Artificial Neural Networks (Gerstner). Solutions for week 7

Convolutional neural networks

Exercise 1. Backprop in convnet (in class exercise)

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.

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

$$h_{ijk} = \sigma(a_{ijk}) \tag{2}$$

$$y_o = \sum_{ijk} v_{ijko} h_{ijk} \tag{3}$$

a. With loss $L = \frac{1}{2}\sum_o (\hat{y}_o - y_o)^2$, compute $\partial L/\partial w_{1111}$.

b. Compare your result to the one you obtain with a dense (fully-connected) layer, i.e. $a_k = b_k + \sum_x \sum_y \sum_c I_{xyc} w_{xyck}$, $h_k = \sigma(a_k)$ and $y_o = \sum_k v_{ok} h_k$.

For comparison reasons we could also choose an unusual indexing of the hidden neurons, i.e. $a_{ijk} = b_{ijk} + \sum_x \sum_y \sum_c I_{xyc} w_{xycijk}$ together with Equation 2 and Equation 3, (think of $2^3 = 8$ hidden neurons being indexed as $111, 112, 121, 122, \ldots$) and compute $\partial L/\partial w_{111111}$.

c. How does the result for the convolutional network change, if you introduce a max pooling layer after the convolutional layer?

Solution:

a.

$$\partial L/\partial w_{1111} = -\sum_o (\hat{y}_o - y_o) \sum_{ijk} v_{ijko} \sigma'(a_{ijk}) I_{ij1} \tag{4}$$

b.

$$\partial L/\partial w_{1111} = -\sum_o (\hat{y}_o - y_o) \sum_k v_{ok} \sigma'(a_k) I_{111} \tag{5}$$

$$\text{or} \quad \partial L/\partial w_{111111} = -\sum_o (\hat{y}_o - y_o) \sum_k v_{11ko} \sigma'(a_{11k}) I_{111} \tag{6}$$

c. The sum does then not run anymore over all i and j but only over those that were "winning", i.e. had the maximal value in the group if hidden activations seen by one max-pooling filter.

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

$$\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right) \times k\,. \tag{7}$$

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:

Exercise 3. Translation equivariance and translation invariance

On the right there is an image we would like to process with a convolutional network with one convolution layer with two 3x3 filters (depicted below the image), relu non-linearity, one 2x2 max-pooling layer (stride 2).

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

b. Compute the output of the max-pooling layer. Use stride 1 and all biases = 0.

c. 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?

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

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

Solution:

a. 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.

b. 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

c. 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

d. Yes.

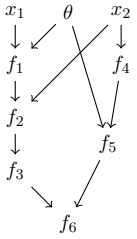
Exercise 4. Reverse-mode automatic differentiation

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

$$f(x_1, x_2, \theta) = \sin(\theta x_1 + x_2) + \theta x_2^2 \,. \tag{8}$$

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

$$f(x_1, x_2, \theta) = f_6(f_3(f_2(f_1(x_1, \theta), x_2)), f_5(\theta, f_4(x_2))) \,, \tag{9}$$



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
$$t_n = \sum_{c \in \mathrm{ch}(n)} \frac{\partial f_c}{\partial f_n} t_c \,. \tag{10}$$
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. 8 manually.

b. Draw the AST of a fully connected network with 2 hidden layers 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:

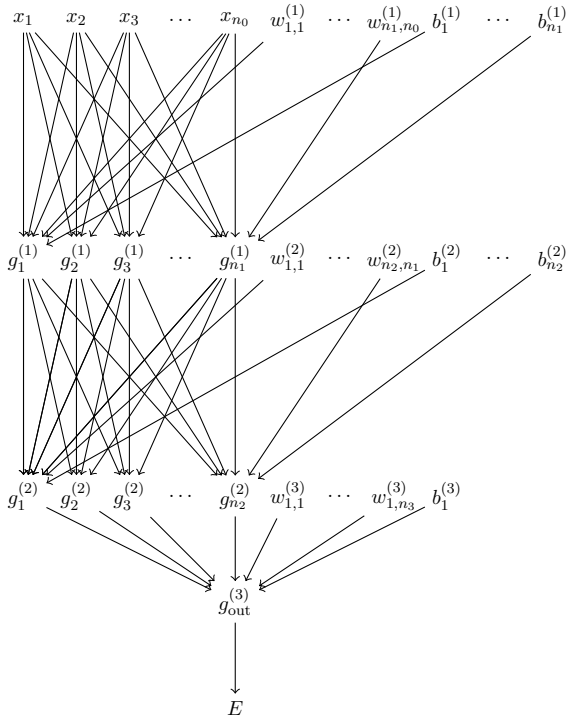
a. Steps (1) and (2) are completed above. For Steps (3) and (4), we follow the nodes in the reverse schedule in order:

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

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.



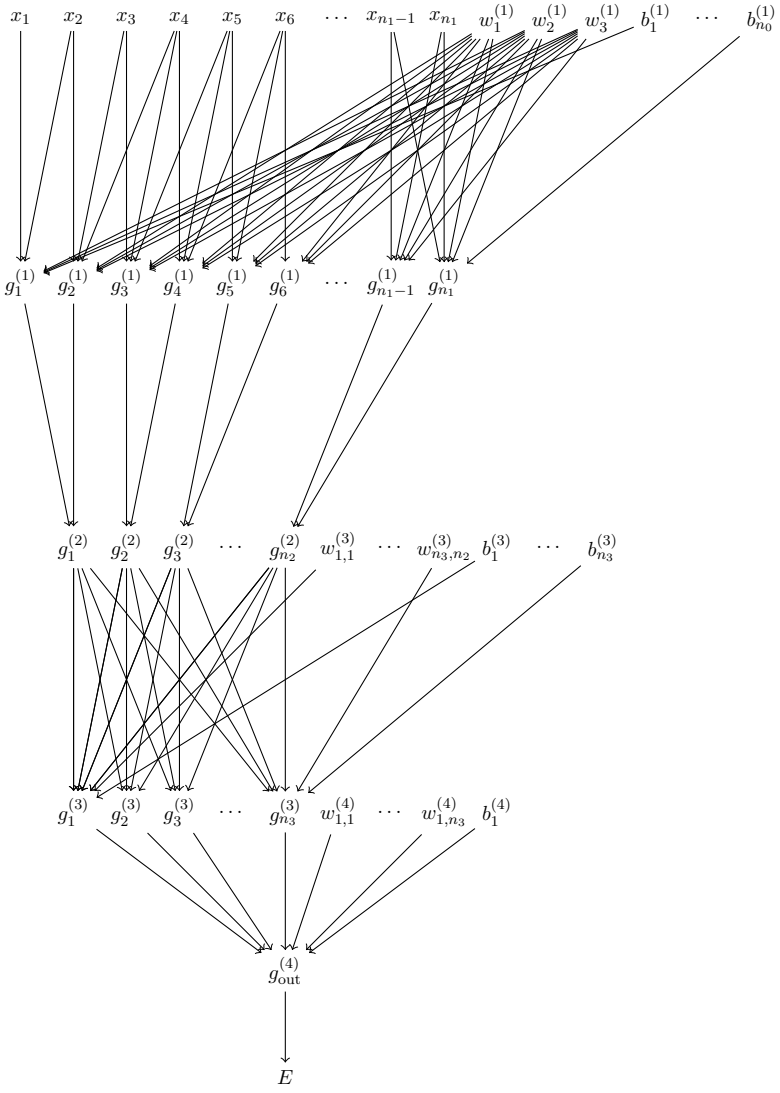
We consider taking the derivative with respect to $w_{1,1}^{(1)}$ as an example, which in AutoDiff corresponds to finding $t_{w_{1,1}^{(1)}}$. The backwards schedule of nodes to find this derivative is $E, g_{\text{out}}^{(3)}, g_1^{(2)} \dots g_{n_2}^{(2)}, g_1^{(1)}, w_{1,1}^{(0)}$

We assume quadratic error and denote the target as y. Where possible, we substitute the formula for $\delta_i^{(n)}$ from BackProp in the lecture slides. Starting with $t_E = 1$,

$$
\begin{aligned}
t_{g_{\text{out}}^{(3)}} &= \frac{\partial E}{\partial g_{\text{out}}^{(3)}} t_E = -\left(y - g_{\text{out}}^{(3)}\right) \\
t_{g_i^{(2)}} &= \frac{\partial g_{\text{out}}^{(3)}}{\partial g_i^{(2)}} t_{g_{\text{out}}^{(3)}} = -w_{1i}^{(3)} g_{\text{out}}'^{(3)}\left[a_{\text{out}}^{(3)}\right]\left(y - g_{\text{out}}^{(3)}\right) \\
&= -w_{1i}^{(3)} \delta_{\text{out}}^{(3)} \\
t_{g_1^{(1)}} &= \sum_{i=1}^{n_3} \frac{\partial g_i^{(2)}}{\partial g_1^{(1)}} t_{g_i^{(2)}} = -\sum_{i=1}^{n_3} w_{i1}^{(2)} g_i'^{(2)}\left[a_i^{(2)}\right] w_{1i}^{(3)} \delta_{\text{out}}^{(3)} \\
&= -\sum_{i=1}^{n_3} w_{i1}^{(2)} \delta_i^{(2)} \\
t_{w_{11}^{(1)}} &= \frac{\partial g_1^{(1)}}{\partial w_{11}^{(0)}} t_{g_1^{(1)}} = -x_1 g_1'^{(1)}\left[a_1^{(1)}\right] \sum_{i=1}^{n_3} w_{i1}^{(2)} \delta_i^{(2)} \\
&= -x_1 \delta_1^{(1)}
\end{aligned}
$$

which corresponds to the calculation of the weight update for $w_{11}^{(1)}$ 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