

26: Intro to Deep Learning

Jerry Cain
March 11, 2024

[Lecture Discussion on Ed](#)



Deep Learning

Innovations in deep learning



AlphaGO (2016)

Lisa Yan, Chris Piech, Mehran Sahami, and Jerry Cain, CS109, Winter 2024

Making history

AlphaGo is the first computer program to defeat a professional human Go player, the first to defeat a Go world champion, and is arguably the strongest Go player in history.

Deep learning and neural networks are core theories and technologies behind the current AI revolution.

Errata:

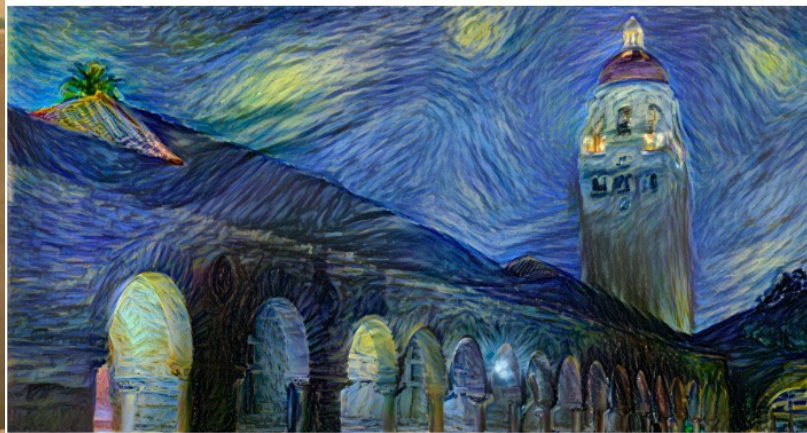
- Checkers is the last **solved** game (from game theory, where perfect player outcomes can be fully predicted from any gameboard).
https://en.wikipedia.org/wiki/Solved_game
- The first machine learning algorithm defeated a world champion in Chess in 1996.
[https://en.wikipedia.org/wiki/Deep_Blue_\(chess_computer\)](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer))

Computers making art



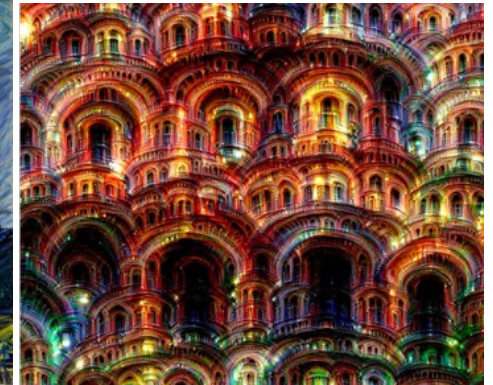
The Next Rembrandt

<https://medium.com/@DutchDigital/the-next-rembrandt-bringing-the-old-master-back-to-life-35dfb1653597>



A Neural Algorithm of Artistic Style

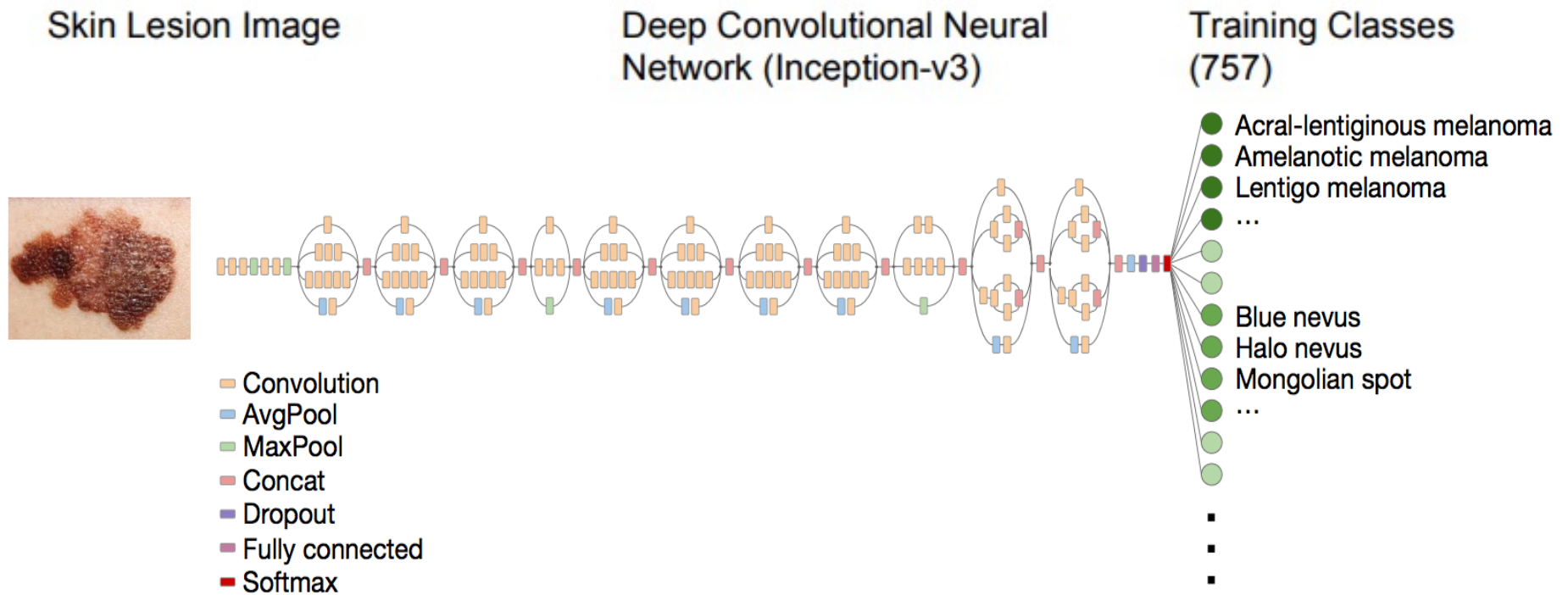
<https://arxiv.org/abs/1508.06576>
<https://github.com/jcjohnson/neural-style>



Google Deep Dream

<https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

Detecting skin cancer



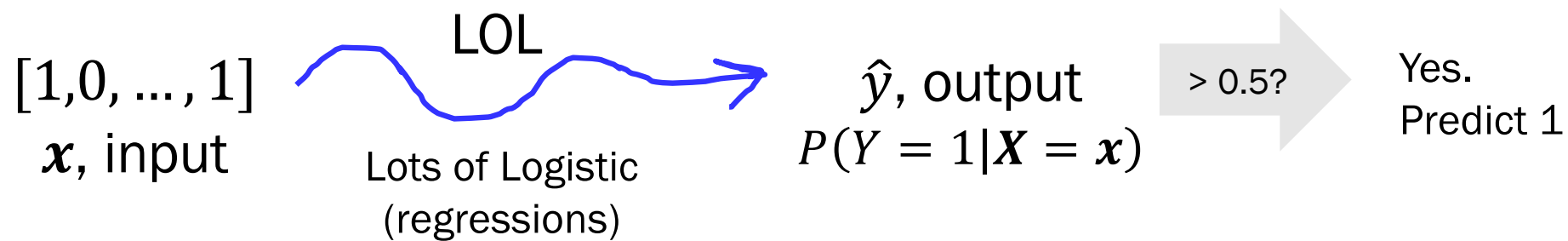
Esteva, Andre, et al. "Dermatologist-level classification of skin cancer with deep neural networks." *Nature* 542.7639 (2017): 115-118.

Lisa Yan, Chris Piech, Mehran Sahami, and Jerry Cain, CS109, Winter 2024

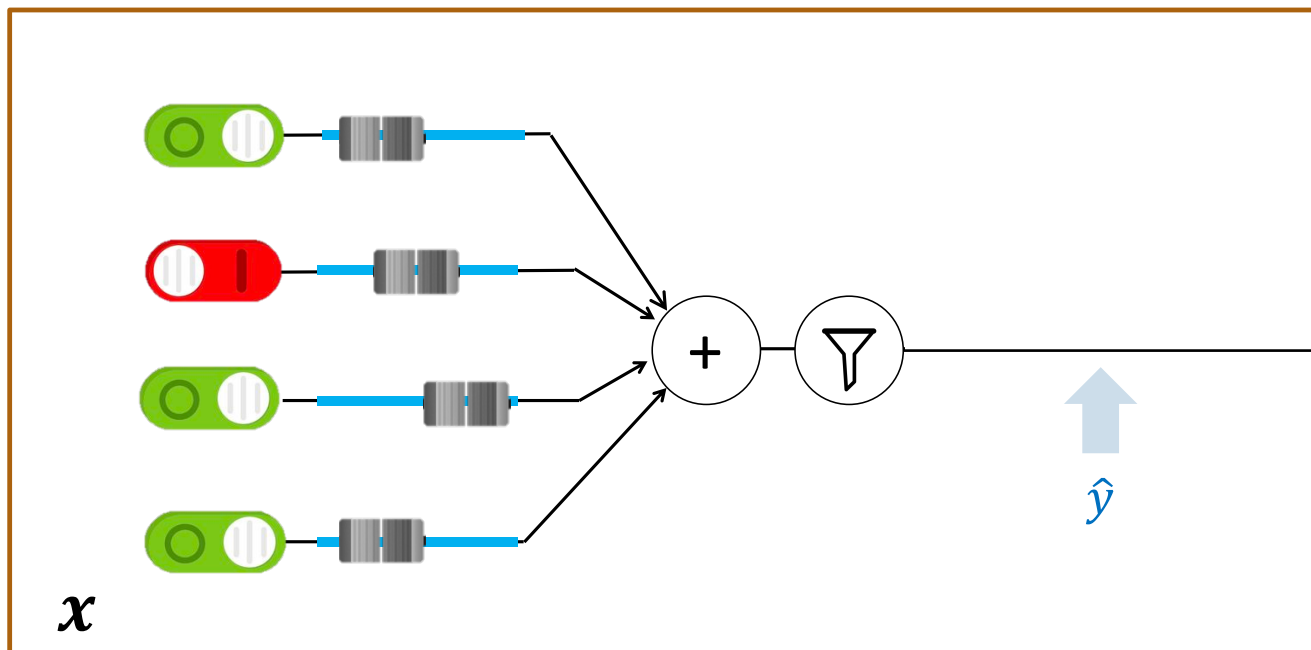
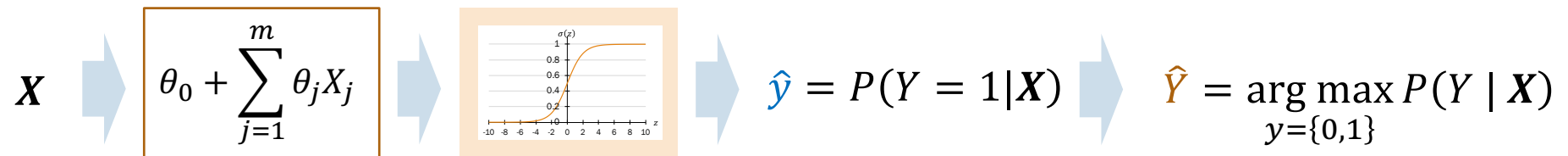
Deep learning

def **Deep learning** is
maximum likelihood estimation
with neural networks.

def A **neural network** is,
at its core, many logistic
regression units stacked on top
of each other.

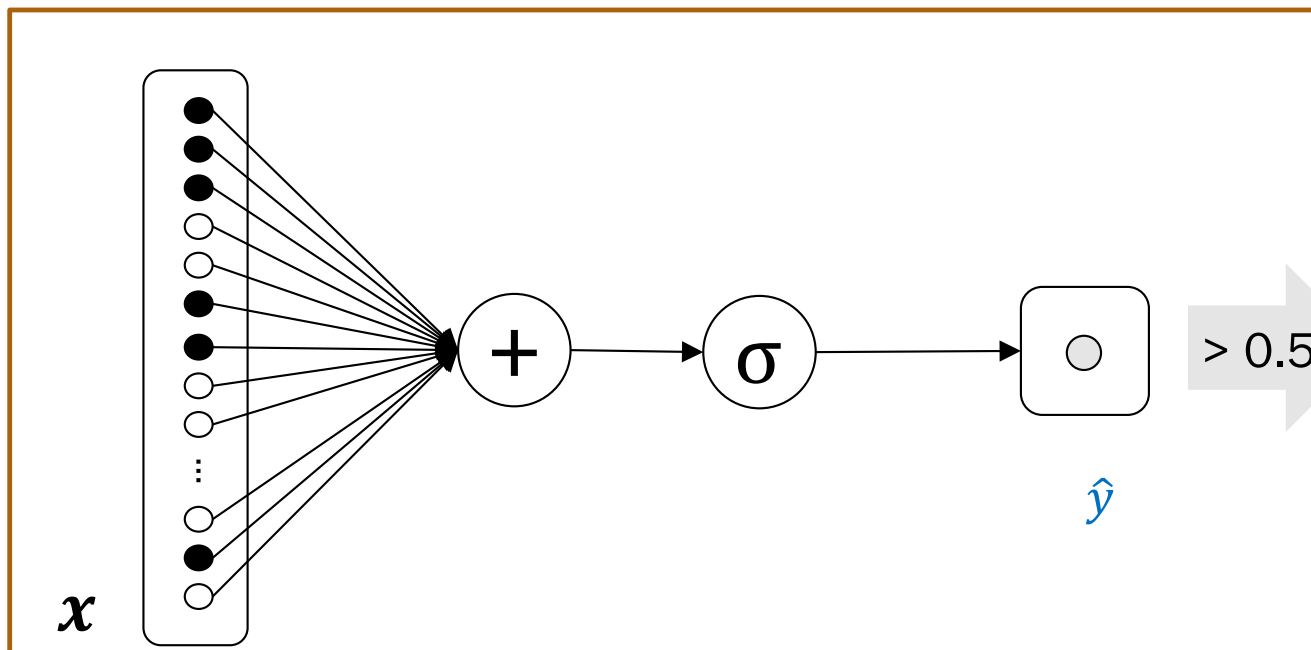
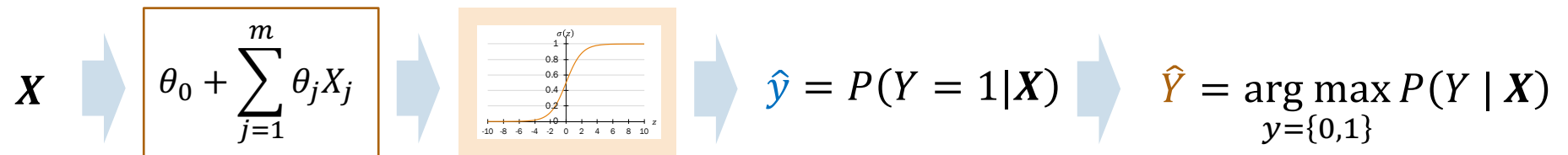


Logistic Regression Model



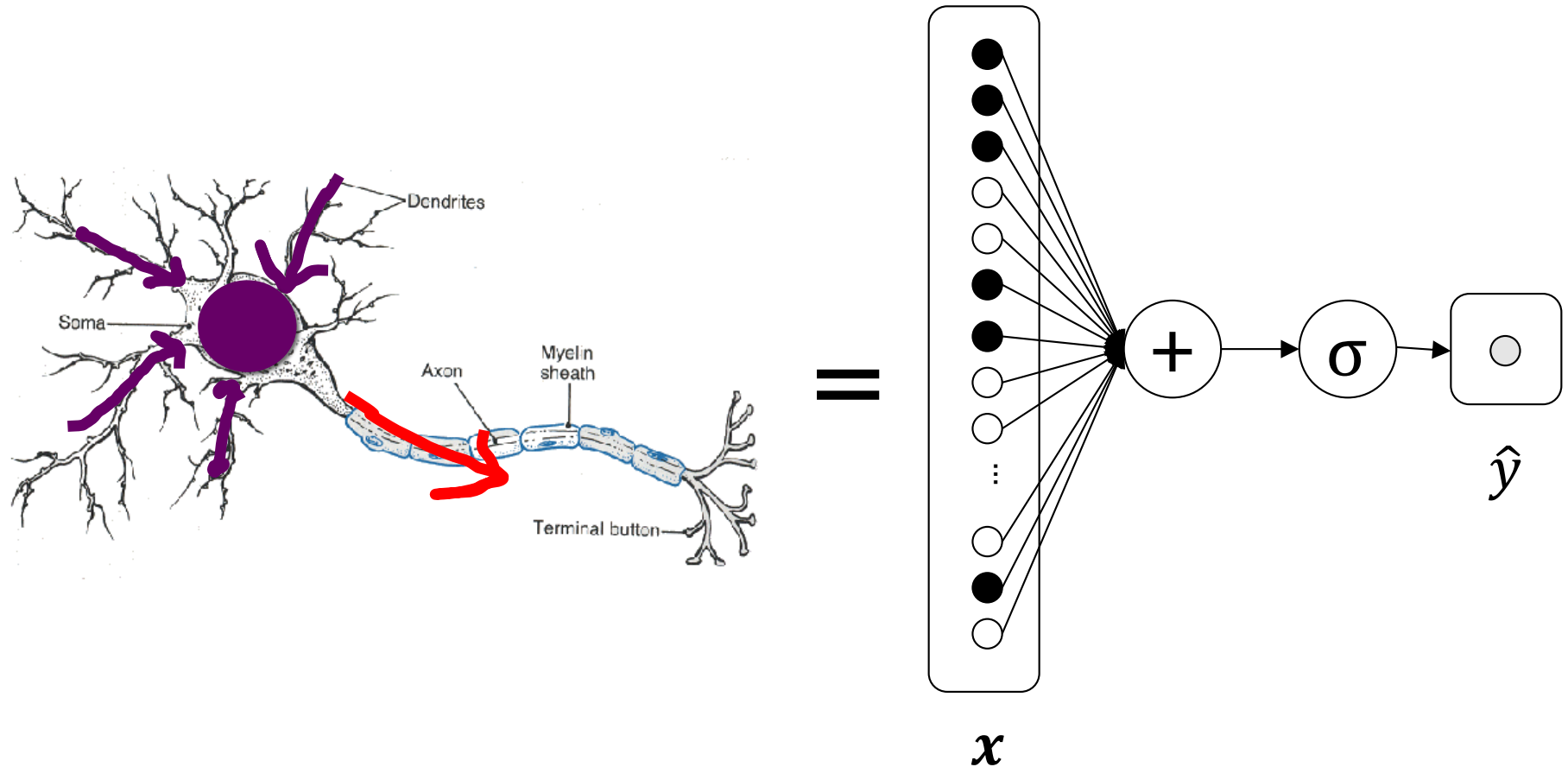
Let's focus on the model up to \hat{y} .

Logistic Regression Model



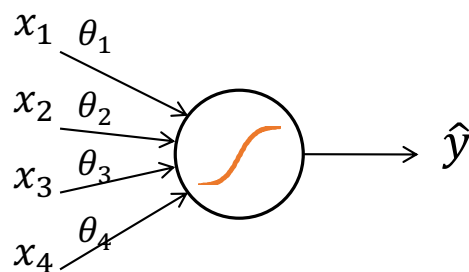
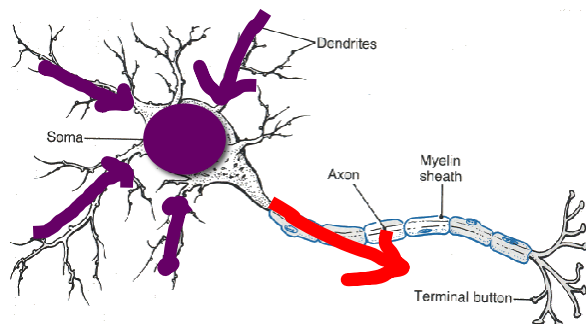
Let's focus on the model up to \hat{y} .

One neuron = One logistic regression



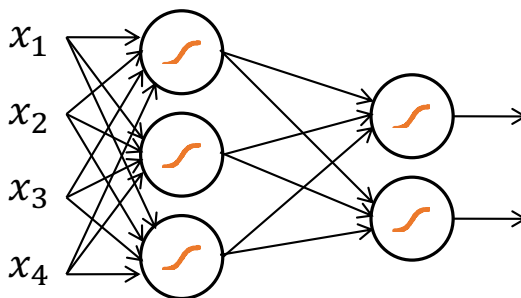
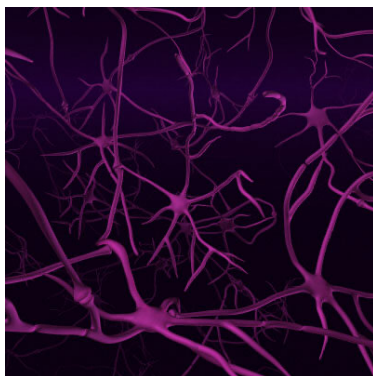
Biological basis for neural networks

A neuron



One neuron =
one logistic
regression

Your brain

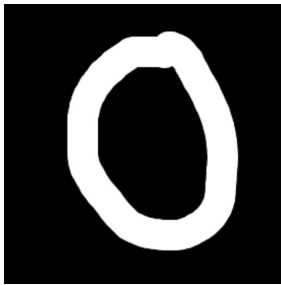


Neural network =
many logistic
regressions

(or rather, someone else's brain)

Digit recognition example

Input image



Input feature vector

$$\mathbf{x}^{(i)} = [0,0,0,0, \dots, 1,0,0,1, \dots, 0,0,1,0]$$

Output label

$$y^{(i)} = 0$$

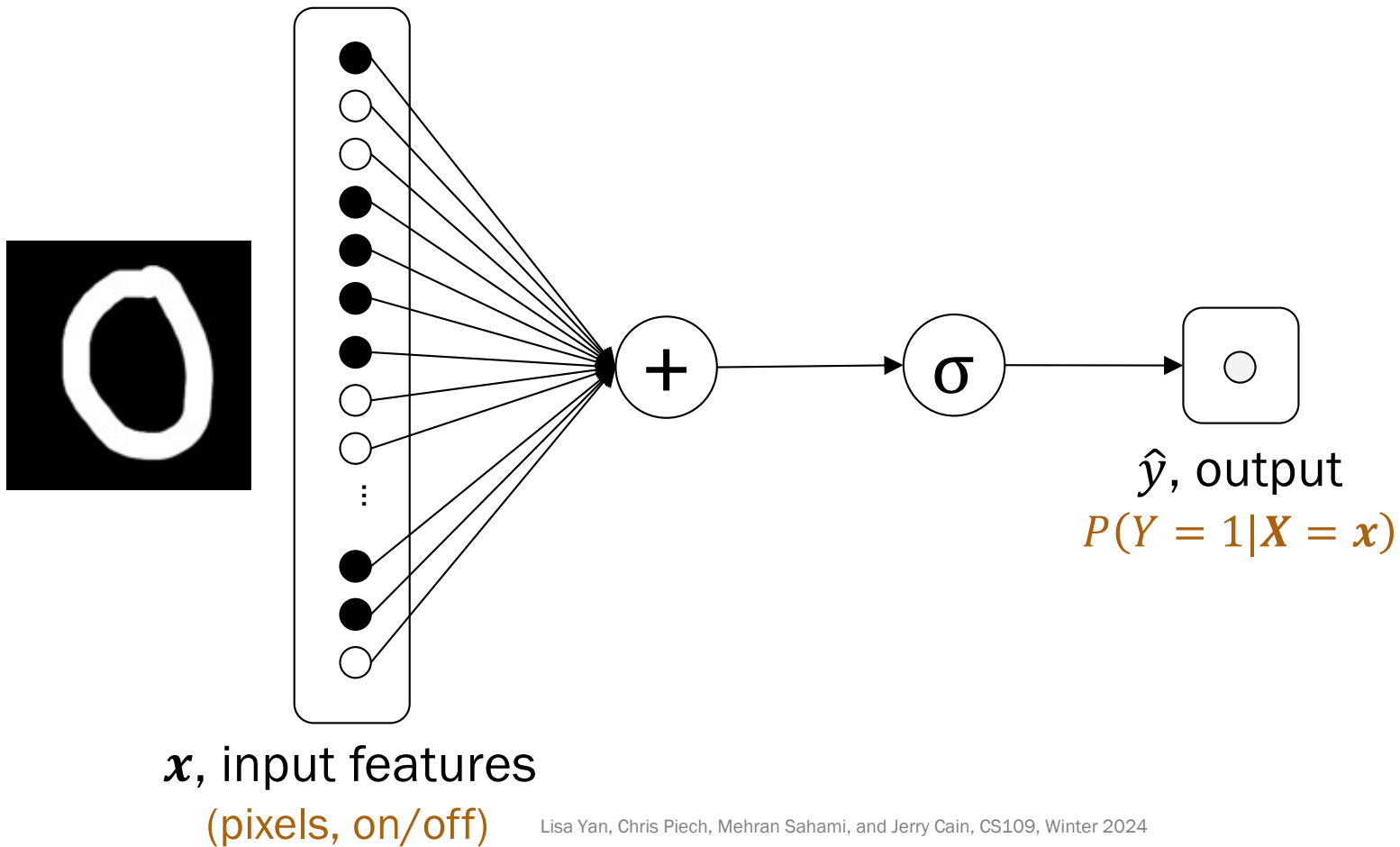


$$\mathbf{x}^{(i)} = [0,0,1,1, \dots, 0,1,1,0, \dots, 0,1,0,0]$$

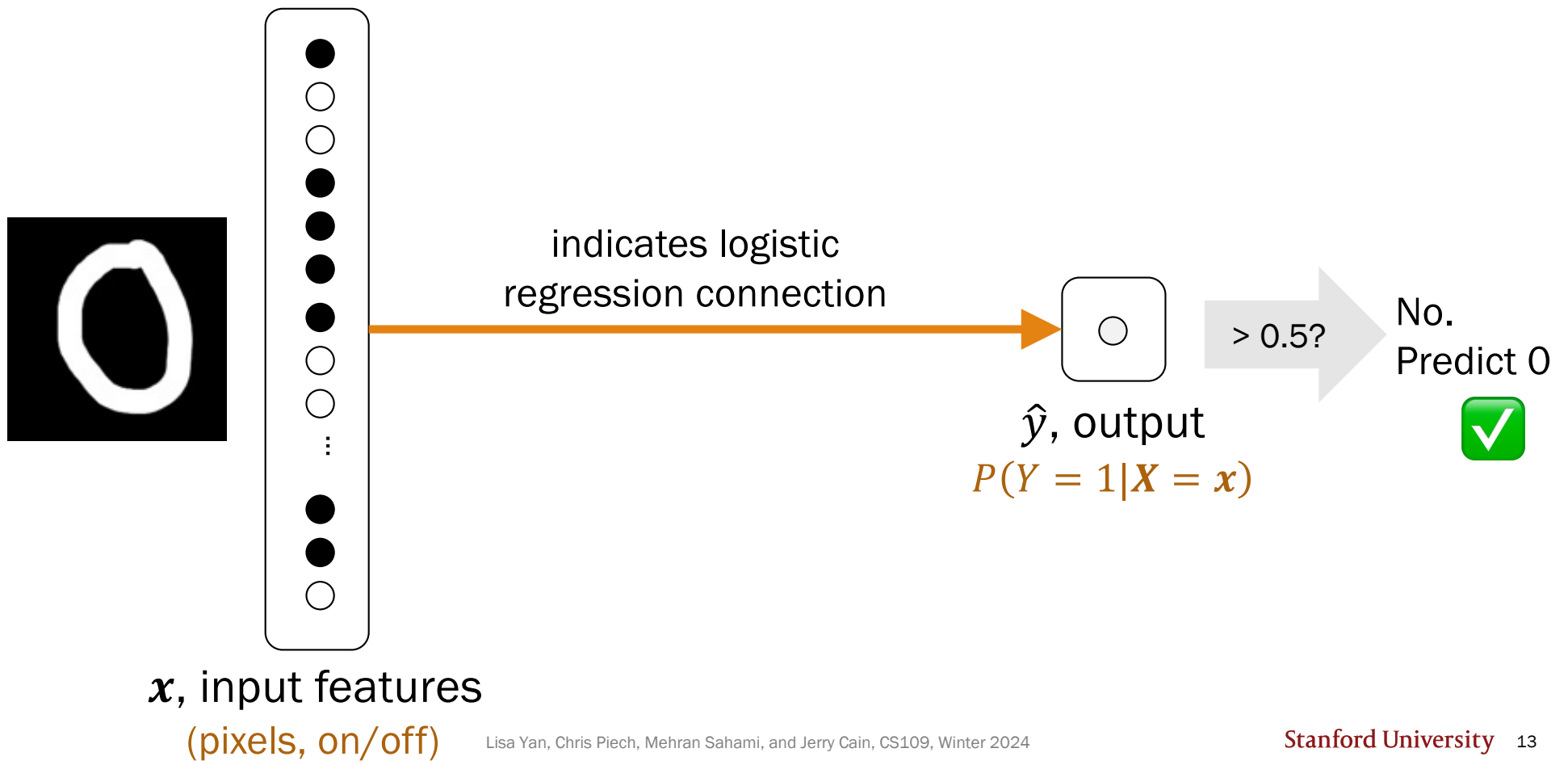
$$y^{(i)} = 1$$

We make feature vectors from digitized pictures of numbers.

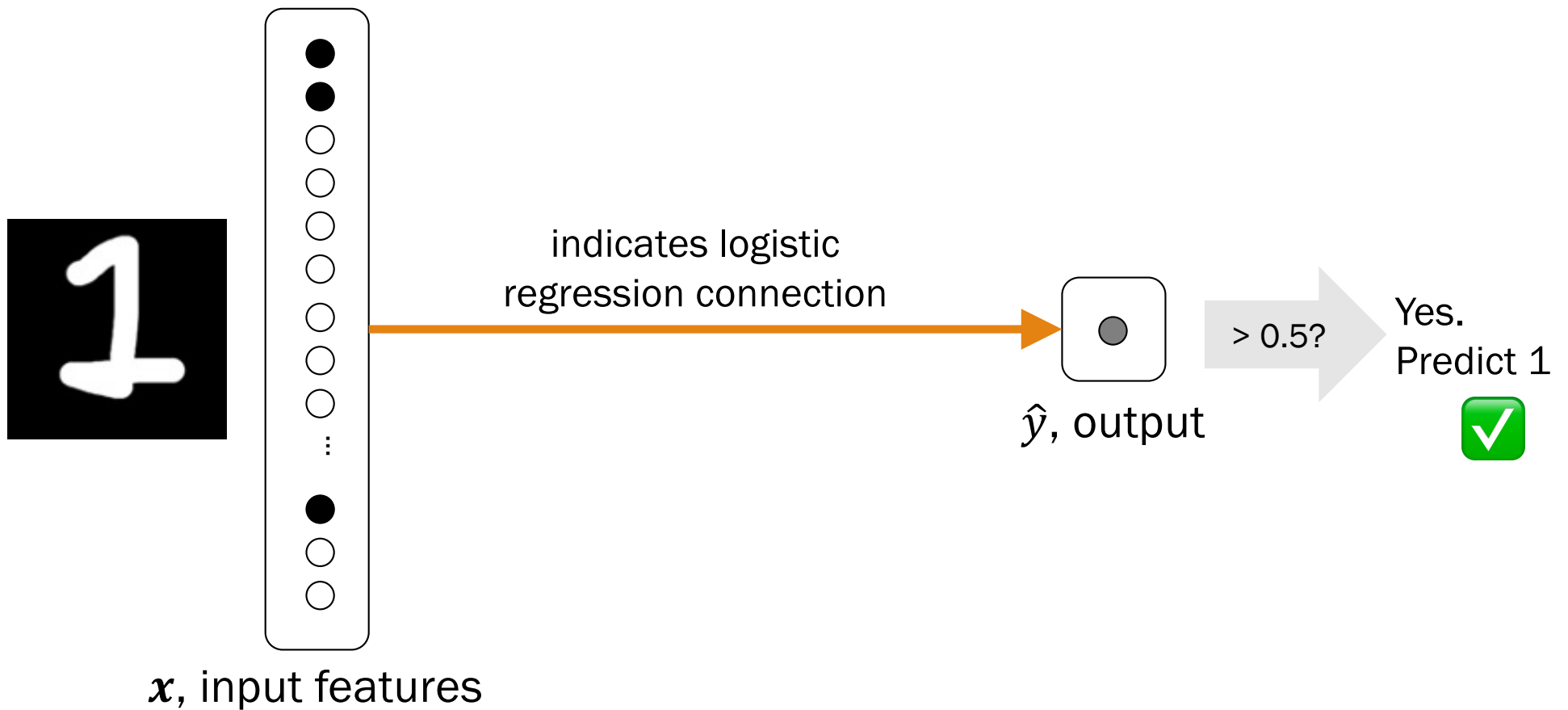
Logistic Regression



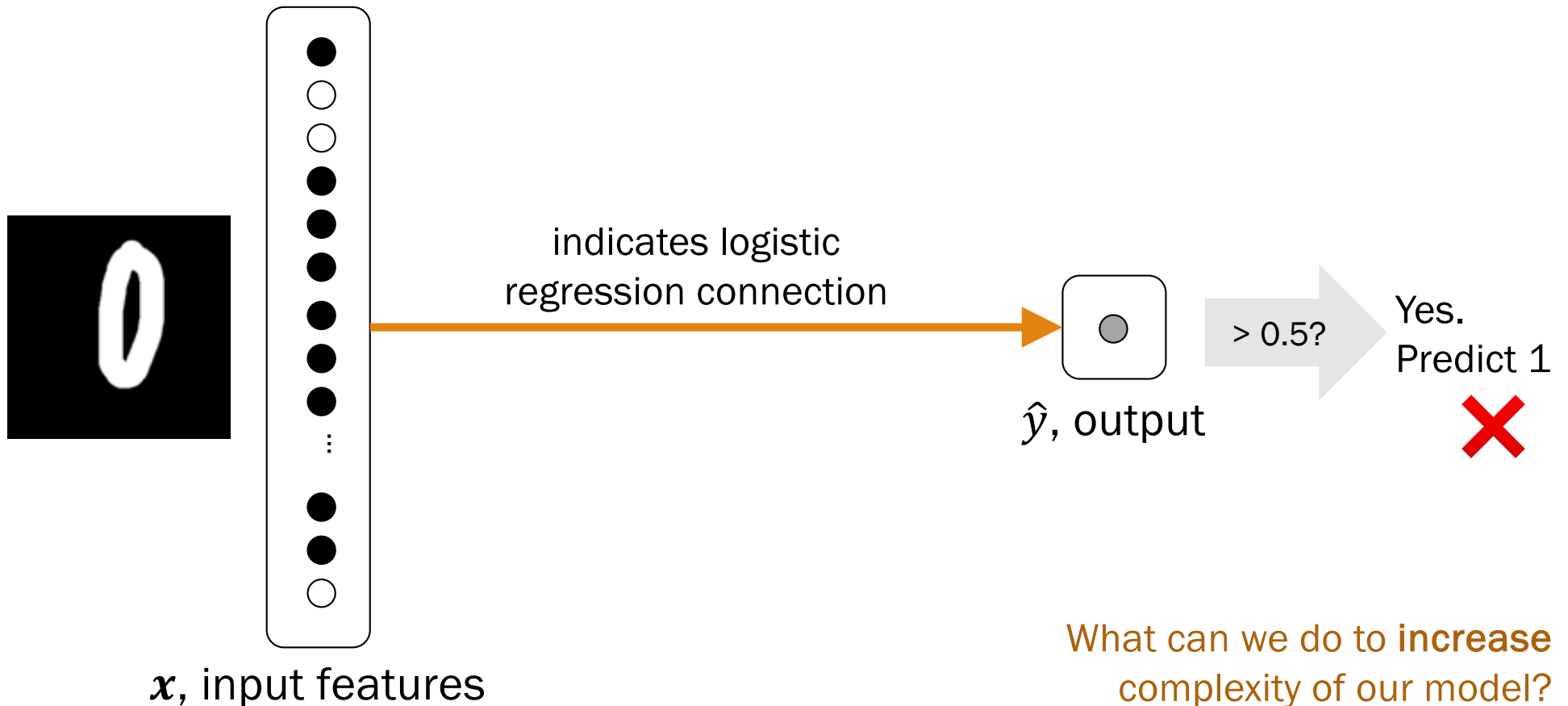
Logistic Regression



Logistic Regression



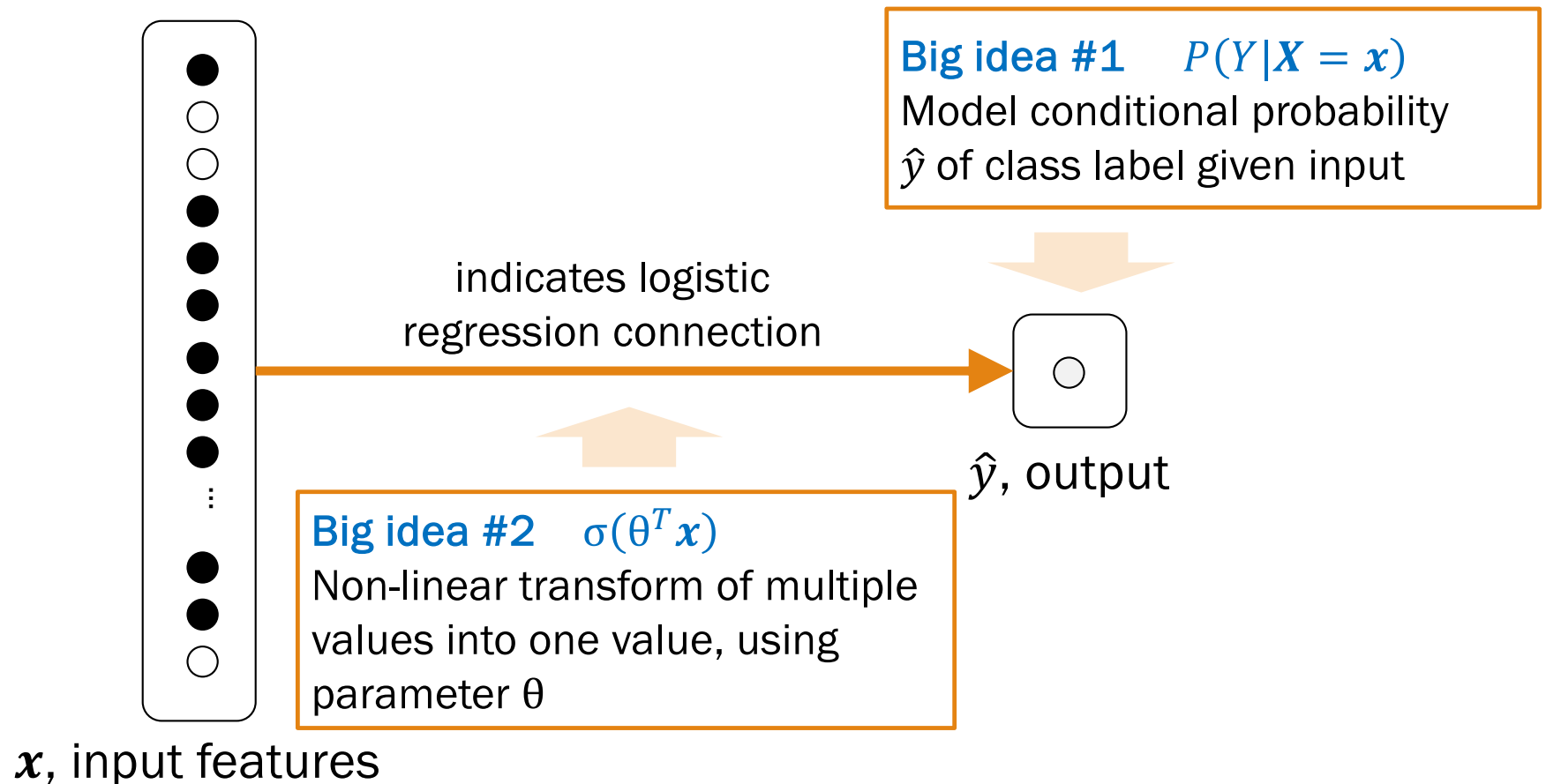
Logistic Regression



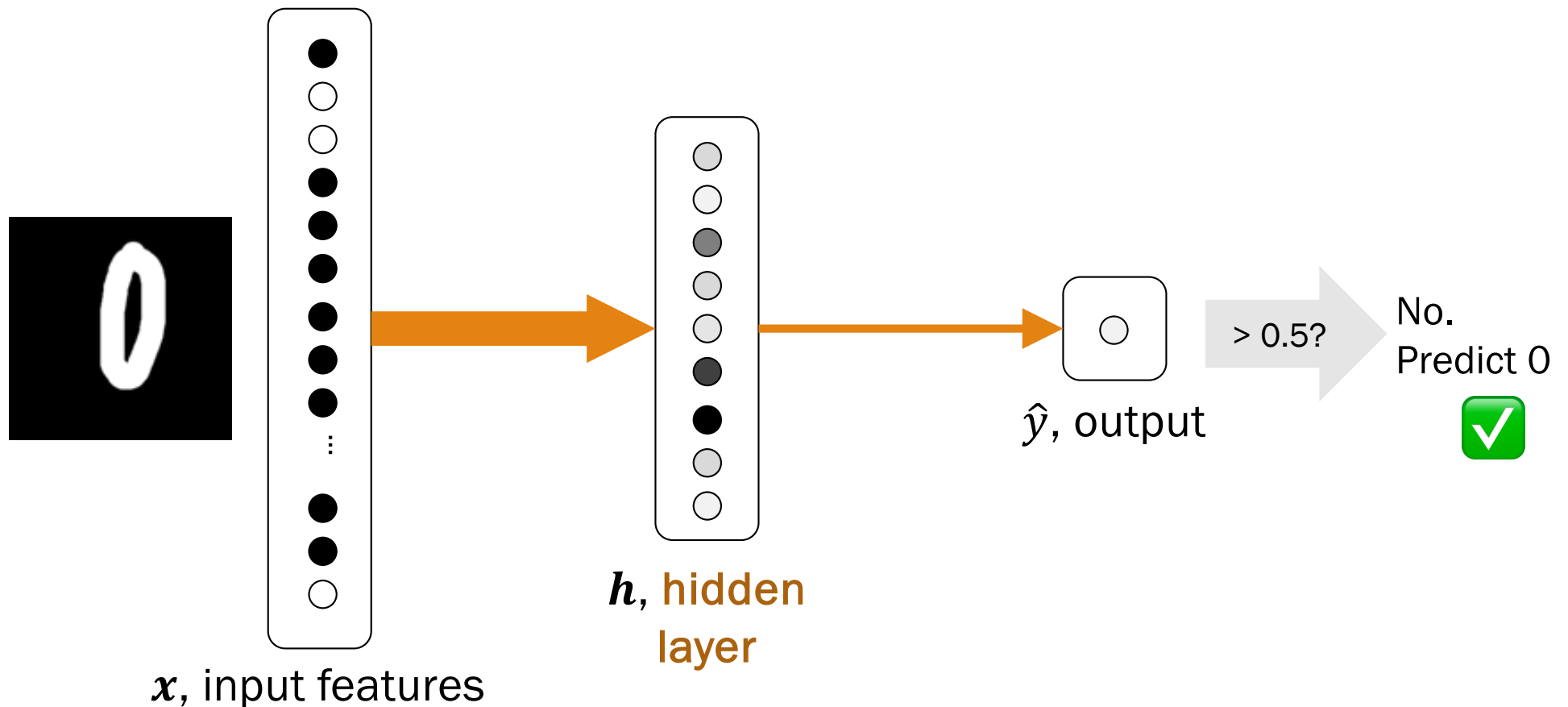
What can we do to **increase** complexity of our model?

Take two big ideas from Logistic Regression

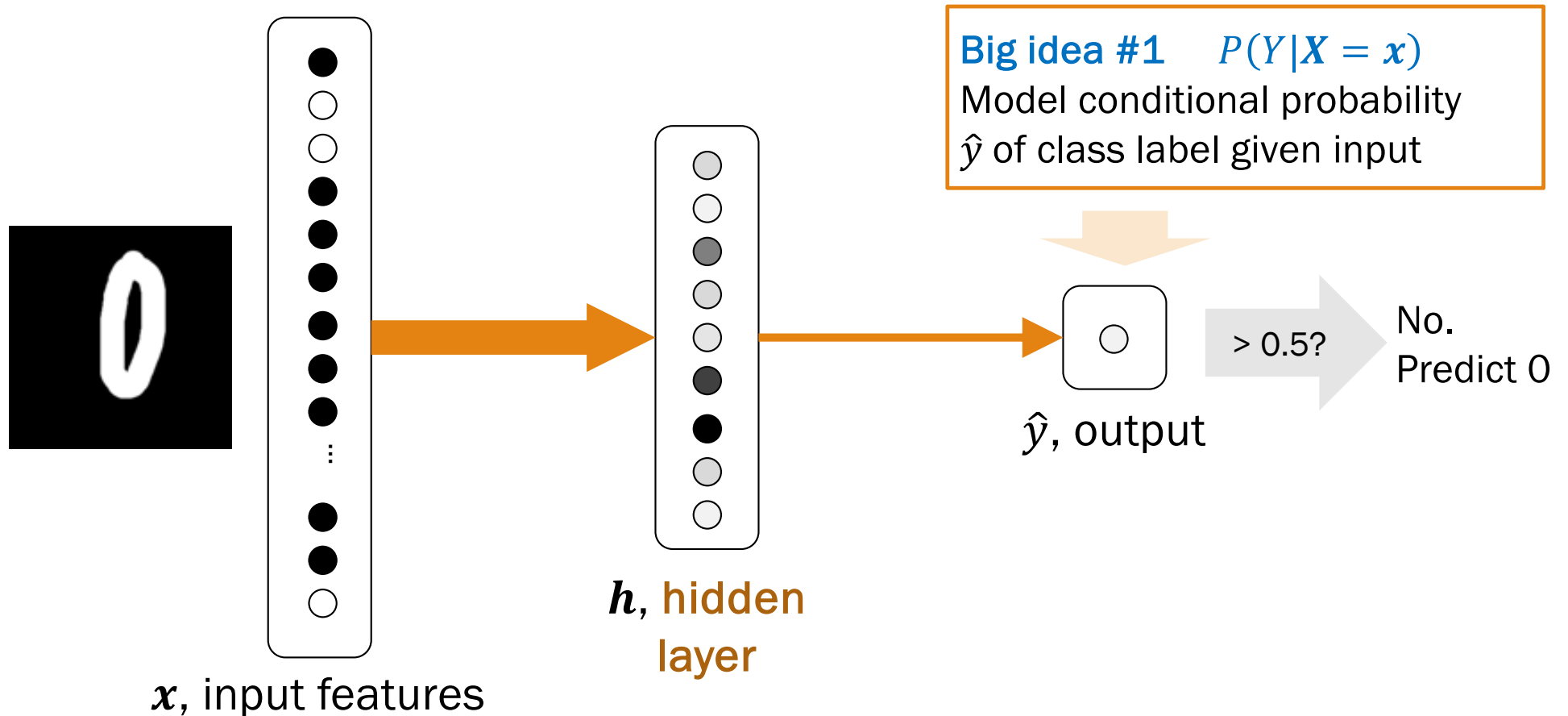
Review



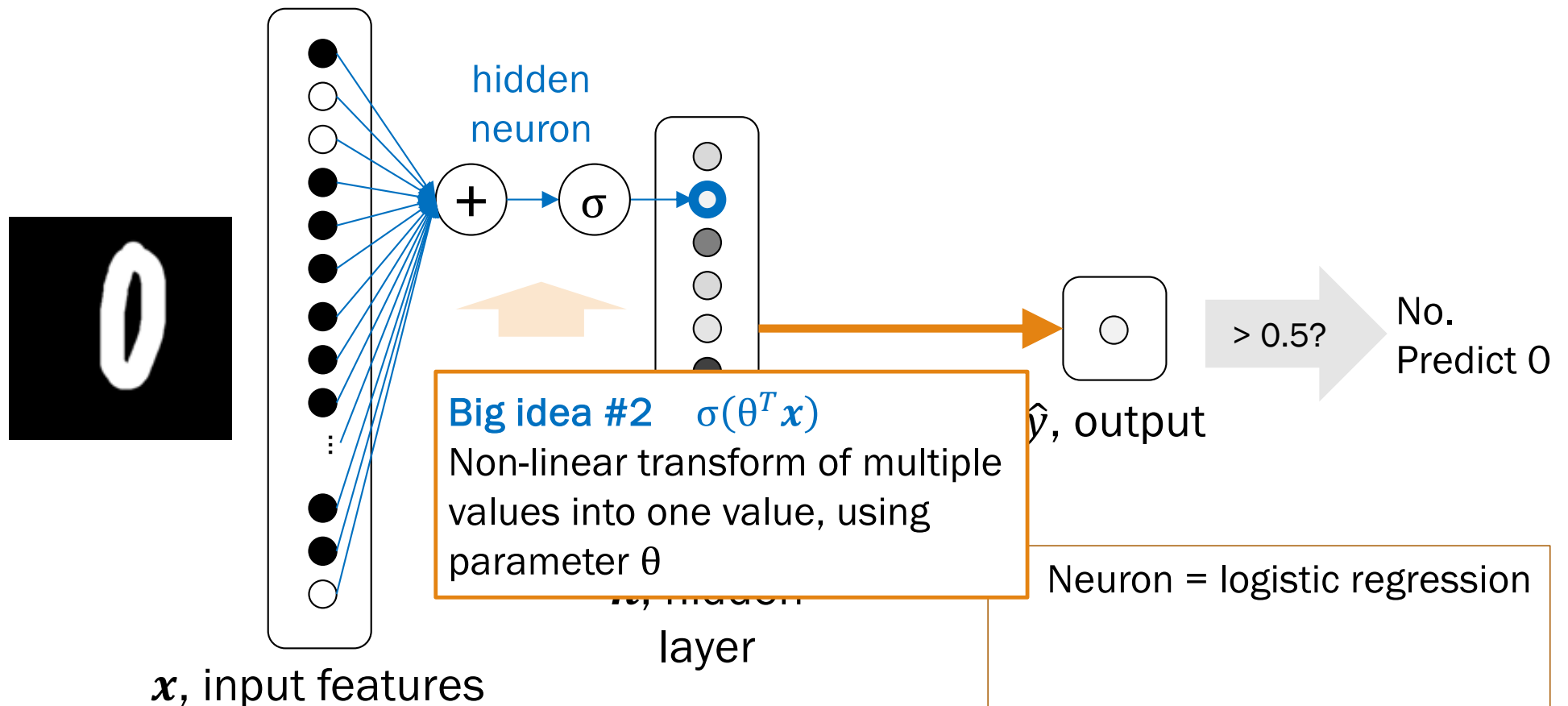
Introducing: The Neural network



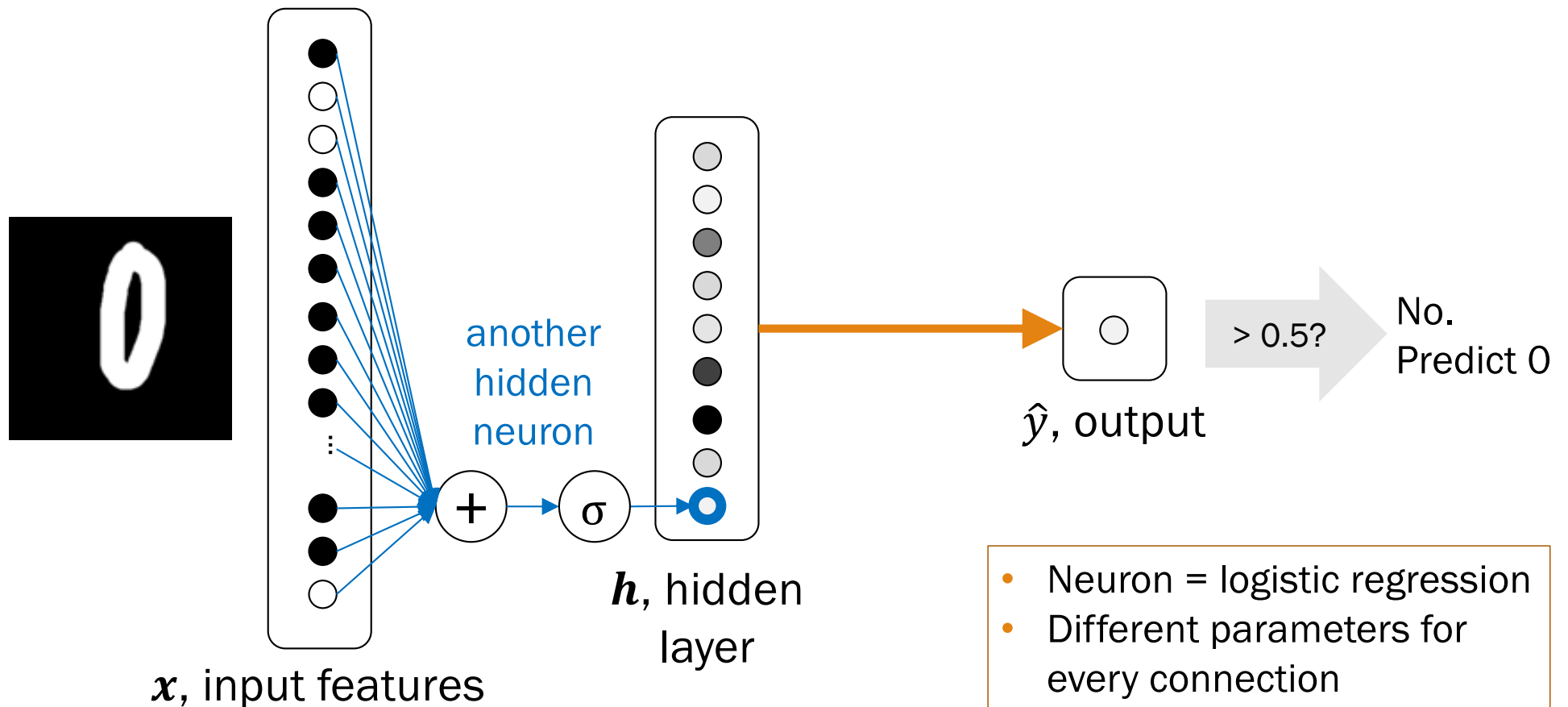
Neural network



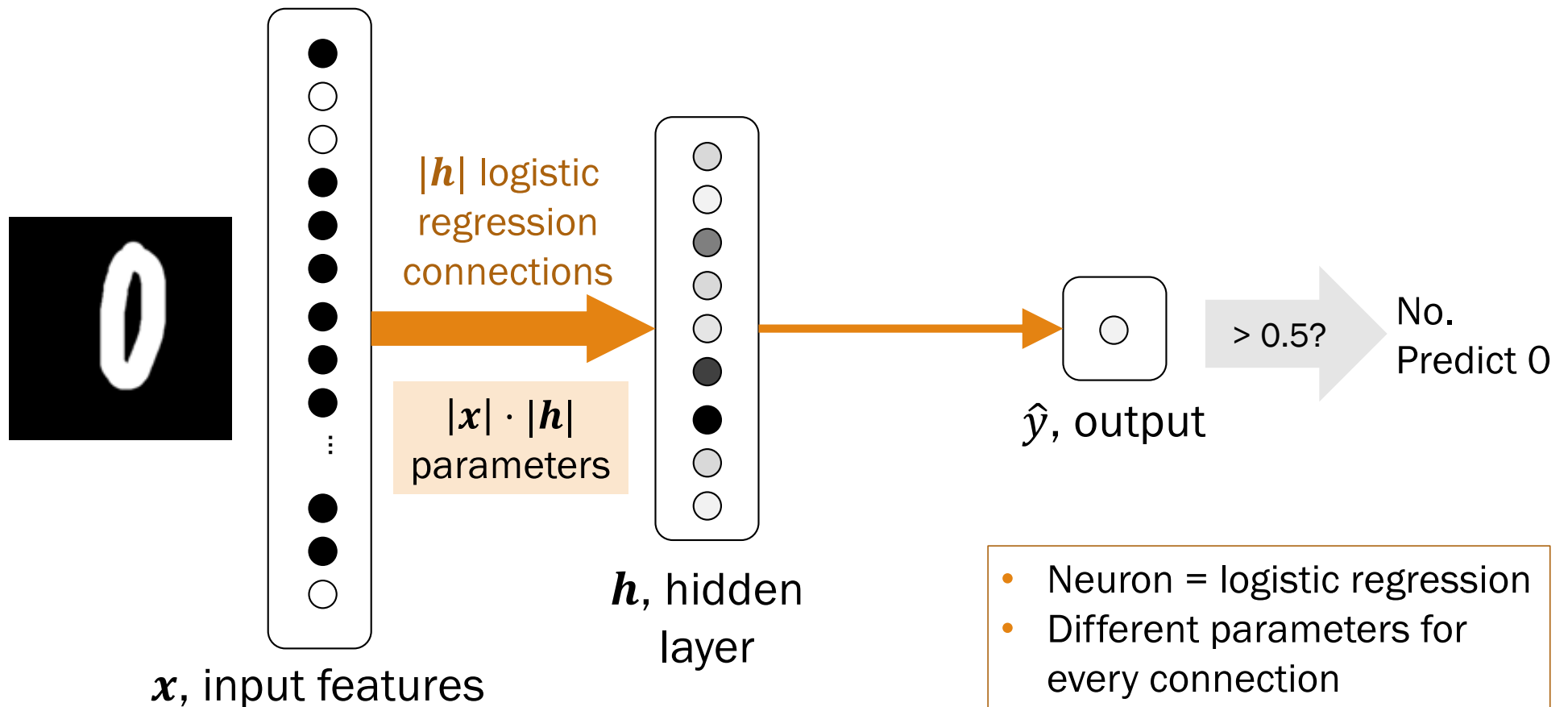
Feed neurons into other neurons



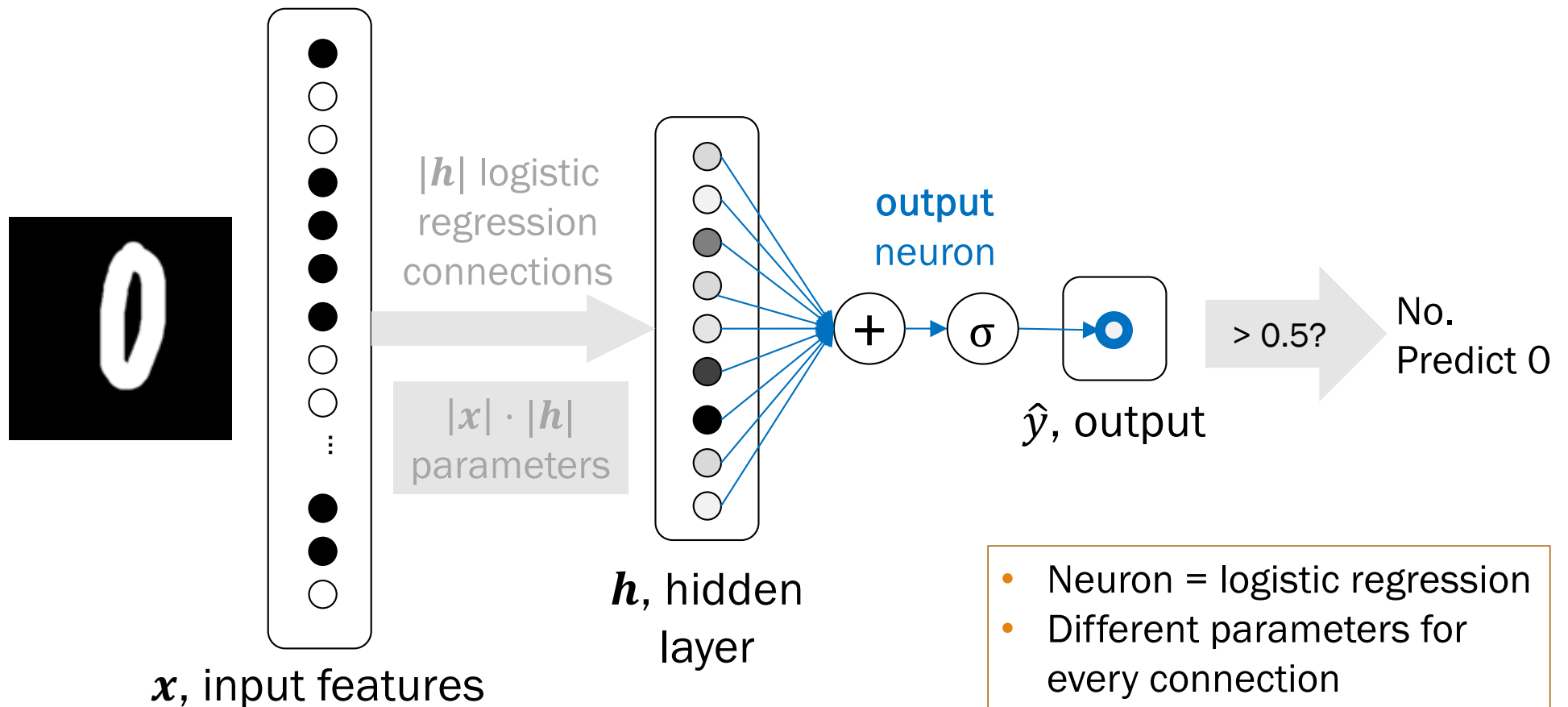
Feed neurons into other neurons



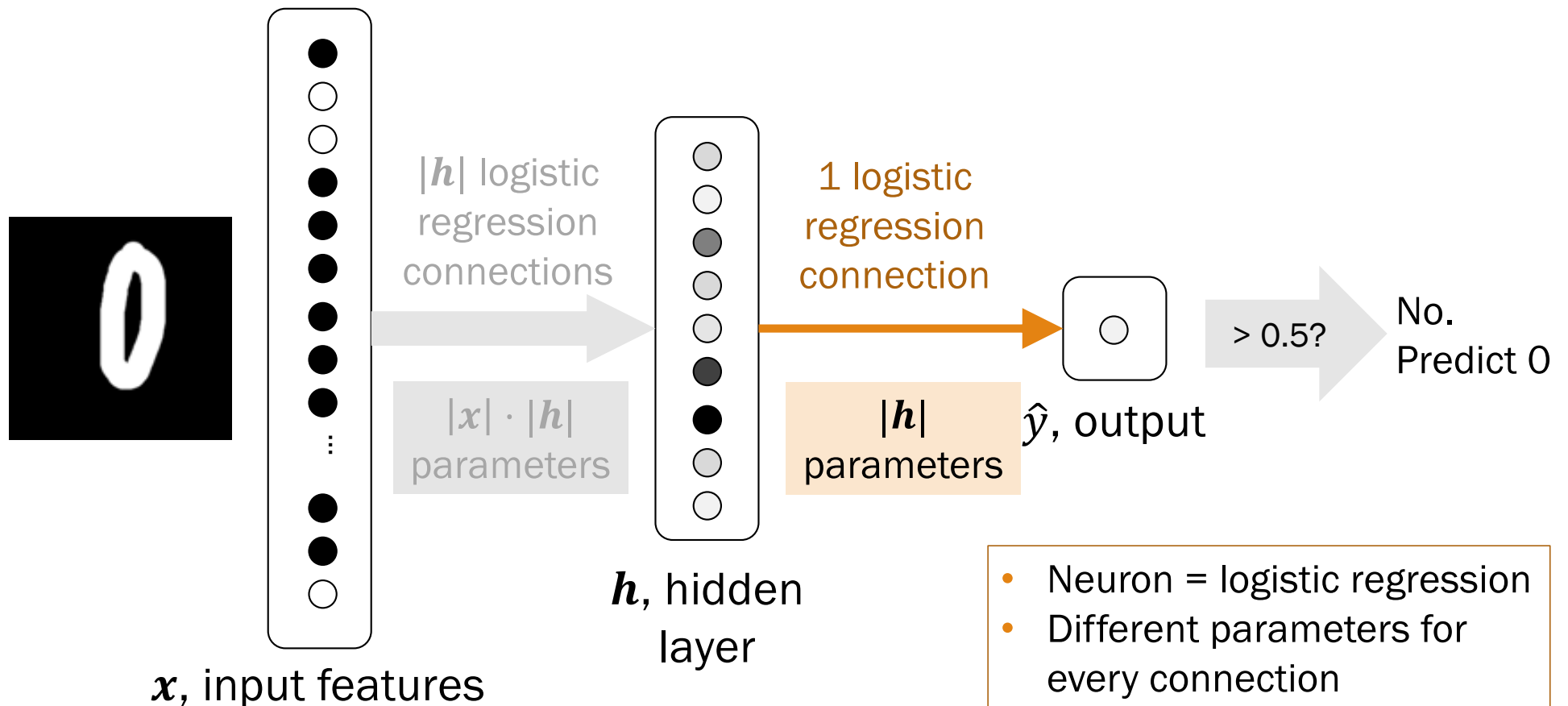
Feed neurons into other neurons



Feed neurons into other neurons



Feed neurons into other neurons



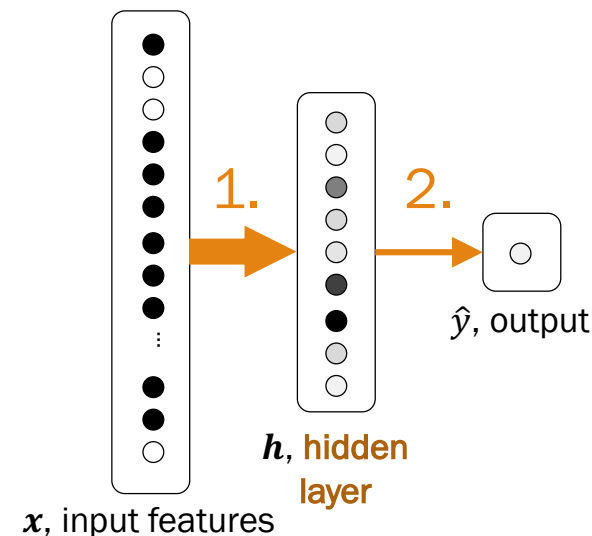
Why doesn't a linear model introduce “complexity”?

Neural network:

1. for $j = 1, \dots, |\mathbf{h}|$:

$$h_j = \sigma\left(\theta_j^{(h)T} \mathbf{x}\right)$$

2. $\hat{y} = \sigma\left(\theta^{(\hat{y})T} \mathbf{h}\right) = P(Y = 1 | \mathbf{X} = \mathbf{x})$



Linear network:

1. for $j = 1, \dots, |\mathbf{h}|$:

$$h_j = \theta_j^{(h)T} \mathbf{x}$$

2. $\hat{y} = \sigma\left(\theta^{(\hat{y})T} \mathbf{h}\right) = P(Y = 1 | \mathbf{X} = \mathbf{x})$



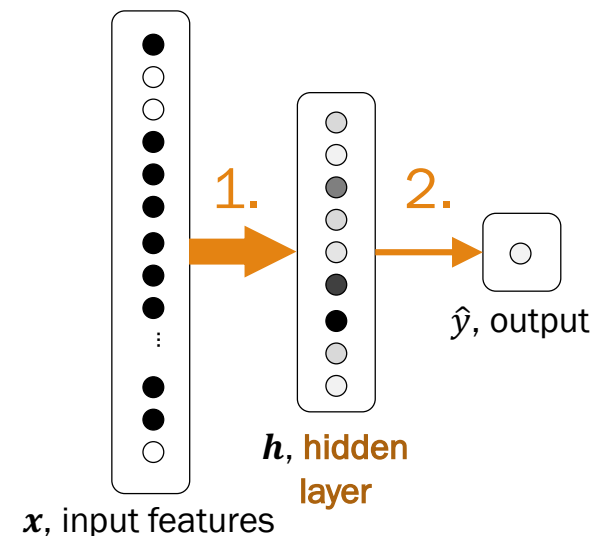
Why doesn't a linear model introduce “complexity”?

Neural network:

1. for $j = 1, \dots, |\mathbf{h}|$:

$$h_j = \sigma(\theta_j^{(h)T} \mathbf{x})$$

2. $\hat{y} = \sigma(\theta^{(\hat{y})T} \mathbf{h}) = P(Y = 1 | \mathbf{X} = \mathbf{x})$



Linear network:

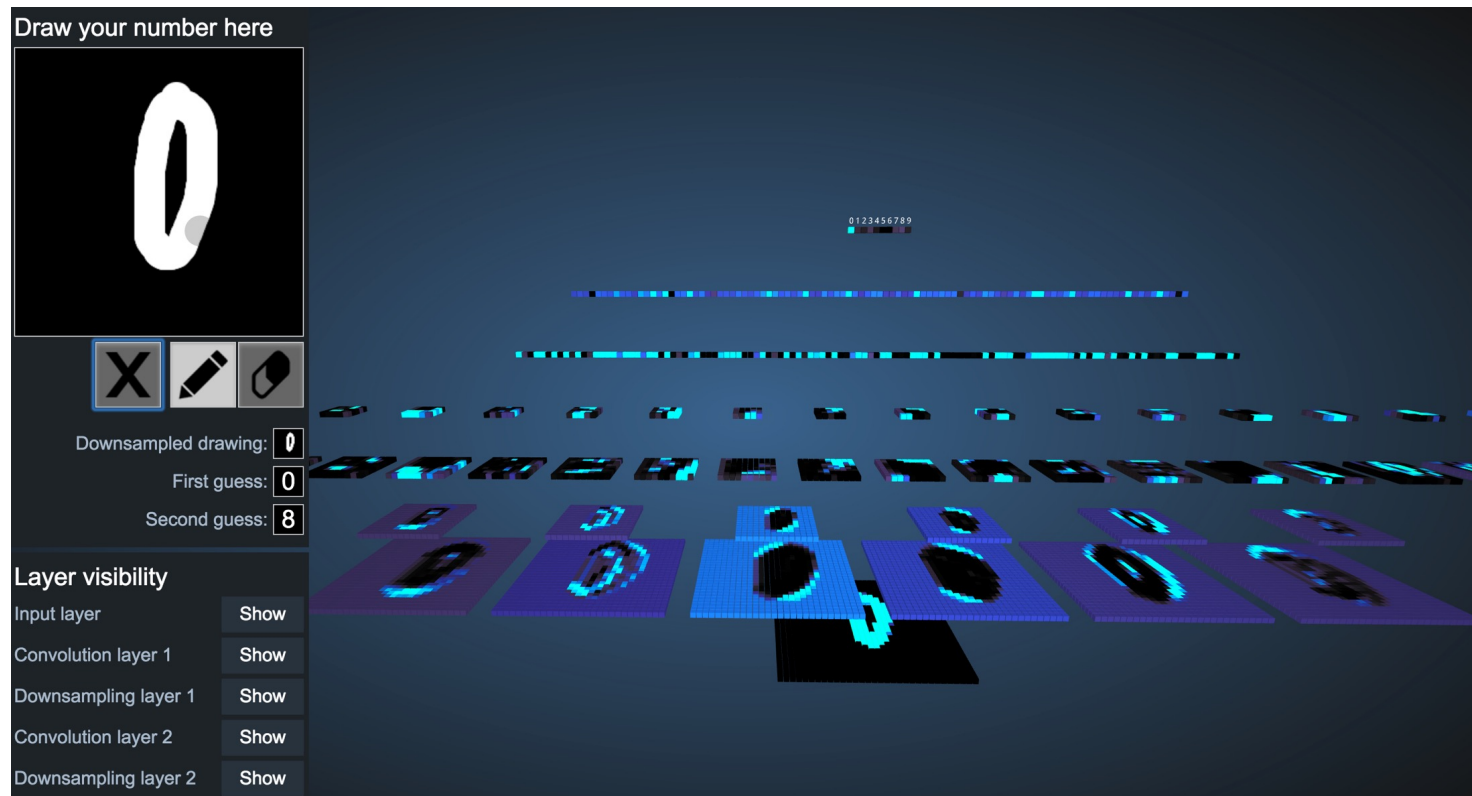
1. for $j = 1, \dots, |\mathbf{h}|$:

$$h_j = \theta_j^{(h)T} \mathbf{x}$$

2. $\hat{y} = \sigma(\theta^{(\hat{y})T} \mathbf{h}) = P(Y = 1 | \mathbf{X} = \mathbf{x})$

The linear model is effectively a single logistic regression with $|\mathbf{x}|$ parameters.

Demonstration



https://adamharley.com/nn_vis/

Neural networks

A neural network (like logistic regression) gets intelligence from its parameters θ .

Training

- Learn parameters θ
- Find θ_{MLE} that maximizes likelihood of training data (MLE)

Testing/ Prediction

For input feature vector $\mathbf{X} = \mathbf{x}$:

- Use parameters to compute $\hat{y} = P(Y = 1 | \mathbf{X} = \mathbf{x})$
- Classify instance as:
$$\begin{cases} 1 & \hat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Neural networks

A neural network (like logistic regression) gets intelligence from its parameters θ .

Training

- Learn parameters θ
- Find θ_{MLE} that maximizes likelihood of training data (MLE)

How do we learn the $|\mathbf{x}| \cdot |\mathbf{h}| + |\mathbf{h}|$ parameters?

Gradient ascent + chain rule!

Training: Logistic Regression

Review

1. Optimization problem:

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$



$$\hat{y} = \sigma(\theta^T \mathbf{x}^{(i)}) = P(Y = 1 | \mathbf{X} = \mathbf{x})$$

2. Compute gradient

Find $|\mathbf{x}|$ parameters

3. Optimize

```
initialize params
repeat many times:
  compute gradient
  params += η * gradient
```

Training: Neural networks

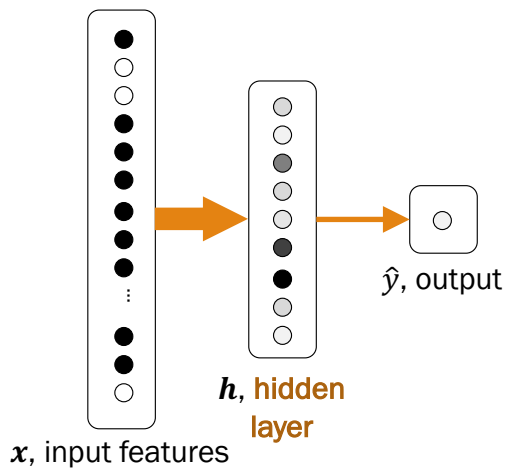
1. Optimization problem:

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

2. Compute gradient

3. Optimize

1. Same output \hat{y} , same log conditional likelihood



for $j = 1, \dots, |\mathbf{h}|$:

$$h_j = \sigma(\theta_j^{(h)^T} \mathbf{x})$$

$$\hat{y} = \sigma(\theta^{(\hat{y})^T} \mathbf{h}) = P(Y = 1 | \mathbf{X} = \mathbf{x})$$

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

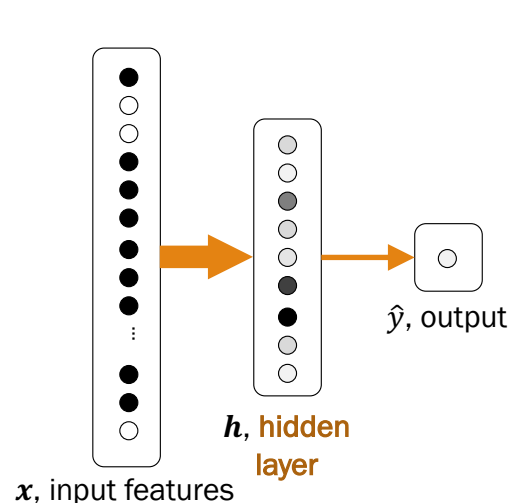
$$L(\theta) = \prod_{i=1}^n P(Y = y^{(i)} | \mathbf{X} = \mathbf{x}^{(i)}, \theta)$$

Binary class labels:
 $Y \in \{0, 1\}$

$$= \prod_{i=1}^n (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}}$$

$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

(model is a little more complicated)



$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | x^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$
$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

for $j = 1, \dots, |\mathbf{h}|$:

$$h_j = \sigma \left(\theta_j^{(h)} x \right) \quad \text{dimension } |x|$$

$$\hat{y} = \sigma \left(\theta^{(\hat{y})} h \right) = P(Y = 1 | X = x) \quad \text{dimension } |h|$$

To optimize for
log conditional likelihood,
we now need to find:

$$|\mathbf{h}| \cdot |\mathbf{x}| + |\mathbf{h}| \quad \text{parameters}$$

2. Compute gradient

1. Optimization problem:

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

$$h_j = \sigma \left(\boxed{\theta_j^{(h)}}^T \mathbf{x} \right) \quad \text{for } j = 1, \dots, |\mathbf{h}| \quad \hat{y} = \sigma \left(\boxed{\theta^{(\hat{y})}}^T \mathbf{h} \right)$$

2. Compute gradient

Take gradient with respect to all θ parameters

3. Optimize

Calculus refresher #1:

Derivative(sum) =
sum(derivative)

Calculus refresher #2:

Chain rule 🌟🌟🌟

3. Optimize

1. Optimization problem:

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

$$h_j = \sigma \left(\theta_j^{(h)}{}^T \mathbf{x} \right) \quad \text{for } j = 1, \dots, |\mathbf{h}| \quad \hat{y} = \sigma \left(\theta^{(\hat{y})}{}^T \mathbf{h} \right)$$

2. Compute gradient

Take gradient with respect to all θ parameters

3. Optimize

```
initialize params
repeat many times:
  compute gradient
  params += η * gradient
```

Training a neural net

1. Optimization problem:

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Wait, did we just skip something difficult?

2. Compute

3. Optimize

```
initialize params
repeat many times:
  compute gradient
  params += η * gradient
```

2. Compute gradient via backpropagation

1. Optimization problem:

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^n f(y^{(i)} | \mathbf{x}^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)$$

$$LL(\theta) = \sum_{i=1}^n y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

$$h_j = \sigma \left(\theta_j^{(h)} \right)^T \mathbf{x} \quad \text{for } j = 1, \dots, |\mathbf{h}| \quad \hat{y} = \sigma \left(\theta^{(\hat{y})} \right)^T \mathbf{h}$$

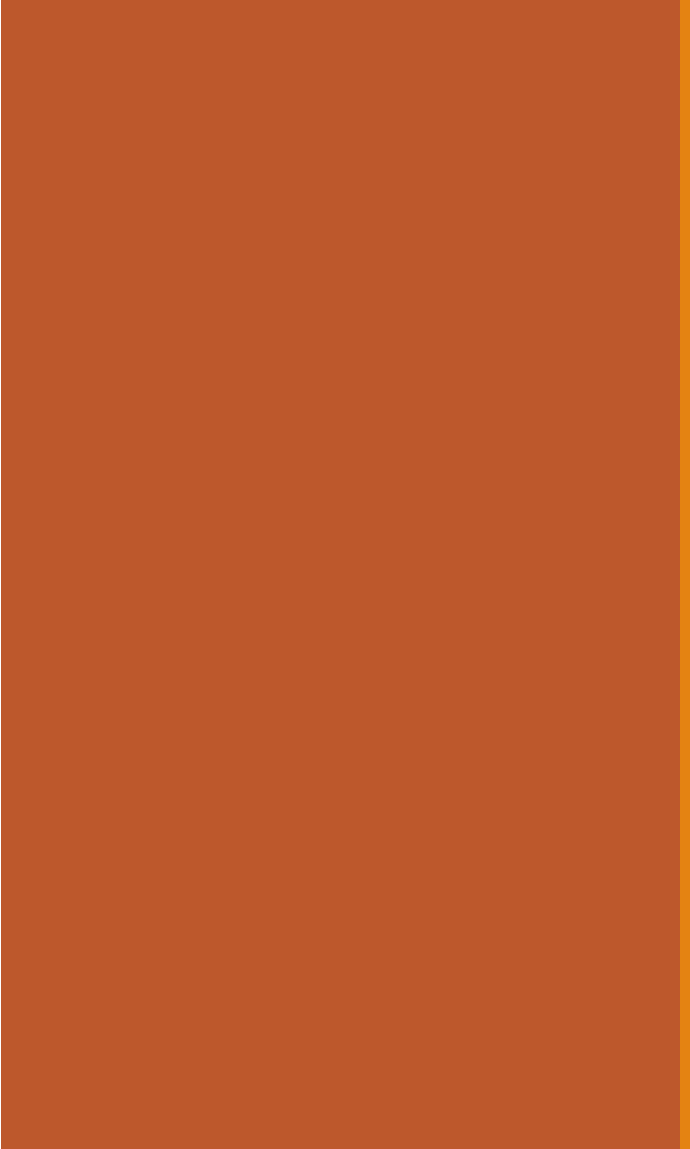
2. Compute gradient

Take gradient with respect to all θ parameters

3. Optimize

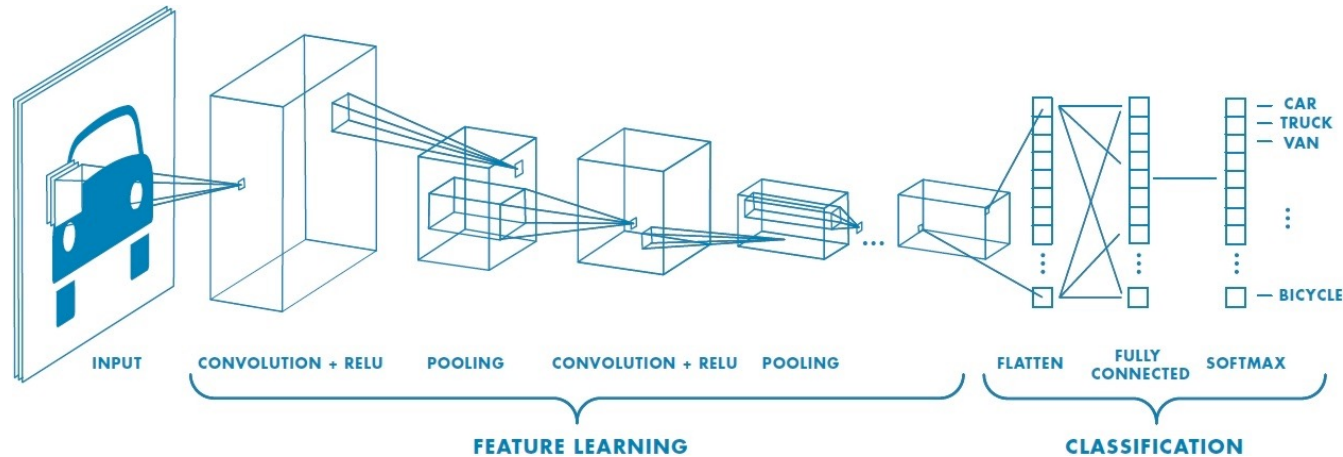
initial
repeat
compute
parameters

Learn the tricks behind
backpropagation in
CS229, CS231N, CS224N,
etc.



Beyond the basics

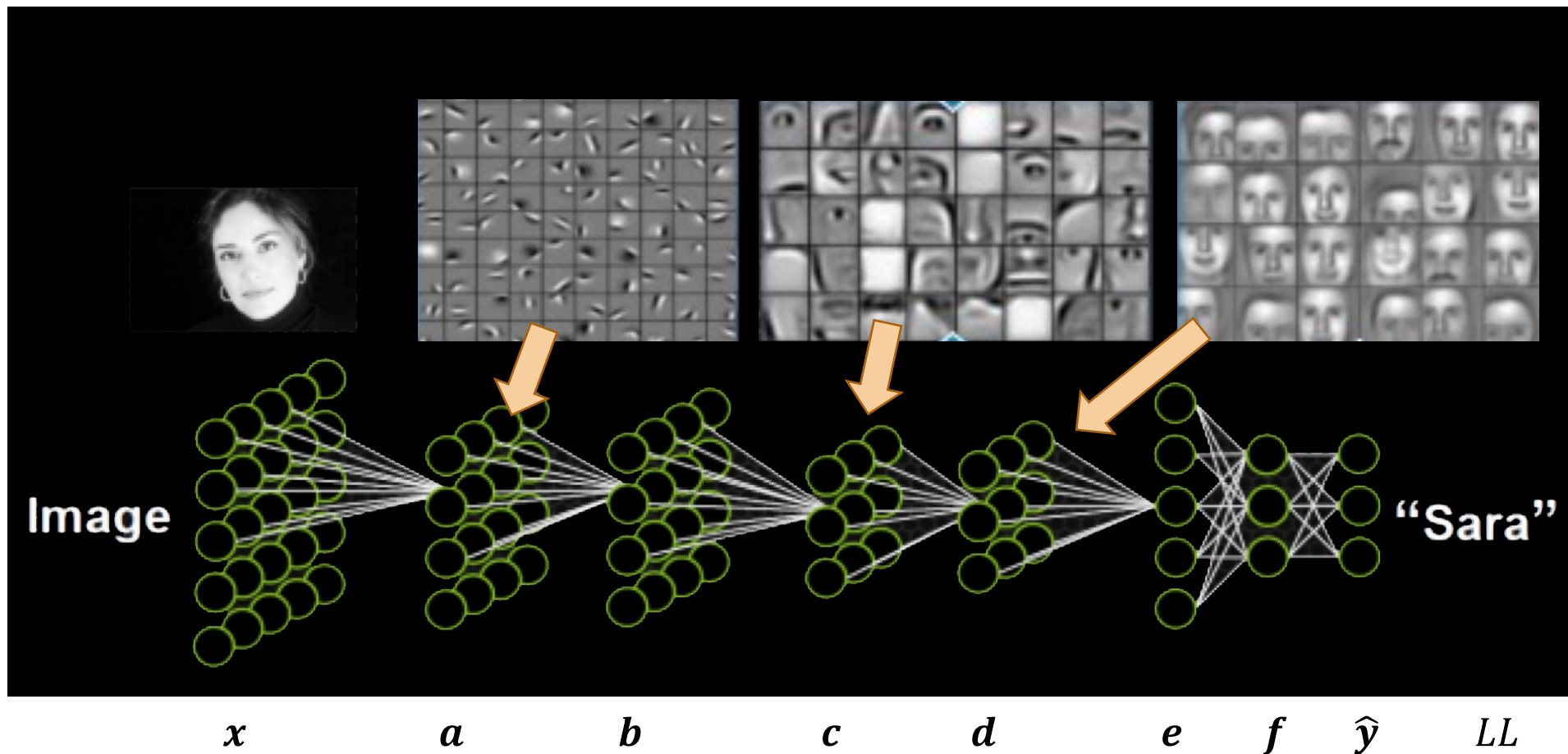
Shared weights?



It turns out if you want to force some of your weights to be shared over different neurons, the math isn't much harder.

Convolution is an example of such weight-sharing and is used a lot for vision (Convolutional Neural Networks, CNN).

Neural networks with multiple layers



Lisa Yan, Chris Piech, Mehran Sahami, and Jerry Cain, CS109, Winter 2024

Stanford University

Neurons learn features of the dataset



Neurons in later layers will respond strongly to high-level features of your **training data**.

If your training data is faces, you will get lots of face neurons.

If your training data is all of YouTube...



...you get a cat neuron.



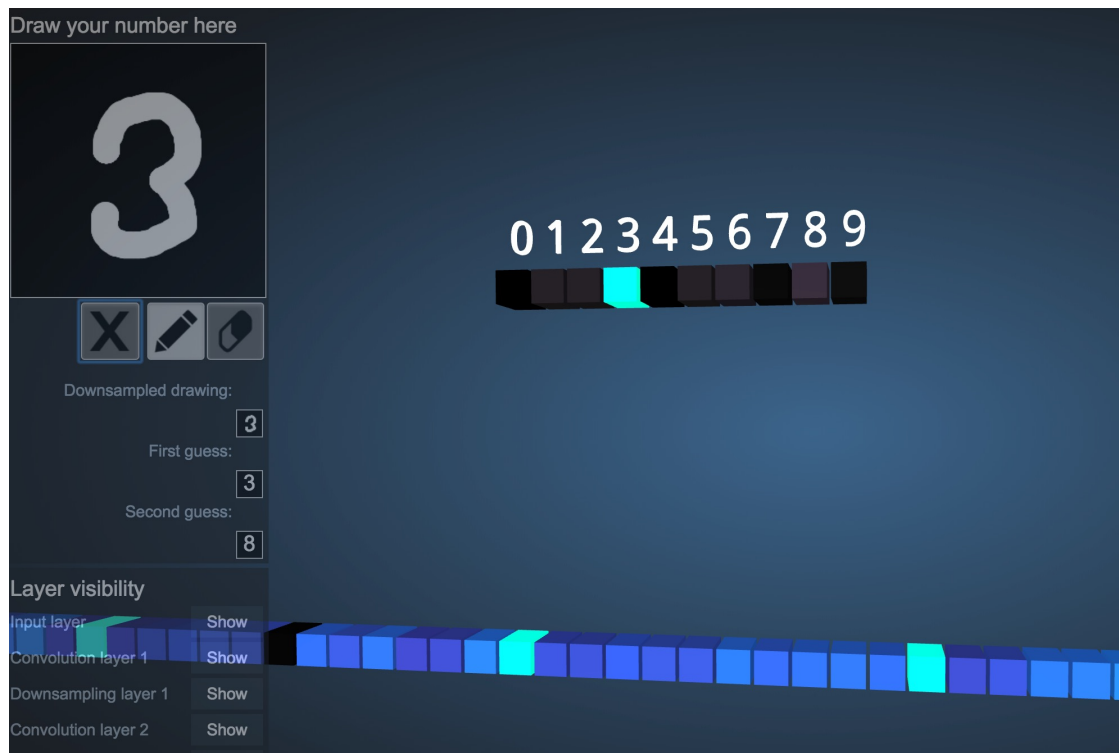
Top stimuli in test set



Optimal stimulus found by numerical optimization



Multiple outputs?



Softmax is a generalization of the sigmoid function.

$\text{sigmoid}(z)$: value in range $[0, 1]$

$z \in \mathbb{R}$:

$$P(Y = 1 | \mathbf{X} = \mathbf{x}) = \sigma(z)$$

(equivalent: Bernoulli p)



$\text{softmax}(z)$: k -dimensional values in range $[0, 1]$ that add up to 1

$\mathbf{z} \in \mathbb{R}^k$:

$$P(Y = i | \mathbf{X} = \mathbf{x}) = \text{softmax}(\mathbf{z})_i$$

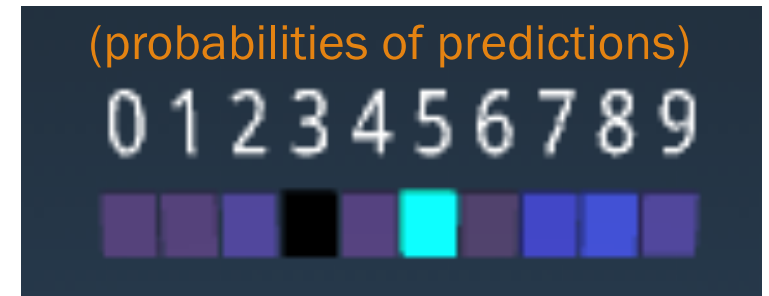
(equivalent: Multinomial p_1, \dots, p_k)

Softmax test metric: Top-5 error

$Y = y$	$P(Y = y X = x)$
5	0.14
8	0.13
7	0.12
2	0.10
9	0.10
4	0.09
1	0.09
0	0.09
6	0.08
3	0.05

Top-5 classification error

What % of datapoints did *not* have the correct class label in the top-5 predictions?



ImageNet classification

22,000 categories

14,000,000 images

Hand-engineered features
(SIFT, HOG, LBP),
Spatial pyramid,
SparseCoding/Compression

...
smoothhound, smoothhound shark, *Mustelus mustelus*
American smooth dogfish, *Mustelus canis*
Florida smoothhound, *Mustelus norrisi*
whitetip shark, reef whitetip shark, *Triaenodon obesus*
Atlantic spiny dogfish, *Squalus acanthias*
Pacific spiny dogfish, *Squalus suckleyi*
hammerhead, hammerhead shark
smooth hammerhead, *Sphyrna zygaena*
smalleye hammerhead, *Sphyrna tudes*
shovelhead, bonnethead, bonnet shark, *Sphyrna tiburo*
angel shark, angelfish, *Squatina squatina*, monkfish
electric ray, crampfish, numbfish, torpedo
smalltooth sawfish, *Pristis pectinatus*
guitarfish
roughtail stingray, *Dasyatis centroura*
butterfly ray
eagle ray
spotted eagle ray, spotted ray, *Aetobatus narinari*
cownose ray, cow-nosed ray, *Rhinoptera bonasus*
manta, manta ray, devilfish
Atlantic manta, *Manta birostris*
devil ray, *Mobula hypostoma*
grey skate, gray skate, *Raja batis*
little skate, *Raja erinacea*
...



Stingray



Mantaray

ImageNet classification challenge

~~22,000 categories~~

1000 categories

smoothhound shark, *Mustelus mustelus*
blacktip shark, *Carcharhinus limbatus*
Florida smoothhound, *Mustelus norrisi*

14,000,000 images

1,200,000 images in train set

hobbiton obseus

200,000 images in test

Hand-engineered features
(SIFT, HOG, LBP),
Spatial pyramid,
SparseCoding/Compression

test

smooth hammerhead, *Sphyrna zygaena*
smalleye hammerhead, *Sphyrna tudes*
shovelhead, bonnethead, bonnet shark, *Sphyrna tiburo*
angel shark, angelfish, *Squatina squatina*, monkfish
electric ray, crampfish, numbfish, torpedo
smalltooth sawfish, *Pristis pectinatus*
guitarfish
rougthead stingray, *Dasyatis centroura*
butterfly ray
eagle ray
spotted eagle ray, spotted ray, *Aetobatus narinari*
cownose ray, cow-nosed ray, *Rhinoptera bonasus*
manta, manta ray, devilfish
Atlantic manta, *Manta birostris*
devil ray, *Mobula hypostoma*
grey skate, gray skate, *Raja batis*
little skate, *Raja erinacea*
...

ImageNet challenge: Top-5 classification error

(lower is better)

99.5%

Random guess

$$P(\text{true class label not in 5 guesses}) = \frac{\binom{999}{5}}{\binom{1000}{5}} = \frac{995}{1000}$$

ImageNet challenge: Top-5 classification error

(lower is better)

99.5%

Random guess

25.8%

Pre-Neural Networks

5.1%

Humans
(2014)

16.4%

GoogLeNet
(2015)

Russakovsky et al., ImageNet Large Scale Visual Recognition Challenge. IJCV 2015

Szegedy et al., Going Deeper With Convolutions. CVPR 2015

Hu et al., Squeeze-and-Excitation Networks. Preprint arXiv 2017

Lisa Yan, Chris Piech, Mehran Sahami, and Jerry Cain, CS109, Winter 2024

ImageNet challenge: Top-5 classification error

(lower is better)

99.5%

Random guess

25.8%

Pre-Neural Networks

5.1%

Humans
(2014)

16.4%

GoogLe Net
(2015)

2.25%

SENet
(2017)

Russakovsky et al., ImageNet Large Scale Visual Recognition Challenge. IJCV 2015

Szegedy et al., Going Deeper With Convolutions. CVPR 2015

Hu et al., Squeeze-and-Excitation Networks. Preprint arXiv 2017

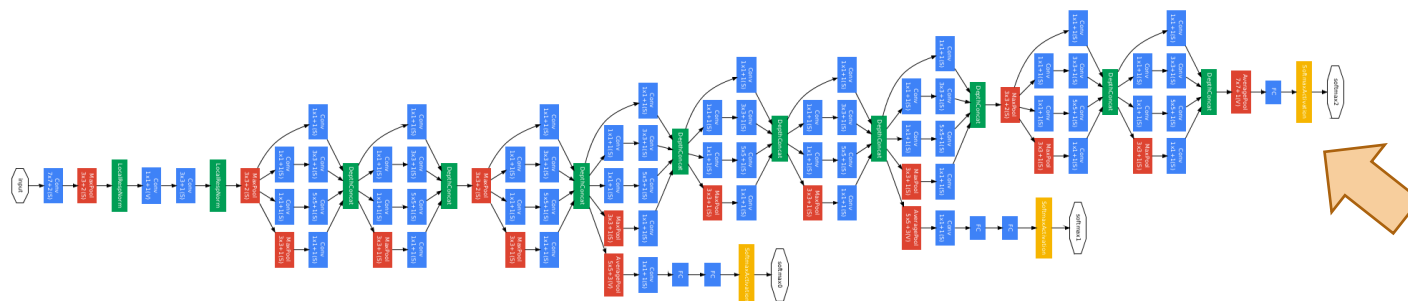
Lisa Yan, Chris Piech, Mehran Sahami, and Jerry Cain, CS109, Winter 2024

GoogLeNet (2015)



Google Brain

1 Trillion Artificial Neurons
(btw human brains have 1 billion neurons)



Multiple,
Multi class output

22 layers deep!

Speeding up gradient descent

minimizes loss (a function of prediction error)

```
initialize  $\theta_j = 0$  for  $0 \leq j \leq m$   
repeat many times:
```

```
  gradient[j] = 0 for  $0 \leq j \leq m$ 
```

```
  for each training example  $(x, y)$ :
```

```
    for each  $0 \leq j \leq m$ :
```

```
      compute gradient
```

```
   $\theta_j -= \eta * \text{gradient}[j]$  for all  $0 \leq j \leq m$ 
```

1. What if we have 1,200,000 images in our training set?

2. How can we speed up the update?

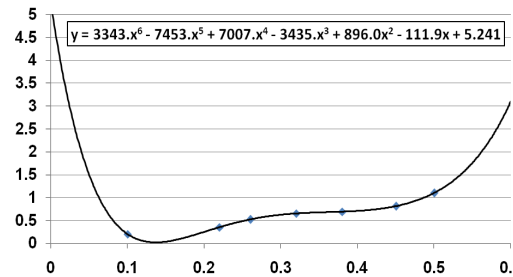
Our **batch gradient descent** (over the entire training set) will be slow + expensive.

1. Use **stochastic gradient descent** (randomly select training examples with replacement).
2. **Momentum update** (Incorporate “acceleration” or “deceleration” of gradient updates so far)

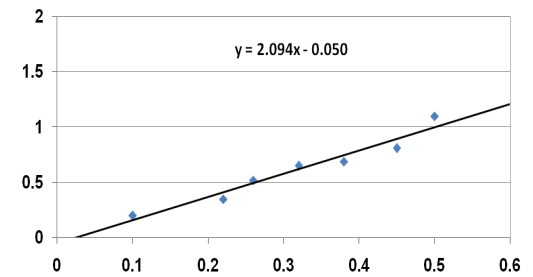
Good ML = Generalization

Overfitting

Fitting the training data too well, such that we lose generality of model for predicting new data



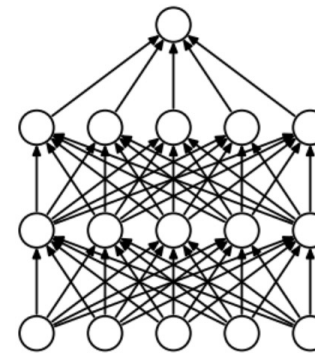
perfect fit, but bad predictor for new data



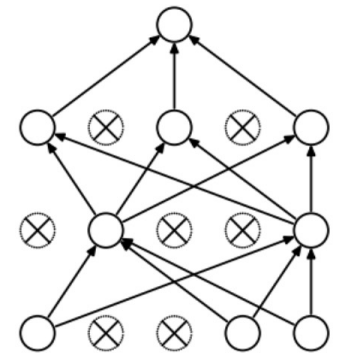
more general fit + better predictor for new data

Dropout

During training, randomly leave out some neurons each training step.
It will make your network more robust.



(a) Standard Neural Net



(b) After applying dropout.

Making decisions?



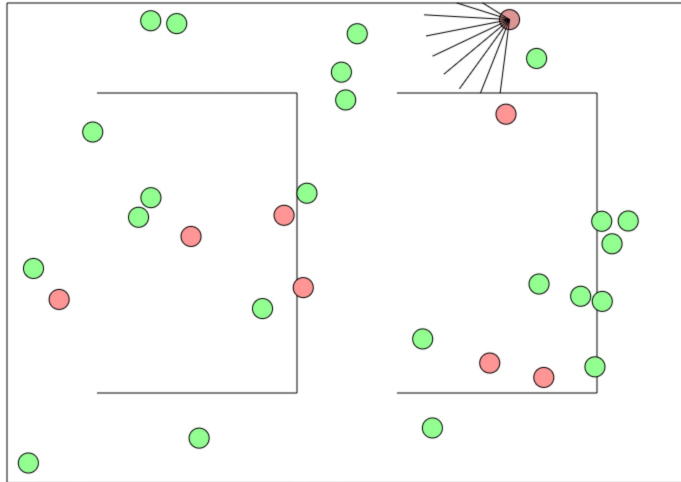
Not everything is classification.

Deep Reinforcement Learning

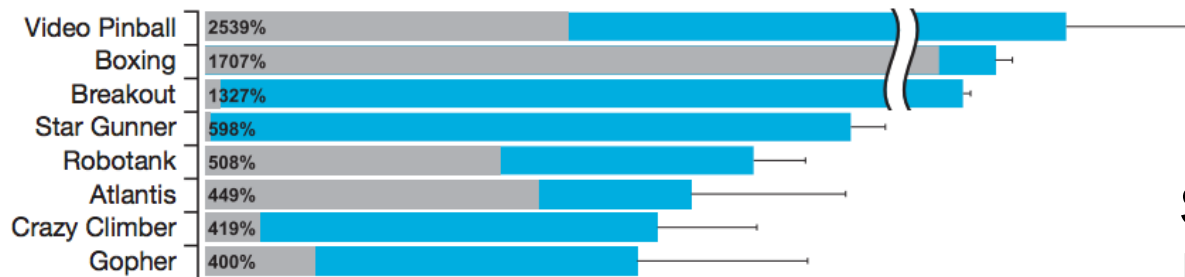
Instead of having the output of a model be a probability, you make output an expectation.

<http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html>

Deep Reinforcement Learning



<http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html>



Deep Mind Atari Games
Score compared to best
human