Exercises
Deep Learning
Fall 2018

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Fernando Pérez-Cruz
Web http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/

Series Monday, Oct 29, 2018
(Deep Learning, Exercise series 5)

Problem 1 (Properties of a convolutional layer):

Let $f \in \mathbb{R}^{m \times n}$ represent a 2D image and $k \in \mathbb{R}^{(2p+1)\times(2q+1)}$ represent a 2D kernel, where $m, n, p, q \in \mathbb{Z}^+$ and $2p+1 \leq m, 2q+1 \leq n$. $f(x,y)$ is the pixel value at position (x,y), where $1 \leq x \leq m$ and $1 \leq y \leq n$; similarly, $k(u,v)$ is the kernel value at position (u,v), however, $-p \leq u \leq p$ and $-q \leq v \leq q$; and $x, y, u, v \in \mathbb{Z}$.

The output of convoluting the image and the kernel is defined as

$$(f * k)(x,y) = \sum_{u=-p}^{p} \sum_{v=-q}^{q} f(x-u, y-v)k(u,v).$$

We define the translation of the image f by (s,t) as

$$(\tau_{(s,t)}f)(x,y) = f(x-s, y-t).$$

1. Prove that the convolution is translational invariant, i.e.

$$\tau_{(s,t)}(f * k) = (\tau_{(s,t)}f) * k.$$

2. Is the size of the convolution output $f * k$ the same as the size of the input image f? If not, write down the output size and also propose a method to make the convolution output size the same as the input image size.

Problem 2 (Local connectivity and parameter sharing in CNNs):

We input an RGB image of size $m \times m \times 3$ to a CNN of which the first layer is a convolutional layer and has K kernels each of size $p \times p$. and during the convolution operation, the network only retains the values from window with full signal support.

1. Write down the number of first-layer parameters of a fully-connected neural network that has the same number of outputs in the first layer as the CNN.
2. Write down the number of first-layer parameters of a locally-connected neural network that has the same connections between the input layer and the first layer as the CNN but the connection weights are not shared.
3. When the image size is $128 \times 128 \times 3$ and kernel size is 5×5, what is the ratio of the number of the first-layer parameters between the CNN and the locally-connected neural networks? And between the CNN and the fully-connected neural networks?

Problem 3 (Backpropagation through convolutional layers):

Given a CNN of which the l-th layer is a convolutional layer with just one kernel w, we denote the input of the l-lth layer, which is also the output of the previous layer, as $y^{(l-1)}$ and the output of the l-th layer is $y^{(l)} = y^{(l-1)} * w$. The joint function of the remaining part (i.e. after the l-th layer) of the CNN and the objective function is represented as $L = J(y^{(l)}) \in \mathbb{R}$.

Derive $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial y^{(l-1)}}$ (assume we already know $\frac{\partial L}{\partial y^{(l)}}$).

Problem 4 (Practical: CNNs in tensorflow):

The purpose of this exercise is for you to get familiar with convolutional neural networks as well as some important related concepts. We will use Tensorflow as well as the popular MNIST digit recognition dataset. The digit recognition task with CNNs on MNIST is a classical task which is covered in many online tutorials. We here follow the tensorflow MNIST tutorial [1] and you should not hesitate to consult it as a reference when necessary. Some of the material here directly comes from the second tutorial so we recommend reading it afterwards.

We provide a runnable skeleton implementation of the training procedure and the basic CNN. It uses `tf.flags` to provide bash-style arguments. Use the `--out_name` flag to give a meaningful name to your run (this is important when using tensorboard later).

- Sketch the architecture implemented on a piece of paper. Convince yourself that the skeleton implements a single-layer CNN consisting of a convolution followed by a relu, max-pooling, a first fully connected layers, a relu, a second fully connected layer and a soft-max.
- The CNN has `accuracy` and `loss` tensors. Create a scalar summary for each to plot the accuracy and watch the curves in tensorboard (Refer to [2] for troubleshooting tensorboard. Chrome recommended.)
- How would you add a summary for a value not computed by a tensor? For example, use python's `time.time()` to measure how long running a training step takes and add a `step_time` summary for that. This is *extremely* useful in practice to justify a computationally expensive sophistication (or not). Typically one would update this only every n steps by the average.
- Add dropout (`tf.nn.dropout`) on the fully connected layer. Don't provide a python float as keep_probability, use a `tf.placeholder`! Add a float parameter to the `tf.flags` that determines the keep probability at training time. Make sure to feed a 1.0 at test time.
- Run experiments with different keep probabilities and observe the result in tensorboard. How did the accuracy change? Did something else change?
- Regularize the weight matrix and the bias of the fully-connected layer by adding a l2-regularizer of the form
$$\lambda \left(\sum_{i,j} W_{i,j}^2 + \sum_i b_i^2 \right)$$
to the loss function. Add a summary that shows the actual value of the regularizer without λ. Run with different values of λ and inspect tensorboard.
- Add a second convolution between the first convolution (i.e. the max-pooling operation) and the first fully connected layer. Use relu and max-pooling again. To provide the right arguments to the convolution, it is crucial to know the shape of your input and to understand how `tf.nn.conv2d` [3] works.
- Experiment with the filter sizes for both layers. How far can you get the accuracy on the dev set? (don't forget your other classes)
- Train your CNN using different learning rates. For each learning rate plot a curve showing how the classification error decreases over each iteration. Fix the best learning rate for the successive steps.
- Batch normalization can be used for decreasing the training time. It also helps to reduce the strong dependence on parameter initialization and acts as a sort of regularizer. Use the *batch_norm* function from the template and add it after the two convolutional layers. Train your modified CNN and analyze the results. Does the batch-normalization function helps? Is the drop-out layer still useful?

1 https://www.tensorflow.org/tutorials/estimators/cnn#train_eval_mnist

2 https://www.tensorflow.org/guide/summaries_and_tensorboard

3 https://www.tensorflow.org/api_docs/python/tf/nn