Assignment 7: Advanced Convolutional Neural Networks

MA-INF 2313: Deep Learning for Visual Recognition

Due Date: 31.01.2021

Assistants: Jan Müller, Leif Van Holland

Theoretical Task (15 pts)

a) Convolution layers with weight constraints *(7 points)*

Consider a neural network with general feed-forward structure, where each unit computes a weighted sum of the inputs,

$$a_j = \sum_i w_{ji} z_i$$

where z_i is the activation of a unit, or input, that sends a connection to unit j, and w_{ji} is the weight associated with that connection. Let h be a nonlinear activation function, then $z_i = h(a_i)$. We now wish to modify this neural network such that multiple weights are constrained to have the same value. Discuss how the standard back-propagation algorithm must be modified in order to ensure that the constraints are satisfied when evaluating the derivatives of the error function with respect to the adjustable parameters in the network.

b) Back-propagation in residual networks *(8 points)*

A breakthrough in the architecture of neural networks for classification problems was the development of so-called "residual networks". With the proposed architecture, the resulting networks live up to the name "Deep Learning". Networks with hundreds of layers have been successfully trained.

Let us assume that $F : V \to V$ is a function between elements of a real-valued vectorspace. A plain feedforward architecture, which we used in the previous exercises, models the function F by applying different layers successively. E.g:

$$F(x) = h_n \text{ and } h_i = a_i(W_i \cdot h_{i-1}) \text{ and } h_0 = x$$

where a_i is the activation function. However, you can assume that a_i is the identity function!

We could also say that our plain model assumes that F can be decomposed into simpler functions $F = F_m \circ ... \circ F_1$, and that it models each of the simpler functions F_i by one or more layer/activation function. We will call each sub-network, which models one simple function F_i, a block.

The main assumption of residual learning is that the simpler functions F_i can be modeled as $F_i(x) = x + H_i(x)$, where $H_i(x)$ is the residual function. Hence, a residual block does not

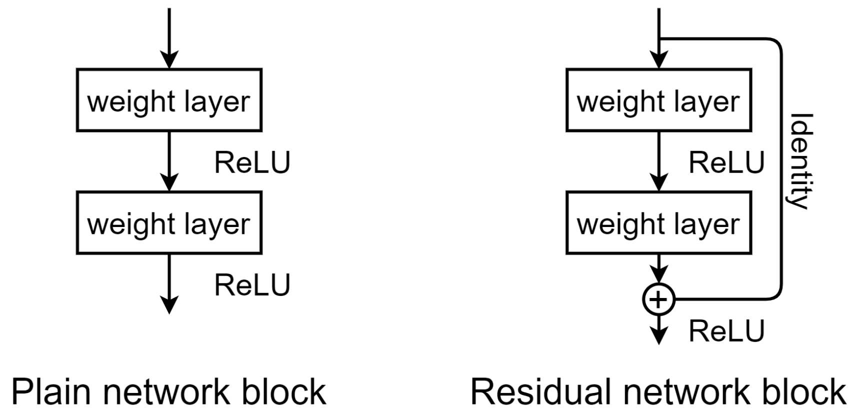

Figure 1: An illustration of two sub-network blocks. Left: A block in a plain network. Right: A residual block with two layers

estimate F_i directly, instead the block estimates the residual function H_i with a couple of layers, and computes $F_i(x) = x + H_i(x)$ by using a skip connection within the block. Figure 1 illustrates a two-layer block of a plain network and a two-layer residual block. A residual network also assumes that F can be decomposed into simpler functions F_i but the network uses the residual block to estimate these simpler functions. A simple example of the residual network is

$$F(x) = h'_n \text{ and } h'_i = h'_{i-1} + a_i(W_i \cdot h'_{i-1}) \text{ and } h'_0 = x$$

where W_i are the weights, and a_i is the non-linear activation functions. (Again you can assume that a_i is the identity function).

Your task now is to:

- Compute the derivative of the plain network's output h_n w.r.t. a previous layer h_l, $n > l$.
- Compute the derivative of the residual network's output h'_n w.r.t. a previous layer h'_l, $n > l$.
- Compare the two derivatives. Use your comparison to explain why deep residual networks are feasible, whereas increasing the depth of a plain network has diminishing returns.

Programming Task (15 pts)

a) A small residual network (*8 points*)
In your first programming task, you will implement a small residual network. The first step is

to implement a residual block according to its depiction in Figure 1. In your implementation the two weight layers are convolution layers with a 3×3 kernel size, and the ReLU activation is applied in between the convolution layers and after computing $x + H(x_i)$.

- Implement a class `ResBlock`, which inherits `th.nn.Module`. The constructor has to take the number of features of the block as an argument and should instantiate the two layer conv. network, which estimates the residual function $H_i(x)$.
- Implement a member function `forward(self, x)` which evaluates the residual function $H_i(x)$ and computes the output of the residual block $F_i(x) = a_i(x + H_i(x))$.

Now that you implemented a residual block, we can use it as a building block to create a small "ResNet".

- Implement a class `ResNet`, which inherits `th.nn.Module`. The constructor has to accept the following arguments: The number of input channels, the number of classes, and a list of integers. The length of the list describes the number of residual blocks in your network, whereas the i-th value in the list determines the number of feature channels of the i-th residual block.
- The class should implement the following network:

$$\begin{aligned}\text{Input} &\rightarrow \text{Conv}_{7x7,\text{stride=2,padding=3}} \rightarrow \text{ReLU}\\ &\rightarrow \text{ResBlock}_1 \rightarrow \text{Conv}_{3x3,\text{padding=1}} \rightarrow \text{ReLU}\\ &\rightarrow \text{ResBlock}_2 \rightarrow \text{Conv}_{3x3,\text{padding=1}} \rightarrow \text{ReLU}\\ &\dots\\ &\rightarrow \text{ResBlock}_n \rightarrow \text{AvgPooling2d}_{\text{size=2}} \rightarrow \text{Dense}_{numClasses}.\end{aligned}$$

  The convolution layer between the "ResBlocks" are used to increase the number of features and decrease the number of pixels.
- Implement a member function `forward(self, x)` which returns the currently predicted class probabilities of this ResNet.

Finally, we are able to evaluate our ResNet on the CIFAR-10 dataset. We use an Adam optimizer with a learning rate of $\eta = 0.0005$, parameters $\beta_1 = 0.9, \beta_2 = 0.95$, and minimize cross-entropy as the loss.

- Train your ResNet model on the CIFAR-10 dataset for 25 epochs, and plot the loss values and accuracy. You ResNet model should have 3 ResBlocks with 32, 64, and 128 feature channels.

- Compare its performance to last week's simpler CNN classifier.

b) Towards a competitive CIFAR-10 classifier (*7 points*)
Your final programming task is to combine all previous techniques in order to create a CIFAR-10 classifier which achieves a test accuracy of 85% or higher within the following constraints:

- You are free to use any of the architectures we implemented in the tutorial, but your model may have at most 1.8 million trainable parameters. You can use torchsummary to check if the number of trainable weights are within margin.
- You can use any optimizer, regularization, initialization and data augmentation techniques, which was part of an exercise.

Report the average test accuracy and standard deviation of at least 3 runs. Comment on your selection of hyperparameters.