The Theory of Deep Learning

Student name: *Stefan Ringer*

Course: *MLSIP Lab Course Session 3 Homework*
Due date: *12.05.2021*

Task 1: Perceptrons

Scale invariance of Perceptrons:

Suppose we take all the weights and biases in a network of perceptrons, and multiply them by a positive constant, $c > 0$. Show that the behaviour of the network doesn't change.

We solve the problem by induction. First we analyze the elementary element (**induction start**), a single perceptron. The Original Perceptron is given by

$$y(x) = \begin{cases} 0 & , w \cdot x + b \leq 0 \\ 1 & , w \cdot x + b > 0 \end{cases}$$

After multiplication with the constant it is unchanged (c has to be positive for the inequality signs not change their direction. Since 0 is on the other side of the inequality, we can cancel c out ie divide by c):

$$y'(x) = \begin{cases} 0 & , c \cdot w \cdot x + c \cdot b \leq 0 \\ 1 & , c \cdot w \cdot x + c \cdot b > 0 \end{cases} = \begin{cases} 0 & , w \cdot x + b \leq 0 \\ 1 & , w \cdot x + b > 0 \end{cases} = y(x)$$

Next is the **induction step** (concatenating two neural layers). For this we analyze the repeated concatenation of two layers. This concatenation is obviously invariant to the c multiplication: Since the output of the first layer neurons is unchanged (see above), it follows that the inputs of the second layer is unchanged. We reduce the problem in the second layer to the same one as in the elementary example above. Thus it follows that the concatenation of two layers is unchanged.

Thirdly we do the **full induction**: For any arbitrarily often concatenated neural we repeatedly invoke the former conclusion and thus notice that a network of any depth made out of perceptrons is invariant under scaling of all weights and biases with a positive scalar c.

Sigmoid neurons limiting in Perceptrons:

Suppose we have the same setup as the last problem - a network of perceptrons. Suppose also that the overall input to the network of perceptrons has been chosen. We won't need the actual input value, we just need the input to have been fixed. Suppose the weights and biases are such that $wx + b = 0$ for the input x to any particular perceptron in the network. Now replace all the perceptrons in the network by sigmoid neurons, and multiply the weights and biases by a positive constant $c > 0$. Show that in the limit as $c \to \infty$ the behaviour of this network of sigmoid neurons is exactly the same as the network of perceptrons. How can this fail when $wx + b = 00$ for one of the perceptrons?

$$\sigma(c \cdot x) = \frac{1}{1 + \exp\left(-c \cdot (w \cdot x - b)\right)}$$

$$\lim_{c \to \infty} \sigma(c \cdot x) = \begin{cases} \frac{1}{1+\exp(-\infty)} = \frac{1}{1+0} = 1 & \text{if } w \cdot x + b \leq 0 \\ \frac{1}{1+\exp(+\infty)} = \frac{1}{1+\infty} = 0 & \text{if } w \cdot x + b > 0 \end{cases} = y(x)$$

We can see that this behavior is identical to the one of the perceptron. The reason is that the slope of the transition area goes to infinity and we get the hard binary classifier/step function. We can then invoke the statement of the task above to get the respective statement for the whole network.

However this argument breaks down if we allow $w \cdot x + b$ to be exactly 0. Since

$$\sigma(c \cdot x) = \frac{1}{1 + \exp\left(-c \cdot 0\right.} = \frac{1}{1 + \exp\left(0\right)} = \frac{1}{1+1} = \frac{1}{2}$$

this configuration then doesn't converge to either 0 or 1 and remains unclassified.[1]

[1] An interesting thing to consider here is the predator prey dynamics that the logistic equation describes: 0 and 1 means extinction of either species. The ratio $\frac{1}{2}$ is the only (instable) static equilibrium.

Task 2: Convolutional neural networks (CNN)

CNN Diagram:

Draw the diagram of a CNN that has following characteristics. Name each element of the CNN diagram and indicate the size of the corresponding tensor. You may assume that the sizes are perfectly divisible by parameters, when needed:

- acts on images of size N ×N
- has M feature maps with kernel size 5 × 5 and stride length 1 for convolution
- uses pooling with a 2 × 2 pooling window that is shifted across the feature maps with a stride length of 2
- has a densely connected layer of K neurons after the pooling layer
- distinguishes 10 different classes

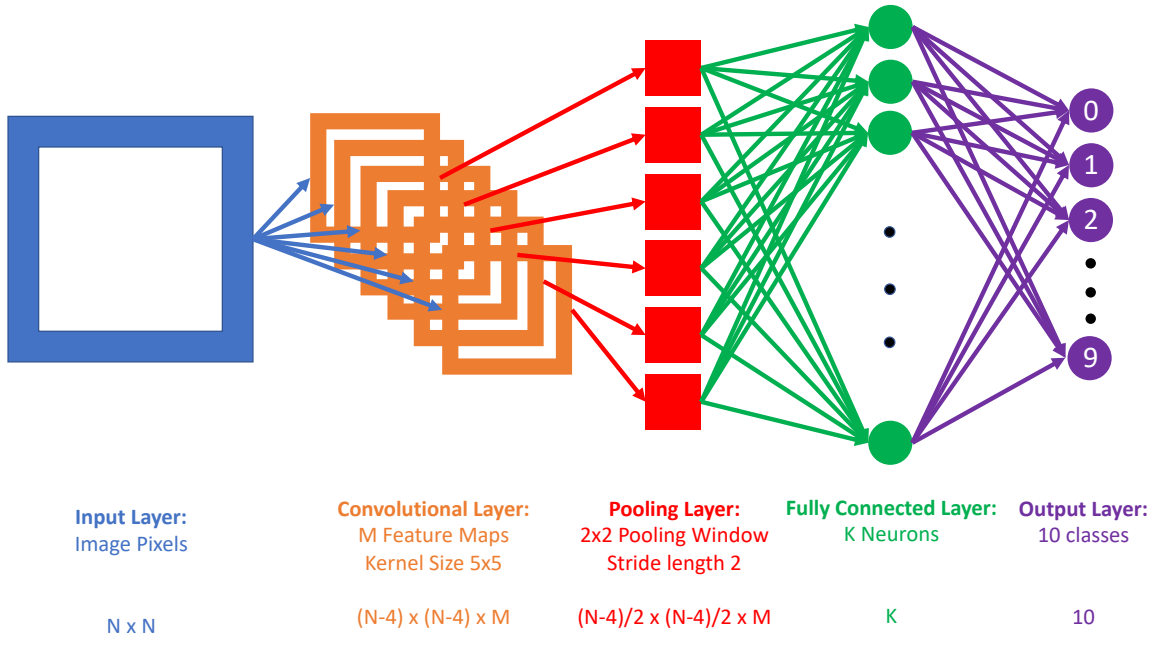

Figure 1: The CNN architecture as described above

Number of connections:

How many input connections does each neuron in the first densely connected layer have? How many input connections does each neuron in the output layer has? Assume the output layer is also densely connected. First derive the quantities in a general form, then set N = 28, M = 20, K = 100.

- $(N-4)^2 \cdot M \cdot \frac{1}{4} = (24)^2 \cdot 20 \cdot \frac{1}{4} = 2880$
- $K = 100$

Shared weights:

Briefly explain the term "shared weights" in the context of CNN. How are they used in a CNN to detect different features in an image? Why does it not matter for the feature detection where in an image the feature is located (translation invariance of CNN)?

Each feature map in a CNN is created of weighing elements of the previous ladder which are within a window (convolution kernel). For every sliding position of this local window over the whole input layer we get an element of the output feature map. The weights of this kernel are different (& learned!) position within the window. In theory they could also depend on where the sliding window itself is with respect to the input layer, this however is deliberately not done if weight sharing is employed. We expect that the features learned at this level should be fairly general (eg edges, corners...) so that they should be applicable throughout the entire image. Also this avoids overfitting since we reduce the parameters of the network and thus its capacity to just memorize the input. Also this should provide some sort of translation invariance and thus help the network generalize better.

We use several feature maps for detecting several features.

Task 3: Activation functions

ReLu:

Give the equation for the rectified linear unit (ReLU) function, ReLU(x). Draw the function for $x[4, 4]$.

$$ReLU(x) = max(0, x)$$

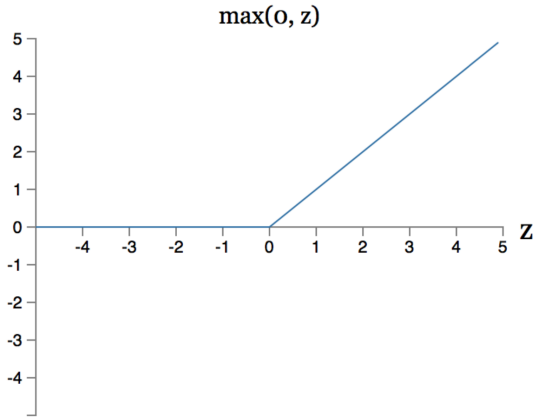

Figure 2: The ReLU activation function

Softmax:

Give the equation for the softmax activation function of the i-th neuron, softmax(x_i). Based on this equation, explain why softmax is often used in the output layer of a neural network that is used as a classifier.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

The normalization ensures that the sum of the components of the output vector is 1. Furthermore each component will be in the intervall $]0, 1[$. Therefore the output vector can be interpreted as a probability (mass) function. This makes interpretation of the inference done by the network very accessible.

Task 4: Reducing Overfitting

Loss- and classification-error:

Explain the difference between the loss and the classification error.

The loss is calculated in the training on the training data via the costfunction. From it we derive the gradient for adaption of the weights. The ultimate goal however is to reduce the classification error of the trained network which is the proportion of wrongly classified images that the network hadn't seen in the training stage (eg because the images came from the validation set).

Test set:

Explain why during training it's a good idea to regularly verify the loss of the classifier on a dataset that is independent of the training set (commonly called the test set). Does the loss evaluated on the test set always represent the true classification performance?

This is a counter meassure to recognize overfitting. Overfitting is memorizing the test data and not being able to generalize to unseen test data. It is characterized by low loss and high classification error. Thus we can intervene early on when we notice this.

Data augmentation:

An effective method to reduce overfitting is to use more training data. Explain the term data augmentation and give at least three examples of data augmentation for images.

If we give the network more data that it has to correctly classify in the training state then it is more likely that it will do this by generalizing rather than memorizing. Methods of augmentation can be rotating, translating, projective transforming, noise adding etc.

Regularization:

Regularization has empirically been shown to improve generalization of neural networks. In this method the cost function is extended by a term that penalizes large weights or enforces some other structural properties. Explain intuitively why regularization reduces overfitting.

Sometimes these constraints and penalties are designed to encode specific kinds of prior knowledge. Other times, these constraints and penalties are designed to express a generic preference for a simpler model class in order to promote generalization. Sometimes penalties and constraints are necessary to make an underdetermined problem determined. Gernerally speaking this works since we reduce the flexibility of the model by constraining the "volume" of possible parameter combinations.

Dropout:

Explain the idea of dropout method and draw a relation to the ReLU activation function.

During training, we are dropping neurons out, that means we temporarily are removing them from the network, along with all its incoming and outgoing connections.

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.[2].

If no dropout is there, neurons may change in a way that they fix up the mistakes of the other neurons. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data.

Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation as a side-effect. As such, it may be used as an alternative to activity regularization for encouraging sparse representations.

[2] A bit like in a school class when from time to time random pupils are called so that not only a few are learning. This improves the overall group performance