Prof. Dr. Andreas Hotho, Albin Zehe, Konstantin Kobs
Lehrstuhl für Data Science

01./02.06.2021

4. Assignment in "Machine Learning for Natural Language Processing"

Summer Term 2021

1 General Questions

1. How are convolutions interpreted in NLP tasks?
2. What is the effect of pooling layers (max/average pooling) in Convolutional Neural Networks? How often are they commonly used?

2 Neural Network Hiccups

Dying ReLUs

A frequently used activation function for neural networks is the ReLU-function:

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & \text{else} \end{cases}$$

ReLUs sometimes suffer from the so-called "dying ReLU" problem. In this assignment, you will see what this means and how it can occur.

Assume a neural network with a scalar output, that is, the final layer of the network consists of only one neuron $o = \sum_{i=1}^{n} w_i h_i$, where w are the weights of the layer and h is the output of the previous layer. Let's put a ReLU activation after that layer, to get $y = ReLU(o)$.

We use squared error as the loss function of the network:

$$L = se(y, t) = \frac{1}{2}(t - y)^2,$$

where t is the true label for the input.

Let $w = \begin{pmatrix} 0.2 & -0.3 & 0.5 \end{pmatrix}$, $h = \begin{pmatrix} 0.1 & 0.5 & 10.0 \end{pmatrix}$ and $t = 0.7$.

Perform the following steps:

1. Compute the gradient $\frac{\partial L}{\partial w}$!
2. Update the weights w using gradient descent with a learning rate of $\lambda = 0.1$
3. Repeat steps (1) and (2) with the updated weights, the input $h = \begin{pmatrix} 0.3 & 0.1 & 1.0 \end{pmatrix}$ and $t = 0.1$.
4. Repeat steps (1) and (2) with the updated weights, the input $h = \begin{pmatrix} 0.5 & 0.25 & 5 \end{pmatrix}$ and $t = 1$.

Describe what you find!

3 Python

3.1 Implementing Neural Networks Part 3 — Dropout

In this assignment, you will implement a neural network "library" yourself, using `Python` and `Numpy` (**`import numpy as np`**). The tool is inspired by PyTorch's implementation. This week, you will implement Dropout regularisation.

For this, implement a new module (`forward` and `backward` function) that is initialised with a parameter p, denoting the probability that a weight is dropped. Do not forget to scale the resulting weights to make up for the missing weights.

```
class Dropout:
    def __init__(self, p=0.5):
        self.p = p

    def forward(self, x: np.array) -> np.array:
        #...

    def backward(self, grad: np.array = np.array([[1]]))
     ↪  -> np.array:
        #...
```

Modify the implementation of the `NeuralNetwork` to include dropout with a specified rate (`p:float=0.5` in the constructor) after every hidden layer! Then check that each run of the program will result in different results due to the random cancellation of weights.