Exercises
Deep Learning
Fall 2020

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Professor Thomas Hofmann
Web http://www.da.inf.ethz.ch/teaching/2020/DeepLearning

Series Monday, Oct 12, 2020
(Deep Learning, Exercise series 4 - solutions)

Solution 1 (Backpropagation and Computational Graphs):

a) From the chain rule we have that

$$\frac{\partial l}{\partial h_1} = \frac{\partial l}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1} \left(= \frac{\partial l}{\partial h_2}\frac{\partial h_2}{\partial h_1}\right) \tag{1}$$

which can be represented as a computational graph as follows:

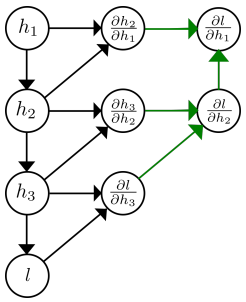

Figure 1: Gradient computations are shown in black, application of the chain rule in green.

b) Using the chain rule, we can derive the following expression for $\frac{\partial l}{\partial x}$ from the given DAG:

$$\frac{\partial l}{\partial x} = \left(\frac{\partial l}{\partial h_2}\frac{\partial h_2}{\partial h_1} + \frac{\partial l}{\partial h_3}\frac{\partial h_3}{\partial h_1}\right)\frac{\partial h_1}{\partial x}$$

We then compute each partial derivative to obtain:

$$\frac{\partial l}{\partial h_2} = h_3 = 2x + y, \quad \frac{\partial l}{\partial h_3} = h_2 = 8x^2, \quad \frac{\partial h_2}{\partial h_1} = 4h_1 = 8x, \quad \frac{\partial h_3}{\partial h_1} = 1, \quad \frac{\partial h_1}{\partial x} = 2$$

$$\frac{\partial l}{\partial x} = 48x^2 + 16xy$$

forward pass	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_1}$ forward differentiation	reverse differentiation
$\boldsymbol{z}_1 = \boldsymbol{w}_1 \boldsymbol{x}_1$	$\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{w}_1} = \boldsymbol{x}_1$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_1} = \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{w}_1}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_1}$, $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_2} = \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_2}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_2}$
$\boldsymbol{z}_2 = \boldsymbol{w}_2 \boldsymbol{x}_2$	$\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_1} = 0$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_1} = \frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_3}$, $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_2} = \frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{z}_2}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_3}$
$\boldsymbol{z}_3 = \boldsymbol{z}_1 + \boldsymbol{z}_2$	$\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{w}_1} = \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{w}_1} + \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{w}_1} = \boldsymbol{x}_1$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}_3} = \boldsymbol{w}_3$
$y = \boldsymbol{w}_3 \boldsymbol{z}_3$	$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_1} = \frac{\partial \boldsymbol{w}_3}{\partial \boldsymbol{w}_1}\boldsymbol{z}_3 + \frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{w}_1}\boldsymbol{w}_3 = \boldsymbol{x}_1 \boldsymbol{w}_3$	

Table 1: Differentiation modes

Solution 2 (Forward- vs. Backward differentiation):

a) $\boldsymbol{y} = \boldsymbol{w}_3(\boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_1\boldsymbol{x}_1)$

b) The graph is depicted in Figure 2.

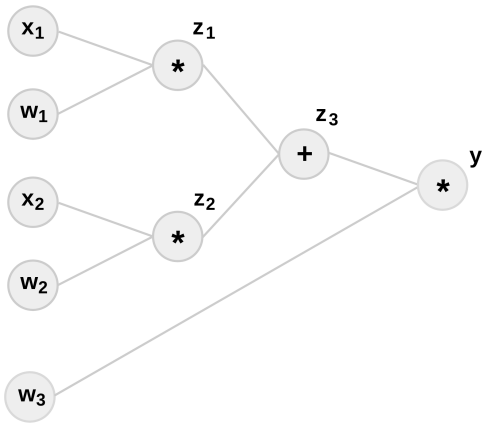

Figure 2: Computational graph

c) The computations are shown in Table 1.

Note how in forward mode the order of evaluation is top to bottom and all partial derivatives requiered in any give row are available from above.

In backward mode, the order of evaluation is in fact from bottom to top. Contrary to the forward mode, each lower term is calculated before a higher term and as a result the operands of each higher term are available for usage from below.

Note that the derivatives $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_2}$ and $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_3}$ are also readily available after one single backward pass!

d) One forward differentiation pass computes the derivatives of all outputs with respect to one parameter. Computing the full gradient thus amounts to $O(\#\text{parameters})$. One backward differentiation pass computes the derivative of one output with respect to all parameters. Computing the full gradient thus amounts to $O(\#\text{outputs})$. Since the number of outputs is usually way smaller than the number of trainable parameters in neural networks, backward mode differentiation (a.k.a. backpropagation) is the standard algorithm for computing gradients.

Solution 3 (Practical backpropagation for a feed-forward neural network):

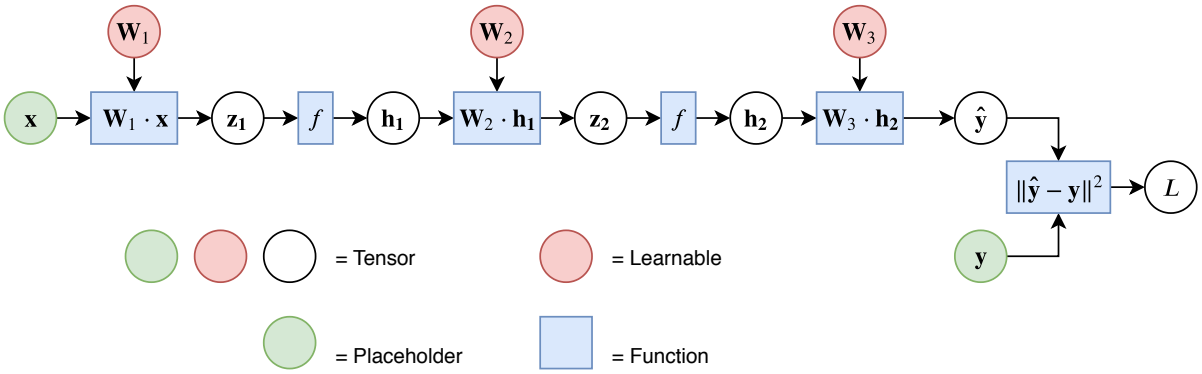

Figure 3: Computational graph for a feed forward network. Circles correspond to tensors, rectangles to functions.

1.

$$\frac{\partial L}{\partial \mathbf{W}_3} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_3}$$
$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_2}} \cdot \frac{\partial \mathbf{h_2}}{\partial \mathbf{z_2}} \cdot \frac{\partial \mathbf{z_2}}{\partial \mathbf{W}_2}$$
$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_2}} \cdot \frac{\partial \mathbf{h_2}}{\partial \mathbf{z_2}} \cdot \frac{\partial \mathbf{z_2}}{\partial \mathbf{h_1}} \cdot \frac{\partial \mathbf{h_1}}{\partial \mathbf{z_1}} \cdot \frac{\partial \mathbf{z_1}}{\partial \mathbf{W}_1}$$

Note that some parts of the chain overlap (highlighted in colors): a realistic implementation caches these results in the graph and reuses them.

2.

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{W}\mathbf{x}) = \mathbf{W} \tag{2}$$
$$\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}) = \text{diag}(f'(\mathbf{x})) \tag{3}$$
$$\frac{\partial}{\partial \mathbf{x}} \max(\mathbf{0}, \mathbf{x}) = \text{diag}(u(\mathbf{x})) \quad \text{(Heaviside step function)} \tag{4}$$
$$\frac{\partial}{\partial \hat{\mathbf{y}}} \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = 2(\hat{\mathbf{y}} - \mathbf{y}) \tag{5}$$

The Heaviside step function is simply 1 for $x > 0$, and 0 for $x \leq 0$.

3.

$$\frac{\partial L}{\partial \mathbf{W}_3} = 2(\mathbf{W}_3\mathbf{h_2} - \mathbf{y})\,\mathbf{h_2}^\intercal = \frac{\partial L}{\partial \hat{\mathbf{y}}}\mathbf{h_2}^\intercal$$

Therefore, for a generic linear layer $\mathbf{z} = \mathbf{W}\mathbf{x}$:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{z}}\mathbf{x}^\intercal$$

4.

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{z_2}}\mathbf{h_1}^\intercal$$
$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \mathbf{z_1}}\mathbf{x}^\intercal$$

5.

$$\frac{\partial L}{\partial \mathbf{z_1}} = \frac{\partial L}{\partial \mathbf{h_1}} \odot u(\mathbf{z_1})$$

Where $\odot$ denotes the Hadamard product, i.e. element-wise multiplication. For a general element-wise function $\mathbf{y} = f(\mathbf{x})$, we obtain:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \odot f'(\mathbf{x})$$

As a side note, some derivatives can be expressed as a function of the output (instead of the input). This is the case for ReLU (where $\frac{dy}{dx} = u(y)$) and the sigmoid (where $\frac{dy}{dx} = y(1-y)$). In these cases, the framework can save GPU memory by discarding the input tensor – for instance, in PyTorch this behavior can be enabled with the `inplace` flag (e.g. `nn.ReLU(inplace=True)`), which overwrites the original tensor when computing activations.

6.

$$\frac{\partial L_1}{\partial \mathbf{W}_1} = \frac{\partial L_1}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_2}} \cdot \frac{\partial \mathbf{h_2}}{\partial \mathbf{z_2}} \cdot \frac{\partial \mathbf{z_2}}{\partial \mathbf{h_1}} \cdot \frac{\partial \mathbf{h_1}}{\partial \mathbf{z_1}} \cdot \frac{\partial \mathbf{z_1}}{\partial \mathbf{W}_1}$$

$$\frac{\partial L_2}{\partial \mathbf{W}_1} = \frac{\partial L_2}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_2}} \cdot \frac{\partial \mathbf{h_2}}{\partial \mathbf{z_2}} \cdot \frac{\partial \mathbf{z_2}}{\partial \mathbf{h_1}} \cdot \frac{\partial \mathbf{h_1}}{\partial \mathbf{z_1}} \cdot \frac{\partial \mathbf{z_1}}{\partial \mathbf{W}_1}$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L_1}{\partial \mathbf{W}_1} + \frac{\partial L_2}{\partial \mathbf{W}_1} = \left(\frac{\partial L_1}{\partial \hat{\mathbf{y}}} + \frac{\partial L_2}{\partial \hat{\mathbf{y}}} \right) \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_2}} \cdot \frac{\partial \mathbf{h_2}}{\partial \mathbf{z_2}} \cdot \frac{\partial \mathbf{z_2}}{\partial \mathbf{h_1}} \cdot \frac{\partial \mathbf{h_1}}{\partial \mathbf{z_1}} \cdot \frac{\partial \mathbf{z_1}}{\partial \mathbf{W}_1}$$

In simple words, when we encounter a node with multiple outgoing edges – as it is the case with $\hat{\mathbf{y}}$ in this example – we first accumulate all the gradients in that node's *backward buffer*, and then we proceed with backpropagating through the parent nodes.