Machine Learning Worksheet 07

Neural Networks 1

Problem 1: Consider a two-layer neural network in which the hidden-unit nonlinear sigmoid activation functions $\phi(\cdot)$ are given by

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{1}$$

Show that there exists an equivalent network, which computes exactly the same function, but with hidden unit activation functions given by $\tanh(x)$.

Lets assume a two-layer neural network with only one linear output unit and sigmoid hidden units. We already know how we must adopt the weights in the second layer, if we change the hidden units to tanh. From the same exercise one also can deduce that the weights in the first layer simply must be halved in order to construct a network equivalent to the original one. This can be extended straight forward to a network with multiple outputs and also to a network with sigmoid output(s) (which also could be substituted by tanh units).

Problem 2: Show that the derivative of the sigmoid activation function given in Eq. (1) can be expressed in terms of the function value itself, thus allowing for efficient implementation. Also derive the corresponding result for the tanh activation function.

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \qquad \tanh(x) = \frac{e^x + e^{-x}}{e^x + e^{-x}}$$

thus

$$\begin{aligned} \sigma'(x) &= -\frac{-e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x)) \\ \tanh'(x) &= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} = 1 - \tanh^2(x) \end{aligned}$$

These facts are usually used for efficient implementation of the gradient.

Problem 3: If we have multiple target variables $\boldsymbol{z}$, and we assume that they are independent conditional on $\boldsymbol{x}$ and $\boldsymbol{w}$ with shared noise precision β then the conditional distribution of the target values is given by

$$p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1}\boldsymbol{I})$$

Show that maximising the resulting likelihood function under the above conditional distribution for a multi-output neural network is equivalent to minimising a sum-of-squares error function.

Assuming N i.i.d. input-target pairs $(\boldsymbol{x}_i, \boldsymbol{z}_i)$ with $\boldsymbol{z}_i \in \mathbb{R}^d$ for $i = 1 \dots N$ the likelihood is

$$\prod_{i=1}^{N} p(\boldsymbol{z}_i \mid \boldsymbol{x}_i, \boldsymbol{w}) = \prod_i \left(\frac{\beta}{2\pi}\right)^d \exp\left(-\frac{1}{2}(\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)^T \beta \boldsymbol{I} (\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)\right)$$

which is equivalent (due to the diagonal nature of the covariance matrix) to

$$\prod_i \left(\frac{\beta}{2\pi}\right)^d \exp\left(\sum_{k=1}^{d} -\frac{\beta}{2}(y^k(\boldsymbol{x}_i, \boldsymbol{w}) - z_i^k)^2\right)$$

As usual we only look at the negative log likelihood and thus have to minimise (for maximising $\boldsymbol{w}$)

$$\sum_i \sum_{k=1}^{d} \frac{\beta}{2}(y^k(\boldsymbol{x}_i, \boldsymbol{w}) - z_i^k)^2$$

Problem 4: You know that the sum of squared errors is related to the Gaussian distribution—differently put, if you assume a normal distribution of the data around their expectation, the maximum likelihood estimate (MLE) is reached when the summed squared errors is minimised.

The same is true for a Laplace distribution and the sum of absolute errors. In particular, if the data observes a Laplacian distribution

$$p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} p(z_n|x_n, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \frac{1}{2\beta} \exp\left(-\frac{|z_n - y(x_n, \boldsymbol{w})|}{\beta}\right)$$

then minimising the summed absolute errors

$$\sum_{n=1}^{N} |z_n - y(x_n, \boldsymbol{w})|$$

leads to MLE. In these equations, $\boldsymbol{x}$ is the vector of all inputs, x_n is the input of sample n, while $y(x_n, \boldsymbol{w})$ is the neural network prediction on x_n. Then, z_n is the desired output for x_n.

Show that the MLE of the Laplace distribution minimises the sum of absolute errors.

Taking the negative logarithm, we obtain the error function

$$\frac{\beta}{2} \sum_{n=1}^{N} \big|z_n - y(x_n, \boldsymbol{w})\big| - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi).$$

Maximising the likelihood function is equivalent to minimising the error given by the sum of absolutes

$$\sum_{n=1}^{N} |z_n - y(x_n, \boldsymbol{w})|.$$

Problem 5: Experiment with the uploaded Jupyter notebook neuralnetworks1.ipynb.

Plot several (at least 6 total) training curves for different values of the learning rate, when classifying XOR with `mlp_xor.NeuralNetwork(X,y,0.1)` as well as the $\sin(x)$ data with `mlp_sin.NeuralNetwork(X,y,0.1)`.

Done.

Problem 6: Look at the uploaded Jupyter notebook neuralnetworks1.ipynb. In the definition of the input pattern,

```
X = np.array([  [0,0,1],
                [0,1,1],
                [1,0,1],
                [1,1,1] ])
```

why is the input in the last column always 1?

It represents the bias.

Problem 7: Look at the uploaded Jupyter notebook neuralnetworks1.ipynb. What is wrong with the use of this notebook for the XOR problem?

The Bernoulli loss should have been used for more efficient training.