Homework #3

CSE 446/546: Machine Learning
Prof. Kevin Jamieson
Due: November 20, 2023 11:59pm
Points A: 82; B: 5

Conceptual Questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* True or False: Training deep neural networks requires minimizing a convex loss function, and therefore gradient descent will provide the best result.

b. *[2 points]* True or False: It is a good practice to initialize all weights to zero when training a deep neural network.

c. *[2 points]* True or False: We use non-linear activation functions in a neural network's hidden layers so that the network learns non-linear decision boundaries.

d. *[2 points]* True or False: Given a neural network, the time complexity of the backward pass step in the backpropagation algorithm can be prohibitively larger compared to the relatively low time complexity of the forward pass step.

e. *[2 points]* True or False: Neural Networks are the most extensible model and therefore the best choice for any circumstance.

What to Submit:

- **Parts a-f:** 1-2 sentence explanation containing your answer.

a. False. The loss function for deep nerual networks might not convex. There may have many local minima-points that best result not guaranteened by gradient descent.

b. False. Initializing all weights with zero will cause all neurons to learn the same feature in training process. Then the later update of weights will evolve neorons symmetrically.

c. True. Real world data usually don't have linear relations. ReLU and sigmoid are common non-linear activation functions in a neural networks.

d. False. Both backword and forward complexity are similar. The forward complexity for each layer depends on the number of neurons in the current layer, the number of neurons in the previous layer. The backward pass, gradients are computed for each neuron's weights based on the partial derivative of the loss function with respect to those weights. Both should be $O(n)$, where n is the number of computations needed.

e. False. Neural networks are not necessarily fit all problems. We should consider data, problem complexity and other factors to decide if we need to use neural networks or others.

Kernels

A2. *[5 points]* Suppose that our inputs x are one-dimensional and that our feature map is infinite-dimensional: $\phi(x)$ is a vector whose ith component is:

$$\frac{1}{\sqrt{i!}} e^{-x^2/2} x^i \ ,$$

for all nonnegative integers i. (Thus, ϕ is an infinite-dimensional vector.) Show that $K(x, x') = e^{-\frac{(x-x')^2}{2}}$ is a kernel function for this feature map, i.e.,

$$\phi(x) \cdot \phi(x') = e^{-\frac{(x-x')^2}{2}} \ .$$

Hint: Use the Taylor expansion of $z \mapsto e^z$. (This is the one dimensional version of the Gaussian (RBF) kernel).

What to Submit:

- Proof.

$$\begin{aligned}
\phi(x) \cdot \phi(x') &= \sum_{i}^{\infty} \frac{1}{\sqrt{i!}} e^{-x^2/2} x^i \frac{1}{\sqrt{i!}} e^{-x'^2/2} x'^i \\
&= \sum_{i}^{\infty} \frac{1}{i!} e^{-x^2/2} x^i e^{-x'^2/2} x'^i \\
&= e^{-x^2/2} e^{-x'^2/2} \sum_{i}^{\infty} \frac{x^i x'^i}{i!} \\
&= e^{-\frac{x^2+x'^2}{2}} \sum_{i}^{\infty} \frac{(xx')^i}{i!} \\
&= e^{-\frac{x^2+x'^2}{2}} e^{xx'} \\
&= e^{-\frac{x^2+x'^2-2xx'}{2}} \\
&= e^{-\frac{(x-x')^2}{2}}
\end{aligned}$$

A3. This problem will get you familiar with kernel ridge regression using the polynomial and RBF kernels. First, let's generate some data. Let $n = 30$ and $f_*(x) = 4\sin(\pi x)\cos(6\pi x^2)$. For $i = 1, \dots, n$ let each x_i be drawn uniformly at random from $[0, 1]$, and let $y_i = f_*(x_i) + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, 1)$. For any function f, the true error and the train error are respectively defined as:

$$\mathcal{E}_{\text{true}}(f) = \mathbb{E}_{X,Y}\left[(f(X) - Y)^2\right], \qquad \widehat{\mathcal{E}}_{\text{train}}(f) = \frac{1}{n}\sum_{i=1}^{n} (f(x_i) - y_i)^2 \ .$$

Now, our goal is, using kernel ridge regression, to construct a predictor:

$$\widehat{\alpha} = \arg\min_{\alpha} \|K\alpha - y\|_2^2 + \lambda \alpha^\top K \alpha \ , \qquad \widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i k(x_i, x)$$

where $K \in \mathbb{R}^{n \times n}$ is the kernel matrix such that $K_{i,j} = k(x_i, x_j)$, and $\lambda \geq 0$ is the regularization constant.

a. *[10 points]* Using leave-one-out cross validation, find a good λ and hyperparameter settings for the following kernels:

- $k_{\text{poly}}(x, z) = (1 + x^\top z)^d$ where $d \in \mathbb{N}$ is a hyperparameter,
- $k_{\text{rbf}}(x, z) = \exp(-\gamma \|x - z\|_2^2)$ where $\gamma > 0$ is a hyperparameter[1].

[1]Given a dataset $x_1, \dots, x_n \in \mathbb{R}^d$, a heuristic for choosing a range of γ in the right ballpark is the inverse of the median of all $\binom{n}{2}$ squared distances $\|x_i - x_j\|_2^2$.

We strongly recommend implementing either grid search or random search. **Do not use sklearn**, but actually implement of these algorithms. Reasonable values to look through in this problem are: $\lambda \in 10^{[-5,-1]}$ and $d \in [5, 25]$. You do **not** need to search over γ (you can use the heuristic given in the footnote), but if you would like to, a reasonable place to start would be to sample from a narrow gaussian distribution centered at the value described in the footnote.
Report the values of d, λ, and γ for both kernels.

b. *[10 points]* Let $\widehat{f}_{\text{poly}}(x)$ and $\widehat{f}_{\text{rbf}}(x)$ be the functions learned using the hyperparameters you found in part a. For a single plot per function $\widehat{f} \in \left\{\widehat{f}_{\text{poly}}(x), \widehat{f}_{\text{rbf}}(x)\right\}$, plot the original data $\{(x_i, y_i)\}_{i=1}^{n}$, the true $f(x)$, and $\widehat{f}(x)$ (i.e., define a fine grid on $[0, 1]$ to plot the functions).

What to Submit:

- **Part a:** Report the values of d, γ and the value of λ for both kernels as described.
- **Part b:** Two plots. One plot for each function.
- **Code** on Gradescope through coding submission.
- RBF kernel: gamma= 10.541635373659384 lambda= 0.0011721022975334804
  Poly kernel: d= 19 lambda= 2.592943797404667e-05

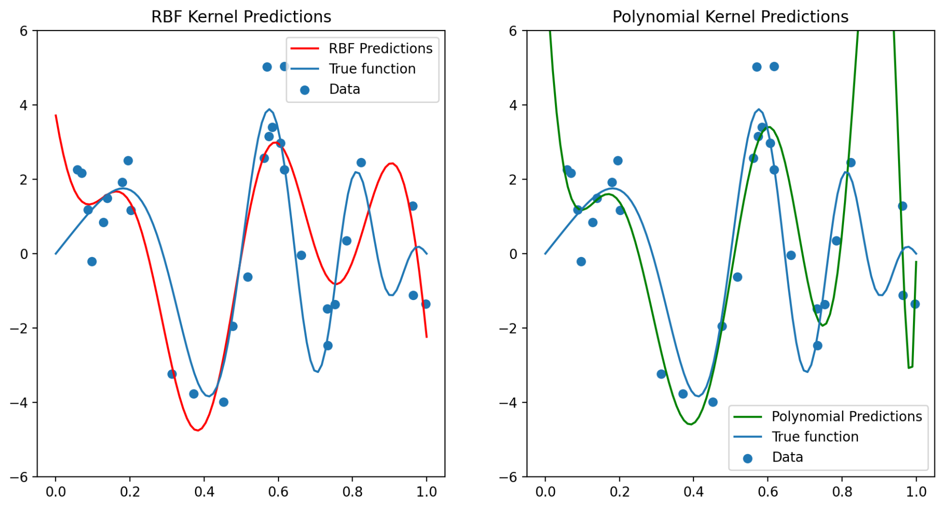


Figure 1: A3

-

Perceptron

Introduction to PyTorch

Resources

For questions A.4 and A.5, you will use PyTorch. In Section materials (Week 6) there is a notebook that you might find useful. Additionally make use of PyTorch Documentation, when needed.

A4. PyTorch is a great tool for developing, deploying and researching neural networks and other gradient-based algorithms. In this problem we will explore how this package is built, and re-implement some of its core components. Firstly start by reading `README.md` file provided in `intro_pytorch` subfolder. A lot of problem statements will overlap between here, readme's and comments in functions.

a. *[10 points]* You will start by implementing components of our own PyTorch modules. You can find these in folders: `layers`, `losses` and `optimizers`. Almost each file there should contain at least one problem function, including exact directions for what to achieve in this problem. Lastly, you should implement functions in `train.py` file.

b. *[5 points]* Next we will use the above module to perform hyperparameter search. Here we will also treat loss function as a hyper-parameter. However, because cross-entropy and MSE require different shapes we are going to use two different files: `crossentropy_search.py` and `mean_squared_error_search.py`. For each you will need to build and train (in provided order) 5 models:

- Linear neural network (Single layer, no activation function)
- NN with one hidden layer (2 units) and sigmoid activation function after the hidden layer
- NN with one hidden layer (2 units) and ReLU activation function after the hidden layer
- NN with two hidden layer (each with 2 units) and Sigmoid, ReLU activation functions after first and second hidden layers, respectively
- NN with two hidden layer (each with 2 units) and ReLU, Sigmoid activation functions after first and second hidden layers, respectively

For each loss function, submit a plot of losses from training and validation sets. All models should be on the same plot (10 lines per plot), with two plots total (1 for MSE, 1 for cross-entropy).

c. *[5 points]* For each loss function, report the best performing architecture (best performing is defined here as achieving the lowest validation loss at any point during the training), and plot its guesses on test set. You should use function `plot_model_guesses` from `train.py` file. Lastly, report accuracy of that model on a test set.

The Softmax function

One of the activation functions we ask you to implement is softmax. For a prediction $\hat{y} \in \mathbb{R}^k$ corresponding to single datapoint (in a problem with k classes):

$$\text{softmax}(\hat{y}_i) = \frac{\exp(\hat{y}_i)}{\sum_j \exp(\hat{y}_j)}$$

What to Submit:

- **Part b:** 2 plots (one per loss function), with 10 lines each, showing both training and validation loss of each model. Make sure plots are titled, and have proper legends.
- **Part c:** Names of best performing models (i.e. descriptions of their architectures), and their accuracy on test set.
- **Part c:** 2 scatter plots (one per loss function), with predictions of best performing models on test set.
- **Code** on Gradescope through coding submission

Neural Networks for MNIST

A5. In Homework 1, we used ridge regression to train a classifier for the MNIST dataset. In Homework 2, we used logistic regression to distinguish between the digits 2 and 7. Now, in this problem, we will use PyTorch to build a simple neural network classifier for MNIST to further improve our accuracy.

We will implement two different architectures: a shallow but wide network, and a narrow but deeper network. For both architectures, we use d to refer to the number of input features (in MNIST, $d = 28^2 = 784$), h_i to refer to the dimension of the i-th hidden layer and k for the number of target classes (in MNIST, $k = 10$). For the non-linear activation, use ReLU. Recall from lecture that

$$\mathrm{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \ . \end{cases}$$

Weight Initialization

Consider a weight matrix $W \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. Note that here m refers to the input dimension and n to the output dimension of the transformation $x \mapsto Wx + b$. Define $\alpha = \frac{1}{\sqrt{m}}$. Initialize all your weight matrices and biases according to $\mathrm{Unif}(-\alpha, \alpha)$.

Training

For this assignment, use the Adam optimizer from `torch.optim`. Adam is a more advanced form of gradient descent that combines momentum and learning rate scaling. It often converges faster than regular gradient descent in practice. You can use either Gradient Descent or any form of Stochastic Gradient Descent. Note that you are still using Adam, but might pass either the full data, a single datapoint or a batch of data to it. Use cross entropy for the loss function and ReLU for the non-linearity.

Implementing the Neural Networks

a. *[10 points]* Let $W_0 \in \mathbb{R}^{h \times d}$, $b_0 \in \mathbb{R}^h$, $W_1 \in \mathbb{R}^{k \times h}$, $b_1 \in \mathbb{R}^k$ and $\sigma(z)\colon \mathbb{R} \to \mathbb{R}$ some non-linear activation function applied element-wise. Given some $x \in \mathbb{R}^d$, the forward pass of the wide, shallow network can be formulated as:

$$\mathcal{F}_1(x) \coloneqq W_1 \sigma(W_0 x + b_0) + b_1$$

Use $h = 64$ for the number of hidden units and choose an appropriate learning rate. Train the network until it reaches 99% accuracy on the training data and provide a training plot (loss vs. epoch). Finally evaluate the model on the test data and report both the accuracy and the loss.

b. *[10 points]* Let $W_0 \in \mathbb{R}^{h_0 \times d}$, $b_0 \in \mathbb{R}^{h_0}$, $W_1 \in \mathbb{R}^{h_1 \times h_0}$, $b_1 \in \mathbb{R}^{h_1}$, $W_2 \in \mathbb{R}^{k \times h_1}$, $b_2 \in \mathbb{R}^k$ and $\sigma(z) : \mathbb{R} \to \mathbb{R}$ some non-linear activation function. Given some $x \in \mathbb{R}^d$, the forward pass of the network can be formulated as:

$$\mathcal{F}_2(x) \coloneqq W_2 \sigma(W_1 \sigma(W_0 x + b_0) + b_1) + b_2$$

Use $h_0 = h_1 = 32$ and perform the same steps as in part a.

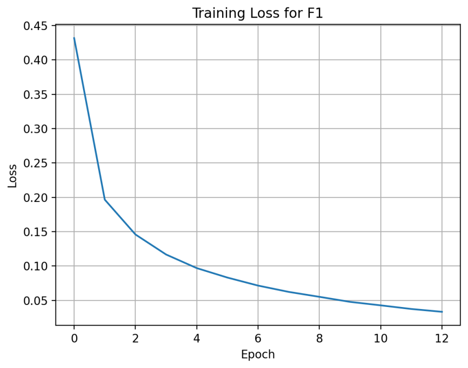
c. *[5 points]* Compute the total number of parameters of each network and report them. Then compare the number of parameters as well as the test accuracies the networks achieved. Is one of the approaches (wide, shallow vs. narrow, deeper) better than the other? Give an intuition for why or why not.

Using PyTorch: For your solution, you may not use any functionality from the `torch.nn` module except for `torch.nn.functional.relu` and `torch.nn.functional.cross_entropy`. You must implement the networks $\mathcal{F}_1$ and $\mathcal{F}_2$ from scratch. For starter code and a tutorial on PyTorch refer to the sections 6 and 7 material.

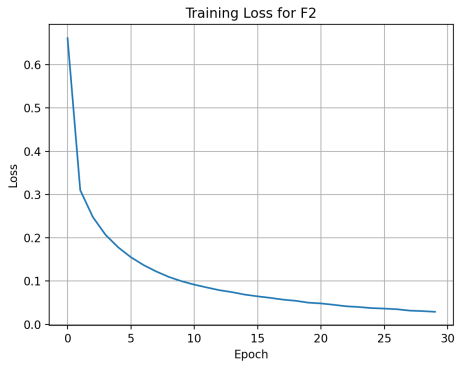
What to Submit:

- **Parts a-b:** Provide a plot of the training loss versus epoch. In addition, evaluate the model trained on the test data and report the accuracy and loss.
- **Part c:** Report the number of parameters for the network trained in part (a) and for the network trained in part (b). Provide a comparison of the two networks as described in part (c) in 1-2 sentences.
- **Code** on Gradescope through coding submission.

- F1 Test Accuracy: 0.975, F1 Test Loss: 0.0847005844116211
- F2 Test Accuracy: 0.9678, F2 Test Loss: 0.14048632979393005
- F1 Total Parameters: 50890, F2 Total Parameters: 26506
- Despite having nearly twice as many parameters, the F1 model performs better than the F2 model. This might indicate that for the MNIST dataset, having more parameters in a single hidden layer is more beneficial than distributing them across multiple layers.



(a) F1



(b) F2

Figure 2: A5

Administrative

A6.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)

b. 50 hours.