CS689: Machine Learning - Fall 2020

Homework 3: Solutions

1. (*20 points*) **Dual for Logistic Regression:** While logistic regression does not have a useful Lagrange dual, an alternative dual can be derived using the fact that $\log \sigma(z) = \min_{\alpha \in [0,1]} \alpha \cdot z - H(\alpha)$ where $\sigma(z)$ is the logistic function and $H(\alpha)$ is the entropy of the Bernoulli distribution $H(\alpha) = -\alpha \log(\alpha) - (1-\alpha)\log(1-\alpha)$. Using this fact, we can express the MAP objective for logistic regression via a saddle point problem as shown below:

$$\min_{\mathbf{w}} \max_{\alpha_{1:N}} \quad \frac{\lambda}{2}\|\mathbf{w}\|_2^2 - \sum_{n=1}^{N} (\alpha_n y_n \mathbf{x}_n \mathbf{w} - H(\alpha_n)) \quad \text{s.t.} \quad \forall n \;\; \alpha_n \in [0,1]$$

a. (*14 pts*)Derive a dual objective for logistic regression that only has the α's as parameters by minimizing $\mathbf{w}$ out of the above saddle point problem. Simplify the resulting dual objective as much as possible. Show your work.

Example Solution: To begin, we note that we can exchange the minimum and the maximum yielding the problem below:

$$\max_{\alpha_{1:N}} \min_{\mathbf{w}} \quad \frac{\lambda}{2}\|\mathbf{w}\|_2^2 - \sum_{n=1}^{N} (\alpha_n y_n \mathbf{x}_n \mathbf{w} - H(\alpha_n)) \quad \text{s.t.} \quad \forall n \;\; \alpha_n \in [0,1]$$

We can now solve the inner minimization problem with the α_n values fixed to any feasible values. To do so, we simply find the stationary equations for $\mathbf{w}$ and then solve them analytically:

$$\nabla_{\mathbf{w}} \left(\frac{\lambda}{2}\|\mathbf{w}\|_2^2 - \sum_{n=1}^{N} (\alpha_n y_n \mathbf{x}_n \mathbf{w} - H(\alpha_n)) \right) = 0$$

$$\lambda \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^T = 0$$

$$\mathbf{w}_* = \frac{1}{\lambda} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^T$$

We now substitute the optimal value $\mathbf{w}_*$ back into the saddle point problem, to obtain the final solution:

$$\max_{\alpha_{1:N}} \quad \frac{\lambda}{2}\|\frac{1}{\lambda}\sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^T\|_2^2 - \sum_{n=1}^{N} \left(\alpha_n y_n \mathbf{x}_n \left(\frac{1}{\lambda} \sum_{m=1}^{N} \alpha_m y_m \mathbf{x}_m^T \right) - H(\alpha_n) \right) \quad \text{s.t.} \quad \forall n \;\; \alpha_n \in [0,1]$$

$$\max_{\alpha_{1:N}} \frac{1}{2\lambda}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \mathbf{x}_n\mathbf{x}_m^T - \frac{1}{\lambda}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \mathbf{x}_n\mathbf{x}_m^T + \sum_{n=1}^{N} H(\alpha_n) \quad \text{s.t.} \quad \forall n \;\; \alpha_n \in [0,1]$$

$$\max_{\alpha_{1:N}} \sum_{n=1}^{N} H(\alpha_n) - \frac{1}{2\lambda}\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_n\alpha_m y_n y_m \mathbf{x}_n\mathbf{x}_m^T \quad \text{s.t.} \quad \forall n \;\; \alpha_n \in [0,1]$$

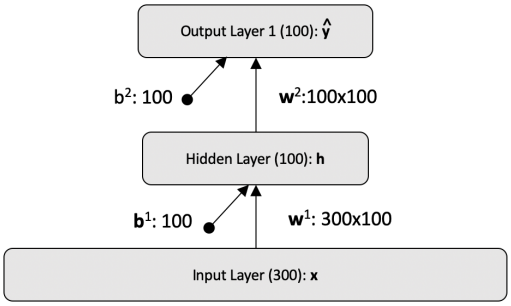
b. (*6 pts***)** Give two reasons why one one might want to optimize this dual objective for logistic regression instead of the primal objective.

Example Solutions: Some reasons why you might want to optimize the dual are (1) like for SVMs, the dual only depends on inner products and thus we can apply the use of a kernel function $\mathcal{K}(\mathbf{x}_n, \mathbf{x}_m)$ to make the model non-linear; (2) when D is larger than N, the dual has fewer parameters to learn; (3) learning in the dual can be faster than in the primal when $D > N$ if we pre-compute and cache the inner products as $\mathbf{K}_{nm} = \mathbf{x}_n\mathbf{x}_m^T$; (4) using the dual removes the $\exp()$ terms from the objective, which can reduce issues with numerical stability due to overflow or underflow of $\exp()$; (5) the gradient of the dual does not need and special functions, making it potentially faster to compute than the primal.

Some properties that do *not* hold for the dual are (1) this dual does not have a support vector property (this holds for SVMs because of hinge loss, but does not hold exactly for the logistic loss); (2) the dual does not have simpler constraints than the primal (the primal objective is the logistic loss with ℓ_2 regularization – it has no constraints); (3) the dual is not easier to solve in this case than the primal in general since the dual is constrained and the primal is not; (4) the number of parameters in the dual is not always less than the primal.

2. (*20 points*) **Neural Networks for Sequence-to-Sequence Time Series Forecasting:** In this question, you will implement a neural network model for time series forecasting. The inputs for each data case n are an array of values $\mathbf{x}_n = [x_{n1}, ..., x_{nT}]$ where x_{nt} corresponds to the value of the t^{th} time point for time series n. T is the same for all instances in this problem. The desired output $\mathbf{y}_n$ corresponds to the next T' values of time series n. Specifically, $\mathbf{y}_n = [y_{n1}, ..., y_{nT'}]$ where y_{nt} corresponds to the value of time series n at time $T+t$. T' is the same for all instances in this problem. This problem is thus a special case of the general sequence-to-sequence problem where the input and output sequences have the same length for all data cases.

The architecture diagram for the basic model you will implement is shown below. We will use the following model specification: $\hat{y}_{nt} = \mathbf{h}_n\mathbf{w}_t^2 + b_t^2$ and $h_{nk} = \text{relu}(\mathbf{x}_n\mathbf{w}_k^1 + b_k^1)$ (Note: superscripts refer to layer index, not powers). The loss function we will use to learn the model is the mean squared error over the data cases and output time series values. Let $\hat{\mathbf{y}}_n = [\hat{y}_{n1}, ..., \hat{y}_{nT'}]$ be the predicted outputs for time series n. The loss function is thus: $\frac{1}{NT'}\sum_{n=1}^{N}\sum_{t=1}^{T'}(y_{nt} - \hat{y}_{nt})^2$.



The data set we are using for this problem consists of one-lead echocardiogram (ECG) traces[1]. These data have significant structure, but will be very challenging to predict accurately. This problem has not been investigated on this data set previously, so optimal architectures are not known, nor is the floor for achievable loss values.

a. (*10 pts*) Starting from the provided template (`nn.py`), implement a Scikit-Learn compatible class for the model shown above. You will develop your implementation using NumPy and PyTorch. You are strongly encouraged to use automatic differentiation methods to develop your implementation. In the implementation of `fit` for this question, use the RMSprop optimizer included with PyTorch. Use a learning rate of $1e-3$ and the RMSprop optimizer's built-in weight decay set to $1e-4$ and momentum set to 0.9. You should learn the model using a batch size equal to the full training set size.

Example Solution: Nothing to report.

b. (*5 pts*) Using the provided training data set, learn the model for 400 epochs (in this case epochs and iterations are the same as we are learning the model on full batches). Report the loss on both the training and validation sets.

Example Solution: Train MSE: 0.01827. Validation MSE: 0.019623

c. (*5 pts*) Provide line plots of the first five validation set outputs and their corresponding predictions. There should be one plot per validation case showing the true and predicted time series values. Be sure to provide a legend and axis labels.

Example Solution: See Figure 1.

3. (*60 points*) **More Neural Networks:** In this problem, your task is to attempt to improve on the performance of the model and learning approach presented in Question 2. You can make any modifications that you like to the model or the learning algorithm, but the best model you select is limited to one minute of compute time per epoch as measured in the Gradescope autograding environment.

a. (*20 pts*) Begin by exploring different models and/or algorithms for this problem. You must try at least one CNN and one RNN architecture. Describe at a high level the different architectures and/or algorithm

[1] See `https://en.wikipedia.org/wiki/Electrocardiography`

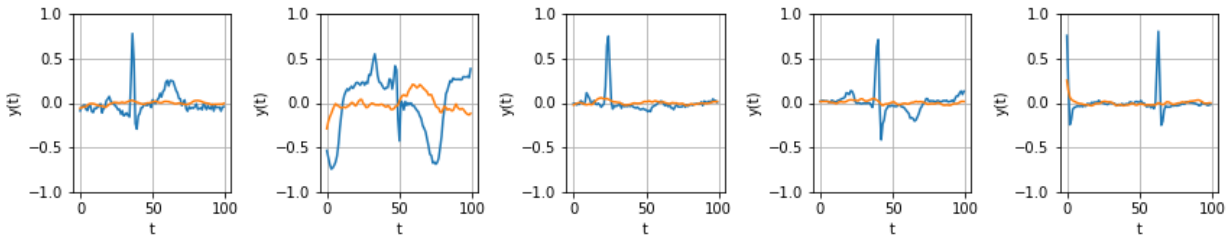


Figure 1: Plots of true and predicted time series for the first five data cases in the validation set. Blue lines are the true values. Orange lines are the predicted values.

choices that you tried. Summarize your results using tables or figures. Upload the code for one of your RNN and one of your CNN architectures in `rnn.py` and `cnn.py`.

Example Solution: For an RNN model, we used an encoder-decoder architecture with a GRU encoder and a fully connected decoder. We explored the model using numbers of hidden units in the GRU from the set $\{5, 10, 20\}$. The final GRU hidden state was passed through a linear layer to produce the final 100-dimensional prediction. To help to reduce memory requirements, we used the final 100 time points of each input sequence as input to the GRU only. We learn the model with a batch size of 1000 for 400 epochs. We use the RMSProp optimizer with momentum of 0.9, weight decay of 10^{-6} and an initial step size of $1e-3$.

For a CNN model, we construct a 1D convolutional encoder with fully connected decoder. We explored the model using numbers of filters in the set $\{5, 10, 20, 100\}$. To capture large sections of ECG cycles, we use a kernel size of 100. We set the stride to 50, half the kernel size. We apply an ReLU non-linearity following the convolution layer. This results in a filter map containing a small number of values per kernel, thus we do not apply any pooling. Instead, we flatten the resulting filter maps, and add a fully-connected decoder, which predicts the 100-dimensional output. We learn the model using full batches for 1000 epochs. We use the RMSProp optimizer with momentum of 0.9, weight decay of 10^{-6} and an initial step size of $1e-3$.

We also conduct additional experiments with the given single-layer MLP. We consider learning the model with numbers of hidden units equal to $\{5, 10, 20, 100, 200, 300\}$. The optimization details are the same as for Question 2 except for the number of epochs, which we increase to 1000.

Finally, we try a two-layer MLP using ReLU activation function. We consider hidden layer sizes in the set $\{(10, 25), (20, 50), (30, 75)\}$. For this model, some exploration showed that the model was able to learn much faster with the addition of a batch normalization transformation. We add a batch norm layer immediately before the application of the ReLU activation function for both hidden layers. We also found somewhat better performance for this model when training using mini batches instead of full batches. We found a batch size of 200 to work the best in informal testing. We use an initial step size of $1e-3$ and weight decay set to $1e-4$. We use the RMSProp optimizer. Informal testing showed that lower momentum values seemed to work better, so we selected 0.5 as the momentum value. We use 2000 epochs for learning.

The results of these experiments are shown in Table 1. As we can see, we were able to achieve a small improvement in the model from Question 2 (FC1NN) by increasing the number of training iterations as well as the number of hidden units. Based on the architectures and hyperparameters used, the RNN and CNN

K	FC1NN		RNN		CNN	
	Train	Validation	Train	Validation	Train	Validation
5	0.02009	0.02011	0.01996	0.01981	0.02000	0.02125
10	0.01987	0.2003	0.01964	0.01965	0.01924	0.02123
20	0.01940	0.01990	0.01944	0.01983	0.01812	0.2252
100	0.01813	0.01958	-	-	0.01331	0.02567
200	0.01777	0.01948	-	-	-	-
300	0.01768	0.01944	-	-	-	-

K1,K2	FC2NN	
	Train	Validation
10, 25	0.01889	0.1946
20, 50	0.01749	0.01864
30, 75	0.016874	0.01852

Table 1: The left table shows results for the FC1NN model, the RNN model and the CNN model. The right table shows results from the FC2NN model. All values are MSE on the indicated data set.

models were not very successful compared to the FC1NN model. The RNN model was found to be quite slow to learn in this configuration while its performance is similar to FC1NN at the same hidden dimension sizes. By contrast, the CNN model was found to significantly overfit in the configuration tested. While the error on the training set was significantly reduced compared the the FC1NN and RNN models, the validation set error was much higher. Finally, as we can see, the 2-layer fully-connected model (FC2NN) is able to achieve better generalization performance than any of the other models considered. The model with 30 hidden units in the first layers and 75 hidden units in the second layer achieves a validation error of 0.01852 this is a modest 5.6% reduction of the error found using the model from Question 2.

b. (*10 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 experimental procedures you used to select the single best model. Also describe why you selected this approach.

Example Solution: We decided to select the best model using minimum validation set MSE as the criteria. The primary motivation for selecting this approach is speed of computation. Some of the models assessed like the RNN require several minutes to train on a laptop. Using a more complex model selection approach such as K-fold cross validation would have resulted in a K-fold increase in training time, which would have been prohibitive for these experiments.

c. (*5 pts*) For the single best model that you selected, provide an architecture diagram (similar to the one shown above) or specify your architecture in detail using equations (Note: this question is not asking you to include a listing of the PyTorch code defining your model in your report). Upload code for your best model in `best_nn.py`.

Example Solution: See Figure 2.

d. (*10 pts*)Describe all details of the learning algorithm used for your best model. How did you select final values for any hyperparameters?

Example Solution: We use the RMSProp optimizer. We use an initial step size of $1e-3$ and weight decay set to $1e-4$. We selected 0.5 as the momentum value. We used batches of size size of 200. We learn the model for 2000 epochs. These optimization hyperparameters were selected via informal exploration. Larger

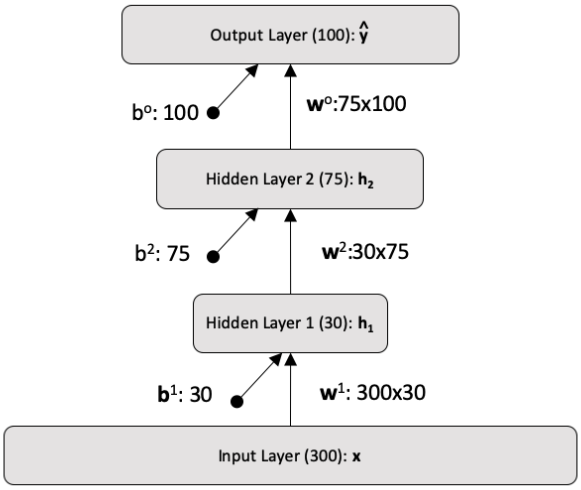


Figure 2: Architecture for selected best model

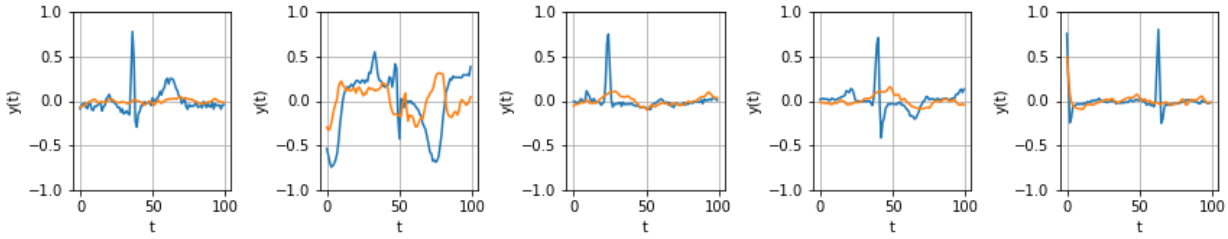


Figure 3: Plots of true and predicted time series for the first five data cases in the validation set for the selected best model. Blue lines are the true values. Orange lines are the predicted values.

learning rates resulted in failure of the objective to decrease reasonably after the first few iterations, while smaller rate seemed to also work, but converge much slower. The optimizer appeared to have difficulty with larger momentum values, so smaller values were tried. 0.5 appeared to work better than 0 in informal testing. The main hyperparameter that we evaluated was the layer widths, which control the model capacity. To set these parameters, we selected several options to try and evaluated them using validation set MSE.

e. (*5 pts*) Provide line plots of the first five validation set outputs and their corresponding predictions using your best model. There should be one plot per validation case showing the true and predicted time series values. Be sure to provide a legend and axis labels.

Example Solution: See Figure 2.

f. (*10 pts*) Using the provided test data, compute predictions for all test data cases. Save a Numpy array of predictions to your code folder using `np.save("predictions.npy", ...)`. Upload the prediction file to Gradescope for scoring. (Note: point allocations for this question will be based on the performance of your best model relative to an improved baseline model developed by the course staff, as well as to the best

solutions found by other students in the class).

Example Solution: Nothing to report.