Neural Network Theory and Applications
Assignment 1

Tianyue Cao
119034910071

April 22, 2020

Problem 1

One variation of the perceptron rule is:

$$W^{new} = W^{old} + \alpha e p^T$$

$$b^{new} = b^{old} + \alpha e$$

where α is the learning rate. Prove convergence of this algorithm. Does the proof require a limit on the learning rate? Explain.

1. Solution

Theorem (Block Novidoff, 1962)[1] If the training data $D = \{(x_1, y_1), ..., (x_N, y_N)\}$ is linearly separable with margin γ by a unit norm hyperplane $w_*(||w_*|| = 1)$ with $b = 0$, then perceptron training converges after $\frac{R^2}{\gamma^2}$ errors during training (assuming $||x|| < R$ for all x). The output of the algorithm is noted as a.

Proof Similarly to the proof process of the Novidoff Theorem, we can prove the convergence of the variation of the perceptron. For the training data $D = \{(p_1, t_1), ..., (p_N, t_N)\} \in \mathbb{Z}^m$, we assume the dataset is linear seperatable, label $t_i \in \{-1, 1\}, i \in 1, 2, ..., N$. For convinence, note that $\hat{w} = (W^T, b)^T$ and $\hat{p} = (p^T, 1)^T$. Then $\hat{w} \cdot \hat{p} = Wp^T + b$.

Assume the classification hyperplane is $\hat{w}_{opt}\hat{p}^T = w_{opt}p^T + b_{opt} = 0$ and satisfies $\|\hat{w}_{opt}\| = 1$. Since the dataset is classified correctly, then for fininte $i \in 1, 2, ..., N$, we have

$$t_i(\hat{w}_{opt}\hat{p}_i) = t_i(w_{opt}p_i + b_{opt}) > 0.$$

It has lower bound

$$\gamma = \min t_i(w_{opt}p_i + b_{opt}) \tag{1.1}$$

i.e

$$\begin{cases} \hat{w}_{opt}\hat{p}_i \geq \gamma & t_i = 1 \\ \hat{w}_{opt}\hat{p}_i \leq -\gamma & t_i = -1 \end{cases}$$

According to the variation of the perceptron algorithm, we assume the original $\hat{w}_0 = 0$, if the sample is misclassified, then $\hat{w}_0$ should be modified, and the index of the parameter should plus 1. E.g. after $k-1$ times of modification, $\hat{w}$ should be represented as $\hat{w}_{k-1}$, i.e. $\hat{w}_{k-1} = (W_{k-1}^T, b_{k-1})^T$.

Assume after $k-1$ times of modification, (p_i, t_i) is misclassified by the hyperplane, then

$$t_i(\hat{w}_{k-1}\hat{p}_i) = t_i(W_{k-1}p_i + b_{k-1}) \leq 0 \tag{1.2}$$

$\hat{w}_k$ should be modified:

$$\hat{w}_k = \hat{w}_{k-1} + \alpha e \hat{p}_i \tag{1.3}$$

According to equation (1.1) and (1.3),

$$\hat{w}_k\hat{w}_{opt} = \hat{w}_{k-1}\hat{w}_{opt} + \alpha(t_i - a_i)\hat{w}_{opt}\hat{p}_i \geq \hat{w}_{k-1}\hat{w}_{opt} + 2\alpha\gamma$$

Then

$$\hat{w}_k\hat{w}_{opt} \geq \hat{w}_{k-1}\hat{w}_{opt} + 2\alpha\gamma \geq \hat{w}_{k-2}\hat{w}_{opt} + 4\alpha\gamma \geq \ldots \geq 2k\alpha\gamma \tag{1.4}$$

According to equation (1.2) and (1.3),

$$\begin{aligned}\|\hat{w}_k\|^2 &= \|\hat{w}_{k-1}\|^2 + 2\alpha(t_i - a_i)\hat{w}_{k-1}\hat{p}_i + \alpha^2(t_i - a_i)^2\|\hat{p}_i\|^2 \\ &\|\hat{w}_{k-1}\|^2 + 4\alpha^2\|\hat{p}_i\|^2\end{aligned}$$

Let $R = \max_{1\leq i\leq N} \|\hat{p}_i\|^2$, then

$$\|\hat{w}_k\|^2 \leq \|\hat{w}_{k-1}\|^2 + 4\alpha^2R^2$$

Then

$$\|\hat{w}_k\|^2 \leq \|\hat{w}_{k-1}\|^2 + 4\alpha^2R^2 \leq \|\hat{w}_{k-2}\|^2 + 8\alpha^2R^2 \leq \ldots \leq 4k\alpha^2R^2 \tag{1.5}$$

According to equation (1.4) and (1.5),

$$2k\alpha\gamma \leq \hat{w}_k\hat{w}_{opt} \leq \|\hat{w}_k\|\|\hat{w}_{opt}\| \leq 2\sqrt{k}\alpha R$$

So the max iteration k should satisfy

$$k \leq (\frac{R}{r})^2$$

where $R = \max_{1\leq i\leq N} \|\hat{p}_i\|$, $\gamma = \min_i\{t_i(\hat{w}_{opt}\hat{p}_i)\}$.

And as shown in the proof, it does not require a limit on the learning rate.

Problem 2

Suppose the output of each neuron in a multilayer perceptron network is

$$x_{kj} = f(\Sigma_{i=1}^{N_{k-1}}(u_{kji}x_{k-1}^2 + v_{kji}x_{k-1,i}) + b_{kj}),$$

$$\text{for } k = 2,3,\ldots,M \text{ and } j = 1,2,\ldots,N_k$$

where both u_{kji} and v_{kji} are the weights connecting the i-th unit in the layer k-1 to the j-th unit in the layer k, b_{kj} is the bias of the j-th unit in the layer k, N_k is the number of units if the $k(1 \leq k \leq M)$, and $f(\cdot)$ is the sigmoid activation function.
The structure of the unit is shown as Figure 1, and this network is called multi-layer quadratic perceptron (MLQP).
Please derive the back-propagation algorithms for MLQPs in both on-line learning and batch learning ways.

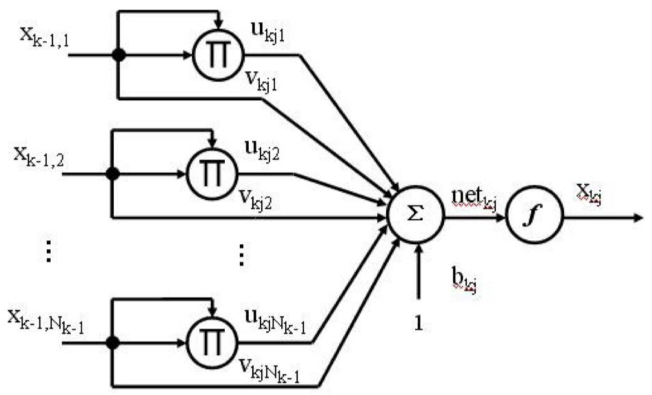

Figure 1: The structure of MLQP.

2. Solution

Online learning

The error for all neurons in the output layer is represented as:

$$\epsilon = \frac{1}{2}\Sigma_{j=1}^{N_M} e_{Mj}^2 \tag{2.6}$$

As shown in Figure 1,

$$net_{kj} = \Sigma_{i=1}^{N_{k-1}}(u_{kji}x_{k-1}^2 + v_{kji}x_{k-1,i}) + b_{kj} \tag{2.7}$$

So $x_{kji} = f(net_{kj})$ and the local gradient can be defined as $\delta_{kj} = -\frac{\partial\epsilon}{\partial net_{kj}}$. Then according to chain rule,

$$\Delta u_{kji} = -\alpha\frac{\partial\epsilon}{\partial u_{kji}} = -\alpha\frac{\partial\epsilon}{\partial net_{kji}}\frac{\partial net_{kji}}{\partial u_{kji}} = \alpha\delta_{kj}x_{k-1,i}^2 \tag{2.8}$$

Similarly,

$$\Delta v_{kji} = \alpha\delta_{kj}x_{k-1,i} \tag{2.9}$$

$$\Delta b_{kj} = \alpha\delta_{kj} \tag{2.10}$$

When $k < M$, i.e. the k-th layer is a hidden layer,

$$\begin{aligned}\delta_{kj} &= -\frac{\partial\epsilon}{\partial net_{kji}} = -\Sigma_{i=1}^{N_{k+1}}\frac{\partial\epsilon}{\partial net_{k+1,i}}\frac{\partial net_{k+1,i}}{\partial x_{kj}}\frac{\partial x_{kj}}{\partial net_{kj}} \\ &= -\Sigma_{i=1}^{N_{k+1}}\delta_{k+1,i}(2u_{k+1,ij}x_{kj} + v_{k+1,ij})f'(net_{kj}) \\ &= -\Sigma_{i=1}^{N_{k+1}}\delta_{k+1,i}(2u_{k+1,ij}x_{kj} + v_{k+1,ij})f(net_{kj})(1-f(net_{kj}))\end{aligned} \tag{2.11}$$

When $k = M$, i.e. the k-th layer is the output layer,

$$\begin{aligned}\delta_{Mj} = -\frac{\partial\epsilon}{\partial net_{kj}} &= -\frac{\partial\epsilon}{\partial e_{Mj}}\frac{\partial e_{Mj}}{\partial x_{Mj}}\frac{\partial x_{Mj}}{\partial net_{kj}} \\ &= e_{Mj}f'(net_{Mj}) \\ &= e_{Mj}f(net_{Mj})(1-f'(net_{Mj}))\end{aligned} \tag{2.12}$$

In conclusion, the online learning back propagation algorithm for MLQP is:

$$u_{kji}^{new} = u_{kji}^{old} + \alpha\delta_{kj}x_{k-1,i}^2$$
$$v_{kji}^{new} = v_{kji}^{old} + \alpha\delta_{kj}x_{k-1,i}$$
$$b_{kj}^{new} = b_{kj}^{old} + \alpha\delta_{kj}$$

where

$$\delta_{kj} = -\Sigma_{i=1}^{N_{k+1}}\delta_{k+1,i}(2u_{k+1,ij}x_{kj} + v_{k+1,ij})f(net_{kj})(1-f(net_{kj})), \quad k < M$$
$$\delta_{Mj} = e_{Mj}f(net_{Mj})(1-f'(net_{Mj}))$$

Batch learning

The error for all neurons in the output layer is represented as:

$$\epsilon = \frac{1}{2B}\Sigma_{a=1}^{B}\Sigma_{j=1}^{N_M}e_{Mj}^2 \tag{2.13}$$

where B is the size of training samples. And

$$\Delta u_{kji} = \alpha\frac{1}{B}\Sigma_{a=1}^{B}\delta_{kj}(a)x_{k-1,i}^2(a) \tag{2.14}$$

$$\Delta v_{kji} = \alpha\frac{1}{B}\Sigma_{a=1}^{B}\delta_{kj}(a)x_{k-1,i}(a) \tag{2.15}$$

$$\Delta b_{kj} = \alpha\frac{1}{B}\Sigma_{a=1}^{B}\delta_{kj}(a) \tag{2.16}$$

Then similar to the derivation process of online learning, δ_{kj} is the same as that in online learning.
In conclusion, the batch learning back propagation algorithm for MLQP is:

$$u_{kji}^{new} = u_{kji}^{old} + \alpha\frac{1}{B}\Sigma_{a=1}^{B}\delta_{kj}(a)x_{k-1,i}(a)^2$$
$$v_{kji}^{new} = v_{kji}^{old} + \alpha\frac{1}{B}\Sigma_{a=1}^{B}\delta_{kj}(a)x_{k-1,i}(a)$$
$$b_{kj}^{new} = b_{kj}^{old} + \alpha\frac{1}{B}\Sigma_{a=1}^{B}\delta_{kj}(a)$$

where

$$\delta_{kj}(a) = -\Sigma_{i=1}^{N_{k+1}}\delta_{k+1,i}(a)(2u_{k+1,ij}x_{kj}(a) + v_{k+1,ij})f(net_{kj}(a))(1-f(net_{kj}(a))), \quad k < M$$
$$\delta_{Mj}(a) = e_{Mj}(a)f(net_{Mj}(a))(1-f'(net_{Mj}(a)))$$

Problem 3

In problem 3, you should program a deep neural network with TensorFlow or PyTorch to solve an emotion recognition task.

Model Structure

- The deep neural network for this task is a three layer network: an input layer, an hidden layer, and an output layer.
- The neurons in the input layer should be the same as input feature dimensions.
- The number of hidden neurons is a hyperparameter which is chosen by yourself. And there are 3 output units for 3 emotions.

Data Description

- The provided data contains differential entropy (DE) features extracted from emotional EEG signals
- Emotions: positive, neutral, negative
- The data can be downloaded from this link: `https://bcmi.cloud:5001/sharing/8h8wHF6dz`

Questions

- Run your code to classify three emotion states (positive, neutral, negative) and compare the training time and generalization performance of different hidden units and learning rates.

3. Solution

The code is can be found in the attached file.

Environments

- Operation system: Linux
- Calculation device: 1080Ti NVIDIA GTX
- Platform: PyTorch 0.4.1

Table 1: **Results on EEG signal dataset with different hyper parameters.**

hidden layer dimension	128	128	128	128	256	256	512	512
learning rate	1e-3	1e-4	1e-4	1e-4	1e-3	1e-4	1e-3	1e-4
batch size	4	4	8	16	4	4	4	4
converge epochs (approx)	2000	1500	1500	2000	1000	300	1000	300
final training loss	0.3113	0.0331	0.2503	0.3383	0.0247	0.0011	0.0043	0.0002
training time per epoch	0.35	0.33	0.21	0.13	0.25	0.28	0.26	0.28
converge training time (approx)	707	500	315	230	254	86	268	84
training accuracy	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
test accuracy	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83

Model

According to the problem description, a 3 layer net was built to do the 3 class classification. Model structure is listed as below:

- **Input -> Hidden**
  Linear: input dimension (310) × hidden dimension
  Activation (Sigmoid) function
- **Hidden -> Output**
  Linear: hidden dimension × output dimension (3)
  Softmax function

Loss function

Here we use cross entropy loss to optimize, which is

$$\mathbb{L} = CE(p,q) = -\Sigma_{i=1}^{C} p_i log(q_i)$$

where C is the number of class, p_i is the ground truth label, q_i is the predicted label. Since the PyTorch implemententation of cross entropy loss is the combination of softmax-log-NLLLoss, so we can remove the softmax function of the output layer.

Experiments and results

We have done experiments on the given dataset with different hyper parameters, the results are shown in Table 1. Also, a figure of loss and accuracy on training dataset with 128 hidden dimension, 1e-4 learning rate, and 4 batch size is shown in Figure 2. The blue line represents accuracy on training dataset and the red line represents loss.

Analysis

According to Table 1, for different hidden layer dimension, with the hidden layer dimension increases, the final training loss decreases and the converge epoch decreases. Larger hidden dimension makes the model more representitive, but it may leads to overfit. In other experiments, if the hidden layer dimension is too small, the net vibrates and is hard

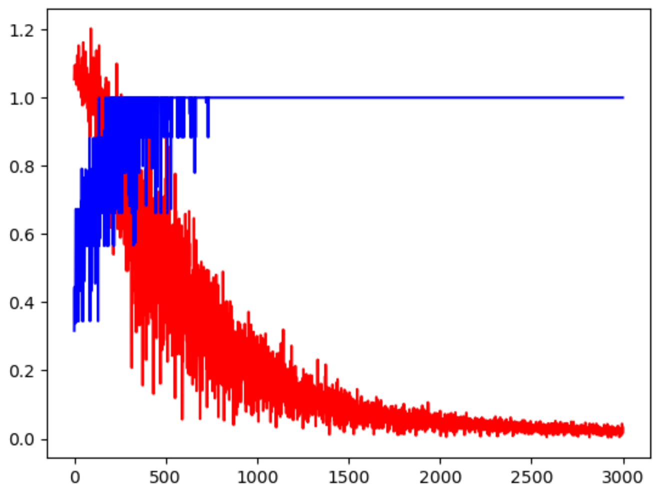

Figure 2: Loss and accuracy on training dataset with 128 hidden dimension, 1e-4 learning rate, and 4 batch size.

to converge.
For a fixed hidden layer dimension, 1e-4 is an appropriate learning rate. With learning rate decreases, converge time increases. If the learning rate is too large, the net will vibrate and be hard to converge; if the learning rate is too small, the net may be stuck in inappropriate local optimization.
Also, different batch size may influence the converge time of the model, in some region, converge time decreases with the batch size increases.
However, the test accuracy for different hyper parameters shown in Table 1 stays the same, which means all these models are converged and can achieve the same performance on the given test set. Though it does not means that the models actually have the same performance. The training dataset and the test dataset are both pretty small, so the accuracy can only be a reference of the performance. But all the models can classify the training data correctly and achieve good performance on the test dataset.

References

[1] A. B. Novikoff. On convergence proofs on perceptrons. *In Proceedings of the Symposium on the Mathematical Theory of Automata,* 1962.