# CMU 10-715: Homework 2 Report

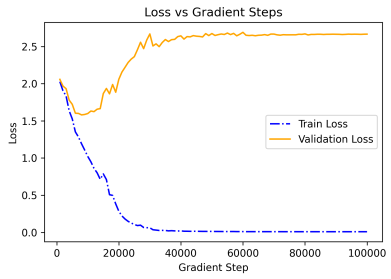
Implementation of Dropout Classifiers

**Abishek Sridhar (Andrew Id: abisheks)**.

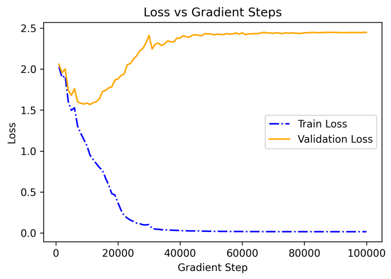
## 1 Results

a I trained the classifiers for hundred thousand steps on Google Colab's GPU.

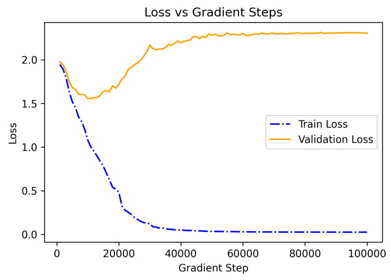
b Refer figure 1.



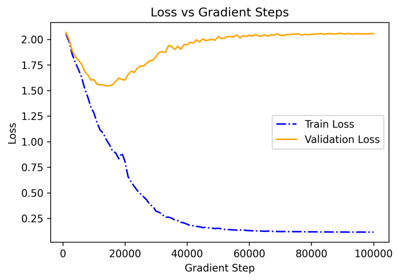
(a) Loss vs Gradient Steps for dropout = 0.0



(b) Loss vs Gradient Steps for dropout = 0.2



(c) Loss vs Gradient Steps for dropout = 0.5



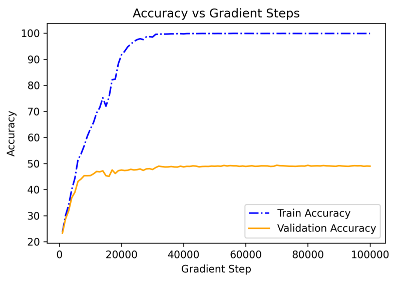
(d) Loss vs Gradient Steps for dropout = 0.8

Figure 1: Loss vs Gradient Step plots for different dropout probabilities

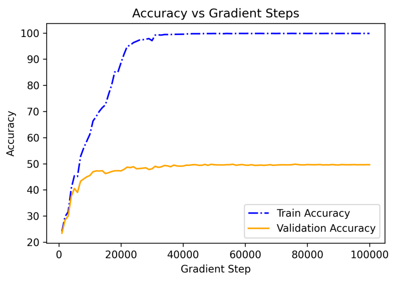
c Refer figure 2.

d Refer table 1 for the final results of the training.

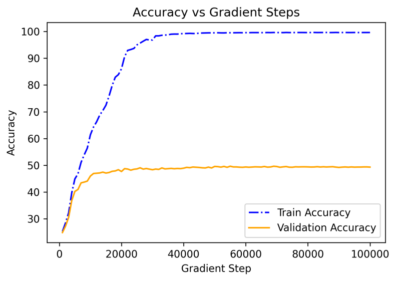
Generalization deals with the ability of the model to classify unseen samples correctly. Regularization is a technique adopted to constrain the



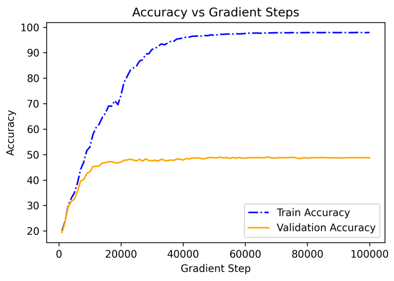
(a) Accuracy vs Gradient Steps for dropout = 0.0



(b) Accuracy vs Gradient Steps for dropout = 0.2



(c) Accuracy vs Gradient Steps for dropout = 0.5



(d) Accuracy vs Gradient Steps for dropout = 0.8

Figure 2: Accuracy vs Gradient Step plots for different dropout probabilities

model's learning in a manner it does not learn spurious features or correlations exclusive to training samples and lose its generalizability.
Lets consider the case of no dropout ($p = 0.0$). There is a large gap between training and validation loss (and accuracy), which indicates the model is not able to generalize well and clearly overfits the training set (so much that the model almost achieves 100% accuracy on the training set). On adding regularization in the form of dropout, we see that the train accuracy and loss increase as the dropout probability increases (i.e) as the model is more and more constrained to not completely fit the training set. The validation loss decreases with $p$, and this might be because the model does not learn the dubious features that make the model confident of its errors. Notice that, the validation loss and accuracy are not inversely correlated here, as observed generally, because the model becomes increasingly confident of its errors during training that increases the loss even though validation accuracy rises.
However, the most important measure of generalization that we care about

here, validation accuracy, increases and then decreases on increasing $p$. This relation between regularization and generalization is what we usually witness for common regularization techniques like R1/R2-regularization, early stopping, etc. where, as the constraint during the training increases, the increasing focus on regularization starts nullifying the ability to generalize after a certain point. Intuitively, for dropout we explain this as below:
As the dropout sets in and $p$ is increased from 0.0 to 0.2, the occasionally dropped neurons force the model to let go of correlations exclusive to training set and focus on more meaningful features for prediction, that aid generalization as well. However, on increasing $p$ further to 0.5 and then to 0.8, neurons units are dropped often and consequently the updates to the weights become sparse. Hence, the model is not able to learn some necessary features as well, that affects the classification on training and validation (unseen) set.

| **Dropout p** | **Train Loss** | **Val Loss** | **Train Acc** | **Val Acc** |
|---|---|---|---|---|
| 0.0 | 0.0095943 | 2.6656424 | 99.90 | 49.00 |
| 0.2 | 0.0145671 | 2.4493456 | 99.84 | 49.62 |
| 0.5 | 0.0264440 | 2.3030356 | 99.63 | 49.32 |
| 0.8 | 0.1155117 | 2.0547932 | 97.93 | 48.70 |

Table 1: Table showing the train and val loss, accuacies for different dropout probabilities

e The technique of dropout is motivated from the effect of ensembling different models to evade overfitting. Here, the different models are obtained by independently dropping each neuron units with a certain probability $p$ for every training case. Hence, training a neural network with $n$ units can be thought of as training a collection of $2^n$ thinned networks (but with the same number of shared parameters).
The ideal way to ensemble this collection of models is to average the predictions of these exponential number of models, but this is infeasible during testing time in terms of space and time. Hence, the paper proposes an approximate method of using a single complete neural network with scaled down weights during test time by multiplying them with neuron retention probability of $1 - p$ (where $p$ is probability of dropping units). We know that each neuron is dropped with probability $p$ during training, and hence the outgoing weights from that neuron are set to zero with probability $p$ during training. Hence, if the learned weight is $W$, the expected value of the weight (under the distribution used to drop neurons at training time) is $(1 - p) \cdot W + p \cdot 0 = (1 - p) \cdot W$. To keep the expected value of the weights same during training and testing time, the weights are multiplied by a factor of $1 - p$, which scales it down. This can be thought of using weights averaged from those of the collection of thinned networks, rather than using the average of predictions at test time as an efficient approxi-

mation.

[**Note:** I am using probability $p$ to denote the probability of dropping neurons (or setting units to zero) as specified in PyTorch. Hence, the probability of retention (which is denoted as dropout probability $p$ in the paper) is $1 - p$ here]