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 - solutions)

**Solution 1 (Properties of convolutional layer):**

1.

$$\begin{aligned}
\tau_{(s,t)}(f * k)(x, y) &= \sum_{u=-p}^{p} \sum_{v=-q}^{q} f(x - s - v, y - t - v)k(u, v) \\
&= \sum_{u=-p}^{p} \sum_{v=-q}^{q} (\tau_{(s,t)} f)(x - v, y - v)k(u, v) \\
&= \big((\tau_{(s,t)} f) * k\big)(x, y).
\end{aligned}$$

2. The size of $f * k$ is $(m - 2p) \times (n - 2q)$, which is smaller than the size of $f$. To make the output size the same as the input size, we can apply padding around the input image so that the padded image has size $(m + 2p) \times (n + 2q)$, and then convolute the padded image with the kernel.

**Solution 2 (Local connectivity and parameter sharing in CNNs):**

Setting clarification: the bias is ignored for simplicity.

1. The total number of outputs of the first layer of the CNN is $3K(m - p + 1)^2$. For each channel of the image, the size of the outptu of the convolutional with a kernel is $(m - p + 1)^2$. Since there are $K$ kernels, so the size of the output from convolution with all kernels for **a** channel is $K(m-p+1)^2$. And with 3 channels in the image, the total output size is $3K(m - p + 1)^2$. To construct a fully-connected neural networks with the same number of outputs in the first layer, the number of parameters needed for the first layer is $\cancel{3Km^2(m - p + 1)^2}$ $9Km^2(m - p + 1)^2$.

2. In the CNN, each node in the first layer is only connected with $p^2$ input nodes. So for the locally-connected neural network that has the same connections between the first layer and the input layer, the number of parameters needed for the first layer is $3Kp^2(m - p + 1)^2$.

3. When the image size is $128 \times 128 \times 3$ and kernel size is $5 \times 5$, the first-layer parameter number ratio between the CNN and the locally connected neural network is $2.168 \times 10^{-5}$; and between the CNN and the fully-connected neural network, the ratio is $1.103 \times 10^{-8}$.

   **Remark:** We can see from the example that by using local connectivity and parameter sharing, CNNs reduce the number of parameters by a large amount which makes it much more computationally efficient to train them.

**Solution 3 (Backpropagation through convolutional layers):**

For simplicity, we first compute the derivative of the loss function with respect to each entry $w_{u,v}, \forall u, v \in$

$\{-k, \ldots, 0, \ldots, k\}$,

$$\begin{aligned}
\frac{\partial L}{\partial w_{u,v}} &= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} \frac{\partial y^{(l)}_{i,j}}{\partial w_{u,v}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} \frac{\partial}{\partial w_{u,v}} (\sum_s \sum_t y^{(l-1)}_{i-s,j-t} w_{s,t}) \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} y^{(l-1)}_{i-u,j-v} \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} y^{(l-1)}_{-(u-i),-(v-j)} \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} \mathtt{rot}_{180^\circ}(y^{(l-1)})_{u-i,v-j} \\
&= \big(\mathtt{rot}_{180^\circ}(y^{(l-1)}) * \frac{\partial L}{\partial y^{(l)}}\big)_{u,v}.
\end{aligned}$$

Therefore, $\frac{\partial L}{\partial w} = \mathtt{rot}_{180^\circ}(y^{(l-1)}) * \frac{\partial L}{\partial y^{(l)}}$.
Similarly,

$$\begin{aligned}
\frac{\partial L}{\partial y^{(l-1)}_{m,n}} &= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} \frac{\partial y^{(l)}_{i,j}}{\partial y^{(l-1)}_{m,n}} \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} \frac{\partial}{\partial y^{(l-1)}_{m,n}} (\sum_u \sum_v y^{(l-1)}_{i-u,j-v} w_{u,v}) \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} w_{i-m,j-n} \\
&= \sum_i \sum_j \frac{\partial L}{\partial y^{(l)}_{i,j}} \mathtt{rot}_{180^\circ}(w)_{m-i,n-j} \\
&= \big(\mathtt{rot}_{180^\circ}(w) * \frac{\partial L}{\partial y^{(l)}}\big)_{m,n},
\end{aligned}$$

hence $\frac{\partial L}{\partial y^{(l-1)}} = \mathtt{rot}_{180^\circ}(w) * \frac{\partial L}{\partial y^{(l)}}$.

**Solution 4 (Practical: CNNs in tensorflow):**

See `train_and_test_MNIST_solution.py`.