ECE 544NA HW1

Jiaqi Mu jiaqimu2
Department of Electrical and Computer Engineering

September 16, 2016

1 Pencil-and-Paper

Suppose that you have a one-layer neural network, of the form $y_i = g(w'x_i + b)$, where $g()$ is some nonlinearity, b is a trainable scalar bias parameter, and $w'x_i$ means the dot product between the trainable weight vector, w, and the i-th training vector, x_i. Suppose you have a training corpus of the form $D = \{(x_1, t_1), ..., (x_n, t_n)\}$. Turn in your derivations of the following 5 things.

- Find the derivatives of $\frac{dE}{dw_j}$ and $\frac{dE}{db}$, where $E = \sum_i((t_i - y_i)^2)$. Your answer should include the derivative of the nonlinearity, $g'(w'x_i + b)$.

  Proof. The derivatives are as follows,

$$\begin{aligned}
\frac{dE}{dw_j} &= \sum_i \frac{d}{dw_j}(t_i - y_i)^2 \\
&= \sum_i 2(t_i - y_i)\frac{d}{dw_j}(t_i - y_i) \\
&= \sum_i 2(t_i - y_i)(-\frac{dy_i}{dw_j}) \\
&= \sum_i 2(y_i - t_i)g'(w'x_i + b)x_{ij}, \\
\frac{dE}{db} &= \sum_i \frac{d}{db}(t_i - y_i)^2 \\
&= \sum_i 2(t_i - y_i)\frac{d}{b}(t_i - y_i) \\
&= \sum_i 2(t_i - y_i)(-\frac{dy_i}{db}) \\
&= \sum_i 2(y_i - t_i)g'(w'x_i + b),
\end{aligned}$$

  where x_{ij} is the j-th element of x_i. □

- Suppose $g(a) = a$, write $\frac{dE}{dw_j}$ without $g'()$.

  Proof. Given that $g(a) = a$, we know $g'(a) = 1$, and therefore the derivatives are,

$$\frac{dE}{dw_j} = \sum_i 2(y_i - t_i)x_{ij}.$$

□

- Suppose $g(a) = \frac{1}{1+\exp(-a)}$. In this case, $g'(w'x_i + b)$ can be written as a simple function of y_i. Write it that way,

  Proof. Since $g(a)$ is the logistic function, we can compute $g'(a)$ by,

  $$g'(a) = -\frac{-\exp(-a)}{(1+\exp(-a))^2} = \frac{1}{1+\exp(-a)}\frac{\exp(-a)}{1+\exp(-a)} = g(a)(1-g(a)).$$

  Thus one can simplify $g'(w'x_i + b)$ by,

  $$g'(w'x_i + b) = g(w'x_i + b)(1 - g(w'x_i + b)) = y_i(1 - y_i).$$

  □

- Use the perceptron error instead: $E = \sum_i \max(0, -(w'x_i + b) \cdot t_i)$.

  Proof. The derivatives are as follows,

  $$\begin{aligned}
  \frac{dE}{dw_j} &= \sum_i \frac{d}{dw_j} \max(0, -(w'x_i + b) \cdot t_i) \\
  &= \sum_i \mathbf{1}_{t_i(w'x_i+b)<0} \frac{d}{dw_j}(-w'x_i + b) \cdot t_i \\
  &= \sum_i -\mathbf{1}_{t_i(w'x_i+b<0)} t_i x_{ij} \\
  \frac{dE}{db} &= \sum_i \frac{d}{db} \max(0, -(w'x_i + b) \cdot t_i) \\
  &= \sum_i \mathbf{1}_{t_i(w'x_i+b)<0} \frac{d}{db}(-w'x_i + b) \cdot t_i \\
  &= \sum_i -\mathbf{1}_{t_i(w'x_i+b<0)} t_i
  \end{aligned}$$

  □

- Use the SVM error instead: $E = \|w\|_2^2 + C\sum_i h(x_i, t_i)$, where $h(x_i, t_i) = \max(0, 1 - t_i(w'x_i + b))$ is the hinge loss, C is an arbitrary constant, and you can assume that t_i is either $+/-1$.

Proof. The derivatives are as follows,

$$\begin{aligned}
\frac{dE}{dw_j} &= \frac{d}{dw_j}\|w\|_2^2 + C\sum_i \frac{d}{dw_j}\max(0, 1 - t_i(w'x_i + b)) \\
&= 2w_j + C\sum_i \mathbf{1}_{t_i(w'x_i+b)<1}\frac{d}{dw_j}(-t_i(w'x_i + b)) \\
&= 2w_j + C\sum_i -\mathbf{1}_{t_i(w'x_i+b)<1}x_{ij}t_i, \\
\frac{dE}{db} &= \frac{d}{db}\|w\|_2^2 + C\sum_i \frac{d}{db}\max(0, 1 - t_i(w'x_i + b)) \\
&= C\sum_i \mathbf{1}_{t_i(w'x_i+b)<1}\frac{d}{db}(-t_i(w'x_i + b)) \\
&= C\sum_i -\mathbf{1}_{t_i(w'x_i+b)<1}t_i
\end{aligned}$$

□

2 Code-From-Scratch

Write a one-layer neural net that takes a single cepstrum as input, and classifies it as ee-set versus eh-set. Write general code for training (using gradient descent), and for testing (using both MSE and classification accuracy). Use your code to train four classifiers: linear, logistic, perceptron, and linear SVM. Note that you'll have to define the targets t_i to be either +1,-1 or +1,0, depending on the classifier.

2.1 Methods:

describe the functions you wrote. Specify how particular lines of code implement particular numbered equations from your pencil-and-paper part. Include a figure showing either the dependency tree of your classes and methods, or a flowchart of training and testing, or something else descriptive.

Proof. I implemented linear classifier, logistic classifier, perceptron classifier, and linear SVM as four different classes in four different files in python as in Table 1. Each class includes three function: `__init__`, `train`, and `test`.

classifier	class	file name
linear	Linear	`linear.py`
logistic	Logistic	`logistic.py`
perceptron	Perceptron	`perceptron.py`
linear SVM	SVM	`svm.py`

Table 1: Four classifiers are implemented in four classes in four files.

- `__init__`: is to initialize the class with parameters in gradient descent such as the step size and the number of iterations.
- `train`: is to train the coefficients with training data.
  - First for each x_i, we construct a new $x_i' = [x_i, 1]$.
  - Then in each iteration, we compute the gradient and use this gradient to update the coefficients. The gradient is from Section 1
  - Finally we save the coefficients for this class.
- `test`: is to test the model with test data.

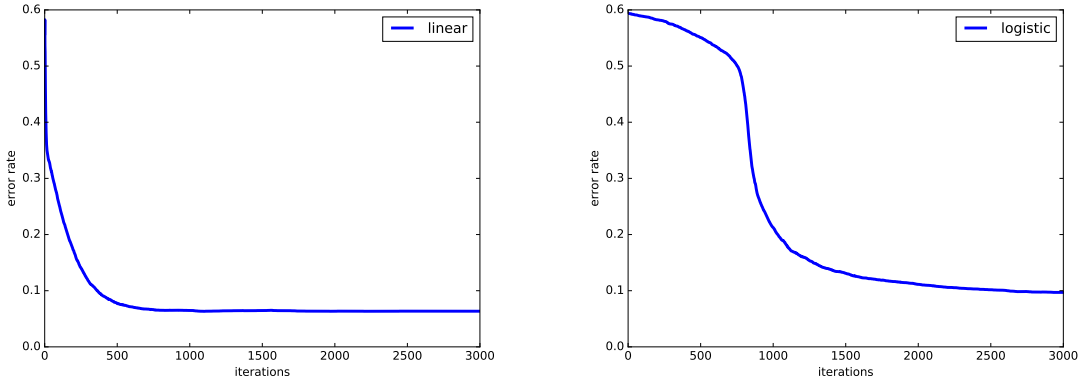
- First for each x, we construct a new $x' = [x, 1]$.
- Then we compute $y = w^T x + b$.
- Finally output a label $\hat{t}$ by thresholding y.

□

2.2 Results

- Provide one figure with four subfigures, showing convergence plots of all four classifiers (abscissa = training iteration, ordinate = training-corpus error rate).

  Proof. The figure is as Figure 1, where we choose the step size to be 2e-6, set the maximum number of iterations to be 10,000, and set the hyperparamter C in SVM to be 1. □



(a) linear (b) logistic

(c) perceptron (d) svm

Figure 1: Convergence rate of four classifiers with training step size 2e-6.

- Provide a figure, showing the training-corpus and development-test-corpus error rates of at least four different SVM training runs, using five different values of the C parameter (abscissa = C, ordinate = error rate).

  Proof. The plot is shown in Figure 2, where we see the lowest error rate of SVM is achieved at $C = 10$. □

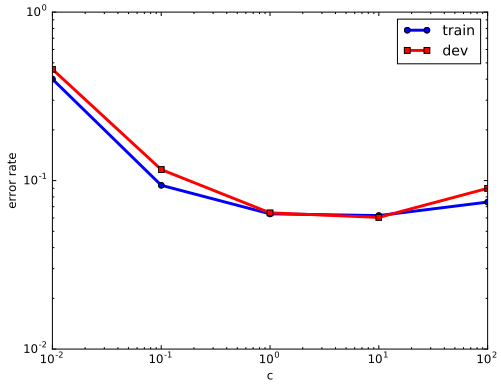


Figure 2: Error rate on training-corpus and development-corpus of SVM with C varied from 1e-2 to 1e2.

- Provide a table, showing the evaluation-test-corpus error rates of all four classifiers (including whichever SVM has lowest error rate on the development-test corpus).

  Proof. We set the step size to be 2e-6, set the number of iterations to be 3000, and set the parameter C to be 10 in SVM. The resulting performances is listed in Table 2. □

classifier	error rate(x100)
linear	3.5
logistic	7.1
perceptron	6.9
linear SVM	3.4

Table 2: Error rate on test set for four classifiers.

- Provide a figure showing a scatter plot of 300 randomly selected training tokens, with 'o' used to signify ee-set and 'x' used to signify eh-set tokens, depicted in the space defined by their first two principal components, with lines drawn across the space in four different colors showing the boundary lines of the four different classifiers.

  Proof. We use `np.random.choice` to randomly generated 300 tokens from the training data, project them into two dimension using PCA. The lines are drawn accordingly in Figure 3. □

3 TensorFlow

Train a one-layer neural net, with logistic output nodes, using TensorFlow. Hint: this will be very similar to MNIST for beginners[1], but with different data. This part of the assignment should be written in Python.

Note that a logistic node is exactly equal to a two-class softmax node. Cross-entropy loss is not the same as MSE loss; you can use whichever one you prefer.

What to turn in:

3.1 Methods

Discuss which TensorFlow functions you used, and how.

Proof. The functions are listed as follows:

- `tf.placeholder`: is used to hold a space for some values,

[1] `https://www.tensorflow.org/versions/r0.10/tutorials/mnist/beginners/index.html`

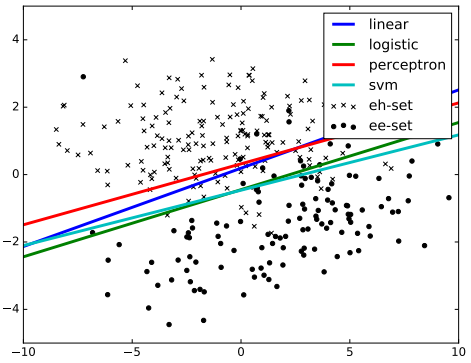


Figure 3: Visualization of four linear classifiers.

- `tf.Variable`: is used to store variables.
- `tf.softmax`: is an implemented softmax function.
- `tf.reduce_mean`: computes the mean over all examples.
- `tf.train.GradientDescentOptimizer`: is a gradient descent optimizer.
- `tf.equal`: is to check whether two elements are equal or not.
- `tf.argmax`: is an argmax function.
- `tf.matmul`: is a multiplication between matrices.
- `tf.initialize_all_variables`: is for initialization
- `tf.Session`: is an interactive session.

□

3.2 Results

Provide a convergence figure (abscissa = training iteration, ordinate = training corpus error rate). Provide a table showing error rate on the training corpus, and on the evaluation-test corpus.

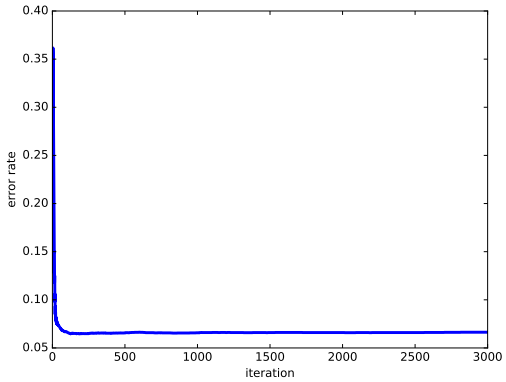


Figure 4: Convergence rate of softmax implemented by tensorflow.

Proof. The figure is in Figure 4, where we set the step size to be 0.2. The performances on the training corpus and test corpus are reported in Table 3. □

train	**test**
6.6	3.6

Table 3: Error rate (x100) on training set and test set for softmax classifier.