ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Models, Algorithms and Data (MAD): Introduction to Computing

Spring semester 2019

Prof. Dr. Jens Honore Walther
Dr. Georgios Arampatzis
ETH Zentrum, CLT
CH-8092 Zürich

Set 12

Issued: 15.05.2019

In this exercise, you will learn about the workhorse behind neural networks training, i.e. the Backpropagation algorithm, and get hands-on experience with `PyTorch`.

In this exercise, we deal with the **multilayer perceptron** (also called feed-forward neural network). We assume that our neural network has L layers, each with dimensions $\{n_0, \ldots, n_L\}$ (selected by the user). Here n_0 is the dimension of the input, and n_L the output dimension. The input $\mathbf{x} = (x_1, \ldots, x_{n_0})$ is mapped to the first hidden layer via

$$a_j^1 = \sum_{i=1}^{n_0} W_{ij}^1 x_i, \quad \forall j \in \{1, \ldots, n_1\}, \quad \text{and} \quad z_j^1 = \varphi_1(a_j^1) \tag{1}$$

Here φ_1 denotes the activation function (non-linearity), and z_j^1 the output of the neuron j in the first layer. In a similar fashion, the equations between consecutive hidden layers are

$$a_j^k = \sum_{i=1}^{n_{k-1}} W_{ij}^k z_i^{k-1}, \quad \forall j \in \{1, \ldots, n_k\}, \quad \text{and} \quad z_j^k = \varphi_k(a_j^k), \tag{2}$$

Finally, in the output layer, we have

$$a_j^L = \sum_{i=1}^{n_{L-1}} W_{i,j}^L z_i^{L-1}, \quad \forall j \in \{1, \ldots, n_L\}, \quad \text{and} \quad y_j = \varphi_L(a_j^L). \tag{3}$$

In vector-matrix notation with $\mathbf{w} = \{W^1, \ldots, W^L\}$ this reads

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \varphi_L\left(W^L \varphi_{L-1}\left(W^{L-1} \varphi_{L-2}\left(\cdots W^2 \varphi_1\left(W^1 \boldsymbol{x}\right)\right)\right)\right) \tag{4}$$

The most common technique to train this network is **supervised learning** based on **Backpropagation**, where **training data** $D = \{(\mathbf{x}_n, \hat{\mathbf{y}}_n)\}_{n=1,\ldots,N}$ are used to learn a mapping from the input to the output which minimized the mean squared error

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left| \hat{\boldsymbol{y}}_n - \boldsymbol{y}\left(\boldsymbol{x}_n, \boldsymbol{w}\right) \right|^2 \tag{5}$$

In the following, we are going to explain how Backpropagation works and derive the mathematical expressions.

Question 1: Shedding light in Backpropagation

In this exercise, we are going to explain how the Backpropagation algorithm works. Backpropagation is the workhorse of state-of-the-art deep neural network architectures. The name is a shortcut for **Backpropagation of errors**. Backpropagation is a method that allows us to compute the **gradient** of the loss function with respect to the **weight** of the neural network. This gradient is expressed mathematically as $\nabla_{\mathbf{w}} E$, where $\mathbf{w}$ denotes the weights and E is the loss function. For simplicity we will derive the Backpropagation rules based on the loss of a single training sample $E_n = |\hat{\mathbf{y}}_n - y(\mathbf{x}_n, \mathbf{y})|^2$. For the gradient we need to compute is the partial derivative $\partial E_n / \partial W_{ij}^k$ for every weight W_{ij}^k. Using the gradient we can formulate an gradient decent update for each weight, moving towards a descent direction in order to minimize the loss.

The first step of the Backpropagation algorithm is the **forward pass**. The input $\mathbf{x}_n$ is fixed. In the forward pass we compute the node values $\mathbf{a}_j^k$ and $\mathbf{z}_j^k$ for each node j in each layer k of the neural network, as well as the final output $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$, according to the equations (1-3). Next, we need to derive the formulas for the **backward pass**.

As a first step, we rewrite the error $\partial E_n / \partial W_{ij}^k$ using the chain rule as

$$\frac{\partial E_n}{\partial W_{ij}^k} = \frac{\partial E_n}{\partial a_j^k} \frac{\partial a_j^k}{\partial W_{ij}^k}, \tag{6}$$

where a_j^k is the output of node j at layer k **before** applying the activation function.

a) Find a formula for the second factor in equation 6, i.e. $\partial a_j^k / \partial W_{ij}^k$, using formula (2).

b) Next, we need an expression for the error gradient δ_j^k. Again, using the chain rule we can write

$$\delta_j^k = \frac{\partial E_n}{\partial a_j^k} = \sum_{i=1}^{n_{k+1}} \frac{\partial E_n}{\partial a_i^{k+1}} \frac{\partial a_i^{k+1}}{\partial a_j^k} = \sum_{i=1}^{n_{k+1}} \delta_i^{k+1} \frac{\partial a_i^{k+1}}{\partial a_j^k}, \tag{7}$$

where we decomposed δ_j^k to contributions by the error gradients of the next layer δ_i^{k+1}. Find an expression for the term $\partial a_i^{k+1} / \partial a_j^k$.
Hint: use the fact that $z_j^k = \varphi^k(a_j^k)$ and a_i^{k+1} is a weighted sum of $z_{j'}^k$ for $j' \in \{1, \ldots, n_k\}$. The final outcome includes the derivative of the activation function φ_k')

c) Now plug the expressions you found for δ_j^k and $\partial a_j^k / \partial W_{ij}^k$ in (6) to formulate the Backpropagation weight update rule $\partial E_n / \partial W_{ij}^k$ for the weight W_{ij}^k. Here, assume you are using a gradient descent optimizer with step-size η to update the weights of the neural network. Formulate the update equation in terms of the weight update ΔW_{ij}^k.

d) [optional] There are many activation functions commonly used in practice (ReLU, tanh, sigmoid, etc.). Can you find one that satisfies the following expression?

$$f'(x) = f(x)\Big(1 - f(x)\Big) \tag{8}$$

e) [optional] You build a deep neural network with **tanh** activation functions and you observe that training converges slowly. After debugging you find out that most **tanh** activation functions **saturate**. Can you justify this behavior based on the Backpropagation algorithm and the form of the nonlinearity? Which activation function would you propose to solve the problem?