Assignment 2

Team: Naveksha and Billy

Part 2 Answers

1. **Cross entropy loss function**

$$L(W) = -\sum_{n=1}^{N} y\,log(y') + (1-y)log(1-y') \text{ where } y' = \sigma(Wx)$$

During training y is either 1 or 0 so one of the two terms will become 0.

Derivative of cross entropy loss w.r.t W

$$\frac{d(L(w))}{dW} = -\frac{d(ylog(y'))}{dW} - \frac{d((1-y)log(1-y'))}{dW}$$

$$\frac{d(L(w))}{dW} = -y\frac{d(log(y'))}{dW} - (1-y)\frac{d(log(1-y'))}{dW}$$

Using chain rule,

$$\frac{d(L(w))}{dW} = \left(-\frac{y}{y'} - \frac{(1-y)}{(1-y')}\right)\frac{dy'}{dW}$$

$$\frac{d(L(w))}{dW} = \left(-\frac{y}{y'} - \frac{(1-y)}{(1-y')}\right)\frac{d(\sigma(Wx))}{dW}$$

Taking derivative of sigmoid function,

$$\frac{d(L(w))}{dW} = \left(-\frac{y}{y'} - \frac{(1-y)}{(1-y')}\right)\sigma(Wx)(1-\sigma(Wx))\frac{d(Wx)}{dW}$$

Simplify

$$\frac{d(L(w))}{dW} = -1 * \left(\frac{y(1-y')+y'(1-y)}{y'(1-y')}\right)y'(1-y')x$$

$$\frac{d(L(w))}{dW} = -1 * (y(1-y') + y'(1-y)) * x$$

$$\frac{d(L(w))}{dW} = (y(y'-1) - y'(y-1))) * x$$

$$\frac{d(L(w))}{dW} = (yy' - y - yy' + y') * x$$

$$\frac{d(L(w))}{dW} = (y'-y) * x$$

2. **Gradient of output loss w.r.t W1**

$$\frac{d(L(W))}{dW1} = \frac{d}{dW1}(\sigma(W2x') - y) * x'$$

Rule of derivative of terms in multiplication

$$\frac{d(L(W))}{dW1} = (\sigma(W2x') - y) * \frac{d}{dW1}x' + \frac{d}{dW1}(\sigma(W2x') - y) * x'$$

Where x' is output from the hidden layer, which is input to the last layer.

$$x' = \sigma(W1x)$$

Taking Gradient w.r.t W2

$$\frac{d^2(L(w))}{dW1dW2} = \frac{d}{dW2}(((\sigma(W2x') - y) * \frac{d}{dW1}x') + \frac{d}{dW2}(\frac{d}{dW1}(\sigma(W2x') - y) * x')$$

Simplify using rule of multiplication and chain rule

$$\frac{d^2(L(w))}{dW1dW2} = ((\sigma(W2x') - y)\frac{d}{dW2}(\frac{d}{dW1}x') + \frac{d}{dW1}x'\frac{d}{dW2}((\sigma(W2x') - y) + \frac{d}{dW2}(\frac{d}{dW1}(\sigma(W2x') - y) * x')$$

Solving a small part of equation

$$\frac{d}{dW2}(\frac{d}{dW1}x') = \frac{d}{dW2}(\sigma(W1x) * (1 - \sigma(W1x) * x) = 0$$

$$\frac{d^2(L(w))}{dW1dW2} = (\frac{d}{dW1}x'\frac{d}{dW2}((\sigma(W2x') - y) + \frac{d}{dW2}(\frac{d}{dW1}(\sigma(W2x') - y) * x')$$

$$\frac{d^2(L(w))}{dW1dW2} = (\frac{d}{dW1}x'\frac{d}{dW2}((\sigma(W2x') - y) + \frac{d}{dW1}(\sigma(W2x') - y) * \frac{d}{dW2}(x') + x'\frac{d}{dW2}(\frac{d}{dW1}(\sigma(W2x') - y))$$

$$\frac{d^2(L(w))}{dW1dW2} = (\frac{d}{dW1}x'\frac{d}{dW2}((\sigma(W2x') - y) + x'\frac{d^2}{dW1dW2}(\sigma(W2x') - y))$$

Solving a small part of equation

$$\frac{d}{dW1}x' = \frac{d}{dW1}(\sigma(W1x)) = \sigma(W1x)(1 - \sigma(W1x))x = x'x - x'^2x$$

$$\frac{d}{dW2}(\sigma(W2x') - y) = \sigma(W2x')(1 - \sigma(W2x'))x' = y'x' - y'^2x'$$

$$\frac{d^2}{dW1dW2}(\sigma(W2x') - y)) = \frac{d}{dW1}(\sigma(W2x')(1 - \sigma(W2x')))$$

$$\frac{d^2}{dW1dW2}(\sigma(W2x') - y)) = \sigma(W2x')\frac{d}{dW1}(1 - \sigma(W2x')) + (1 - \sigma(W2x'))\frac{d}{dW1}\sigma(W2x')$$

Further solving a smaller part of equation

$$\sigma(W2x')\frac{d}{dW1}(1 - \sigma(W2x')) = -\sigma(W2x') * \frac{d}{dW1}(\sigma(W2x'))$$

$$\frac{d^2}{dW1dW2}(\sigma(W2x') - y)) = -\sigma(W2x')\frac{d}{dW1}(\sigma(W2x')) + \frac{d}{dW1}\sigma(W2x') - \sigma(W2x')\frac{d}{dW1}\sigma(W2x')$$

$$\frac{d^2}{dW1dW2}(\sigma(W2x') - y)) = -2\sigma(W2x')\frac{d}{dW1}(\sigma(W2x')) + \frac{d}{dW1}\sigma(W2x')$$

$$\frac{d^2}{dW1dW2}(\sigma(W2x') - y)) = (1 - 2y')\frac{d}{dW1}(\sigma(W2x'))$$

Putting the small parts back

$$\frac{d^2(L(w))}{dW1dW2} = x'(y' - y)$$

3. **Generalize the equation for h hidden layers**

$$\frac{d^h(L(w))}{dW1dWh} = x'_{h-1}(x'_h - x_h)$$

4. **Optimization on vanilla gradient descent**

4.1 Add regularization: (L2 regularization: add a function of square of weights to the loss function)

4.2 Learning rate annealing: Reduce the learning rate with increasing epochs.

Results

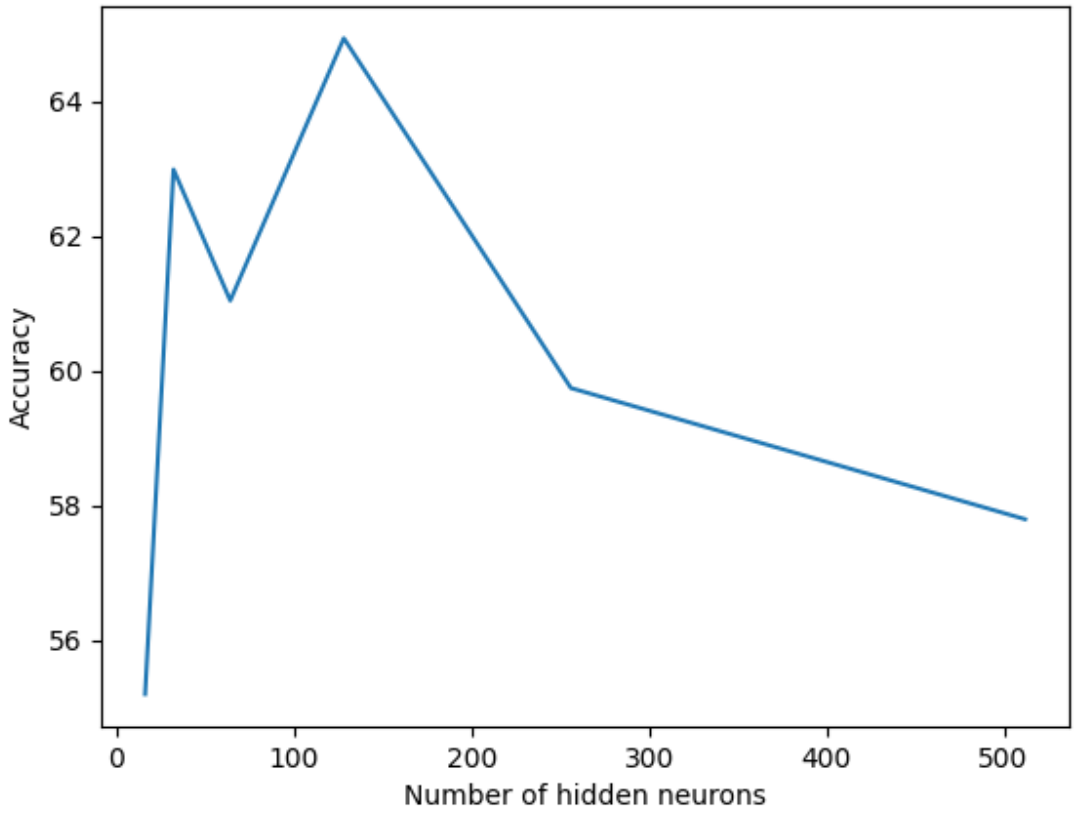
1. **Results for random classifier, Naive Bayes', Logistic regression and Vanilla NN**
   Dataset: Disease.csv

```
(venv) navekshasood@Navekshas-MacBook-Pro MLCL % python script_classify.py
Split 768 rows into train=614 and test=154 rows.
Running learner = Random
Accuracy for Random: 51.94805194805194
Running learner = Naive Bayes
Accuracy for Naive Bayes: 64.93506493506493
Running learner = Logistic Regression
Accuracy for Logistic Regression: 60.3896103896104
Running learner = Neural Network
Accuracy for Neural Network: 53.246753246753244
```

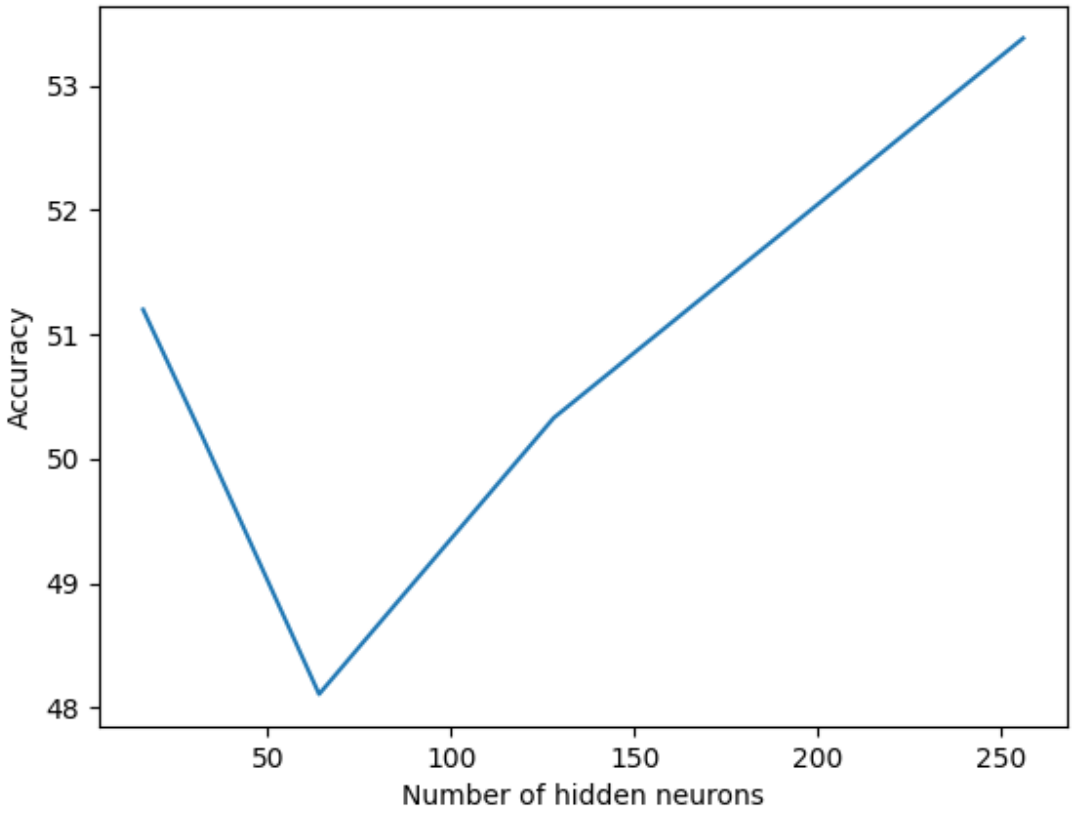
   Dataset: IMDB.csv
   (This takes really long to train, hence I am pasting the accuracy I saved earlier.)
   1. **Max_features = 5000**
      Accuracy for Naive Bayes: 85.23
      Accuracy for Logistic Regression: 62.23
      Accuracy for Neural Net: 83.19
   2. **Max_features = 100**
      Accuracy for Naive Bayes: 50.82
      Accuracy for Logistic Regression: 53.11
      Accuracy for Neural Net: 57.34

2. **Experiments with hidden neurons**
   Dataset: Disease.csv
   Neurons = [16, 32, 64, 128, 256, 512]

```
(venv) navekshasood@Navekshas-MacBook-Pro MLCL % python script_classify.py
Split 768 rows into train=614 and test=154 rows.
Running learner = Neural Network
Accuracy for Neural Network: 55.1948051948052
Running learner = Neural Network
Accuracy for Neural Network: 62.98701298701299
Running learner = Neural Network
Accuracy for Neural Network: 61.038961038961034
Running learner = Neural Network
Accuracy for Neural Network: 64.93506493506493
Running learner = Neural Network
Accuracy for Neural Network: 59.74025974025974
Running learner = Neural Network
Accuracy for Neural Network: 57.7922077922078
```



Dataset: IMDB.csv
Neurons = [16, 32, 64, 128, 256, 512]



The plots depict that the accuracy keeps increasing as we increase the number of hidden neurons or the capacity of the neural net until such time that it starts overfitting and the test accuracy starts declining. Hence, the ideal number of neurons is 128 for disease dataset and 256 or more for IMDB dataset.

3. **Experiments with optimization techniques:**
   Dataset: Disease.csv
   Regularization:

```
(venv) navekshasood@Navekshas-MacBook-Pro MLCL % python script_classify.py
Split 768 rows into train=614 and test=154 rows.
Running learner = Neural Network
Accuracy for Neural Network: 69.48051948051948
```

   Annealing learning rate:

```
(venv) navekshasood@Navekshas-MacBook-Pro MLCL % python script_classify.py
Split 768 rows into train=614 and test=154 rows.
Running learner = Neural Network
Accuracy for Neural Network: 65.5844155844156
```

Dataset: IMDB.csv
Regularization:

```
(venv) navekshasood@Navekshas-MacBook-Pro MLCL % python script_classify.py
Split 50000 rows into train=40000 and test=10000 rows
Running learner = Neural Network
Accuracy for Neural Network: 60.31
```

Annealing learning rate:

```
(venv) navekshasood@Navekshas-MacBook-Pro MLCL % python script_classify.py
Split 50000 rows into train=40000 and test=10000 rows
Running learner = Neural Network
Accuracy for Neural Network: 60.129999999999995
```

Note: We started working on the IMDB dataset before it was posted on canvas, and hence we are working on a csv file we found online. We have checked and made sure that the data is the same.

References:

1. https://www.dropbox.com/s/rxrtz3auu845fuy/Softmax.pdf?dl=0
2. https://prvnk10.medium.com/sigmoid-neuron-and-cross-entropy-962e7ad090d1\
3. https://jmlb.github.io/ml/2017/12/26/Calculate_Gradient_Softmax/
4. https://towardsdatascience.com/improving-vanilla-gradient-descent-f9d91031ab1d