Programming Assignment 1

Neural Network design

SE395: Introduction to Deep Learning

Objective

- (Main) 3-layer Neural Network for Classification without the deep learning framework (only python)

- (Extra credit) 3-layer Neural Network for Classification using a deep learning framework (e.g. pytorch, tensorflow)

Overall steps

1. Prepare the training and test datasets (MNIST)
2. Design a 3-layer Neural Network
3. Design the training process and train the network
4. Test the network on the test data and visualization the results on the report
5. Extra credit (30% of total) – 3-layer NN using deep learning framework

Refer:
https://github.com/suqi/deeplearning_andrewng/blob/master/Course1-DL-basic/Week%204/Deep%20Neural%20Network%20Application:%20Image%20Classification/Deep%20Neural%20Network%20-%20Application%20v3.ipynb

1. Prepare training/test dataset

1. **Download MNIST datasets**
   1. Download link: http://yann.lecun.com/exdb/mnist/
      1. train-images-idx3-ubyte.gz: training set images (9912422 bytes)
         train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
         t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
         t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)
   2. Data load: https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/g3doc/tutorials/mnist/download/

2. **Prepare the datasets for training**
   1. Ex) normalization

2. Design a 5-leyer Neural Network

1. **Design sub-layers and back-propagation**
   1. Linear layer
   2. ReLU & LeakyReLU (LReLU)
      1. LReLU: Freely choose the hyperparameter
   3. SoftMax
   4. Cross-entropy loss
   5. SGD

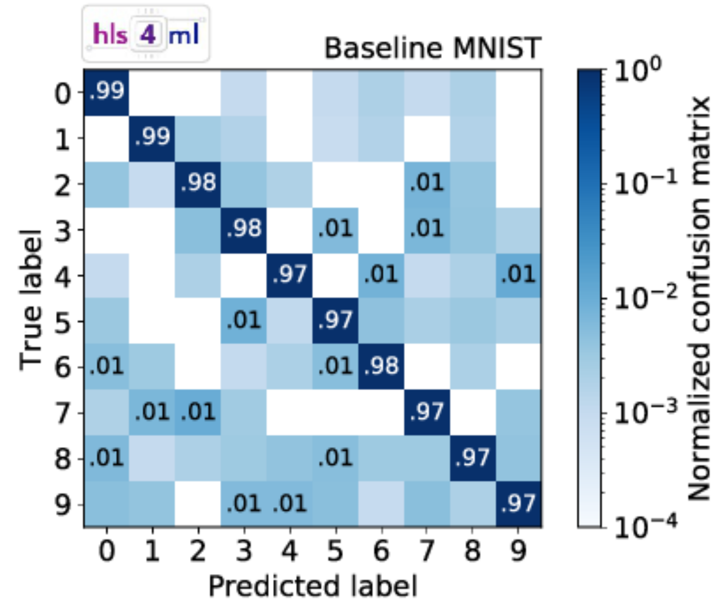
2. **Design two 3-layer Neural Network**
   1. Sequence: Input - Linear-ReLU - Linear-ReLU - Linear-SoftMax
      (The input and output size of NN: input 28x28, output 10)
   2. Sequence: Input - Linear-LReLU - Linear-LReLU - Linear-SoftMax
      (The input and output size of NN: input 28x28, output 10)

3. Design the training process and train the network

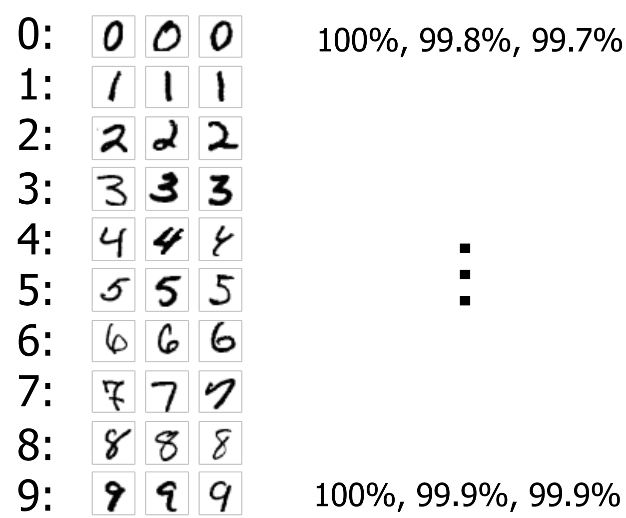
1. Initialize the model parameters
2. Implement and do forward propagation
3. Implement and compute the cross-entropy loss
4. Implement and do backward propagation
5. Implement and update model parameter using gradient descent (SGD)
6. Draw the plot of the loss

4. Test the network on the test data and visualization the results on the report

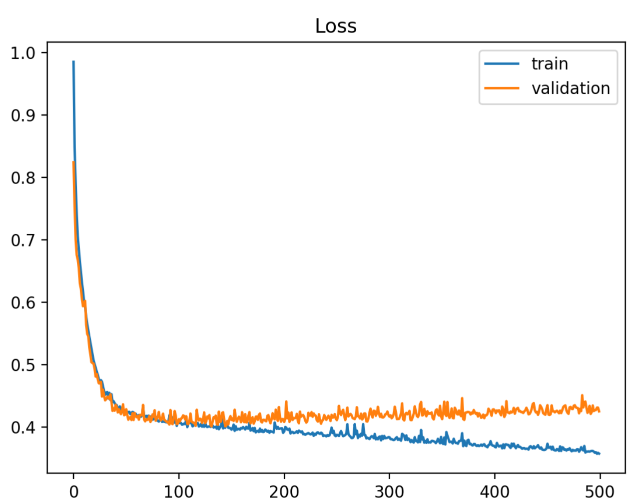
1. **Show 10x10 confusion matrix**
   1. the probability of classification results for all classes
2. **Show top 3 scored images with probability (for each class)**
3. **Show training Loss graph**
   (Train & Validation)



1. 10x10 confusion matrix



2. Top-3 images with probability



3. Loss graph

5. Extra credit (30% of total) 3-layer NN with Pytorch

1. **Design a same neural network using a deep learning framework**
    1. You can use any frameworks (Pytorch, Tensorflow, Keras, etc…)

LINEAR

CLASS `torch.nn.Linear(in_features: int, out_features: int, bias: bool = True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shape:

- Input: $(N, *, H_{in})$ where $*$ means any number of additional dimensions and $H_{in} = \text{in\_features}$
- Output: $(N, *, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} = \text{out\_features}$.

Variables

- **~Linear.weight** – the learnable weights of the module of shape $(\text{out\_features}, \text{in\_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in\_features}}$
- **~Linear.bias** – the learnable bias of the module of shape (out_features). If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in\_features}}$

CROSSENTROPYLOSS

CLASS `torch.nn.CrossEntropyLoss(weight: Optional[torch.Tensor] = None, size_average=None, ignore_index: int = -100, reduce=None, reduction: str = 'mean')` [SOURCE]

This criterion combines `nn.LogSoftmax()` and `nn.NLLLoss()` in one single class.

It is useful when training a classification problem with *C* classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain raw, unnormalized scores for each class.

input has to be a Tensor of size either $(\text{minibatch}, C)$ or $(\text{minibatch}, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the *K*-dimensional case (described later).

This criterion expects a class index in the range $[0, C-1]$ as the *target* for each value of a 1D tensor of size *minibatch*; if *ignore_index* is specified, this criterion also accepts this class index (this index may not necessarily be in the class range).

The loss can be described as:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

or in the case of the `weight` argument being specified:

$$\text{loss}(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$$

The losses are averaged across observations for each minibatch.

Can also be used for higher dimension inputs, such as 2D images, by providing an input of size $(\text{minibatch}, C, d_1, d_2, ..., d_K)$ with $K \geq 1$, where K is the number of dimensions, and a target of appropriate shape (see below).

Submission

Due : Oct 16, 11:59PM
To : lms.dgist.ac.kr

1. **Submission should include (1) Source code, (2) Report**

2. **Report should include the results and results comparison**
   1. For all networks (3-layer NN with ReLU, 3-layer NN with LReLU), show the results and compare (ReLU vs LReLU)
      (a) 10x10 Confusion Matrix,
      (b) Top 3 score images (all classess),
      (c) Training Loss graph
   2. (Optional – extra 30%) Do step 1 using deep learning framework & show the results (a)-(c)

Notice

Due : Oct 16, 11:59PM
To : lms.dgist.ac.kr

1. **Delayed submission**
   1. 25% score will be degraded every 1-day delay & after 3 days delayed, you will get 10% of total score
      (e.g., 100% → 75% (1day) → 50% (2day) → 25% (3day) → 10% (> 3day)

2. **Plagiarism**
   1. No grade for copied codes (from friends and internet)
   2. You can refer source from internet, but do not copy and paste.

3. **Partial credit**
   1. Even though you are not successfully design the network and obtain reasonable result, please send your code.
   2. There will be partial credit for each module implementation.