly initialize the weights and biases according to the dimensionality that is passed into $build\_model$. Then, in our solution, the y labels are encoded using one hot encoding as aforementioned in $calculate\_loss$.

Then, we build the model over $num\_passes = 20000$ iterations.

1. Perform forward propagation
2. Backpropagation
   - $\frac{dL}{d\hat{y}} = \hat{y} - y$
   - $\frac{dL}{da} = (1 - tanh^2 a) * \frac{dL}{d\hat{y}} W_2^T$
   - $\frac{dL}{dW_2} = h^T \frac{dL}{d\hat{y}}$
   - $\frac{dL}{db_2} = np.sum(\frac{dL}{d\hat{y}})$
   - $\frac{dL}{dW_1} = x^T \frac{dL}{da}$
   - $\frac{dL}{db_1} = np.sum(\frac{dL}{da})$
3. Update the weights and biases

$$W = W - \eta \frac{dL}{dW}$$

$$b = b - \eta \frac{dL}{db}$$

This is performed 20000 times for each dimensionality [1, 2, 3, 4].

2 Decision Boundary Plot

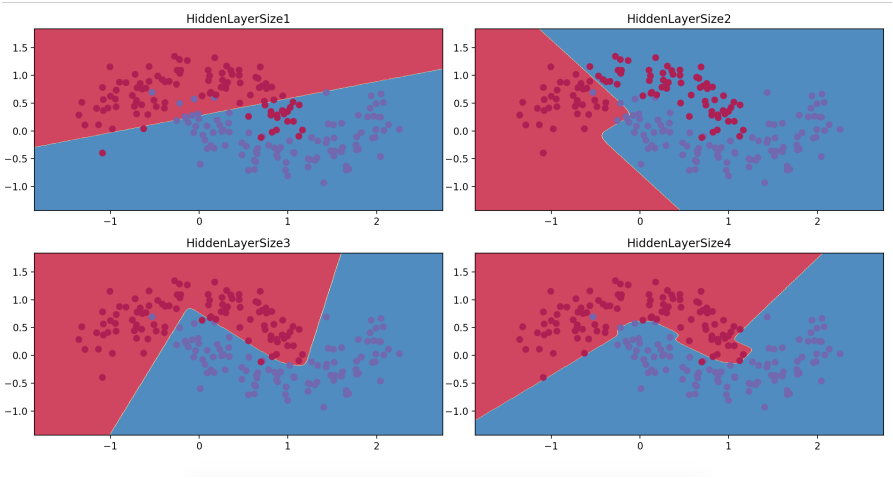

Figure 2.1: Plots for hidden layers [1, 2, 3, 4]

In the write-up, it is said, "Plot your decision boundary for 1, 2, , 4, and 5 hidden nodes." We performed the operaiton on what the given code said and what the resulting image on WebCampus.

2.1 Results

The plots for hidden layer 1, 3, and 4 are similar to the expected plots with some variation. The slope on hidden layer 1 is lower than to be expected. For layer 3, the plot is similar, but less smooth than expected. For layer 4, the slopes within the center of the plot amongst all of the dots are "sharp" and may suggest the effects our non-linearity function had throughout training the NN.

Layer 2 is the plot that we had the most variation with. While it has the similar sloping fashion that the expected result has, it is rotated into the negative direction.