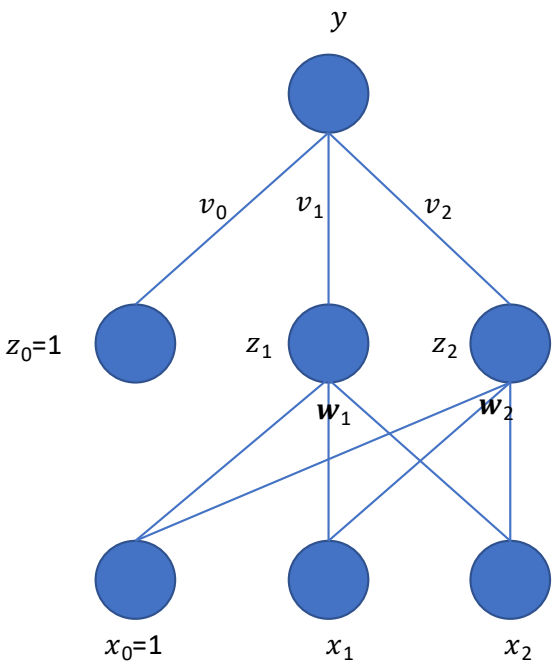
CSCI 5521: Introduction to Machine Learning (Spring 2020)[1]

Homework 4

1. (**30 points**) Consider the following Multilayer Perceptron (MLP) for binary classification,



we have the following error function:

$$E(\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{v}|\boldsymbol{X}) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t),$$

where $y^t = \text{sigmoid}(v_2 z_2 + v_1 z_1 + v_0)$, $z_1^t = \text{ReLU}(w_{1,2} x_2^t + w_{1,1} x_1^t + w_{1,0})$ and $z_2^t = \tanh(w_{2,2} x_2^t + w_{2,1} x_1^t + w_{2,0})$, the rectified linear unit $\text{ReLU}(x)$ is defined as follows

[1]Instructor: Rui Kuang (kuang@cs.umn.edu). TAs: Tianci Song (song0309@umn.edu) and Ruyuan (wanxx199@umn.edu).

$$\text{ReLU}(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{otherwise} \end{cases}$$

(a) Derive the equations for updating $\{\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{v}\}$ of the above MLP.

(b) Now, consider shared weights $\boldsymbol{w} = \boldsymbol{w_1} = \boldsymbol{w_2}$. Derive the equations for updating $\{\boldsymbol{w}, \boldsymbol{v}\}$, i.e. to minimize

$$E(\boldsymbol{w}, \boldsymbol{v}|\boldsymbol{X}) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t),$$

where $y^t = \text{sigmoid}(v_2 z_2 + v_1 z_1 + v_0)$, $z_1^t = \text{ReLU}(w_2 x_2^t + w_1 x_1^t + w_0)$ and $z_2^t = \tanh(w_2 x_2^t + w_1 x_1^t + w_0)$.

Hint: Read Section 11.7.2 to see how Equations 11.23 and 11.24 are derived from Equation 11.22

Hint 2: $\tanh'(x) = 1 - \tanh^2(x)$.

Hint 3: $\text{ReLU}'(x) = \begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{otherwise} \end{cases}$.

2. (**40 points**) Implement a Multilayer Perceptron (MLP) **with stochastic gradient descent** to classify the optical-digit data. Train your MLPs on the "optdigits_train.txt" data, tune the number of hidden units using the "optdigits_valid.txt" data, and test the prediction performance using the "optdigits_test.txt" data. (Read the submission instruction carefully to prepare your submission files.)

(a) Implement a MLP with 1 hidden layer **using the LReLU (Leaky ReLU) activation function**:

$$\text{LReLU}(x) = \begin{cases} 0.01x, & \text{for } x < 0 \\ x, & \text{otherwise} \end{cases}$$

Use the MLP for classifying the 10 digits. Read the algorithm in Figure 11.11 and section 11.7.3 in the textbook. When using the LReLU activation function, the online version of Equation 11.29 becomes:

$$\Delta w_{hj} = \begin{cases} 0.01\eta \Big[\sum_i (r_i - y_i) v_{ih} \Big] x_j & \text{for } \boldsymbol{w_h}^T x < 0 \\ \eta \Big[\sum_i (r_i - y_i) v_{ih} \Big] x_j & \text{otherwise} \end{cases}$$

Try MLPs with $\{3, 6, 9, 12, 15, 18\}$ hidden units. Report and plot the training and validation error rates by the number of hidden units. How many hidden units should you use? Report the error rate on the test set using this number of hidden units.
Hint: When choosing the best stepsize η (between 0 and 1 such as 10^{-5}), you might need to start with some value and, after a certain number of iterations, decrease your η to improve the convergence. Alternatively, you can implement Momentum or Adaptive Learning Rate (section 11.8.1 in the textbook).

(b) Train your MLP with the best number of hidden units obtained. Combine the training set and the validation set as one (training+validation) dataset to run the trained MLP from problem 2(a) with the data. Apply PCA to the values obtained from the hidden units (you can use *PCA()* function, which is included in the module *decomposition* in *scikit-learn* package). Using the projection to the first 2 principal components, make a plot of the training+validation dataset (similar to Figure 11.18 in the textbook). Use different colors for different digits and label each sample with its corresponding digits (the same as you did in HW3). Repeat the same projecting the datasets to the first 3 principal components and do the visualization using 3-D plot. (Hint: you can use the function *scatter()* to visualize the 3-D data). Compare the 2-D and 3-D plots and explain the results in the report.
Note: Change the x-axis and y-axis to log scale in order to better visualize the datapoints.

3. (**30 points**) There are many popular Deep Learning Framework for designing and implementing deep neural networks, for example, Tensorflow, Theano, Mxnet, Pytorch and Keras. In this homework question you will learn how to create simple convolutional neural networks (CNNs) for optdigits classification by using Keras. (Keras is slightly different from other popular deep learning frameworks since it is basically a higher level neural networks API, written in Python and able to run on the top of other deep learning frameworks, such as Tensorflow and Theano. And in this homework, you will be asked to use Tensorflow as backend)

   (a) Read the Keras documentation[2] to get familiar with how to

      i. Load and explore image data.

[2] `https://keras.io/`

ii. Define the network architecture.

iii. Specify training/validation options.

iv. Train the network.

v. Predict the labels of testing data and calculate the classification accuracy.

Read another Keras documentation[3] to define your own activation function.

(b) Create the `LReLU.py`, and implement leaky ReLU as defined in Question 2 by using the functions in backend module 3. **Hint: when implementing activation function via abstract Keras backend API, you don't have to calculate the derivative of customized activation function for backpropagation since Keras uses automatic differentiation instead. As we use Tensorflow as backend, you may consider to implement leaky ReLU with keras.backend.tf.where() function.**

(c) Modify the **Define Network Architecture** section in the `main.py` file to test the following two CNN structures.

i. Input layer → 2D convolution layer (1 filter with size 4) → Batch normalization layer → LReLU layer (use your own customized myLReLULayer class) → Fully connected layer → Softmax layer → Classification layer

ii. Input layer → 2D convolution layer (20 filter with size 3) → Batch normalization layer → LReLU layer (use your own customized myLReLULayer class) → Pooling layer (use max pooling with poolsize 3 and stride size 2) → 2D convolution layer (32 filter with size 3) → Batch normalization layer → LReLU layer (use your own customized myLReLULayer class) → Fully connected layer → Softmax layer → Classification layer

For both network structures, take a screen shot of the Training Process generated by Tensorboard (please check out the official document of Tensorboard for more details [4]), and report the accuracies on the testing data.

[3] `https://keras.io/backend/`

[4] `https://www.tensorflow.org/tensorboard/get_started`

Instructions

- Solutions to all questions must be presented in a report which includes result explanations, and all images and plots.
- All programming questions must be written in Python, no other programming languages will be accepted. The code must be able to be executed from either command line or PyCharm window on the cselabs machines. Each function must take the inputs in the order specified and print/display the required output to either terminal or PyCharm console. For each part, you can submit additional files/functions (as needed) which will be used by the main functions specified below. Put comments in your code so that one can follow the key parts and steps. **Please follow the rules strictly. If we cannot run your code, you will receive no credit.**
- **Question 2:**
  - Train a MLP: MLPTrain(*train_data.txt*: path to training data, *val_data.txt*: path to validation data K: number of output units, H: number of hidden units). The function must return in variables the outputs (Z: a $N \times H$ matrix of hidden unit values, W: a $(D+1) \times H$ matrix of input unit weights, and V: a $(H+1) \times K$ matrix of hidden unit weights). The function must also print the training and validation error rates for the given function parameters.
  - Test a MLP: MLPtest(*test_data.txt*: path to test data file, W: a $(D+1) \times H$ matrix of input unit weights, V: a $(H+1) \times K$ matrix of hidden unit weights). The function must return in variables the outputs (Z: a $N \times H$ matrix of hidden unit values), where N is the number of training samples. The function must also print the test set error rate for the given function parameters.
  - MLPtrain will implement an MLP with D inputs and one input bias unit, H hidden units and one hidden bias unit, and K outputs.
  - `problem2a.py` and `problem2b.py`: scripts to solve the problems 2 (a) and (b), respectively, calling the appropriate functions.
- You may need the following functions in your homework: *hstack*, *vstack*, *tile*, *argmax*, *argmin*.
- For the optdigits data, the first 64 columns are the data and the last column is the label.

Submission

- **Things to submit:**
  1. hw4_sol.pdf: A PDF document which contains the report with solutions to all questions.
  2. MLPtrain.py: The Python code of the *MLPtrain* function.
  3. MLPtest.py: The Python code of the *MLPtest* function.
  4. problem2a.py: Code to solve problem 2 (a).
  5. problem2b.py: Code to solve problem 2 (b).
  6. LReLU.py: Your own customized leaky ReLU function in problem 3(b).
  7. main.py: The modified script for the neural structure in problem 3(c)(ii).
  8. Any other files, except the data, which are necessary for your code.
- **Submit**: hw4_sol.pdf and a zipfile of all other files must be submitted electronically via Canvas.