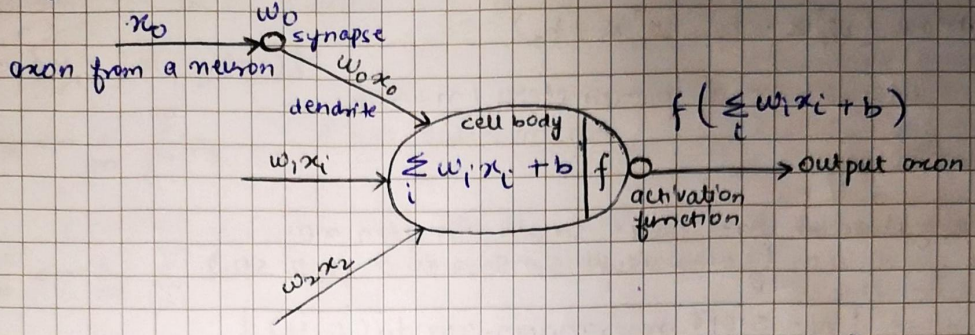
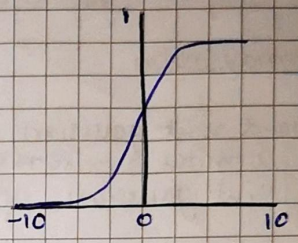
Lecture - 6 Training Neural Networks - I

→ Activation functions



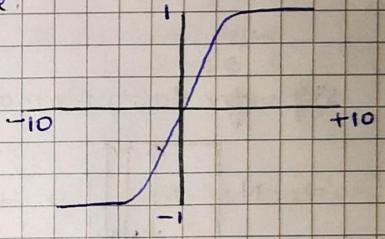
① Sigmoid → $\sigma(x) = \frac{1}{(1+e^{-x})}$

→ Squashes no. to range [0,1]
→ Historically popular as they have nice interpretation as a saturating "firing rate" of a neuron

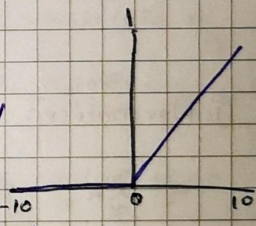


Problems → → Saturated neurons kill the gradients.
→ Sigmoid outputs are not zero-centered.
→ exp() is a bit computationally expensive.

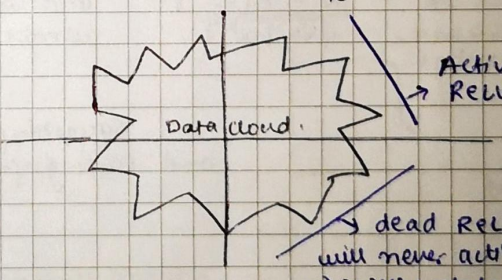
② Tanh(x) → Squashes no. to range [-1,1]
→ Zero-centered.
→ Still kills the gradient when saturated.



③ ReLU (Rectified Linear Unit) → computes $f(x) = \max(0,x)$
→ does not saturates in +ve region
→ computationally very efficient.
→ Converges much faster than sigmoid/tanh
→ more biologically plausible than sigmoid



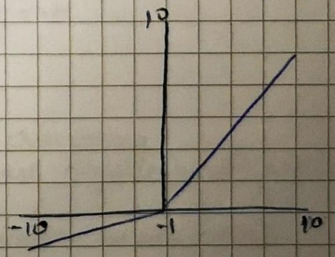
Problems → output not zero-centered
→ gradient when $x < 0$



★ People like to initialize ReLU neurons with slightly positive biasis (eg. 0.01)

④ Leaky ReLU → Does not saturate
→ computationally efficient
→ converges much faster than sigmoid/tanh in practise
→ will not die

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



⑤ Max out "Neuron" → Does not have the basic form of dot product → Nonlinearity
→ Generalises ReLU and Leaky ReLU.
→ Linear Regime ! Does not saturate. ! Does not die.

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem → doubles the no. of parameters / neurons.

→ Data Preprocessing

→ Weight initialization → first idea: small random no.
(gaussian with 0 mean and 1e-2 SD)

$$W = 0.01 * np.random.randn(D, H)$$

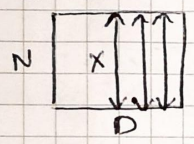
→ Batch normalisation

"You want unit gaussian activations? Just make them so".
Consider a batch of activations at some layer. To make each dimension unit gaussian, apply.

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^k]}}$$

This is a vanilla differentiable function.

Steps for Batch Normalisation: ① compute the empirical mean and variance independently for each dimension.



② Normalize with the above formula.

③ Usually inserted after fully connected or Conv. layers and before nonlinearity.
Problem: do we necessarily want a unit gaussian input to a tanh layer?

① Normalise:

$$\hat{x}^k = \frac{x^k - E[x^k]}{\sqrt{Var[x^k]}}$$

and then allow the network to squash the range if it wants to:

$$y^k = \gamma^k \hat{x}^k + \beta^k$$

squashing and scaling operation

$\beta^k = E[x^k]$

$\gamma^k = \sqrt{Var[x^k]}$

It helps to recover the identity mapping.

Summary : Batch Normalization

Input: Values of x over a mini-batch : $\beta = \{x_1 \dots m\}$;
Parameters to be learned : γ, β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{// normalise}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta = BN_{\gamma,\beta}(x_i) \quad \text{// scale and shift.}$$

+ Improves gradient flow through the network
+ Allows higher learning rate
+ Reduces the strong dependence on initialization
+ Acts as a form of Regularization in a funny way, and slightly reduces the need for a dropout.

→ **Babysitting the learning Process**

* start with the small regularization and find the learning rate that makes the loss go down.
* loss not going down → learning rate too low. } somewhere b/w 1e-3 - 1e-5
  loss exploding → learning rate too high. } 1×10^{-3} to 1×10^{-5}

Hyperparameter optimization.
To choose the values of our hyperparameters, we do: "Cross-Validation"

Strategy. → (i) Coarse - fine cross-validation in stages
first stage → only a few epochs to get a rough idea of what parameters work.
second stage → longer running time, finer search.

Hyperparameters to play with → Network architecture
→ learning rate, it's decay schedule, update type
→ Regularization

loss curve: