16824 HW3 Report

Jagjeet Singh (jagjeets)

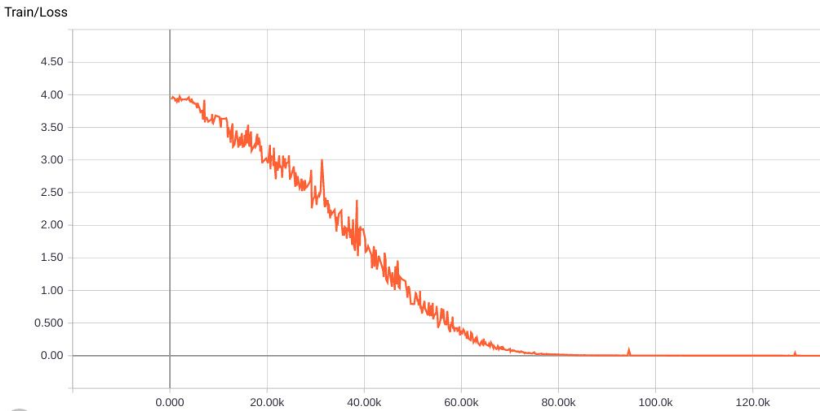
Task 1: Action Classification

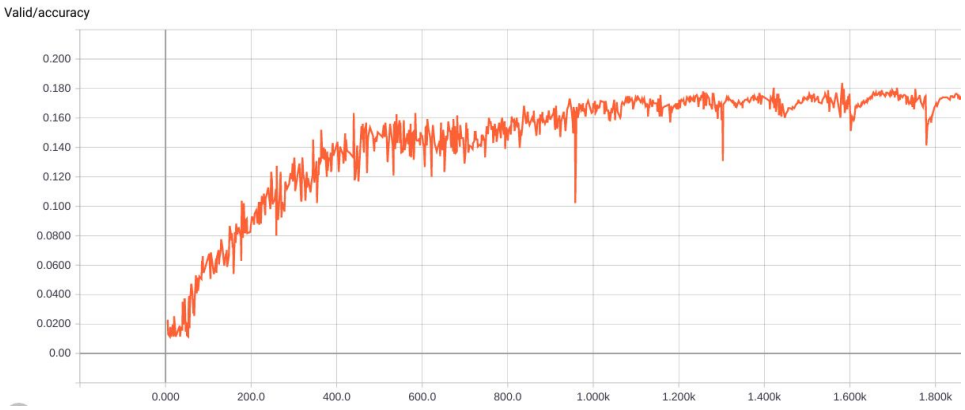
1.1 MLP: Details of the Model:

Ans:

- Framework: PyTorch
- Dataloader: BatchSize x 10 x 512
- TrainVal split: 0.7
- Network Architecture:
    - Linear Layer: 512 -> 256
    - Linear Layer: 256 -> 256
    - Linear Layer: 256 -> 51
- Activation: tanh (at 1st and 2nd Linear layers)
- Initialization: Xavier Normal
- Mean is taken along the 2nd dimension of the network output.
- Loss: Cross Entropy Loss
- Optimizer: Adam
- Learning Rate: 0.0001
- Batch Size: 32
- Evaluation: Max of all the 51 channels is taken as the predicted output and then the model is evaluated on Accuracy.

Plots:





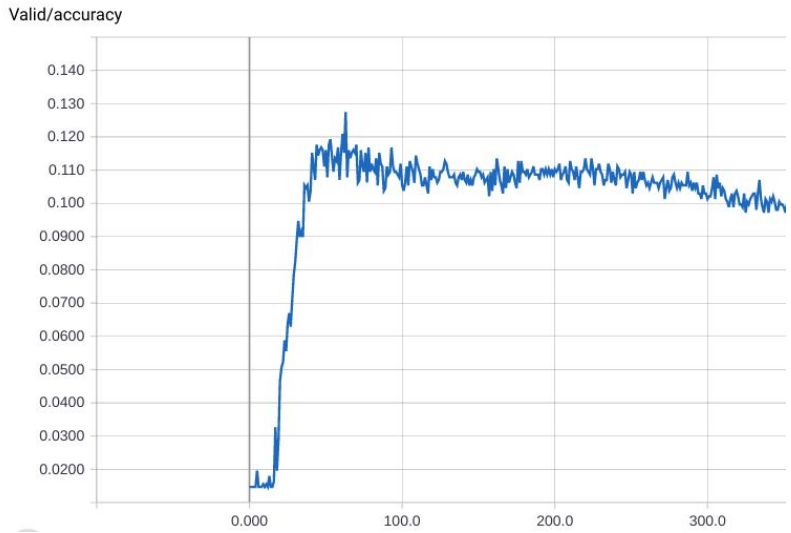
Best Validation Accuracy: 18.2%

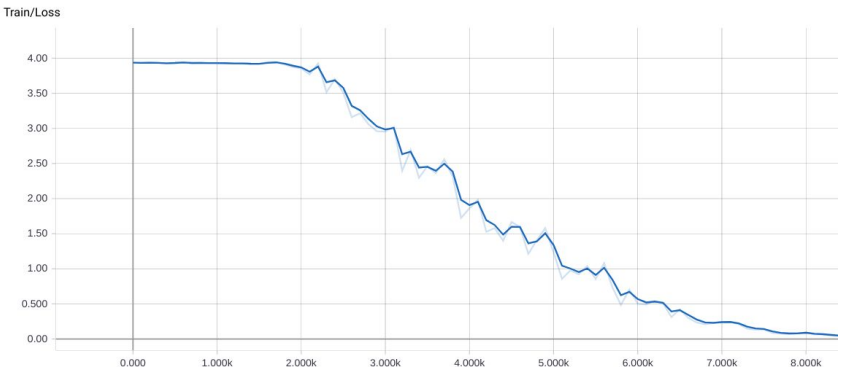
1.2 RNN: Details of the Model:

Ans:

- Framework: PyTorch
- Dataloader: BatchSize x 10 x 512
- TrainVal split: 0.7
- Network Architecture:
    - Linear Layer: 512 -> 256
    - LSTM(input_size=256, hidden_size=256, num_layers=2, bidirectional=False)
    - Linear Layer: 256 -> 51
- Activation: None
- Initialization: Normal for Linear Layer, Zeros for LSTM hidden states
- Mean is taken along the 2nd dimension of the network output.
- Loss: Cross Entropy Loss
- Optimizer: Adam
- Learning Rate: 0.0001
- Batch Size: 32
- Evaluation: Max of all the 51 channels is taken as the predicted output and then the model is evaluated on Accuracy.

Plots:





Best Validation Accuracy: 12.8%

Task 2: Q&A

1. **Why does using a regression loss for surface normal estimation lead to blurry results?**
   Ans:
   - During training, we have access to surface normals which correspond to real image. Hence, regression loss tries to minimize the distance between predicted and ground truth surface normal.
   - However, during testing, the predicted surface normal may not correspond to a real image. For example, horizontal and vertical surfaces are connected by a sharp edge. The input features might not vary too much across the edge but the actual surface normal changes drastically. When predicting the surface normal at this edge, we are trying to infer from the output of similar input features and hence, will end up taking

an intermediate value, i.e., one which is neither vertical nor horizontal. Such behavior at multiple pixels in the image will result in blurry results.

2. **What is Batch Normalization and why is it useful? How does Batch Normalization vary for training and testing phases?**

   Ans:

   - **What**: Batch Normalization is a training trick that reduces the covariate shift between different mini batches. It is applied after the weighted application of inputs but before the activation
   - **Why:** Training assumes that all minibatches have similar distribution. However, in reality, there is a covariate shift which can be huge at times. This can affect the training badly. As such, Batch Normalization is used to first move all batches to a mean of 0 and unit standard deviation. After that, 2 parameters (gamma and beta) are learnt which provide some flexibility in mean and variance.
   - **How:** In training, the batches are normalized using the mean and standard deviation of that particular batch. After this the parameters are learnt which shift the mean normalized value. In testing, everything is the same except that the mean and standard deviation used is obtained from all the TRAINING batches. Also, the parameters (gamma and beta) are obtained from the converged network.

3. **In a two stream network, when should you have shared weights and when separate?**

   Ans:

   - We should have shared weights when the inputs and outputs of the 2 streams belong to the same feature space and both the streams are trying to achieve a similar objective. For example, in Siamese networks, 2 images are passed into 2 streams and the outputs are in the same space. It is useful because there are lesser parameters to be learnt and can achieve objectives like similarity score.
   - Weights should be different when the 2 streams have nothing in common and they just want to use the same features obtained from prior part of the network. For example, in Faster-RCNN, the two streams try to predict the classification and detection score. They belong to totally different space and hence the weights are not shared.

4. **You are training a two stream network for RGBD data (one stream for RGB and the other for D). Your dataset consists of 10,000 images for recognition. How would you initialize the network? How will you avoid overfitting in the network?**

   Ans:

- RGB stream can be directly initialized using a Pre-trained ImageNet. For D-stream, there are different ways of encoding it as an RGB-image:
    - Rendering depth as grayscale and replicating depth for 3 channels.
    - Using surface normals where each dimension of normal corresponds to one of R,G,B
    - Normalize depth values and then use a color mapping like *jet* to convert it to an RGB image
- Once the depth input is transformed, they can also be initialized with pre-trained ImageNet.
- For preventing overfitting, things like Dropout, L1/L2 regularization, reduce network complexity etc. can be used. Data augmentation can be very helpful in this case to avoid overfitting. The depth data can be augmented with missing data patterns.

5. **How would you convert a regression problem to a classification problem?**

   Ans:

   - Instead of predicting the exact output, predict the mean and standard deviation of the output distribution. Then sample the required predictions from the distribution and get the conditional probability of getting the predicted values, given the true values.

6. **In a multi label image classification what loss would you use?**

   Ans:

   - Binary Cross Entropy Loss can be used for multi-label image classification. For each label, we can calculate the one-vs-all binary cross entropy loss and then sum over the losses obtained for all the labels

7. **Why are CNNs used more for computer vision tasks than other tasks?**

   Ans:

   - Conventional MLPs are not positional invariant. However, in Computer Vision, many problem statements don't care about the exact position of features. They are mainly concerned about the presence of any particular object (feature) in the image. As such, this task can be achieved by learning much fewer parameters. CNN work by sharing the parameters for different weighted operations using convolution operations. This is difficult to do in other domains like language and speech because they are ideally not positional invariant.

8. **How do you find the best hyperparameters like learning rate, momentum, type of optimizer etc. for a new task at hand?**

   Ans:

   - There is no rule of thumb to find the best hyperparameters, but there are some best practices. There are well-defined approaches like grid search but it's difficult to get them working on large networks. Some hyperparameters and their usage is as follows:
     - Number of hidden layers
     - Dropout: Use small dropout (0.2 to 0.5) and avoid dropout in smaller networks
     - Network Weight Initialization: Normal/Xavier Initialization work for most cases
     - Activation Function: Avoid ReLU in smaller networks but prefer ReLU in large CNNs. Avoid sigmoid when mean 0 is a requirement.
     - Learning Rate: Start with a high value (eg. 0.1) and keep decaying it. Increase the learning rate if stuck in local minima.
     - Momentum: Mostly 0.9 works. If it doesn't, pick values in the range 0.5-0.9
     - Number of epochs: Keep training until validation accuracy starts to decrease even when training accuracy is increasing.
     - Batch size: Prefer larger batch size if training on GPU (provided they fit in GPU memory). If the input dimension is too large, keep smaller batch size.
     - For example, for learning rate, we can follow a decaying learning rate, for optimizers, Adam is widely used, etc.
     - Optimizer: Prefer SGD for shallow networks and Adam/RMSProp for deep networks
   - Best way to find the values for these is to train multiple times with different setups and keep a track of results of each setup.