Intro to Neural Networks

CS 6120 Natural Language Processing

Northeastern University

Si Wu

Borrowed some slides from Jurafsky and Martin Chapter 6

Logistics

- The first coding assignment is due midnight
- If you for some reason enroll my session after lecture on Tuesday and didn't take the first quiz, come and see me after class
- Reminders:
    - Bring a pen and student ID to lecture
    - Ask your questions on Ed Discussion since there are many repeated questions

Today:

- We will talk about the basics of neural networks (NN)
- There will be more discussion breaks for you to talk with the people around you and find a partner for your project

Transition from linear to non-linear models

- So far we've learned some linear models
  - Logistic regression
  - Naïve Bayes
- We mentioned single-layer perceptron which is linear
  - It's activation function is a Heaviside step function
  - It's decision boundary is linear

$$H(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- But multi-layer perceptron is non-linear
  - Fully connected neurons with non-linear activation functions
    - Backpropagation requires differentiable functions (ReLU, tanh, sigmoid), and step function is not differentiable!
  - Can distinguish data that are not linearly separable

Transition from linear to non-linear models

- For the rest of the semester, we will talk about non-linear models (FFNN, RNN, transformers) that can generally perform better on more complex pattern recognition tasks
- As you'll see in this lecture, we are still using this general linear combination formula which is what we used for logistic regression
  - Although the activation function is different now

$$z = Wx + b$$

- But should you always choose the more complex model… on all tasks??

Feedforward neural networks

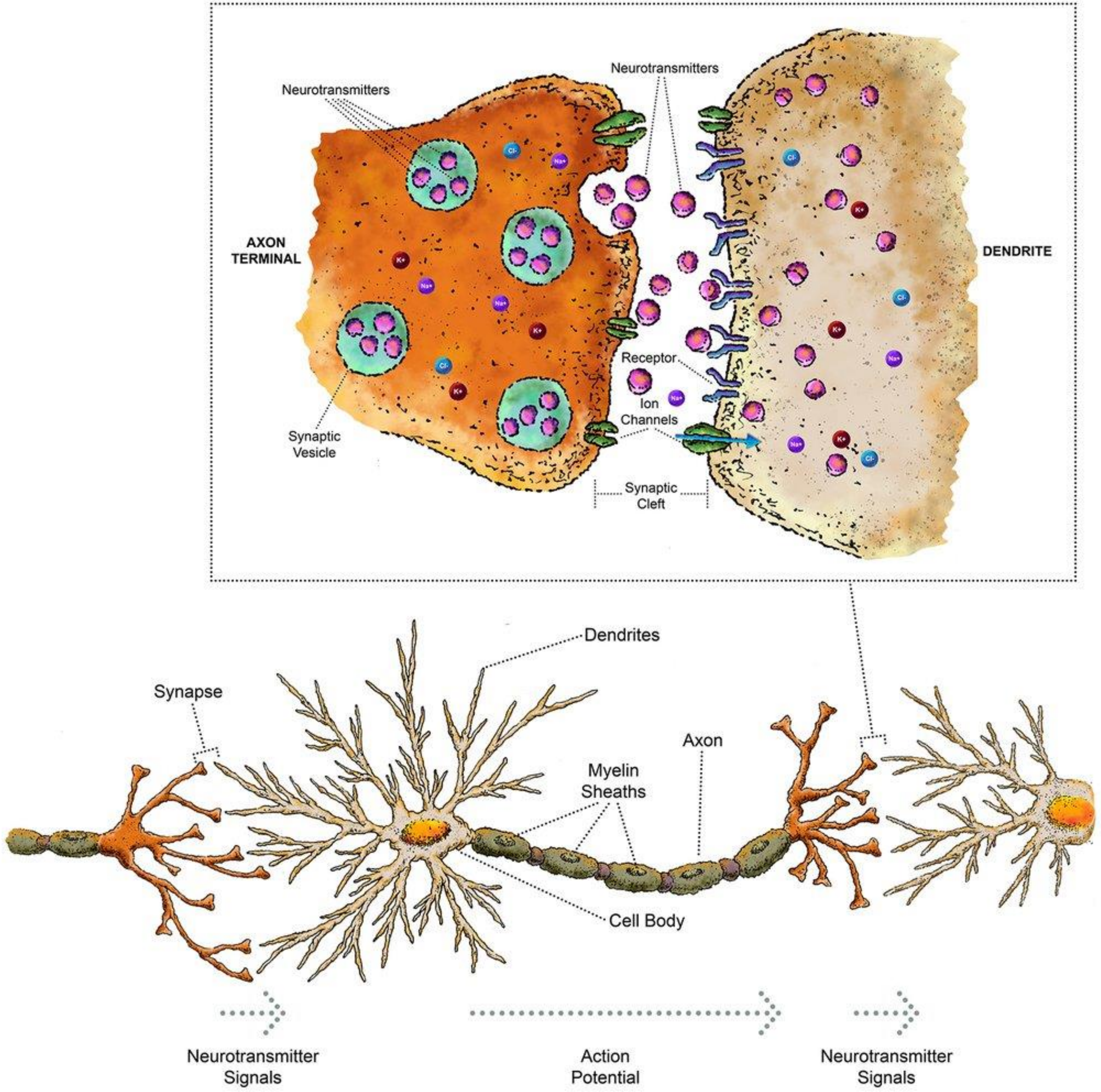


Figure from https://scienceexchange.caltech.edu/topics/neuroscience/neurons

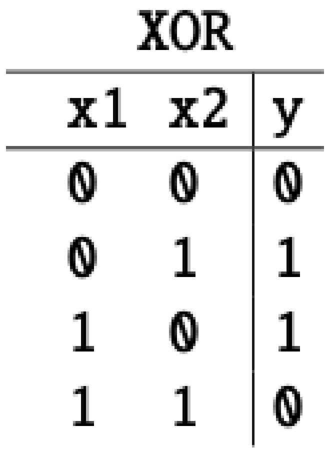
History and background of NNs

- 1943 Warren McCulloch and Walter Pitts proposed a mathematical model of a neuron.
- 1958 Rosenblatt introduced the perceptron but can only solve linearly separable problems.
  - The XOR problem (Minsky and Papert 1969)
- …
- 1986 Rumelhart, Hinton, and William popularized Backpropagation for training FFNNs.
- 80s, CNN
- RNN, LSTM
- Deep learning
  - 2012, AlexNet (Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton)
  - 2014, GAN (Ian Goodfellow et al.)
- Transformer: BERT, GPT, LLM…

- Mathematics, cognitive science, psychology, linguistics, computer vision, NLP… Learning from other fields is often the key to scientific breakthrough!

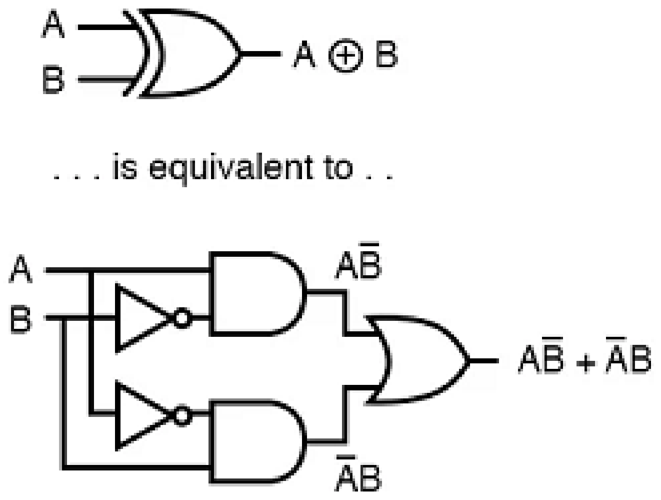
XOR gate

- Exclusive OR

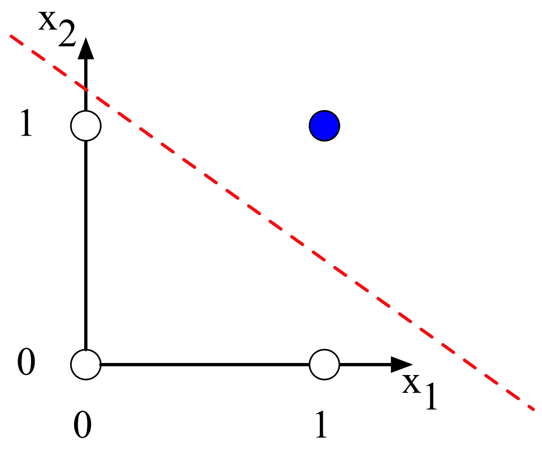


XOR

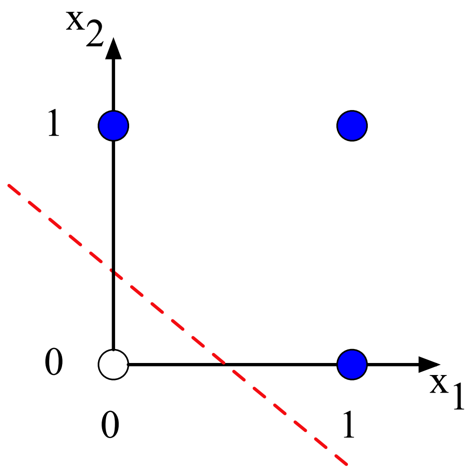
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



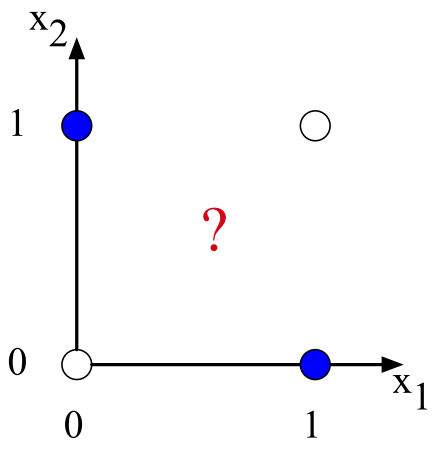
$$A \oplus B = A\bar{B} + \bar{A}B$$



a) x_1 AND x_2



b) x_1 OR x_2



c) x_1 XOR x_2

XOR is not a **linearly separable** function!

XOR gate figure from https://www.allelcoelec.com/blog/XOR-Gate-Explained-Symbol,Truth-Table,Construction-Methods,and-Applications.html?srsltid=AfmBOorLU85xVdi7Yyl7k_GGJMzPeu4xWw0KYg0YsVRQRW1j1oRh0FIR

What’s feedforward neural network?

- **Feedforward neural network**:
  - Input layer → hidden layer(s) → output layer
  - “Feedforward”: only goes forward. No cycles or feedback loops
    - This seems trivial but we will soon learn about recurrent neural networks (RNN) which have feedback loops through hidden states
  - Stacks of fully connected layers: generally, each neuron in a layer is connected to all neurons in the next layers

- **Multi-layer perceptron**: a misnomer of feedforward neural network

Feedforward neural network (FFNN)

- Powerful model
  - Even just one hidden layer, it can learn any functions
  - The non-linearity in FFNNs allows them to perform on more complex classification and language modeling tasks than linear models
- GPU-efficient
  - Fully connected dense layer → matrix multiplications, what GPUs are specialized to do!
  - Parallelizable computation:
    - e.g. you can compute each neuron's output in parallel
- Deep neural networks (DNNs):
  - "**deep**"→ many layers
    - Also where the name "deep learning" is from
  - We will learn more about them: BERT, transformers

Where to get **input features** for FFNN?

- Hand-built, designed features
- Learned features like what we learned from last lecture (on word embeddings)
  - Word2vec (skip-gram)
- In the future, when we learn about neural nets that are deep, we will learn more about how to learn features automatically

The anatomy of a neural network

…Or maybe we should say architecture

The architecture of a neural network

- Components
    - Input layer
    - Hidden layers
        - Each unit has weights and bias
        - Fully connected: each unit is taking the output from all previous units and output goes to all units in the next layers
        - We use a matrix W and a vector b to represent the weights and bias for a single hidden layer
    - Output layer

- We will go over some examples

Non-linear activation function

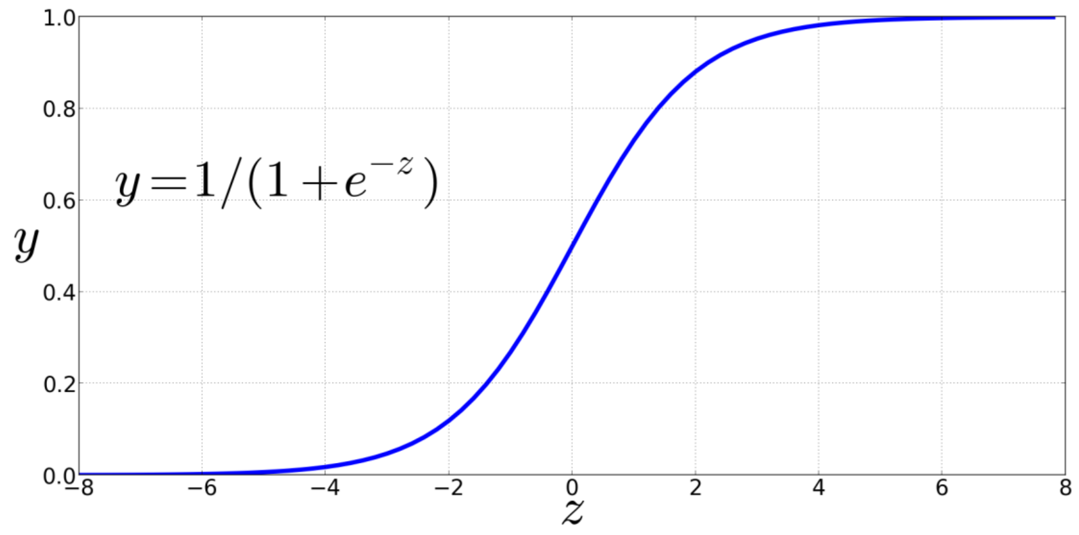
- Sigmoid
  - Differentiable
- The tanh function – a variant of sigmoid but better
  - Ranges from -1 to 1
  - Differentiable
- Rectified linear unit (ReLU)
  - Most commonly used
  - Differentiable everywhere except x=0

The sigmoid function

Recall sigmoid function from previous lecture:

$0 < y < 1$

$$y = s(z) = \frac{1}{1 + e^{-z}}$$



The tanh function

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Range:

-1 < tanh x < 1

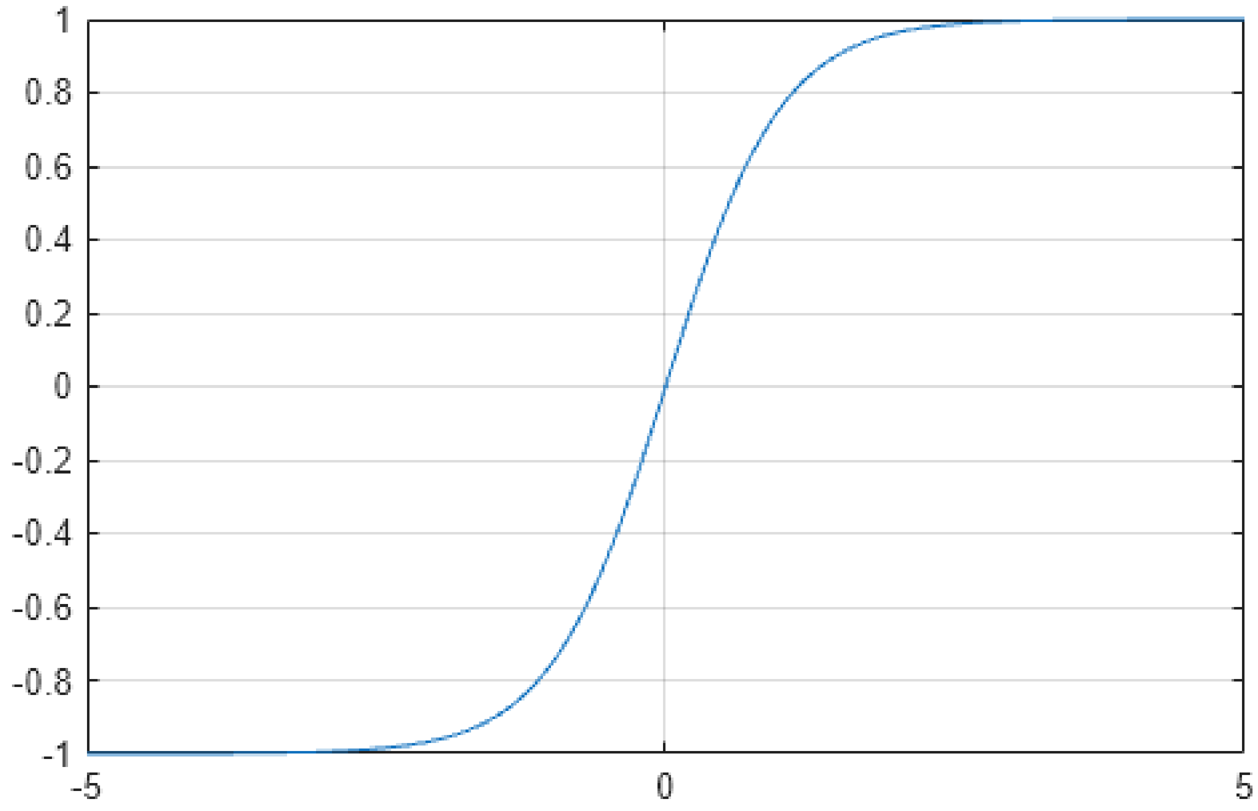


Figure from https://www.mathworks.com/help/matlab/ref/double.tanh.html

The ReLU function

$$\mathrm{ReLU}(x) = x^{+} = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & x \leq 0 \end{cases}$$

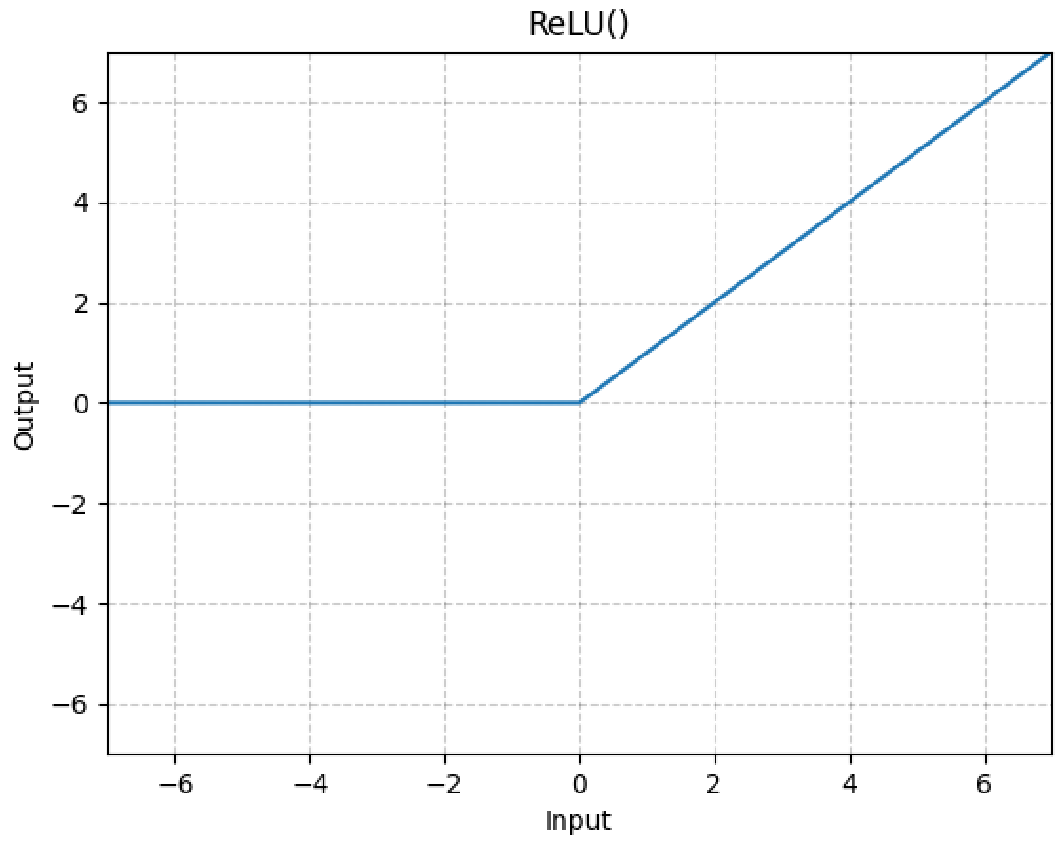


Figure from https://docs.pytorch.org/docs/stable/generated/torch.nn.ReLU.html

where do we apply these activation functions and why do we need them?

- It’s applied at every hidden layer in a FFNN
  - To introduce non-linearity, other wise it will still be a linear model no matter how many you stack!
- Output layer activation function? Depends on the task!
  - Binary classification: sigmoid
  - Multi-class classification: softmax
  - None or linear function: if you need a real number
  - Other functions: depends on the specific tasks, think about the output range of the function

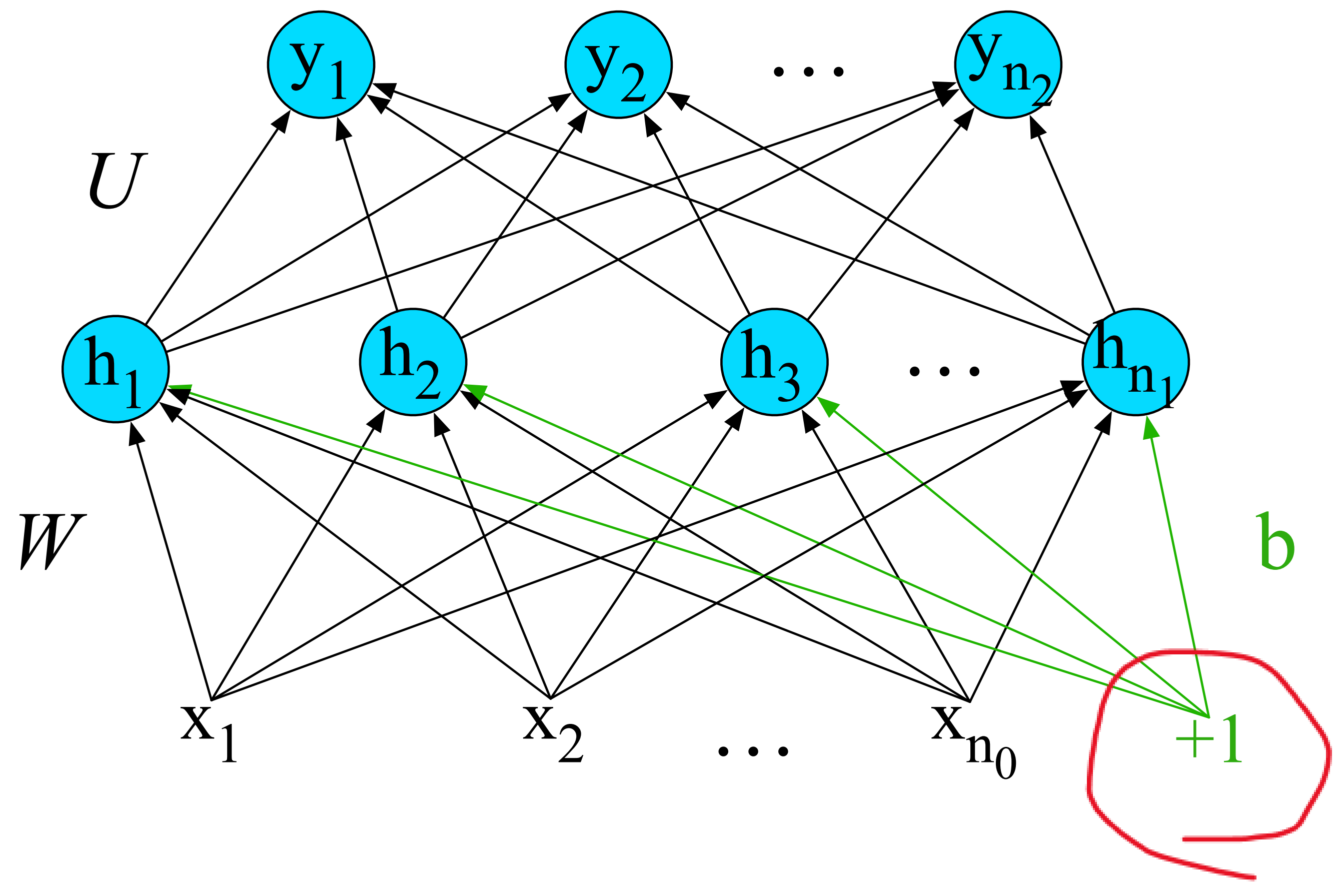
Stacking linear functions without non-linear activation functions

$$\begin{aligned}
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \\
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]}
\end{aligned}$$

$$\begin{aligned}
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]} \\
&= \mathbf{W}^{[2]}(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]} \\
&= \mathbf{W}^{[2]}\mathbf{W}^{[1]}\mathbf{x} + \mathbf{W}^{[2]}\mathbf{b}^{[1]} + \mathbf{b}^{[2]} \\
&\boxed{= \mathbf{W}'\mathbf{x} + \mathbf{b}'}
\end{aligned}$$

Replacing the bias term with a dummy node

Before



$$\mathbf{h}_j = \sigma\left(\sum_{i=1}^{n_0} \mathbf{W}_{ji}\mathbf{x}_i + \mathbf{b}_j\right)$$

Let go over it step by step

Before, we have bias term, so each hidden unit's weighted sum is

$$z = \sum_{i=\underline{1}}^{n} w_i x_i + \underline{b}$$

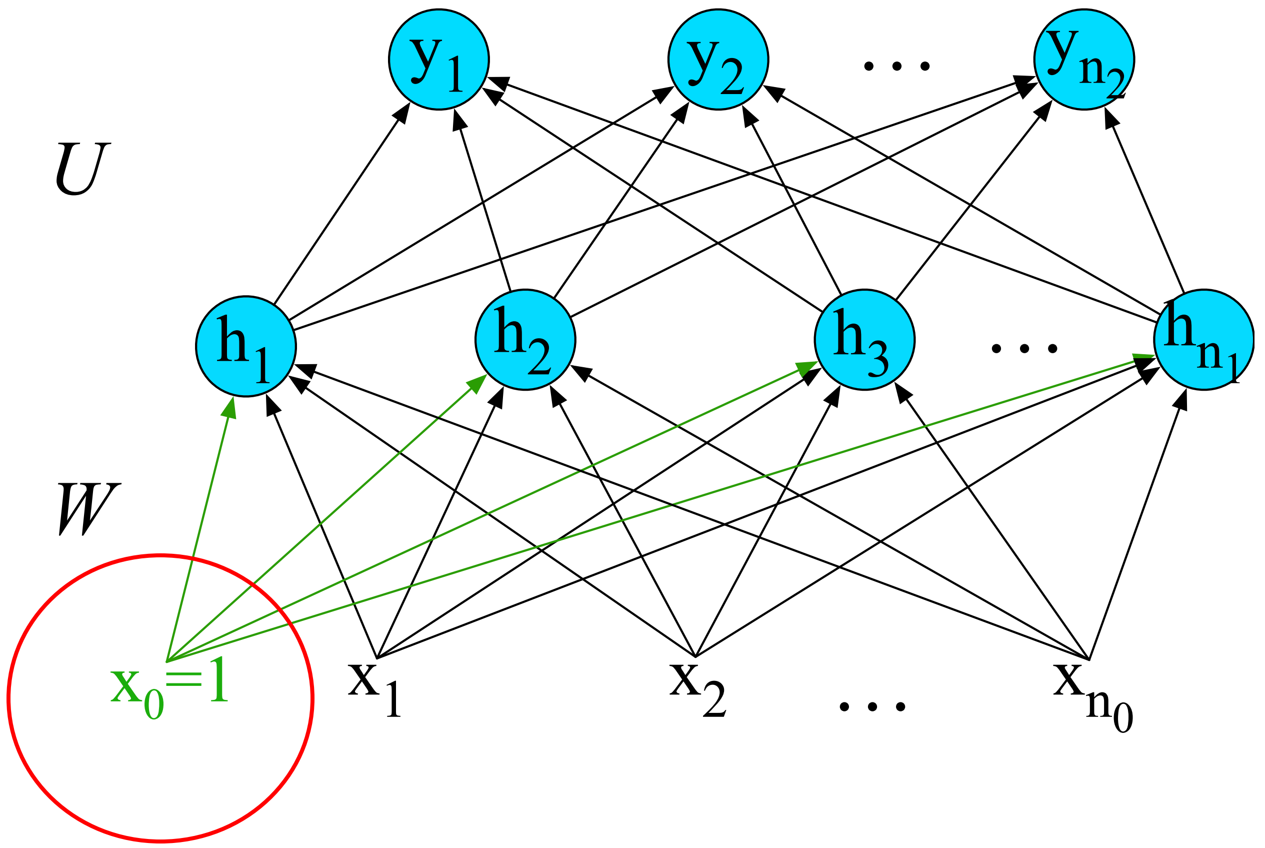
$Now\ Let\ \underline{w_0 = b}\ and\ \underline{x_0 = 1}$

$$z = \underline{w_0} \cdot 1 + \sum_{i=1}^{n} w_i x_i$$

$$\Downarrow$$

$$z = \sum_{i=0}^{n} w_i x_i$$

Replacing the bias term with a dummy node
After

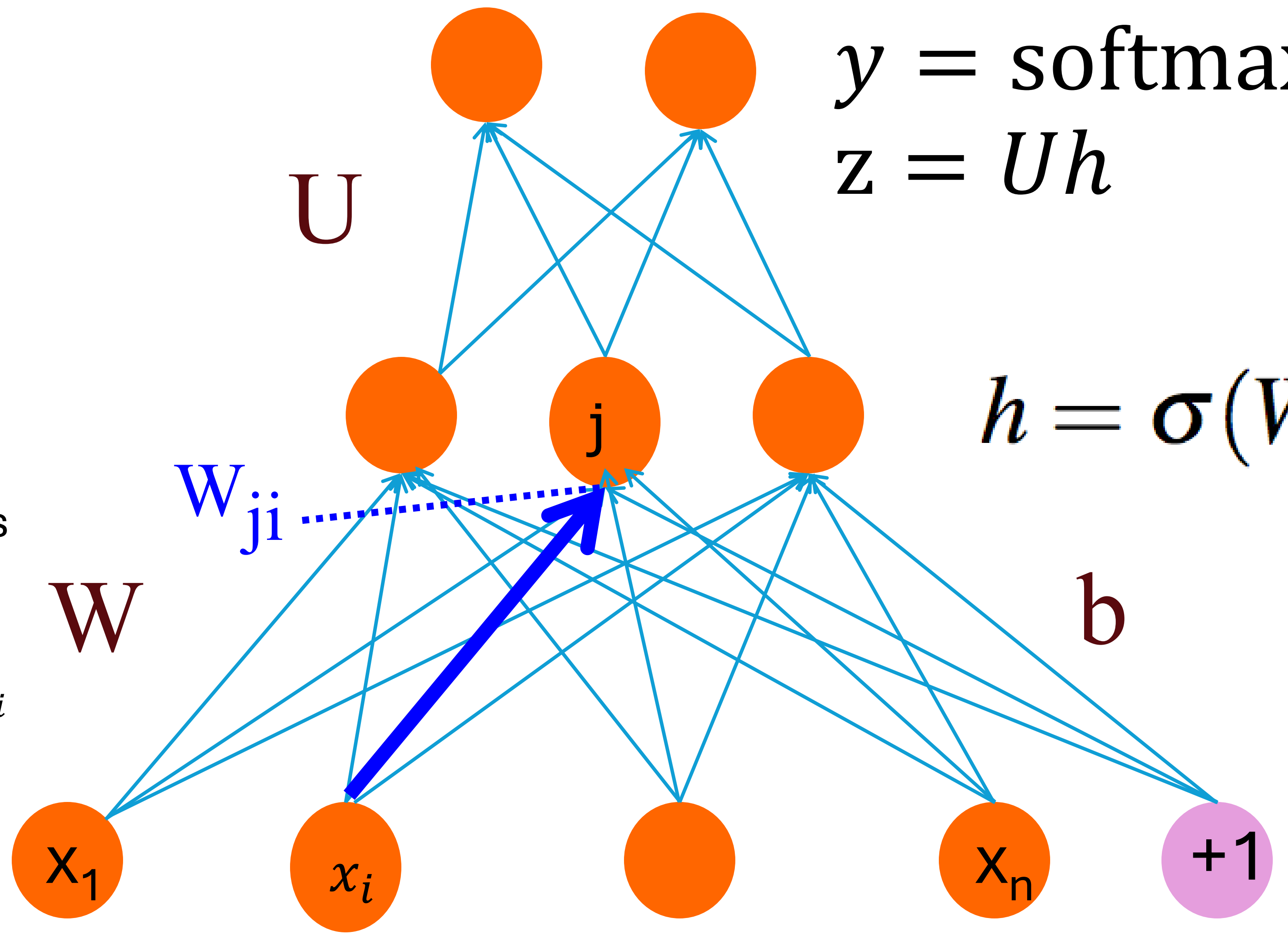


$$\mathbf{h}_j = \sigma\left(\sum_{i=0}^{n_0} \mathbf{W}_{ji}\,\mathbf{x}_i\right)$$

Equations for NN classification with hand features

$$
\begin{aligned}
\mathbf{x} &= [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_d] \quad \text{(each } \mathbf{x}_i \text{ is a hand-designed feature)} \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$

Two-Layer Network with softmax output



$$y = \text{softmax}(z)$$
$$z = Uh$$

$$h = \sigma(Wx+b)$$

Each **row** of W is h_j weights for each input feature

W_{ji} is the weight of input x_i to hidden unit at h_j

n input features, m neuron’s in this layer

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

Each row:
All input features’ weights at this one neuron
Here, it’s neuron position 2

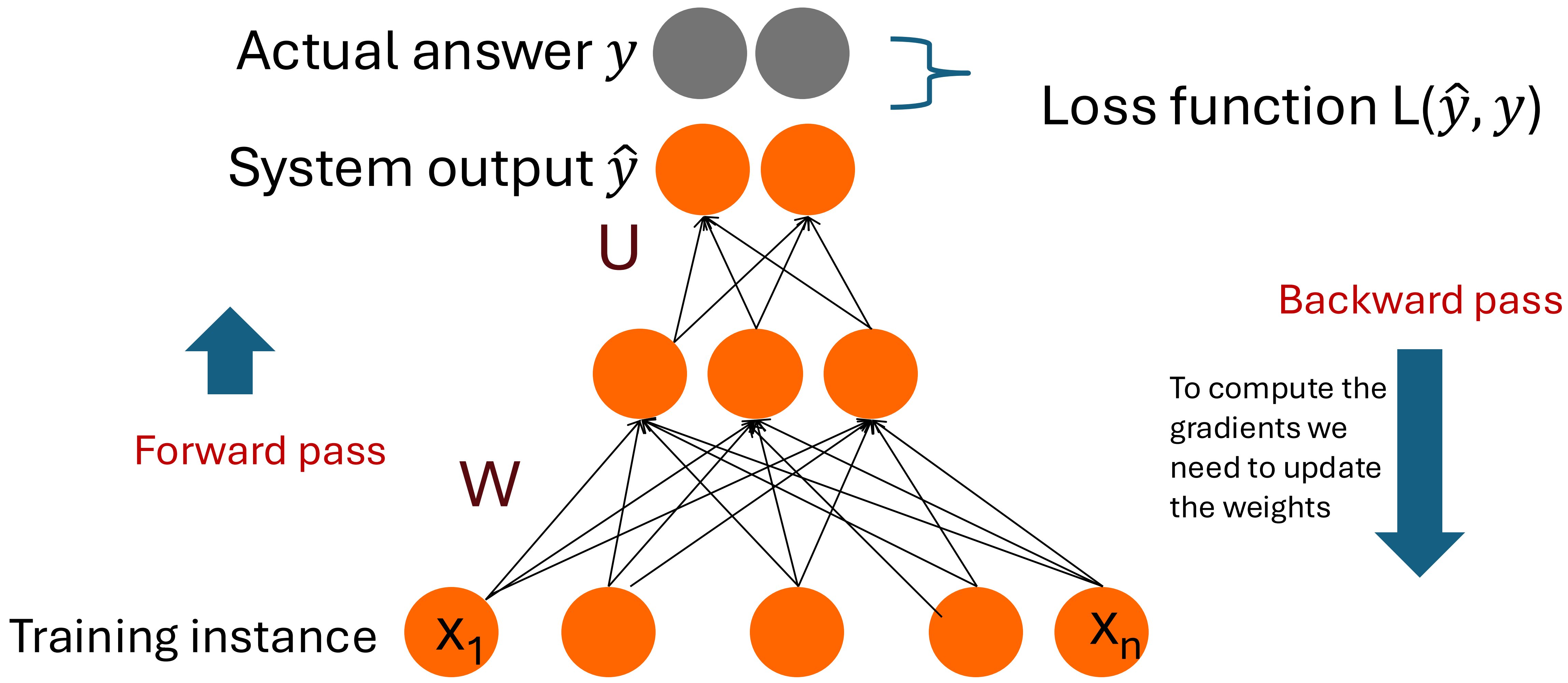
Training neural nets

- Like before, with supervised training we need
  - Loss function: e.g. cross-entropy loss
  - Update weights and bias: gradient descent
    - We need to know the partial derivative of each layer’s weights, that gets complicated!

      → Error backpropagation or backward differentiation

Backpropagation

Intuition: training a 2-layer Network



Intuition: Training a 2-layer network

- For every training tuple (x, y)
  - Run *forward* computation to find our estimate $\hat{y}$
  - Run *backward* computation to update weights:
    - For every output node
    - Compute loss L between true y and the estimated $\hat{y}$
    - For every weight w from hidden layer to the output layer
      - Update the weight
    - For every hidden node
    - Assess how much blame it deserves for the current answer
    - For every weight w from input layer to the hidden layer
      - Update the weight

Vanishing gradient

- During backprop, we compute gradients of the loss wrt weights
- If the net is deep, the gradient is computed via the chain rule
- If the derivative of the activation function is < 1, multiplying many such derivatives will make the final product exponentially smaller!

  →hence the name vanishing!

  For example, sigmoid ranges between 0 to 1, and its derivative is very small, so it’s very bad!
  Tanh is a little better,
  ReLU derivative is either 0 or 1, so better than both sigmoid and tanh

As the gradient approaches 1, weights in earlier layer barely get updated!

Examples of using neural network for NLP tasks

Talk to your neighbors:

What can you use neural networks for?

Use cases for feedforward networks

- Classification:
  - Sentiment analysis
  - Spam detection
  - Word / token classification, PoS tagging
- Language modeling
  - Next word prediction, predicting sequence of words

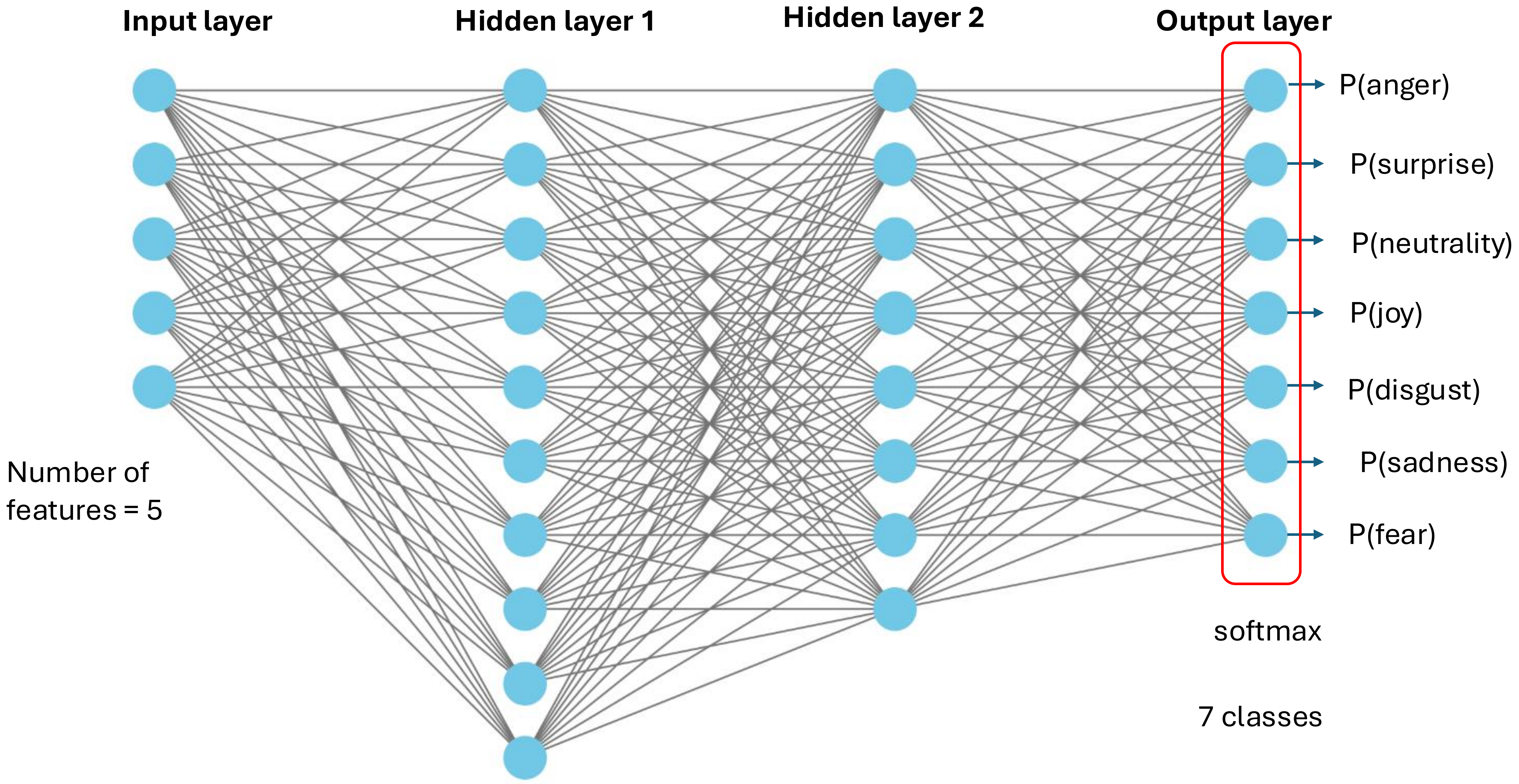
Multiclass classification with neural network

Emotion classification

- Input: text messages
- Desired output: emotion
  - 7 classes: anger, disgust, fear, joy, neutrality, sadness, surprise
- Supervised training: we have the ground truth labels, and split data into train/val/test

What would a neural network look like if we have 2 hidden layers?

(input has 5 features), discuss with your neighbor


Input layer
Hidden layer 1
Hidden layer 2
Output layer
P(anger)
P(surprise)
P(neutrality)
P(joy)
P(disgust)
P(sadness)
P(fear)
Number of features = 5
softmax
7 classes

Why Neural LMs work better than N-gram LMs

- **Training data:**
- We've seen: I have to make sure that the cat gets fed.
- Never seen: dog gets fed
- **Test data:**
- I forgot to make sure that the dog gets ___
- N-gram LM can't predict "fed"!
- Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

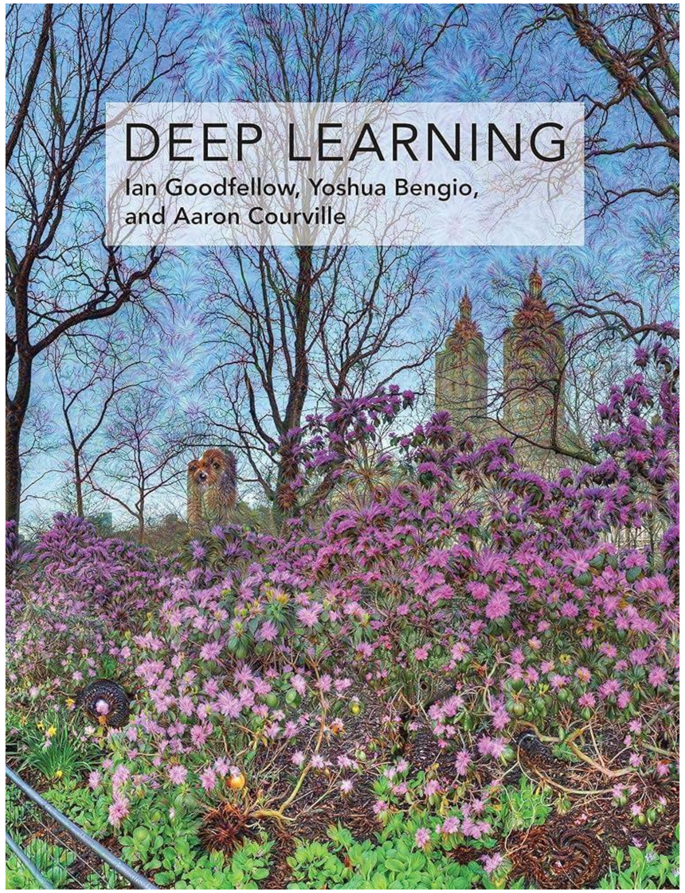
But we still need to handle OOV. Although no smoothing for NN.

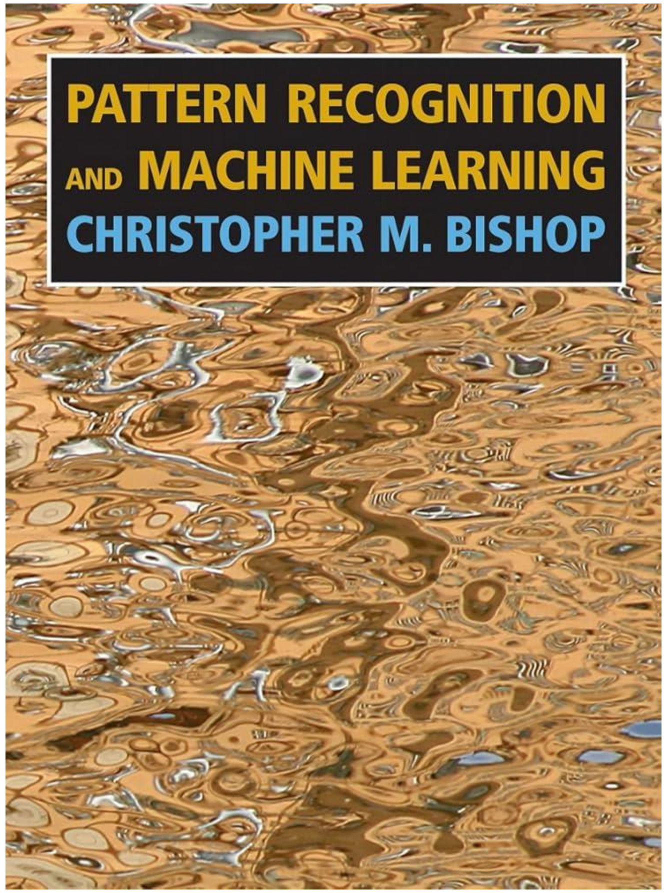
Also NN takes longer to train!

More about neural networks

- We will learn about recurrent neural networks and transformers in future lectures
- And we will talk more about their applications then!

Other textbooks specific to machine learning and deep learning





But we focus more on the applications on language-related tasks in this class!