Deep Learning 09

Juliane Weilbach, Megan Klaiber, Amadeus Hovekamp

January 2019

1 1. Case Study: Model Desing

1.1 a) Model EEG data

First we will split the signal in 70% training and validation and 30% test data. Then the data will be encoded with 'blink'-timestamp as 1 and 'no blink'-timestamp as 0. Because it is now a binary classification problem we would use a CNN with a logistic activation for the output, a negative log likelihood for the loss and the Adam optimizer. Of course we could also use a RNN, but since the blink has no time dependency it is not necessary. A metric could be the accuracy on the data and the ROC-curve.

1.2 b)

Maybe it would be useful to do some preprocessing of the EEG data to transform the brain signals to its purer form by eliminating artifacts of the data. A good method for Preprocessing could be Indendent Component Analysis.[1]

1.3 c) Feedback

It was a nice task to think about modeling EEG data like in practice. We needed 2 hours for this exercise.

[1] Independ ent Component Analysis for EEG Data Preprocessing -Algorithms Comparison, by Izabela Rejer and Paweł Górski

2 Network equivalence

Show that there exists an equivalent network, which computes exactly the same function but with hidden unit activation functions given by tanh(z).

$$\frac{e^z - e^{-z}}{e^z - e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1} = 1 - \frac{2}{e^{2z}+1} = 1 - 2 \cdot \frac{1}{e^{2z}+1} = 2 \cdot \underbrace{\frac{1}{e^{-2z}+1}}_{\sigma(2z)} - 1$$

Figure 1: sigmoid and tanh

2.1 b)

We invested 1 hour for this part of the exercise. It is useful to see the mathematical relations between sigmoid and tanh.

3 Common mistakes

3.1 a)

```
class MyModel(nn.Module):

    def __init__(self):
        super().__init__()
        self.bias = nn.Parameter(torch.randn(20))
        self.linear = nn.Linear(20, 10)

    def forward(self, x):
        return self.linear(x) + torch.log(self.bias)

model = MyModel()
loss = nn.MSELoss()(targets, model(x))
```

Figure 2: Nan values

One problem with this code snippet is that the bias is initialized with torch.randn which could return negative values and the logarithm is not defined for negative values.

3.2 b)

Layer normalization normalizes the inputs across the features. In this code snippet the denominator variance should add an error term.

```
class ZeroMeanUnitVariance(nn.Module):

    def forward(x):
        return (x - x.mean()) / x.var().sqrt()+ ε
```

Figure 3: Layer normalization

3.3 c)

We needed 1 hour for this task. It would have been nice if there had been more common mistakes code snippets.

4 Bayesian Linear Regression

First we will calculate matrix A and its inverse. Then we use this matrix to calculate the weights, which gives us the posterior probability distribution.

4.1 a)

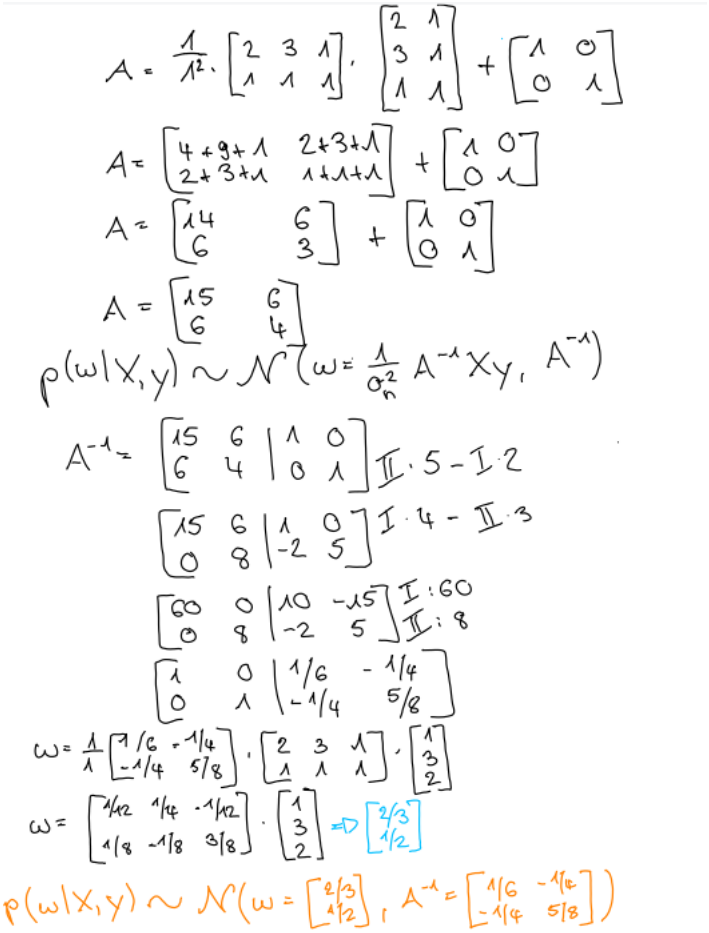


Figure 4: Posterior Calculation

4.2 c)

This was a really nice task for linear algebra refreshment. I am not sure if it really helped me to understand Bayesian Linear Regression. We needed 1 hour.

5 Convolution

Convolving the input we get:

5.1 a)

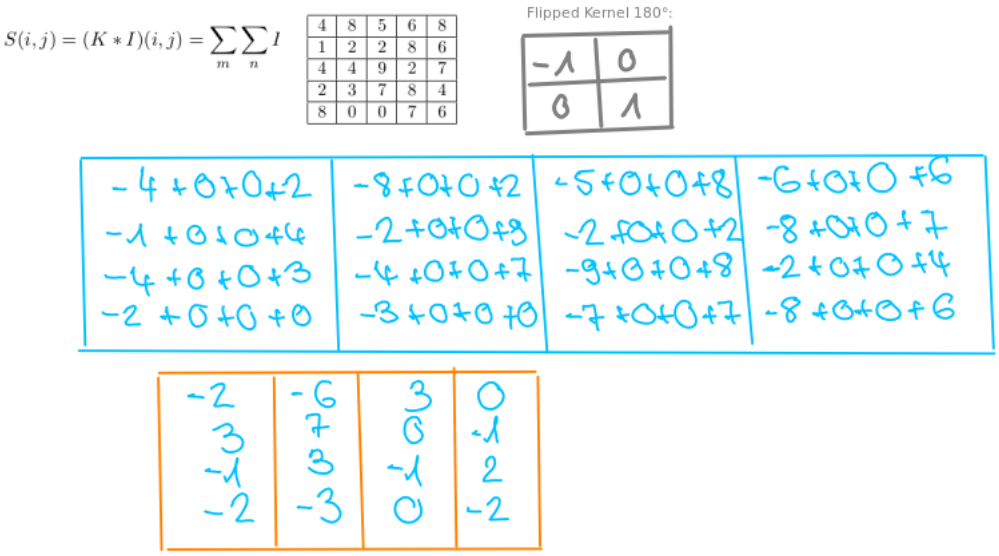
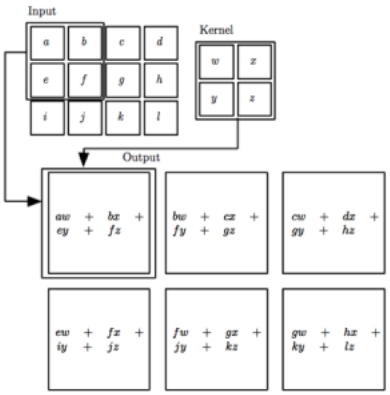


Figure 5: Convolution

5.2 b)

Next we will calculate cross-correlation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$



2	6	-3	0
-3	-7	0	1
1	-3	1	-2
2	3	0	2

$W = ((5-2)/1)+1 = 4$
$H = ((5-2)/1)+1 = 4$
Output = 4x4 ✓

Figure 6: Cross Correlation

5.3 c)

Then take the result of the cross-correlation and pass it through a ReLU layer and then sum over all the elements to get the output. Now consider the target to be 1. Assuming an L1 loss between the output and the target.

ReLU

2	6	0	0
0	0	0	1
1	0	1	0
2	3	0	2

Sum = 18

$L_1 = |1 - 18| = 17$

Figure 7: ReLU

5.4 d)

This task is near to the machine learning CNN exercise. But it is always nice to repeat this.