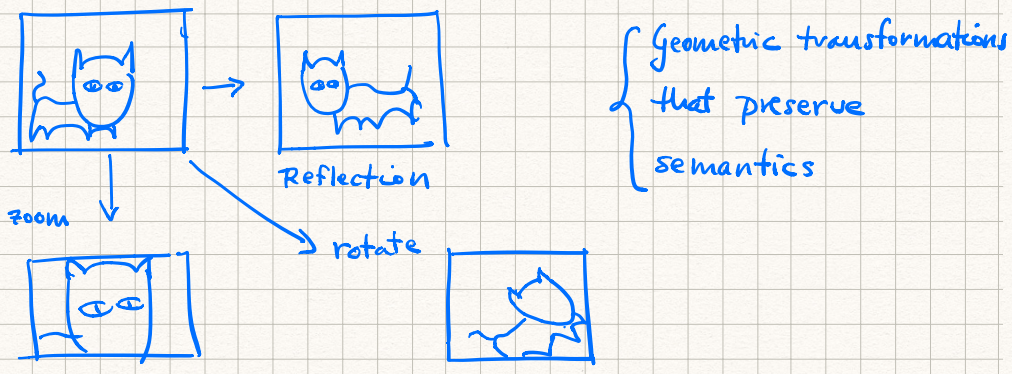
Training neural networks

The data set

Data augmentation

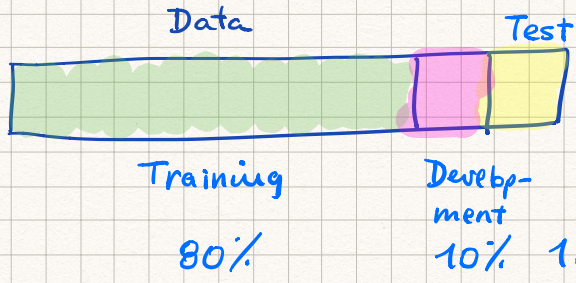
Given a piece of data → replicate



Finding hyperparameters

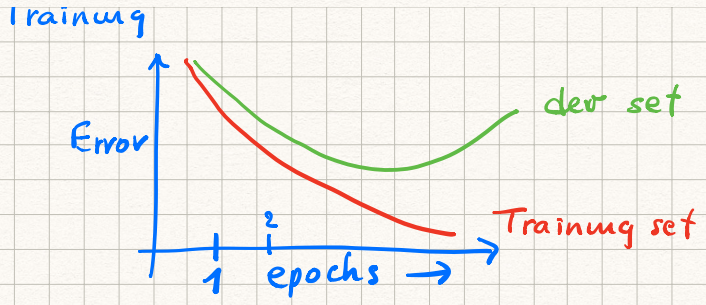
The learning rate γ is a hyperparameter

there are others

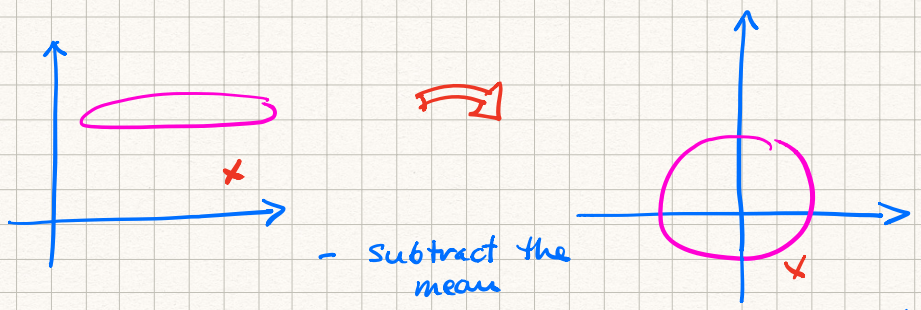


But if we have a lot of data, it can be

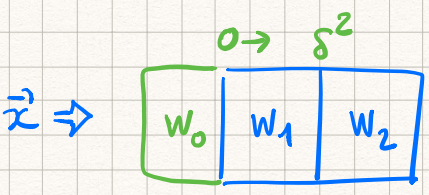
98%, 1%, 1%



Normalizing the data set



- subtract the mean
- divide by the standard deviation



REGULARIZATION

Like in ridge regression, we add a penalty to the error proportional to the size of the weights

$|x|$

m

$$E_{new} = E + \frac{\lambda}{2}\sum_{i=1} \omega_i^2$$

$$\frac{\partial E_{new}}{\partial \omega_i} = \frac{\partial E}{\partial \omega_i} + \lambda \omega_i$$

Correction

$$\Delta \omega_i = -\gamma \left(\frac{\partial E}{\partial \omega_i} + \lambda \omega_i \right)$$

weight decay

{ the adjustment tries to "pull down" the absolute size of the weight

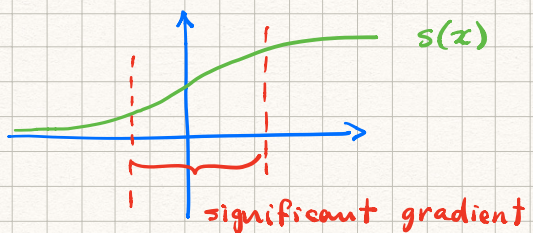
We can use a regularization parameter λ

$$\Delta \omega_i = -\gamma \frac{\partial E}{\partial \omega_i} - \lambda' \omega_i$$

γ: learning rate

λ': weight decay constant

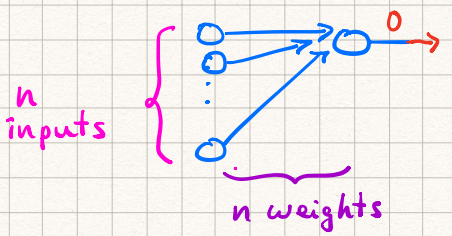
Small weights are better



Initialization

→ Random weights

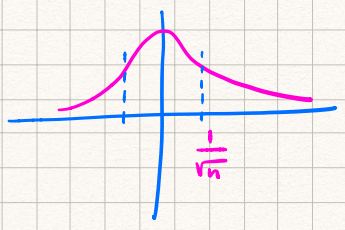
→ Small weights in terms of the inner product



small $\sum x_i w_i$

Xavier initialization

Pick the weights from a Gaussian distribution with zero mean and variance $1/n$



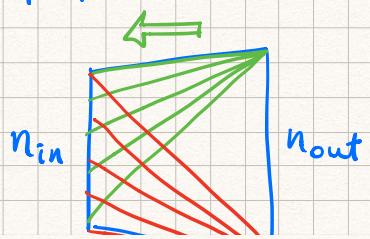
Then $\text{var}(\sum w_i x_i) = n \cdot \text{var}(w_i) \cdot \text{var}(x_i)$

$= \text{var}(x_i)$

$\text{Var}(w_i) = \frac{1}{n}$

Glorot & Bengio

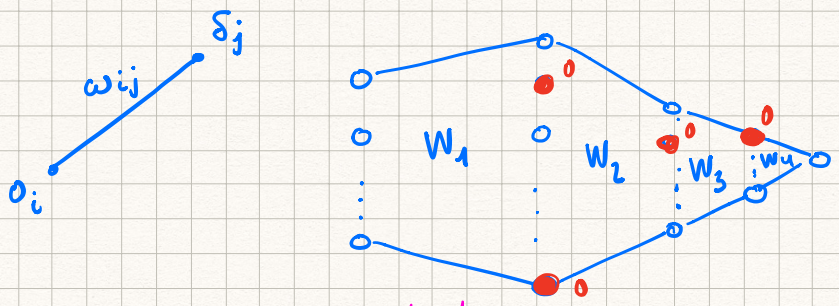
Consider backprop



$$Var(w_i) = \frac{2}{n_{in} + n_{out}}$$

Initialize in this layer so

Dropout regularization



repeat {
- shut-down randomly a certain percentage of the units (20% for example)
- $\Rightarrow$ train for a minibatch
}

- The network learns to be robust
- The weights are "spread" through the network

Activation functions

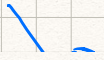
RELU rectifying linear unit

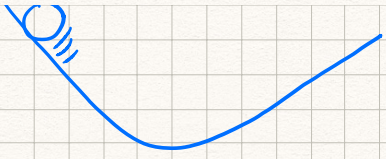
derivative is one

Leaky RELU

- derivative 1 for $x \geq 0$
- " 0.01 for $x < 0$

Backprop with momentum





if a ball is rolling down, it gains "momentum"

In gradient descent we can "remember" the previous correction in step $i-1$ and then:

$$\Delta \vec{w}^{(i)} = \beta \Delta w^{(i-1)} - (1-\beta) \vec{\nabla} E)$$

$\beta \Delta w^{(i-1)}$: previous correction; $\vec{\nabla} E$: gradient

β is a new hyperparameter

for example $\beta = 0.9$

The result is an exponentially weighted average

$$\Delta \vec{w}^{(i)} = \beta \Delta w^{(i-1)} + \cdot (1-\beta) \vec{\nabla} E^{(i)}$$

$$= \beta^2 \Delta w^{(i-2)} + \beta (1-\beta) \nabla E^{(i-1)} + (1-\beta) \vec{\nabla} E^{i}$$

Learning rate decay

Start with γ_0

Let γ decay

$$\gamma = \frac{\gamma_0}{1 + \text{decay-rate} * \text{epoch}}$$

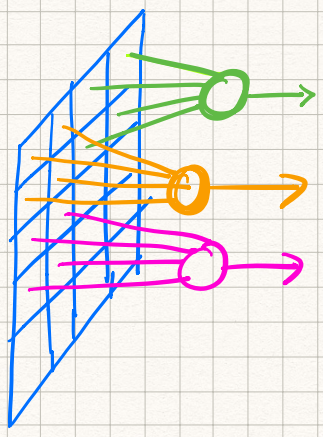
epoch: one pass through data

$$\gamma = \gamma_0 \cdot 0.95^{epoch}$$

$$\gamma = \frac{k}{\sqrt{epoch}} \cdot \gamma_0$$

manual decay

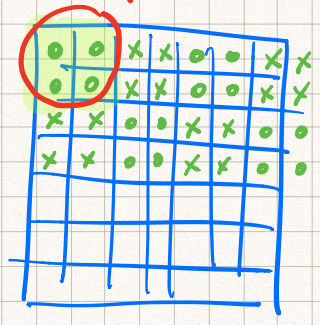
staircase

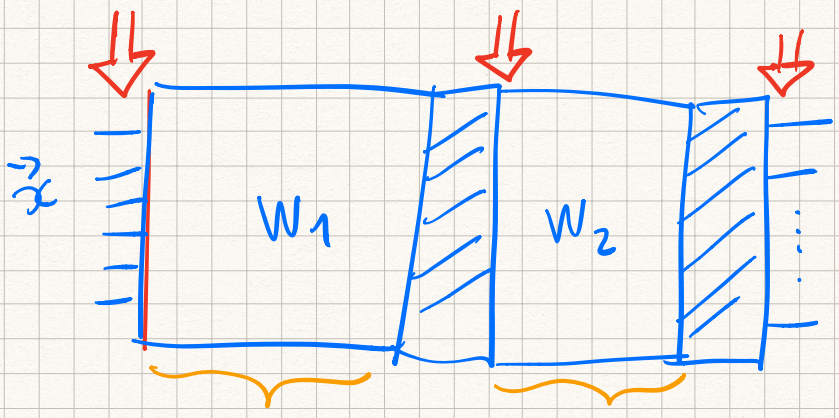


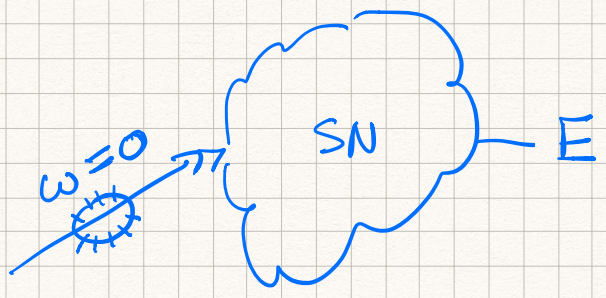
Convolutional NN

CNN

Convolutional Kernel







$$\Delta\omega = -\gamma \frac{\partial E}{\partial \omega} \qquad \frac{\partial E}{\partial \omega}$$