HS 2022

Prof. R. Wattenhofer
Benjamin Estermann

Computational Thinking
Sample Solutions to Exercise 12

1 Neural Networks on Small Devices

a) We need to verify that the memory requirement of our model is less than the 4GB of our device. For this we need to count the numer of parameters. The model has two layers, a hidden layer with 256 nodes and an output layer with 10 nodes. The number of parameters is:

- Hidden layer: each node receives the flattened image and has a parameter per element of the flattened image plus one bias or intercept term. Since we have 256 nodes this amounts to:
$$params_{hidden} = 3771 \cdot 2121 \cdot 3 \cdot 256 + 256 = 6.14 \cdot 10^9$$
- Output layer: each node receives the output values of each hidden node, i.e. 256 and has one parameter per input value plus the intercept. Since we have ten nodes this results in:
$$params_{out} = 256 \cdot 10 + 10 = 2570$$

The total number of parameters is the sum of both. Since we represent each floating point number with 8 bits, i.e. 1 Byte, the memory requirement is larger than 6GB. Therefore, the model cannot run on our device.

b) Two reasons:

- CNNs are translation invariant which helps in image classification tasks, i.e., CNNs do not suffer from objects changing their positions in the image.
- CNNs share weights which reduces the memory requirements by a large amount.

c) We follow a similar reasoning as in part a) but with CNNs. The first layer is different from the other 9 CNN layers in that it receives three channels at the input (RGB), while the other layers receive 128 channels. This way the parameters in the first layer are:

$$params_{l1} = 3 \cdot 3 \cdot 3 \cdot 128 + 128 = 3'584$$

Where the first 3 is the number of input channels, the next two 3s correspond to the 3×3 filter and 128 are the output channels and the added 128 is the intercept/bias term. Similarly the number of parameters of each of the other layers is:

$$params_{l} = 128 * 3 * 3 * 128 + 128 = 147'584$$

The total number of parameters is:

$$params_{total} = params_{l1} + 9 \cdot params_{l} = 1'331'840$$

which means we need roughly 1.33 MB to store this network.

d) Each hidden layer consists of $1 \cdot 3771 \cdot 2121 \cdot 128 \approx 1.02 \cdot 10^9$ activation values. This on its own does not directly lead to an issue as we can fit these activation tensors into memory one at a time. What does lead to an issue is that we have to store all activation values for the computation of the gradients in the Backpropagation algorithm. As we cannot fit all activation tensors together into the memory, training yields an out of memory error or it is terribly slow (when swapping tensors to the hard drive).

e) Most image processing pipelines first crop, subsample and/or rescale the image into a lower resolution as the full resolution is not required for most classification tasks. Another solution is to use a larger stride in the convolutional layers, i.e. subsample within the network. This leads to smaller hidden layer activations and is actually very common in popular CNN architectures.

2 Zurich's Temperature

a) The model is probably overfitting, it has at least three times as many parameters as pixels. An MLP is not translation invariant and fails to generalize patterns that may appear in different parts of the image. This task is better suited for a CNN. A good first thing to try is to use a few convolutional layers that extract features of a manageable size, e.g., 8x8. Then these features are flattened and passed to a linear layer that produces the final output.

b) A main technical problem is that counting is a non-differentiable operation and thus, the gradients will not propagate back to the trainable threshold value. Other problems include that this method cannot differentiate between a white car and snow. It might therefore give an unjustified low temperature to images with many white objects.

c) You could add an RNN after some feature extracting layers. This gives the model the ability to remember the features from an earlier decision and avoid large fluctuations of temperature predictions in temporally correlated images.

d) There are many techniques for semi-supervised training. One simple approach is to pre-train a feature encoder in an unsupervised manner on the unlabeled data, e.g., train an autoencoder to compress the images into a feature vector. The encoder part can then be used as initialization for your final model and fine-tuned on the labeled data.

3 Short Sighted Agent

a) a_2, a_2, a_2 as this yields a return of 9 which is bigger than any return of a policy that ends up in the top terminal state with $R = +1$

b) From right to left we can calculate the values as:

- $V_{\pi^*}(s_2) = \max(0.1 \cdot 1, 0.1 \cdot 10) = 1$
- $V_{\pi^*}(s_1) = \max(-1 + 0.1 \cdot 1, -1 + 0.1 \cdot 1) = -0.9$
- $V_{\pi^*}(s_0) = \max(0.1 \cdot 1, 0.1 \cdot (-0.9)) = 0.1$

c) a_1 as $Q(s_0, a_1) = 0.1 > -0.09 = Q(s_0, a_2)$

d) This happens only if $Q(s_0, a_1) > Q(s_0, a_2)$. We have $Q(s_0, a_1) = \gamma \cdot 1$ and $Q(s_0, a_2) = \gamma \cdot (-1) + \gamma^2 \cdot 0 + \gamma^3 \cdot 10$ as we can only reach a higher return with a_2 in the first step if we go all the way to the terminal state on the right. The inequality therefore reads

$$\gamma > -\gamma + 10\gamma^3$$

and we get that the agent ends up in the top state for

$$\gamma < \sqrt{\frac{1}{5}} \approx 0.447$$

.