6

Training Neural Networks (I)

★ Mini-batch SGD

Loop:

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

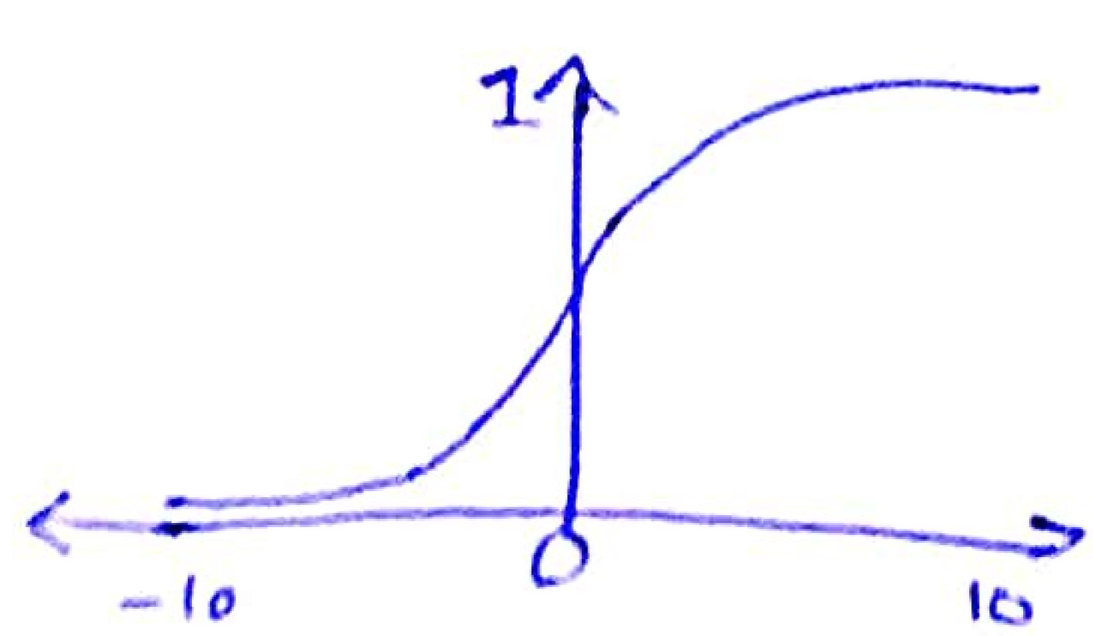
Part 1

- → Activation functions
- → Data Preprocessing
- → Weight Initialization
- → Batch Normalization
- → Babysitting the Learning Process
- → Hyperparameter Optimization

★ Activation Function

⊛ Sigmoid

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



→ Squashes numbers to range [0, 1]

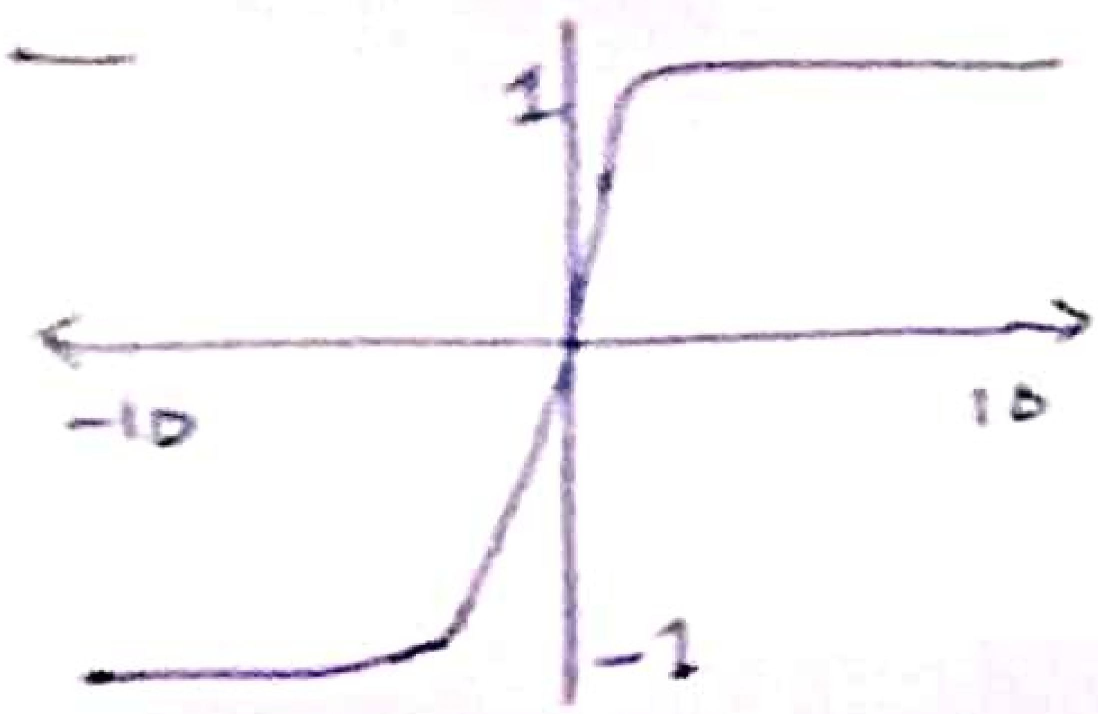
⇒ 3 Problems:

① Saturated neurons "kill" the gradient

② Sigmoid outputs are not zero-centered.

③ exp() is a bit compute expensive.

⊛ tanh(x)

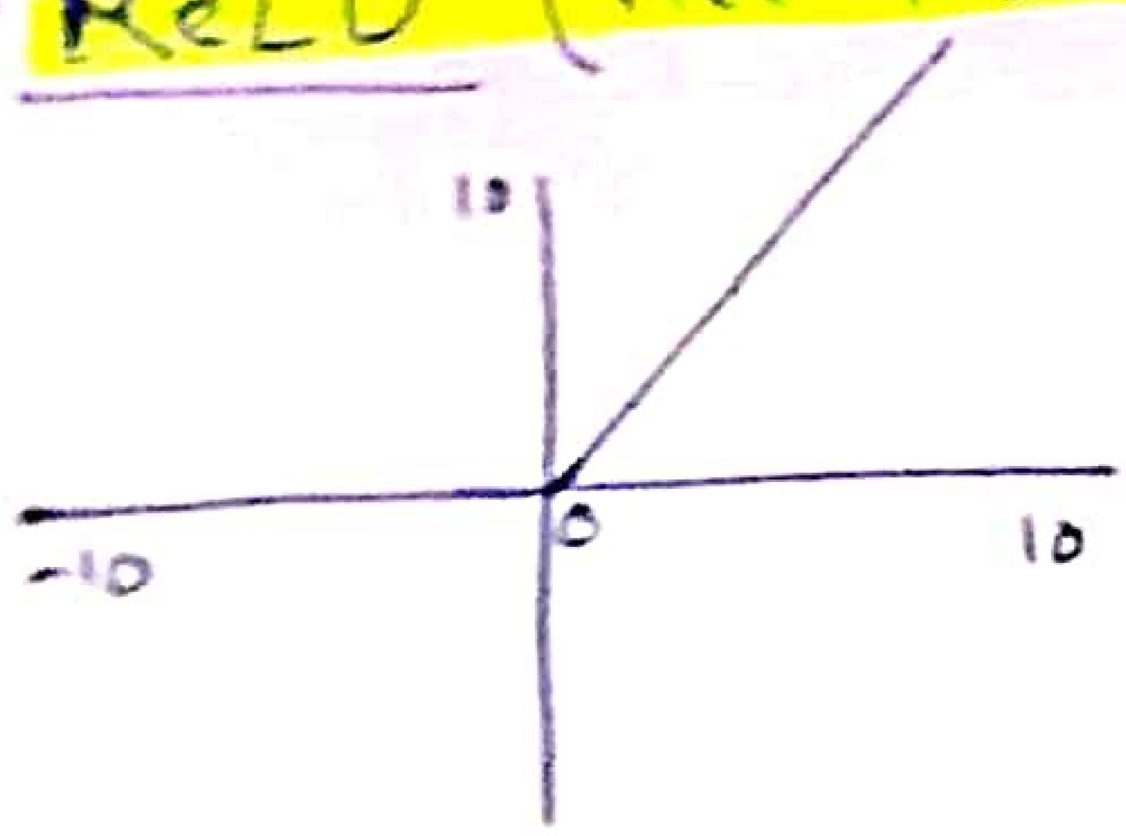


→ Squashes number to range [-1, 1]

→ Zero Centered (nice)

→ Still Kills gradients when saturated.

⊛ ReLU (Rectified Linear Unit)



$f(x) = \max(0, x)$

→ Does not Saturate (in +ve region)

→ Very computationally efficient

→ Converges much faster than sigmoid/tanh in practice (e.g. 6x)

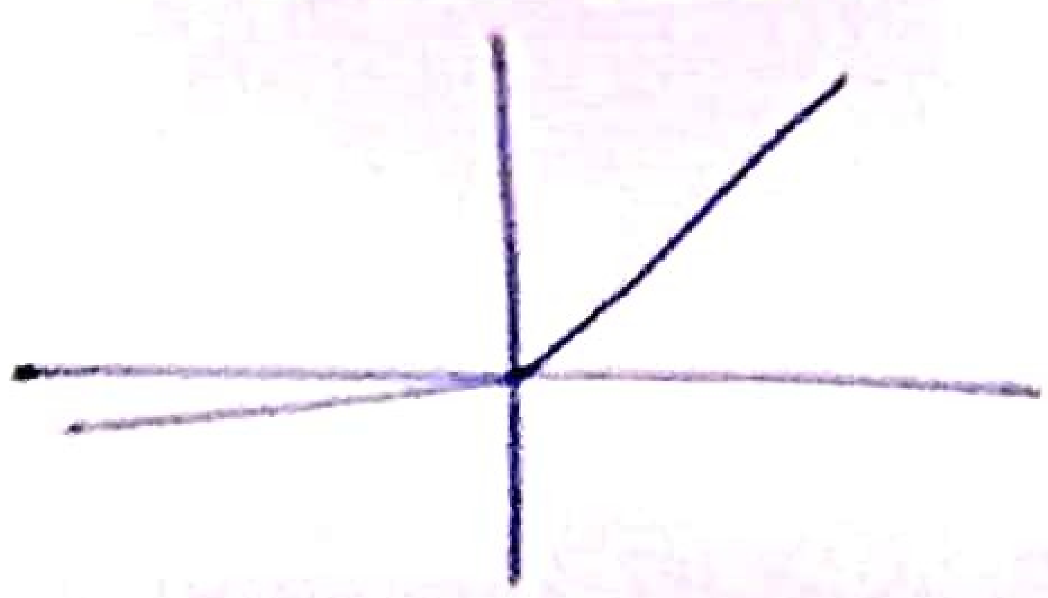
→ Actually more biologically plausible than sigmoid.

→ 2 Problems:

① Not zero centered output

② Gradient when $x < 0$ {die}

⊛ Leaky ReLU



→ Does not Saturate

→ Computationally efficient

→ Converges much faster than sigmoid/tanh in practice!.

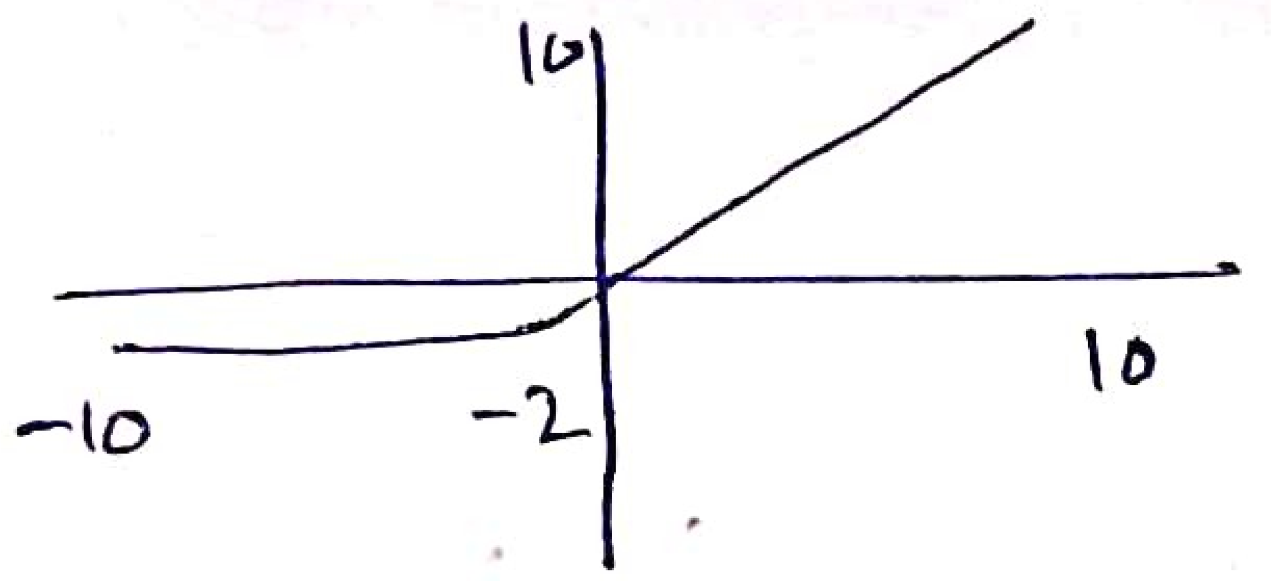
→ will not die.

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

⊛ Parametric ReLU (PReLU)

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

⊛ Exponential Linear Units (ELU)



- → All benefits of ReLU
- → Closer to zero mean output
- → Negative saturation regime compared with Leaky ReLU adds some robustness to noise.
- → Computation requires exp()

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

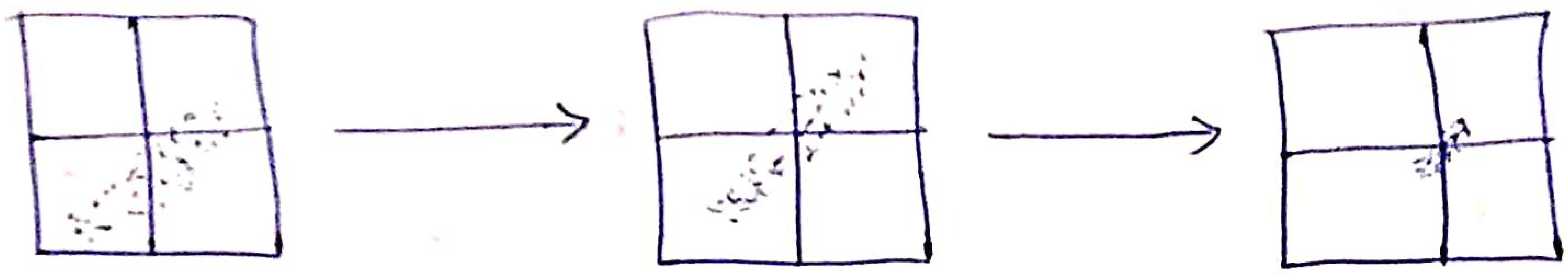
⊛ Maxout Neuron

- → Generalizes ReLU & 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 number of parameters/neuron.

★ Data Preprocessing



Original data → Zero-Centered data → Normalized data

Zero-Centered data:
- → Substract the mean image
- → Subtract per-channel mean

Normalized data:
- → Not common for images

★ Weight Initialization

① $W = 0$

⇒ All neuron will do the same thing

⇒ All neuron will always be same.

② Small random numbers for weights

(Gaussian with zero mean & 10^{-2} standard deviation)

⇒ Works ~ Okay for small networks, but Problems with deeper networks.

③ Xavier initialization (2010)

⇒ Reasonable initialization.

⇒ But when using the ReLU nonlinearity it breaks.

④ He et al. (2015)

↳ Additional $\frac{1}{2}$

★ Batch Normalization

⇒ "You want unit gaussian activations? just make them so?"

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

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

⇒ And then allow the network to squash the range if it wants to:

$$\boxed{y^{(K)} = \gamma^{(K)} \hat{x}^{(K)} + \beta^{(K)}}$$

Note:
→ The network can learn

$$\gamma^{(K)} = \sqrt{Var[x^{(K)}]}$$
$$\beta^{(K)} = E[x^{(K)}]$$

to recover the identity mapping

★ Babysitting the Learning Process

Step1: Preprocess the data

Step2: Choose the architecture

Step3: Double check that the loss is reasonable.

(a) → Zero regularization
(b) → Some regularization
{Loss should go up}

Step 4: Start training with very small amount of data

- Turn of regularization
- Make sure that you can overfit very small Portion of the training data.

{Make the Loss → 0}

Step 5: Start with small regularization & find learning rate that makes the loss go down.

- Loss barely changing ⟹ Learning rate is probably too small

Rough range for learning rate we should be Cross-validating is somewhere [1e-3 ... 1e-5]

★ Hyperparameter Optimization

Cross-validation Strategy

Coarse → fine Cross-validation in stages

First stage: Only a few epochs to get rough idea of what params work

Second Stage: longer runtime, finer search.

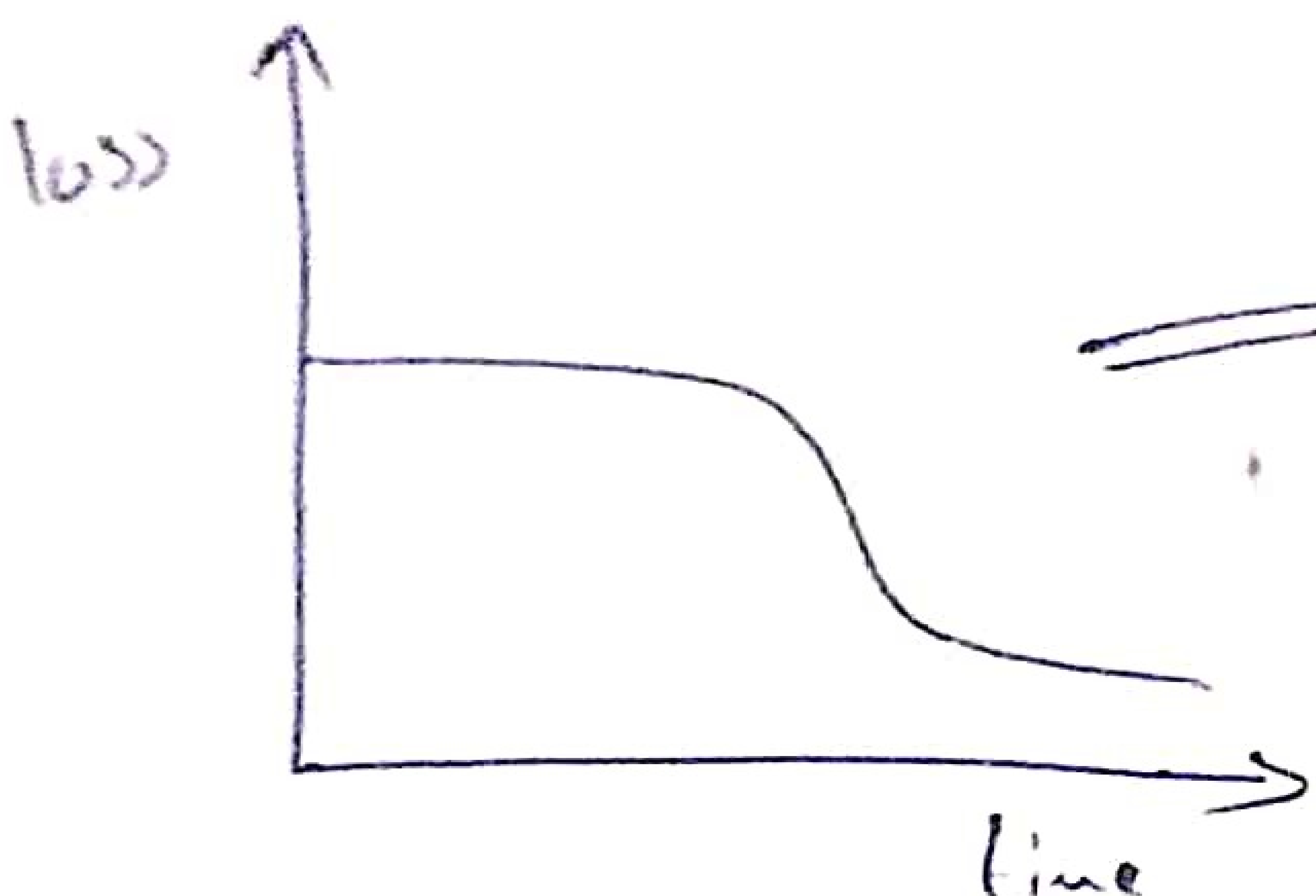
⟹ Random Search is better than Grid Search for hyperparameter.

⟹ Search for hyperparameter in log space when appropriate.

⇒ Hyperparameters to play with:

- → Network architecture
- → Learning rate, its decay schedule, update type
- → regularization (L2 / Dropout strength)

⇒ Monitor & Visualize the loss curve.



→ Bad initialization
a prime suspect