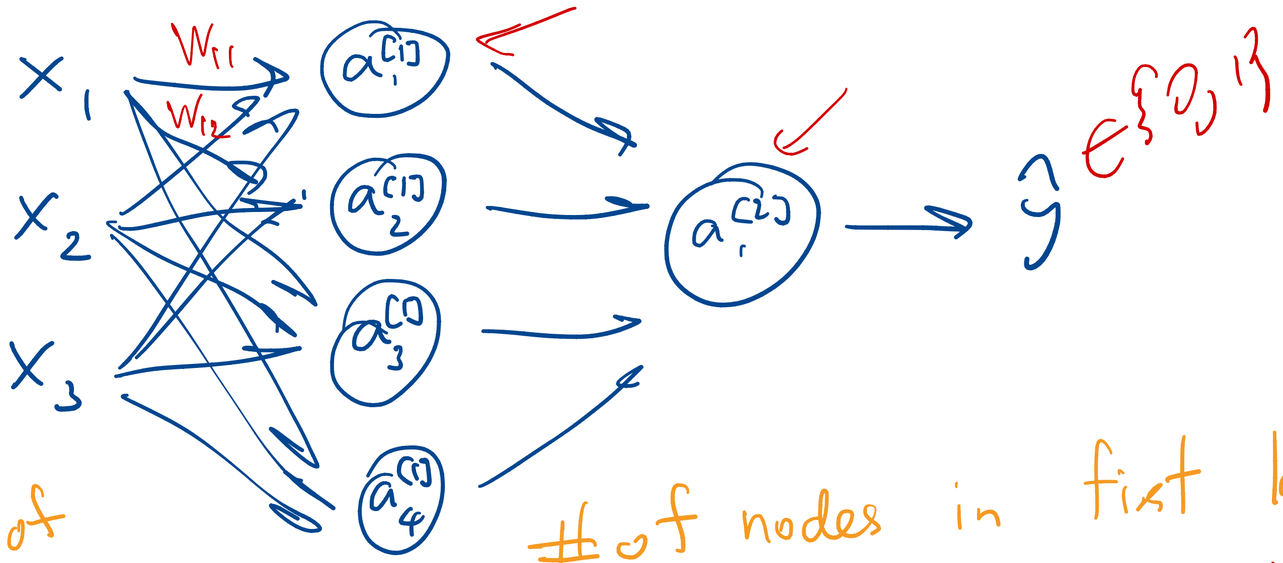
ICCS482 Deep Learning

Lecture 4: Neural Networks

Sunsern Cheamanunkul, Sep 16, 2020.



$X - (3, m)$ $\quad [x^{(1)} x^{(2)} \dots x^{(m)}]$

$W^{[1]} - (4, 3)$

$b^{[1]} - (4, 1)$

$W^{[2]} - (1, 4)$

$b^{[2]} - (1, 1)$

# of features = 3

# of nodes in first layer = 4

# of nodes in layer 1

# of nodes in layer 0

activation function

$$z^{[1]} = W^{[1]} \cdot x + b^{[1]} \qquad a^{[1]} = g(z^{[1]}) \longrightarrow a^{[1]}$$

$z^{[1]} - (4, m)$

$a^{[1]} - (4, m)$

Same dimension

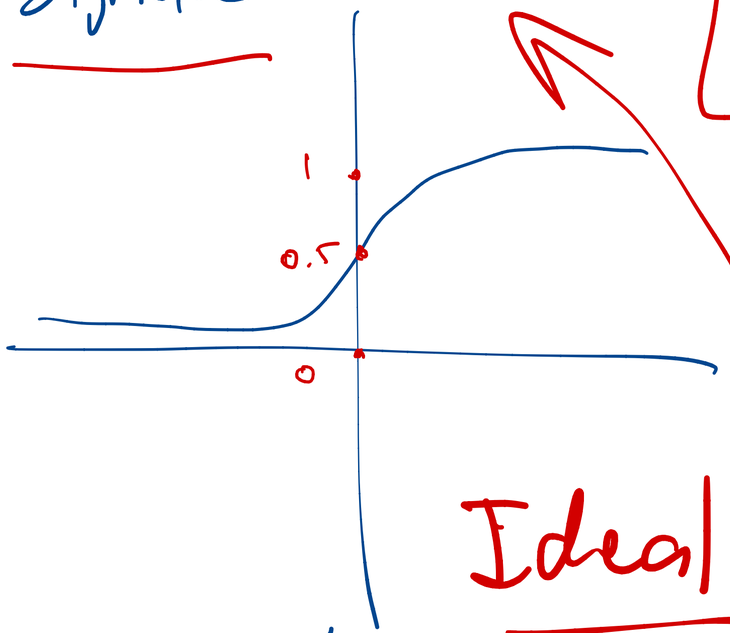
$z^{[2]} - (1, m)$

$a^{[2]} - (1, m)$

Activation Function

Sigmoid

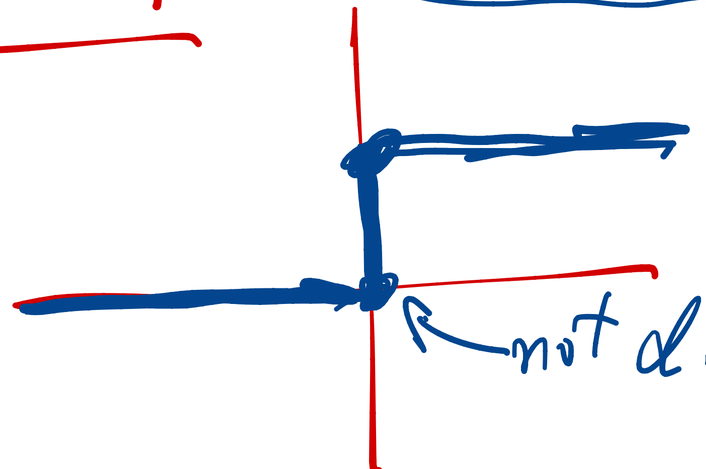
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



no gradient

Ideal

$$step(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ otherwise} \end{cases}$$



not differentiable!

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

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

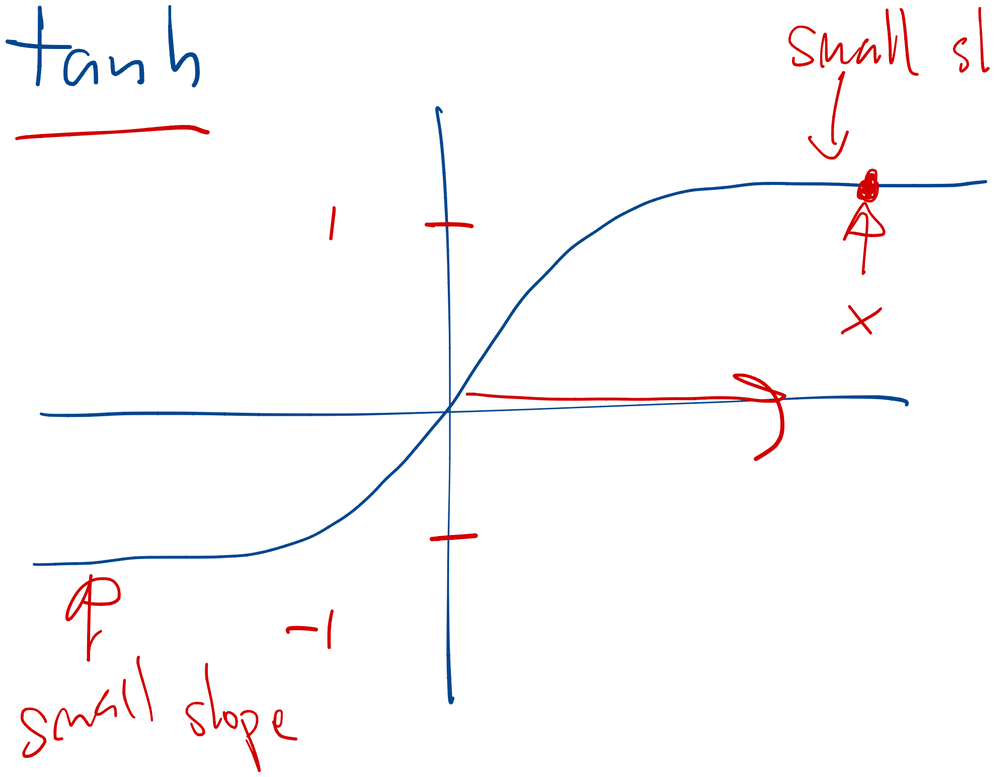
$$= \frac{1}{1+e^{-x}}\left(\frac{1-1+e^{-x}}{1+e^{-x}}\right)$$

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

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

$$= \sigma(x)(1-\sigma(x))$$

tanh



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

$$\frac{d \tanh(x)}{dx} = \frac{(e^{x} + e^{-x})(e^{x} + e^{-x}) - (e^{x} - e^{-x})(e^{x} - e^{-x})}{(e^{x} + e^{-x})^{2}}$$

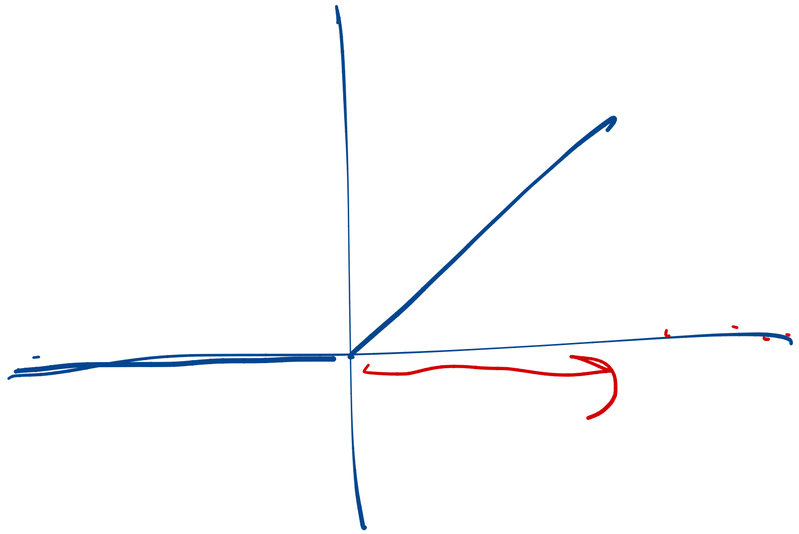
$$= 1 - (\tanh(x))^{2}$$

1. centered around 0
2. Symmetry around 0

ReLU — Rectified Linear Unit

$w := w - \alpha \cdot dw$

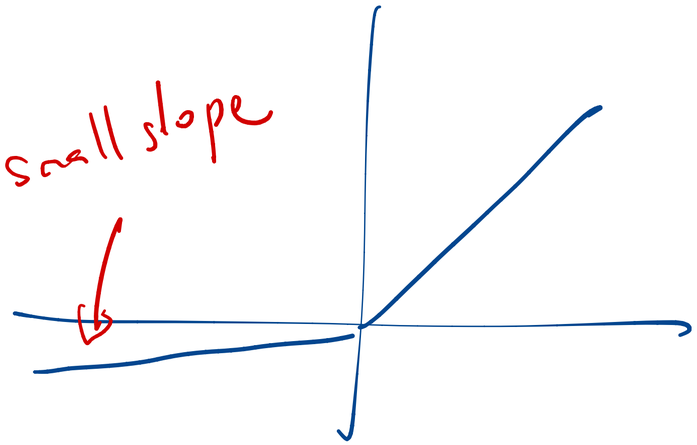
non-zero



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Leaky ReLU



$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Why do we need activation functions?

$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

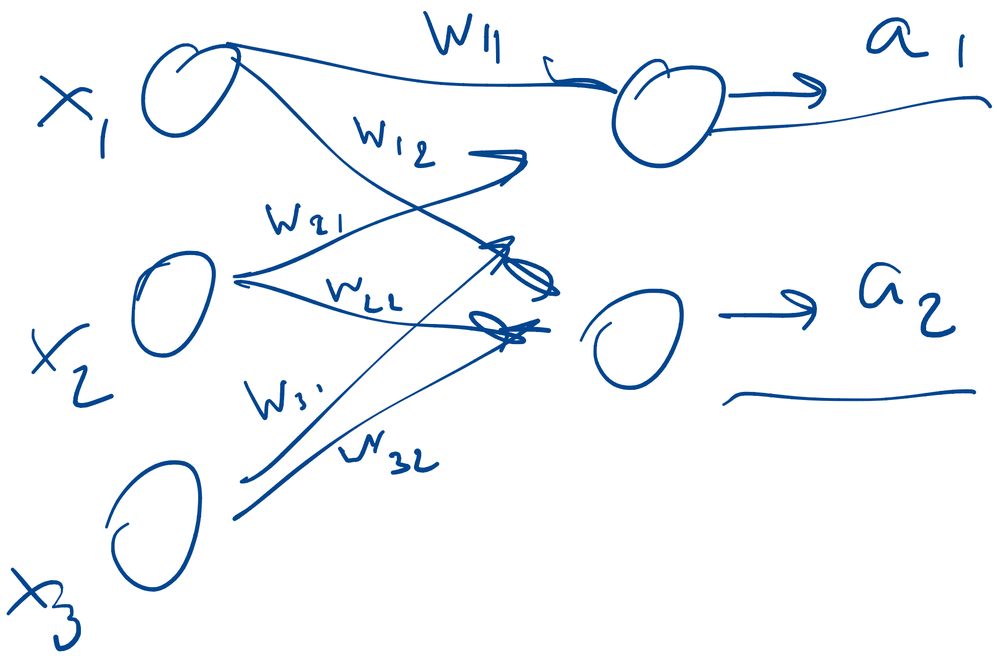
$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]} = \left(W^{[2]} \cdot W^{[1]}\right) x + \left(b^{[2]} + b^{[1]}\right)$$

$$a^{[2]} = g(z^{[2]})$$

NN algorithm

1. Define model structure ← model complexity, bias vs. variant
2. Initialize model params
3. Loop:
   - 3.1) Calc loss (forward prop)
   - 3.2) Calc derivatives (backward prop)
   - 3.3) Apply update rule (GD)
4. Win!

Model Initialization



Idea (1)

$W_{11} = W_{12} = W_{21} = W_{22} \dots = 0$

update

$W_{11} = W_{21}$

$W_{12} = W_{22}$

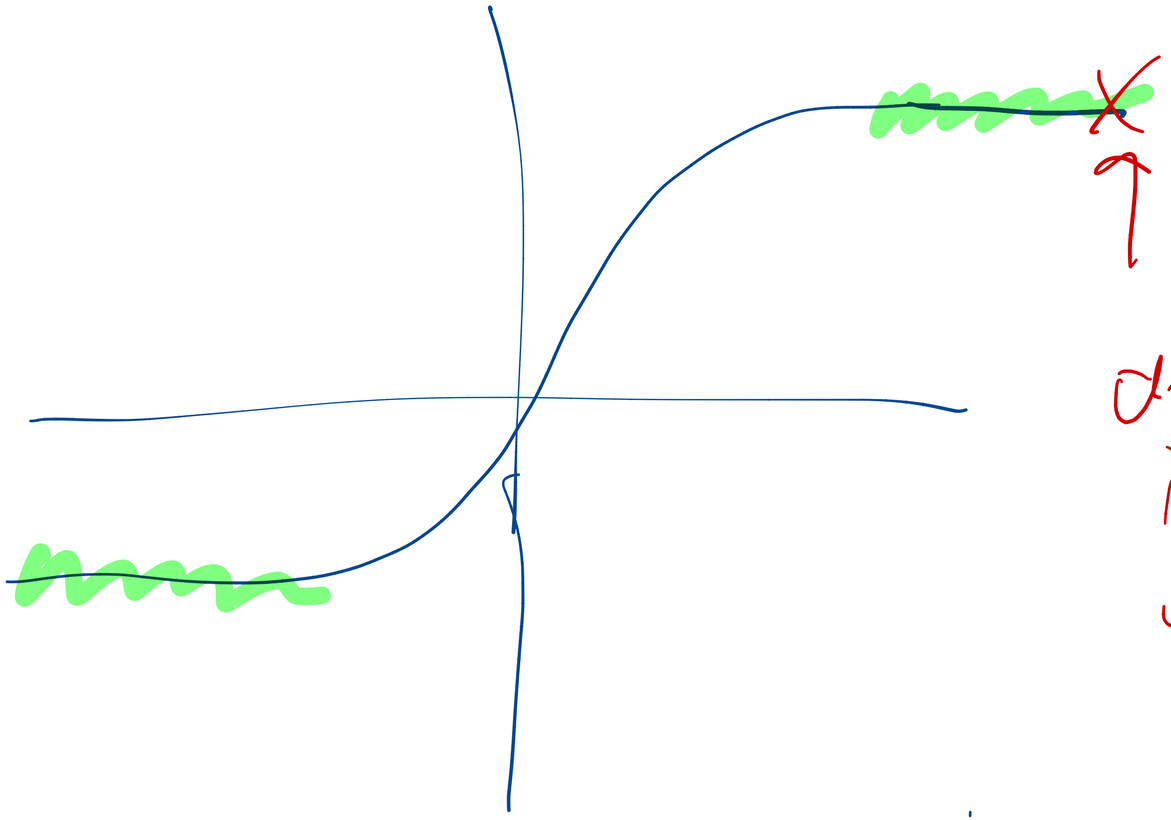
⋮

no!

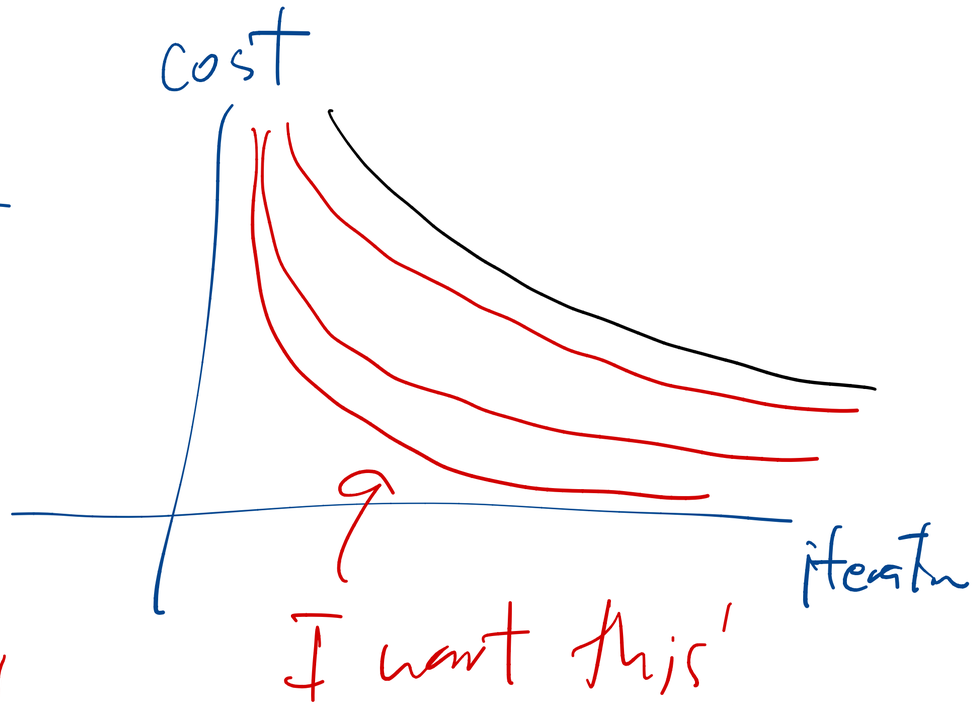
Idea (2)

initialize weights randomly!

Can we use large w?



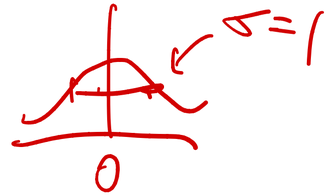
don't initialize weight here.



No, you need to initialize with relatively small weights

$w = np.random.randn(shape) * 0.01$

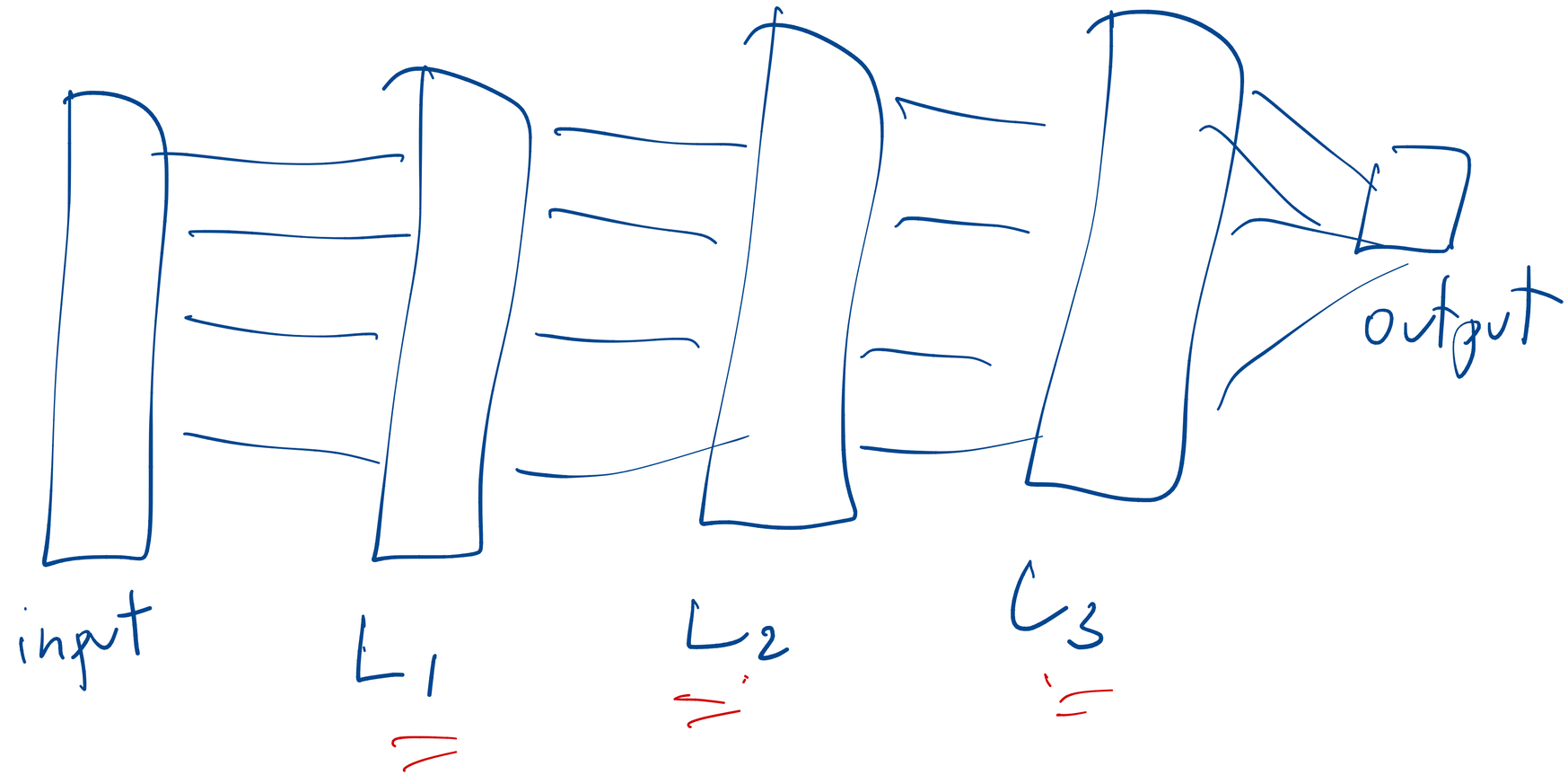
← standard normal distribution



$w = np.random.rand(shape) * 0.01$

↖ uniform(0,1)

Deep Network



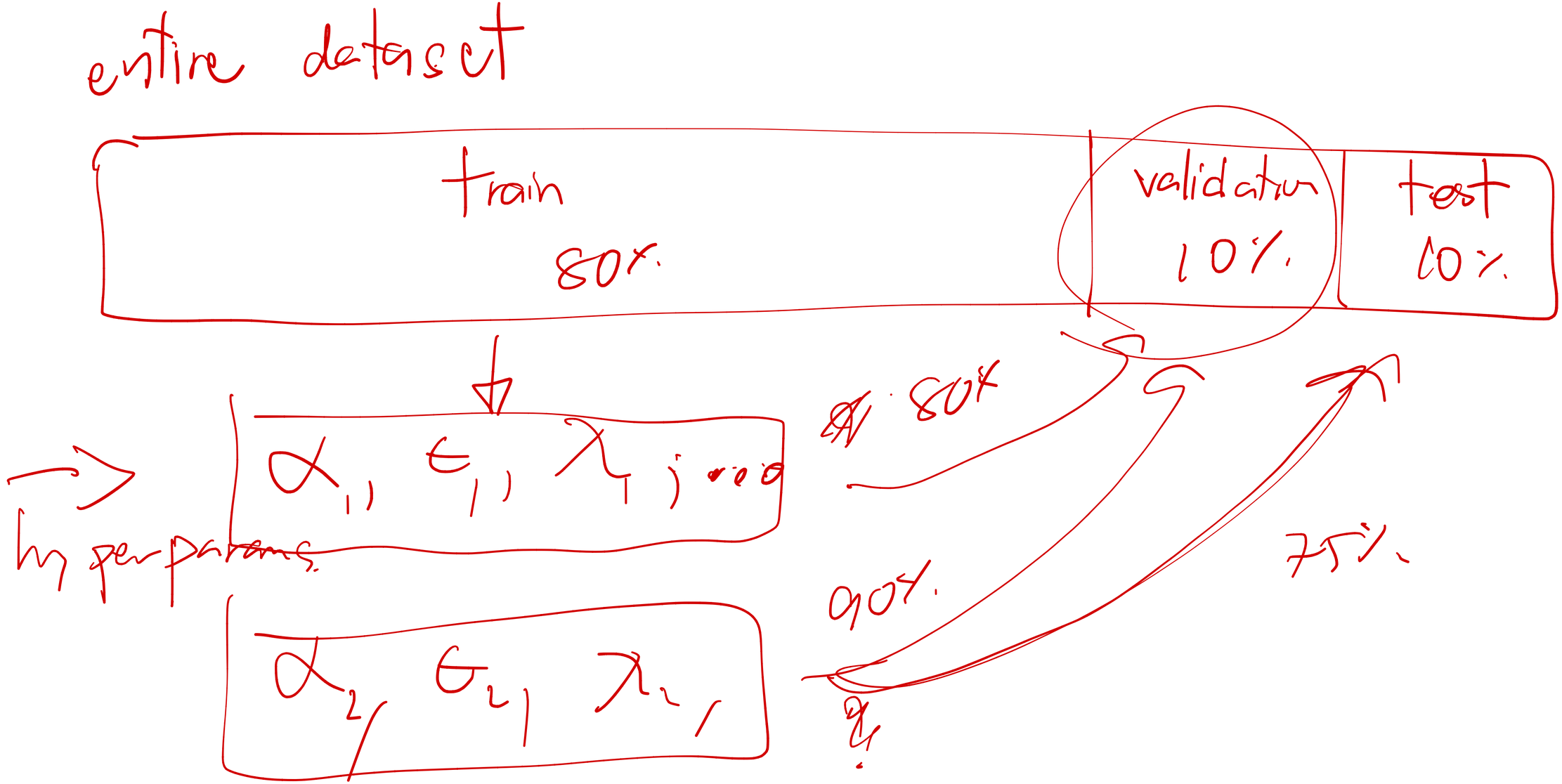
NN params: W's and b's

figured out by GD

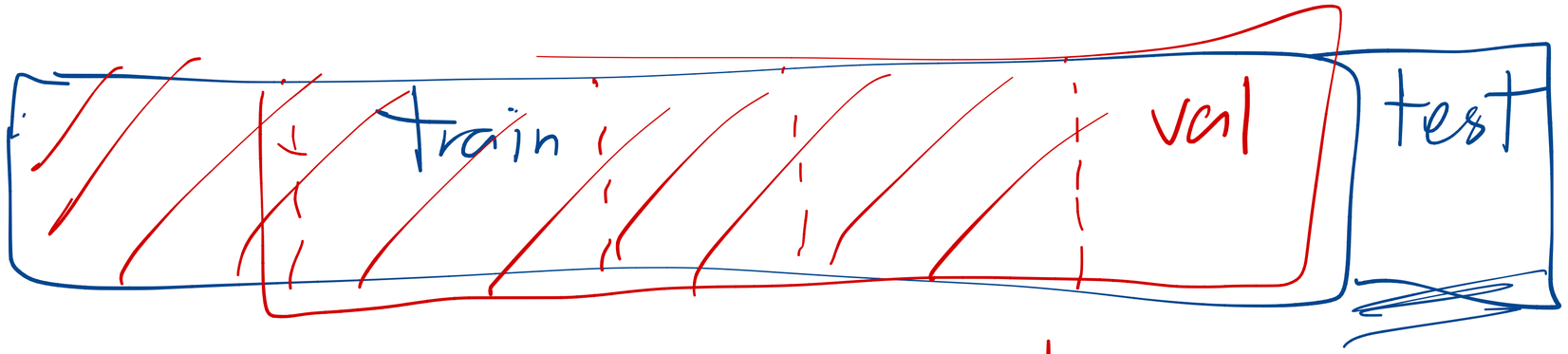
Hyperparams:

- learning rate
- # of hidden nodes
- # of hidden layers
- # of training iterations
- choice of activation fcns
- etc.

need experiment validation using set


entire dataset
train
80%
validation
10%
test
10%
hyperparams
80%
90%
75%

Cross-validation



k-fold CV (cross validation)

Advantage:

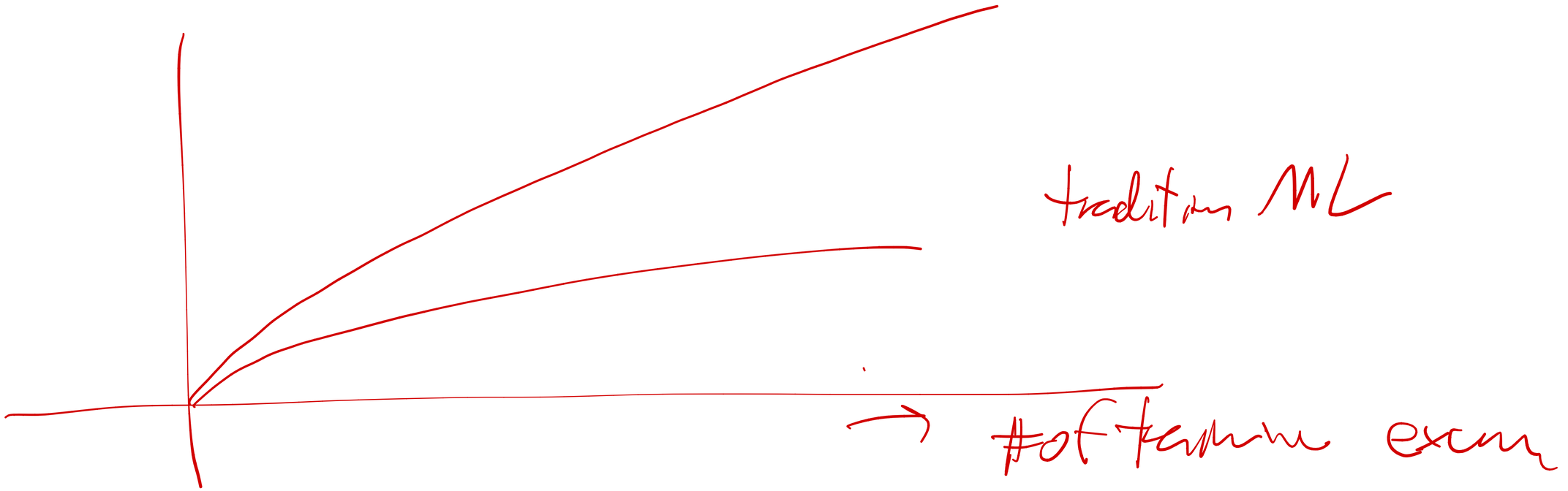
- you get to see the "variance" of your model

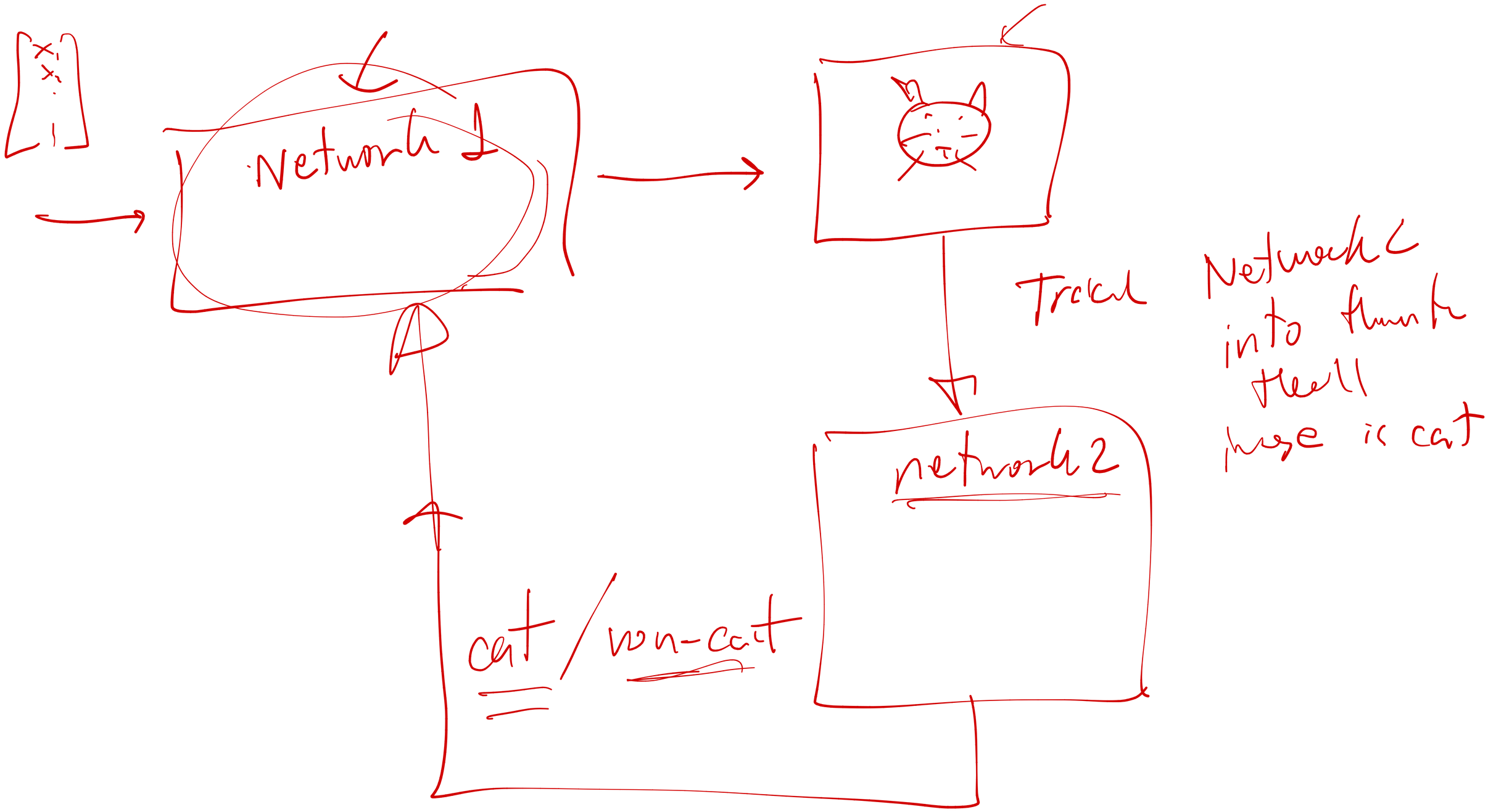
Practice Train-test split

Train 99%

test 1%

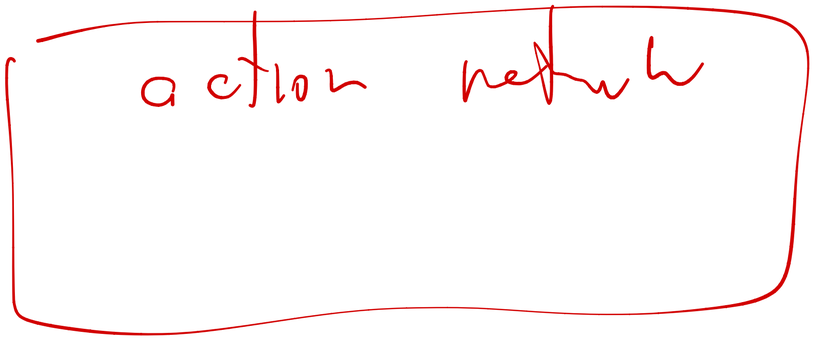
100,000,000





Network C into thumb tell huge is cat


policy network


action network