AI 2
Project 2

Violetta Gkika - 1115201600222

December 23, 2021

1 Exercise 1

From Standford lesson:
Cross-entropy loss with respect to logits

$$\hat{y} = softmax(\theta)$$

$$J = CE(y, \hat{y})$$

The gradient is

$$\frac{\partial J}{\partial \theta} = \hat{y} - y$$

if y is a column vector

$$(\hat{y} - y)^T$$

To compute the gradient of the cross entropy loss function with respect to its logits: We know that

$$\hat{y} = softmax(\theta) = 1/(1 + e^{-y}$$

Cross-entropy loss:

$$J = CE(y, \hat{y}) = -(ylog(y^{)} + (1 - y)log(y - y^{)}$$

2 Exercise 2

For this exercise I kept the names of the nodes same as the letter or operator they represent. For example the node with the sign + is called (node +).

Forward propagation steps/nodes: (node $*$) = $x * m$
(node +) = $(node*)\ + b$
(node Relu) = $max(0, (node+))$
(node Loss) = $(nodeRelu), y*$
So output graph $f = max(0, (x * m) + b) - y*$

To apply backward propagation we should first apply forward propagation. Then to apply backward propagation we compute the downstream gradients of each node by following the rule: [downstream gradient] = [upstream gradient] x [local gradient].
Calculation of local gradients:
$\partial(node*)/\partial(nodex) = \partial x * m\partial x$
$\partial(node+)/\partial(node*) = \partial(node*) + b\partial(node*)$
$\partial(node+)/\partial(nodeb) = \partial(node*) + b\partial b$
$\partial(nodeRelu)/\partial(node+) = \partial max(0 * node+)\partial(node+)$
$\partial(nodeLoss)/\partial(nodey*) = \partial y - y * \partial y*$
$\partial(nodeLoss)/\partial(nodey) = \partial y - y * \partial y$

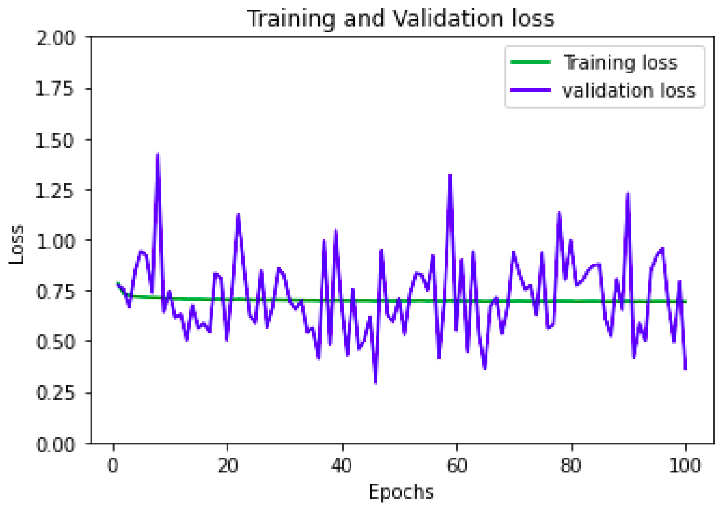
The upstream gradient of f is: $\partial f/\partial f = 1$

3 Exercise 3

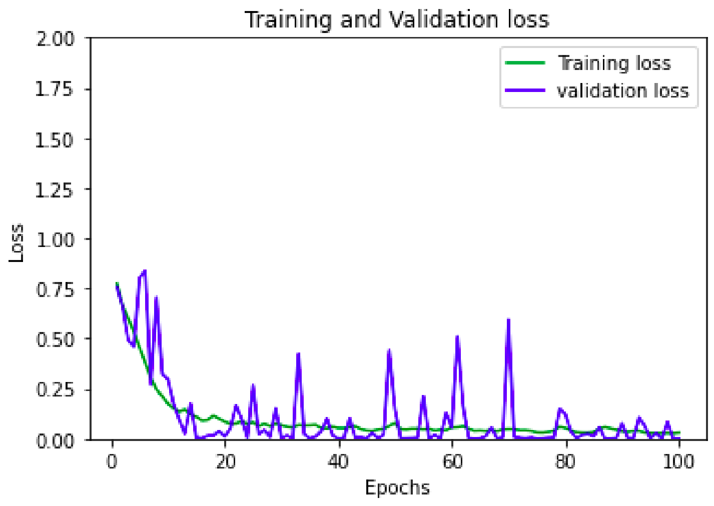
In the project file are implemented two different models. First model uses GloVe and the second one uses BOW vectorization.
In order to run each model, first one should run the three first cells that imports some libraries , read data files and define a preprocces methods. Then you can choose to run both models or one of them by running the cells under the index Model 1 or 2 correspondingly.
The preprocessing step is the same as in the Project 1. In addition , in this process I chose to normalize and standardise data using sklearn MinMaxScaler for normalization and StandardScaler for standartization.
I have chosen as best optimizer the Adam and as loss function Cross Entropy loss. (All the experiments shown uses this two parameters.)

Best model is the one using BOW as a vectorization step, so I performed most of my experimentation on this one.
In the process I experimented with different combinations of parameters but I am just showing briefly some things I tried that helped me optimize my model, other experiments that had a negative impact on the model are excluded eventhough they had e key role on understanding what works best.

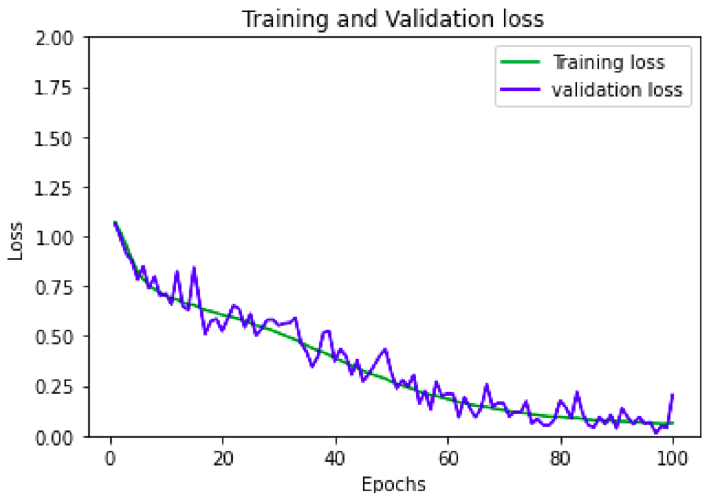
Firstly let's see how the model performs with a simple network of 4 linear layers , as the one presented on the course. Learning rate 0.00001 , batch sizes 64, and input sizes 128, 64, 32, 3. All running for 100 epochs.
The first model shown in the pictures do not even train right.



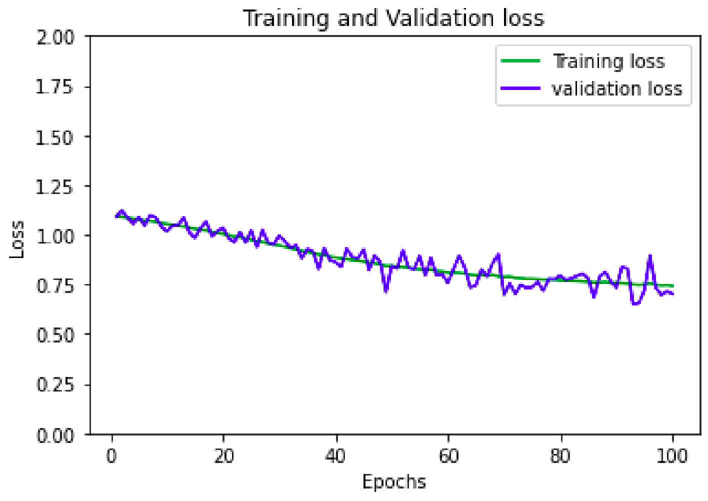
The second model after increasing the learning rate to 0.001 is clearly overfitting.



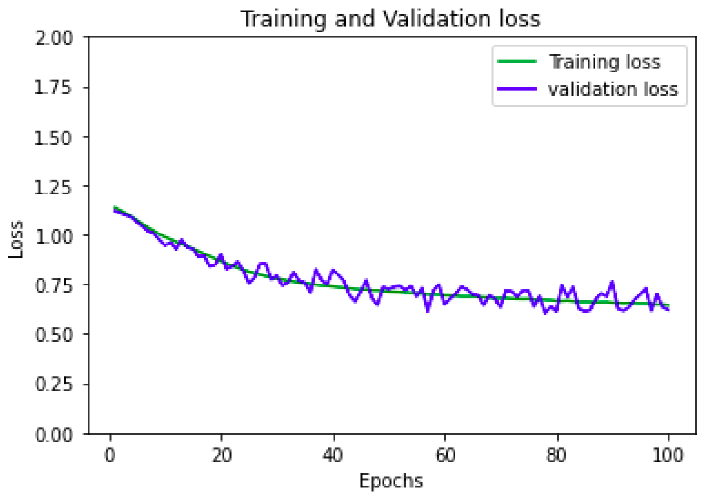
To prevent overfitting I tried lowering the learning rate and increasing the number of batches to 512.There is still some overfitting but this changes had a positive impact since now the validation loss is closer to the training loss.



All this three cases has a F1 score around 0.9 for training and 0.6 for validation. This large gap is also an indicator that the model overfits. What would affect most the model is the activation functions. So, changing the layers of the network lead us loss vs epochs plot shown below. Layers $linear->relu->linear->dropout->linear->relu->linear$. F1 score for training is 0.5272875657539309 and for validation 0.525180334809346. The model do not overfit anymore but the performance is low and the validation curve still fluctuates.

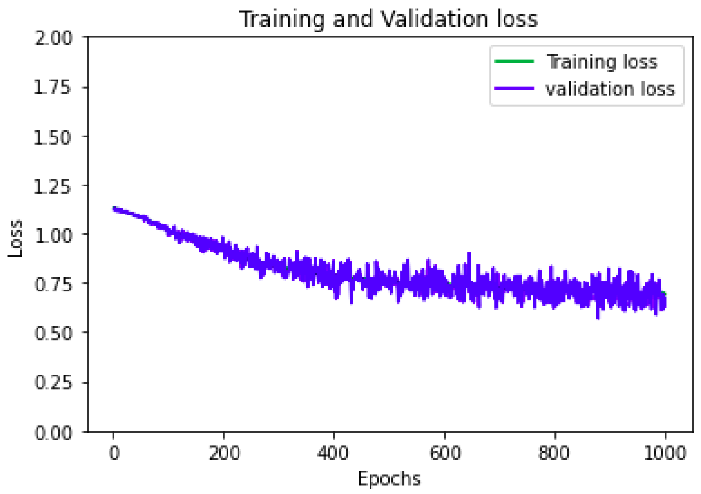


For a learning rate = 0.00001 and layers $linear->dropout->relu->linear->droput->linear->relu->linear$

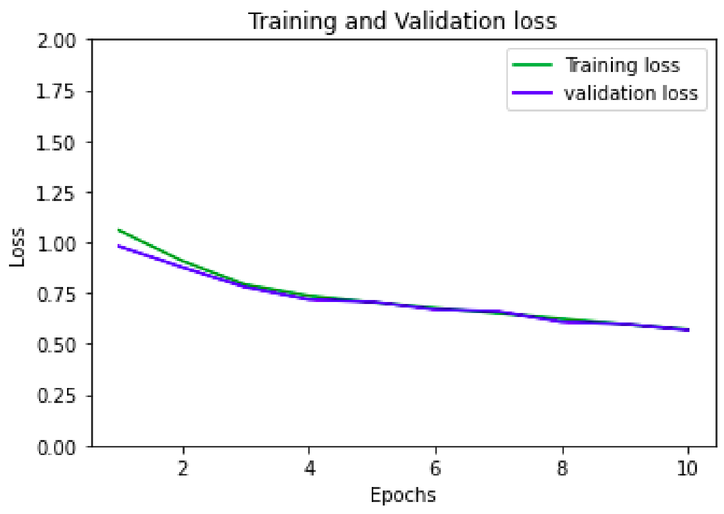


F1 score for train is 0.6290567071916656 and for validation 0.6122619472949633 which is a pretty good score compared with the one from the Project 1 but still lower. The validation curve still fluctuates and this maybe for a number of reasons but this is the model with the best accuracy and the smaller gap between train and validation F1 score.

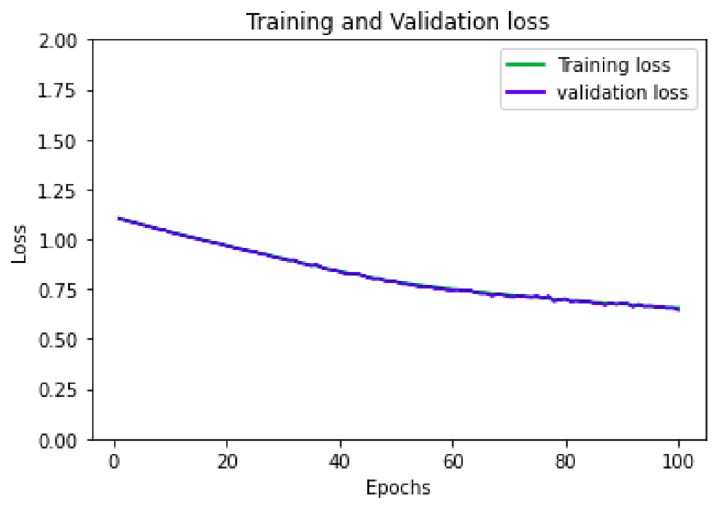
If we increase the number of epoch and lower the learning rate to 0.000001 the performance is better (F1 score for train around 0.67 and for validation around 0.65).



As the number of epochs increase the fluctuation of the validation curves get worse. So for the same network if we lower the number of epochs to 10 and increase the size of batches to 2048. Learning rate 0.001 we can see that the validation curve flattens and we have a almost good accuracy. F1 score for train 0.6510014221916915 and F1 score for validation 0.6306317990295077



What I realize is that for a large number of epochs and a very large number of batches around 4000 we are able to flatten the validation curve but the accuracy decreases.



So, in conclusion the best model has a large number of batches = 2048,10 epochs, learning rate = 0.001 and layers $linear- > dropout- > relu- > linear- > droput- > linear- > relu- > linear$.

Sources:

https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e