Machine Learning Homework Sheet 08

Deep Learning

1 Activation functions

Problem 1: Why do we use non-linear activation functions in neural networks? What purpose do they serve?

Without non-linear activation functions the system would remain linear, and adding layers doesn't change anything.

Problem 2: Consider a neural network where the hidden units use the sigmoid activation function. Show that there exists an equivalent network, which computes exactly the same function, but with hidden units using $\tanh(x)$ as activation functions.

Consider first a neural network with one input layer, *one* hidden layer, and one output layer. We have that

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

and

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

We also have

$$\tanh(x) + 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}} + \frac{e^x + e^{-x}}{e^x + e^{-x}} = \frac{2e^x}{e^x + e^{-x}} = \frac{2}{1 - e^{-2x}} = 2\sigma(2x)$$

So $\sigma(x) = \tanh(x/2)/2 + 1/2$. If we take our weights from the neural network with σ as our activation function, multiply all the input-hidden weights by $1/2$, then our new input x to the hidden units will be exactly $x/2$. If we then scale each of hidden-output weights by $1/2$, and then add $1/2$ to the bias hidden-output weights, we will get the desired result.

To extend it to multiple layers simply notice that we can treat each layer in isolation and consider the previous and next layers as if they are input and output layer respectively.

Problem 3: We already know that the derivative of the sigmoid activation function can be expressed in terms of the function value itself. Show that the derivative of the tanh activation function can also be expressed in terms of the function value itself. Why is this a useful property?

We have

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

thus

$$\tanh'(x) = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} = 1 - \tanh^2(x)$$

This is a useful property since it allows for efficient implementation of the gradient.

2 Numerical stability

Problem 4: In machine learning you quite often come across problems which contain the following quantity

$$y = \log \sum_{i=1}^{N} e^{x_i}$$

For example if we want to calculate the log-likelihood of a neural network with a softmax output we get this quantity due to the normalization constant. If you try to calculate it naively, you will quickly encounter underflows or overflows, depending on the scale of x_i. Despite working in log-space, the limited precision of computers is not enough and the result will be ∞ or $-\infty$.

To combat this issue we typically use the following identity:

$$y = \log \sum_{i=1}^{N} e^{x_i} = a + \log \sum_{i=1}^{N} e^{x_i - a}$$

for an arbitrary a. This means, you can shift the center of the exponential sum. A typical value is setting a to the maximum ($a = \max_i x_i$), which forces the greatest value to be zero and even if the other values would underflow, you get a reasonable result.

Your task is to show that the identity holds.

This is called the *log-sum-exp trick* and is often used in practice.

$$
\begin{aligned}
y &= \log \sum_{i=1}^{N} e^{x_i} \\
e^y &= \sum_{i=1}^{N} e^{x_i} \\
e^{-a} e^y &= e^{-a} \sum_{i=1}^{N} e^{x_i} \\
e^{y-a} &= \sum_{i=1}^{N} e^{-a} e^{x_i} \\
y - a &= \log \sum_{i=1}^{N} e^{x_i - a} \\
y &= a + \log \sum_{i=1}^{N} e^{x_i - a}
\end{aligned}
$$

Problem 5: Similar to the previous exercise we can compute the output of the softmax function $\pi_i = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}}$ in a numerically stable way by introducing an arbitrary a:

$$\frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}} = \frac{e^{x_i - a}}{\sum_{i=1}^{N} e^{x_i - a}}$$

often chosen $a = \max_i x_i$. Show that the above identity holds.

For some arbitrary constant C we have

$$\frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}} = \frac{C e^{x_i}}{C \sum_{i=1}^{N} e^{x_i}} = \frac{e^{x_i + log(C)}}{\sum_{i=1}^{N} e^{x_i + log(C)}}$$

Since C is just an arbitrary constant, we can replace $log(C) = -a$ and get $\frac{e^{x_i - a}}{\sum_{i=1}^{N} e^{x_i - a}}$.

Problem 6: Let the logits (the values before applying sigmoid) of a single training example be x and the corresponding label be y. The sigmoid logistic loss (i.e. binary cross-entropy) for this example is then:

$$-(y \log(\sigma(x)) + (1 - y) \log(1 - \sigma(x)))$$

To ensure stability and avoid overflow, usually implementations use this equivalent formulation:

$$\max(x, 0) - x \cdot y + \log(1 + e^{-\text{abs}(x)})$$

Upload a single PDF file with your solution to Moodle by 16.12.2018, 23:59 CET. We recommend to typeset your solution (using LaTeX or Word), but handwritten solutions are also accepted.
If your handwritten solution is illegible, it won't be graded and you waive your right to dispute that.

You task is to show that the equivalence holds.

$$
\begin{aligned}
& - y \cdot \log(\sigma(x)) - (1-y)\log(1-\sigma(x)) \\
=& - y \cdot \log(1/(1+e^{-x})) - (1-y)\log(e^{-x}/(1+e^{-x})) \\
=& y \cdot \log(1+e^{-x}) + (1-y)\cdot(-\log(e^{-x}) + \log(1+e^{-x})) \\
=& y \cdot \log(1+e^{-x}) + (1-y)\cdot(x + \log(1+e^{-x})) \\
=& (1-y)\cdot x + \log(1+e^{-x}) \\
=& x - x \cdot y + \log(1+e^{-x})
\end{aligned}
$$

For $x < 0$, to avoid overflow in e^{-x}, we reformulate the above:

$$
\begin{aligned}
& x - x \cdot y + \log(1+e^{-x}) \\
=& \log(e^{x}) - x \cdot y + \log(1+e^{-x}) \\
=& - x \cdot y + \log(1+e^{x})
\end{aligned}
$$

Hence, to ensure stability and avoid overflow, the implementation uses this equivalent formulation

$$\max(x, 0) - x \cdot y + \log(1 + e^{-\text{abs}(x)})$$

Upload a single PDF file with your solution to Moodle by 16.12.2018, 23:59 CET. We recommend to typeset your solution (using LaTeX or Word), but handwritten solutions are also accepted.
If your handwritten solution is illegible, it won't be graded and you waive your right to dispute that.