Lecture Plan: Two-layer neural net (or multi-layer perceptron)

1.1 Perceptron

- Threshold function
- Layered representation
- Representation power: XAND

1.2 Sigmoid activation.

- A smooth decision boundary
- Formula and simulation

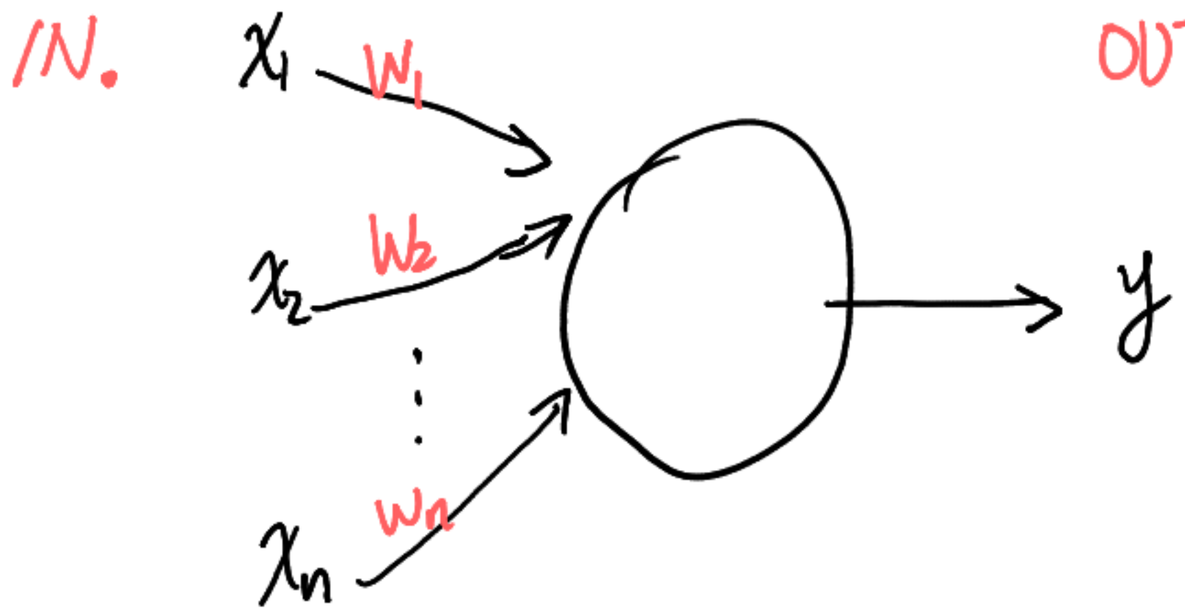
1.3 Architecture of a two-layer neural net

- Feedforward NN, loss function
- Discussion of RNN

1.4 Learning algorithm

- Local search, gradient descent
- Stochastic gradient descent

1.1 Perceptron: A type of artificial neuron developed in the 1950s and 1960s



$$\text{OUT} = \begin{cases} 0 & \text{if } \sum_{j=1}^{n} w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_{j=1}^{n} w_j x_j > \text{threshold} \end{cases}$$

Ex. There's a festival in the city, and you are debating whether to attend or not.

(x_1) Is the weather good? $w_1 = 5$

(x_2) Do they manage health risks related to COVID well? $w_2 = 3$

(x_3) Is the festival near public transit? $w_3 = 1$

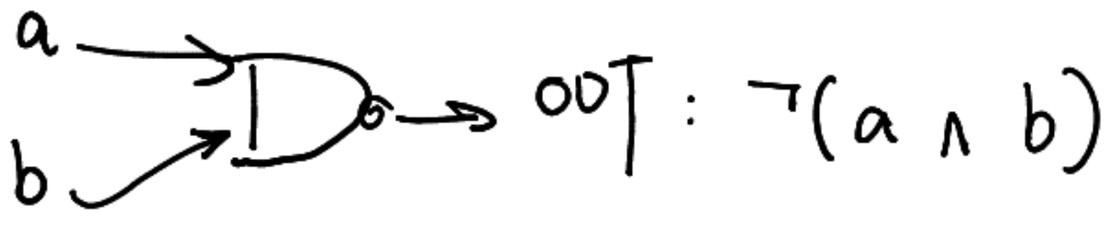
Vector notation.

$$\text{OUT} = \begin{cases} 0 & \text{if } \langle w, x \rangle + b \leq 0 \\ 1 & \text{if } \langle w, x \rangle + b > 0 \end{cases}$$

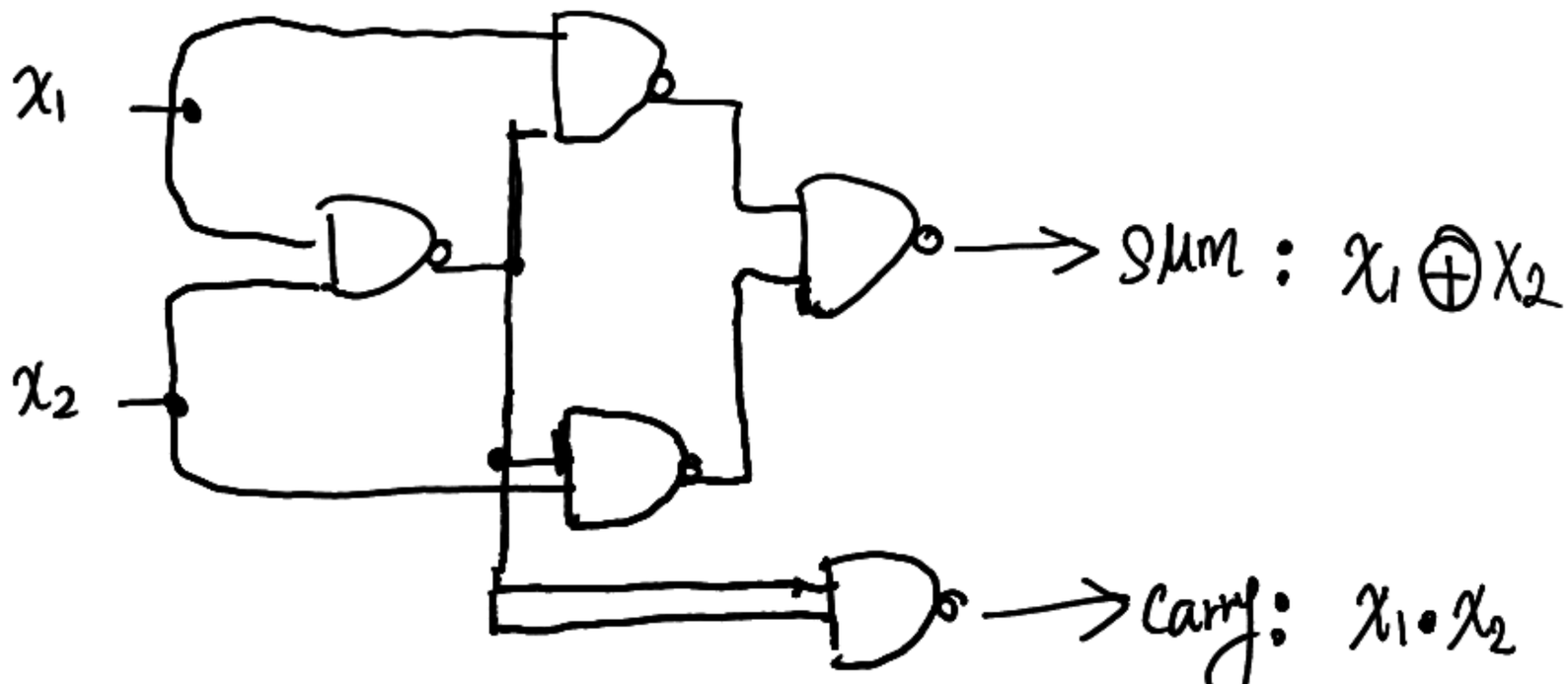
b: threshold (bias), measures how easy it is for the neuron to activate

Representation. We can also use the perceptron to represent logical computation.

XAND.



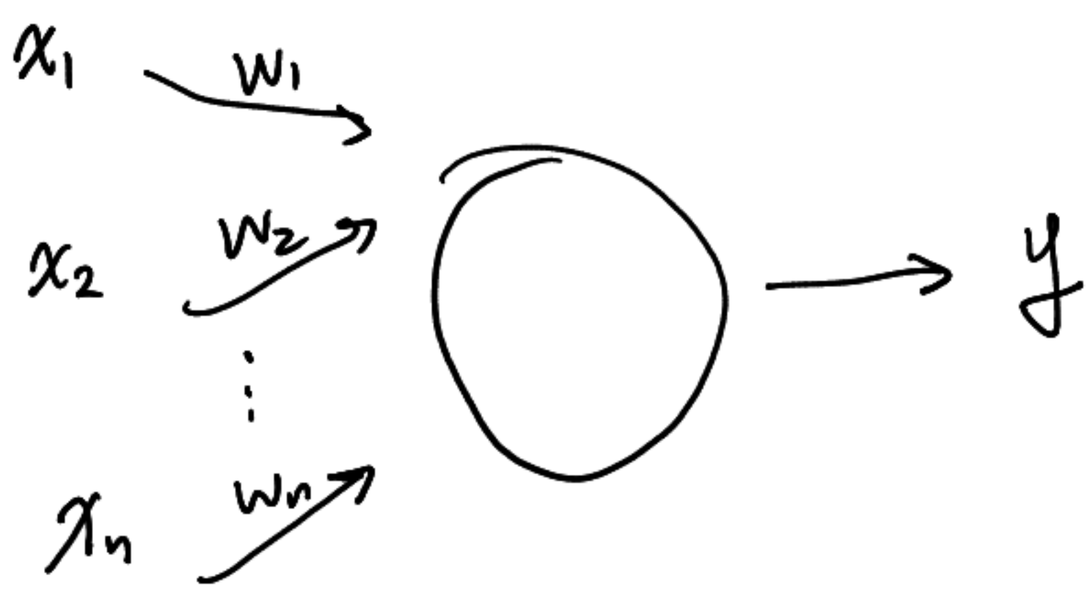
Using perceptron to represent XAND.



To get an equivalent network of perceptrons, we simply replace the XADN gates with the perceptron above!

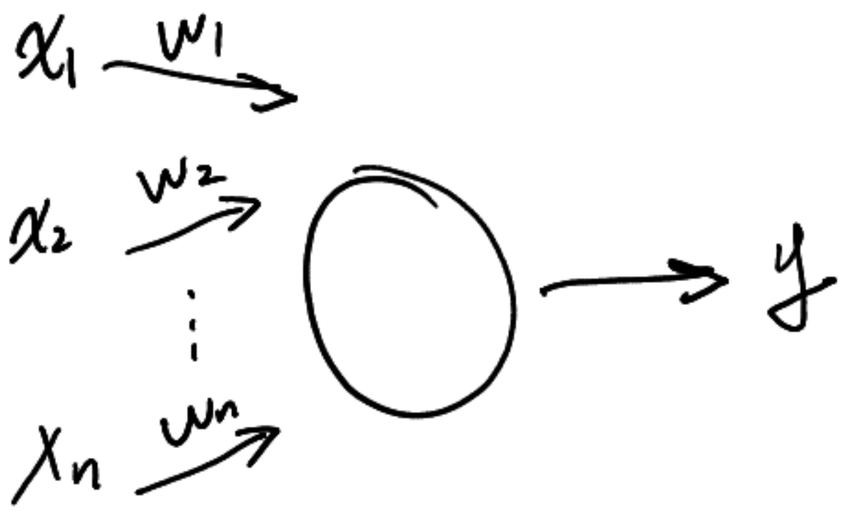
1.2 Sigmoid neurons : A smooth transition

Susceptible to small perturbations.

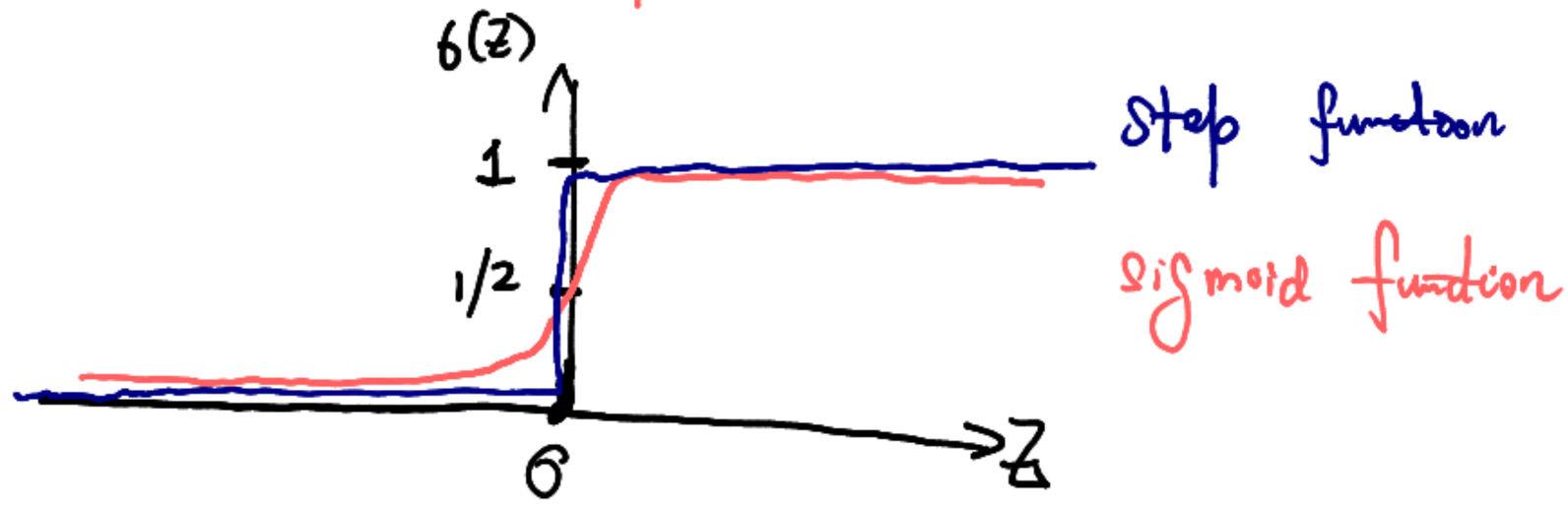


If $\langle x, w \rangle + b \approx \varepsilon$, then a small change of $\frac{\varepsilon}{w_j}$ on x_j flips the output y!

Sigmoid neuron.



Sigmoid function. $\sigma(z) = \frac{1}{1+\exp(-z)}$, for any $z \in \mathbb{R}$



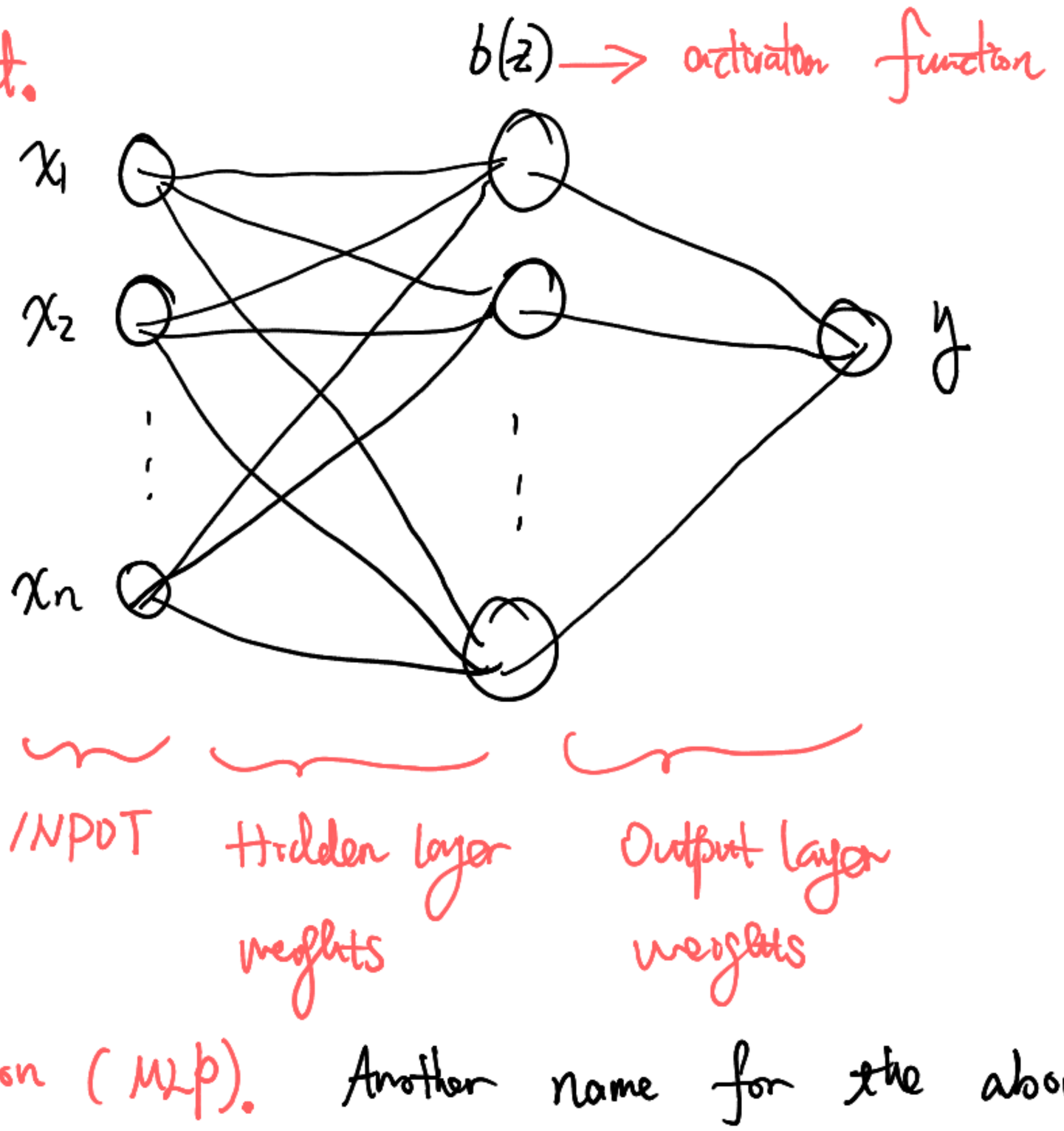
OUT: $y = \frac{1}{1+\exp(-\langle w, x\rangle - b)}$

Intuition: (i) when $\langle w, x\rangle + b$ is very large (say >10),
then y is very close to 1
(ii) when $\langle w, x\rangle + b$ is very small (say <-10),
then y is very close to 0

Remark. (i) One can also change the slope of sigmoid using a multiplier λ in $\exp(-\lambda \cdot z)$,
(ii) Sigmoid is differentiable whereas the step function is not.

1.3 Architecture of a neural network

Two-layer neural net.



Remark.

Multi-layer perceptron (MLP). Another name for the above model.

Other names you might see in the literature. One-hidden-layer neural net; Three-layer neural net..

Design choices.

→→ # of neurons in the hidden layer (width r)

Def. (Over-parametrized models) We say that a model is over-parameterized if # parameters > # of input samples

In the example we have above, suppose that

# of input samples $= n$, and

$x_i \in \mathbb{R}^p$, $\forall \ 1 \le i \le n$

Then # of parameters = # of parameters in the hidden layer $(p \times r)$

$$+ \text{ \# of parameters in the output layer } \quad (r \times 1)$$

$$= p \times r + r \times 1 = (p+1) \times r$$

So a two-layer neural net is overparametrized if

$$(p+1) \times r > n \text{ , or } r > \frac{n}{p+1}$$

Width plays a fundamental role in the study of neural net

Expressiveness ★ The more neurons there are, the more parameters, hence the more "expressive" our neural net is.

Generalization ★ With more parameters, our neural net might be more prune to "over-fitting".

Optimization ★ It turns out the width also impacts how fast our neural net will converge!

We will leave our discussion of width for now and come back to this core concept repeatedly later in the class.

Activation function.

$$\sigma(z) = \frac{1}{1+\exp(-z)} \text{ , sigmoid activation}$$

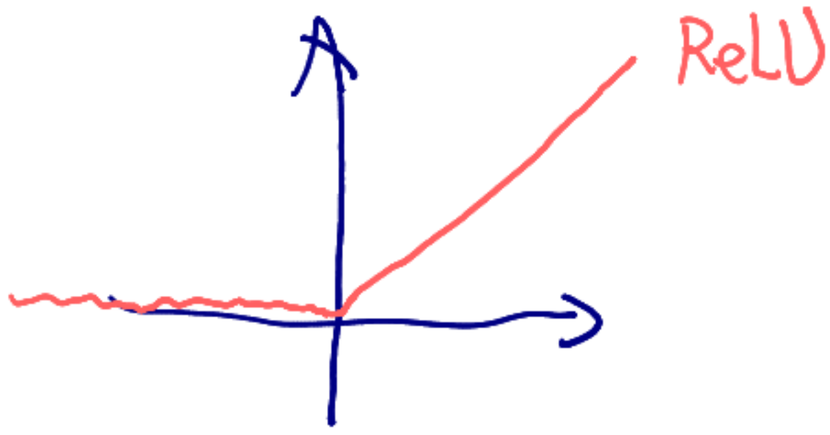
$$\sigma(z) = z \text{ , linear activation}$$

With linear activation, the neural net parametrization becomes:

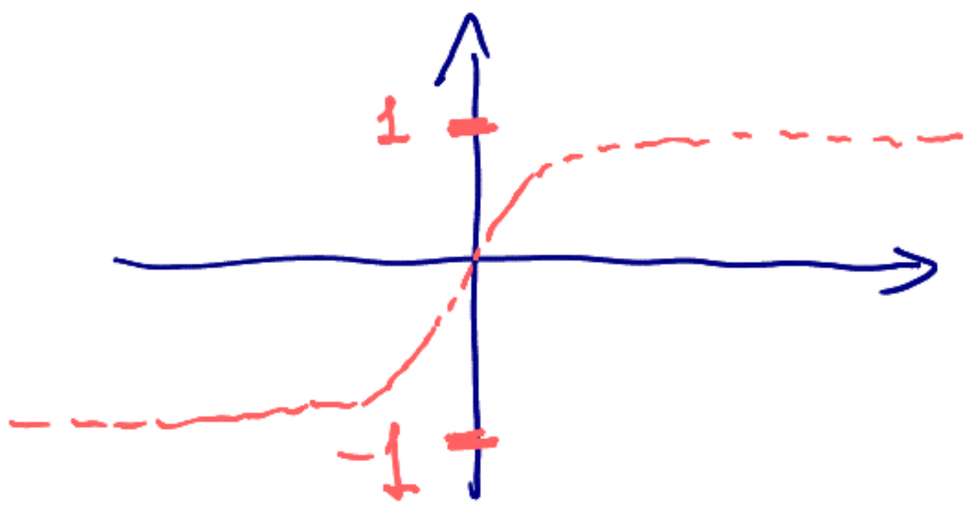
hidden-layer weights · output-layer weights

↳ a.k.a. matrix factorization (or principal component analysis)

$\phi(z) = \text{ReLU}(z) = \max(0, z)$



$\phi(z) = \tanh(z) = \dfrac{e^{2z} - 1}{e^{2z} + 1}$,



Similar to sigmoid but allows for the (-1) mode.

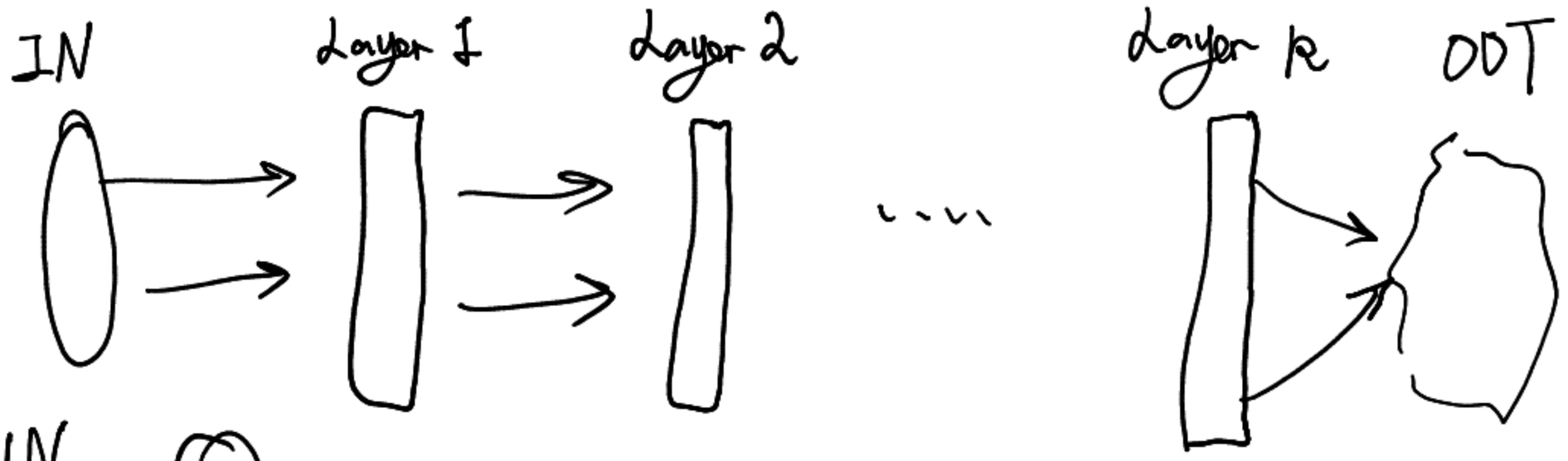
# of output neurons.

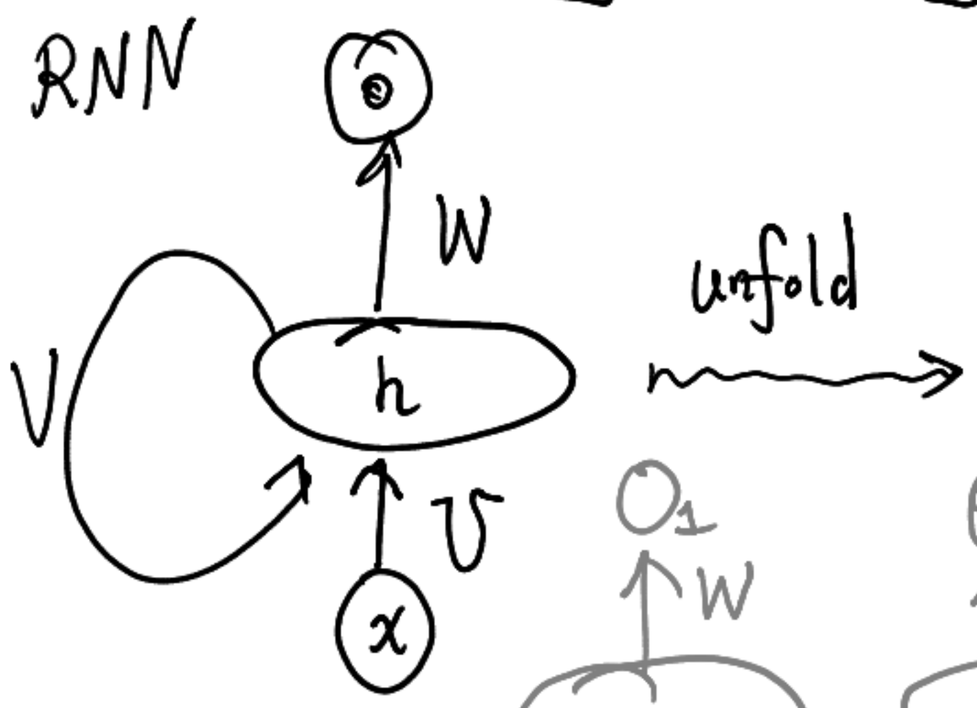
- In the MNIST example, we want 10 output neurons, where each predicts whether the input $\longrightarrow \{0, 1, \ldots, 9\}$.
- For binary classification, we want 2 output neurons.
- For regression analysis, we want 1 output neuron.

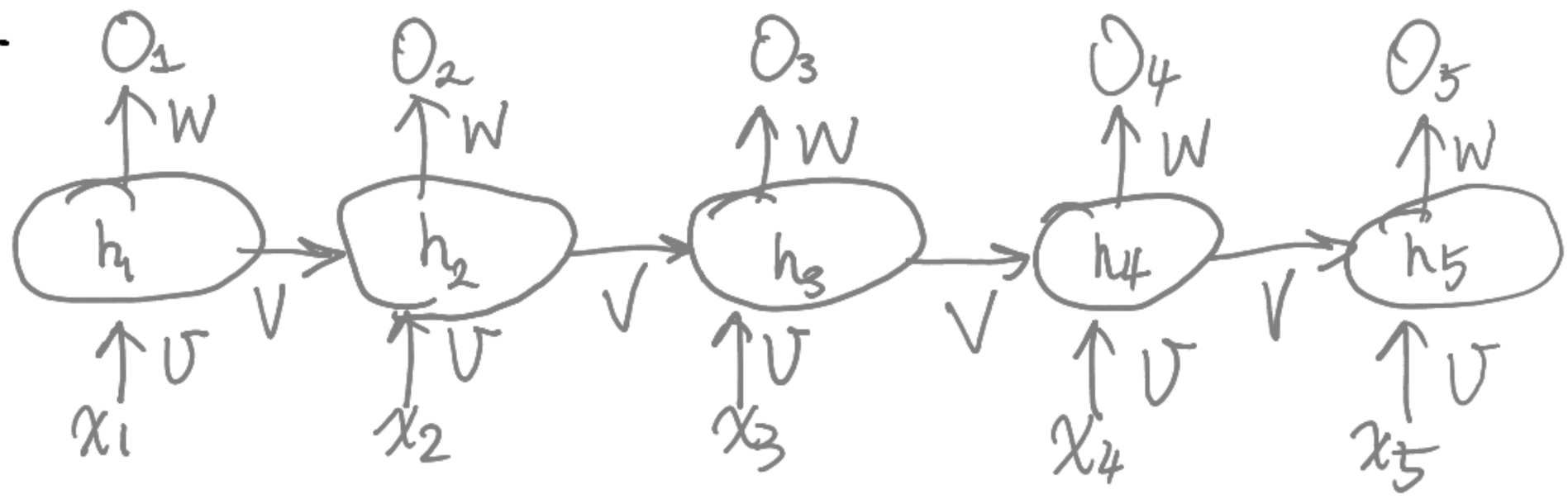
Discussion about Recurrent Neural Net.

So far, we've been talking about feedforward neural net, i.e., the input of the next layer is the output

of the prev. layer.







CS7180 teaches about neural nets

E.g. for sentence classification, only O_5 = label is present (other intermediate outputs are not needed).

To limit our scope, we will focus on feedforward neural net in the class.

1.4. Learning algorithm

MNIST dataset: 60k handwritten digits from 0 to 9.

modified subset of two datasets collected by NIST, the United States' National Institute of Standards and Technology

Loss function.

Quadratic loss function:

$x : 5$, $y^{true} : (0,0,0,0,0,1,0,0,0,0)$

$$loss(x) = \| y(x) - y^{true} \|^2$$

↳ output of the neural net

$$loss\_train = \frac{1}{n} \sum_{i=1}^{n} \| y(x_i) - y_i^{true} \|^2$$

Cross entropy loss function (a.k.a. negative loss likelihood)

$x : 5$, $y^{pred} : (p_0, p_1, \cdots, p_9)$ s.t. $\sum_{i=0}^{9} p_i = 1$

$$Loss(x) = -\log(p_5)$$

Remark. Typically, y^{pred} is obtained from a sigmoid activation function, a.k.a. NN + Softmax

PyTorch (Cross entropy loss)

Softmax — negative log likelihood

Softmax 的 IN $y(x) = (y_0, y_1, \cdots, y_9)$

OUT $\text{Softmax}(y(x)) := \left(\cdots, \frac{\exp(y_i)}{\sum_{j=0}^{9} \exp(y_j)}, \cdots\right)$

The intuition is that is you take the "softmax" of $y(x)$, then you might also select the outcome that is equally likely to "the max".

In PyTorch, $\text{CrossEntropyLoss} := \text{NegativeLogLikelihood}\left(\text{Softmax}(y(x)), y^{\text{true}}\right)$

That is:

- if the input is 5, then CrossEntropyLoss

$$= -\log\left(\frac{\exp(y_5)}{\sum_{j=0}^{9} \exp(y_j)}\right).$$

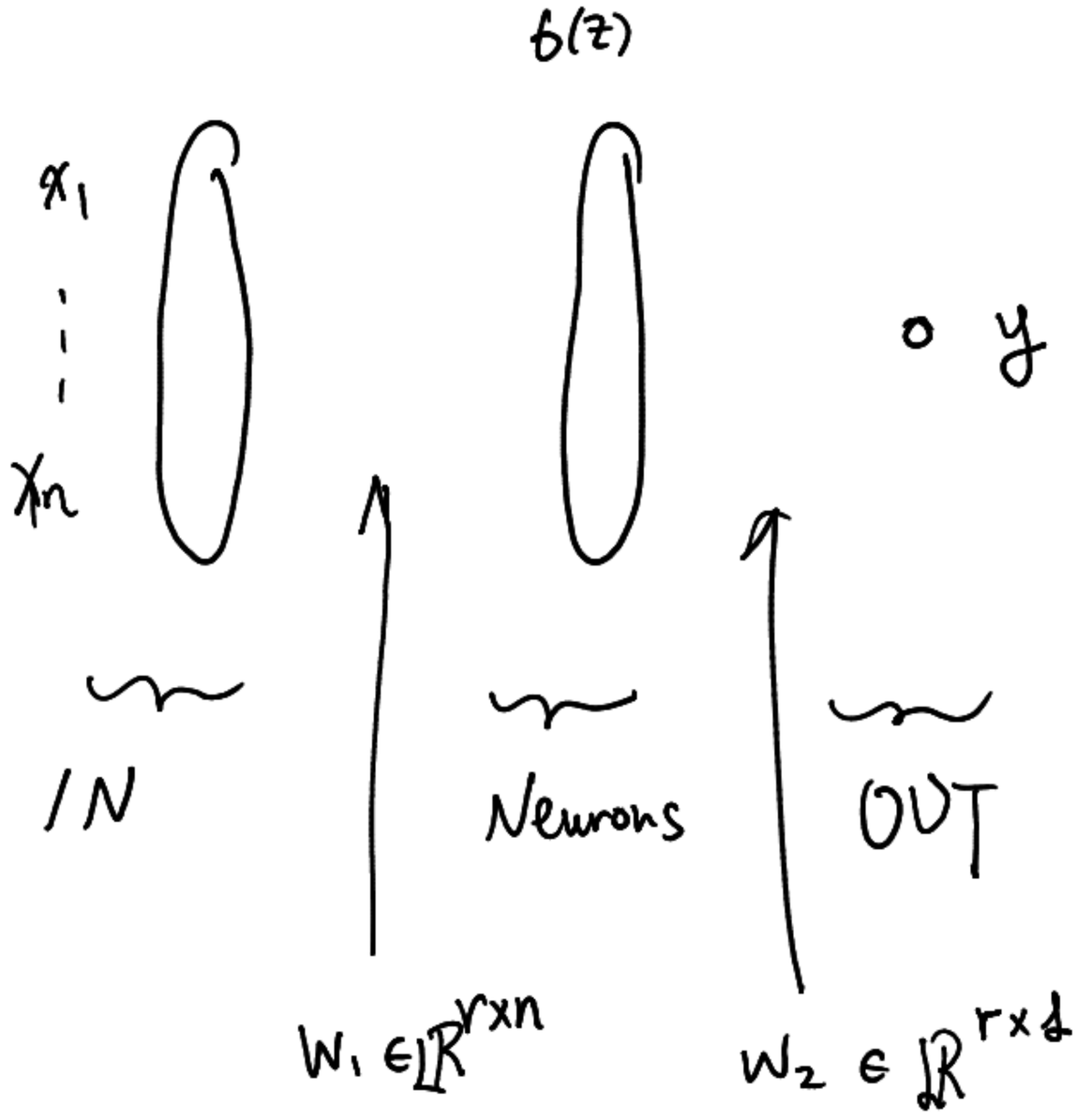
- If there are multiple outputs, then CrossEntropyLoss

e.g. [3 5] $= -\log\left(\frac{\exp(y_3)}{\sum_{j=0}^{9} \exp(y_j)}\right) - \log\left(\frac{\exp(y_5)}{\sum_{j=0}^{9} \exp(y_j)}\right)$.

learning algorithm.

Our objective now becomes:

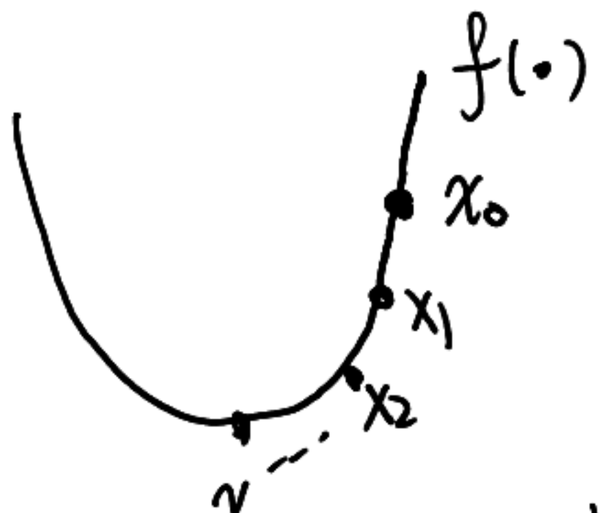
$$\text{Loss}(X, y) = \ell\left(W_2^T \cdot \sigma(W_1 x), y\right)$$



Goal: (Empirical Risk Minimization)

$$\min_{W_1, W_2} \sum_{i \in \text{Training Set}} \text{Loss}(x_i, y_i)$$

Local search algorithm.



$$\min_x f(x)$$

$$x_{i+1} = x_i - \eta \cdot \nabla f(x_i).$$

learning rate, gradient

Ex. $f(x) = (x-1)^2$, $x_0 = 2$.

We have that $\nabla f(x_i) = 2(x_i - 1)$.

Hence $x_{i+1} = x_i - 2\eta(x_i - 1)$.

$$f(x_{i+1}) = (x_{i+1} - 1)^2 = (x_i - 2\eta(x_i - 1) - 1)^2$$

$$= (1 - 2\eta)^2 (x_i - 1)^2$$

$$= (1 - 2\eta)^2 \cdot f(x_i).$$

Initially, $f(x_0) = 1$, therefore

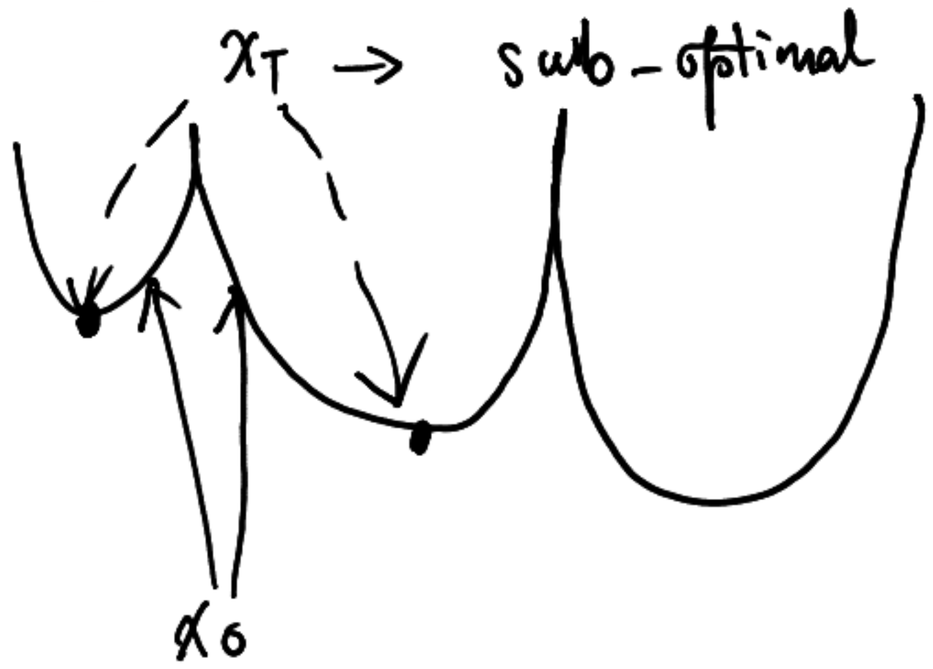
$$\Rightarrow f(x_i) = (1 - 2\eta)^{-2i}$$

So for $f(x_T) \leq \varepsilon$, we want $(1 - 2\eta)^{-2T} \leq \varepsilon$

$$\Leftrightarrow (-2T) \log(1 - 2\eta) \leq \log \varepsilon$$

$$\Leftrightarrow T \geq \frac{\log \frac{1}{\varepsilon}}{2 \log \frac{1}{1 - 2\eta}}.$$

However, in general, the situation is not so nice.



Remark. From this plot you can see why overparametrization is helpful. With multiple random initializations, one of them

will end up in the right "local basin"!

Gradient descent

General form. Initialization : x_0 (say random Gaussian)

Update : $x_{i+1} \leftarrow x_i - \eta \cdot \nabla f(x_i)$

For one 2-layer neural net, we have that

$$f(W_1, W_2) = \sum_{i=1}^{m} \ell\left(W_2^T \cdot \sigma(W_1 \cdot x_i), y_i\right).$$

So we need to calculate :

① $\nabla_{W_1} f(W_1, W_2)$

② $\nabla_{W_2} f(W_1, W_2)$

Batch size, and Stochastic gradient descent.

A computational concern : if m is too large, then we may not want to compute the full gradient !

A practical cure : divide the training dataset into small batches (e.g. 32, 64 etc)

m samples: batch 1, batch 2, ..., batch $\frac{m}{32}$

32 32 32

There are many ways to run stochastic gradient descent.

— Round robin

— Random permutation

It suffices to say that batch size is an important hyper-parameter, and can be quite powerful when tuned properly! We will put out several papers for reading if you are interested in learning more.

Conclusions.

- We learned the basics of a two-layer neural network (a.k.a. multi-layer perceptron)

— Many components to build it up!

& Activation function, loss function, Number of neurons (a.k.a. width)

— How do we implement gradient descent efficiently?