Exercises Week 6

Machine Learning/Advanced Machine Learning
IT University of Copenhagen

Fall 2019

This week we will start with a couple of exercises on the Gaussian distribution, and then we will looks at neural networks.

Exercise W6.1 (programming)

For any real number $x \in \mathbb{R}$, the probability density function (PDF) for a univariate Gaussian distributionis given by

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{W6.1}$$

where $\mu \in \mathbb{R}$ is the mean and $\sigma^2 \in \mathbb{R}^+$ is the variance.

(a) Implement a function in Python that take the three arguments (x, μ, σ^2) and returns the value of $\mathcal{N}(x|\mu, \sigma^2)$.

(b) Make three individual plots of the PDF for $(\mu, \sigma^2) = (0, 1)$, $(\mu, \sigma^2) = (3, 1)$, and $(\mu, \sigma^2) = (0, 5)$ and for $x \in [-5, 5]$ using the implemented function above.

(c) For each of the three setting of (μ, σ^2), draw 10 samples from the Gaussian distribution using the function `numpy.random.normal`. Display these samples as dots along the x-axis of the corresponding plots (i.e. as points with coordinates $(x_i, 0)$ where x_i is the value of the i'th sample).

(d) Use the plots to explain the meaning of the parameters μ and σ^2.

Exercise W6.2 (math and programming)

For $x \in \mathbb{R}^D$ the PDF for a *multivariate* (D-dimensional) Gaussian distributionis defined as

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{W6.2}$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ is the mean, $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ is the covariance matrix, and $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix.

In the bivariate (2-dimensional) case, we have $\mathbf{x} = (x_1, x_2)^\intercal \in \mathbb{R}^2$, and we can write the parameters as

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}, \tag{W6.3}$$

where $\sigma_1, \sigma_2 \in \mathbb{R}^+$ and $\rho \in]-1, 1[$.

(a) Show that in the 2D case we have that $\frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}} = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}}$.

(b) For the 2D case, implement a Python function that take the following arguments $(x_1, x_2, \mu_1, \mu_2, \sigma_1, \sigma_2, \rho)$ and returns the value of $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

(c) Make six individual plots of the contours for the PDF where $\mathbf{x} \in [-3, 3]^2$.

- $(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = (0, 0, 1, 1, 0)$,
- $(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = (1, 1, 1, 1, 0)$,
- $(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = (0, 0, 1, 2, 0)$,
- $(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = (0, 0, 2, 1, 0)$,
- $(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = (0, 0, 1, 1, 0.5)$ and
- $(\mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = (0, 0, 1, 1, -0.75)$.

Hint: If you have defined a function `N(x1, x2, mu1, mu2, sigma1, sigma2, rho)`, then you can use the following code to make a contour plot.

```
import matplotlib.pyplot as plt
X1, X2 = np.meshgrid(np.linspace(-3, 3, 30), np.linspace(-3, 3, 100))
f = np.vectorize(lambda x1, x2: N(x1, x2, 0, 0, 1, 1, 0))
Z = f(X1, X2)
fig, ax = plt.subplots()
ax.axis('equal')
ax.contour(X1, X2, Z)
```

(d) Use the plots to explain the meaning of the parameters μ_1, μ_2, σ_1, σ_2 and ρ.

Exercise W6.3 (math)

The logical operator *exclusive or* is defined as $p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$. The expression $p \oplus q$ is true exactly when one of the variables p or q are true. The corresponding truth table is:

p	q	$p \oplus q$
0	0	0
1	0	1
0	1	1
1	1	0

Consider a two layer neural network with two dimensional input $\mathbf{x} \in \mathbb{R}^2$, two hidden nodes and one dimensional output $y \in \mathbb{R}$, defined by

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{2} w_j^{(2)} h \left(\sum_{i=1}^{2} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_0^{(2)} \tag{W6.4}$$

where

- $w_{10}^{(1)} = 0$ and $w_{20}^{(1)} = -1$,
- $w_{ij}^{(1)} = 1$ for $i, j \in \{1, 2\}$,
- $w_0^{(2)} = 0$, $w_1^{(2)} = 1$ and $w_2^{(1)} = -2$.

and $h(a) = \max(0, a)$ is the rectifier activation function.

Very by hand calculations that this network is equivalent to the *exclusive or* operator, i.e. that $y((0,0)^\intercal, \mathbf{w}) = 0$, $y((1,0)^\intercal, \mathbf{w}) = 1$, $y((0,1)^\intercal, \mathbf{w}) = 1$ and $y((1,1)^\intercal, \mathbf{w}) = 0$.

(a) Do the verification by expanding the sum in equation (W6.4) and inserting the different values of $\mathbf{x}$.

(b) Do the verification by drawing the network, inserting the different values of $\mathbf{x}$ and performing a forward propagation (i.e. calculating the values of all the nodes).

Exercise W6.4 (programming)

We consider the wine data set again. For this exercise we have provided a training set and a test set with filenames:

`wine_X_train.txt`, `wine_t_train.txt` and `wine_X_test.txt`, `wine_t_test.txt`.

We consider all three classes, where Barolo is class 1 or $(1, 0, 0)$, Grignolino is class 2 or $(0, 1, 0)$, and Barbera is class 3 or $(0, 0, 1)$. Start by loading in the dataset (using `np.loadtxt`).

First, we will implement multiclass logistic regression using TensorFlow. We will use the identity basis function $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$ and explicitly add a bias term. This means that we can write the activations as $a_k = \mathbf{w}_k^\intercal \mathbf{x} + b_k$. It is useful to implement this equation as

$$\mathbf{a} = \mathbf{x}W + \mathbf{b} \tag{W6.5}$$

where $\mathbf{a} = (a_1, \ldots, a_K)^\intercal$, $W = [\mathbf{w}_1, \ldots, \mathbf{w}_K]$ and $\mathbf{b} = (b_1, \ldots, b_K)^\intercal$.

(a) Start by implementing the TensorFlow graph for the model, i.e. placeholders to the input $\mathbf{x}$ and the target t, the weights W and biases $\mathbf{b}$, the activation and class probabilities. Initialize the weights with from a Gaussian.

(b) Implement the loss function using `tf.nn.softmax_cross_entropy_with_logits_v2`.

(c) Training the model using *batch* gradient decent (`GradientDescentOptimizer`) and learning rate of 0.0001 and for 50000 epochs.

(d) Calculate the accuracy on the training and test set.

Now we will implement a two layer neural network with 5 hidden nodes and the rectifier activation function for the hidden layer following the same steps as above.

(e) Which model performs the best on the test set?

Exercise W6.5 (math)

Do exercise 12.11.3 in Alpaydin. Note that the exercise text is misleadning: the update rules should be derived in the classification case where (12.19) and (12.20) are assumed.

References

E. Alpaydin. *Introduction to Machine Learning.* The MIT Press., third edition edition, 2014.

C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.