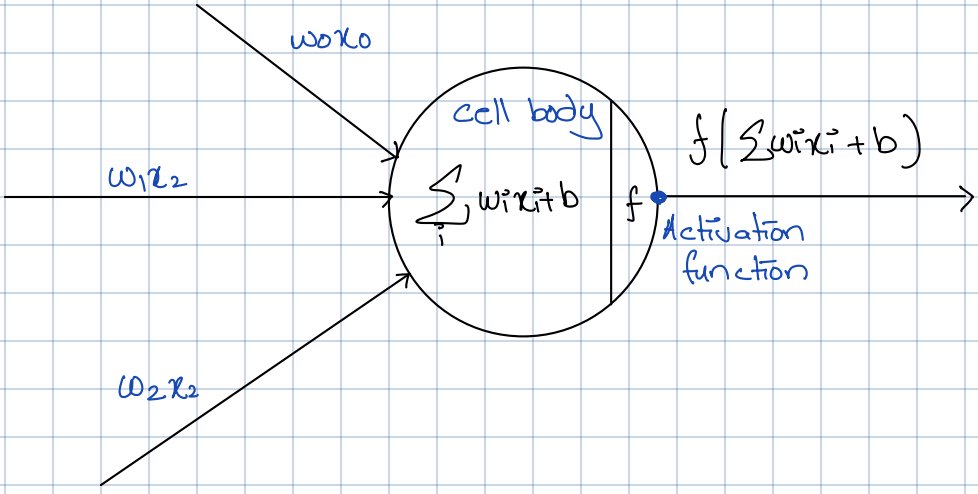
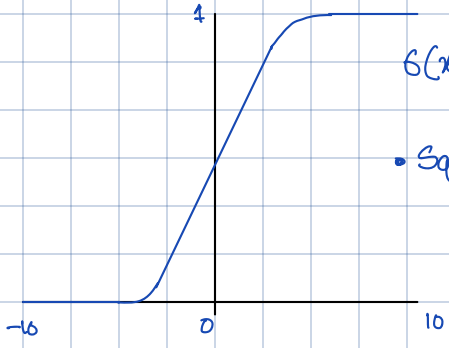
Mini-batch SGD:

Loop:
1) Sample a batch of data
2) Forward prop it through the graph (network), get loss
3) Backprop to calculate gradients
4) Update the parameters using the gradients

Activation Functions:



Sigmoid:

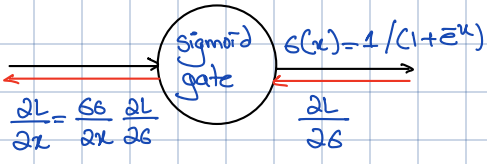


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Squashes numbers to range $[0, 1]$

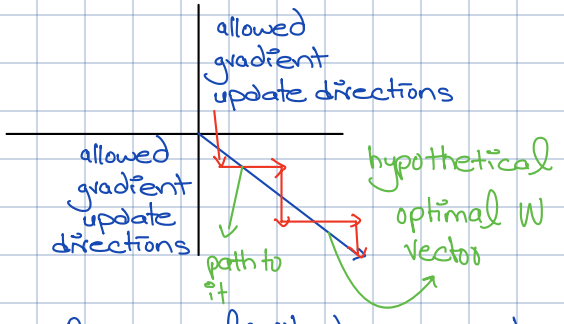
Problems:

1. Can kill off the gradients when flat



When $x = -10$, $g \approx 0$, A very small gradient flowing back
When $x = 0$, $g \approx 0.5$, Resonable gradient
When $x = 10$, $g \approx 0$, gradient is small "kill off"

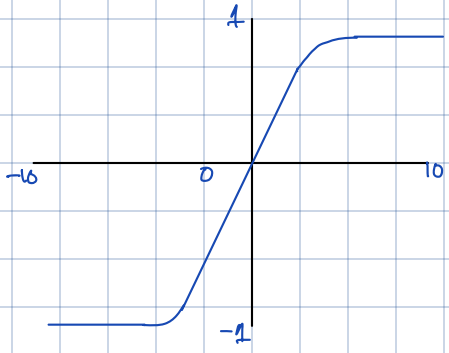
2. Sigmoid outputs are not zero centered
consider all x is +ev,
dw is +ev OR -ev



- Because limited movement directions, we cannot take direct path
- That's why, we want zero-mean data

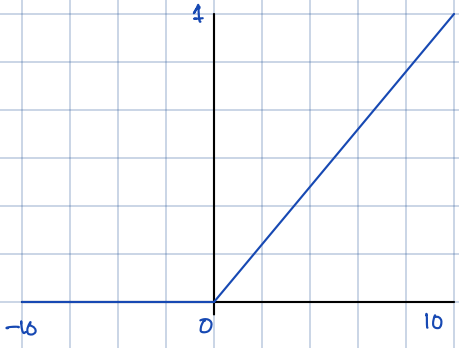
3. exp() is a bit computationally expensive

tanh(x):



- Squashes numbers to range [-1, 1]
- Zero centered
- Kills gradients when it is flat

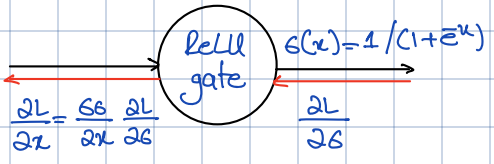
ReLu:



- $f(x) = \max(0, x)$
- Does not become flat (in + region)
- Converges faster

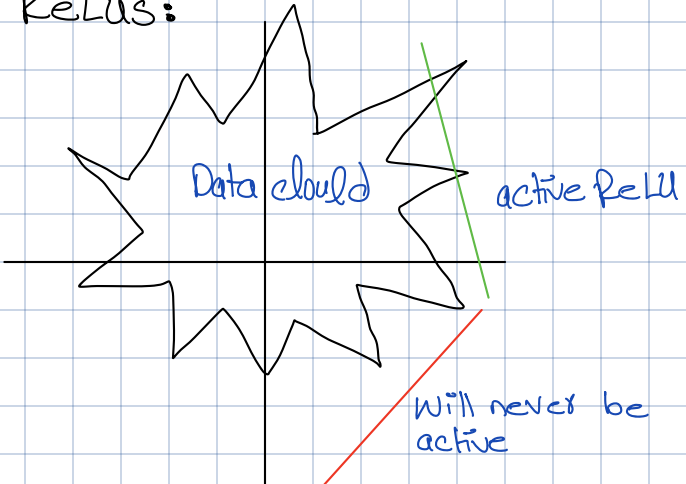
Problem:

- Not zero centred
- Flat in -ve



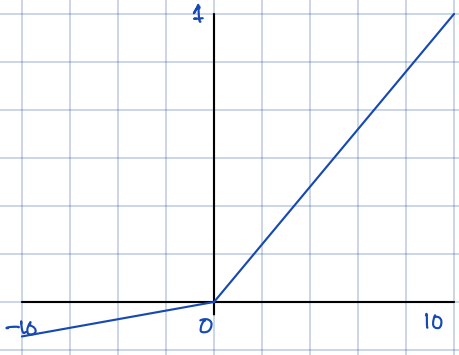
When $x = -10$, $g = 0$, A very small gradient flowing back
When $x = 0$, $g = 0$, "
When $x = 10$, $g \approx > 0$, Reasonable

Dead ReLus:



- Bad initializations, weights off data cloud
- α is high

Leaky ReLU:



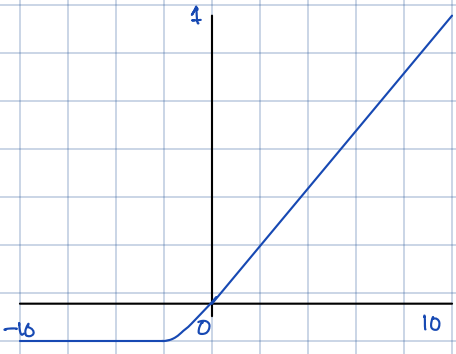
- $f(x) = \max(0.01x, x)$

Parametric ReLU (PReLU)

- $f(x) = \max(\alpha x, x)$

↓ negative slope determined by α

Exponential Linear units (ELU)



- $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$

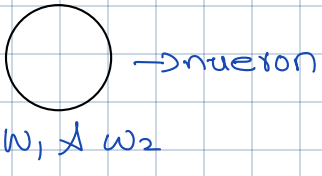
- All benefits of ReLU
- Closer to zero mean
- Negative saturation like leaky ReLU

Maxout "Neuron"

- Does have the basic form: Dot product → non-linearity
- Generalizes ReLU & Leaky ReLU
- Linear Regime, Does not become flat, Does not die

$$\max(w_1^T x + b, w_2^T x + b)$$

- Doubles the no. of params

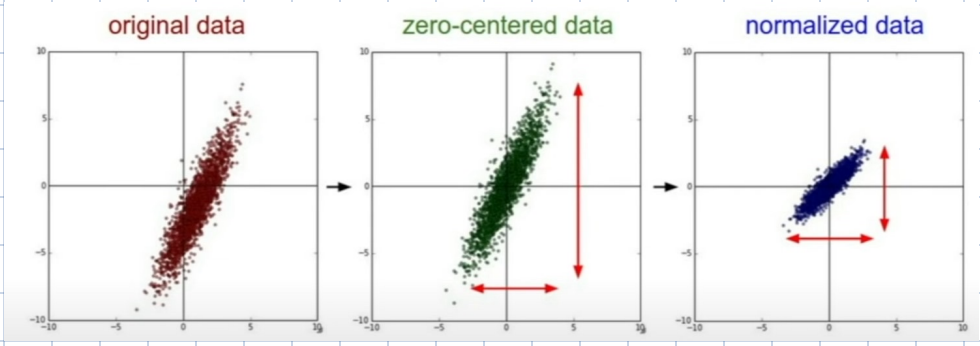


TLDR:

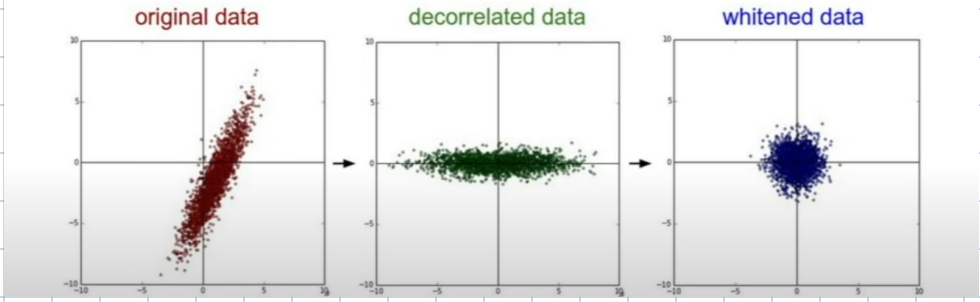
- Use ReLU. Be careful with α
- Try out Leaky relu/Maxout/ELU
- Try out tanh but don't expect much
- Don't use sigmoid

Data pre-processing:

- Normalize data



In practice, you may also see **PCA** and **Whitening** of the data



TLDR:

For images:

- Sub mean
- Sub per channel (RGB) mean

Weight initialization:

1. $W = 0$
   - ↳ All neurons do same thing & update in same way.
2. Small random numbers
   - ↳ Normal dist with 0 mean & 1e-2 std
   - ↳ Works okay for small networks
   - ↳ Due to repeated $W \cdot x$, std shrinks to 0
3. Xavier initialization
   - ↳ Normal dist with 0 mean & std / input number
     - ↳ if no. of input/neurons is small, we divide by this number & get large weights to maintain variance & vice versa. Var input = Var output

$$\frac{dL}{dw} = \frac{dx}{dw} \times \frac{dL}{dz} \quad \text{(local} \times \text{upstream)}$$

$$= x \times 1$$

(x is small)

$$\frac{dL}{dx} = \frac{dx}{dx} \times \frac{dL}{dz} \quad \text{(local} \times \text{upstream)}$$

$$= w \times 1$$

multiplying small w with upstream we get small gradient

↳ Breaks with ReLU as no.of nuerons halves
↳ Adjust for this by dividing by 2 in addition to no.of nuerons

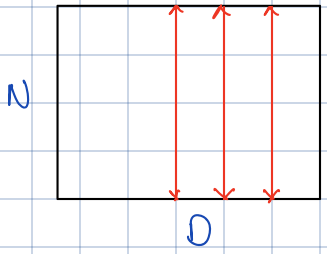
Batch Normalizations:

Consider a batch of activations at some layer. To make each dimension unit normal apply:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \quad , \quad y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

NOTE:
network can learn
$\gamma^{(k)} = \sqrt{Var[x^{(k)}]}$
$\beta^{(k)} = E(x^k)$
to recover the identity mapping

- Make this happen at every layer.



- mean along D

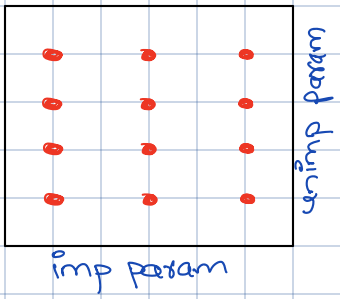
- Added after Fully connected / Conv layers & before non linearity

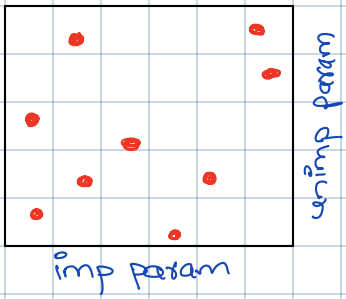
Baby-sitting learning process:

1. Zero mean data
2. Choose architecture
3. Loss is reasonable
4. Start with small data so that we can overfitting
5. Start small reg & choose α that makes go down $[1e-3, \cdots, 1e-5]$

Hyper-parametre optimizations:

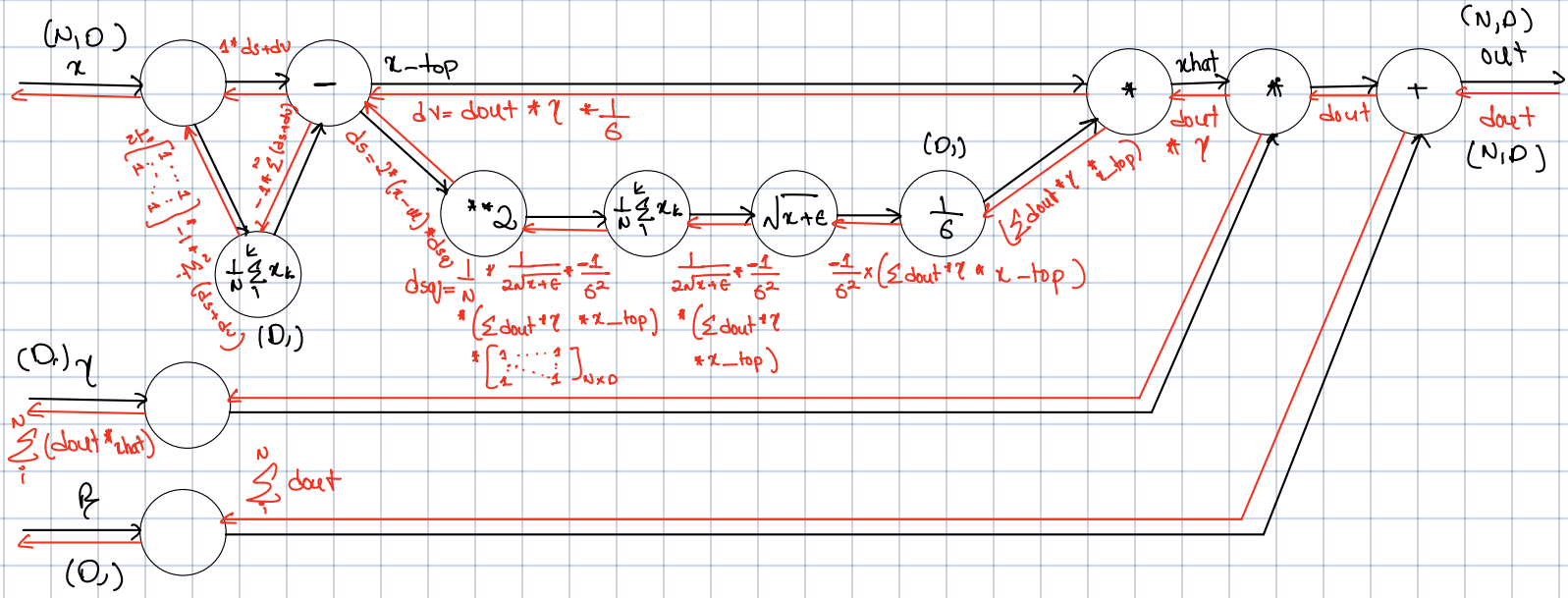
Random search vs Grid search





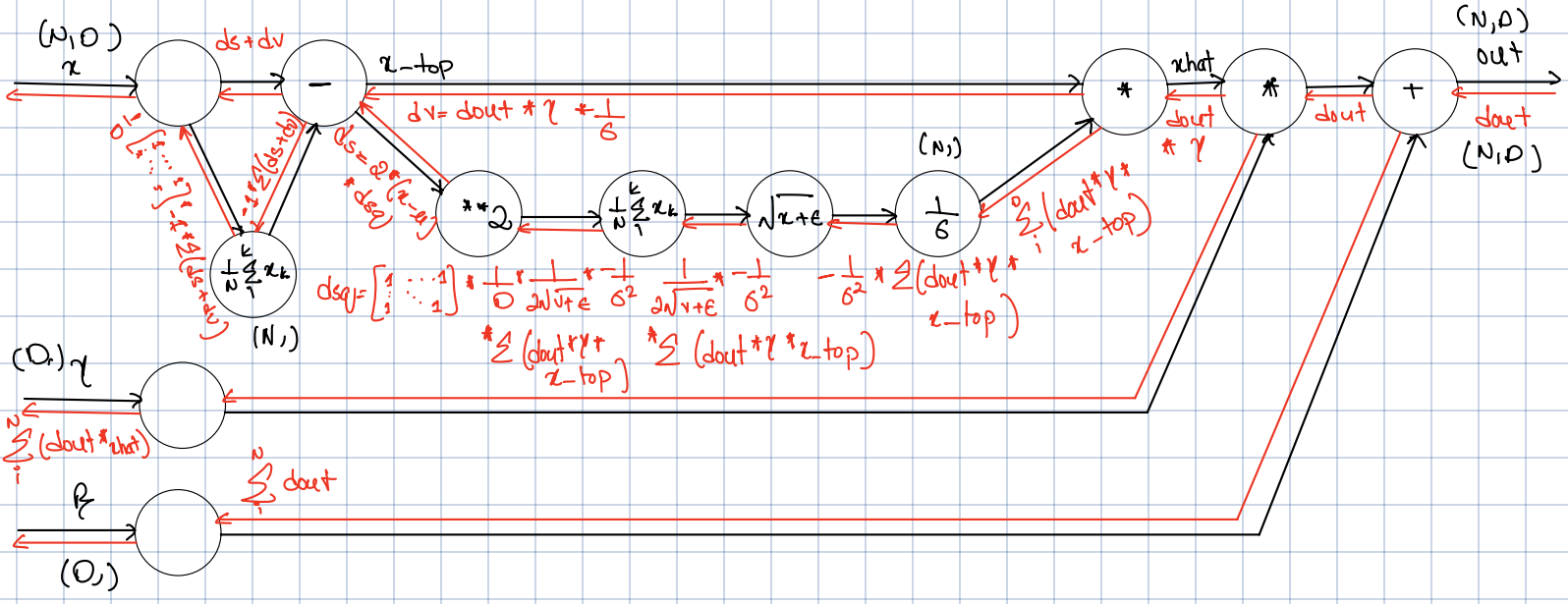
Assignment helper

Q2. Batch-norm:



$$dx = 1 * ds + dv * \frac{1}{N}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & & 1 \end{bmatrix} * -1 \sum_i^N (ds+dv)$$

Layer Norm:



$$dx = ds + dv - \frac{1}{D}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & & 1 \end{bmatrix} * \sum (ds+dv)$$