CS689: Machine Learning - Fall 2019

Homework 3: Solutions

Questions:

1. (*20 points*) **Lagrangian Dual for Absolute Loss Regression:** Consider the problem of finding optimal linear regression model weights in the RRM framework using the absolute loss and a squared 2-norm regularizer:

$$\theta_* = \arg\min_{\theta} C \sum_{n=1}^{N} |y_n - f(\mathbf{x}_n, \theta)| + \|\mathbf{w}\|_2^2$$

where $\theta = [\mathbf{w}, b]$ and $f(\mathbf{x}, \theta) = \mathbf{w}\mathbf{x}^T + b$. This problem is convex, but not differentiable. However, the problem can be converted into an alternative linearly constrained form where the objective function is quadratic in an expanded set of variables $[\theta, \epsilon]$ where $\epsilon = [\epsilon_1^+, \epsilon_1^-, ..., \epsilon_N^+, \epsilon_N^-]$:

$$\begin{aligned}\theta_*, \epsilon_* = \arg\min_{\theta,\epsilon} C &\sum_{n=1}^{N} (\epsilon_n^+ + \epsilon_n^-) + \|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad &\forall n \;\; y_n \leq f(\mathbf{x}_n, \theta) + \epsilon_n^+ \\ &\forall n \;\; y_n \geq f(\mathbf{x}_n, \theta) - \epsilon_n^- \\ &\forall n \;\; \epsilon_n^+ \geq 0 \\ &\forall n \;\; \epsilon_n^- \geq 0\end{aligned}$$

a. (*5 pts*) Derive the Lagrangian function for the constrained formulation shown above.

Example Solutions:

$$\begin{aligned}L = C &\sum_{n=1}^{N} (\epsilon_n^+ + \epsilon_n^-) + \|\mathbf{w}\|_2^2 + \sum_{n=1}^{N} a_n (y_n - f(\mathbf{x}_n, \theta) - \epsilon_n^+) - \sum_{n=1}^{N} \widehat{a}_n (y_n - f(\mathbf{x}_n, \theta) + \epsilon_n^-) \\ &- \sum_{n=1}^{N} \mu_n \epsilon_n^+ - \sum_{n=1}^{N} \widehat{\mu}_n \epsilon_n^- \\ = \|\mathbf{w}\|_2^2 &+ \sum_{n=1}^{N} (a_n - \widehat{a}_n)(y_n - \mathbf{w}\mathbf{x}_n^\top - b) + \sum_{n=1}^{N} (C - a_n - \mu_n)\epsilon_n^+ + \sum_{n=1}^{N} (C - \widehat{a}_n - \widehat{\mu}_n)\epsilon_n^-\end{aligned}$$

b. (*10 pts*) Derive the Lagrangian dual for the constrained formulation shown above.

Example Solutions: We first set the derivatives of L with respect to each of the primal variables to zero and solve for the constrains for the dual problem:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}} &= 0 \Rightarrow \mathbf{w} = \frac{1}{2}\sum_{n=1}^{N}(a_n - \widehat{a}_n)\mathbf{x}_n, \\
\frac{\partial L}{\partial b} &= 0 \Rightarrow \sum_{n=1}^{N} a_n - \widehat{a}_n = 0, \\
\frac{\partial L}{\partial \epsilon_n^+} &= 0 \Rightarrow C = a_n + \mu_n, \quad \text{for all } n = 1, \dots, N, \\
\frac{\partial L}{\partial \epsilon_n^-} &= 0 \Rightarrow C = \widehat{a}_n + \widehat{\mu}_n, \quad \text{for all } n = 1, \dots, N.
\end{aligned}$$

With the above constraints, we can simplify L as follows:

$$\begin{aligned}
L &= \|\mathbf{w}\|_2^2 + \sum_{n=1}^{N}(a_n - \widehat{a}_n)(y_n - \mathbf{w}\mathbf{x}_n^\top - b) + \sum_{n=1}^{N}(C - a_n - \mu_n)\epsilon_n^+ + \sum_{n=1}^{N}(C - \widehat{a}_n - \widehat{\mu}_n)\epsilon_n^- \\
&= \frac{1}{4}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)\mathbf{x}_n\mathbf{x}_m^\top + \sum_{n=1}^{N}(a_n - \widehat{a}_n)y_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)\mathbf{x}_n\mathbf{x}_m^\top \\
&= \sum_{n=1}^{N}(a_n - \widehat{a}_n)y_n - \frac{1}{4}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)\mathbf{x}_n\mathbf{x}_m^\top.
\end{aligned}$$

The Lagrange dual problem is therefore

$$\begin{aligned}
\text{minimize} \quad & \sum_{n=1}^{N}(a_n - \widehat{a}_n)y_n - \frac{1}{4}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)\mathbf{x}_n\mathbf{x}_m^\top \\
\text{subject to} \quad & 0 \le a_n \le C \quad \text{for all } n = 1, \dots, N \\
& 0 \le \widehat{a}_n \le C \quad \text{for all } n = 1, \dots, N \\
& \sum_{n=1}^{N} a_n - \widehat{a}_n = 0.
\end{aligned}$$

c. (*5 pts*) Explain the advantages of solving the Lagrangian dual problem instead of the constrained version of the primal problem.

Example Solutions:

- The complexity of solving the dual quadratic problem scales with N while the complexity of solving the primal quadratic problem scales with D, the dimensionality of the data. When $N \ll D$, solving the dual can be more efficient.
- The dual problem allows the use of kernel trick, which replaces $\mathbf{x}_n\mathbf{x}_m^\top$ by $\mathcal{K}(\mathbf{x}_n, \mathbf{x}_m)$ with any (reproducing) kernel $\mathcal{K}(\cdot, \cdot)$.
- After the model is learned, making predictions with the dual can be faster than with the primal if $N \ll D$.

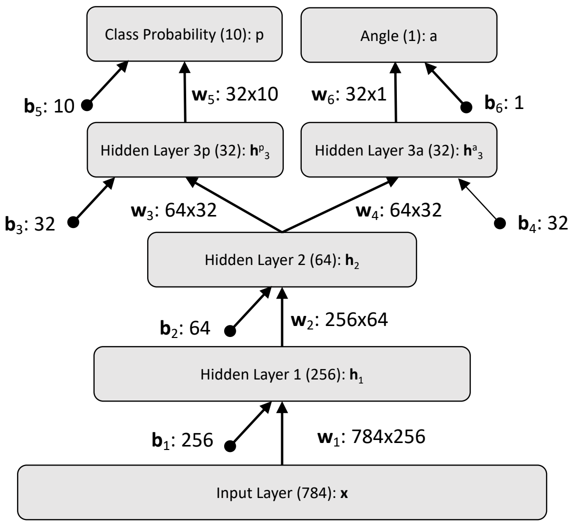
2. (*60 points*) **Multi-Output Neural Networks:** In this question, you will implement a custom neural network model that simultaneously solves a regression problem (detecting the angle of an object in an image), and a multi-class classification problem (determining the class of the object present in an image) using a composite loss. The prediction model $f(\mathbf{x}, \theta)$ will simultaneously produce a vector of class probabilities $\mathbf{p} = f^c(\mathbf{x}, \theta)$ where $\mathbf{p}_y$ gives the probability for class y, and an angle output $f^a(\mathbf{x}, \theta) \in \mathbb{R}$. A data set for this problem is a set of triples $\mathcal{D} = \{(\mathbf{x}_n, y_n, a_n) | 1 \leq n \leq N\}$ where $\mathbf{x}_n$ is the feature vector, $y_n \in \{0, ..., C-1\}$ is the class label, and a_n is the angle in degrees. The objective function for this learning problem is shown below where α is a parameter that trades off between the classification loss (cross entropy, L_{ce}) and a loss defined on angles ($L_{\cos}$).[1]

$$\theta_* = \arg\min_{\theta} \sum_{n=1}^{N} \alpha L_{ce}(y_n^c, f^c(\mathbf{x}_n, \theta)) + (1-\alpha) L_{cos}(a_n, f^a(\mathbf{x}_n, \theta))$$

$$L_{ce}(y, \mathbf{p}) = -\sum_{i=0}^{C} [y = i] \log p_i$$

$$L_{\cos}(a, a') = 0.5(1 - \cos(0.01745 \cdot (a - a')))$$

In this question, you will begin by implementing an architecture for this problem consisting of an input layer, three hidden layers, and two parallel output layers (one each for localization and classification). The inputs are 28x28 images, resulting in 784-long input vectors. All hidden layers will be use RELU non-linearities. The class probability output layer will use a softmax non-linearity. The angle output layer will consist of one linear unit. The diagram of the network architecture you will implement is shown below. All layers joined by arrows in the figure are fully connected.



a. (*40 pts*) Starting from the provided template (`nn.py`), implement a Scikit-Learn compatible class for the model shown above. You can develop your implementation using NumPy and Torch (version 1.2). You are strongly encouraged to use automatic differentiation methods to develop your implementation. In the

[1] This form of the loss assumes that the cos function expects inputs in radians, while the data set specifies that angles are provided in degrees. 0.01745 is then the approximate conversion factor from degrees to radians.

implementation of `fit` for this question, use the Adam optimizer included with Torch. Use a learning rate of $1e-4$ and the Adam optimizer's built-in weight decay with a regularization constant of $1e-4$. Your fit function should use mini-batches of 64 examples. To ensure that learning is numerically stable, it is recommended that you do not explicitly compute the softmax function during learning and prediction to avoid issues with underflow/overflow. See `nn.functional.cross_entropy` for methods to help with this.

b. (*5 pts*) Using the provided training data set, learn the model using $\alpha = 0.5$ for 20 epochs.[2] Report the classification error rate and the mean absolute error (MAE) of the angle predictions obtained on both the training and validation sets.[3] Training should take less than 5 minutes if properly implemented.

Example Solutions: We ran the reference model 10 times to establish a range of error values that that a correct implementation should exhibit. The upper and lower ranges are shown in the table below.

dataset	classification error	mean absolute error
training	$[0.00, 0.01]$	$[7.8, 8.3]$
validation	$[0.05, 0.07]$	$[8.7, 9.3]$

Table 1: Error ranges on training and validation data.

c. (*15 pts*) Use your implementation to perform experiments to assess how changing the value of α affects the classification and angle prediction results on the validation set. In particular, vary the value of α from 0 to 1 in steps of 0.1 and plot the classification error rate and the mean absolute error of the angle prediction task on the validation data. Use two plots, one for each performance measure. Note that setting $\alpha = 0$ is equivalent to solving the angle prediction task on its own, while setting $\alpha = 1$ is equivalent to solving the classification problem on its own. Is there a setting of α where jointly learning these two tasks using this model provides a benefit compared to learning the angle prediction task on its own?

Example Solutions:

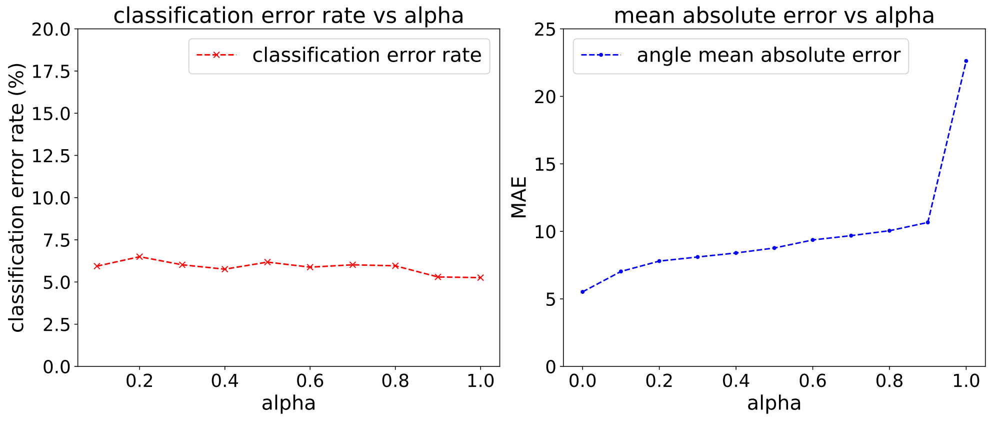


Figure 1: classification and angle prediction on validation data with different α

[2]One epoch of training consists of one full sweep through the data set such that every data case is seen one time (the last batch may have size less than 64.

[3]Predict the class with the highest probability in all questions

No setting of α from 0 to 1 such that the jointly learning performs better than only angle prediction task ($\alpha = 0$). From Figure 1, we can see that the MAE of angles increase as α increases, while the classification errors are not strongly influenced by α with the range and stepping we are using. This trend can help us selecting the optimal α.

3. (*20 points*) **More Neural Networks:** In this problem, your task is to attempt to improve on the predictive performance of the model presented in Question 2. You must still use a single multi-output neural network model, but you are free to experiment with any other architecture and algorithm choices. Performance should still be assessed in terms of the classification error rate and the mean absolute error of the angle predictions. Answer the following questions.

a. (*5 pts*) Begin by exploring different models and/or algorithms. Describe at a high level some of the different architectures and/or algorithm choices that you tried.

Example Solutions: Noticing that the dataset is MNIST handwritten digits, we experiment with an architecture with two convolutional layers followed by four fully connected layers. We consider two choices for the number of channels and the kernel size for each convolutional layer, and two choices for the sequence of fully connected layers. For completeness, we show the choices for each part of the architecture in Table 2. We also considered training models with different optimizers including `SGD` with momentum, `Adagrad` and `Adam`. We use $\alpha = 0.1$ as this makes the angle loss much lower while not impacting the classification loss that much. [4]

	Conv$_1$				**Conv$_2$**				Fully Connected
	in	out	kernel	padding	in	out	kernel	padding	number of hidden nodes
setting$_1$	1	20	3×3	2	20	50	3×3	2	800, 200, 64, 32, (10, 1)
setting$_2$	1	32	5×5	2	32	64	5×5	2	1000, 256, 64, 32, (10, 1)

Table 2: network parameter sets

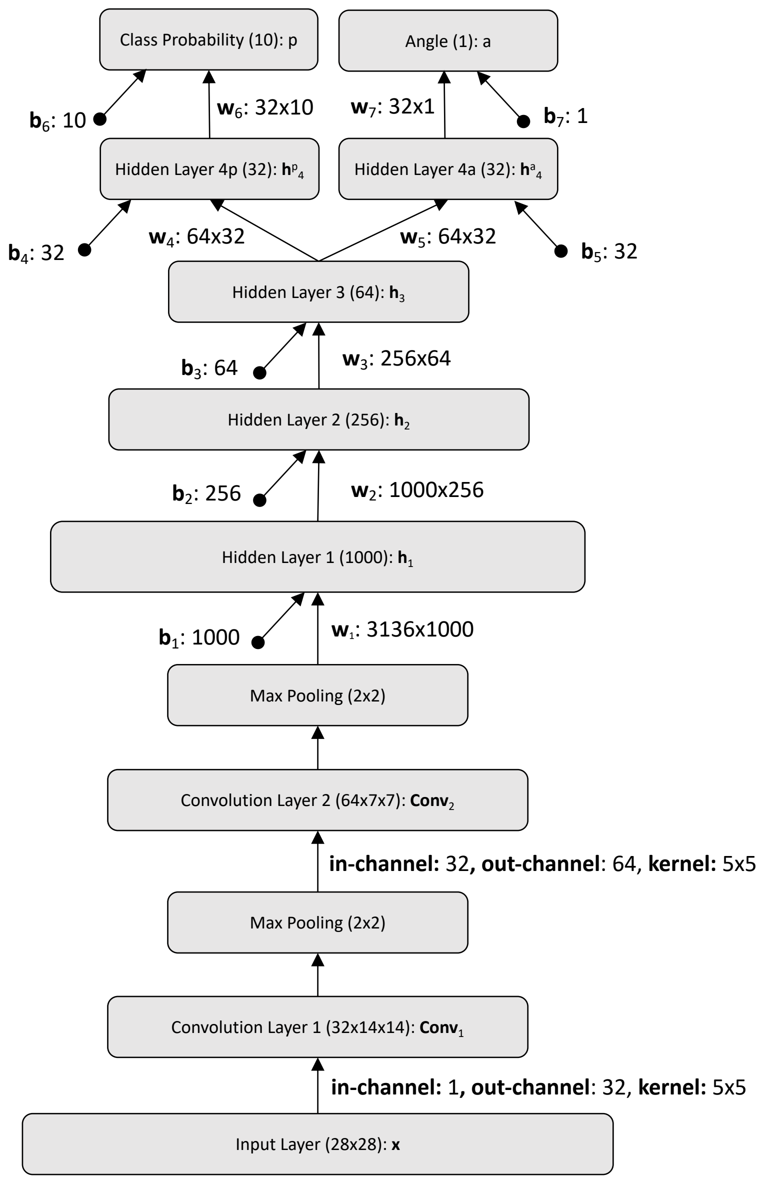
b. (*5 pts*) Next, you will need to select a single best model/algorithm combination among the choices you explored in part (a). As your answer to this question, describe what procedure you used to selected the single best model.

Example Solutions: To select the single best model, we used validation error on the given validation set as the criterion. Specifically, among all models tested, we selected the model with the lowest sum of classification error and MAE.

c. (*5 pts*) For the single best model that you selected, provide a detailed architecture diagram (similar to the one shown above) and describe all details of the learning algorithms used.

Example Solutions: The layers are connected with `ReLU` activation function. And training is using `Adam` as optimizer, initial learning rate = $1e-4$, $\alpha = 0.1$, batch size = 64 and epochs = 20. The selected model architecture is shown in Figure 2.

[4] Note: it is not required to list specific hyper-parameter values for layer sizes, etc. in your solution.



Class Probability (10): p
Angle (1): a
b6: 10
w6: 32x10
w7: 32x1
b7: 1
Hidden Layer 4p (32): hp4
Hidden Layer 4a (32): ha4
b4: 32
w4: 64x32
w5: 64x32
b5: 32
Hidden Layer 3 (64): h3
b3: 64
w3: 256x64
Hidden Layer 2 (256): h2
b2: 256
w2: 1000x256
Hidden Layer 1 (1000): h1
b1: 1000
w1: 3136x1000
Max Pooling (2x2)
Convolution Layer 2 (64x7x7): Conv2
in-channel: 32, out-channel: 64, kernel: 5x5
Max Pooling (2x2)
Convolution Layer 1 (32x14x14): Conv1
in-channel: 1, out-channel: 32, kernel: 5x5
Input Layer (28x28): x

Figure 2: the best network structure

d. (*5 pts*) Using the provided test data, compute predictions for the class labels and angles. Save the array of class predictions to your code folder using `np.save("class_predictions.npy", ...)`. Save the array of angle predictions to your code folder using `np.save("angle_predictions.npy", ...)`. Upload both files to Gradescope along with your code for scoring. Your model should be implemented in `best_nn.py`.

Example Solutions: Comparing `nn.py` with `best_nn.py`, under the same model parameter $\alpha = 0.1$ and learning algorithm, the best model outperformed the default neural network. The class labels and angles are compared in Table 3

	classification error	mean absolute error
`nn.py`	[0.03040, 0.04010]	[5.8700, 6.1810]
`best_nn.py`	[**0.0090**, **0.01220**]	[**4.3060**, **4.8800**]

Table 3: networks performance comparison on test data