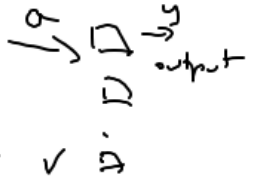
$w = w_{j_0,i_0}$

$$\frac{\partial L}{\partial w} = \sum_k (t_k - y_k)(-1)\, \varphi'\Big(\sum_j v_{kj}\, a_j\Big)\, v_{kj_0}\, \varphi'\Big(\sum_i w_{j_0,i}\, x_i\Big)\cdot x_{i_0}$$

$\sum_j$: net output of the 2nd layer

$\sum_i$: net output of the 1st layer



φ – activation function

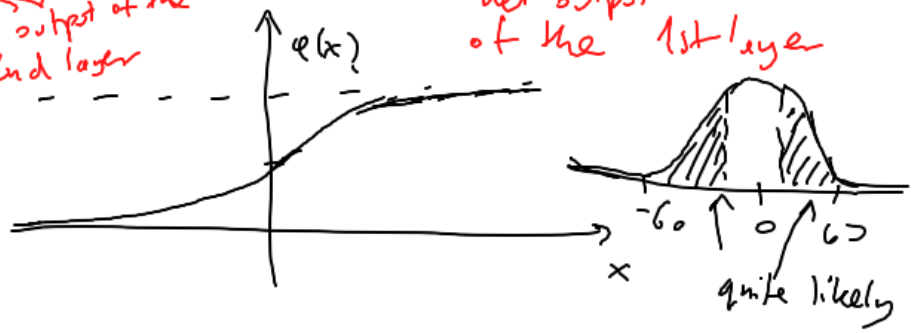
$$a = \varphi\Big(\sum_i w_{j_0,i}\, x_i\Big)$$

output of the 1st layer

new w = old w $- \varepsilon \cdot \frac{\partial L}{\partial w}$

e.g. $\varphi(x) = \frac{1}{1+e^{-x}}$

$$\varphi'(x) = \varphi(x)\cdot(1-\varphi(x))$$



if x is away from zero, then φ' is small (very small e.g. for $|x| > 10$)

It's important to keep net outputs close to 0

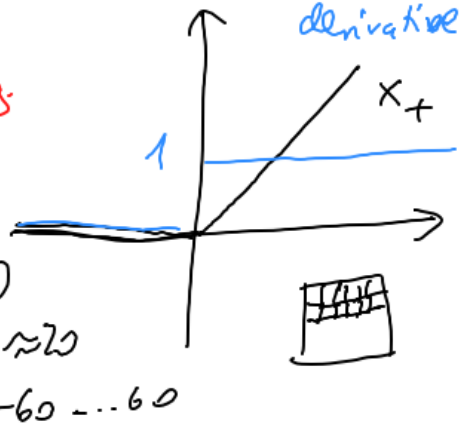
- normalize the input
- initialize the weights in the right way

e.g. if $w \sim N(0, \frac{1}{\sqrt{785}})$ (785 inputs x)

Let's suppose half of x's are 1, half are 0

$\sum_i w_{j_0 i}\, x_i \sim N(0, \sqrt{392}^2)$

we want e.g. $\frac{1}{2}$ here



Ridge regression (1st assignment):
we took all the training data and calculated the loss function based on that

This method is too expensive with the use of neural networks.

- The simplest way: online learning: (example: 2nd assignment, images — classification)
  - take some image, compute the output of the network, do the backprop. and weights adjustment
  - repeat for all images in the dataset
- repeat some number of times (called epochs)
- it's good to shuffle the training set between epochs (also before the first one)

epochs: 1 2 ... 375
epoch 2: 369 17 ... 1

Batch learning: take some number B of images
and take the average gradient for the adjustment

$$\text{new } w = \text{old } w - c \cdot \left(\frac{\partial L^{(1)}}{\partial w} + \dots + \frac{\partial L^{(B)}}{\partial w} \right) \cdot \frac{1}{B}$$

$\frac{\partial L^{(1)}}{\partial w}$: derivative of the loss f. w/r to the weight w calculated using the image no 1

$\frac{1}{B}$: one can incorporate it into c

regularization of weights:

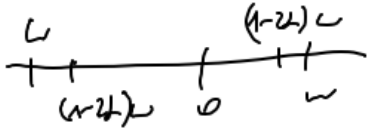
take: (regularized loss function) = (usual loss function) $+ \alpha \sum_{L} w^2$

w: weight (but not for biases)

$\alpha \geq 0$ hyper parameter small

$$\text{new } w = \text{old } w - c\left(\frac{\partial L}{\partial w} + 2\alpha w \right)$$

or $\text{new } w = \text{old } w \cdot (1 - 2\alpha) - c \frac{\partial L}{\partial w}$



Some history:

'40 first models

'50 first implementation (hardware)

'80 first applications – convolutional neural networks

2000 ...

Thm. (Cybenko)

Let σ – sigmoidal, cont. function
$\hookrightarrow \sigma(-\infty)=0,\ \sigma(\infty)=1,\ \sigma\nearrow$

Then the finite sums of the form

$$G(x) = \sum_{j=1}^{m} \alpha_j\ \sigma(y_j^T \cdot x + \theta_j)$$

(y_j: weights, x: input, θ_j: bias)

are dense in $(C([0,1]^n), \|\cdot\|_{sup})$.

$, x \in [0,1]^n$

In other words, if $F: [0,1]^n \to \mathbb{R}$ continuous then for any $\varepsilon > 0$ one can find G as above so that $|F(x) - G(x)| < \varepsilon \quad \forall x \in [0,1]^n$