Artificial Neural Networks

1st Assginment

Saeid Cheshmi- 9822027

1. *Dataset*

In exercise 2, we worked with Fashion-MNIST which consists of 10 classes with 60,000 samples as training set and 10,000 as testing set. Each image in the dataset has size of 28 * 28 pixels and 1 channel (gray scale). Our goal is designing a simple MLP to classify these images.

2. Dataset Loading and Preprocessing

For dataset loading we used '**torchvision'** module which is a sub-module of **PyTorch** which is used for images problems. We split our dataset into train, validation and test sets. We considered 50,000 examples as train set, 10,000 as validation set and 10,000 as test set.
We didn't use any specific augmentation for images but converted them to Tensor. Also, we shuffle the datasets and used batch size of 64 for batching the data.
We checked for **Cuda** availability and use GPU if it is available. Also, we set random seed for reproducible results.

3. *Models*

3.1. Simple Model

At First, we proposed a simple MLP which consists of 3 layers. They contain 64,64 and 10 neurons respectively. We used CrossEnropyLoss as our loss function and Adam for optimizing our weights. We considered learning rate of 0.001 and 15 number of epochs. After training this model we got 91.69 % accuracy on train set and 88.02% accuracy on test set. Fig1 shows the training and validation loss.

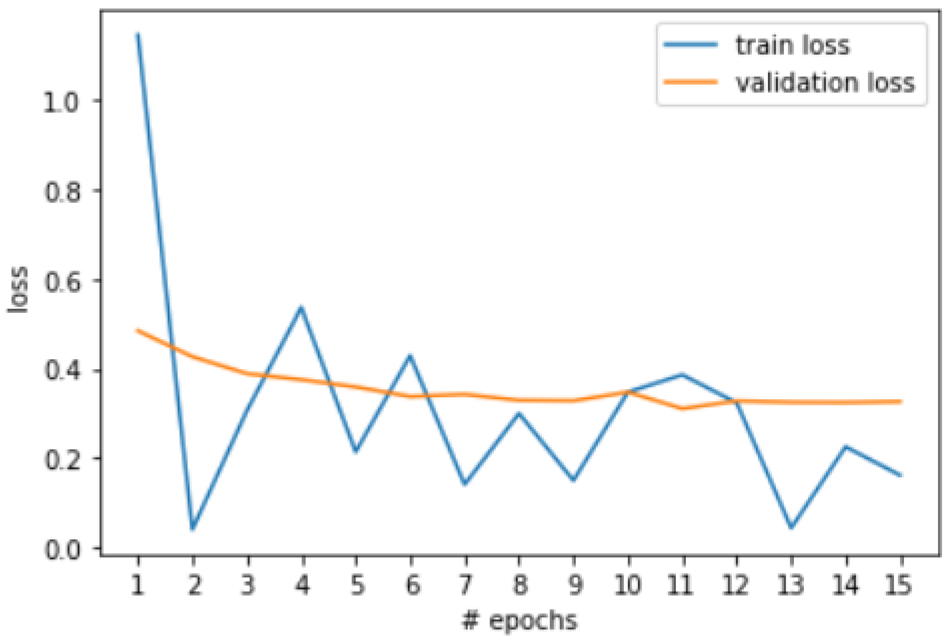

Figure 1. Simple Model training and validation losses

3.2. Second Model

For second model, we increased the neurons of each hidden layers of last model. Here, we have 128 neurons in each hidden layer. After training the model with the same above configuration which mentioned for last model, we got nearly 92% accuracy on train set and 87.21 % accuracy on test set. As you can see, we had a little improvement on training set but our test accuracy decreased.

3.3. Third Model

For third model we increased both of neurons and number of hidden layers to increase model complexity. In this model we have 64, 128, 256 neurons in hidden layers. After training model with the configurations as above, we got 92.20 % accuracy on training set and 88% accuracy on test set. it shows our we got better accuracy on training set but our model didn’t have significantly improvement on test set.

4. Dropout

One way to address overfitting is using dropout technique. Dropout refers to the practice of disregarding certain nodes in a layer at random during training. We added dropout with probability of 25% in each hidden layer of our third model. After training model with 15 epochs, we got 91.55% accuracy on train set and 87.62% accuracy on test set. As you can see, dropout didn’t have any specific effect on this problem. It may be due to low number of epochs. In below figure shows train and validation loss.

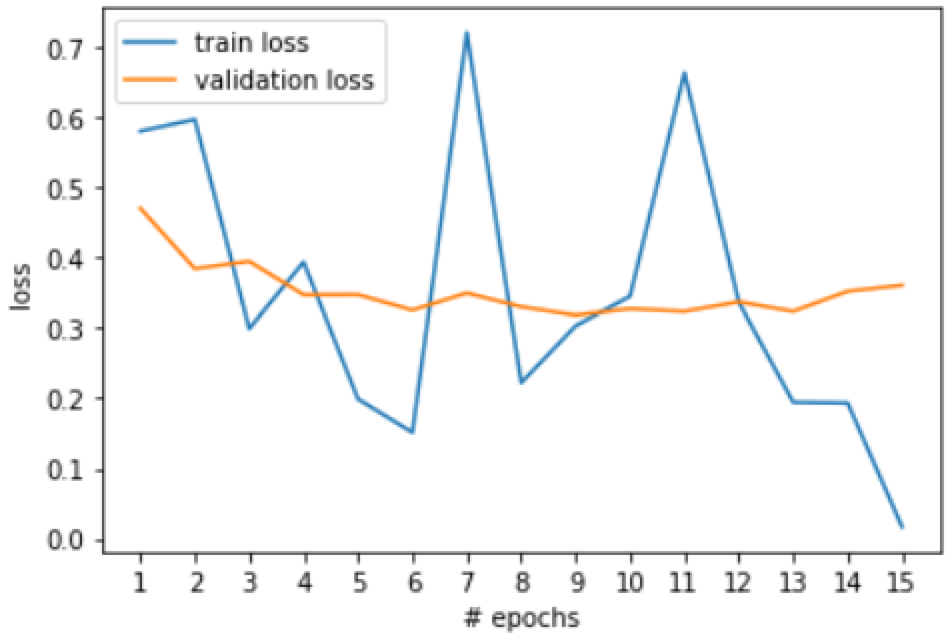

Figure 2. Dropout model

5. Early Stopping

Early stopping is another way to avoid overfitting. In early stopping we track a metric and if that metric doesn't improve during training we stop training. Usually, we choose validation loss or validation accuracy for this purpose. Pytorch doesn't have built-in early stopping, so we implemented a **EarlyStopping** class for this purpose. It gets an integer as patience which mean how many epochs waits for stopping. Here we used validation loss for early stopping. We trained our model with 20 epochs and 3 patience, but after 11 epochs validation loss didn't improve and early stopping function stopped training. We got 90.52% accuracy on training set and 86.56% accuracy on testing set. you can see train and validation loss below.

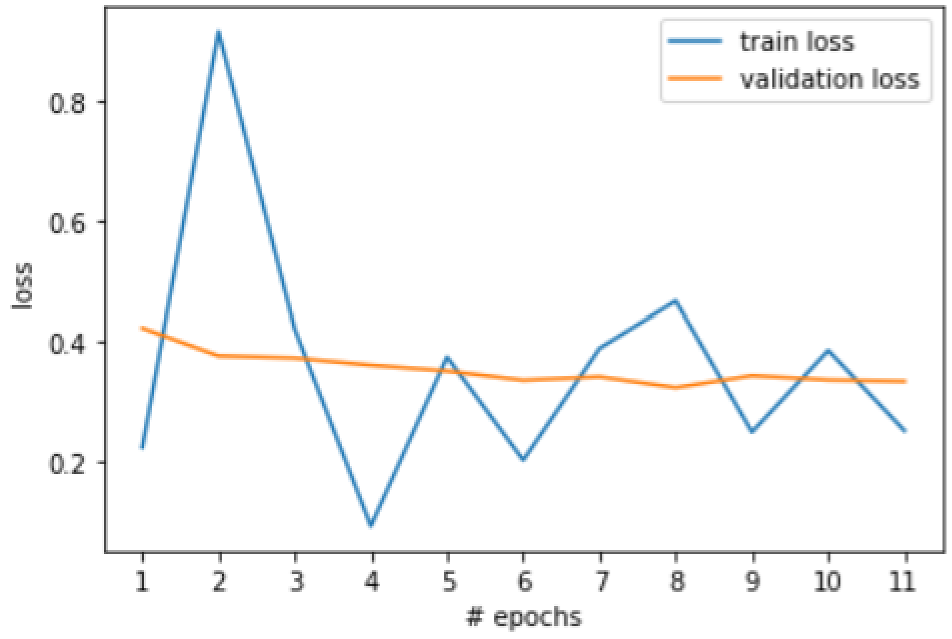

Figure 3. Training with early stopping

6. BatchNormalization

Batch-Normalization is an algorithmic method which makes the training of deep neural networks faster and more stable. It consists of normalizing activation vectors from hidden layers using mean and variance of the current batch. Batch-Normalization can be apply after or before non-linear activation function. It also helps model to avoid overfitting. We applied batchnorm after activation function. After training model for 15 epochs we got nearly 93% accuracy on train set and 88.46% accuracy on test set. We can see a little improvement on both test and train accuracy. In Fig4 you can see train and validation loss.

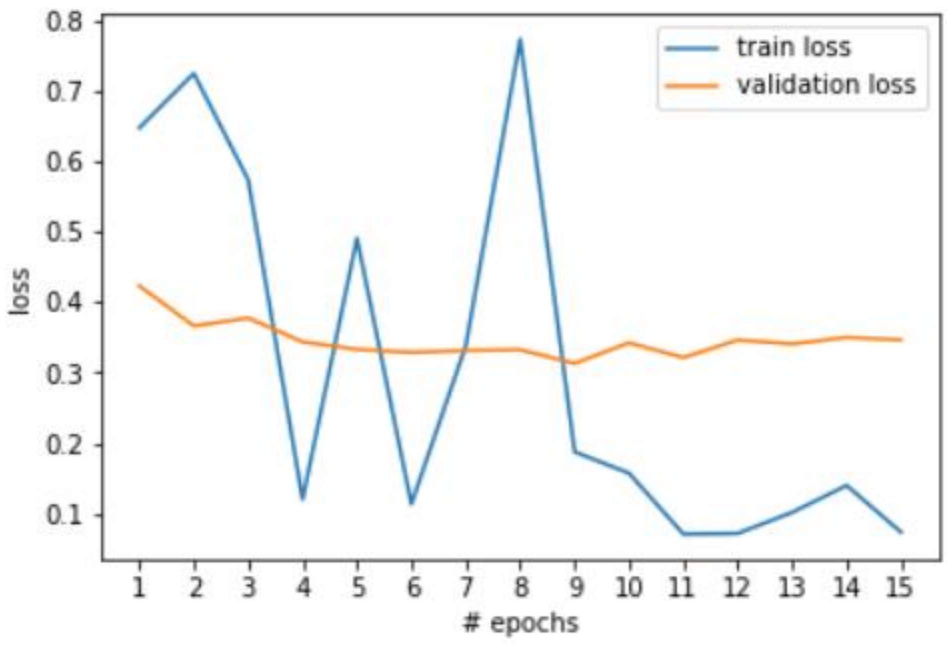

Figure 4. Batch Normalization

7. L1 and L2 Regularization

Adding norm L1 and L2 term to loss function is another way to avoid overfitting. PyTorch doesn’t have built-in function for this purpose. We implemented these techniques, you just set which norm you want to and lambda coefficient which determines effect of regularization. For L1 regularization we set lambda to 0.001 and trained the model. After training we got 83.73% accuracy on train set and 82.43% accuracy on test set. We can see L1 regularization had very good impact on reducing overfitting. We got closer accuracy on train and test set.

We also trained model with L2 regularization. We set lambda to 0.001 again. We got 91.66% accuracy on train set and 87.73% accuracy on test. As you can see norm L1 has more regularization impact.

In PyTorch you can apply L2 regularization in another way. You can set the ‘weight_decay’ parameter of your optimizer as lambda coefficient. We also used this method and got nearly 90% accuracy on train set and 86.86% on test set.

8. Exercise 1

L1 and L2 regularization are most common type of regularization. We add a regularization term to cost function and try to minimize it.

Cost function = Loss () + Regularization term

By adding regularization term to cost function, the values of weight matrices decreases because it assumes a neural network with small weights leads to simpler model. It also reduces overfitting.

L1 regularization:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

L1 penalty which is an absolute value of the magnitude of the coefficient or weights using which we restrict the size of coefficients The weights may be reduced to zero here. In L1 regularization less important weights shrinks to zero. This works as a feature selection when we have a lot of features.

The goal of L1 regularizer is to minimize norm of parameter vector. This is as the same of optimizing a neuron’s parameters or a single layer in neural network. In other word, it is a method of maximizing the area of the parameter hyperspace that the true parameter vector is within.

L1 regularization is easy to implement, it also is robust to outliers. It creates sparsity in the solution which means that most of coefficients of solution are zero. It tends to less important features or noise term will be zero.

L2 regularization:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

We add an L2 penalty which is basically square of the magnitude of the coefficient of weights. By using L2 we can perform small changes in the weights and using these penalties in regularization we can penalize large weights more severely. L2 regularization forces the weights to be small but does not make them zero and does not give the sparse solution. This type of regularization isn’t robust to outliers. Also, L2 can learn more complex pattern.

L1 regularization leads to sparse solution in comparison to L2 regularization. Here sparsity means most of values have exact zero value. In the field of neural network, we can say that the L2 penalty is used for decaying the weights that is why it is the most used approach for regularization or weight size reduction.

Also, L1 it is very useful when we are trying to compress our model.

- Intuitive difference between L1 and L2 regularization is that L1 tries to estimate median of data while L2 tries to estimate mean of data.
  In L1 regularization, while we take derivative of cost function it will estimate around the median of dataset. For example, suppose we take an arbitrary data point. If we go in some direction in some distance d, in backward direction the values when calculating loss, the values of left side of the chosen data point will have a lesser loss value while on another side will contribute more in the loss function calculation. So, to minimizing the loss function we should estimate a value around median of data.
  On the other hand, in L2 regularization when we calculate the loss function gradient, the loss function tries to minimize the loss by subtracting it from mean.
  As mentioned earlier, another difference between these two types of regularization is that L1 regularization tries to eliminate not important features.

In conclusion, L1 regularization needs less computation in comparison to L2 regularization. If we want our model be robust to outliers and learn simple patterns, we can use L1 regularization, but if we want our model learn more complex patterns, we can use L2 regularization. We also can use both like elastic net.

9. References

- https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf
- https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c
- https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/