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

15./16.06.2021

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

Summer Term 2021

1 General Questions

1. How can we apply a CNN model to text?
   a) What is the input to the network?
   b) How do we work with different text lengths in one batch and across batches?
   c) What are the sizes of the CNN kernels?

> ? Something to think about
> Could we use **mean**-over-time pooling in the TextCNN by Kim? What problem can arise in situations where we have texts of different length?

2 Convolutions

2.1 Manual Convolution

Given the matrix

$$M = \begin{pmatrix} 0 & 6 & 1 & 1 & 6 \\ 7 & 9 & 3 & 7 & 7 \\ 3 & 5 & 3 & 8 & 3 \\ 8 & 6 & 8 & 0 & 2 \\ 4 & 3 & 2 & 9 & 1 \end{pmatrix}$$

and a kernel

$$k = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Calculate the convolution $C = M * k$ using zero padding to make the output matrix C have the same size as M and a stride of 1! Is there a difference between a convolution and a cross correlation in this case?

2.2 Rotated Convolution

When deriving the formula for backpropagation over a convolutional layer, a substitution of a form similar to this is done:

$$\begin{aligned} \frac{\partial L}{\partial x_{a,b}} &= \sum_{m=-l'}^{l'} \sum_{n=-l'}^{l'} \delta_{h_{a+m,b+n}} w_{-m+l'+1,-n+l'+1} \\ &= \delta_{h_{a-l':a+l',b-l':b+l'}} * rot_{180}(w) \end{aligned}$$

where x is the input matrix, $w \in \mathbb{R}^{r \times r}$ is the kernel matrix and $l' := \left\lfloor \frac{r}{2} \right\rfloor$ is half the width/height of the kernel matrix.

Show that the double sum on the left-hand side can indeed be expressed by the convolution with rotated kernel on the right-hand side! It is not necessary to mathematically prove this relation. There are other ways, such as tracking the transformation of single indices.

3 Python

3.1 Implementing Neural Networks Part 4 — Optimisers

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 adapt the optimisers from the first assignment to train your neural network.

1. In the first assignment, you have already implemented some common optimisers for neural networks (SGD, Nesterov Momentum and Adam).

In this exercise, you will add optimizer functions for SGD, simple Momentum, and Adam to our framework. Add classes `SGD`, `Momentum`, and `Adam` to your code. Their constructor should get the instantiated `NeuralNetwork` object and the necessary hyper-parameters such as the learning rate. The classes should provide a method called `update` that gets the weight and bias gradients from the gradient calculation. This method takes one step of the corresponding optimizer and updated the weights and biases of the given neural network model.

```
class SGD:
    def __init__(self, nn:object, lr:float):
        #...
        pass

    def update(self, W_grads: List[np.array], b_grads:
    ↪  List[np.array]) -> None:
        #...
        pass


class Momentum:
    def __init__(self, nn: object, lr: float, mu:
    ↪  float):
        #...
        pass

    def update(self, W_grads: List[np.array], b_grads:
    ↪  List[np.array]) -> None:
        #...
        pass


class Adam:
    def __init__(self, nn: object, lr: float, beta1:
    ↪  float, beta2: float):
        #...
        pass

    def update(self, W_grads: List[np.array], b_grads:
    ↪  List[np.array]) -> None:
        #...
        pass
```

2. Train the neural network you have implemented so far using these optimisers! For example, you can use the network to learn the `XOR`-Function:

x	y	`XOR(x,y)`
0	0	0
0	1	1
1	0	1
1	1	0