Data Mining & Machine Learning

CS57300
Purdue University

February 22, 2017

Introducing More ML/DM Tasks

Supervised Learning

- Training data

$$\{x_i, y_i\}_i$$

- Trying to learn

$$p(y|x)$$

- Assumption

$$x_i, y_i \sim p(y|x)p(x)$$

- Test data

$$\{x_i, y_i\}_i$$

- Assumption

$$x_i, y_i \sim p(x, y)$$

Unsupervised Learning

- Training data

  $\{x_i\}_i$

- Trying to learn

  $x_i \sim p(x)$

- Assumption

  $x_i \sim p(x)$

- Test data

  $\{x_i\}_i$

- Assumption

  $x_i \sim p(x)$

Semi-supervised Learning

- Training data
  $\{x_i, y_i\}_i$
  $\{x_j\}_j$
- Trying to learn
  $x_i, y_i \sim p(y|x)p(x)$

- Assumption
  $x_i, y_i \sim p(y|x)p(x)$
  $x_j \sim p(x)$

- Test data
  $\{x_i, y_i\}_i$

- Assumption
  $x_i, y_i \sim p(y|x)p(x)$

Multi-task Learning

- Training data

  $\{x_i, y_{i,1}, \ldots, y_{i,M}\}_i$

- Trying to learn

  $p(y_1, \ldots, y_M|x)$

- Assumption

  $x_i, y_{i,1}, \ldots, y_{i,M} \sim p(y_1, \ldots, y_M|x)p(x)$

- Test data

  $\{x_i, y_{i,1}, \ldots, y_{i,M}\}_i$

- Assumption

  $x_i, y_{i,1}, \ldots, y_{i,M} \sim p(y_1, \ldots, y_M|x)p(x)$

Transfer Learning

- Training data
  $\{x_i, y_{i,1}, \ldots, y_{i,M}\}_i$
- Trying to learn
  $p(y_1|x)$
- Assumption
  $x_i, y_{i,1}, \ldots, y_{i,M} \sim p(y_1, \ldots, y_M|x)p(x)$

- Test data
  $\{x_i, y_{i,1}\}_i$
- Assumption
  $x, y_1 \sim p(y_1|x)p(x)$

Domain Adaptation

- Training data

  $\{x_i, y_i\}_i$

  $\{z_j\}_j$

- Trying to learn

  $p(y_i|x_i)$

  ↖ in a way that is robust to p(x)

- Assumption

  $x_i \sim p(x)$

  $y_i \sim p(y|x)$

  $z_j \sim q(z) \approx p(z)$

- Test data

  $\{z_i, y_i\}_i$

- Assumption

  $z_i \sim q(z)$

  $y_i \sim p(y|z)$

  Related to covariate shift

One-Shot Learning

- Training data

  $\{x_i, y_i\}_i$

  $\text{where } y_i \in \{1, \ldots, C\}$

  $\{(z_1, C+1), \ldots, (z_M, C+M)\}$

- Trying to learn

  $p(y|x)$

- Assumption

  $x_i, y_i \sim \mathbf{1}_{\{y \in \{1,\ldots,C\}\}} p(y|x)p(x)$

- Test data

  $\{x_i, y_i\}_i$

  $\text{where } y_i \in \{C+1, \ldots, C+M\}$

- Assumption

  $x_i, y_i \sim \mathbf{1}_{\{y \in \{C+1,\ldots,C+M\}\}} p(y|x)p(x)$

Zero Shot Learning

- Training data

  $\{x_i, y_i\}_i$

  where $y_i \in \{1, \ldots, C\}$

  $\{z_j\}_j$ E.g., words

- Trying to learn

  $p(y|x)$

- Assumption

  $x_i, y_i \sim \mathbf{1}_{\{y \in \{1,\ldots,C\}\}} p(y|x)p(x)$

  $z \sim p(z|c), \quad c = 1, \ldots, C + M$

  Words describing classes

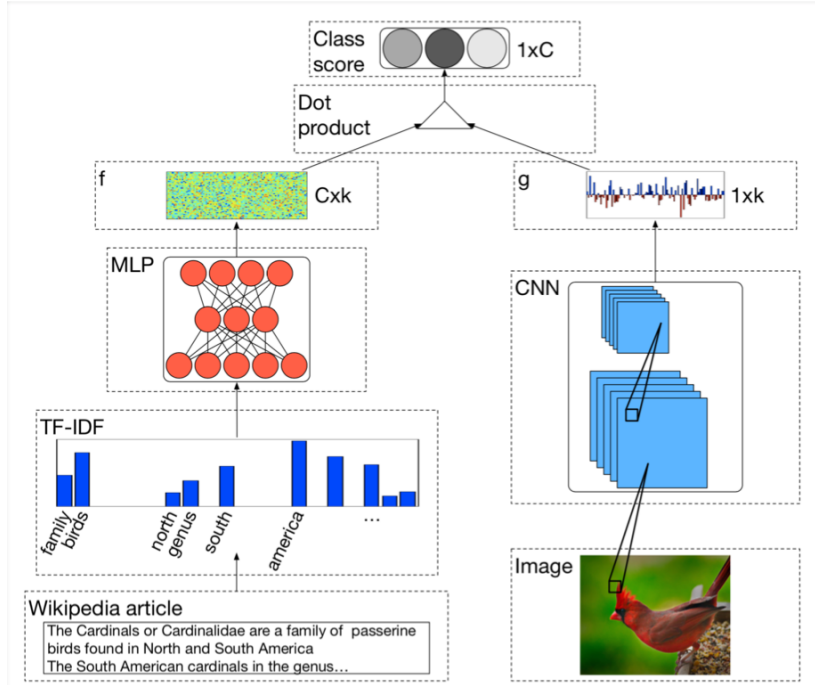
- Test data

  $\{x_i, y_i\}_i$

  where $y_i \in \{C+1, \ldots, C+M\}$

- Assumption

  $x_i, y_i \sim \mathbf{1}_{\{y \in \{C+1,\ldots,C+M\}\}} p(y|x)p(x)$



Predicting Deep Zero-Shot Convolutional Neural Networks Using Textual Descriptions.
LJ Ba, K Swersky, S Fidler, R Salakhutdinov - ICCV, 2015

Deep Learning (Introduction)

Tasks Deep Learning Excels

- Image/Video Recognition

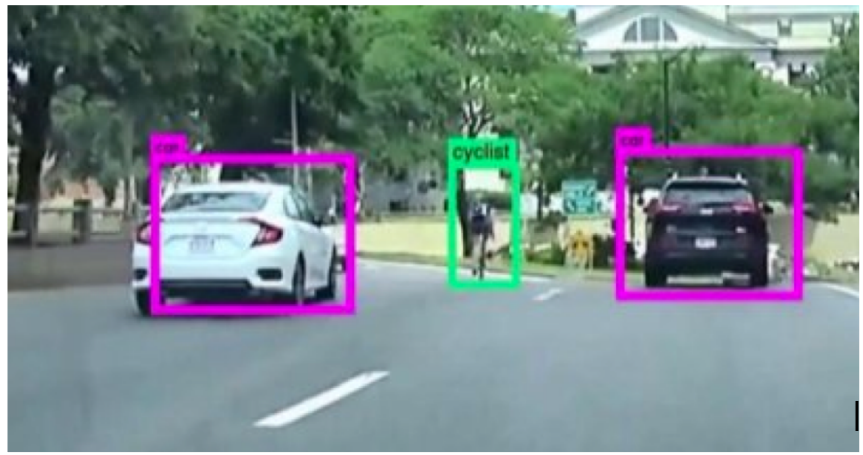


Image by Neurala

- Text Recognition / Prediction
  - The quick brown fox jumps over the lazy dog

- Speech Recognition

⇨ quick brown fox

(CVPR 2016) Face2Face: Real-time Face Capture and Reenactment of RGB Videos, Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, Matthias Nießner

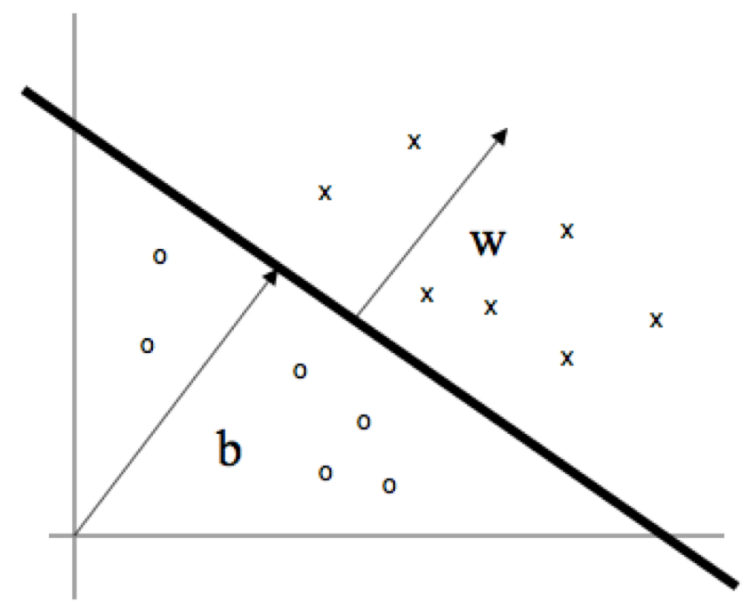
Perceptron Review

Perceptron

Model:

$$f(x) = \sum_{i=1}^{m} w_i x_i + b$$

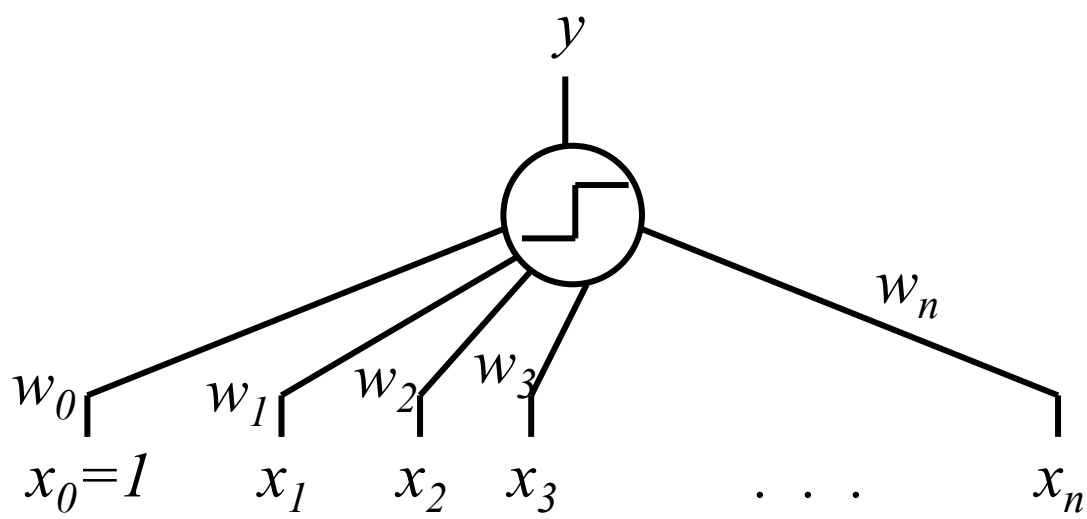
$$y = sign[f(x)]$$



Dot product is product of:
(i) projection of x onto w, and (ii) the length of w

Dot product is 0 if x is perpendicular to w
Add b, if >0 then positive class, if <0 then negative class

Perceptron Structure

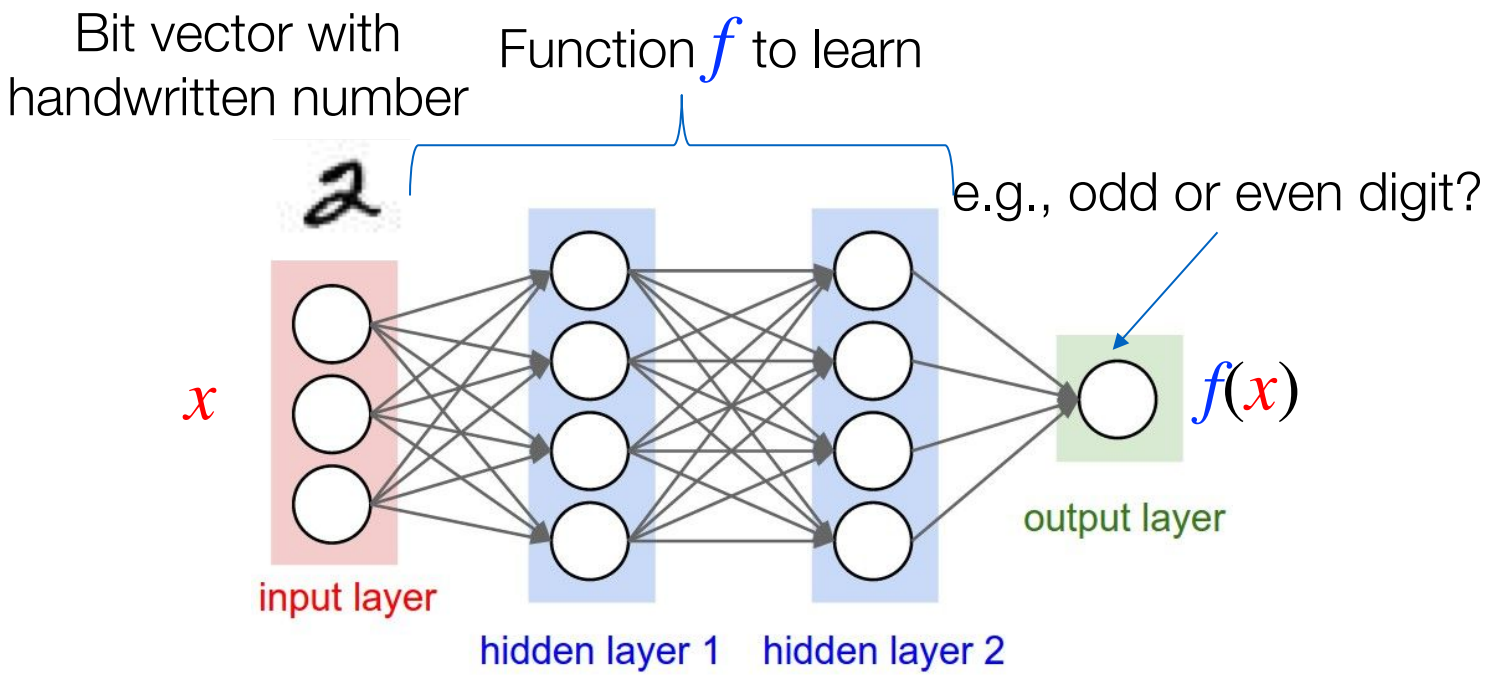


$$y = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 \text{ otherwise} \end{cases}$$

$b = x_0 w_0$ compensates for fixed threshold "> 0”

Neural Network

- More general than perceptron
- *Input layer* is a integer or real-valued vector, representing the data
- *Hidden layers* represent a latent (hidden) variables (hard to interpret)
- *Output layer* represents prediction we want to make



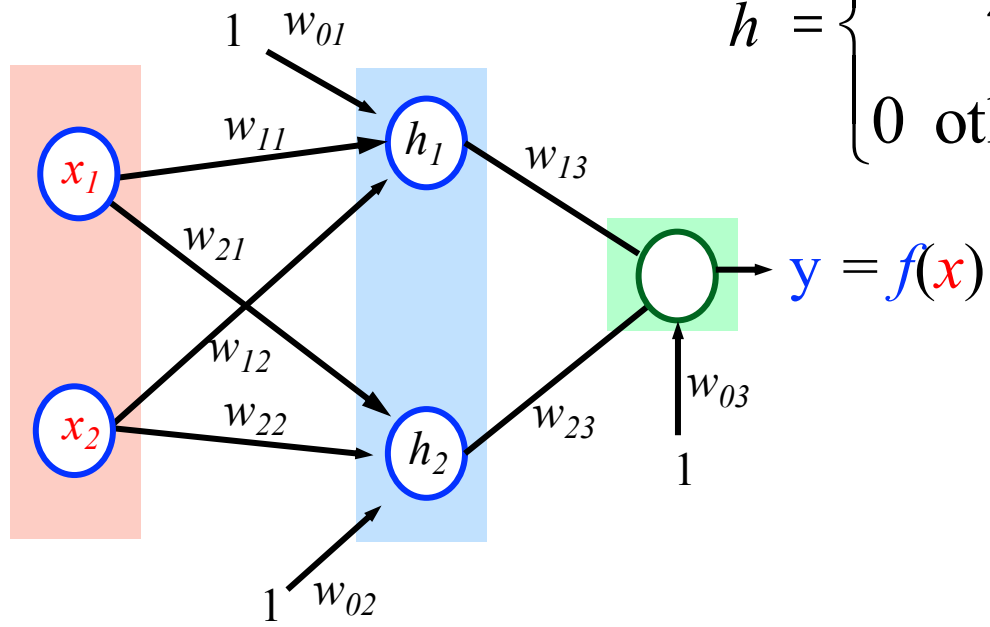
Neural Nets

- Pro: More general than perceptrons
  - Not restricted to linear discriminants
  - Multiple outputs: one classification each
- Con: No simple, guaranteed training procedure
  - Use greedy, hill-climbing procedure to train
  - “Gradient descent”, “Backpropagation”

Example: Solving the XOR Problem

$$h = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 \text{ otherwise} \end{cases}$$

Network Topology:
2 hidden nodes
1 output



Desired behavior:

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

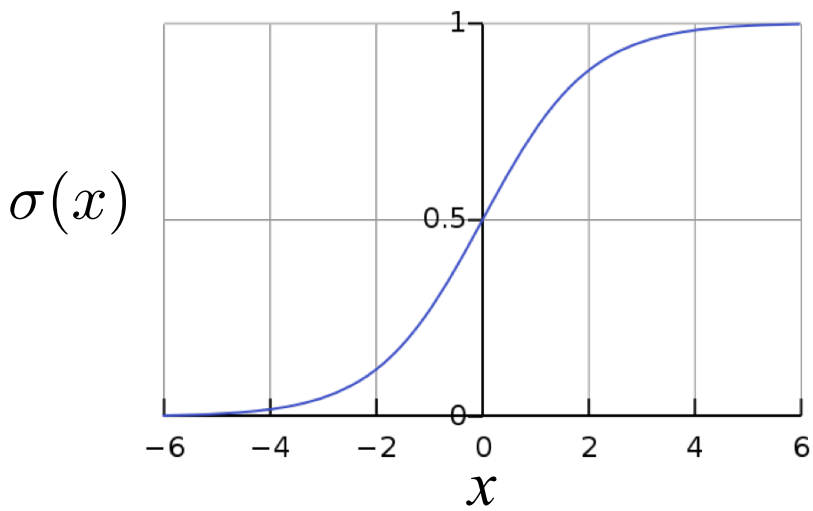
Weights:

- $w11 = w21 = 1$
- $w12 = w22 = 1$
- $w01 = -3/2$; $w02 = -1/2$; $w03 = 1/2$
- $w13 = -1$; $w23 = 1$

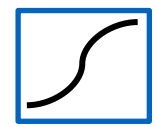
More complex examples

Logistic (neuron) Activation (non-linear filter)

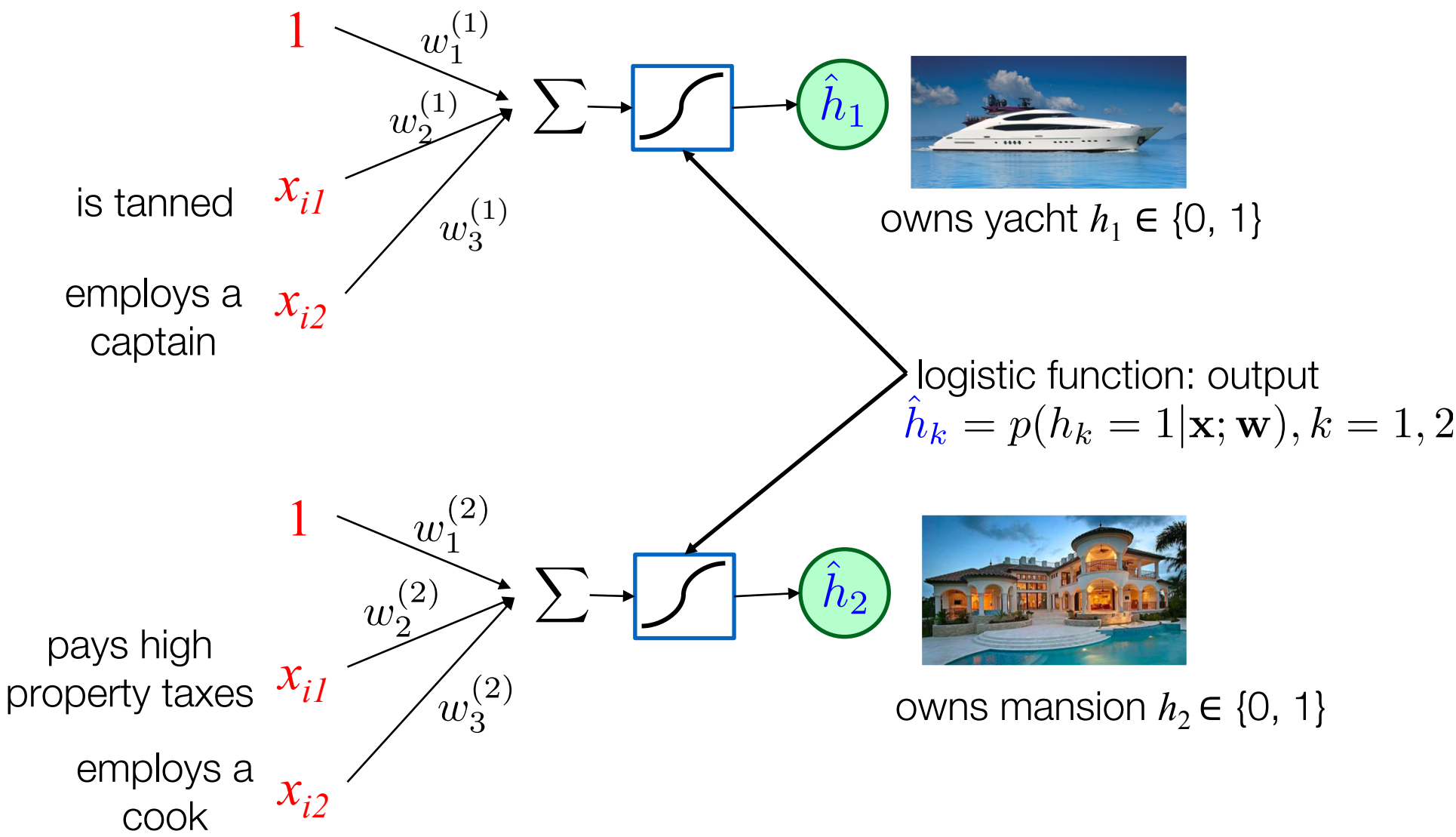
- If input is $x = \mathbf{w}^T\mathbf{x}$, the output will look like a probability $\sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$
- $p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$



$$\sigma(x) = \frac{\exp(x)}{1 + \exp(x)}$$

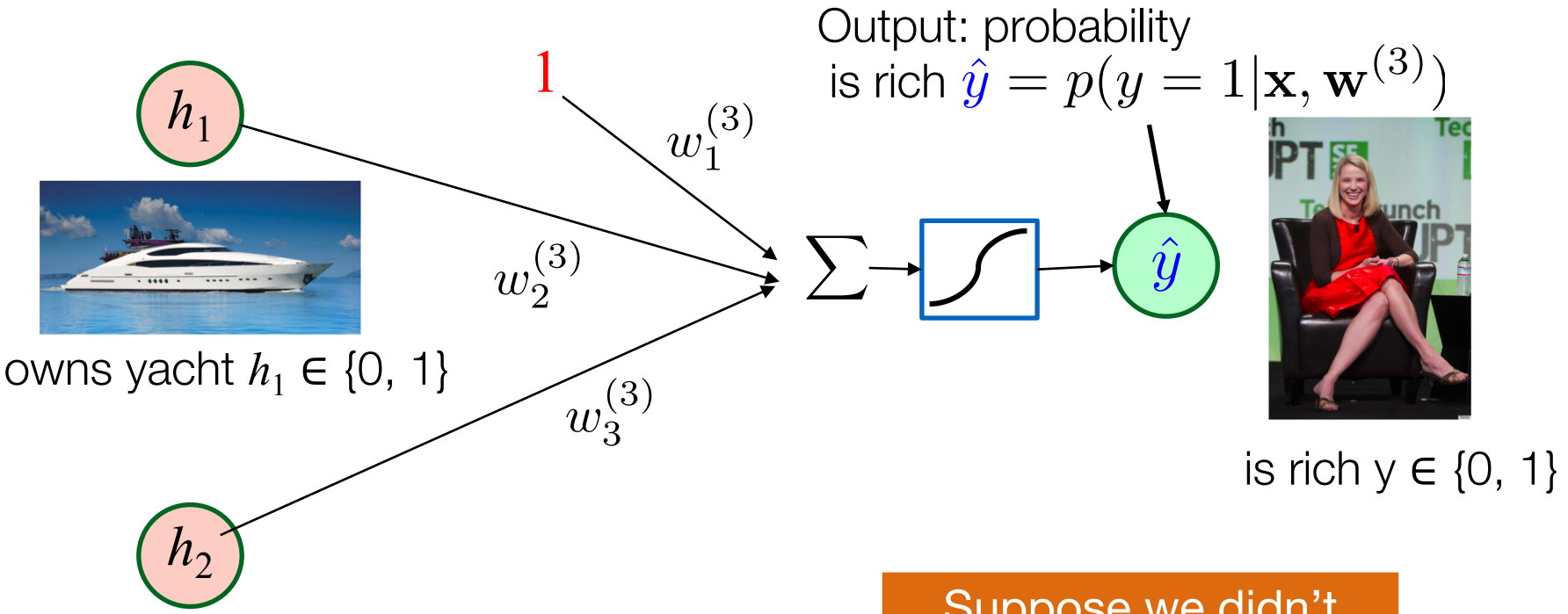
- We will represent the logistic function with the symbol: 

Single Neurons (Layer 1) : Logistic Regression



Top Neuron in Isolation (Layer 2)

- Assume we know these facts: yacht and mansion ownership



Suppose we didn't know whether person has mansion, yacht

Can we use previous models together with the above model?

Assume unobserved Intermediate Values (Hidden Layers)
Model M1

(hidden) owns yacht:
probability ∈ [0,1]

is rich y ∈ {0, 1}

is tanned

employs a captain

pays high property taxes

employs a cook

(hidden) owns mansion:
probability ∈ [0, 1]

Mansion and yacht just aids to determine if person is rich

We don't know / don't care if person owns these items

Equivalent model: Fully connected with zero weights between yacht and mansion parts of the model

Model M2

owns yacht
probability ∈ [0,1]

is rich y ∈ {0, 1}

owns mansion
probability ∈ [0, 1]

is tanned

employs a captain

pays high property taxes

employs a cook

1 x_{i1} x_{i2} 1 x_{i1} x_{i2}

$w_1^{(1)}$ $w_2^{(1)}$ $w_3^{(1)}$ $w_1^{(2)}$ $w_2^{(2)}$ $w_3^{(2)}$

$w_5^{(1)} = 0$ $w_4^{(1)} = 0$ $w_4^{(2)} = 0$ $w_5^{(2)} = 0$

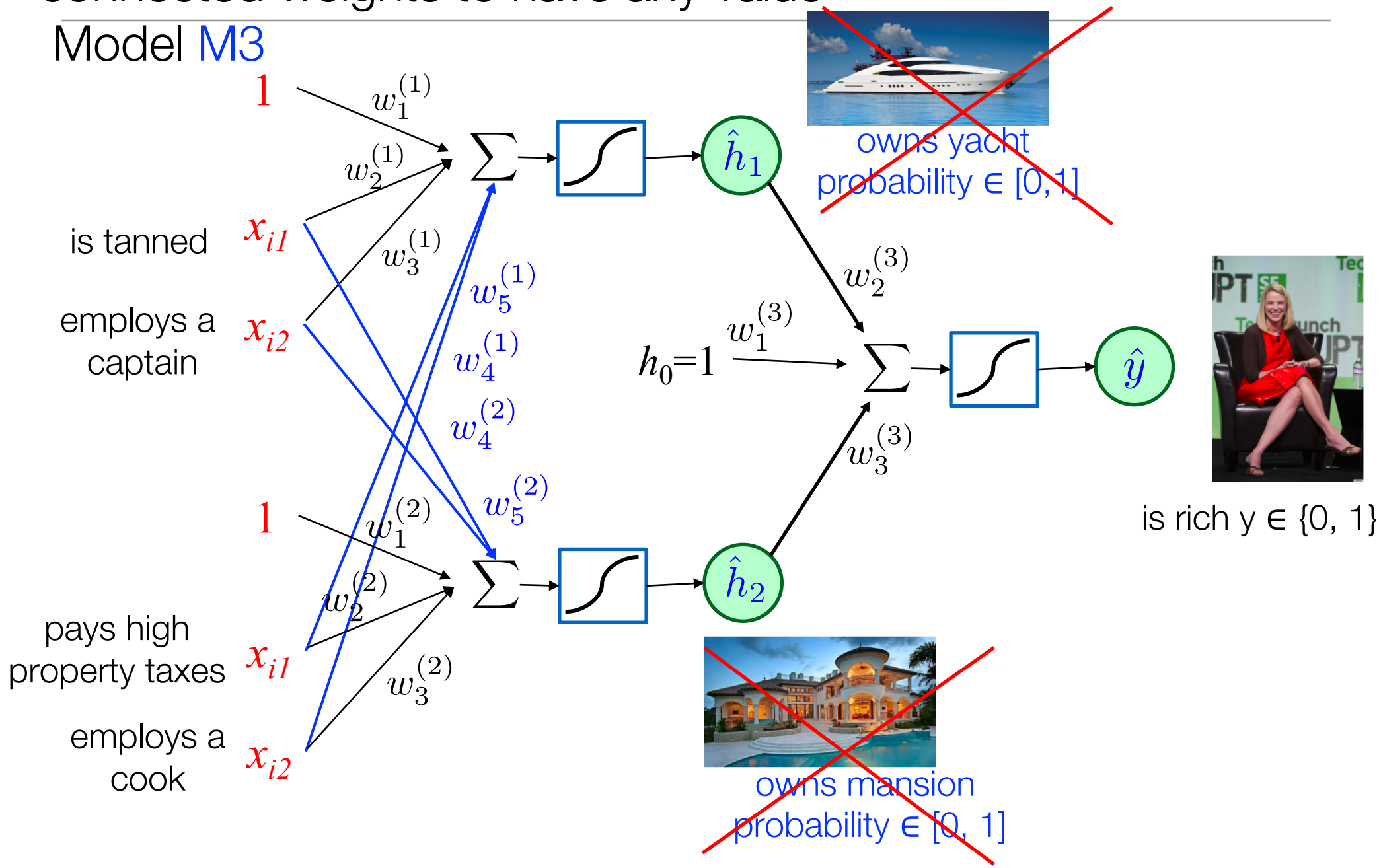
$\hat{h}_1$ $\hat{h}_2$ h_0=1

$w_2^{(3)}$ $w_1^{(3)}$ $w_3^{(3)}$

$\hat{y}$

More flexible model: Fully connected allowing inter-connected weights to have any value

Model M3



Model search for our example

- Training data: $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- $\mathbf{x} = (1, x_{11}, x_{12}, x_{21}, x_{22})$
- $\mathbf{w}^{(k)} = (b, w_2^{(k)}, w_3^{(k)}, w_4^{(k)}, w_5^{(k)})$, $k = 1, 2$, where $b = w_1^{(1)} + w_1^{(2)}$
- $\mathbf{w}^{(3)} = (w_1^{(3)}, w_2^{(3)}, w_3^{(3)})$

Optimize using maximum likelihood estimation

Characteristics of user *i (tan, captain, cook,...)*

$$\arg\max_{\mathbf{w}^{(1)},\mathbf{w}^{(2)},\mathbf{w}^{(3)}} \frac{1}{n}\sum_{i=1}^{n} \log p(y = y_i|\mathbf{x}_i; \mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)})$$

User *i* in training data: is rich $y_i \in \{0, 1\}$

$$= \frac{1}{n}\sum_{i=1}^{n} y_i \log \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma((\mathbf{w}^{(3)})^T \mathbf{h}(\mathbf{x}_i))),$$

where $\sigma(x) = \frac{\exp(x)}{1+\exp(x)}$, and $\mathbf{h}(\mathbf{x}) = (1, \hat{h}_1(\mathbf{x}), \hat{h}_2(\mathbf{x}))$,

$$\hat{h}_1(\mathbf{x}) = p(h_1 = 1|\mathbf{x}) = \sigma((\mathbf{w}^{(1)})^T \mathbf{x})$$

and

$$\hat{h}_2(\mathbf{x}) = p(h_2 = 1|\mathbf{x}) = \sigma((\mathbf{w}^{(2)})^T \mathbf{x})$$

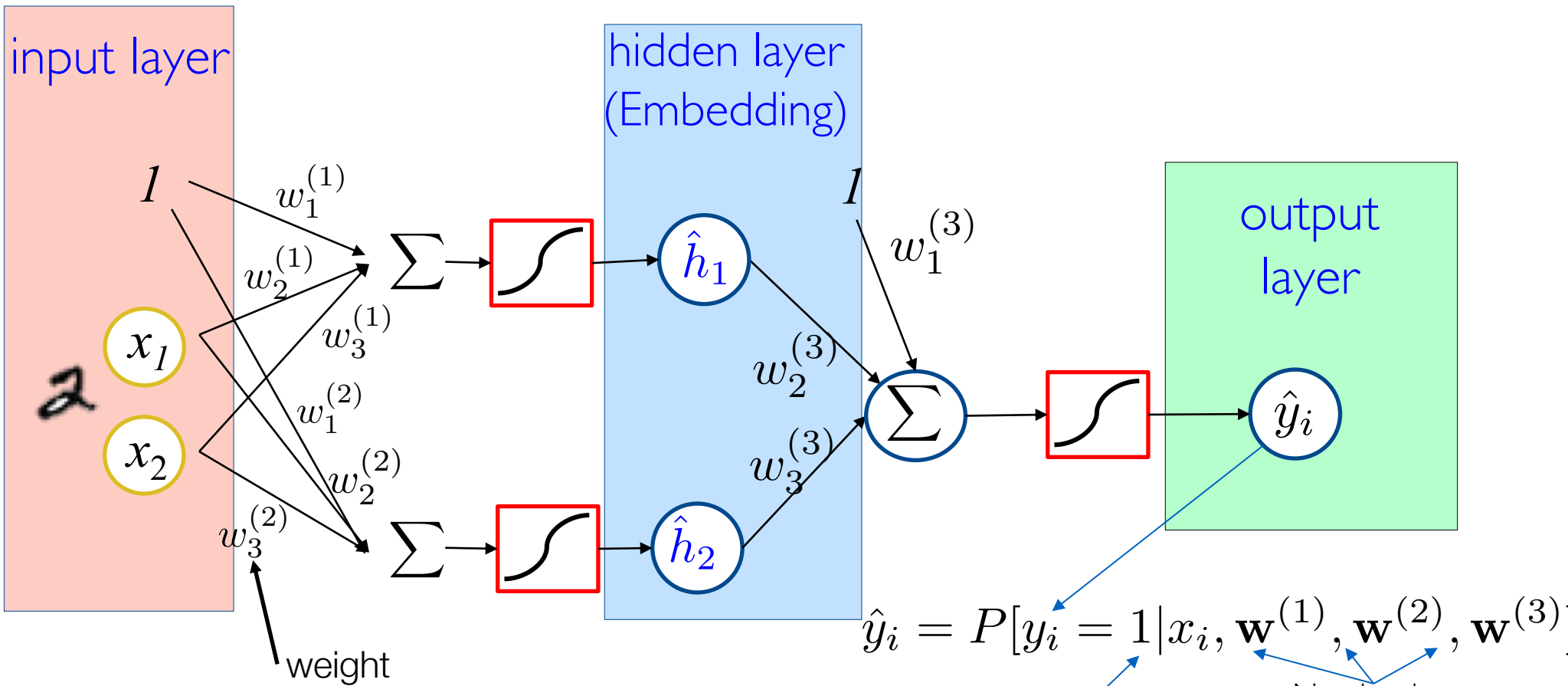
Questions

- What is the training data?
- What is the model?
- What is the score function?
- What is the search procedure?

- Which model is more likely to overfit the data? M1, M2, or M3?
    - Why are M1 and M2 equivalent?

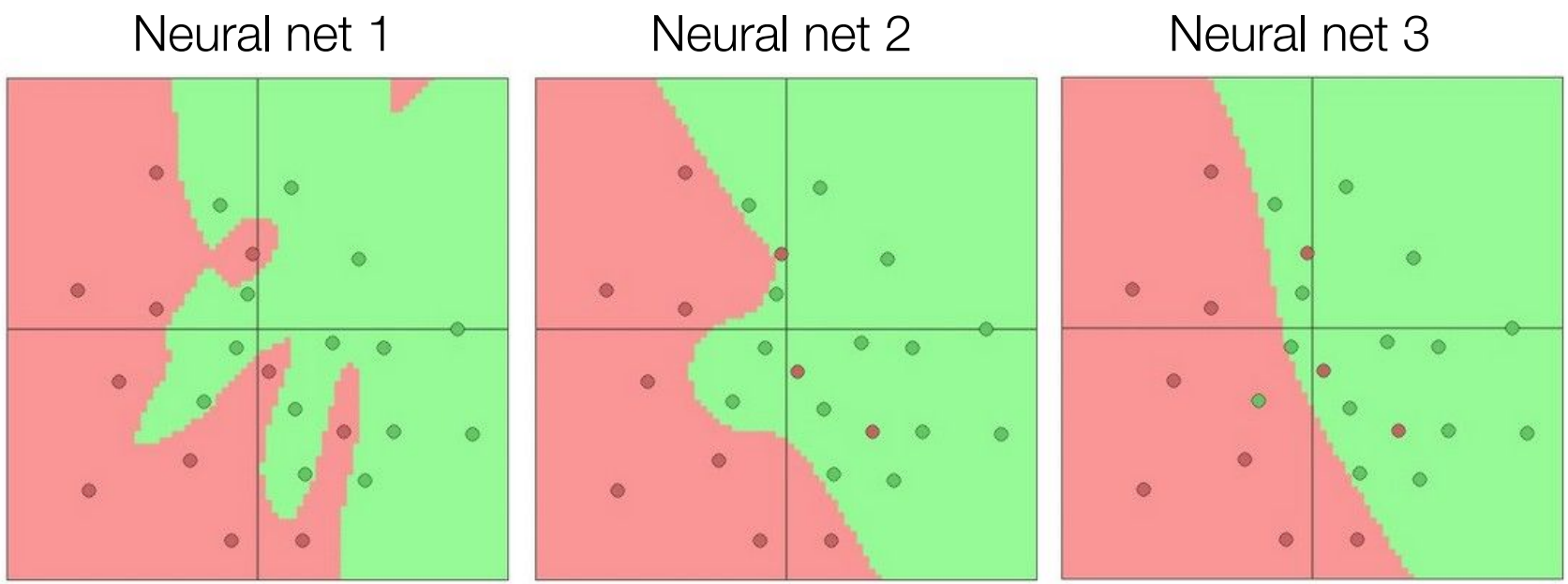
Neural Network Definitions

Neural networks are a combination of linear and non-linear functions



Neural Network = Ensemble of "Simple" Models

- Just like with ensemble classifiers, more complex models (e.g. logistic) requires care not to overfit the training data



Next Class: Searching for good neural network parameters (optimization)