Semi-supervised learning on MNIST Hand Written Data

Tuesday, March 7th, 2017

1 Semi-supervised learning using Psudo-label

1.1 Model Architecture & Configuration

- Architecture
  The overall architecture of our used model for this assignment is as follows:

  Dataset: MNIST Handwrtten Digits
  Labeled data: 3,000 $\Rightarrow$ After data augmentation: 18,000
  Unlabeled data: 47,000
  Validation data: 10,000

  We have a simple two-layer convolutional neural network which has kernels of size 5x5. After each layer, we apply max-pooling and activate the units with a Relu nonlinear function. At the second layer, we also apply a dropout with probability of 0.5 to control overfitting.

  In order to perform the semi-supervised learning task, we have implemented a psudo-label approach. Since we are having much more unlabeled data than labeled data, we have first increased the size of the labeled data using data augmentation, the augmentation methods we used will be described in a later section:

  Using data augmentation has increase our labeled data to a size of 18,000, which is more comparable to the unlabeled data. For each epoch we have a ratio of 1:2 for labeled and unlabeled data when the model enters the psudo-label phase. We also use a pre-trained kernel to initialize our neural network, that is:
  we first learned 10 clusters in the filter size of the first convolutional layer from the training data using K-means clustering. Then, the 10 clusters are utilized to initialize filter weights of the first convolutional layer.

  After data preparation and first-layer kernel initialization, We begin training the model with only the labeled data for 100 epochs, then using the 100-epoch trained model to apply prediction on the unlabeled data, such that we have generated the psudo labels for the unlabeled ones. For the following epochs, we update our optimizer using a new momentum which is calculated using the formulae (10) - (13) in [1]. The loss function that we use is still the *Cross Entropy*, but for the unlabeled data, we need to enforce a coefficient $\alpha(t)$ which is to balance the difference between the data (labeled and unlabeled) for model performance. $\alpha(t)$

is calculated using the formula (16) in [1]. Then after one epoch training, we use the new predicted psudo labels for the next training round, and keep doing the same until the stop criteria reach.

The final model architecture using data augmentation, and psudo-labels altogether gives us a validation accuracy of 0.9828.

- Configuration
  Batch size: Labeled: 64, Unlabeled: 128
  Number of Epochs for training: 400
  Network:
  conv1: in: 1 channel , out 20 hidden units conv2: in: 20 hidden units, out 40 units
  initial learning rate: 0.01
  initial momentum : 0.7

1.2 Comparsion of improvements using methods in Section 2 and Psudo labels

In this section, we will show our initial experiments of using the different boosting methods mentioned above (K-Mean kernel initialization, data augmentation and psudo-labels). The learning rate and momentum are fixed in these experiments. Validation accuracy after 300 epoches for each model are shown in Table 1. Train/validation/validation accuracy plots are shown in Figure 1.2

- K-means Clustering
  First of all, we begin with learning 10 clusters of size 5×5 using all the labeled training data set. In each image, we shift the filter window from left to right and top to bottom by a step size of 2 pixels. For each image, we will obtain 100 5×5 matrices. After a single loop of 3000 training images, we apply the K-means clustering to get 10 clusters and employ them to initialize the filter weights of the first convolutional layer in the model.

  After this step, the performance has been improved by 0.96 percentage point.

- Data Augmentation

  Methods used:

  - Rotation:
    We rotated each of our original training data randomly from -30 degree to 45 degree. Since by rotating these certain degrees, the images could still be recognizably the same digit. While, at the pixel level, these rotated images could be seen as new training data.
  - Shift:
    We defined 8 directions: upper left, upper, upper right, left, right, lower left, lower, lower right. Then we moved each of our original digit 4 pixels in the directions we defined.
  - Zoom:
    We zoomed each of our original training digits by 0.7 or 1.3 times randomly.

  Originally, we have 3000 labeled data as our training data. Then, we apply our data augmentation methods mentioned above. First, we use the rotation method 3 times to generate 9000 new data and thus expand our training data from 3000 to 12000. Second, we use the shift method to expand our training data from 12000 to 15000. Finally, we apply zoom method to expand our training data to 18000.

Training the model with data augmentation has increased the validation accuracy by 0.99 percentage point.

- Psudo-labels
  Psudo-label is used after epoch 100, then momentum and the unlabeled data loss coefficient is changing in each future epoch as we explained in Section 3.1. This psudo-label method has increased the validation accuracy by 0.81 percentage point.

Models	Validation Accuracy
Training with initial starter code setup	97.07
With kernel initialization using KMeans	98.03
With data augmentation	98.06
With psudo labels	97.88

Table 1: Experimental validation accuracy with different methods

1.3 Discussion

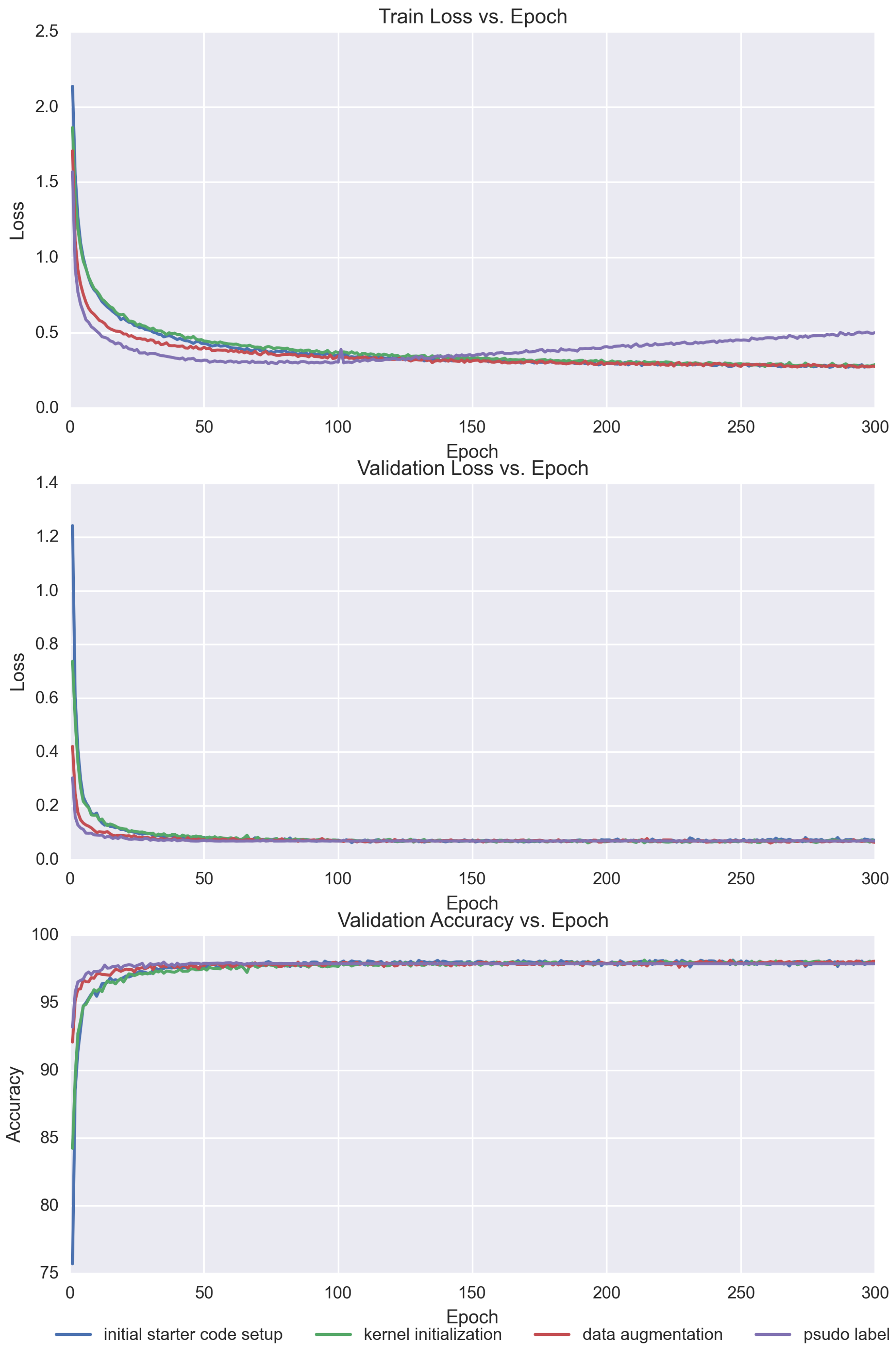
From Table 1, we can see that even without any boosting methods, using a small labeled dataset of size 3000 can already train a quite good model for the MNIST handwritten digit recognition task. From Figure 1.2, it is clear that all models converge to a steady state. Data augmentation has the fastest convergence. All boosting methods give lower initial train/validation loss and higher initial validation accuracy.

Furthermore, as we experimented in using different configurations for the model, we have found that more the model is easy to get to saturated point where a small number of epochs is good enough for model training, such as around 300 or 400 epoches with a reasonable learning rate. More hidden units does not give much boosting as the task is not very complex, a slightly higher number compared with the original started code is already good enough, in our case, we double the hidden units in each layer which generates better result that the ones with a multiple of 4.

References

[1] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. 2013.

Figure 1:



Train Loss vs. Epoch
Loss
Epoch
Validation Loss vs. Epoch
Loss
Epoch
Validation Accuracy vs. Epoch
Accuracy
Epoch
initial starter code setup
kernel initialization
data augmentation
psudo label