Artificial Neural Networks (Gerstner). Solutions for week 9

Convolutional neural networks

Exercise 1. Backprop in ConvNets

Consider a very simple convolutional neural network with 3 dimensional input (e.g. RGB image), one convolution layer with 5x5x3 filters, stride 1, non-linearity σ and one linear layer, i.e.

$$a_{ijk} = b_k + \sum_{x=1}^{5}\sum_{y=1}^{5}\sum_{c=1}^{3} I_{i+x-1,j+y-1,c}\, w^{(1)}_{xyck} \tag{1}$$

$$x^{(1)}_{ijk} = \sigma(a_{ijk}) \tag{2}$$

$$\hat{y}_o = \sum_{ijk} w^{(2)}_{ijko} x^{(1)}_{ijk} \tag{3}$$

a. With loss $L = \frac{1}{2}\sum_o (t_o - \hat{y}_o)^2$, compute $\partial L/\partial w^{(1)}_{1135}$.

b. Add a max pooling layer of 2×2 region that takes the outputs of Eq. (2) and transforms it into

$$x^{(1,\mathrm{out})}_{ijk} = \max\{x^{(1)}_{ijk}, x^{(1)}_{(i+1)jk}, x^{(1)}_{i(j+1)k}, x^{(1)}_{(i+1)(j+1)k}\} \tag{2b}$$

before sending it to the output. In other words, use Eq. (3) but with the upper index $(1, \mathrm{out})$ instead of (1) and calculate again the gradient. What changes?

Derive the mathematical result and give an interpretation in words.

c. Compare your result to the one you obtain with a dense (fully-connected) layer, i.e. $a_k = b_k + \sum_x \sum_y \sum_c I_{xyc} w^{(1)}_{xyck}$, $x^{(1)}_k = \sigma(a_k)$ and $\hat{y}_o = \sum_k w^{(2)}_{ok} x^{(1)}_k$.

Give an interpretation in words regarding the differences between results from (c) and (b), or (c) and (a).

Solution:

a.

$$\partial L/\partial w^{(1)}_{1135} = -\sum_o (t_o - \hat{y}_o)\,\partial \hat{y}_o/\partial w^{(1)}_{1135} \tag{4}$$

$$\partial L/\partial w^{(1)}_{1135} = -\sum_o (t_o - \hat{y}_o)\sum_{ijk} w^{(2)}_{ijko}\sigma'(a_{ijk})\,\partial a_{ijk}/\partial w^{(1)}_{1135} \tag{5}$$

$$\partial L/\partial w^{(1)}_{1135} = -\sum_o (t_o - \hat{y}_o)\sum_{ij} w^{(2)}_{ij5o}\sigma'(a_{ij5}) I_{ij3}. \tag{6}$$

b.

$$\partial L/\partial w^{(1)}_{1135} = -\sum_o (t_o - \hat{y}_o)\sum_{ij} w^{(2)}_{ij5o}\sigma'(a_{i^*(i,j)j^*(i,j)5}) I_{i^*(i,j)j^*(i,j)3}. \tag{7}$$

With max pooling the sum over the hidden activations does not run over all i and j anymore but only over those that were “winning”, i.e. had the maximal value in the group if seen by one max-pooling filter. In other words, the filter update focuses on the location that fits best to the filter.

c.

$$\partial L/\partial w^{(1)}_{1135} = -\sum_o (t_o - \hat{y}_o) w^{(2)}_{o5}\sigma'(a_5) I_{113} \tag{8}$$

In a dense network, the gradient update for the weights does not involve a summation over the input pixels, i.e. the summation over i and j. The spatial structure of the image is neglected and each input pixel is learned with a separate weight.

Exercise 2. Computing volume sizes

Given a volume of width n, height n and depth c.

a. Show that the convolution with k filters of size $f \times f \times c$ with stride s and padding p leads to a new volume of size
$$\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right) \times k\,. \tag{9}$$

b. What padding p do you have to choose such that the input and output volumes have the same width and depth for stride $s = 1$. Check you result for the special case of $n = 4$ and $f = 3$.

Solution:

a. With padding p on both sides, the width of the padded image is $n + 2p$. If we position the left edge of a filter at the left edge of the padded image – let us call this position $x = 1$ – the right border of a filter of width f will be at $x = f$. If we position the right edge of the filter at the right border of the padded image, i.e. at $x = n+2p$ the left edge of the filter is at $x = n+2p-f+1$. With stride 1 there are thus $n + 2p - f + 1$ different possibilities to position the filter relative to the image. With larger stride $s > 1$, we count only every s possibility for the first $n + 2p - f$ positions and the last, i.e. $\frac{n+2p-f}{s} + 1$. The same reasoning applies for the height and all k filters. Note that not all choices of n, p, f, s are reasonable, as there could be fractions.

b.
$$n + 2p - f + 1 = n \tag{10}$$
$$\Rightarrow p = \frac{f-1}{2} \tag{11}$$

Thus, having odd filter width and height f allows to have the same padding on the left and the right side. This is one of the reasons, odd filter widths are more common than even ones.

Exercise 3. Test of translation invariance

We start with a 1-dimensional, simple case. Suppose we apply a 1-dimensional convolutional processing stage with one filter to our 1-dimensional image and obtain a filter bank x of size 21x1. The values of these filter outputs are denoted by $x(i)$ with $1 \leq i \leq 21$.

a. Suppose that the global maximum value of $x(i)$ is at $i = 11$. We apply max-pooling with stride 1 and a pooling window size of 11. How stable is the result of the pooling with respect to shifts in x that could stem from shifts in the original input image?

b. Suppose that the values $x(i)$ are monotonically increasing with a maximum at $i = 21$. We apply the same max-pooling (stride 1, window size 11). How stable is the result of the pooling in this case?

Now we move to a 2-dimensional case. On the right there is an image we would like to process with a 2-layer convolutional network. The network has one convolutional layer (two 3x3 filters, depicted below the image, stride 1, ReLU non-linearity) and one max-pooling layer (pool window 2x2, stride 2). The image is already padded (1x1 zero-padding) and no further padding is applied later.

0	0	0	0	0	0
0	1	2	0	3	0
0	1	2	0	1	0
0	4	1	1	2	0
0	0	0	2	1	0
0	0	0	0	0	0

image

0	1	0
1	0	1
0	1	0

filter 1

0	1	1
0	1	0
1	1	0

filter 2

c. Determine the width, height and depth of the volume after max-pooling.

d. Compute the output of the max-pooling layer. Use all biases = 0.

e. Move the image one pixel to the left (padding column with zeros). How does the output of the convolution layer change? How many output units of the max-pooling layer have changed?

f. Is it possible to find a translation of the original image under which the value for each output unit of the max-pooling layer is invariant?

Solution:

a. For all positions of our pooling window, the maximum of $x(i)$ always lies inside the pooling window. Since we do max-pooling, the output of the pooling layer will always be the same, i.e. the maximum of $x(i)$. We see that in this case the result of the pooling is very stable and very invariant to shifts in x (and thus to translations of the image), because most of the units in the max-pooling layer will not change their output if we shift x.

b. In this case the output of the max-pooling changes for every position when sliding the pooling window over $x(i)$ because the input inside the pooling window always has its maximum at the highest index. We see that now the result is unstable and not invariant under shifts in x (and thus changing with translations of the image) because the output of the max-pooling units will all change if we shift x.

c. The volume after convolution is 4x4x2, since there are 4 different positions for the x and y direction at which we can place the 3x3 filters and we have 2 filters. After max-pooling with 2x2 filters and stride 2 we are left with a volume of 2x2x2.

d. We compute the output of the convolution with the filters and highlight in bold the maximum for each max-pooling filter:

3	3	5	1
7	4	4	**5**
2	**7**	**5**	3
4	3	2	4

output with filter 1

2	5	2	4
8	**9**	5	**7**
7	3	4	**6**
5	2	5	3

output with filter 2

e. Because of the equivariance property of the convolution we can move columns 2 to 4 to the left and compute the new rightmost column. Three out of eight output units of the max pooling layer change their value under this transformation.

3	**5**	1	3
4	4	**5**	1
7	5	3	2
3	2	**4**	1

output with filter 1

5	2	4	1
9	5	**7**	2
3	4	**6**	1
2	**5**	3	0

output with filter 2

f. Yes.

Exercise 4. Reverse-mode automatic differentiation

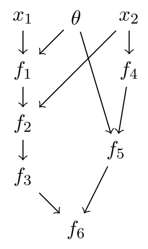
The backprop algorithm derived above is a special case of an algorithm called *Reverse-mode automatic differentation.* Modern deep learning software packages rely on this algorithm, as it allows flexible exploration of different architectures and cost functions without having to adapt the standard BackProp algorithm for each special case. To understand reverse-mode automatic differentation we look at the function

$$f(x_1, x_2, \theta) = \sin(\theta x_1 + x_2) + \theta x_2^2 \,. \tag{12}$$

Using the simple functions $f_1(x, y) = xy$, $f_2(x, y) = x + y$, $f_3(x) = \sin(x)$, $f_4(x) = x^2$, $f_5(x, y) = xy$ and $f_6(x, y) = x + y$, we can write the above function as

$$f(x_1, x_2, \theta) = f_6(f_3(f_2(f_1(x_1, \theta), x_2)), f_5(\theta, f_4(x_2)))\,, \tag{13}$$

that has the Abstract Syntax Tree (AST) or Computation Graph depicted on the right.



The reverse-mode Automatic Differentation algorithm proceeds now as follows:

AutoDiff

1. Determine all children of the variable(s) of interest. In the example, using θ, this includes f_1, f_2, f_3, f_5, f_6.
2. Find a reverse ancestral (backwards) schedule of nodes. All of the children of a node should be scheduled before the node itself. In the previous example with θ, the schedule could be $f_6, f_3, f_2, f_1, f_5, \theta$. For the full graph, this could be $f_6, f_3, f_2, f_1, x_1, f_5, \theta, f_4, x_2$.
3. Start with the first node n_1 in the reverse schedule and define $t_{n_1} = 1$, e.g. $t_{f_6} = 1$.
4. For the next node n in the reverse schedule, find the child nodes $\mathrm{ch}(n)$. Then define
$$t_n = \sum_{c \in \mathrm{ch}(n)} \frac{\partial f_c}{\partial f_n} t_c \,. \tag{14}$$
5. The total derivative of f with respect to node n is given by t_n.

a. Show that $t_\theta = \partial f / \partial \theta$ by following the steps of the algorithm, i.e. by computing $t_6, t_3, \ldots$, and comparing it to the result you get by differentiating Eq. **??** manually.

b. Draw the AST of a fully connected network with 2 hidden layers with biases and a single output, and convince yourself that the backpropagation algorithm is a special case of reverse-mode automatic differentation.

c. Draw the AST of a network with one 1D convolutional layer (with filter length 3 and stride 1), followed by one max-pooling layer (with $k = 2$), one dense layer, and a single output. For simplicity, assume only 1 filter is used in the convolutional layer.

Solution:

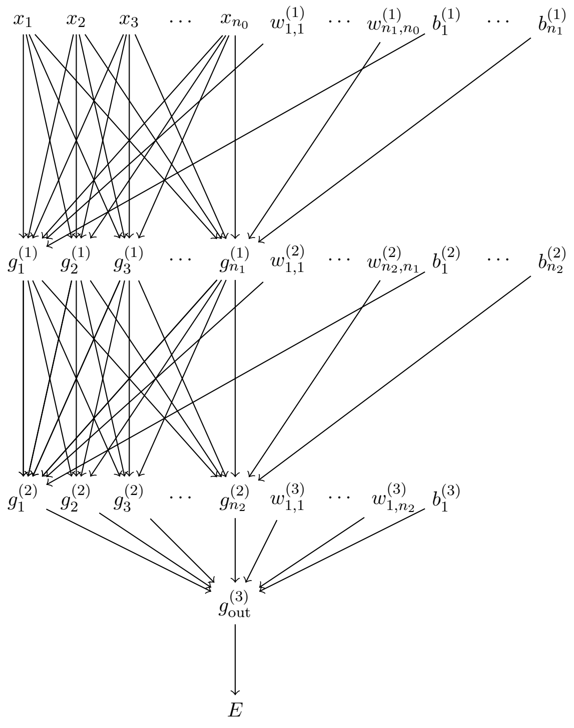
a. Steps (1) and (2) are completed above. For Steps (3) and (4), we follow the nodes in the reverse schedule in order:

$$\begin{aligned}
t_{f_6} &= 1 \\
t_{f_3} &= \frac{\partial f_6}{\partial f_3} t_{f_6} = \frac{\partial(\sin(\theta x_1 + x_2) + \theta x_2^2)}{\partial(\sin(\theta x_1 + x_2))}(1) = 1 \\
t_{f_2} &= \frac{\partial f_3}{\partial f_2} t_{f_3} = \frac{\partial(\sin(\theta x_1 + x_2))}{\partial(\theta x_1 + x_2)}(1) = \cos(\theta x_1 + x_2) \\
t_{f_1} &= \frac{\partial f_2}{\partial f_1} t_{f_2} = \frac{\partial(\theta x_1 + x_2)}{\partial(\theta x_1)} \cos(\theta x_1 + x_2) = \cos(\theta x_1 + x_2) \\
t_{f_5} &= \frac{\partial f_6}{\partial f_5} t_{f_6} = \frac{\partial(\sin(\theta x_1 + x_2) + \theta x_2^2)}{\partial(\theta x_2^2)}(1) = 1 \\
t_\theta &= \frac{\partial f_1}{\partial \theta} t_{f_1} + \frac{\partial f_5}{\partial \theta} t_{f_5} = \frac{\partial(\theta x_1 + x_2)}{\partial \theta} \cos(\theta x_1 + x_2) + \frac{\partial(\theta x_2^2)}{\partial \theta}(1) \\
&= x_1 \cos(\theta x_1 + x_2) + x_2^2
\end{aligned}$$

We see that t_θ is equal to the result we would expect from differentiating taking $t_\theta = \partial f / \partial \theta$ directly. For completeness, evaluating the rest of the graph gives

$$\begin{aligned}
t_{x_1} &= \frac{\partial f_1}{\partial x_1} t_{f_1} = \frac{\partial(\theta x_1)}{\partial x_1} \cos(\theta x_1 + x_2) = \theta \cos(\theta x_1 + x_2) \\
t_{f_4} &= \frac{\partial f_5}{\partial f_4} t_{f_5} = \frac{\partial(\theta x_2^2)}{\partial(x_2^2)}(1) = \theta \\
t_{x_2} &= \frac{\partial f_2}{\partial x_2} t_{f_2} + \frac{\partial f_4}{\partial x_2} t_{f_4} = \frac{\partial(\theta x_1 + x_2)}{\partial x_2} \cos(\theta x_1 + x_2) + \frac{\partial(x_2^2)}{\partial x_2}(\theta) \\
&= \cos(\theta x_1 + x_2) + 2\theta x_2
\end{aligned}$$

b. The AST is shown below.

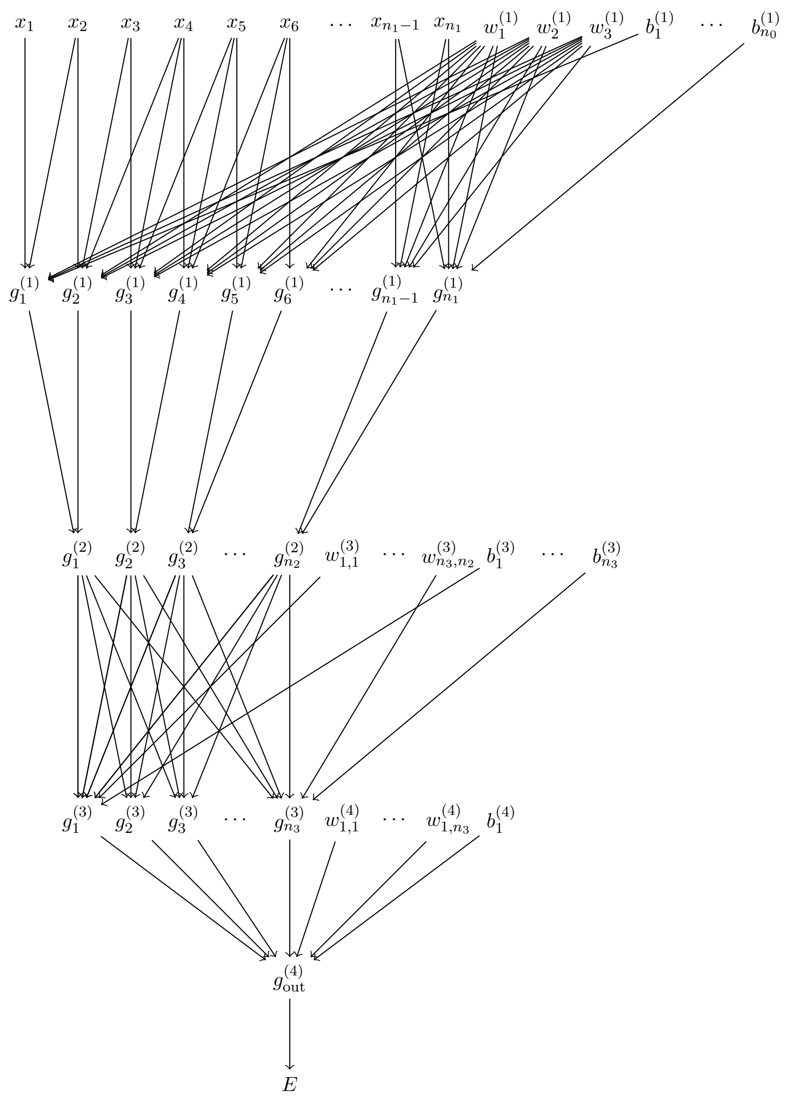

x_1 x_2 x_3 $\cdots$ x_{n_0} $w^{(1)}_{1,1}$ $\cdots$ $w^{(1)}_{n_1,n_0}$ $b^{(1)}_1$ $\cdots$ $b^{(1)}_{n_1}$
$g^{(1)}_1$ $g^{(1)}_2$ $g^{(1)}_3$ $\cdots$ $g^{(1)}_{n_1}$ $w^{(2)}_{1,1}$ $\cdots$ $w^{(2)}_{n_2,n_1}$ $b^{(2)}_1$ $\cdots$ $b^{(2)}_{n_2}$
$g^{(2)}_1$ $g^{(2)}_2$ $g^{(2)}_3$ $\cdots$ $g^{(2)}_{n_2}$ $w^{(3)}_{1,1}$ $\cdots$ $w^{(3)}_{1,n_2}$ $b^{(3)}_1$
$g^{(3)}_{\text{out}}$
E

We consider taking the derivative with respect to $w^{(1)}_{1,1}$ as an example, which in AutoDiff corresponds to finding $t_{w^{(1)}_{1,1}}$. The backwards schedule of nodes to find this derivative is $E, g^{(3)}_{\text{out}}, g^{(2)}_1 \dots g^{(2)}_{n_2}, g^{(1)}_1, w^{(0)}_{1,1}$ We assume quadratic error and denote the target as y. Where possible, we substitute the formula for $\delta^{(n)}_i$ from BackProp in the lecture slides. Starting with $t_E = 1$,

$$\begin{aligned}
t_{g^{(3)}_{\text{out}}} &= \frac{\partial E}{\partial g^{(3)}_{\text{out}}} t_E = -\left(y - g^{(3)}_{\text{out}}\right) \\
t_{g^{(2)}_i} &= \frac{\partial g^{(3)}_{\text{out}}}{\partial g^{(2)}_i} t_{g^{(3)}_{\text{out}}} = -w^{(3)}_{1i}\, g'^{(3)}_{\text{out}}\left[a^{(3)}_{\text{out}}\right]\left(y - g^{(3)}_{\text{out}}\right) \\
&= -w^{(3)}_{1i}\,\delta^{(3)}_{\text{out}} \\
t_{g^{(1)}_1} &= \sum_{i=1}^{n_2} \frac{\partial g^{(2)}_i}{\partial g^{(1)}_1} t_{g^{(2)}_i} = -\sum_{i=1}^{n_2} w^{(2)}_{i1}\, g'^{(2)}_i\left[a^{(2)}_i\right] w^{(3)}_{1i}\,\delta^{(3)}_{\text{out}} \\
&= -\sum_{i=1}^{n_2} w^{(2)}_{i1}\,\delta^{(2)}_i \\
t_{w^{(1)}_{11}} &= \frac{\partial g^{(1)}_1}{\partial w^{(0)}_{11}} t_{g^{(1)}_1} = -x_1 g'^{(1)}_1\left[a^{(1)}_1\right] \sum_{i=1}^{n_2} w^{(2)}_{i1}\,\delta^{(2)}_i \\
&= -x_1 \delta^{(1)}_1
\end{aligned}$$

which corresponds to the calculation of the weight update for $w^{(1)}_{11}$ in BackProp, after multiplying by $-\eta$ for stochastic gradient descent. By symmetry, the weight update has the same form for any weight in the first layer. Similarly, we can show that the AutoDiff update corresponds to BackProp in the other two layers as well (and, by induction, any layer in an arbitrarily deep network). We therefore conclude that BackProp is a special case of AutoDiff.

c. The AST is shown below. Note that, in fully connected layers, unit–to–unit connections are dense and weight–to–unit connections are sparse. Conversely, in convolutional layers, unit–to–unit connections are sparse and weight–to–unit connections are dense.



x_1 x_2 x_3 x_4 x_5 x_6 $\cdots$ x_{n_1-1} x_{n_1} $w_1^{(1)}$ $w_2^{(1)}$ $w_3^{(1)}$ $b_1^{(1)}$ $\cdots$ $b_{n_0}^{(1)}$
$g_1^{(1)}$ $g_2^{(1)}$ $g_3^{(1)}$ $g_4^{(1)}$ $g_5^{(1)}$ $g_6^{(1)}$ $\cdots$ $g_{n_1-1}^{(1)}$ $g_{n_1}^{(1)}$
$g_1^{(2)}$ $g_2^{(2)}$ $g_3^{(2)}$ $\cdots$ $g_{n_2}^{(2)}$ $w_{1,1}^{(3)}$ $\cdots$ $w_{n_3,n_2}^{(3)}$ $b_1^{(3)}$ $\cdots$ $b_{n_3}^{(3)}$
$g_1^{(3)}$ $g_2^{(3)}$ $g_3^{(3)}$ $\cdots$ $g_{n_3}^{(3)}$ $w_{1,1}^{(4)}$ $\cdots$ $w_{1,n_3}^{(4)}$ $b_1^{(4)}$
$g_{\text{out}}^{(4)}$
E