CS 639: Foundation Models
Machine Learning: Review

Fred Sala

University of Wisconsin-Madison

Jan. 22, 2026

Announcements

- **OH:** Thurs 2:30-4:00 PM in Morgridge 5514
  - Starting **next week**!
- **Resources**
  - https://mlstory.org/ : fun book by Hardt and Recht
- Class roadmap:

Thursday Jan. 22	ML Review
Tuesday Jan. 27	Deep Learning I
Thursday Jan. 20	Deep Learning II
Tuesday Feb. 3	Self-Supervised Learning

Mostly Review

Outline

- **General Supervised Learning Review**
  - Features, labels, function classes, training, generalization
- **Linear Models**
  - Parametrization, training, logistic regression
- **Neural Networks**
  - Perceptrons, setup, training, limitations, multilayer perceptrons, loss functions

Outline

- **General Supervised Learning Review**
  - Features, labels, function classes, training, generalization
- **Linear Models**
  - Parametrization, training, logistic regression
- **Neural Networks**
  - Perceptrons, setup, training, limitations, multilayer perceptrons, loss functions

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances/points $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \to \mathcal{Y}$
  - Maps to output space
- Set of *models* $\mathcal{H} = \{h | h : \mathcal{X} \to \mathcal{Y}\}$
  - a.k.a. *hypothesis/function* class

Get

- Training set of instances for unknown target function,

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$

safe

poisonous

safe

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

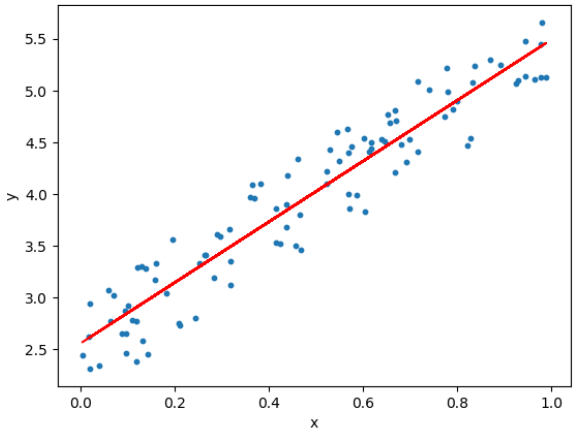
  - Output space:
    - **Binary** $\mathcal{Y} = \{-1, +1\}$ safe poisonous
    - **Continuous** $\mathcal{Y} \subseteq \mathbb{R}$ $13.23°$

Output Space: Classification vs. Regression

Choices of $\mathcal{Y}$ have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**
  - Note: doesn't have to be binary

- Continuous: "**regression**"
  - Example: linear regression



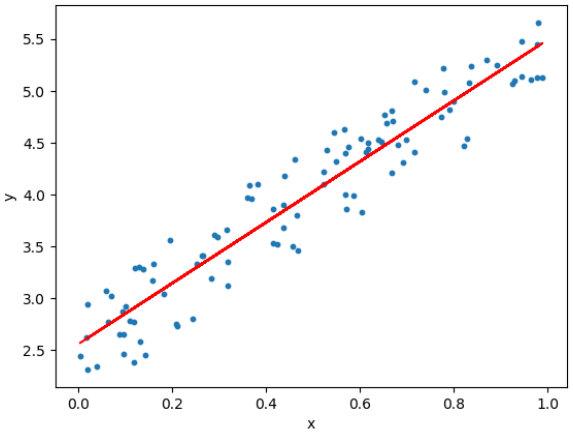
- There are other types…

Hypothesis/Function Classes

We talked about $\mathcal{X}, \mathcal{Y}$ what about $\mathcal{H}$?

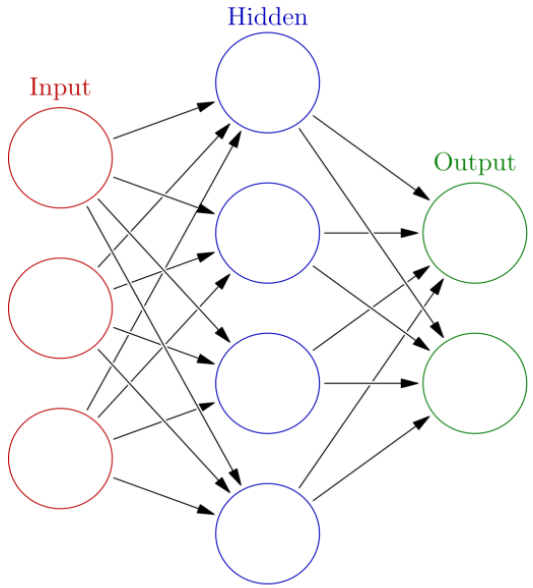
- Pick specific class of models. Ex: **linear models**:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$



- Ex: **feedforward neural networks**

$$f^{(k)}(x) = \sigma(W_k^T f^{(k-1)}(x)))$$



- **Parameters:** θ, w.

Wikipedia

Training The Model

- Training/learning: get data, make the model match that data
- So, get

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$

- Modify f until $f(x^{(i)})$ is close to $y^{(i)}$
  - But our real goal is to make sure we are good at predicting **new** data (not in our training set)
    - This is **generalization**

Training The Model

Modify f until $f(x^{(i)})$ is close to $y^{(i)}$ → **how**?

- Need some way to measure “close”. We’ll use a function called a **loss** (call it l)
- Loss represents how "bad” our model is. We’ll minimize it (i.e., tweak the parameters of f)

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

Best parameters
= best function f

Training The Model

Modify f until $f(x^{(i)})$ is close to $y^{(i)}$ → **how**?

- Choose a loss. Minimize it w.r.t. its parameters
  - Ex: linear models:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

Best parameters
= best function f

Linear model class f

$$= \frac{1}{n} \sum_{i=1}^{n} \ell(\theta_0 + x_i^T \theta, y_i)$$

Square loss

$$= \frac{1}{n} \sum_{i=1}^{n} (\theta_0 + x_i^T \theta - y_i)^2$$

Train vs **Test**

Now we've trained, have some f parametrized by θ

- Train loss is small → f predicts most x_i correctly
- How does f do on points not in training set? **"Generalizes"?**
- To evaluate this, reserve a **test** set. Do **not** train on it!

$$\underbrace{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)}_{\text{Training Data}} \quad \underbrace{(\mathbf{x}_{n+1}, y_{n+1}), \ldots, (\mathbf{x}_{n+p}, y_{n+p})}_{\text{Test Data}}$$

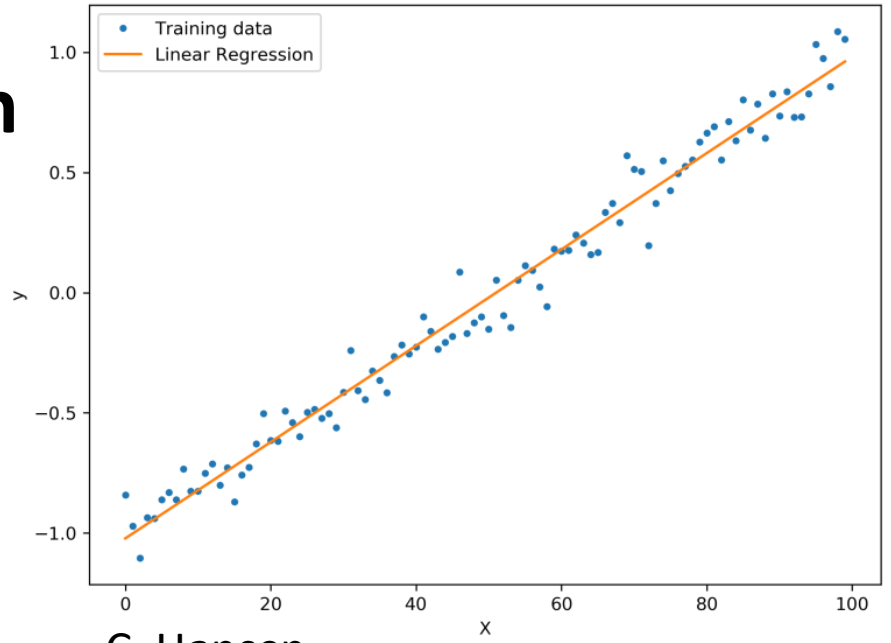
Break & Questions

Outline

- **General Supervised Learning Review**
  - Features, labels, function classes, training, generalization
- **Linear Models**
  - Parametrization, training, logistic regression
- **Neural Networks**
  - Perceptrons, setup, training, limitations, multilayer perceptrons, loss functions

Back to **Linear Models**

Simplest type of model. Let's do **regression**



C. Hansen

- **Inputs**: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
  - *x*'s are vectors, *y*'s are scalars.
  - "**Linear**": predict a linear combination of x components + intercept

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d = \theta_0 + x^T \theta$$

- **Want**: parameters θ

Linear Regression Setup

Problem Setup

- Goal: figure out how to minimize square loss
- Let's organize it. Train set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
  - Since $f(x) = \theta_0 + x^T\theta$, use a notational trick by augmenting feature vector with a constant dimension of 1: $x = \begin{bmatrix} 1 \\ x \end{bmatrix}$
  - Then, with this one more dimension we can write (θ contains θ_0 now)

$$f(x) = x^T\theta$$

Linear Regression Setup

Problem Setup

- Train set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
- Take train features and build a *n* x (*d*+1) matrix, and make *y* a vector:

$$X = \begin{bmatrix} x_1^T \\ \ldots \\ x_n^T \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \ldots \\ y_n \end{bmatrix}$$

- Then, the loss we seek to minimize is

$$\frac{1}{n} \|X\theta - y\|^2$$

Finding The Estimated Parameters

Have our loss: $\frac{1}{n}\|X\theta - y\|^2$

- Could optimize it with SGD, or other optimization approaches
  - This is what we'll do to **train our models**
- But the minimum also has a closed-form solution

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Hat: indicates an estimate / a learned quantity

Not always invertible…

"Normal Equations"

Linear Regression → Classification?

What if we want the same idea, but *y* is 0 or 1?

- Need to convert the $\theta^T x$ to a probability in [0,1]

Logistic function

$$p(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Why does this work?

- If $\theta^T x$ is really big, $\exp(-\theta^T x)$ is really small → *p* close to 1
- If really negative exp is huge → *p* close to 0

"Logistic Regression"

Break & Questions

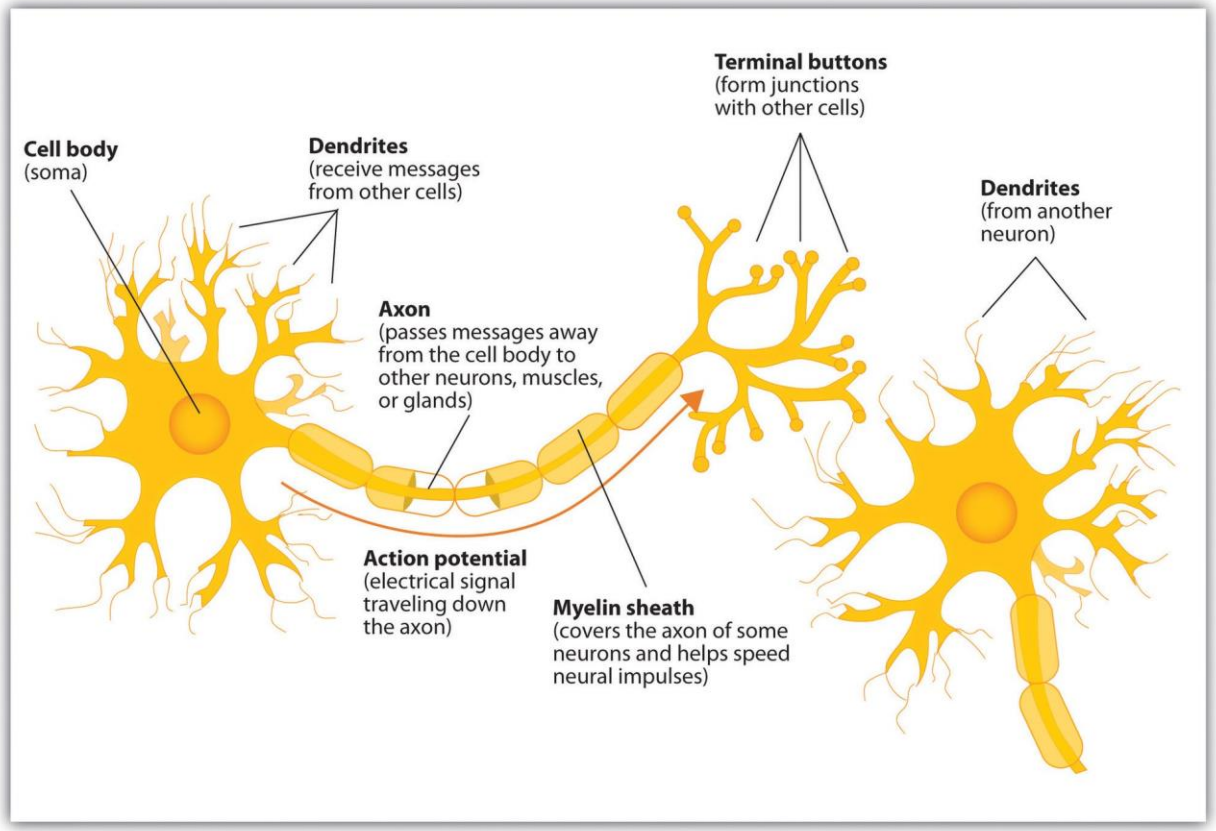
Outline

- **General Supervised Learning Review**
  - Features, labels, function classes, training, generalization
- **Linear Models**
  - Parametrization, training, logistic regression
- **Neural Networks**
  - Perceptrons, setup, training, limitations, multilayer perceptrons, loss functions

Neural Networks

- (Loosely) inspired by human brains
- Networks of simple and homogenous units

(wikipedia)

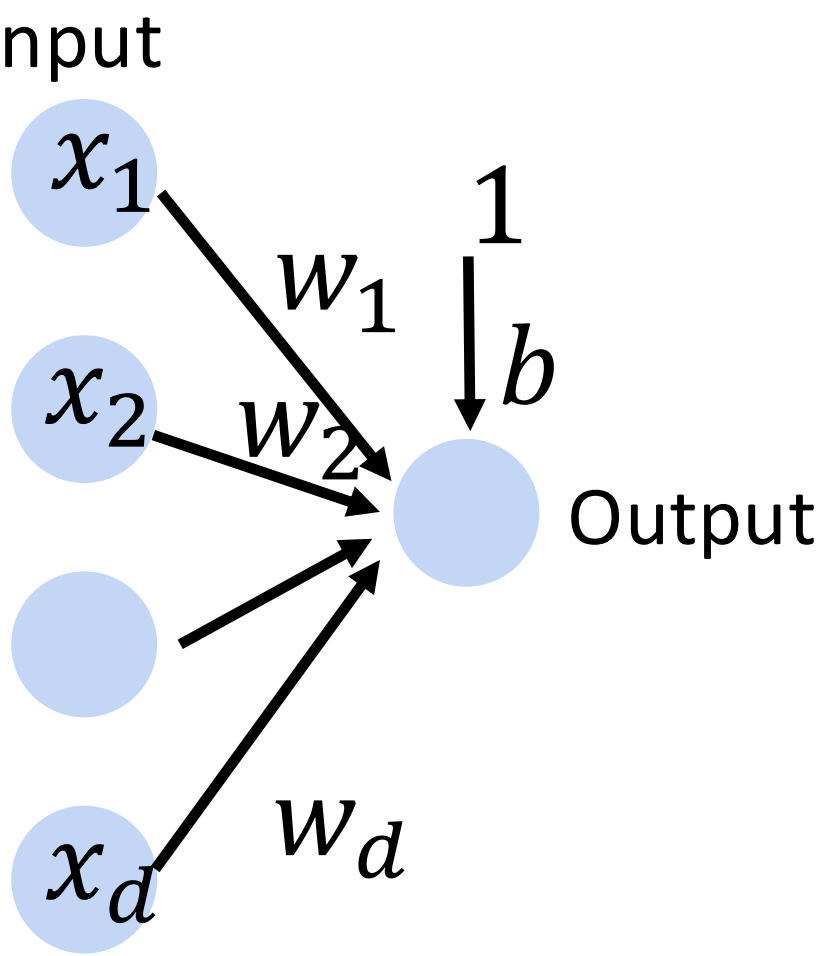


Linear **Perceptron**

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$f = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

Cats vs. dogs?



Perceptron

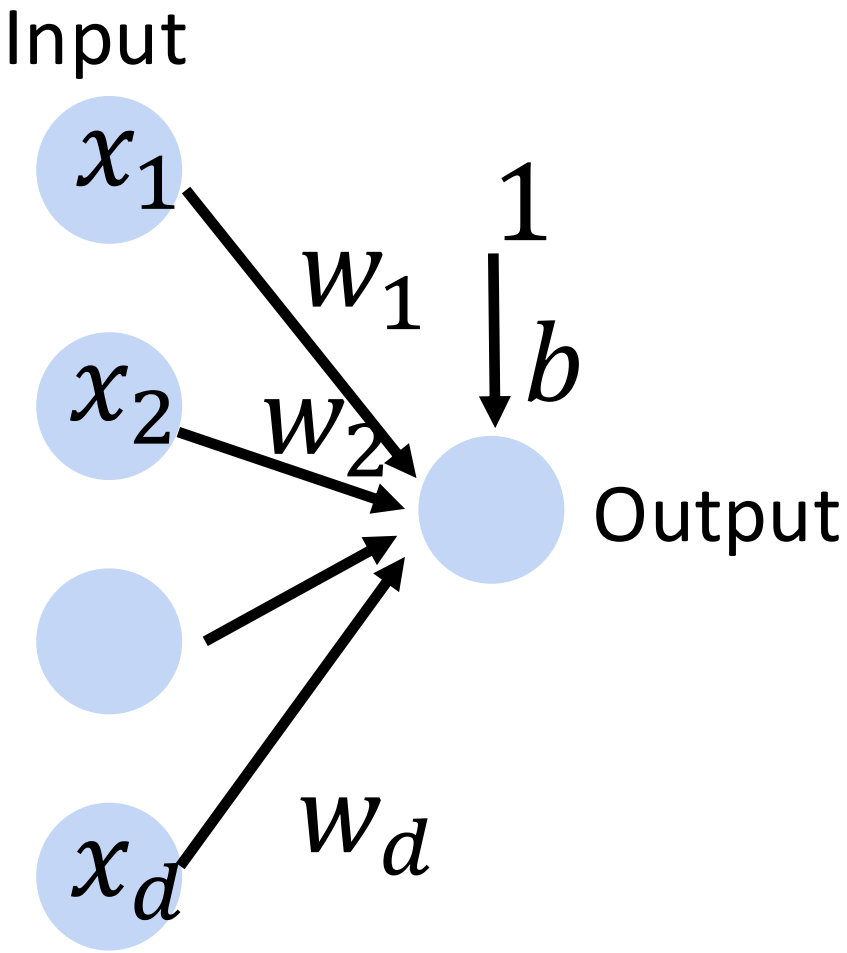
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

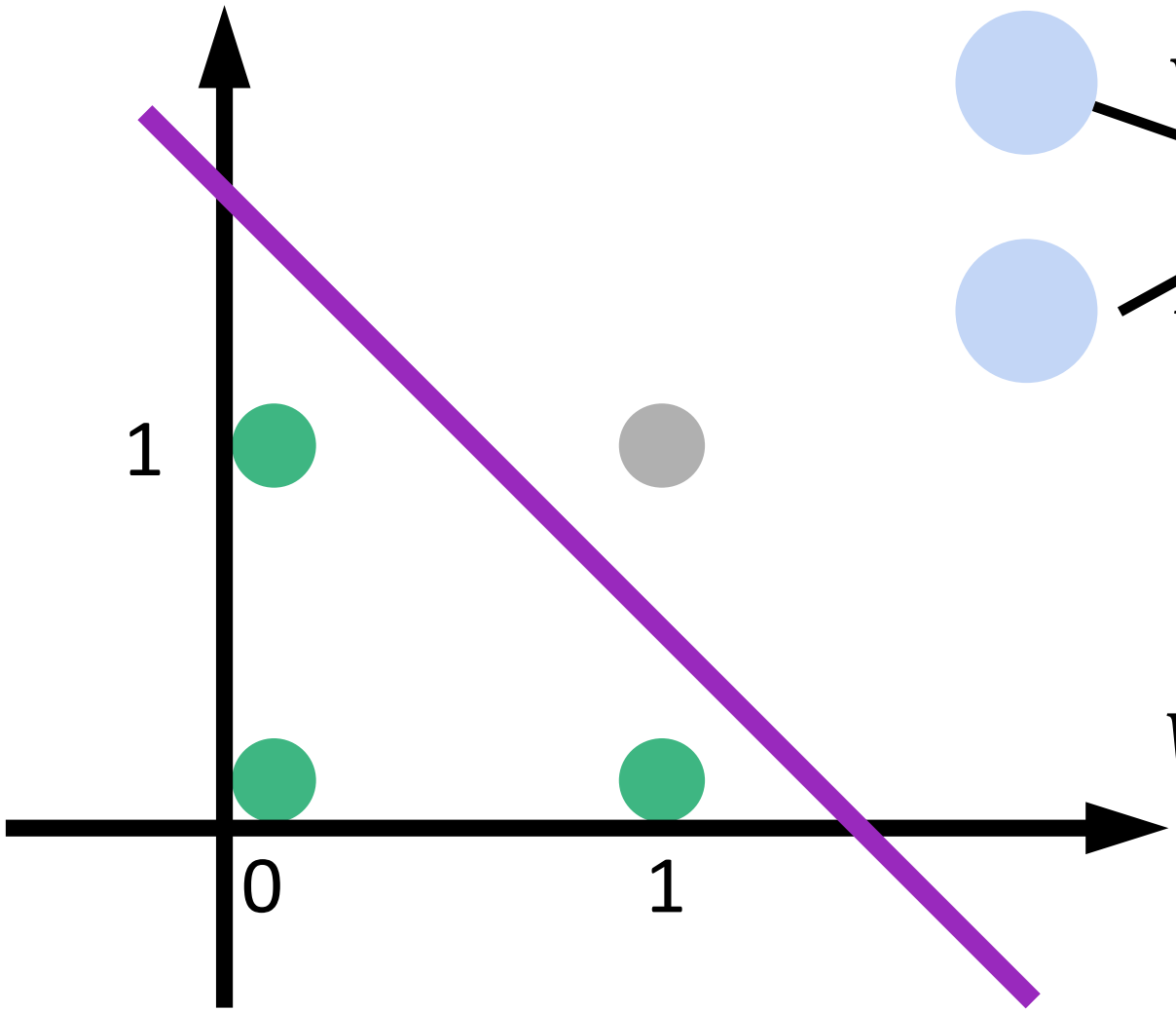
Activation function

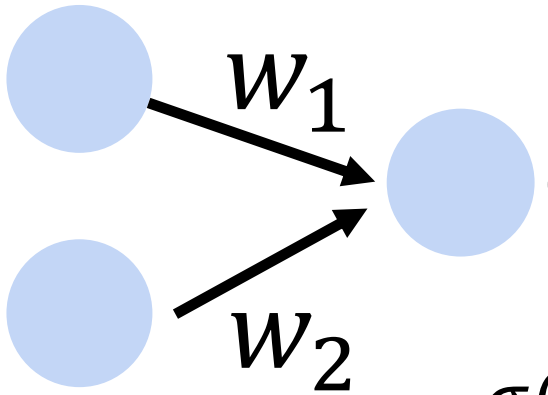
Cats vs. dogs?



Learning **AND** function using perceptron

The perceptron can learn an AND function





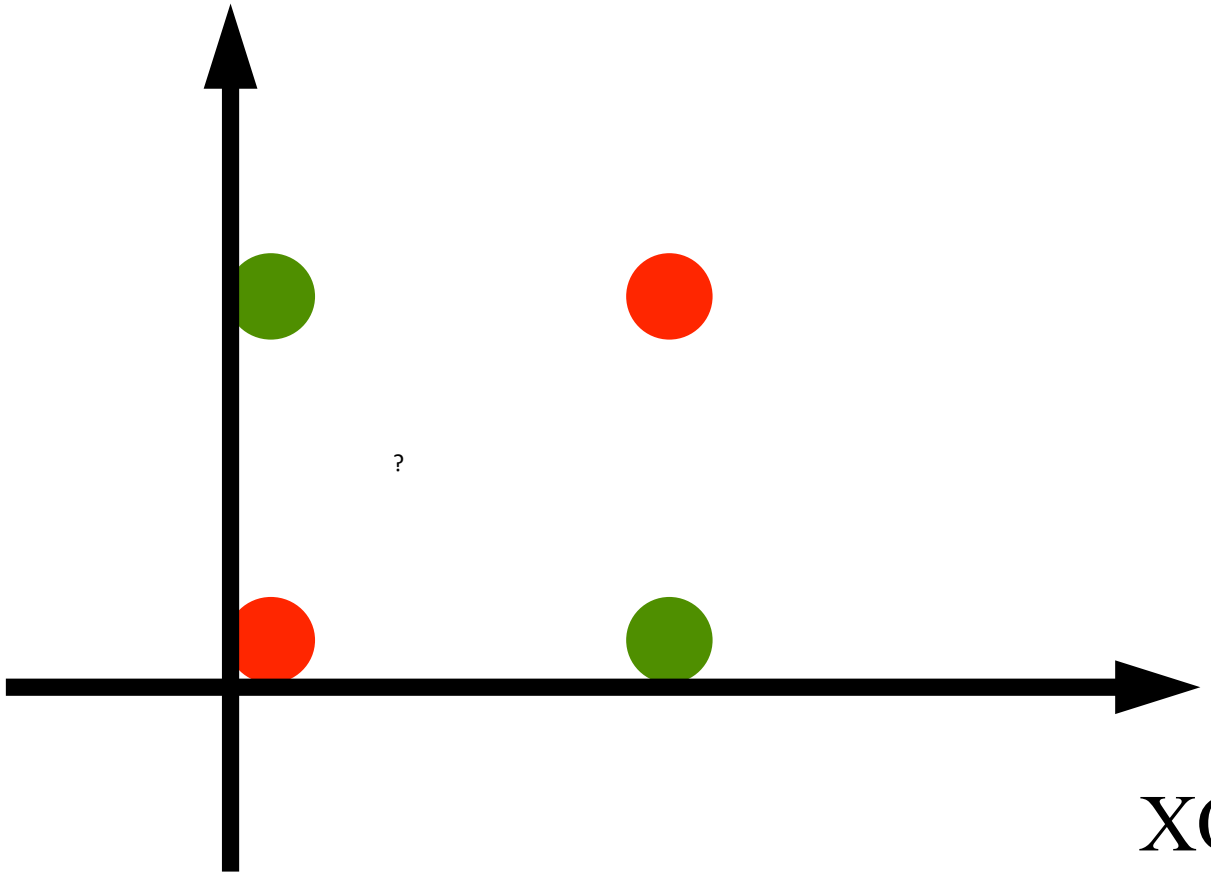
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -1.5$$

The limited power of a single neuron

The perceptron cannot learn an **XOR** function
(neurons can only generate linear separators)



$$x_1 = 1, x_2 = 1, y = 0$$

$$x_1 = 1, x_2 = 0, y = 1$$

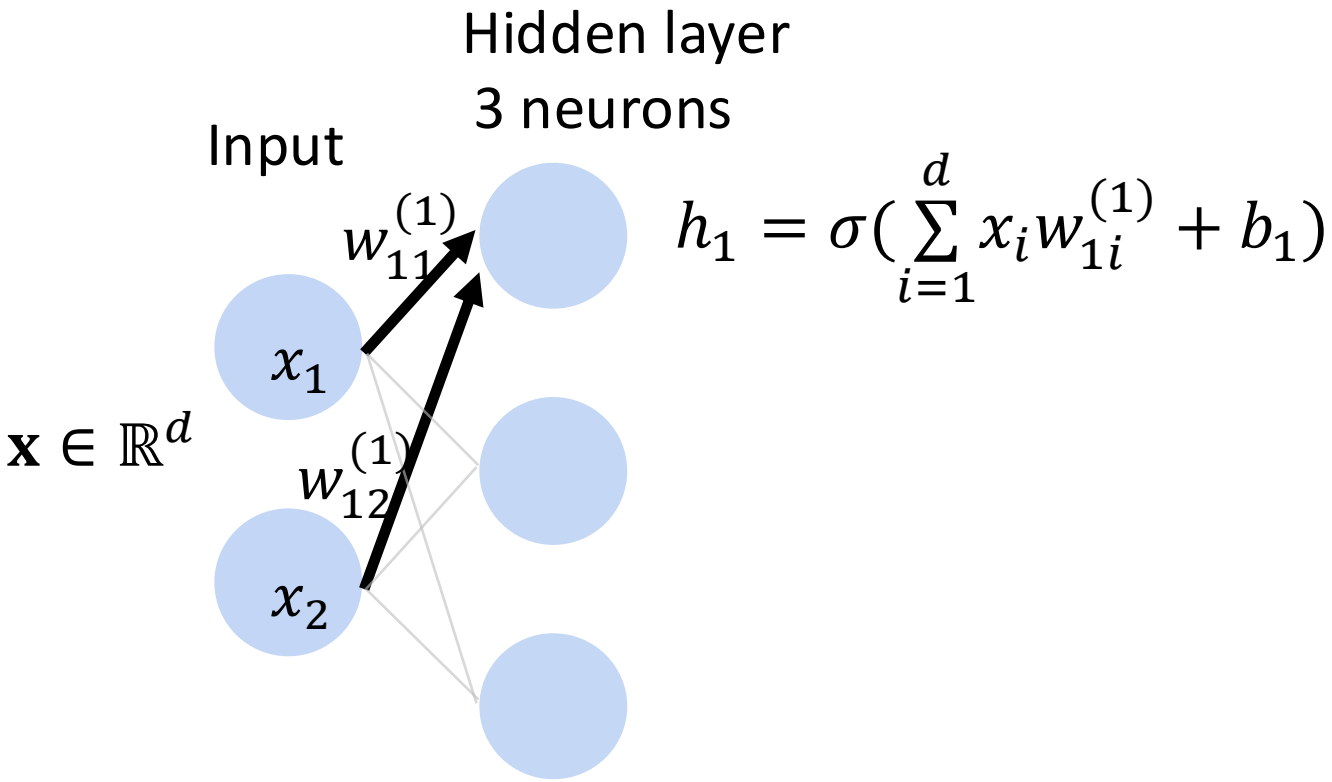
$$x_1 = 0, x_2 = 1, y = 1$$

$$x_1 = 0, x_2 = 0, y = 0$$

$$\mathrm{XOR}(x1,x2) = (x1 \wedge \neg x2) \vee (\neg x1 \wedge x2)$$

Multi-layer perceptron: Example

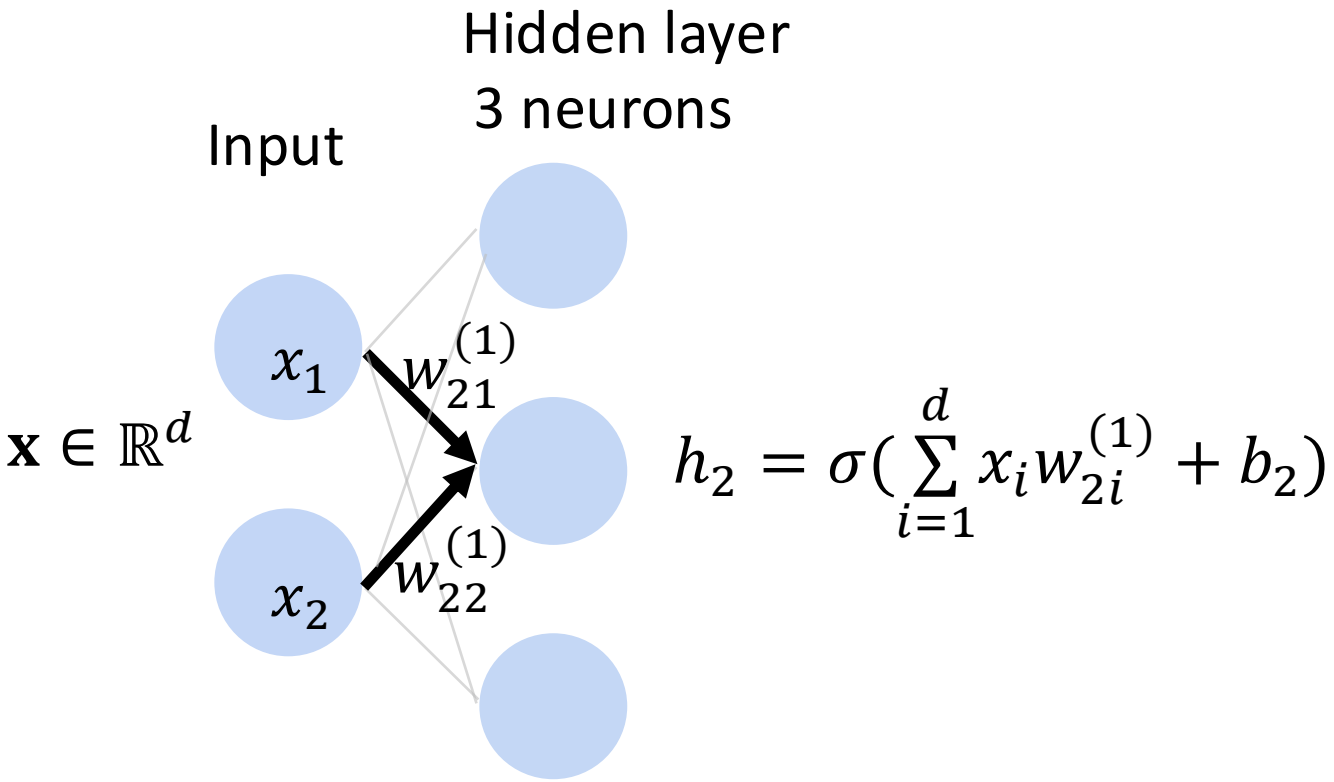
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1)$$

Multi-layer perceptron: Example

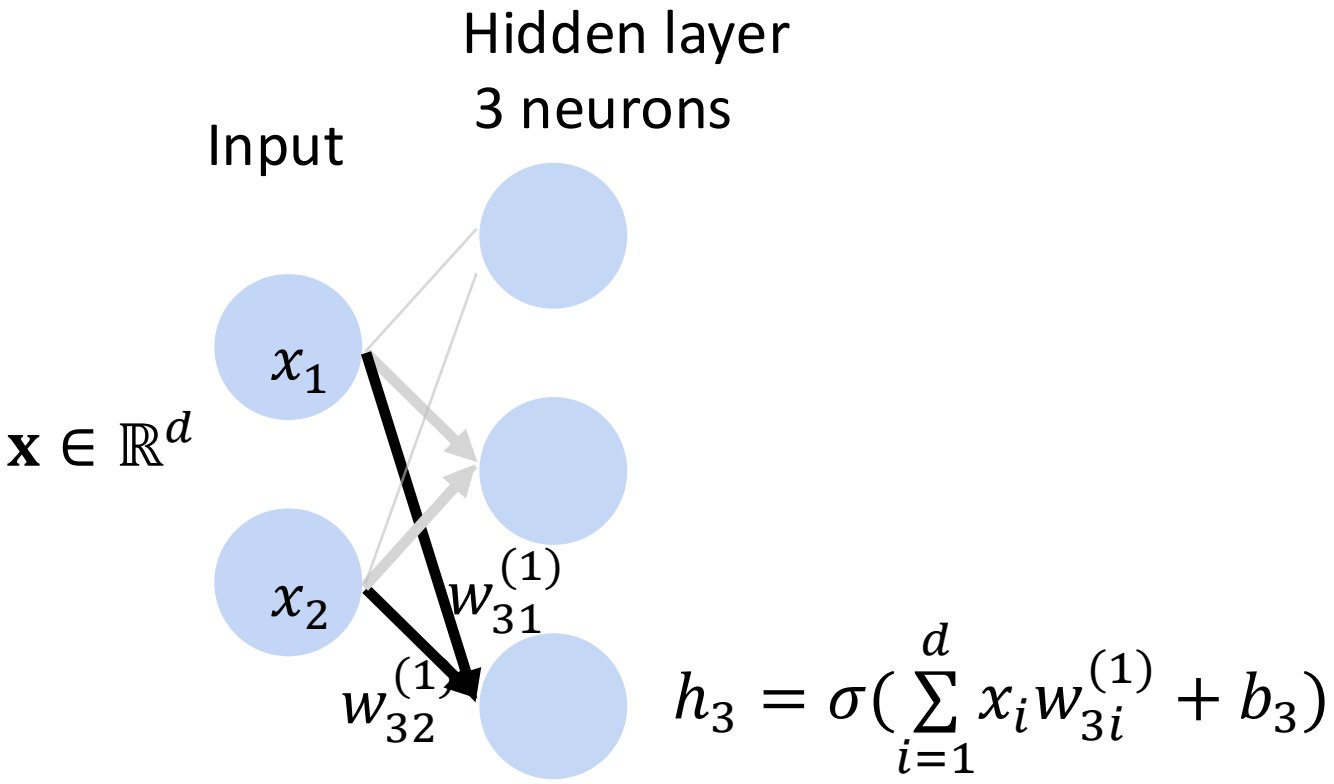
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_2 = \sigma\left(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2\right)$$

Multi-layer perceptron: Example

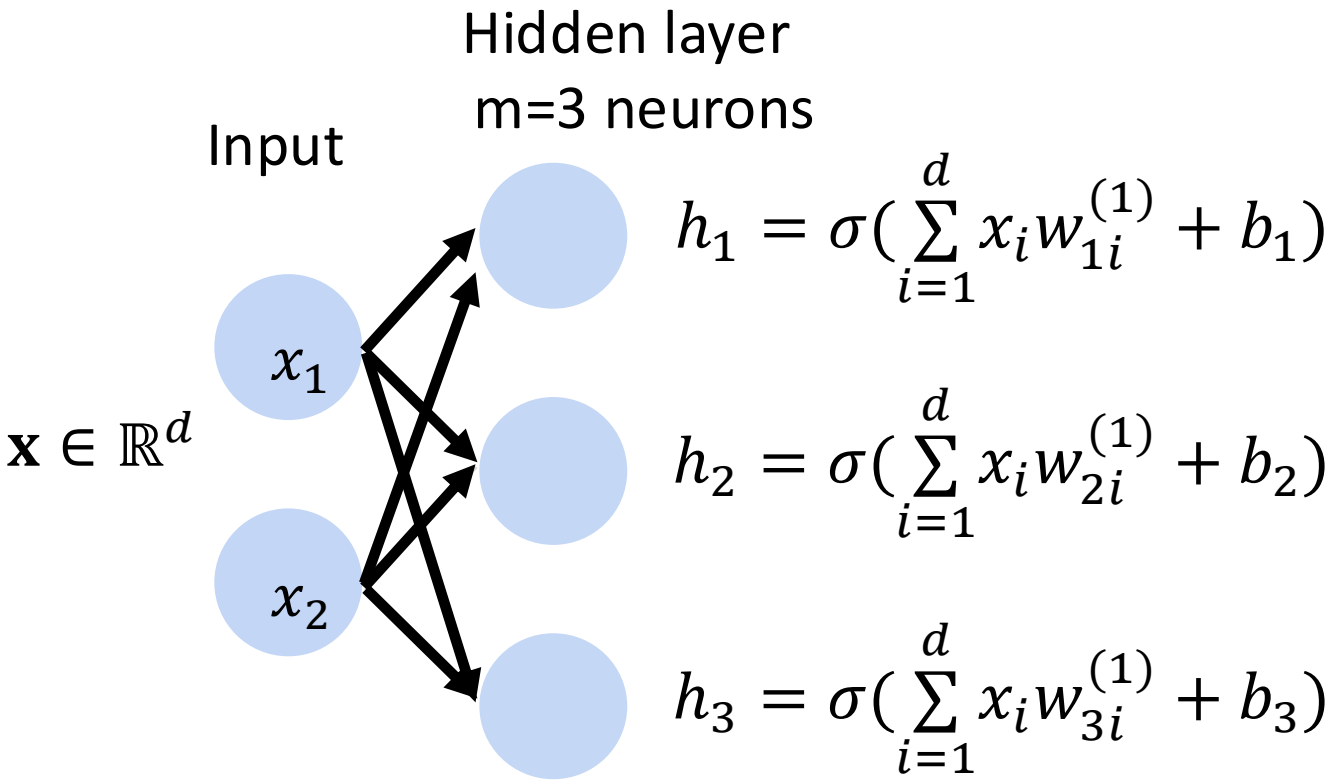
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_3 = \sigma\left(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3\right)$$

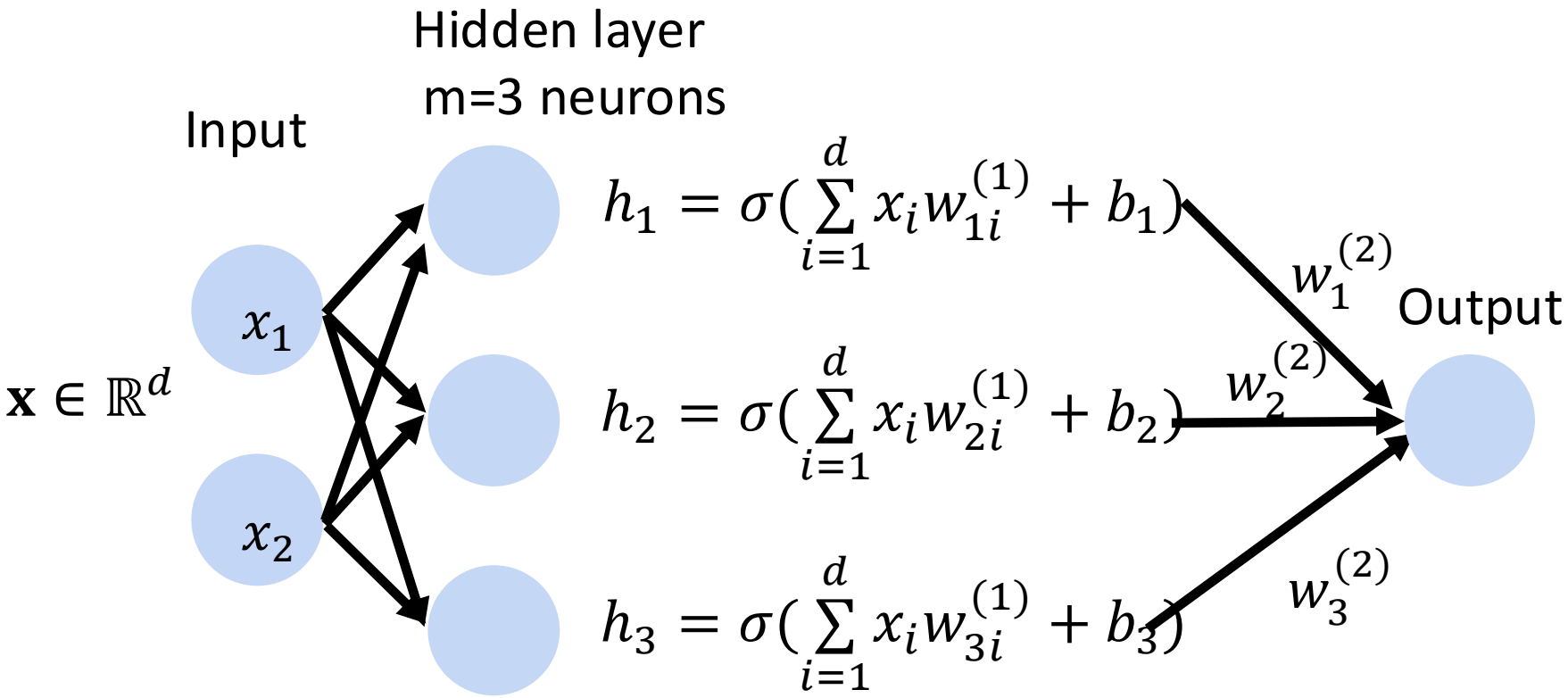
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



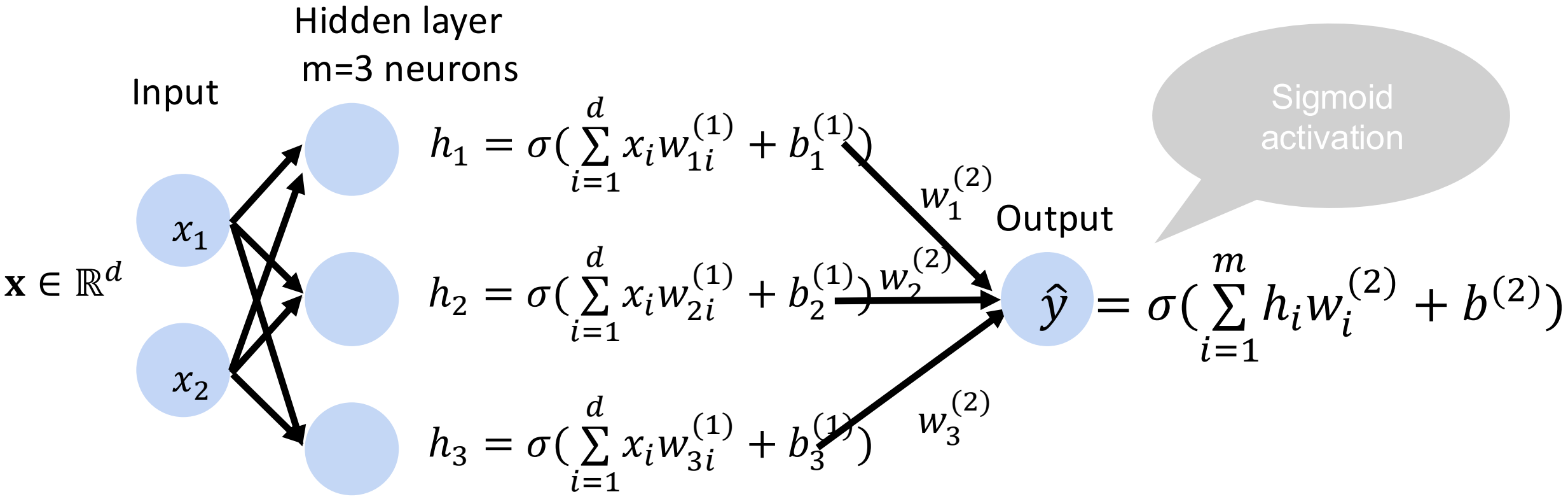
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2

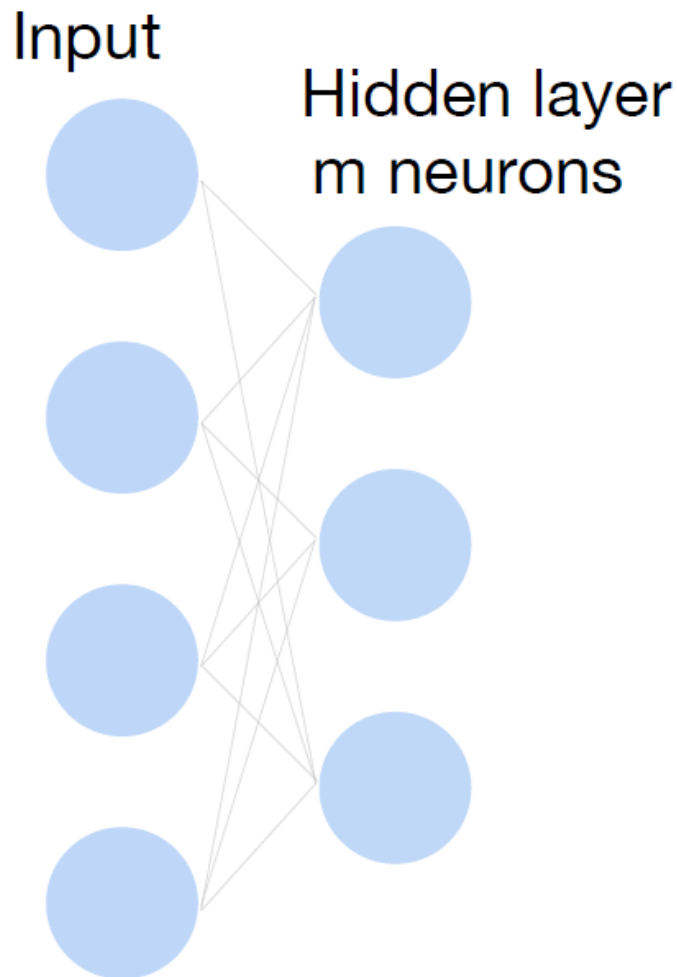


Multi-layer perceptron: Matrix Notation

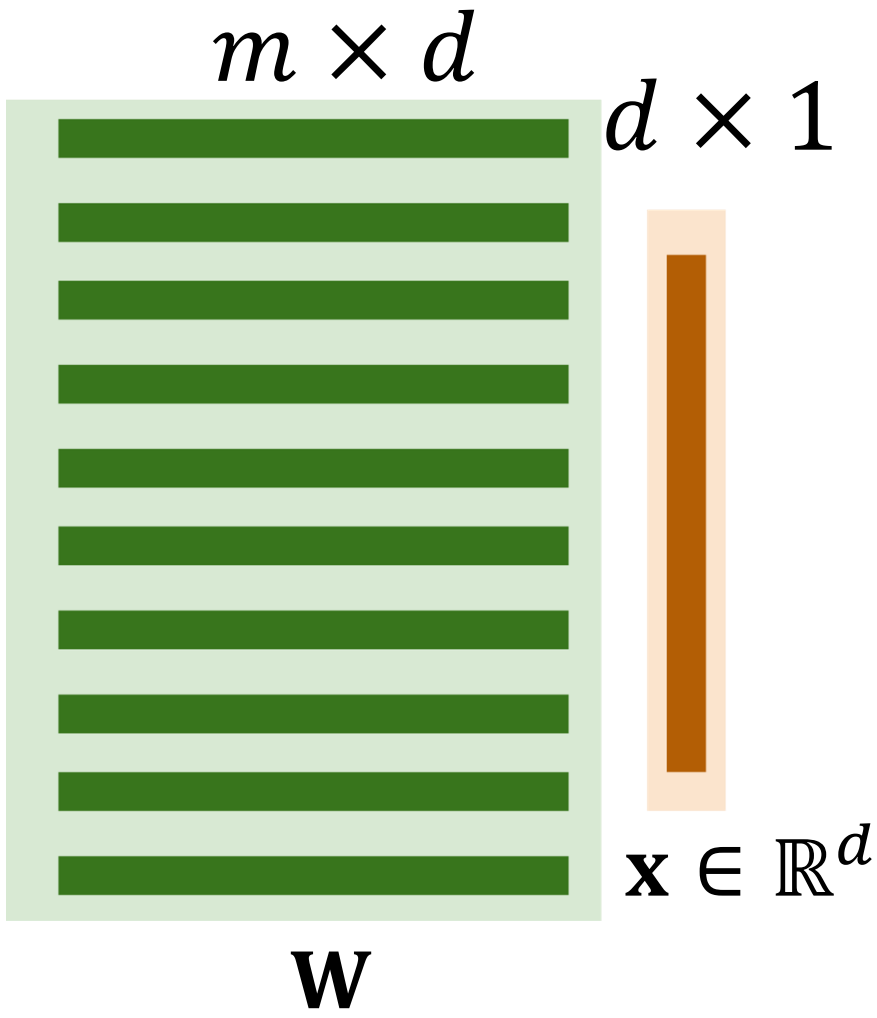
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h} \in \mathbb{R}^m$$

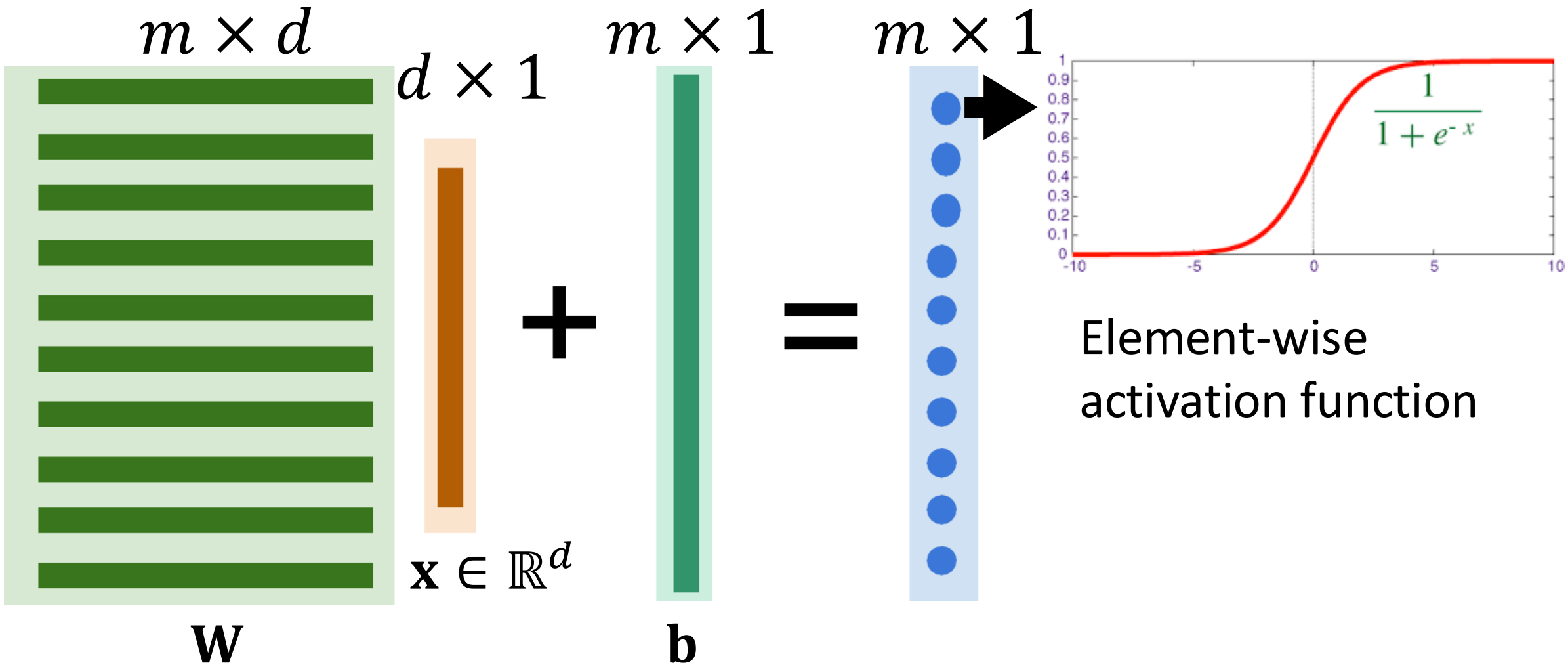


Multi-layer perceptron: **Matrix Notation**



Multi-layer perceptron: **Matrix Notation**

Key elements: linear operations + Nonlinear activations



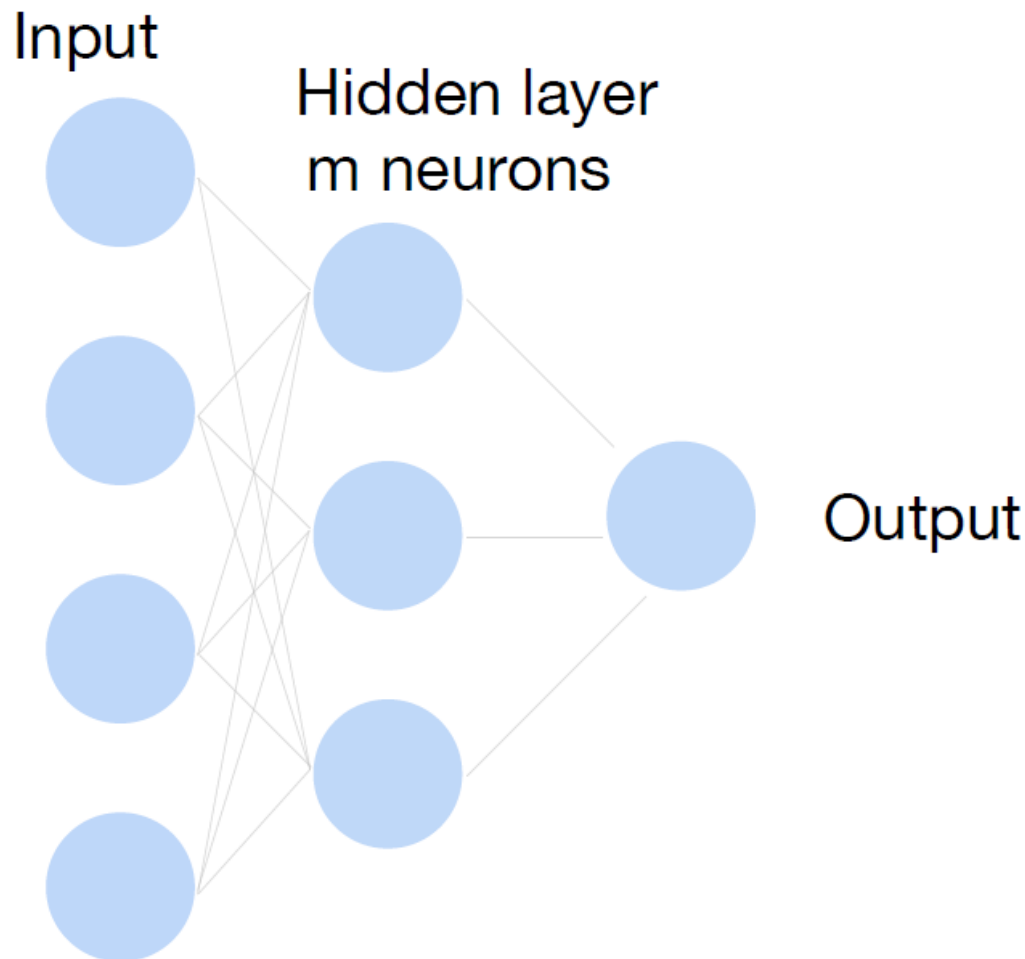
Element-wise activation function

Multi-layer perceptron: Matrix Notation

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

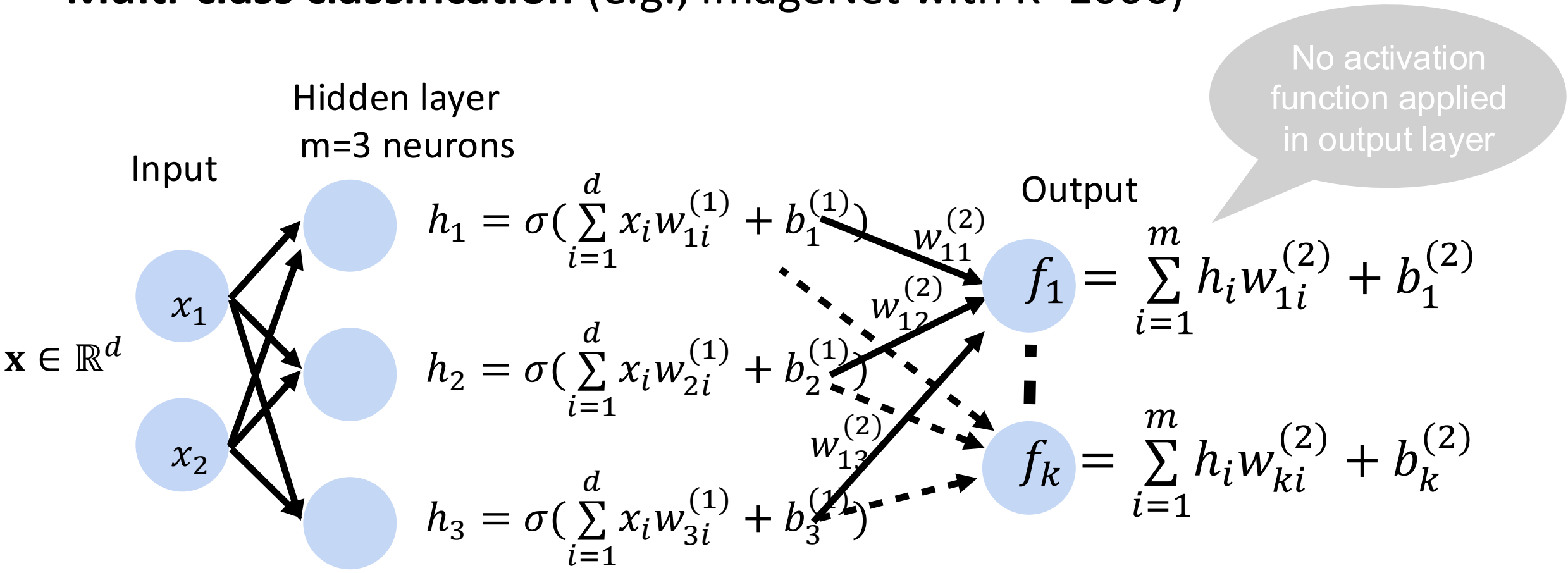
$$\hat{y} = \sigma(\mathbf{w}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$$



Neural network for k-way classification

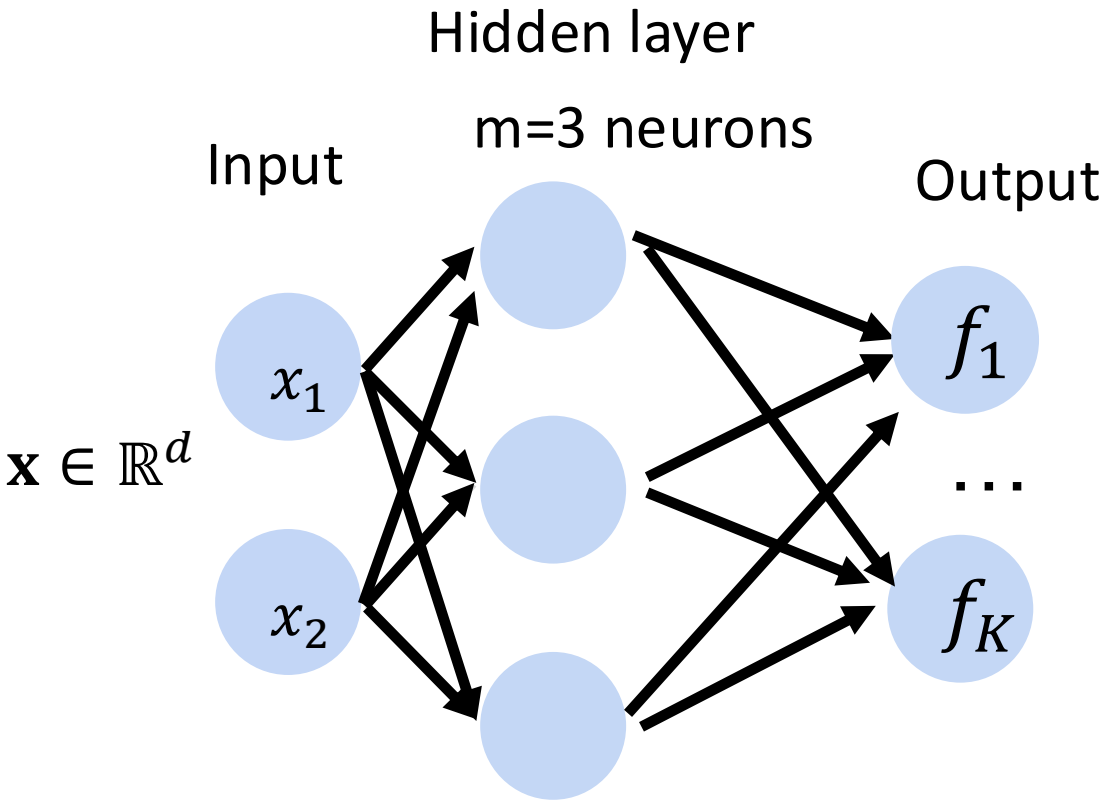
- K outputs in the final layer

Multi-class classification (e.g., ImageNet with K=1000)



Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)

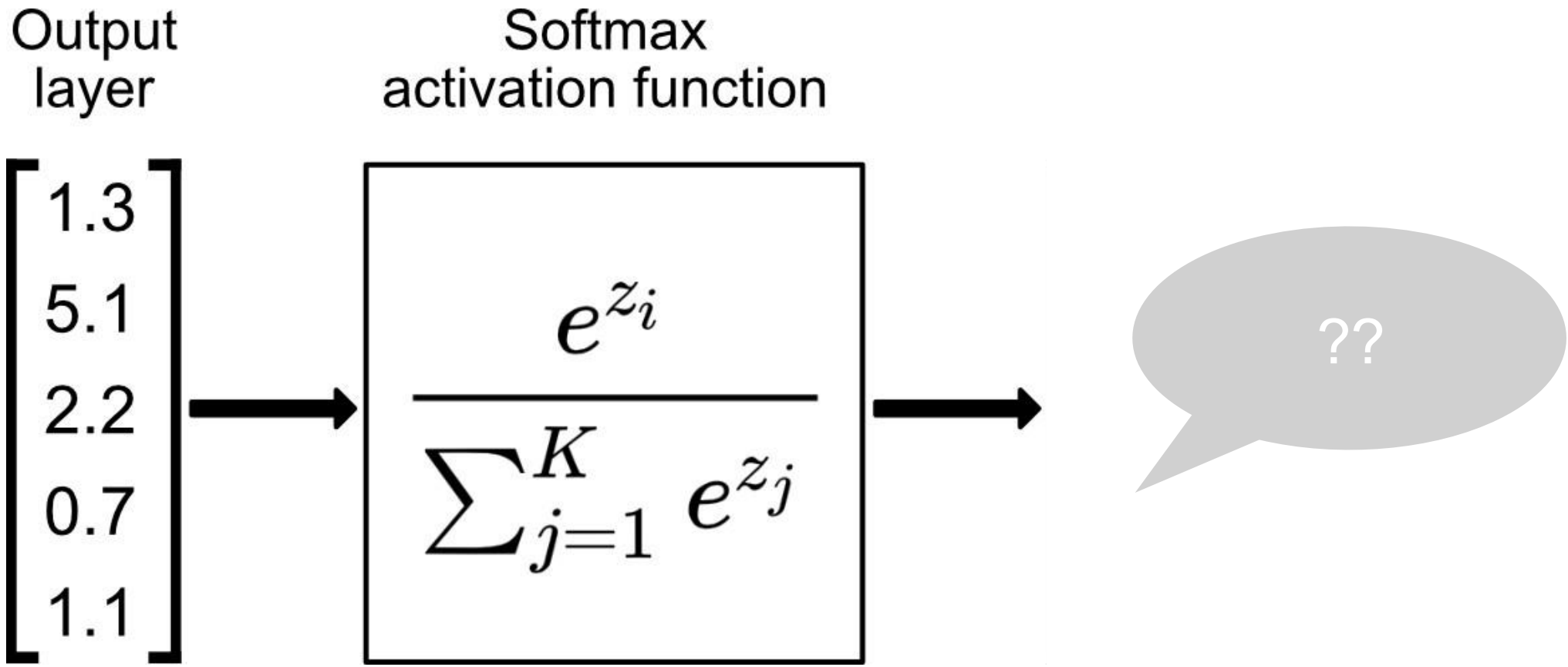


$$p(y|\mathbf{x}) = softmax(f)$$

$$= \frac{\exp(f_y(x))}{\sum_{k=1}^{K} \exp(f_k(x))}$$

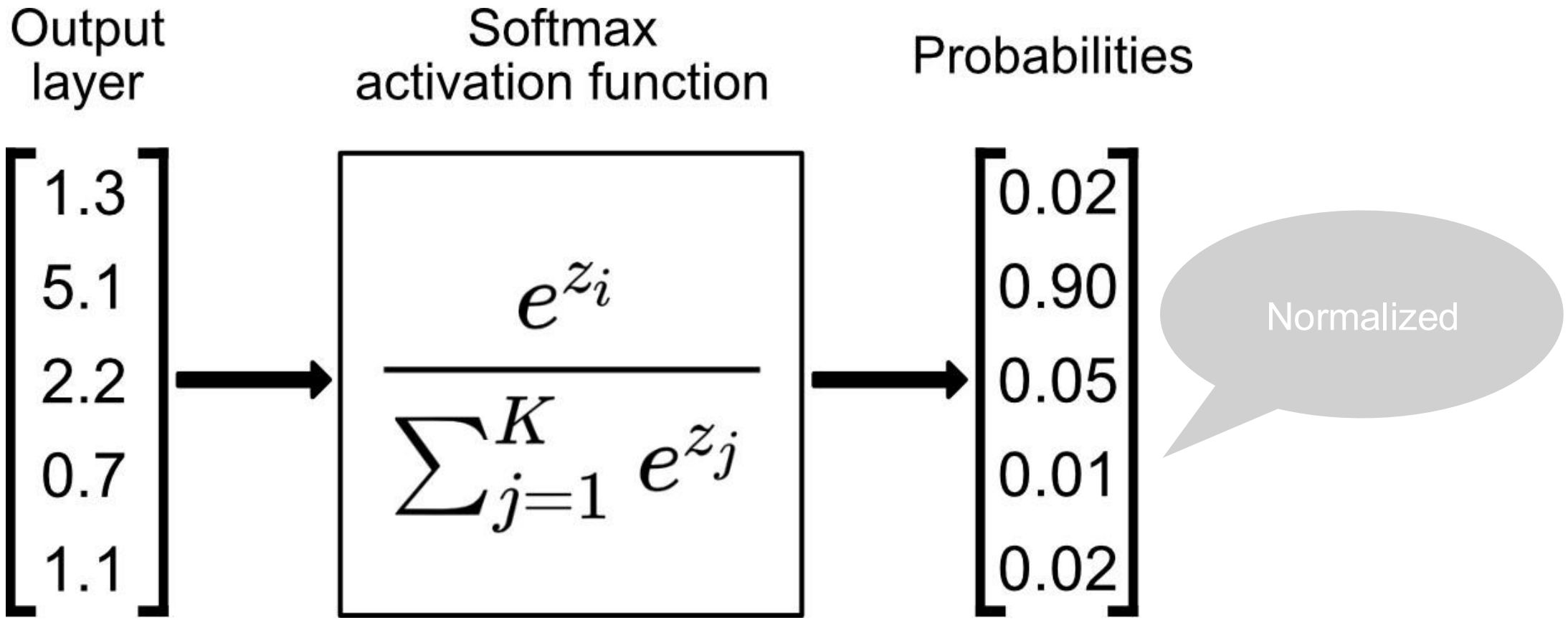
Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)



Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)



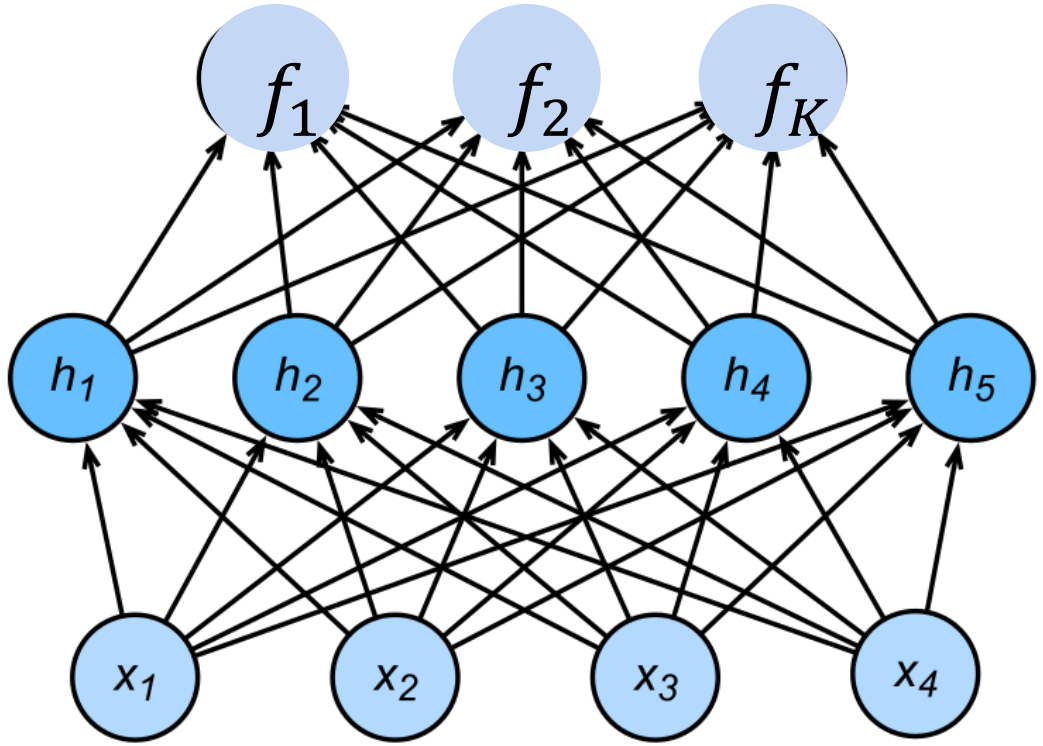
More complicated neural networks

$$p_1, p_2, \ldots, p_K = \text{softmax}(f_1, f_2, \ldots, f_K)$$

Output layer

Hidden layer

Input layer

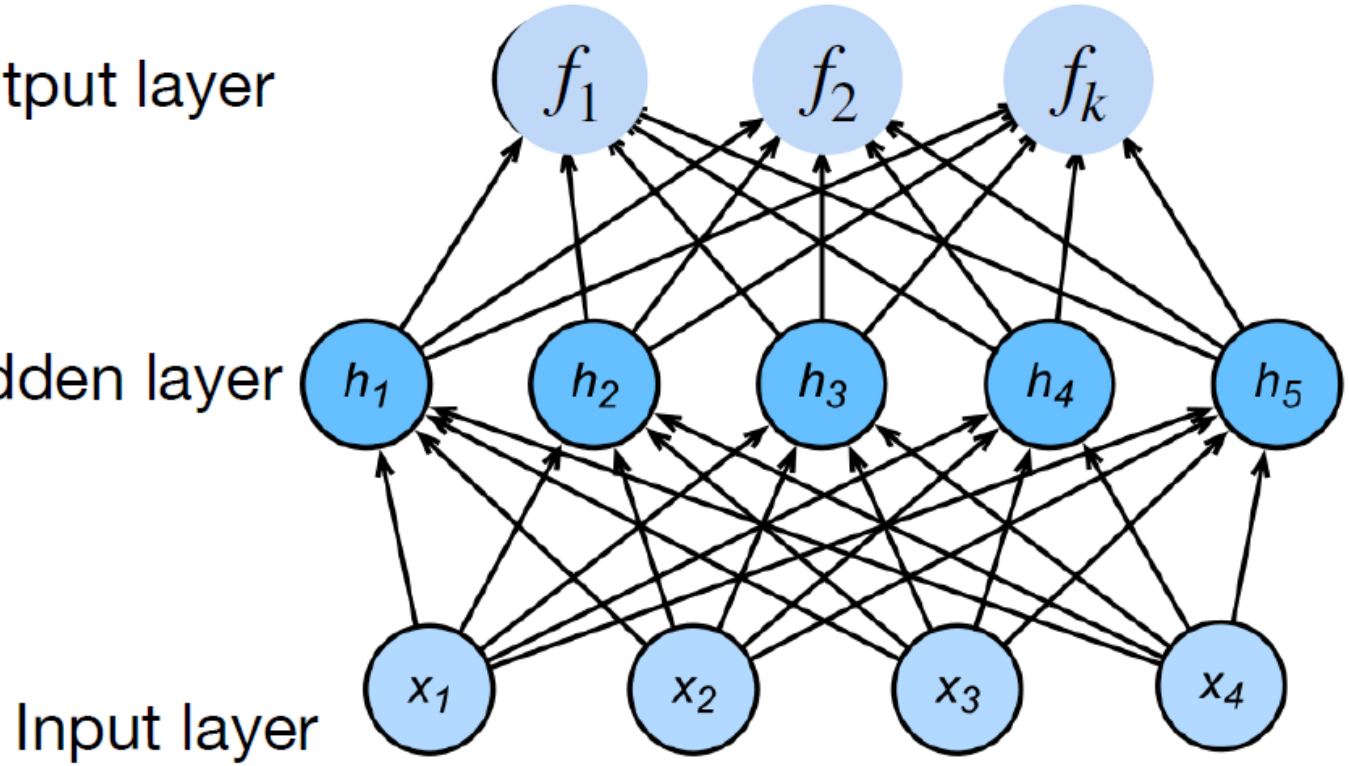


More complicated neural networks

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^m$

$$p_1, p_2, \ldots, p_K = \text{softmax}(f_1, f_2, \ldots, f_K)$$

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{f} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathbf{p} = \text{softmax}(\mathbf{f})$$



More complicated neural networks: multiple hidden layers

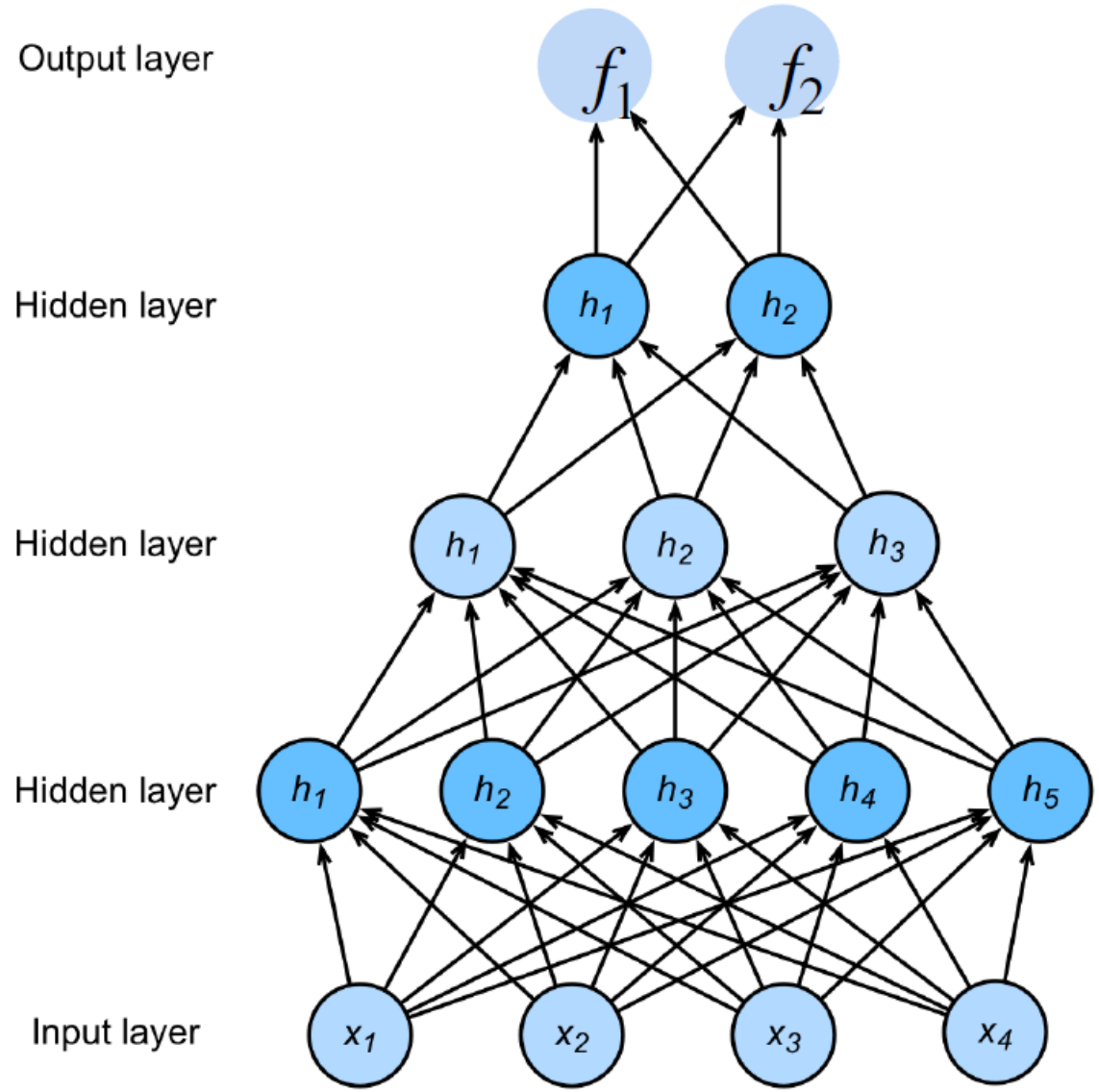
$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

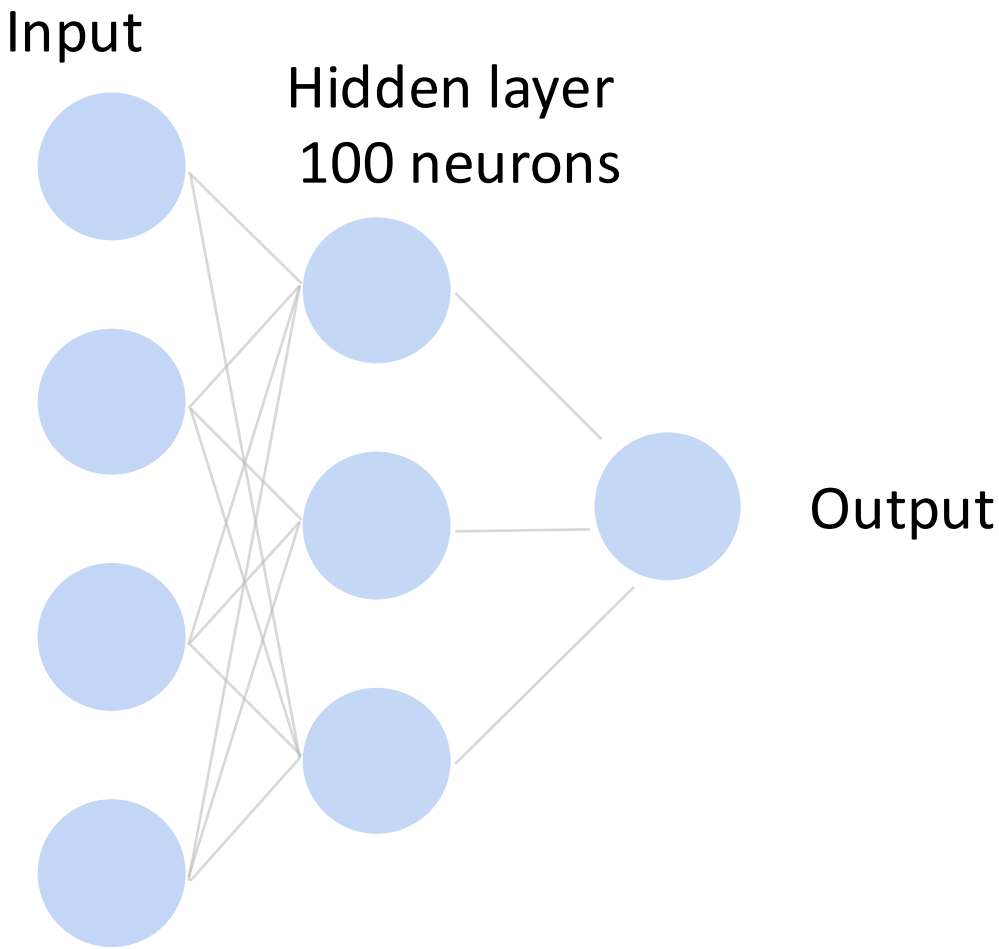
$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

$$\mathbf{p} = \text{softmax}(\mathbf{f})$$



How to train a neural network?

Classify cats vs. dogs



How to train a neural network? Binary classification

$\mathbf{x} \in \mathbb{R}^d$ One training data point in the training set D

$\hat{y} \in [0,1]$ Model output for example **x**

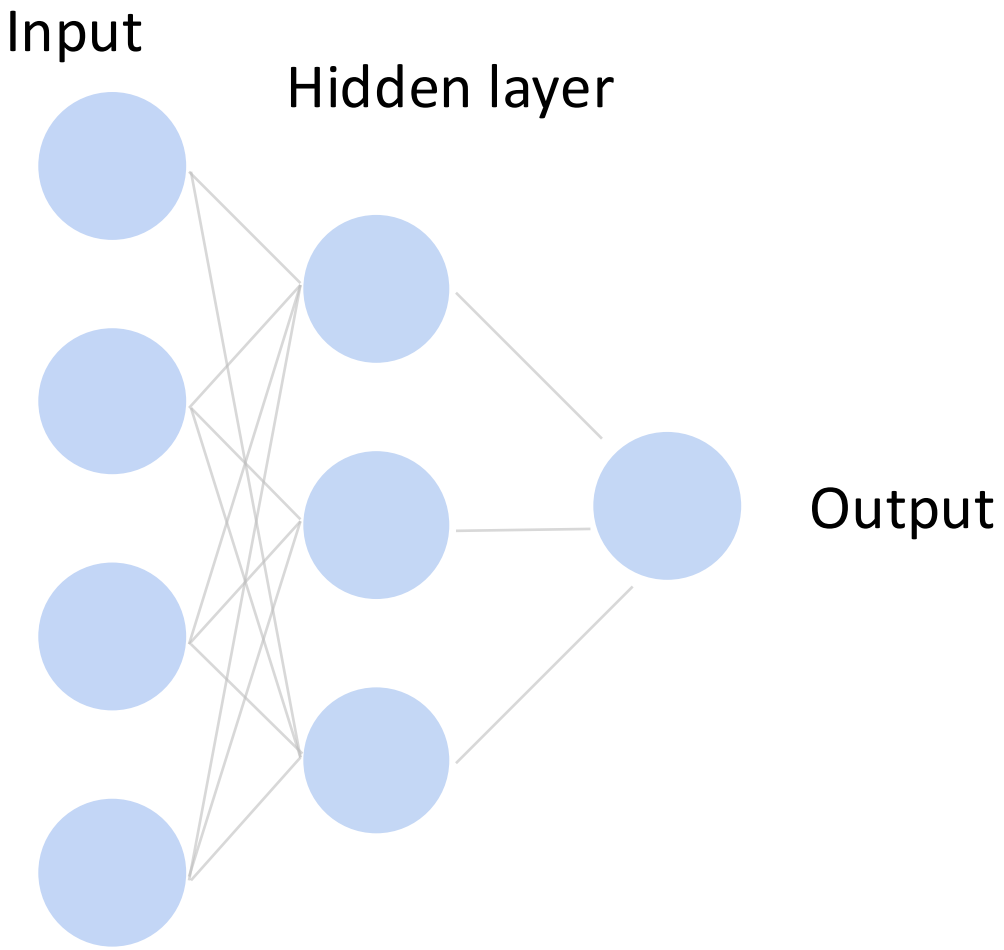
(This is a function of all weights W: $\hat{y} = g(W)$)

y Ground truth label for example **x**

Learning by matching the output to the label

We want $\hat{y} \to 1$ **when** $y = 1$,

and $\hat{y} \to 0$ **when** $y = 0$

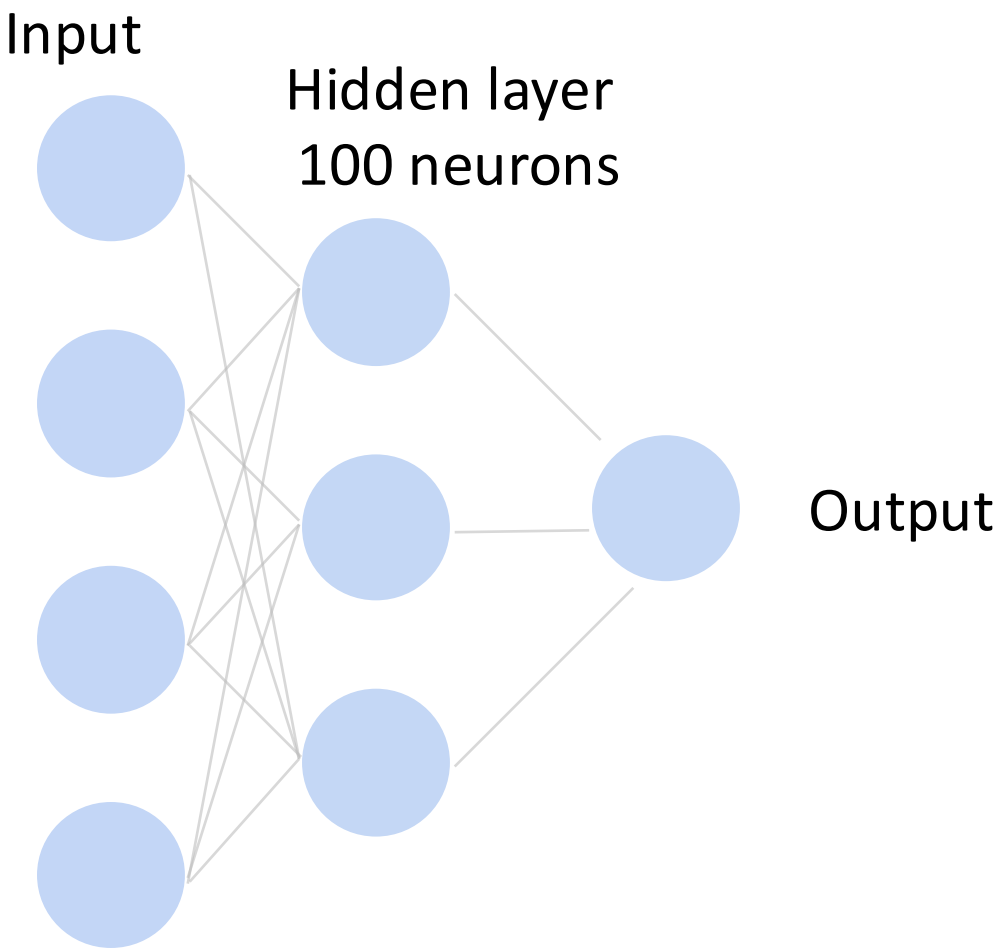


How to train a neural network? Binary classification

Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \ell(\mathbf{x}, y)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$



Negative log likelihood

Also known as binary cross-entropy loss

- **We'll use all the time for LLMs**

How to train a neural network? Multiclass

Loss function: $$\frac{1}{|D|}\sum_{(\mathbf{x},y)\in D}\ell(\mathbf{x},y)$$

Per-sample loss:

$$\ell(\mathbf{x},y)=\sum_{k=1}^{K}-Y_k\log p_k=-\log p_y$$

where Y is one-hot encoding of y

Also known as cross-entropy loss or softmax loss