

PS4: Support Vector Machines and Neural Networks**1 SVM on a Toy Dataset (40%)**

You are given a data set D with a single feature $x \in \mathbb{R}$ and a corresponding label $y \in \{+, -\}$. The data set contains three positive examples at $x = \{-3, -2, 3\}$ and three negative examples at $x = \{-1, 0, 1\}$ (see Figure 1).

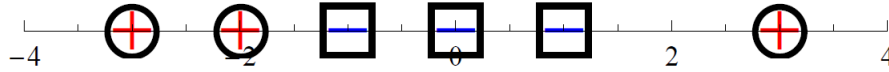


Figure 1: Data set for SVM feature map task

1. Can this data set (in its current feature space) be perfectly separated using a linear separator? Why or why not?
2. Let's define a simple feature map $\phi(u) = (u, u^2)$ which transforms points in \mathbb{R} to points in \mathbb{R}^2 . Apply ϕ to the data and plot the points in the new \mathbb{R}^2 feature space.
3. Can a linear separator perfectly separate the points in the new \mathbb{R}^2 feature space induced by ϕ ? Why or why not?
4. Give the analytic form of the kernel that corresponds to the feature map ϕ in terms of only x and x' . Specifically define $k(x, x')$.
5. Construct a maximum-margin separating hyperplane. The hyperplane will be a line in \mathbb{R}^2 , which can be parameterized by its normal equation, i.e., $w_1 z_1 + w_2 z_2 + c = 0$ for appropriate choices of w_1, w_2, c . Here, $(z_1, z_2) = \phi(x)$ is the result of applying the feature map ϕ to the original feature x .
Give the values for w_1, w_2, c . Also, explicitly compute the margin for your hyperplane. You do not need to solve a quadratic program to find the maximum margin hyperplane. Instead, let your geometric intuition guide you.
6. On the plot of the transformed points (from part 2), plot the separating hyperplane and the margin, and circle the support vectors.
7. Draw the decision boundary of the separating hyperplane in the original feature space.

8. Compute the coefficients α and the constant b in Equation (1) for the kernel k and the support vectors $SV = \{u_1, u_2\}$ you found in part 6.

$$h(x) = \text{sign} \left(\sum_{i=1}^{|SV|} \alpha_i y_i k(x, u_i) + b \right) \quad (1)$$

Hint: Think about the dual form of the quadratic program and the constraints placed on the α values.

9. If we add another positive ($y = +$) point to the training set at $x = 5$, would the hyperplane or margin change? Explain.

2 Perceptron (20%)

You are given the following training set:

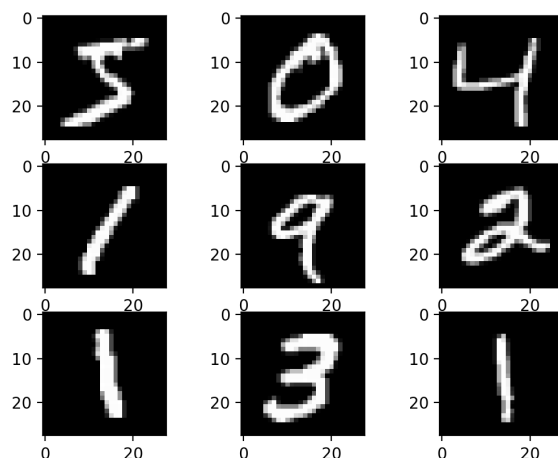
Sample	x_1	x_2	x_3	y
1	1	4	1	1
2	1	2	3	1
3	0	0	1	1
4	-1	4	0	1
5	1	0	-2	0
6	-1	-1	1	0
7	0	-4	0	0
8	1	0	-3	0

Run two iterations of the perceptron learning algorithm on this data set. Start with initial weights of 0, and use a learning rate of $\alpha = 0.5$. Don't forget to add a bias neuron $x_0 = 1$.

3 MNIST Image Classification (40%)

In this problem, you will implement a simple neural network to classify grayscale images of handwritten digits (0-9) from the MNIST data set. The data set contains 60,000 training images and 10,000 testing images of handwritten digits. Each image is 28×28 pixels in size, and is generally represented as a flat vector of 784 numbers. It also includes labels for each example, a number indicating the actual digit (0-9) handwritten in that image. A sample of a few such images are shown below.

You can download the starter code for this task from Canvas (`nn_starter.py`). The starter code splits the set of 60,000 training images and labels into a sets of 50,000 examples as the training set and 10,000 examples for validation set. You will implement a neural network with a single hidden layer and cross-entropy loss, and train it with the provided data set. Use the sigmoid function as activation for the hidden layer, and softmax function for the output layer.



Recall that for a single example (\mathbf{x}, y) , the cross-entropy loss is:

$$CE(y, \hat{y}) = - \sum_{k=1}^K y_k \log \hat{y}_k$$

where $\hat{y} \in \mathbb{R}^K$ is the vector of softmax outputs from the model for the training example \mathbf{x} , and $y \in \mathbb{R}^K$ is the ground-truth vector for the training example \mathbf{x} such that $y = [0, \dots, 0, 1, 0, \dots, 0]^T$ contains a single 1 at the position of the correct class (also called a "one-hot" representation).

The starter code already converts labels into one-hot representations for you.

For n training examples, we average the cross-entropy loss over the n examples:

$$J(W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}) = \frac{1}{n} \sum_{i=1}^n CE(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_k^{(i)} \log \hat{y}_k^{(i)}$$

Instead of batch gradient descent or stochastic gradient descent, common practice is to use mini-batch gradient descent for deep learning tasks. In this case, the cost function is defined as follows:

$$J_{MB} = \frac{1}{B} CE(y^{(i)}, \hat{y}^{(i)})$$

where B is the batch size, i.e., the number of training examples in each mini-batch.

1. Implement both forward-propagation and back-propagation for the above loss function. Initialize the weights of the network by sampling values from a standard normal distribution. Initialize the bias/intercept term to 0. Set the number of hidden units to be 300, and learning rate to be 5. Set $B = 1,000$ (mini batch size). This means that we train with 1,000 examples in each iteration. Therefore, for each epoch, we need 50 iterations to cover the entire training data.

The images are pre-shuffled. So you don't need to randomly sample the data, and can just create mini-batches sequentially.

2. Train the model with mini-batch gradient descent as described above. Run the training for 30 epochs. At the end of each epoch, calculate the value of loss function averaged over the entire training set, and plot it (y -axis) against the number of epochs (x -axis). In the same figure, plot the value of the loss function averaged over the validation set, and plot it against the number of epochs.
3. Similarly, in a new figure, plot the accuracy (on y -axis) over the training set, measured as the fraction of correctly classified examples, versus the number of epochs (x -axis). In the same figure, also plot the accuracy over the validation set versus number of epochs. Submit both plots along with the code.
4. All this while you should have stayed away from the test data completely. Now that you have convinced yourself that the model is working as expected, it is finally time to measure the model performance on the test set. Once we measure the test set performance, we report it whatever value it may be, and NOT go back and refine the model any further.

Hint: Be sure to vectorize your code as much as possible! Training can be very slow otherwise.