Deep Learning: Lecture 4

Alexander Schönhuth

UU

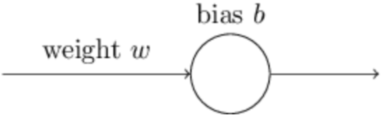
October 30, 2019

Preventing Slow Learning

Slow Learning II

Sigmoid Function

Drawback

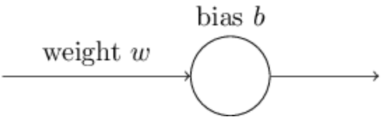


Cost function:

$$C = \frac{(y-a)^2}{2} = \frac{(y-\sigma(z))^2}{2} = \frac{(y-\sigma(wx+b))^2}{2} \tag{1}$$

Sigmoid Function

Drawback



Cost function:

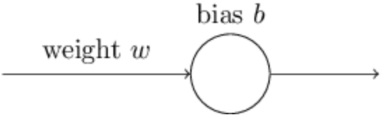
$$C = \frac{(y-a)^2}{2} = \frac{(y-\sigma(z))^2}{2} = \frac{(y-\sigma(wx+b))^2}{2} \tag{1}$$

Training input $x = 1$, desired output $y = 0$:

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z) \quad \text{and} \quad \frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z) \tag{2}$$

Sigmoid Function

Drawback



Cost function:

$$C = \frac{(y-a)^2}{2} = \frac{(y-\sigma(z))^2}{2} = \frac{(y-\sigma(wx+b))^2}{2} \tag{1}$$

Training input $x = 1$, desired output $y = 0$:

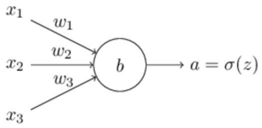
$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z) \quad \text{and} \quad \frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z) \tag{2}$$

When $\sigma(z) \approx 0$ or $\sigma(z) \approx 1$, we have $\sigma'(z) \approx 0$ ☞ learning slows down

Sigmoid Neurons

Remedy: Alternative cost function

Consider



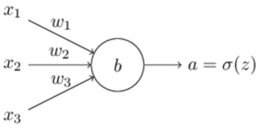
where $z = \sum_j w_j x_j + b$.

Issue: Sigmoid activation and quadratic cost make unfortunate combination

Sigmoid Neurons

Remedy: Alternative cost function

Consider



where $z = \sum_j w_j x_j + b$.

Issue: Sigmoid activation and quadratic cost make unfortunate combination

Solution: Use alternative cost function: *cross entropy*:

$$C = -\frac{1}{m} \sum_x [y(x) \log a(x) + (1 - y(x)) \log(1 - a(x)) \tag{3}$$

where x runs over all m training examples.

Cross Entropy

Cross entropy:

$$C = -\frac{1}{m}\sum_x [y(x)\log a(x) + (1-y(x))\log(1-a(x))] \tag{4}$$

where x runs over all m training examples.

Remarks

- $C \geq 0$: log's are negative because of $a(x) = \sigma(z) \in [0,1]$, minus sign in front
- C close to zero if $y(x) \approx a(x)$ (considering $y(x) \in \{0,1\}$)
- If $y(x) \in [0,1]$, cross entropy C is minimal iff $a(x) = y(x)$.

Cross Entropy

Substituting $a = \sigma(z)$ into (3), we obtain

$$\begin{aligned} \frac{\partial C}{\partial w_j} &= -\frac{1}{m}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y(x))}{1-\sigma(z)}\right)\frac{\partial \sigma}{\partial w_j} \\ &= -\frac{1}{m}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y(x))}{1-\sigma(z)}\right)\sigma'(z)x_j \end{aligned} \tag{5}$$

Further simplifying yields:

$$\frac{\partial C}{\partial w_j} = \frac{1}{m}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z)-y) \tag{6}$$

Cross Entropy

Realizing that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, we finally obtain

$$\frac{\partial C}{\partial w_j} = \frac{1}{m} \sum_x x_j(\sigma(z) - y(x)) \tag{7}$$

Similarly

$$\frac{\partial C}{\partial b} = \frac{1}{m} \sum_x (\sigma(z) - y(x)) \tag{8}$$

Fast Learning

Fast Learning II

Cross Entropy

Multineuron output

Cross entropy also works for more than one output neuron. Let $y(x) = (y_1(x), ..., y_d(x))$ be the true labels, while $a^L(x) = (a^L_1(x), ..., a^L_d(x))$ are the actual output values.

Then multi output neuron cross entropy is defined by

$$C = -\frac{1}{m}\sum_x \sum_j [y_j(x) \log a^L_j(x) + (1 - y_j(x)) \log(1 - a^L_j(x))] \quad (9)$$

where $j = 1, ..., d$.

Softmax

Consider the case of J outputs $a_j^L, j = 1, ..., J$. Let (as usual)

$$z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L \tag{10}$$

be the input to the corresponding J neurons making the output layer.

Then the *softmax activation* is defined by

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \tag{11}$$

SOFTMAX

Note that

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1 \tag{12}$$

- All outputs are positive
- A softmax layer can be thought of as a probability distribution over the J different possible outputs.
- *Observation*: Softmax output values depend on the inputs to all output neurons, and not only on the particular one that generates the output.

Softmax

Cost Function

Let $(x, y(x))$ be one training example, where $y(x) \in \{1, ..., J\}$. Then the *log-likelihood cost* is defined to be

$$-\log a^L_{y(x)} \tag{13}$$

Let here $y_j = 1$ iff $j = y(x)$ and $y_j = 0$ iff $j \neq y(x)$ (in abuse of earlier notation). Then we obtain

$$\frac{\partial C}{\partial b^L_j} = a^L_j - y_j \tag{14}$$

$$\frac{\partial C}{\partial w^L_{jk}} = a^{L-1}_k (a^L_j - y_j) \tag{15}$$

Note that (14) and (15) are, apart from not summing over many training examples here, identical to (7) and (8).

So, what is better, sigmoid + cross-entropy, or softmax + loglikelihood? It depends, in fact both can lead to good results in many cases.

Alternative Activation Functions

Tangens Hyperbolicus

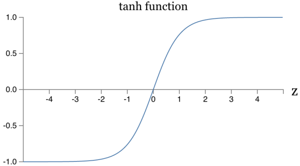
Tangens hyperbolicus is defined by

$$\tanh(z) := \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{16}$$

It holds that

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2} \tag{17}$$

so tanh turns out to be a scaled version of the sigmoid function σ.



Tangens hyperbolicus is a scaled version of a sigmoid

Tangens Hyperbolicus

Motivation

Remember that

$$\frac{\partial C}{\partial w_{jk}^{l+1}} = a_k^l \delta_j^{l+1} \tag{18}$$

When using sigmoid neurons, $a_k^l \in [0, 1]$, hence non-negative, while for tangens hyperbolicus $a_k^l \in [-1, 1]$, so possibly also negative. Hence, if $\delta_j^{l+1} > 0$ (or $\delta_j^{l+1} < 0$) then

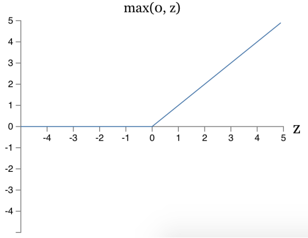
- all weights w_{jk}^{l+1} will decrease (or increase) for sigmoid neurons
- some weights will decrease, and some weights will increase (or vice versa) for tanh neurons

The latter case can be advantageous.

However, empirically, tanh were not found to have decisive advantages over sigmoid neurons.

Alternative Activation Functions

Recitified Linear Units



Rectifying function

The rectified linear function with input z is defined by

$$\max(0, z) \tag{19}$$

so a *rectified linear neuron* with input $\mathbf{x}$, weight vector $\mathbf{w}$ and bias b is defined by

$$\max(0, \mathbf{w}\mathbf{x} + b) \tag{20}$$

Rectified Linear Neurons

Properties

- No theoretical deep understanding available
- Rectified linear neurons do not saturate on positive input
  ☞ no learning slowdown
- When input is negative, rectified linear neurons stop learning entirely!
- Empirically, rectified linear neurons have been proven to be of great use in image recognition

Rectified Linear Neurons

Literature

- "What is the best multi-stage architecture for object recognition?", `http://yann.lecun.com/exdb/publis/pdf/jarrett-iccv-09.pdf`
- "Deep sparse rectifier neural networks", `http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf`
- "ImageNet classification with deep convolutional neural networks", `https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional pdf`
- Papers provide interesting details about choice of cost functions, setting up the output layer, and regularization.

Thanks for your attention