EE 559 **Homework 7** Posted: Sat., 4/1/2023
Chugg/Jenkins Due: Sat, 4/8/2023, 4:00 PM

1. **Hyper-parameter Optimization for MLP on FMNIST**

In this problem you will train an MLP with one hidden layer for the Fashion MNIST dataset. Use ReLU for the hidden-layer activation function. You will split the training dataset using an 80/20 split for a train/val split. This yields: 48,000 train images, 12,000 val images, and 10,000 test images. You will repeatedly train the MLP to optimize the following parameters:

- M: number of hidden nodes/units (the bias is not counted here – use convention from backprop handout which is same as that in tensorflow/pytorch)
- η: learning rate. Use the standard SGD optimizer, with shuffling and mini-batches.
- B: the mini-batch size.
- λ: the L2 regularizer coefficient

You can use whatever training platform you like – e.g., tensorflow.keras, PyTorch, sklearn, or your own code. You have reference code for this exact training task using tensorflow.keras and PyTorch (GitHub's Lecture directory) and the TA's showed example of MLP training in sklearn in discussion.

(a) Start with $M = 48$ hidden nodes, $\eta = 0.01$, $\lambda = 10^{-3}$ and $B = 32$. These were used in the tf.keras/PyTorch examples shown by Prof. Chugg and available in the GitHub repository in the Lecture directory. Vary the batch size and time how long it takes to reach an accuracy of 80%. Run 5 times and average the run times. Training runs will vary because the weights are randomly initialized. Select the batch size that has smallest run-time. Report this batch size. Also, for this batch size, report the sample mean and sample standard deviation of the 5 run-times. Use `np.mean()` and `np.std()` for this computation. For the remainder of this problem, use this batch size.

(b) Perform a grid search over the following hyper-parameters:

- $\eta \in \{0.001, 0.01, 0.1\}$
- $\lambda \in \{1e-4, 1e-3, 1e-2\}$
- Number of hidden nodes $M \in \{40, 80, 160\}$

Train to 30 epochs for each of the 27 combinations. For each value of hidden nodes, produce learning curves (include train/val and do for both loss/accuracy plots). Use the best val performance to select the value of these 3 hyper-parameters.

(c) For the best hyper-parameters found in part (b), run 5 training runs out to 100 epochs. Report the best accuracy (over epochs) on val for each run - this is 5 numbers. Compute, mean, max, and std deviation for these 5 values.

(d) Take best model from part c (highest val accuracy) and evaluate on test. Report the test accuracy. Report the number of trainable parameters and all hyper-parameters used to obtain this final best model.

Notes:

- Since you will be performing many training runs, you should use for loops.

- Learn how to save and load model files so that you can retrieve the best model. This is done in the PyTorch code. We will provide example code to save the best model using tf.keras. See https://github.com/keithchugg/ee559_spring2023/tree/main/hw_helpers
- The sample PyTorch code used standard data loaders, which is good practice in general. However, for a small dataset such as FMNIST, the entire dataset can be loaded into RAM. This is what the tf.keras code is doing. We will provide you with a custom dataloader for PyTorch that will load the full dataset and speed up the training. (Same link as above)

2. RBF network for 2D function approximation based on data

You will use an RBF network to learn a regression function based on the provided synthetic dataset. You may use any sklearn, numpy, and scipy functions.

The data has 2 features, and feature space extends over
$0 \le x_1 \le 2,\ 0 \le x_2 \le 2.$

Use Gaussian RBF basis functions:

$$\underline{\phi}_m(\underline{x}) = \exp\left\{-\gamma \left\|\underline{x} - \underline{\mu}_m\right\|^2\right\}, \quad \gamma > 0, \quad m = 1,2,\cdots,M$$

and use as our main error measure the root-mean-squared error:

$$\text{RMSE} = (\text{MSE})^{1/2}$$

so that it can be numerically compared with the predicted and given outputs.

Tips:

(1) You will use 3 different techniques (described below in (c), (d), and (e)) for choosing the basis function centers $\underline{\mu}_m$.

(2) You might find the function `sklearn.metrics.pairwise.rbf_kernel` useful, because you will need to compute the RBF of all possible pairs between a data point $\underline{x}_i$ and a basis function center $\underline{\mu}_m$.

(3) Implement the first layer as a nonlinear transformation (given above).

(4) You can implement the second layer as a standard linear regression problem, for example by using `sklearn.linear_model.LinearRegression`.

The dataset:

(1) is in `HW7_Pr2_datasetA.zip` and is divided into training and testing sets, having $N_{Tr} = 4000$, $N_{Test} = 2000$, and number of features D=2;

(2) can be loaded by:

```
X_train = np.load('datasetA_X_train.npy')
X_test = np.load('datasetA_X_test.npy')
y_train = np.load('datasetA_y_train.npy')
y_test = np.load('datasetA_y_test.npy')
```

(a) Calculate an approximate default choice γ_d for γ. This can be done as follows:

(i) Calculate the average spacing, α, between basis-function centers as $\left(\frac{\Delta_{x1}\Delta_{x2}}{M}\right)^{\frac{1}{2}}$, in which

Δ_{x1} = extent (width) of feature space in the x_1 dimension

Δ_{x2} = extent (width) of feature space in the x_2 dimension

Comment: this approximation assumes the data, and the RBF centers $\underline{\mu}_m$, are approximately uniformly distributed over the feature space.

Tip: For $D > 2$, the expression for α would be $\left(\frac{\Delta_{x1}\Delta_{x2}\cdots\Delta_{xD}}{M}\right)^{\frac{1}{D}}$, in which D = number of features.

(ii) For γ_d, choose the half-width of the RBF to cover a few average spacings, thus let $\sigma = 5\alpha$. You can relate σ to γ by comparing:

$$\underline{\phi}_m(\underline{x}) = \exp\left\{-\gamma\left\|\underline{x}-\underline{\mu}_m\right\|^2\right\} = \exp\left\{-\frac{\left\|\underline{x}-\underline{\mu}_m\right\|^2}{2\sigma^2}\right\}.$$

Give an expression for γ_d in terms of M, in simplest form.

Tips:

(1) Include γ_d in your range of γ during model selection.

(2) For the L_2-norm calculation, you may find `sklearn.metrics.pairwise.euclidean_distances` useful.

(b) For comparison to the below systems, compute the RMSE of a trivial system that always outputs the sample mean value $\bar{y}$ on the training-set data.

In each of parts (c)-(e) below, you will try a different method for choosing RBF centers; each part can be done independently of the other two. For each part, do the following (i)-(vi).

(i) Use MSE linear regression for the second layer, without regularization.

(ii) Use model selection for finding a good value for γ and any other hyperparameter that is being optimzed (such as M in part (d), or K in part (e)). Use 4-fold cross validation based on the given training set.

Tip: for γ, choose values that range from $0.01\gamma_d$ to $1000\gamma_d$, on a log scale (e.g., values $\gamma = 0.01, 0.1, 1, 10, 100, 1000$).

(iii) Report on the cross validation RMSE for each value (c) or pair of values ((d) or (e)) tried, in 2 tables: one table for RMSE (mean over the 4 folds) and one table for RMSE (standard deviation over the 4 folds). Report the best mean value (or pair of values) found.

(iv) Plot training and validation RMSE vs. γ. (For parts (d) and (e), use your best value of M = M ∗ or K = K ∗ for the plot.) (1 plot for each of (c), (d), (e).)

(v) For parts (d) and (e), answer the following. If computational complexity were an issue, what is the smallest value of M or K (and its associated γ) that would give RMSE at least a factor of 10 lower than the trivial system of (b)? Call this model your reduced-computational-complexity (RCC) model. What factor reduction in number of hidden units (dimensionality of the expanded feature space) from the original M=3000 in part (c) does this RCC model represent?

(vi) For parts (d) and (e), plot in original 2D feature space, the training data points $\underline{x}_i$ and the cluster centers $\underline{\mu}_m$ for your best values of hyperparameters. Then repeat the plots for your RCC model, and again for your lowest-complexity-model (M=30 or K=30). (Total of 3 plots for (d), and 3 plots for (e).)

Tip: use small enough dots for your data-point symbols so that it is easy to visualize the data points.

(vii) Also for parts (d) and (e), plot the validation error and its standard deviation vs. the second hyperparameter (M for (d), K for (e)), using the best γ for each value of M or K. (The value of best γ may depend on M or K.)

Tip: for cross validation, you may find `sklearn.model_selection.KFold` useful to directly generate pairs of train-validation sets. See document page for more information.

(c) Choose the basis function centers as the data points: $\underline{\mu}_m = \underline{x}_m$, $m = 1,2,\cdots,N$, in which N is the number of training data points during each fold in cross validation. For this part, the only hyperparameter to choose during model selection is γ.

(d) Randomly choose the basis function centers, without replacement, from the training-set data. Use number of basis function centers M varying from 30 to 300 (e.g., values 30, 60, 100, 300, 600).

Tip: one possible way is to firstly get the number of centers M and then use functions like `sklearn.utils.shuffle` or `np.random.choice` (no replacement) etc. to randomly choose the basis-function centers from the training data. If you use `np.random.choice`, let `replace = False` in the arguments.

In this part you have 2 hyperparameters to find during model selection (γ and M).

(e) Use K-means clustering to choose basis function centers for a given K; vary K using model selection (e.g., use values 30, 60, 100, 300, 600). For each value of K, choose your initial cluster centers randomly (i.e., in sklearn's K-means).

Tip: You can use the sklearn K-means clustering function `sklearn.cluster.KMeans` (set init = 'random'), and use `.cluster_centers_` to get the centers (c.f. Discussion 10).

In this part you have 2 hyperparameters to find during model selection (γ and K).

(f) Give the d.o.f. and number of constraints for the second layer (linear regressor) for each of (c), (d), and (e), for your best model of each; and again for your RCC model for each of (d), (e).

(g) Run the best model from each of (c), (d), and (e); and run the RCC model of (d), (e), on your test set. Report the RMSE of each (5 models total).

(h) Compare and comment on your results from (b)-(g). Specifically, observe and try to explain differences in performance for different values of *M* (or *K*) and γ during model selection.

(i) The dataset was drawn from the following target function:

$$y(x) = 10\cos\left(\frac{\pi}{2}x_1\right)\sin\left(\frac{5\pi}{x_1^2+1}\right)\sin(\pi x_2)$$

To visuialize the target function, plot $y(x_1, x_2)$ vs. x_1 for $x_2 = 0.5$ and for $x_2 = 1.5$. Also plot $y(x_1, x_2)$ vs. x_2 for $x_1 = 0.3$. (3 plots total)

To compare the prediction with the target, plot $\hat{f}(x_1, x_2)$ and $y(x_1, x_2)$ vs. x_1 for $x_2 = 0.5$, for the following cases: your best result from each of (c), (d), (e); and also your RCC model from each of (d), (e). (5 plots total for comparing prediction with target)