Deep Learning for Computer Vision
Assignment Sheet 3 - Due 24.12.17 23:59

Quick exercise Q&A: 19.12.17, 14:30
Lecture evaluation: 15.12.17, 11:20
Next exercise group: 9.1.18, 13:00

Exercise 3.1 Derivatives (20 points)
This exercise closes the (small) gap in the lecture in the backpropagation algorithm. Let $f : \mathbb{R}^N \to \mathbb{R}^M$ and $g : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$. Compute the derivative of $h : \mathbb{R}^N \to \mathbb{R}$, $h(\mathbf{x}) := g(f(\mathbf{x}), f(\mathbf{x}))$ in terms of the derivatives of f and g.

Exercise 3.2 Backpropagation (10 + 15 points)
The goal of this exercise is to see and understand how to do backpropagation on a minimalistic example of two-layered fully connected network. Note, in the illustration below, the input is on the bottom, output on the top. Pairs of numbers show (layer index, neuron index). Indices for ws show input neuron on bottom, connected output neuron on top.

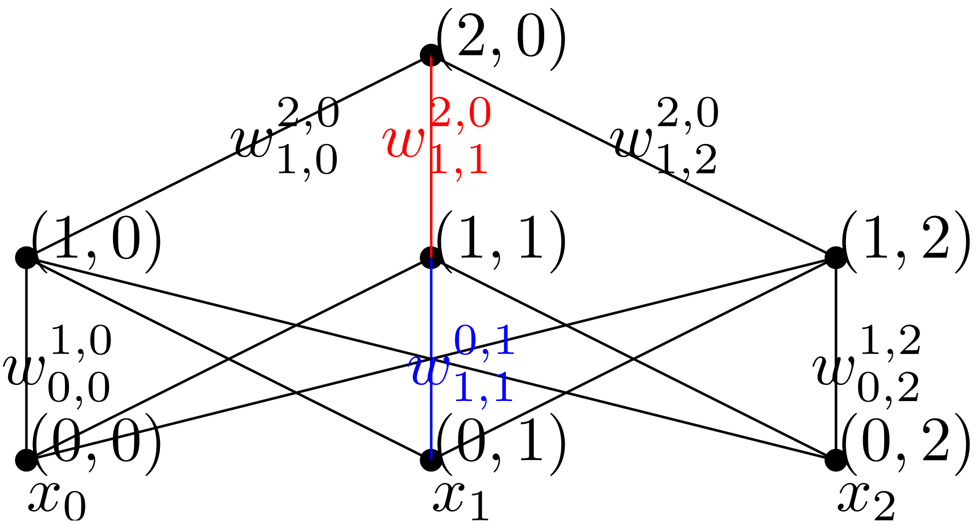

Figure 1: A two layer regression fully-connected architecture

The network calculates the function $f : \mathbb{R}^3 \to \mathbb{R}$ given by

$$f(\mathbf{x}; \mathbf{w}) = \sum_{i=0}^{2} w_{1,i}^{2,0} \left(\sum_{j=0}^{2} w_{0,j}^{1,i} x_j \right) \quad \text{and loss} \quad E(\mathbf{w}) = \frac{1}{L} \sum_{l=1}^{L} \| d_l - f(\mathbf{x}_l; \mathbf{w}) \|_2^2. \tag{1}$$

- Calculate the partial derivative of the loss with respect to the weight $w_{1,1}^{2,0}$ (depicted in red).

- Calculate the partial derivative of the loss with respect to the weight $w_{0,1}^{1,0}$ (depicted in blue).

Exercise 3.3: Activation functions and batch normalization (10 + 10 + 10 + 10 points)
The purpose of this exercise is to show how one can try different activation functions (PReLU) and batch normalization to get better results or faster convergence.

- Write your own Tensorflow leaky ReLU layer with a parameter $\alpha \geq 0$. Reminder, the leaky ReLU is defined as

$$\text{LeakyReLU}_{\alpha}(x) = \begin{cases} x & x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2}$$

- Train the architecture from Exercise 2.2, but replace the activations with leaky ReLU units (choose the same α for all layers for simplicity). Play around with different values of α and draw graphs with test and train loss/accuracy.
- Now, switch to parametric ReLU, i.e. train α as if it were another parameter. What's the optimal value, and how does it compare to the previous experiments?
- Add batch normalization (as described in the lecture) before the activations. How does it influence training and testing loss? How does it influence convergence?
- (Bonus) Add (non-trivial) convolutional layers to your network, and see how many you can stack while still being able to train the network. Use correct initializations, try e.g. weight regularization/dropout to prevent overfitting. Bonus points: min(20, number of working layers divided by 2). You might want to switch to a more interesting dataset, e.g. `https://www.cs.toronto.edu/~kriz/cifar.html`.

```python
# by Keith Randall 2013, posted on Stackexchange
n = 50

from turtle import *
speed("fastest")
left(90)
forward(3*n)
color("orange", "yellow")
begin_fill()
left(126)
for i in range(5):
    forward(n/5)
    right(144)
    forward(n/5)
    left(72)
end_fill()
right(126)

color("dark green")
backward(n*4.8)
def thing(d, s):
    if d <= 0: return
    forward(s)
    thing(d-1, s*.8)
    right(120)
    thing(d-3, s*.5)
    right(120)
    thing(d-3, s*.5)
    right(120)
    backward(s)
thing(15, n)
backward(n/2)

import time
time.sleep(60)
```

Happy holidays !