Brain Inspired Computing (WS 21): Exercise sheet 4

Hand in on 23.11.2021, 14:00

Name(s): Group:

Question:	1	2	Total
Points:	50	50	100
Score:			

Exercise 1: The backpropagation algorithm (50 points)

In this exercise you will derive the backprop algorithm. Backprop has been an important ingredient in many of the machine learning successes of the last years[1].

As introduced in the lecture, the gradient along which the weights will be updated is:

$$\frac{d\mathcal{L}(t)}{dW^l_{ki}} = \underbrace{\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{z}^L(t)}}_{\text{Ⓐ}} \cdot \frac{d\boldsymbol{z}^L(t)}{d\boldsymbol{z}^{L-1}} \cdots \underbrace{\frac{d\boldsymbol{z}^{l+1}(t)}{d\boldsymbol{z}^l(t)}}_{\text{Ⓑ}} \cdot \underbrace{\frac{d\boldsymbol{z}^l(t)}{dW^l_{ki}}}_{\text{Ⓒ}} \tag{1}$$

with $\mathcal{L}(t) = -\frac{1}{2}\sum_j (z^L_j(t) - z^*_j(t))^2$.

As a prerequisite to deriving the algorithm, we need to calculate the terms of equation (1). Show that the components

(a) (5 points) of Ⓐ are given by $\frac{\partial \mathcal{L}(t)}{\partial z^L_j(t)} = -\left(z^L_j(t) - z^*_j(t)\right)$

(b) (5 points) of Ⓑ are given by $\frac{dz^{l+1}_j(t)}{dz^l_m(t)} = \varphi'(v^{l+1}_j(t)) \cdot W^{l+1}_{jm}$ for $l < L$.

(c) (10 points) of Ⓒ are given by $\frac{dz^l_j}{dW^l_{ki}} = \varphi'(v^l_j(t)) \cdot z^{l-1}_i(t) \cdot K_{jk}$ where K_{jk} is 1 if $k = j$ and 0 otherwise ("Kronecker delta").

We'll now turn our attention to the actual weight update algorithm, deriving it from last ($l = L$) to first layer ($l = 1$).

(d) (10 points) For $l = L$, derive an expression for $\delta^L_k(t)$ as a function of z^L_k, z^*_k and φ', such that $\frac{d\mathcal{L}(t)}{dW^L_{ki}} = \delta^L_k(t) \cdot z^{L-1}_i(t)$. This gives us an update rule for the last layer's weights.

[1] C.f. Deep learning, Yann LeCun, Yoshua Bengio & Geoffrey Hinton, Nature 521, 436–444, 28 May 2015, `https://doi.org/10.1038/nature14539`

(e) (10 points) For $l = L - 1$, derive an expression for $\delta_k^l(t)$ as a function of δ_k^{l+1}, W_{kj}^{l+1} and φ', such that $\frac{d\mathcal{L}(t)}{dW_{ki}^l} = \delta_k^l(t) \cdot z_i^{l-1}(t)$. Identify $\delta_k^L(t)$ in your result.

For all "lower" layers $l < L - 1$, it turns out that this structure of the update rule stays the same (which we will not show here).

We complement the weight-update with an update rule for the biases.

(f) (10 points) Show that $\frac{d\mathcal{L}(t)}{db_j^l} = \delta_j^l(t)$ (Hint: You can build on your previous results.)

Exercise 2: Learning with the Backpropagation algorithm (50 points)

You are given a Jupyter notebook called `bic_4.2_mlp.ipynb` together with 3 datasets. This script implements an example feed-forward multi-layer perceptron that already solves the XOR dataset `xor_data.npz`. However, in this exercise, the Backprop function is not correctly implemented.

(a) (10 points) Starting from this example script, implement the correct Backprop function. Since the weights and biases set in the example are already correct, your implementation should not change the weights. Plot the cost over time to verify this.

(b) (10 points) With your implemented backprop algorithm in place, change one of the weights or biases in the starting configuration and check that the network performance returns back to perfect by plotting the cost over time.

(c) (15 points) Load the data sets `func_approx_training.npz` and `func_approx_test.npz`, each containing a numpy array X and Y with 30 samples each. Modify the existing script to train a network with 1 input, and $K = 5$ hidden units and 1 output unit with the data set. This time, use the nonlinearity φ`= lambda x: scipy.special.expit(x)` (whose derivative is $\varphi(x) * \varphi(-x)$). Initialize the weights randomly[2]. Train the network over $T = 10^6$ iterations and verify the resulting weights by plotting the network's response to the test data set against the input. Plot the cost over time. Repeat this with $K = 40$ hidden units and calculate the average cost of the network on the test data set after training for both, $K = 5$ and $K = 40$.

(d) (15 points) Load the data set `bar_data.npy` and modify the script from c) to train a network with 25 inputs, 10 hidden units and 2 output units. Use the same nonlinearity as in c) and 100,000 iterations. Plot the cost over time. Verify the learned network by plotting the output units' responses $(z_1^L, z_2^L)(x(t))$ to each value in the data set. Describe what the output units code for (Hint: Inspect the input data with `imshow` and `reshape(x, 5, 5)`).

Note: Please make sure your final notebook executes correctly in order!

[2]Use `V = np.random.normal(0., 1., (K,1))`, `W = np.random.normal(0., 1., (1,K))` and `b = np.random.normal(0., 1., K)`