# CS 577 - Homework 1

Shahab Rahimirad

Mar 10th 2024

## 1 Question 1

We use the softmax function to get probabilities for each class in the out put that sum to 1. We also use an argmax function to get the highest value among these probabilities.

$$softmax(x)_i = \frac{\exp(x_i)}{\Sigma_j \exp(x_j)}$$

The loss function we use is negative log likelihood or cross entropy. We also add an l1 regularization term to it. We denote cross entropy loss as $L_{ce}$.

$$L_{ce}(\hat{y}, y) = -\log p(y|x) = -\log p(y|x, w) = -\log \Pi p(y_i|x_i, w)$$

$$\Pi p(y_j|x_i, w) = \Pi_{i=1}^{N} \frac{\exp(-w_k x_i)}{\Sigma_{k=0}^{C} \exp(-w_k x_i)}$$

$$-\log p(y|x, w) = \sum_{i=1}^{N} (X_i W_{k=y_i} + \log \sum_{k=0}^{C} \exp(-X_i W_k))$$

Once we derive the gradient:

$$\nabla_{W_k} L_{ce} = \sum_{i=1}^{N} \left( X_i^T I_{[y_i=k]} - X_i^T \frac{\exp(-X_i W_k)}{\sum_{k=0}^{C} \exp(-X_i W_k)} \right)$$

If we assume $P_i$ to be the output of the softmax function which is probability of predicting class $i$, then we can rewrite the loss gradient as:

$$\nabla_{W_k} L_{ce} = \sum_{i=1}^{N} X_i^T I_{[y_i=k]} - \sum_{i=1}^{N} X_i^T P_i$$

$I_{[y_i=k]}$ is one if $y_i$ is in class $k$. This is the same as one-hot encoding of the label:

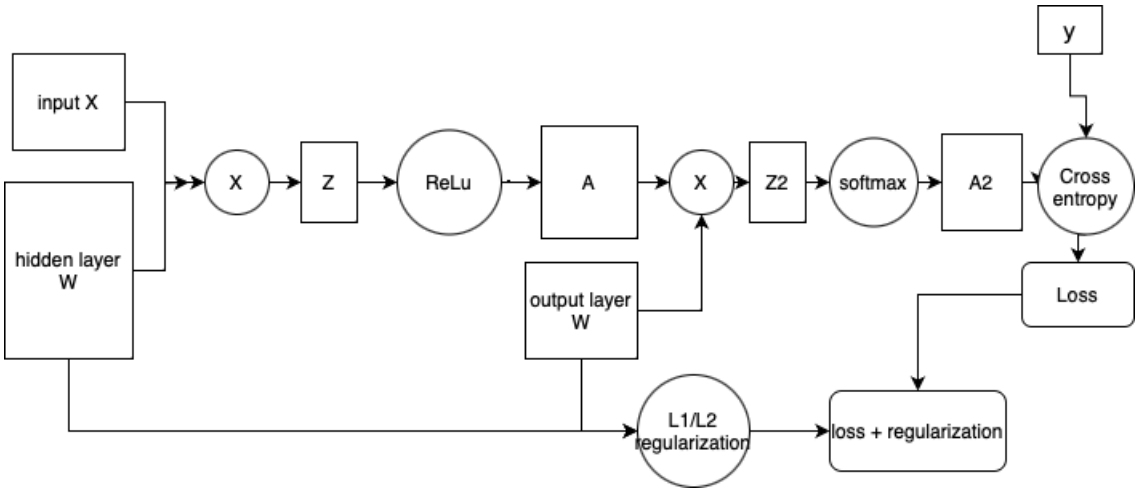
$$X^T (Y_{\text{onehot-encoded}} - P)$$

If we take the derivitave of the L1 norm, we get $sign(w)$ which is added to the gradient of loss with a regularizaiton term, called $\mu$.

For the feature selection of the training data, we use tfidf value of each word.

## 2 Question 2

Our Neural Network consists of one hidden layer with 50 nodes. The input layer is 3658 nodes for the dimensions of the feature space (which is the tfidf value of each token for that tweet) and the output layer is 6 nodes for the one-hot encoding of the output. The activation function in the hidden layer is the ReLu function and the activation function of the output layer is the softmax function. The loss function used in the neural network implementation is the cross-entropy loss (with one-hot encoded targets), combined with either L1 or L2 regularization. We used stochastic gradient descent with Adam optimization process to train this neural network.




The partial derivatives used in back propagation are as follows:

$$\frac{\partial(loss + reg)}{\partial loss} = \frac{\partial(loss + reg)}{\partial reg} = 1$$

$$\frac{\partial loss}{\partial A_2} = -(y/A_2)$$

considering l1 regularization,

$$\frac{\partial reg}{\partial W_1} = \lambda Sign(W_1)$$

$$\frac{\partial reg}{\partial W_2} = \lambda Sign(W_2)$$

$$\frac{\partial A_2}{\partial Z_2} = softmax'(Z2) = A_2(1 - A_2)$$

$$\frac{\partial A_1}{\partial Z_1} = relu'(Z_1) = \{1 \text{ if } Z1 > 0, 0 \text{ otherwise}\}$$

$$\frac{\partial Z_1}{\partial W_1} = X$$

$$\frac{\partial Z_2}{\partial W_2} = A_1$$

chaining these partial derivatives for the backward propagation are:

$$\frac{\partial(loss+reg)}{\partial A_2} = \frac{\partial(loss+reg)}{\partial loss}\frac{\partial loss}{\partial A_2} = -(y/A_2)$$

$$\frac{\partial(loss+reg)}{\partial Z_2} = \frac{\partial(loss+reg)}{\partial A_2}\frac{\partial A_2}{\partial Z_2} = -(y/A_2) * A_2 * (1-A_2) = A_2 - y$$

$$\frac{\partial(loss+reg)}{\partial W_2} = \frac{\partial(loss+reg)}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = (A_2 - y) * A_1^T + \lambda Sign(W_2)$$

$$\frac{\partial \text{Loss+reg}}{\partial Z_1} = \left(\frac{\partial \text{Loss+reg}}{\partial Z_2}\right)\cdot\left(\frac{\partial Z_2}{\partial A_1}\right)\cdot\left(\frac{\partial A_1}{\partial Z_1}\right) = (A_2-y)\cdot W_2^T\cdot \text{relu}'(Z_1) = (A_2-y)\cdot W_2^T\cdot(A_1>0)$$

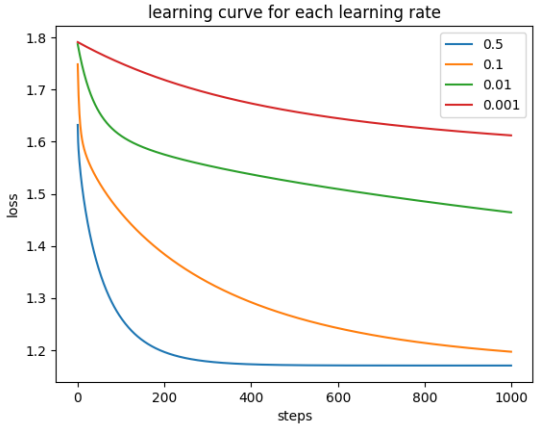
$$\frac{\partial \text{Loss+reg}}{\partial W_1} = \left(\frac{\partial \text{Loss+reg}}{\partial Z_1}\right)\cdot\left(\frac{\partial Z_1}{\partial W_1}\right) = ((A_2-y)\cdot W_2^T\cdot(A_1>0))\cdot X^T + \lambda Sign(W_1)$$

# 3 Question 3

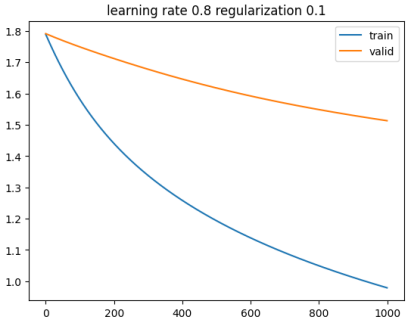
Multiple hyperparameters were tuned in the learning process. For logistic regression these parameters were the learning rate, the regularizing rate, and the regularization type. For neural networks these hyper parameters were the learning rate, regularization rate and type, number of hidden nodes, and batch size for gradient descent.

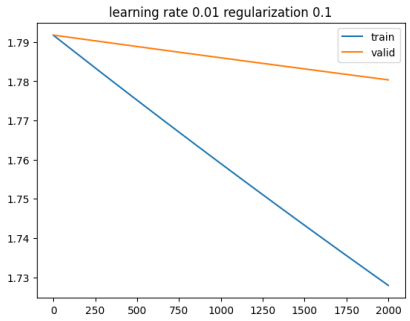
To find the best hyperparameters, we first use 5-fold cross validation and randomly divide the training data into 5 separate groups and train the model 5 times, each time considering one of the groups as a validation set. We average the model's performance on the validation sets and use this average performance to perform a grid search among the hyperparameters to find the best ones. We then once again try these hyperparamters with a single train-validation grouping to draw the learning graphs and decide if any overfiting is happening or not.

For the case of logistic regression, the learning rate (or $\eta$) was the main hyperparameter that was tunes. We can see in the graph comparing the training loss of the train data that the higher the learning rate, the earlier the model converges.




We can see by comparing the validation loss on the train data and validation data to see this effect also on the validation set




For the Neural network, the effect of the regularization rate $\lambda$ was analyzed.

It was seen that as the regularization decreases, the overfiting became worse to the extend that the validation loss started increasing in the early epochs. We know this increase in validation loss in caused by overfitting because the training loss curve converges too fast. So if we were only comparing training loss we might think a lower lambda is better, but adding the validation loss shows that overfitting is happening. As seen in the graphs below that with less regularization, the validation loss still decreases.