COMS 4771, 23rd October 2017

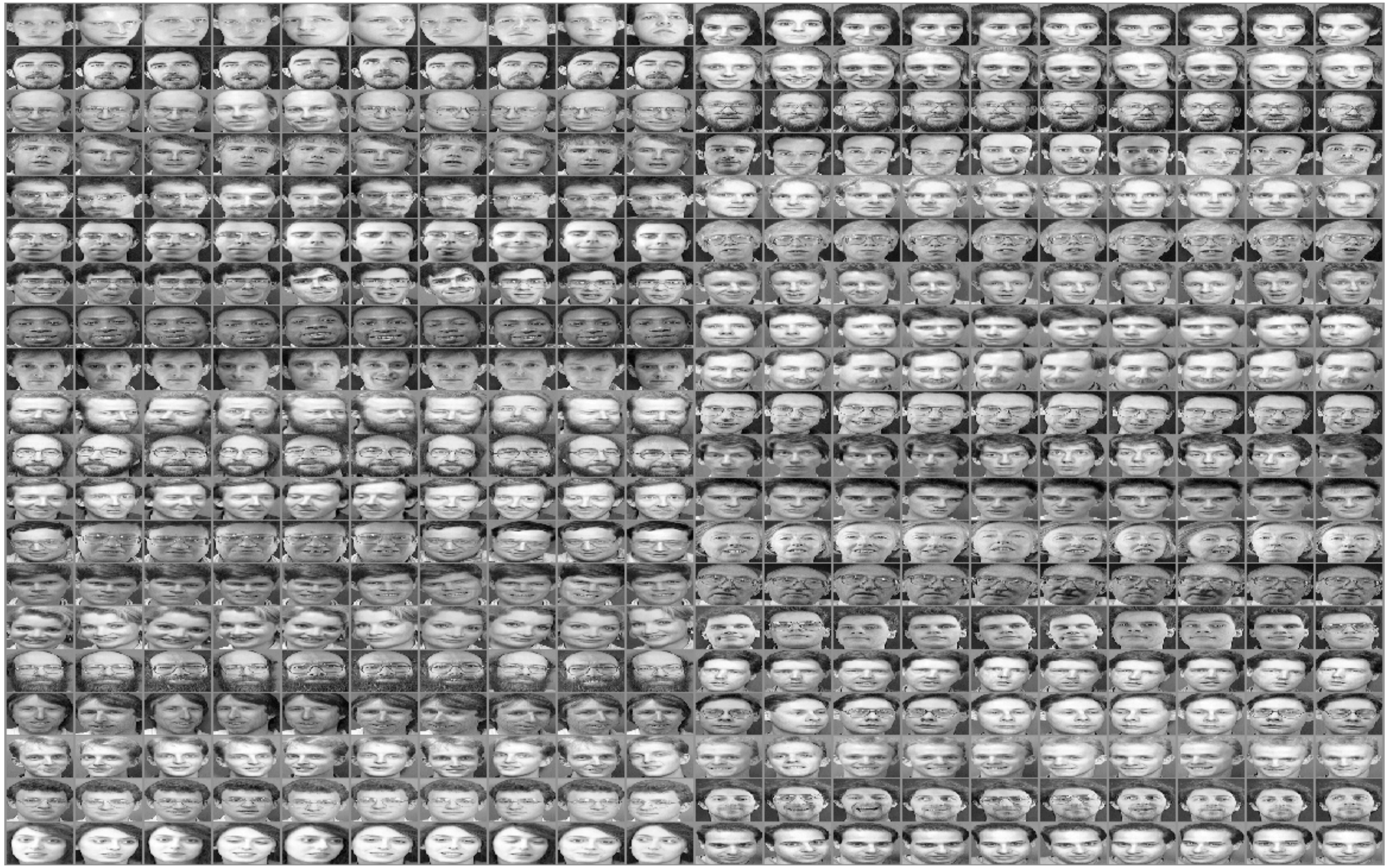
Neural Networks

James McInerney

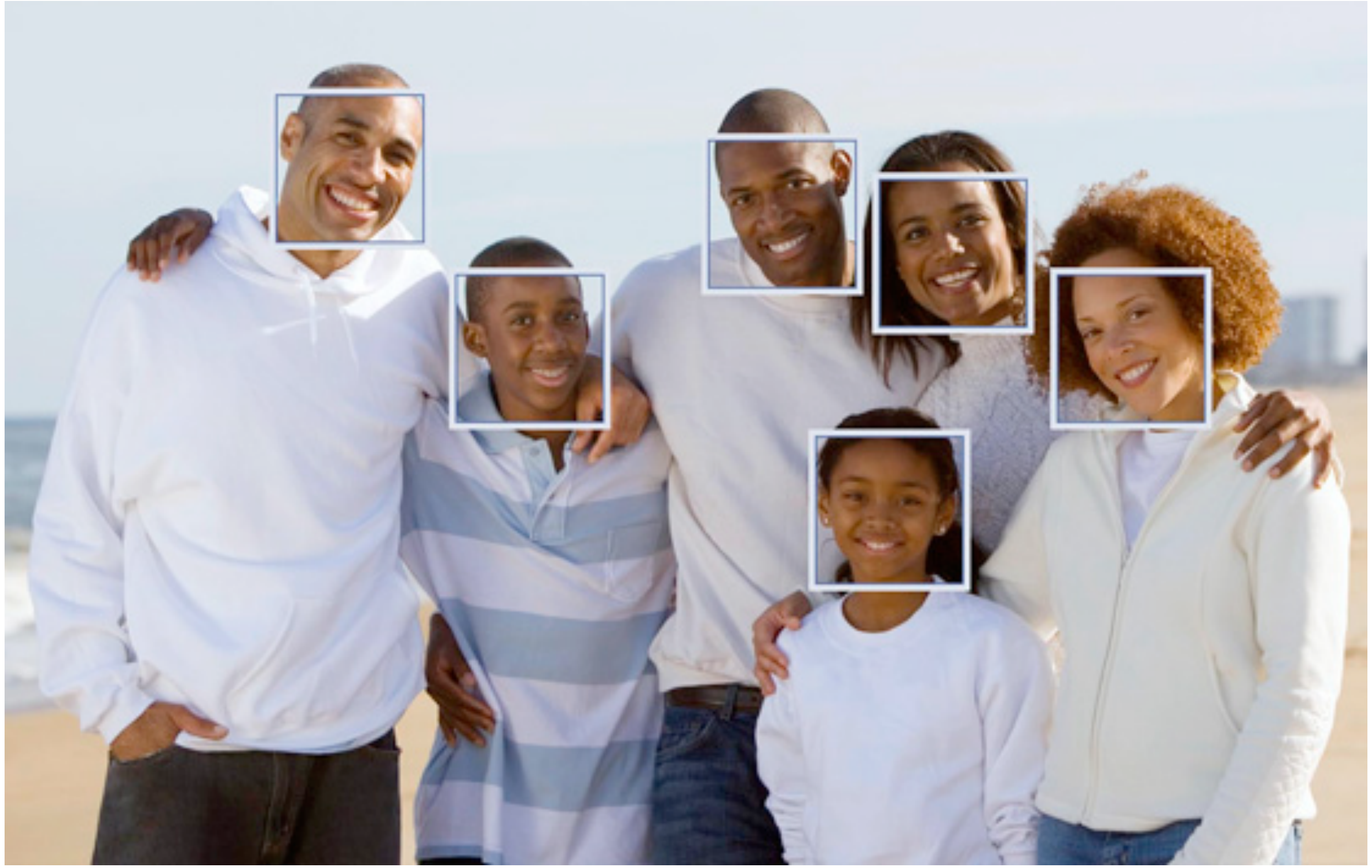
Adapted from slides by Francisco J. R. Ruiz

Columbia University
Data Science Institute

Introduction



Introduction



Introduction

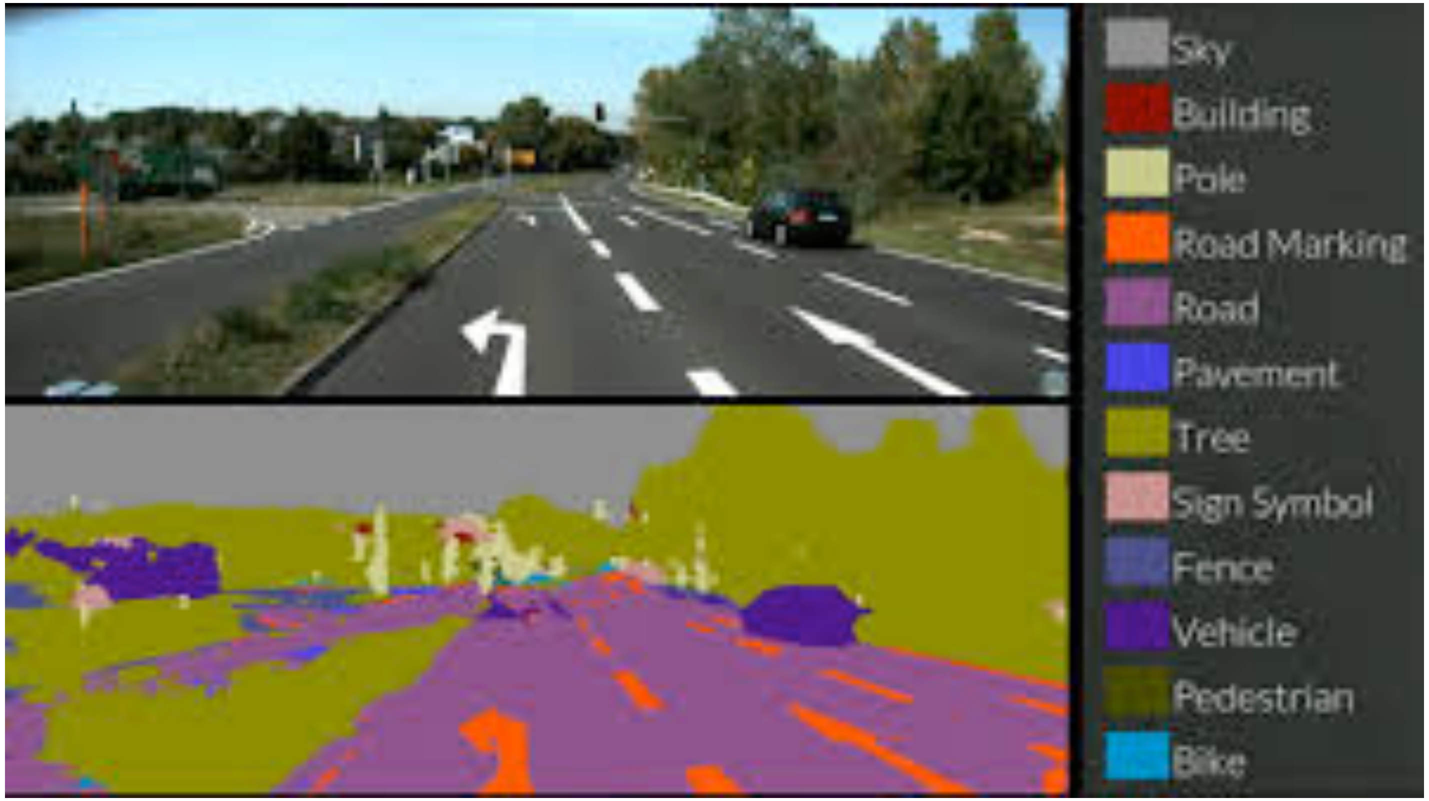
Deep Learning: Applications

Visually similar images

Deep Learning: Applications

Deep Learning: Applications

Deep Learning: Applications



Deep Learning: Applications

DeepDrumpf
@DeepDrumpf

DeepDrumpf @DeepDrumpf · 8 nov. 2016
[I told Ohio] my promise to the American voter: If I am elected President, I will grow your money. $500 billion a year to be a Republican.

DeepDrumpf
@DeepDrumpf
Seguir

[Math is a] common democrat lie. It can't make the budget great. I'll have the best economy. #debatenight

Deep Learning: Brief History

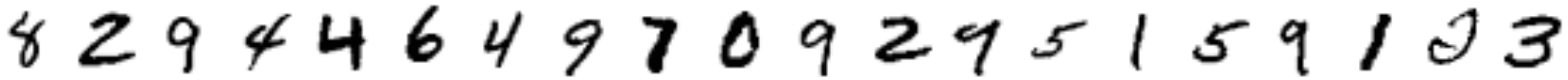
- Deep learning has become a hot topic recently…

  … but it dates back to the 70s

- It has been "re-discovered" recently due to
  - Massive amounts of data
  - Computational capabilities

Neural Networks

- Running example: Classification

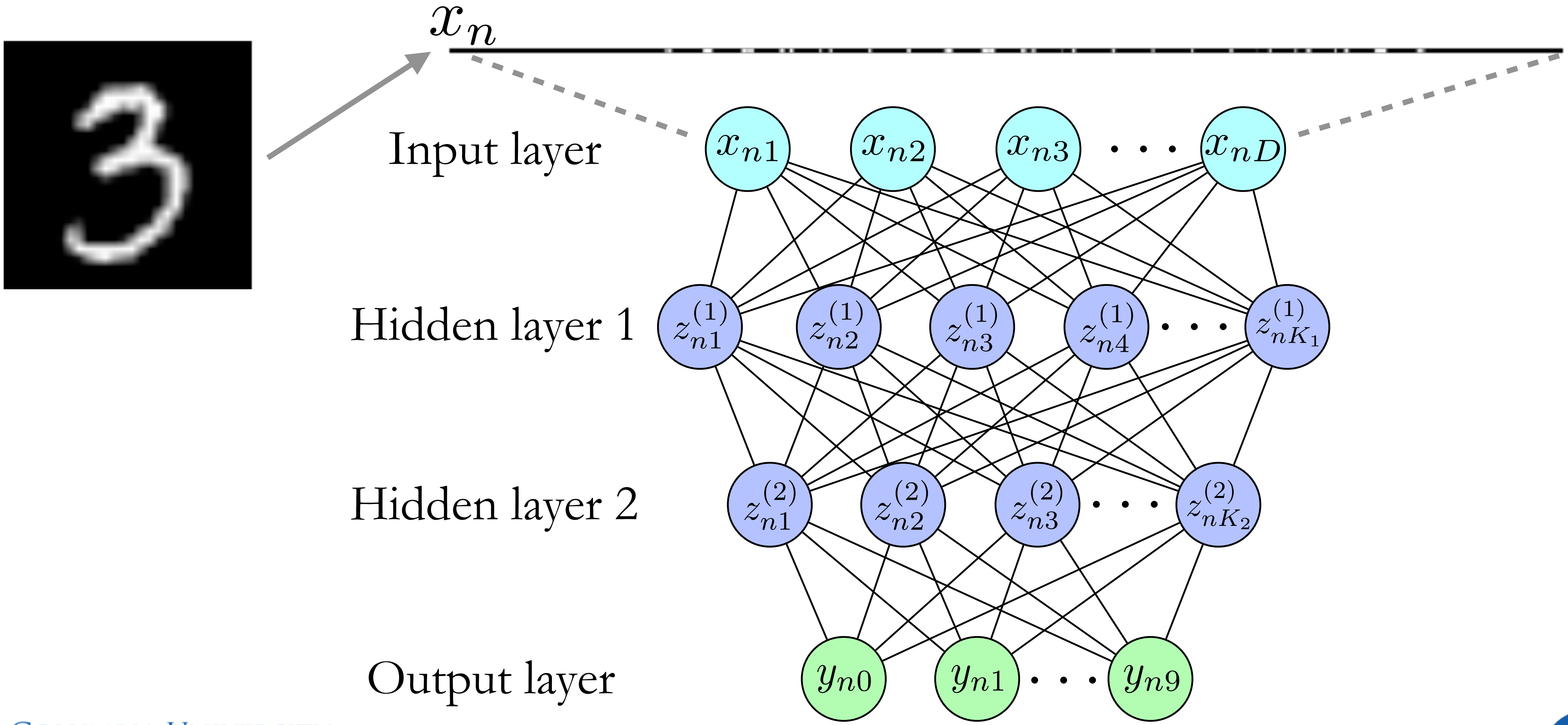


- NNs can also be applied for regression (and even for unsupervised learning)

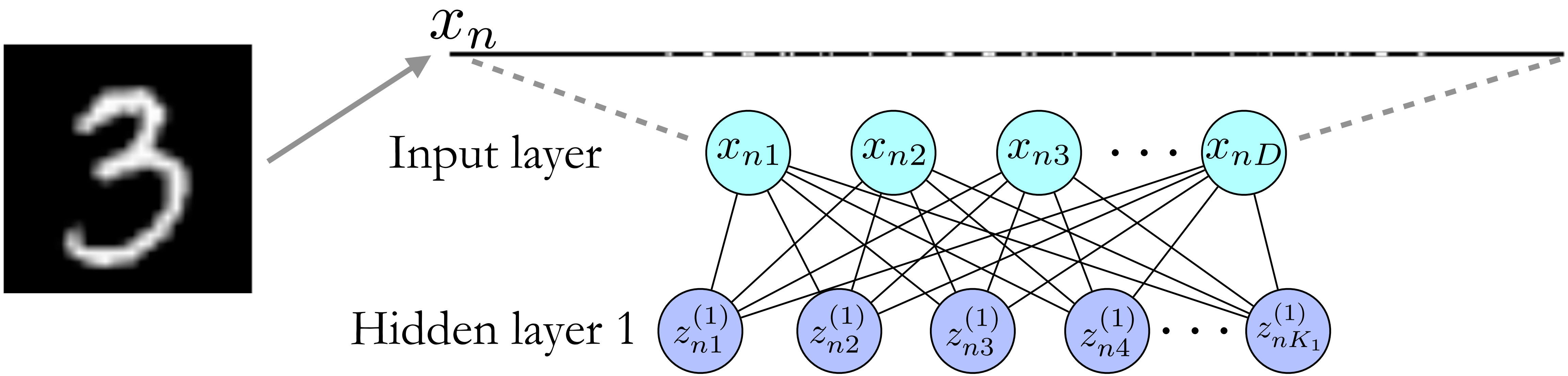
Neural Networks

- In neural networks, we have a cascade of layers from the input to the output
  - Input layer
  - Hidden layers (any number)
  - Output layer

Neural Networks

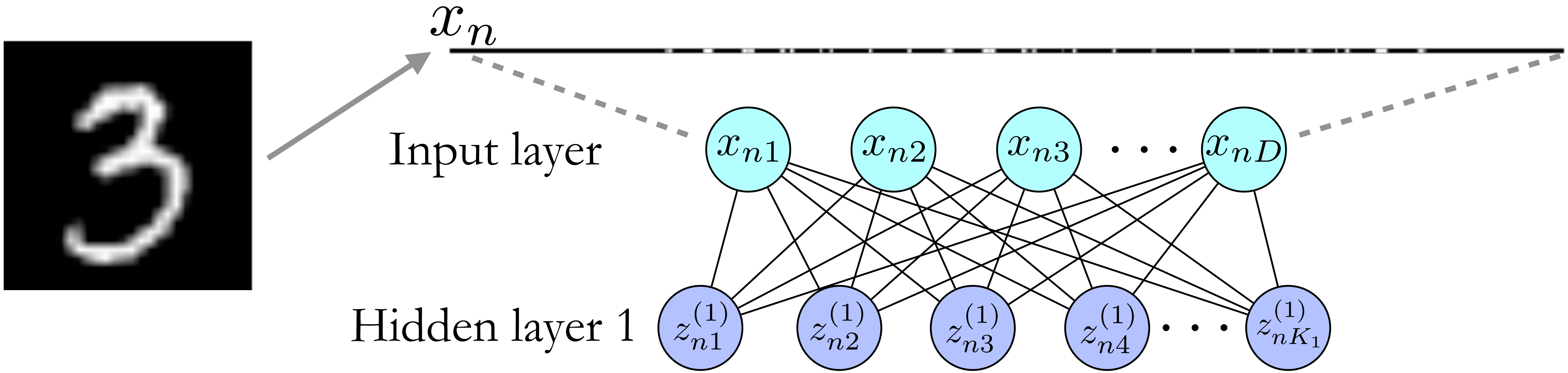


Neural Networks



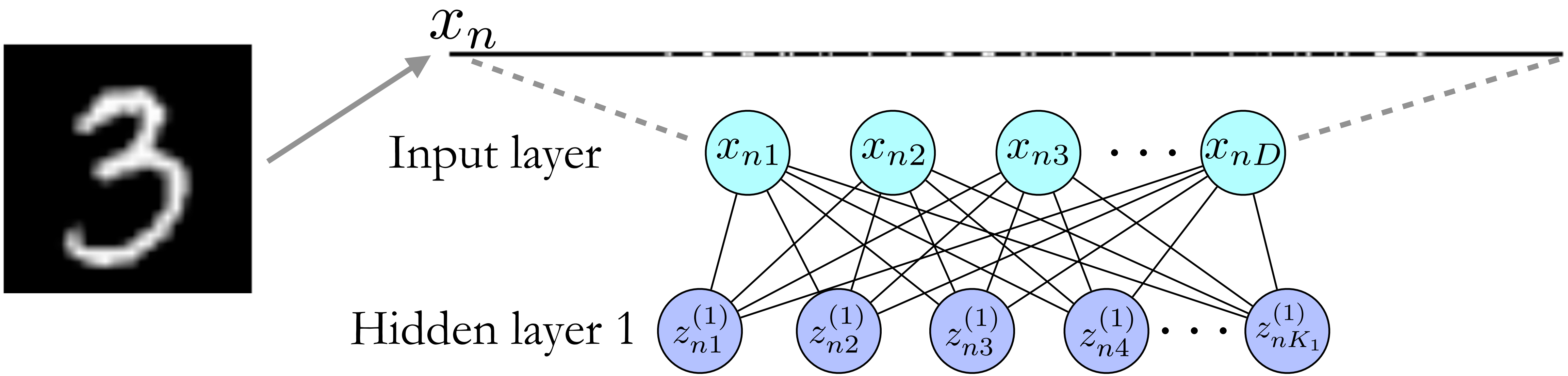
- Each layer:
  - Dot product + Non-linear function:

Neural Networks



- Each layer:
  - Dot product + Non-linear function: $\mathbf{z}_n^{(1)} = f_1(\mathbf{x}_n \mathbf{W}^{(1)})$
  - Number of weights? $D \times K_1$

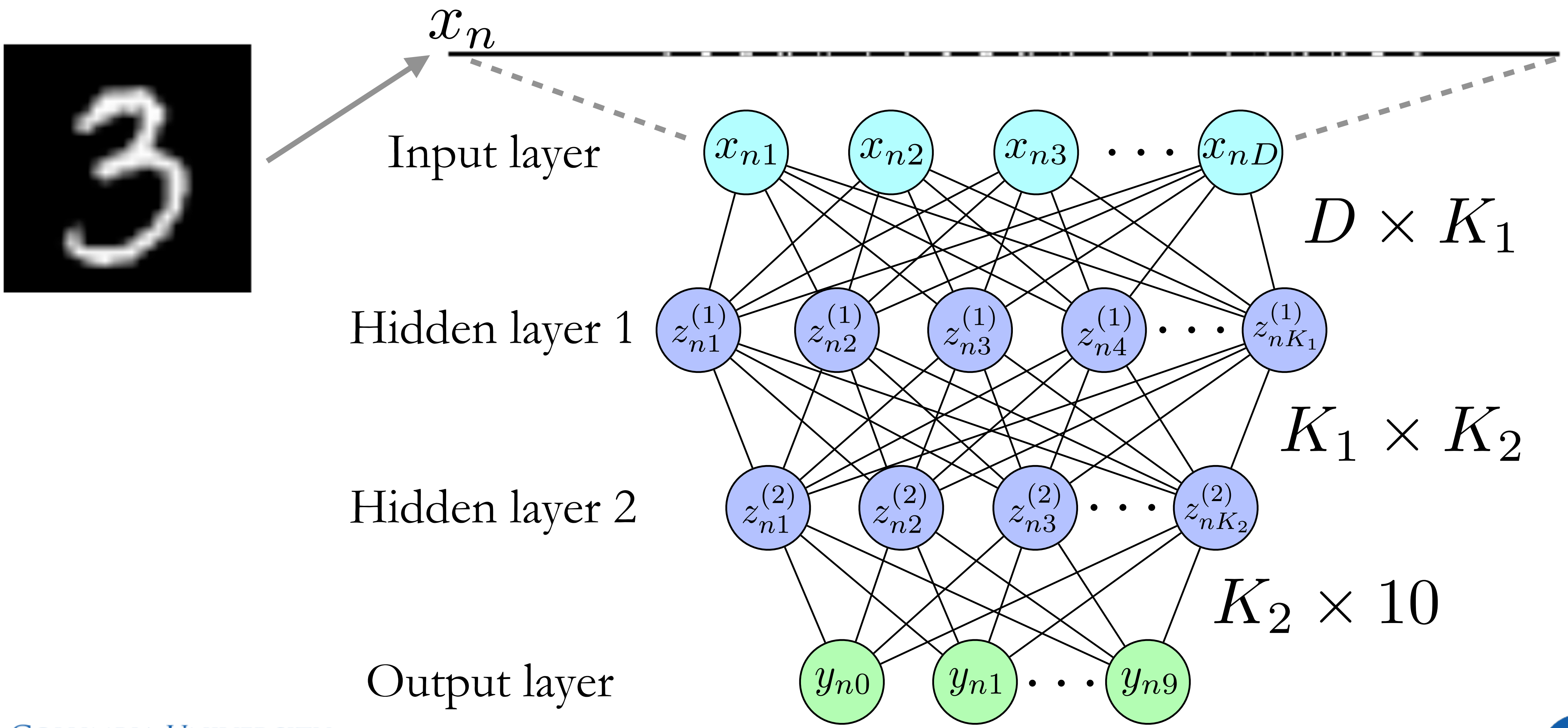
Neural Networks



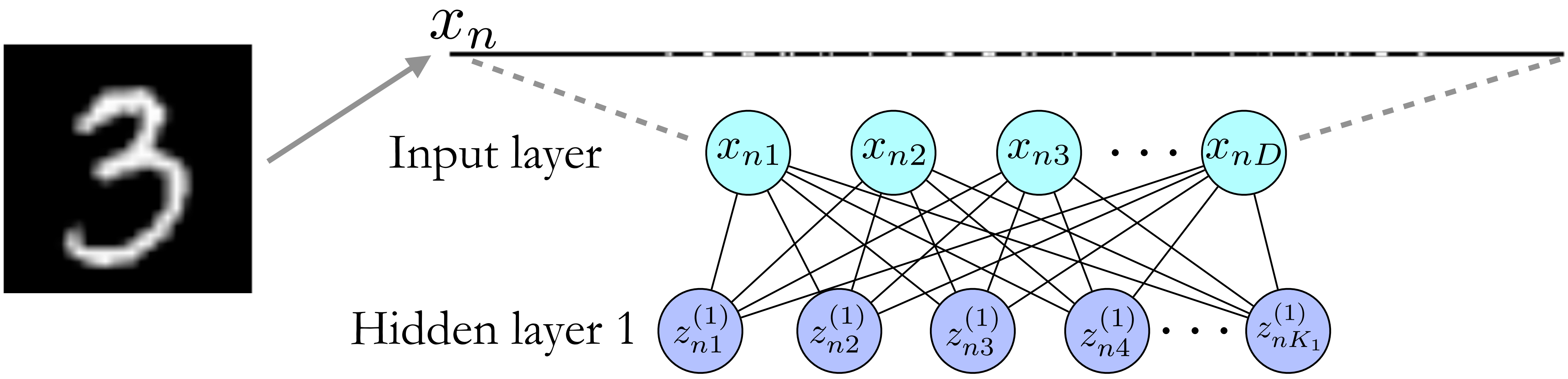
- Number of weights for each layer:

$$\text{Input dimension} \times \text{Output dimension}$$

Neural Networks



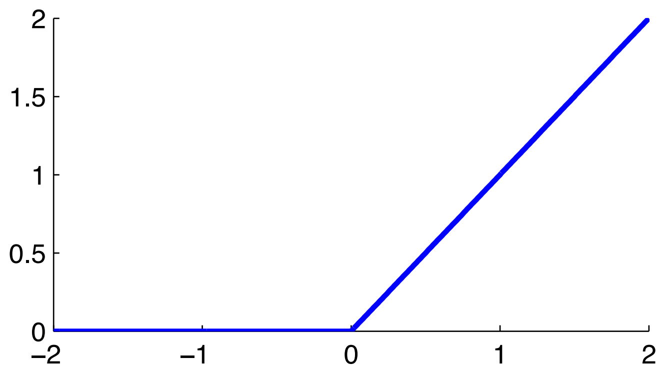
Neural Networks



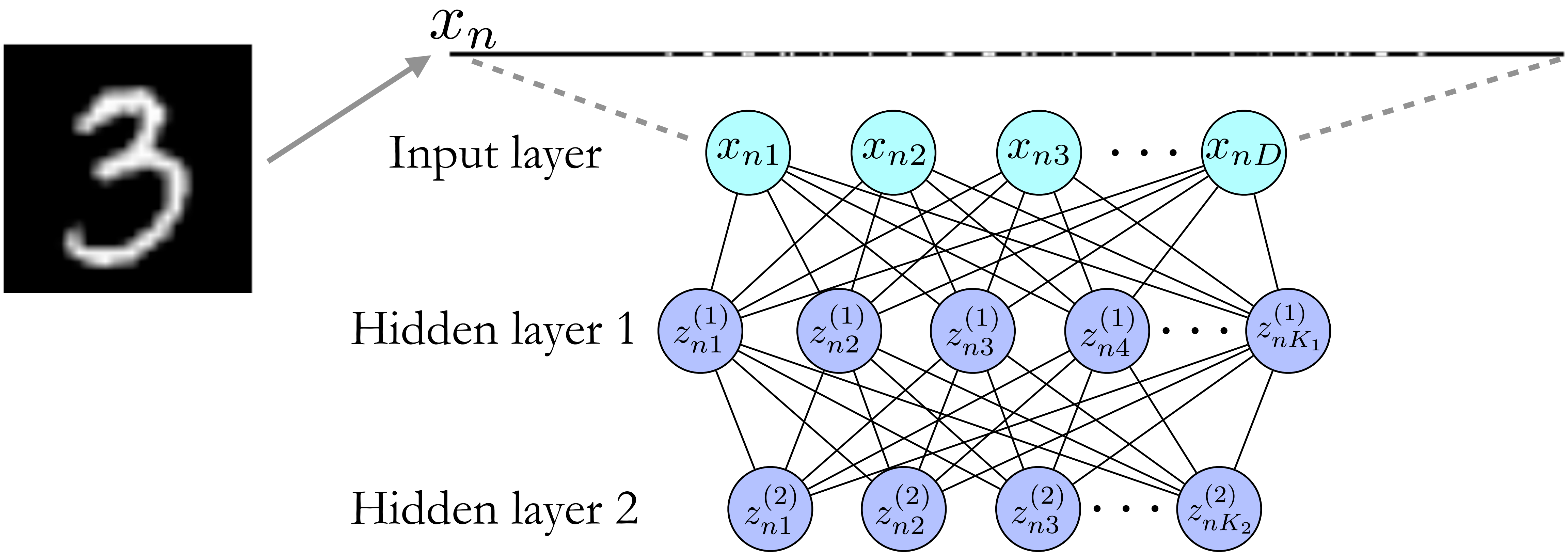
$$\mathbf{z}^{(1)}_n = f_1(\mathbf{x}_n \mathbf{W}^{(1)})$$

- Non-linear function: ReLU

$$f_1(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

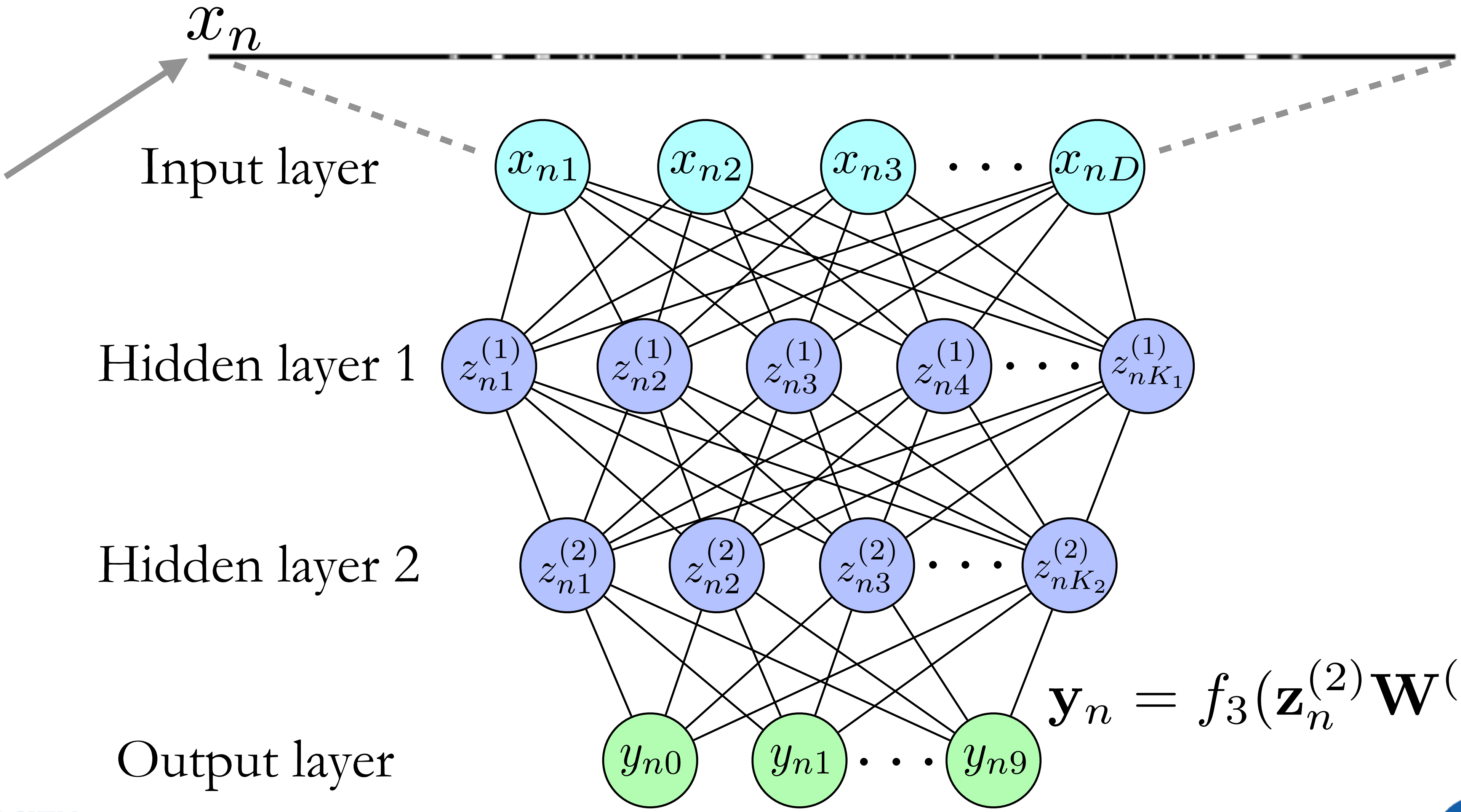


Neural Networks



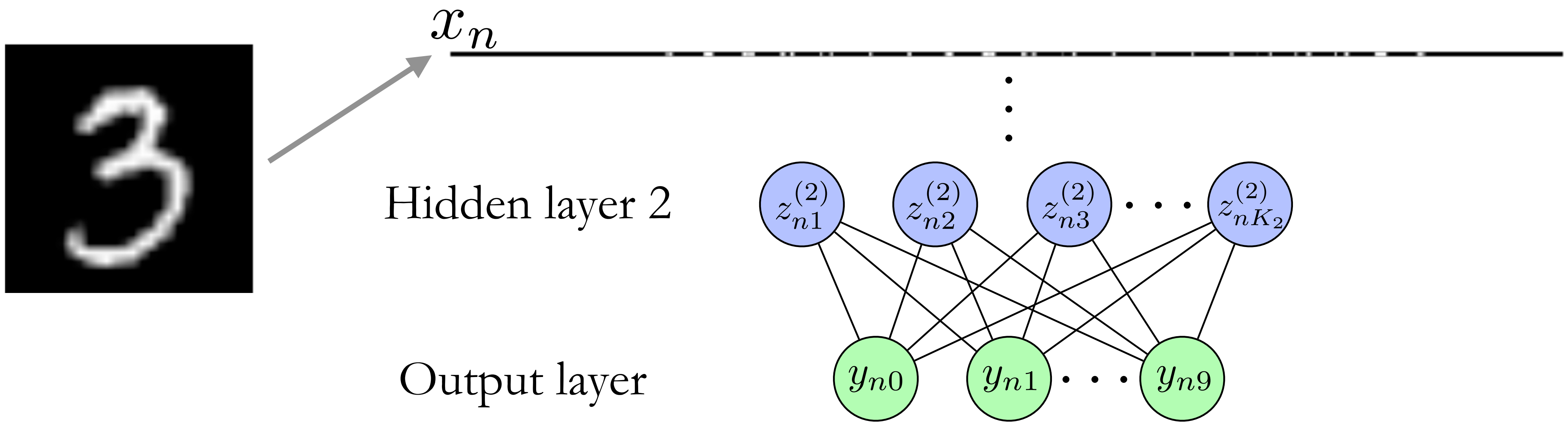
$$\mathbf{z}_n^{(2)} = f_2(\mathbf{z}_n^{(1)}\mathbf{W}^{(2)})$$

Neural Networks



$$\mathbf{y}_n = f_3(\mathbf{z}_n^{(2)}\mathbf{W}^{(3)})$$

Neural Networks



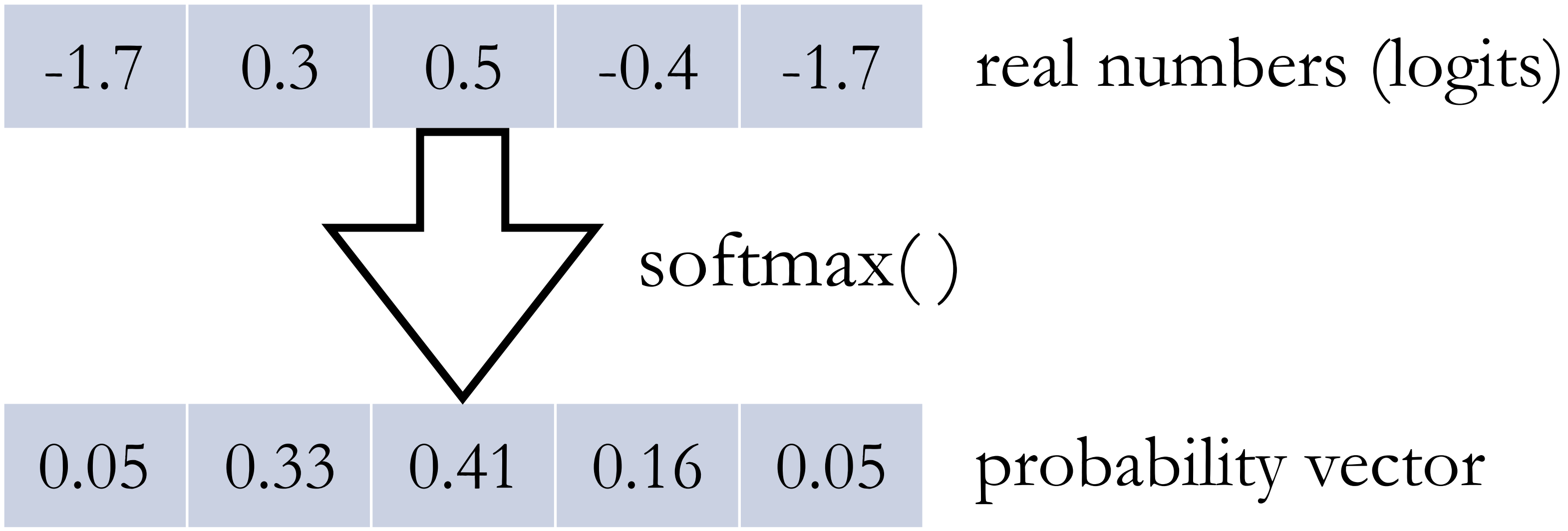
$$\mathbf{y}_n = f_3(\mathbf{z}_n^{(2)}\mathbf{W}^{(3)})$$

The output should be the probability for each class

- The non-linear function is a softmax

An Aside: Softmax

- The softmax is a function that inputs a vector of *reals* and outputs a *probability* vector



An Aside: Softmax

- The softmax is a function that inputs a vector of *reals* and outputs a *probability* vector

-1.7	0.3	0.5	-0.4	-1.7	real numbers (logits)

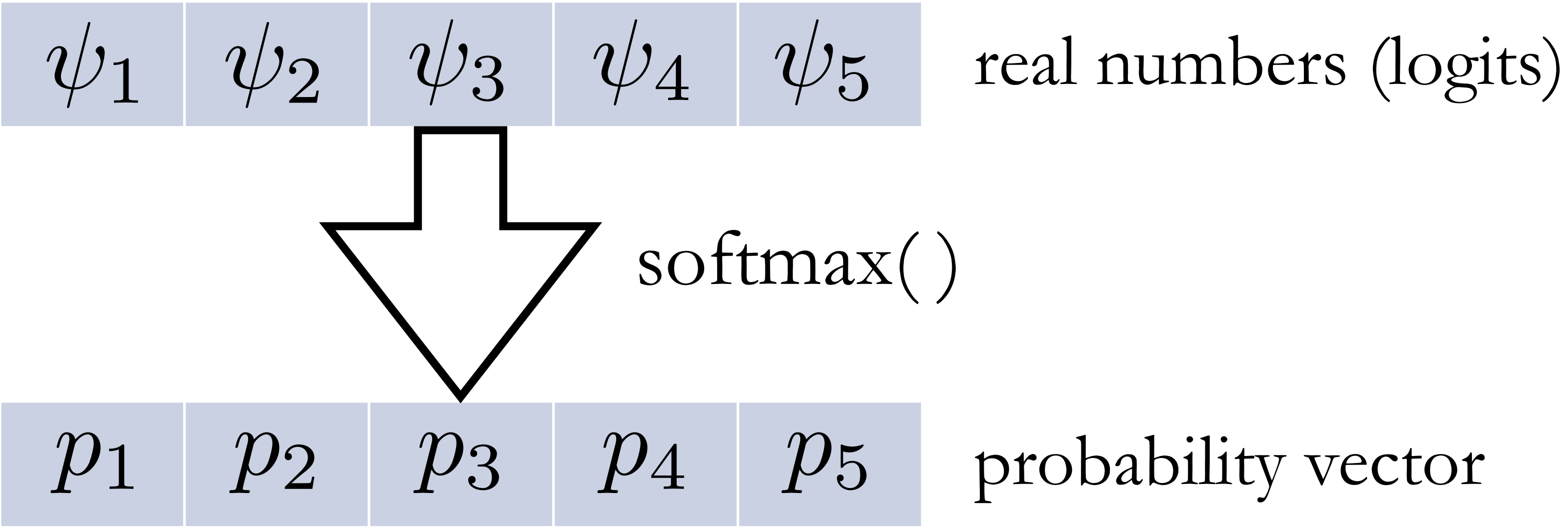
softmax( )

0.05	0.33	0.41	0.16	0.05	probability vector

$$0.41 = \frac{e^{0.5}}{e^{-1.7} + e^{0.3} + e^{0.5} + e^{-0.4} + e^{-1.7}}$$

An Aside: Softmax

- The softmax is a function that inputs a vector of *reals* and outputs a *probability* vector



$$p_i = \frac{e^{\psi_i}}{\sum_j e^{\psi_j}}$$

Inference

- To fit the model, we need to learn
  - The weights of all layers
  - The biases (intercepts) of all layers

Inference

- To fit the model, we need to learn
  - The weights of all layers
  - The biases (intercepts) of all layers
- Define your error (loss) function on the training data

$$\mathcal{L} = \sum_{n=1}^{N} \log(\hat{y}_n)$$

$\downarrow$ predicted probability for the observed class

Inference

- We maximize the loss with respect to weights and biases
- Gradient ascent

$$\nabla \mathcal{L} = \sum_{n=1}^{N} \nabla \log(\hat{y}_n)$$

  - Backpropagation
  - Too expensive: N can be large

Other Types of Neural Networks

- Feed-forward neural networks
  - Fully-connected networks
  - Convolutional networks
- Recurrent neural networks