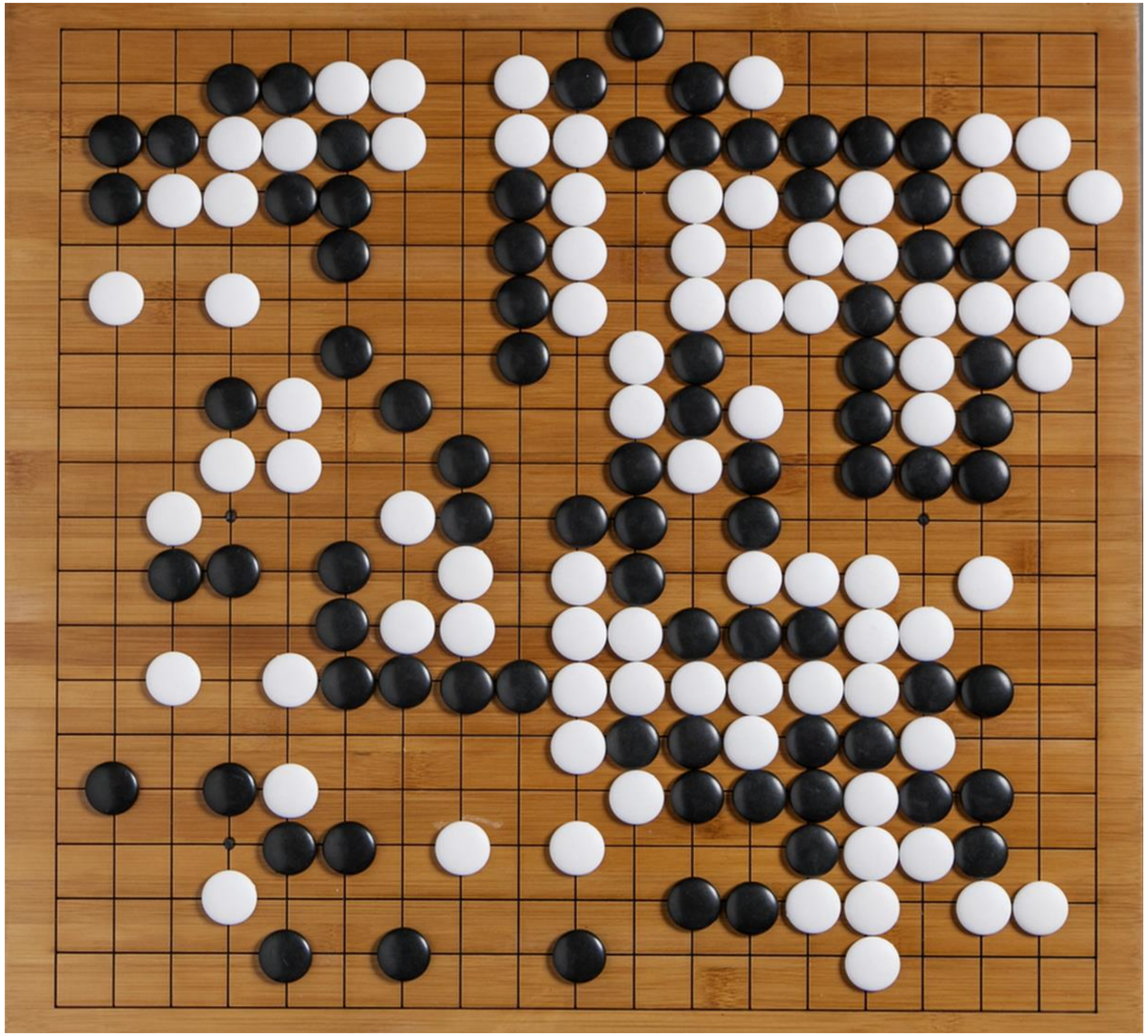
Deep Learning

Chris Piech
CS109, Stanford University

Innovations in deep learning



AlphaGO (2016)

Deep learning (neural networks) is the core idea driving the current revolution in AI.

Notes:

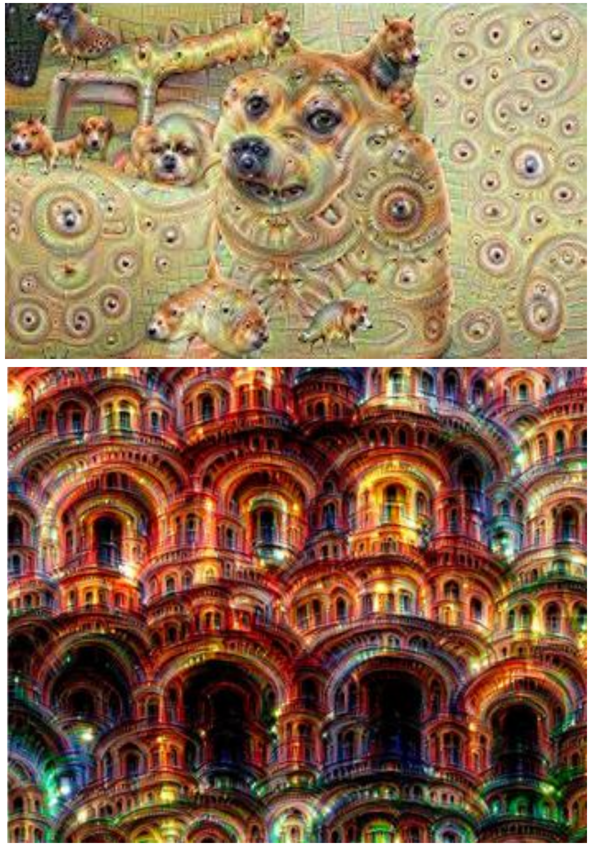
- Checkers is the last **solved** game (from game theory, where perfect player outcomes can be fully predicted from any gameboard).
  https://en.wikipedia.org/wiki/Solved_game
- The first machine learning algorithm defeated a world champion in Chess in 1996.
  https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

Self Driving Cars

Computers making art

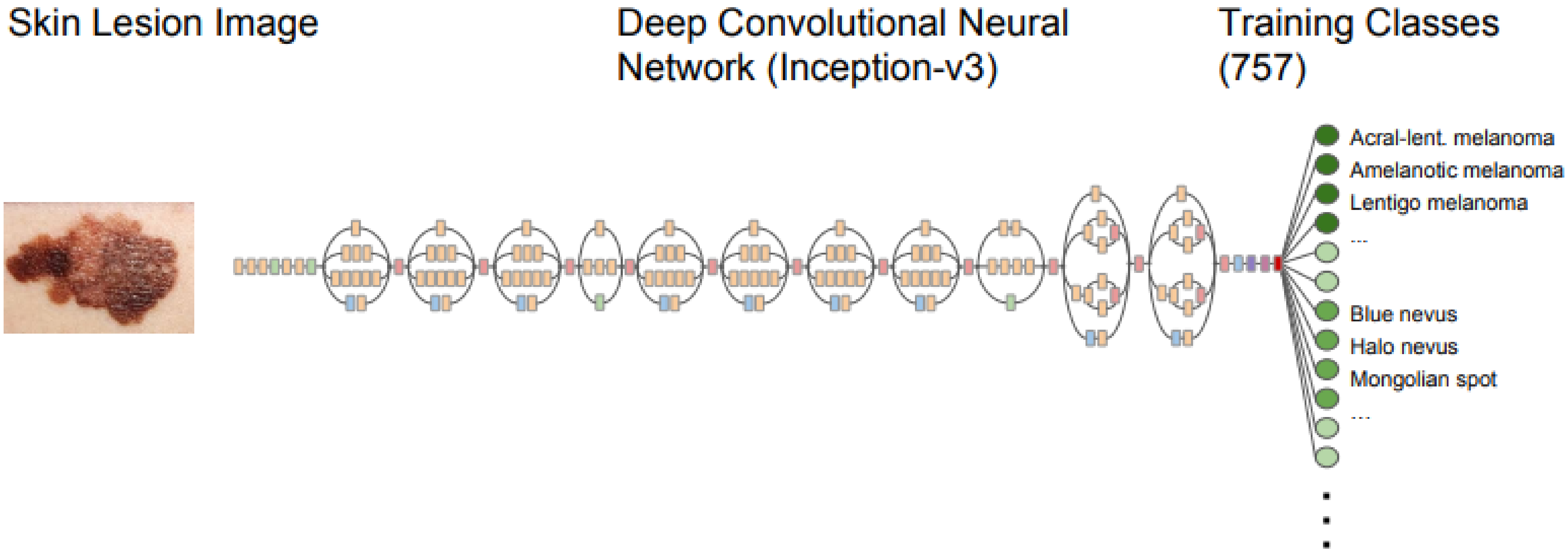
The Next Rembrandt
https://medium.com/@DutchDigital/the-next-rembrandt-bringing-the-old-master-back-to-life-35dfb1653597

A Neural Algorithm of Artistic Style
https://arxiv.org/abs/1508.06576
https://github.com/jcjohnson/neural-style



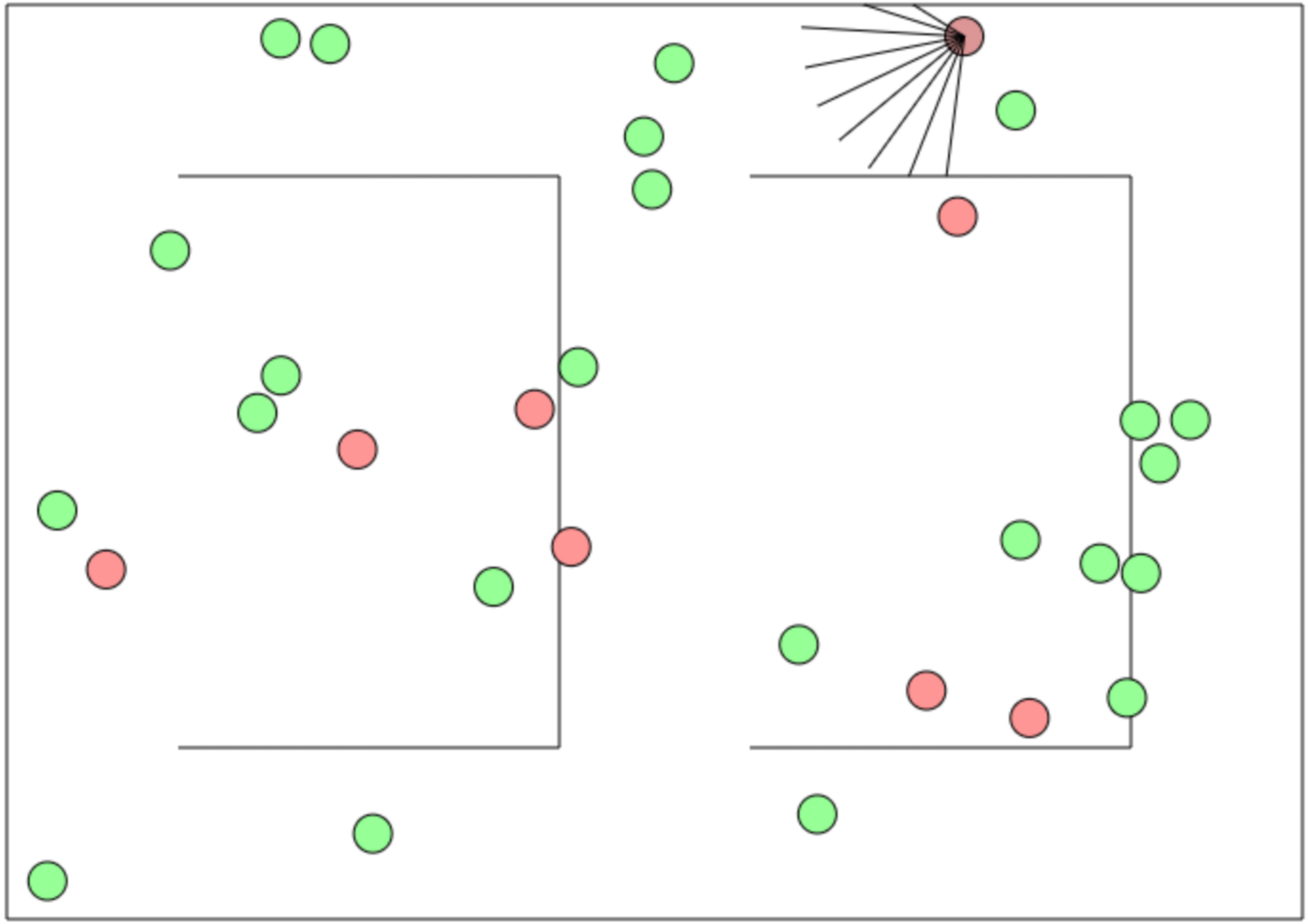
Google Deep Dream
https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Detecting skin cancer



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

Our Little Buddy from Last Class



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

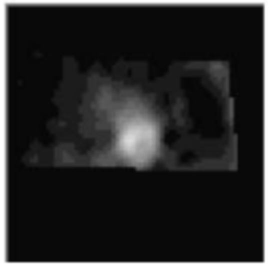
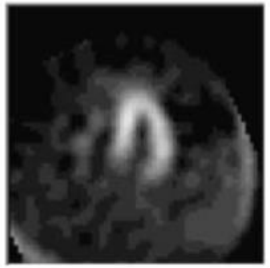
Logistics

After Break Schedule

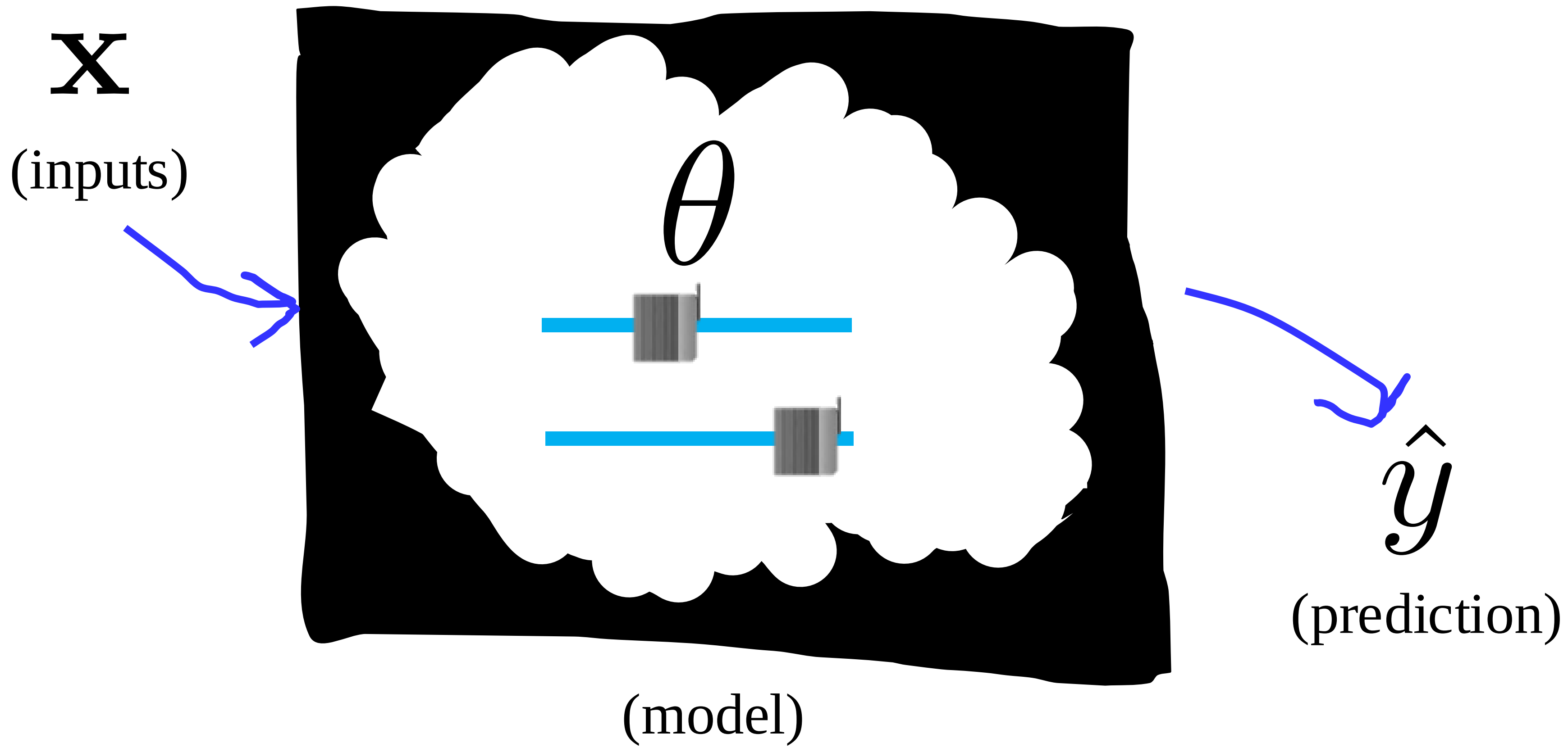
<table>
<tr><th></th><th>Monday</th><th>Tuesday</th><th>Wednesday</th><th>Thursday</th><th>Friday</th></tr>
<tr><td rowspan="2">W10</td><td colspan="3">PEP</td><td rowspan="2"></td><td rowspan="2">PSet 6 is Due</td></tr>
<tr><td>Deep Learning</td><td></td><td>Diffusion</td></tr>
<tr><td>W11</td><td></td><td>Final Exam</td><td></td><td></td><td></td></tr>
</table>

Review

Classification Task

	ROI 1	ROI 2		ROI *m*	Output
			…		
Heart 1	0	1		1	0
Heart 2	1	1		1	0
			⋮		⋮
Heart *n*	0	0		0	1

Machine Learning

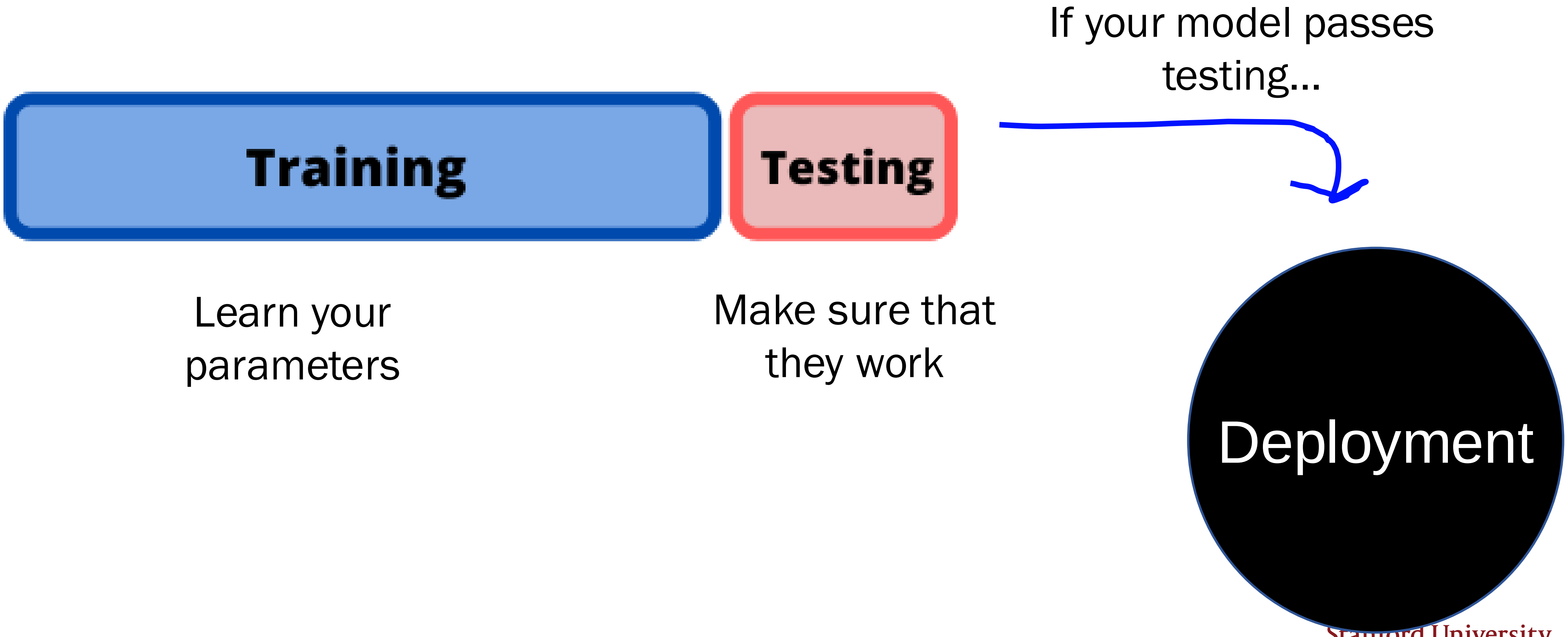


The Training / Testing Paradigm

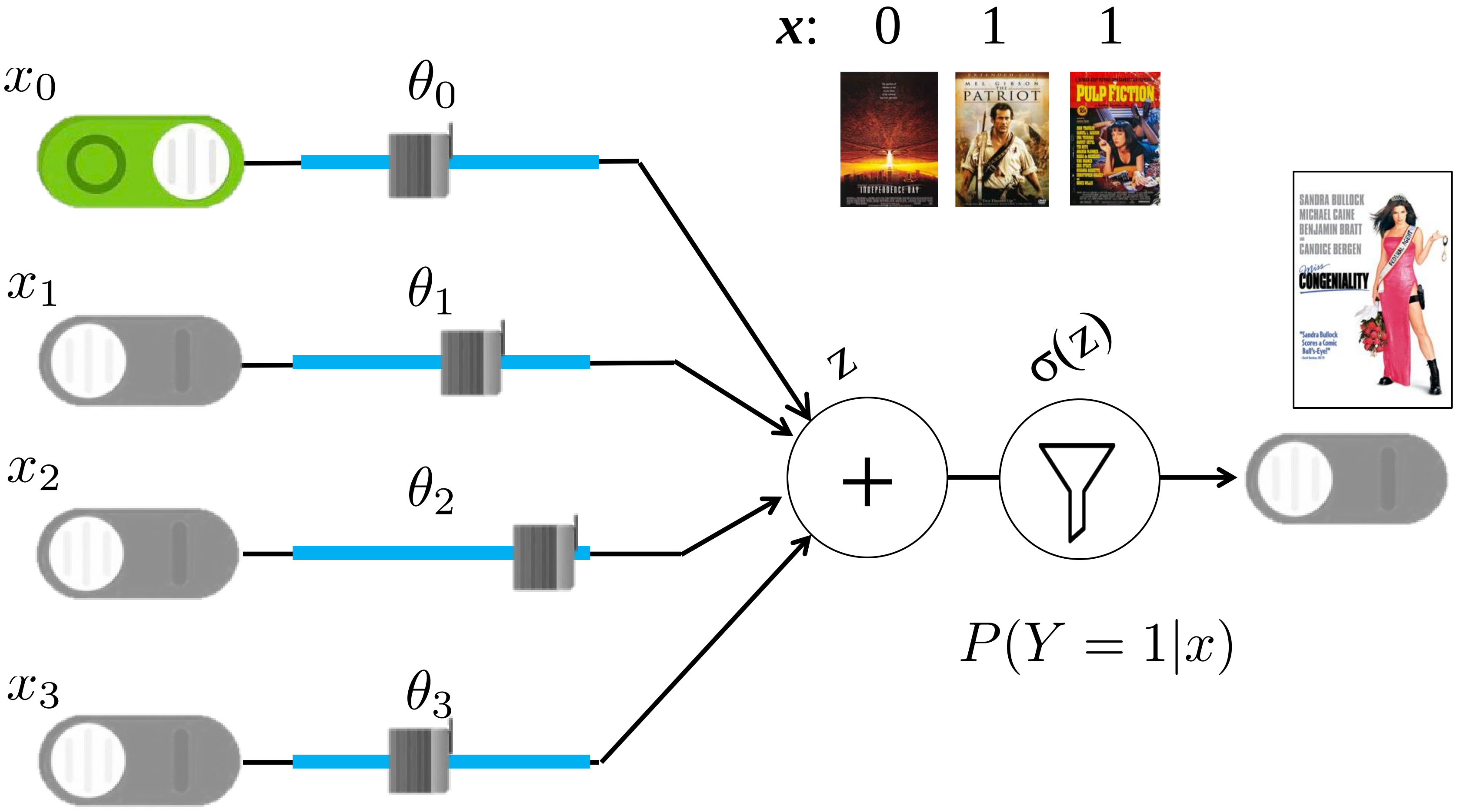
Dataset

Deployment

The Training / Testing Paradigm

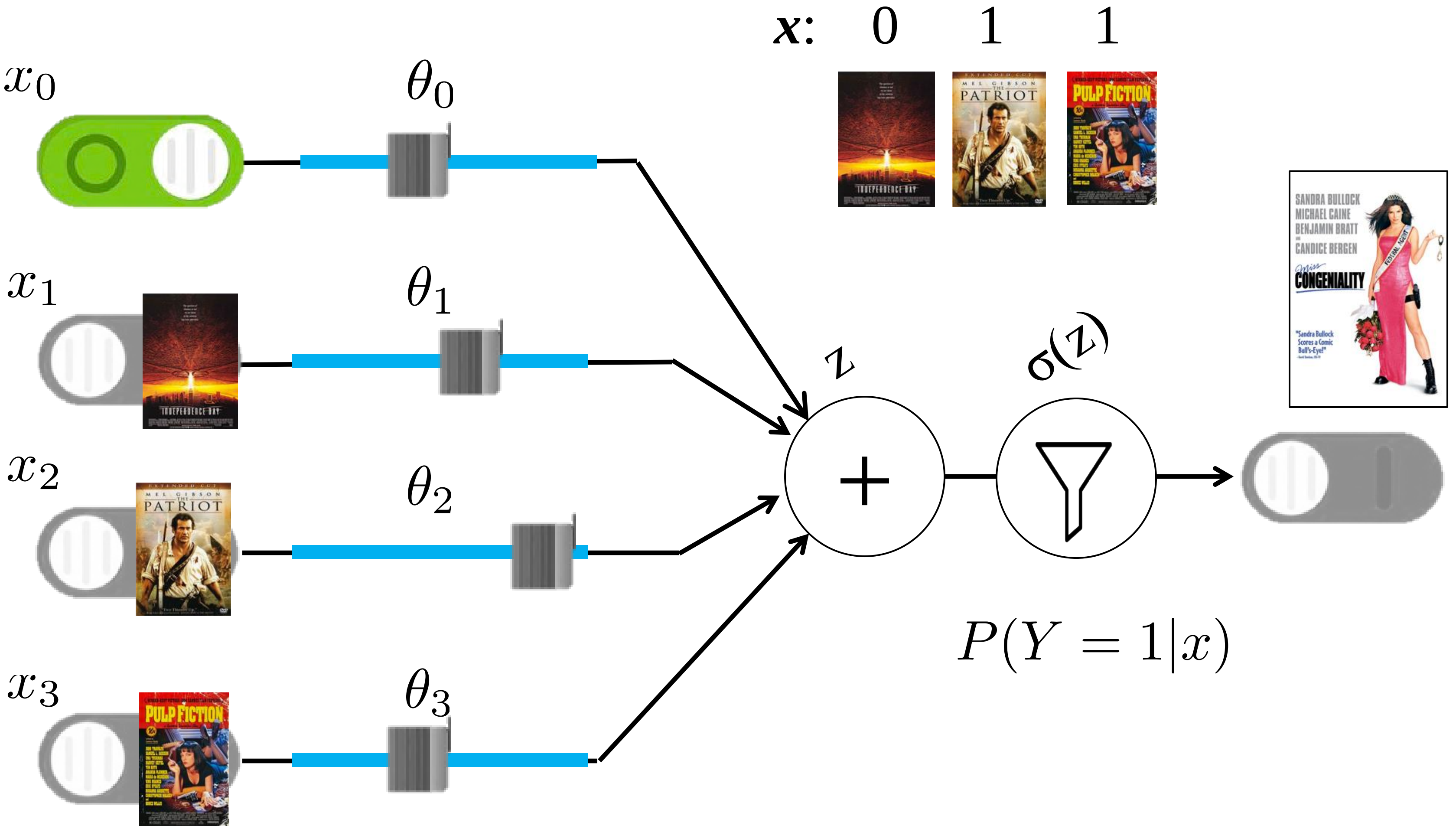


Logistic Regression



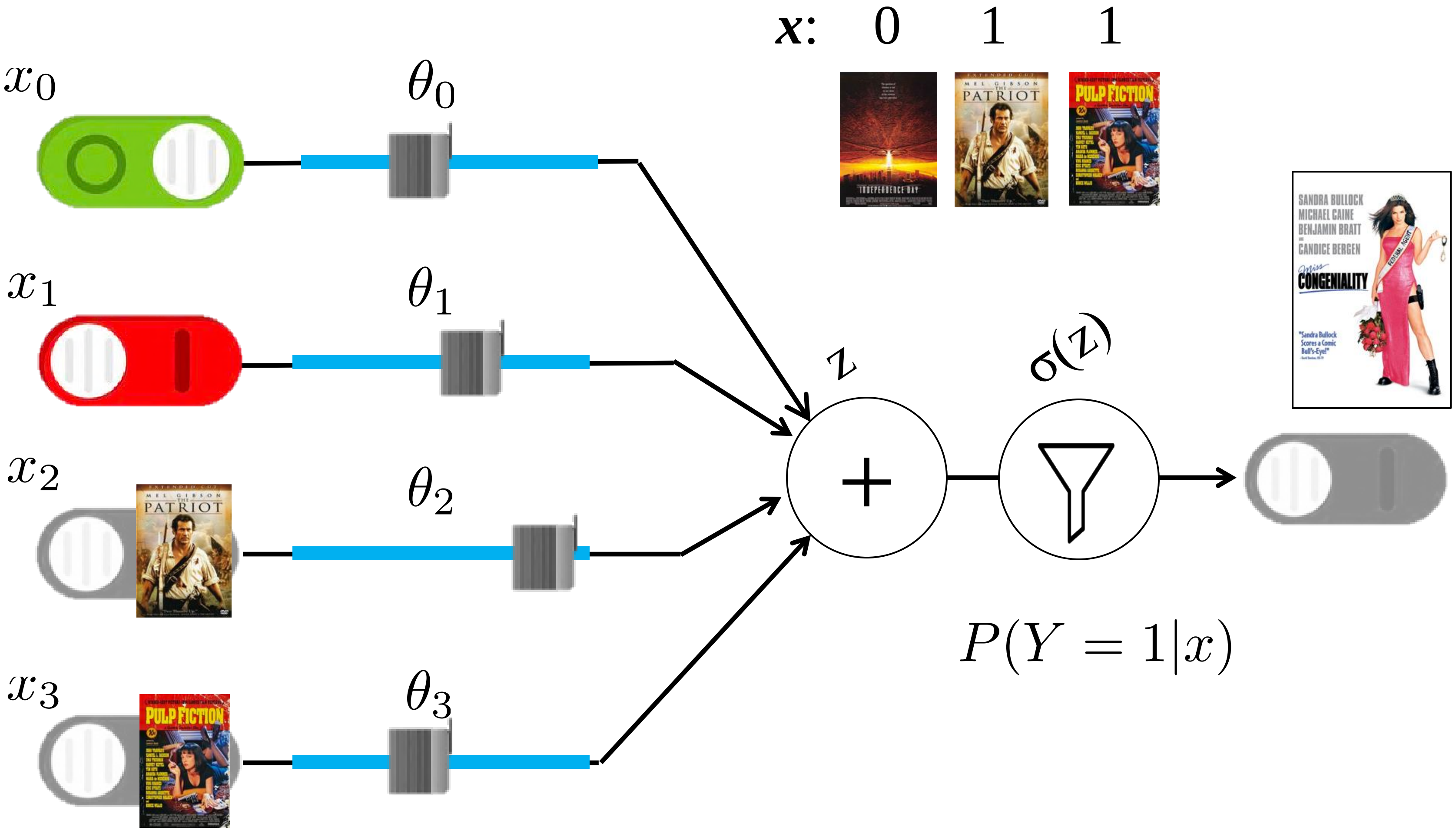
$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Logistic Regression



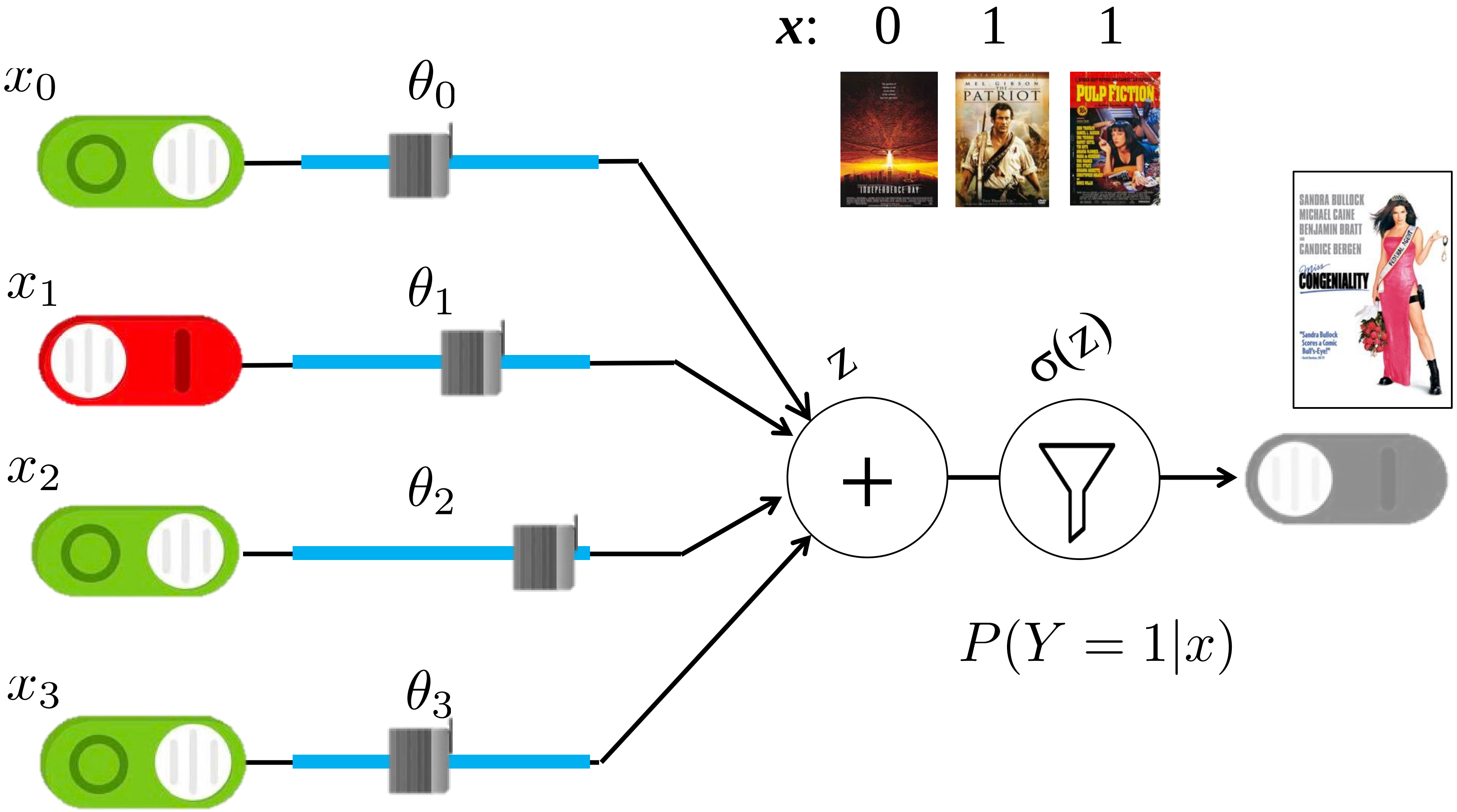
$$P(Y=1|X=\mathbf{x}) = \sigma(\theta^T\mathbf{x})$$

Logistic Regression



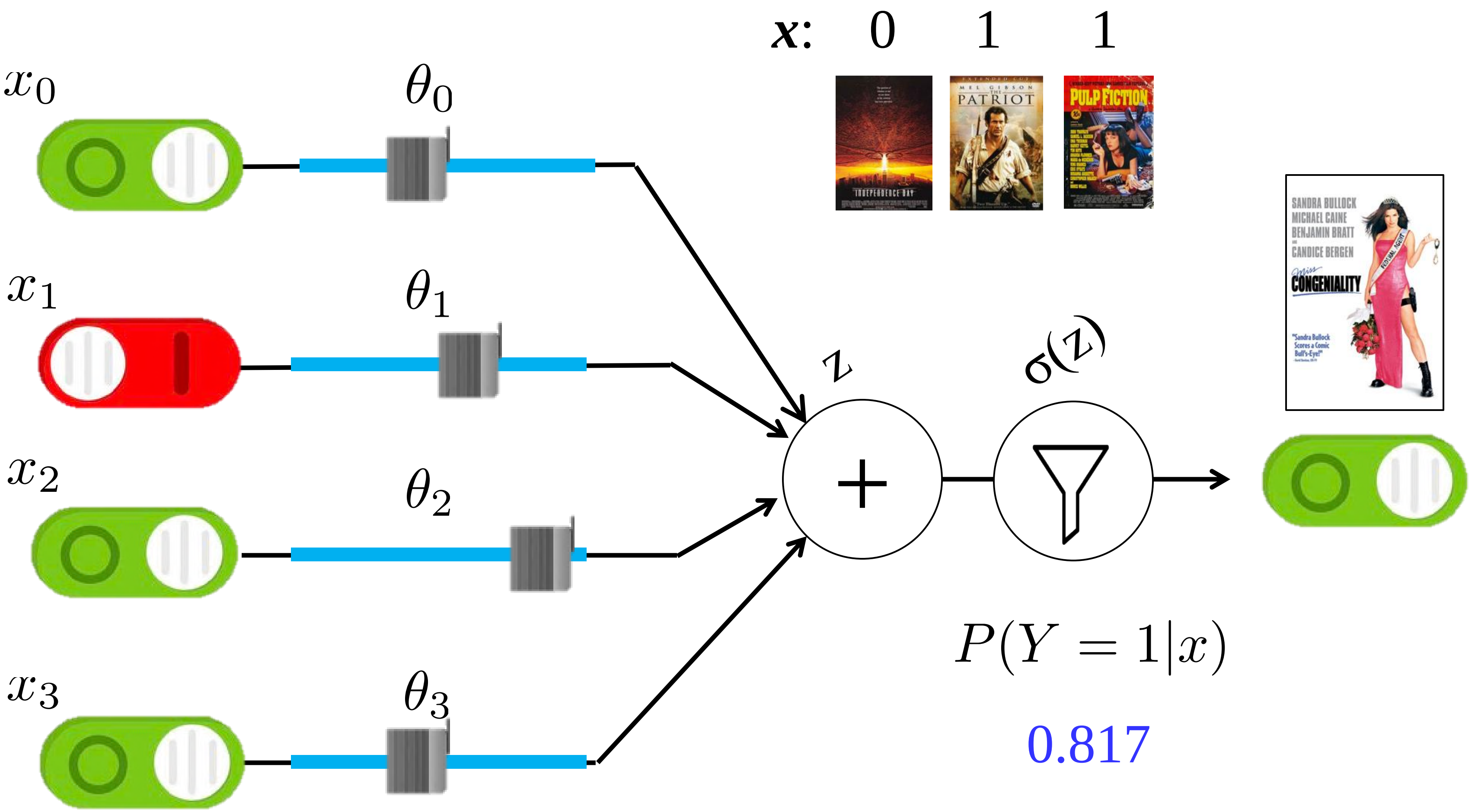
$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Logistic Regression



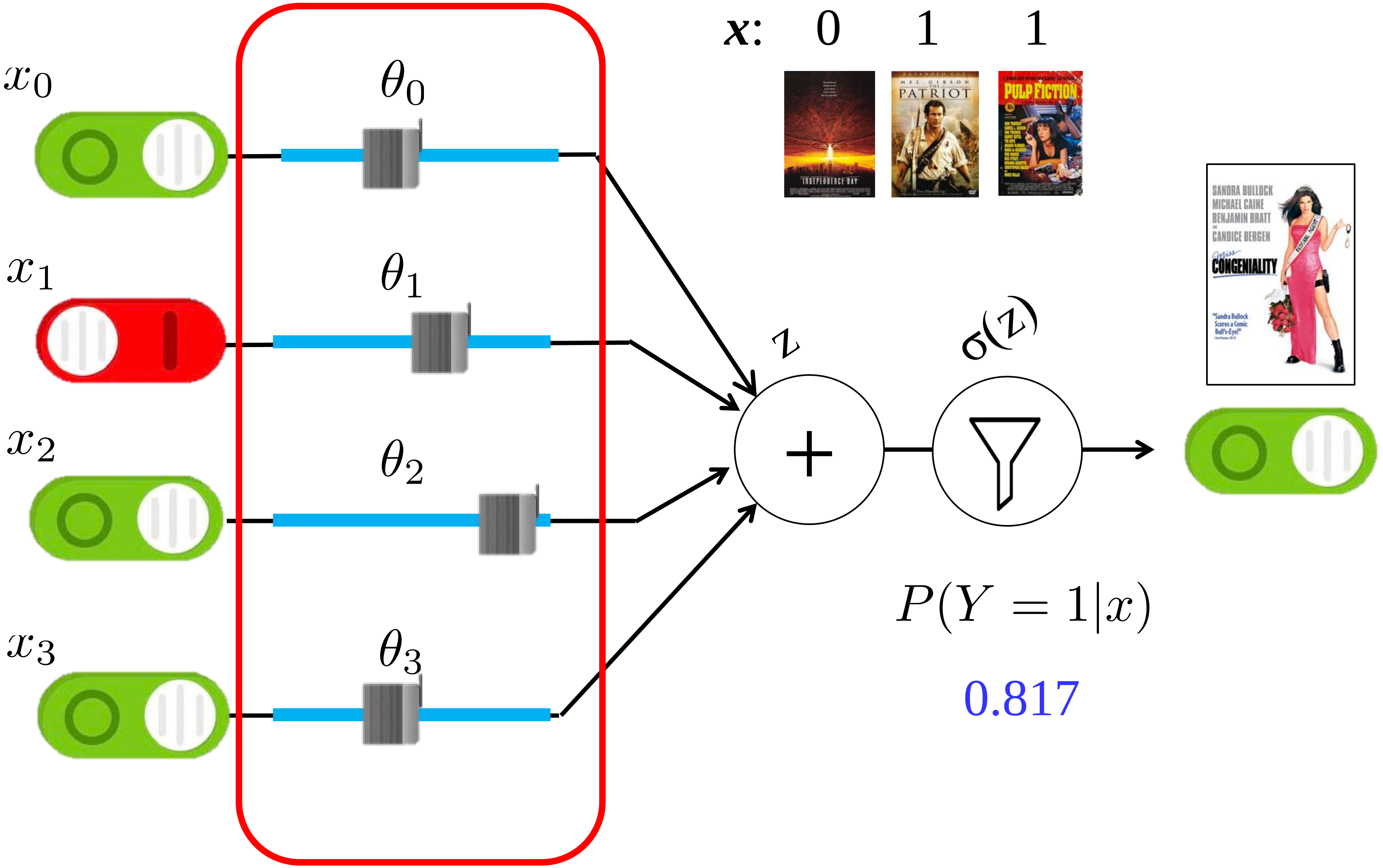
$$P(Y = 1|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Logistic Regression



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

Logistic Regression



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

Math for Logistic Regression

1. Make logistic regression assumption

$$P(Y=1|X=\mathbf{x}) = \sigma(\theta^T\mathbf{x})$$
$$P(Y=0|X=\mathbf{x}) = 1-\sigma(\theta^T\mathbf{x})$$

Often call this $\hat{y}$

2. Calculate the log likelihood for all data

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma(\theta^T\mathbf{x}^{(i)}) + (1-y^{(i)}) \log[1-\sigma(\theta^T\mathbf{x}^{(i)})]$$

3. Get derivative of log likelihood with respect to thetas

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T\mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Logistic Regression Training

Initialize: $\theta_j = 0$ for all $0 \le j \le m$

Repeat many times:

gradient[j] = 0 for all $0 \le j \le m$

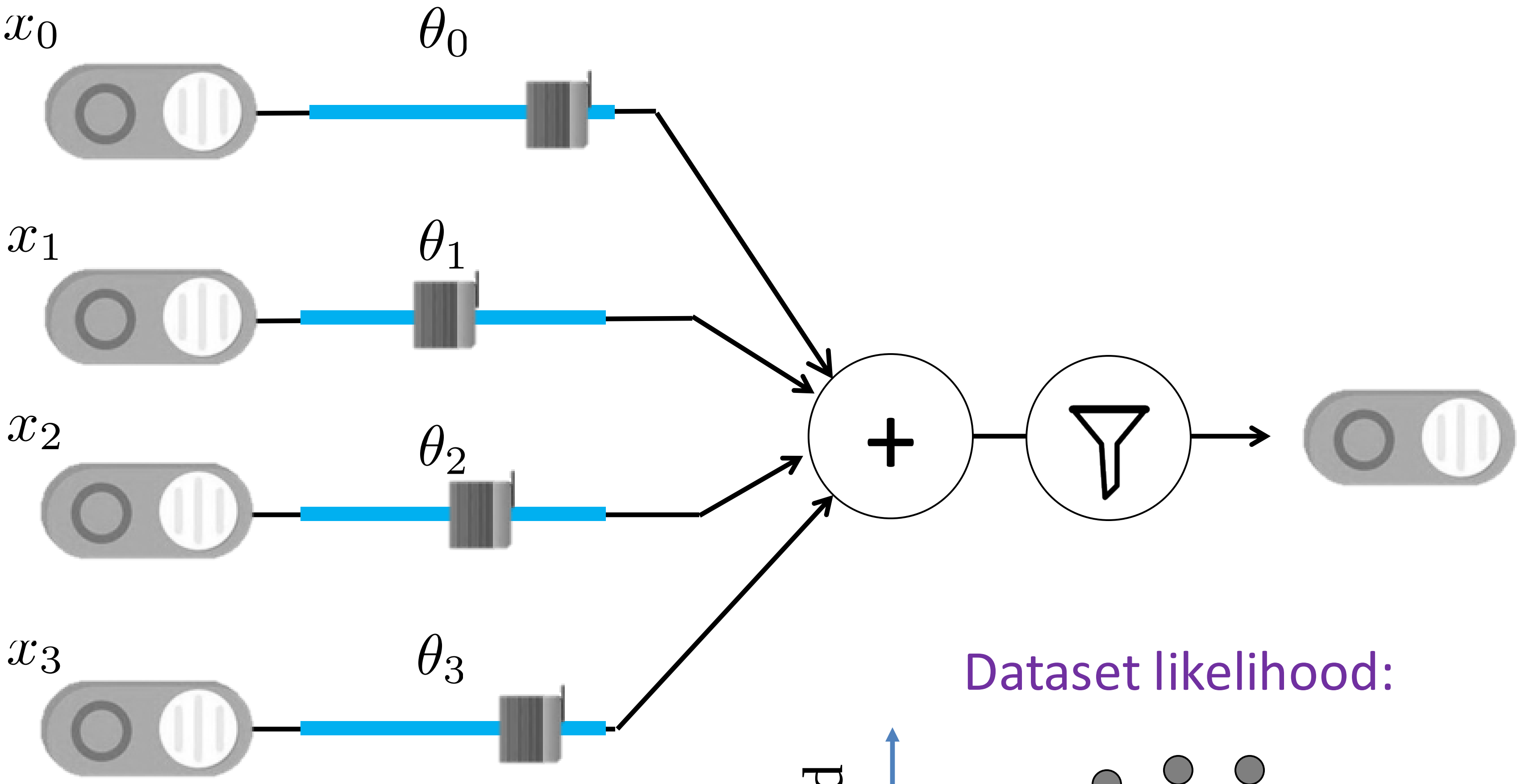
For each parameter j

For each training example (x, y):

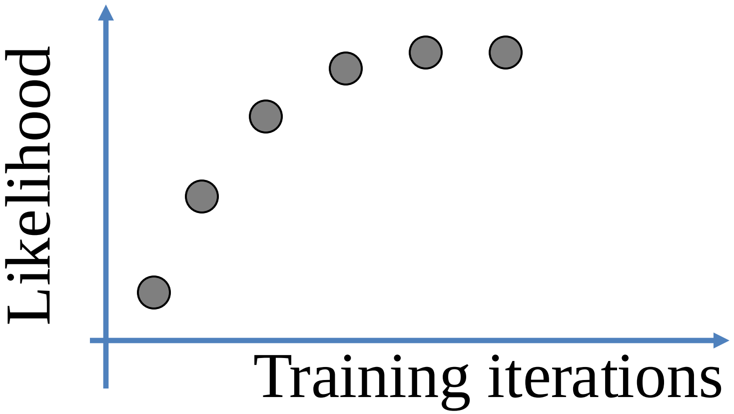
gradient[j] $\mathrel{+}= x_j \left(y - \dfrac{1}{1 + e^{-\theta^{\mathrm{T}} \mathbf{x}}} \right)$

θ_j += η * gradient[j] for all $0 \le j \le m$

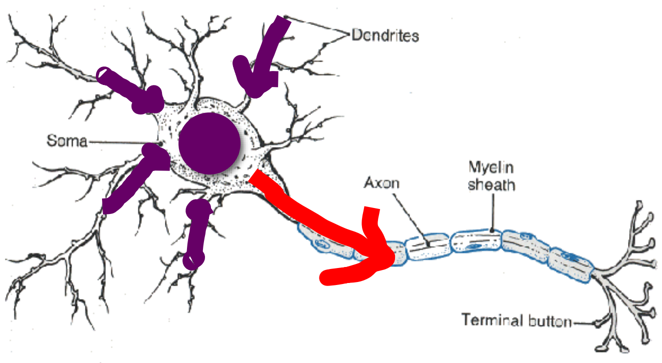
Training

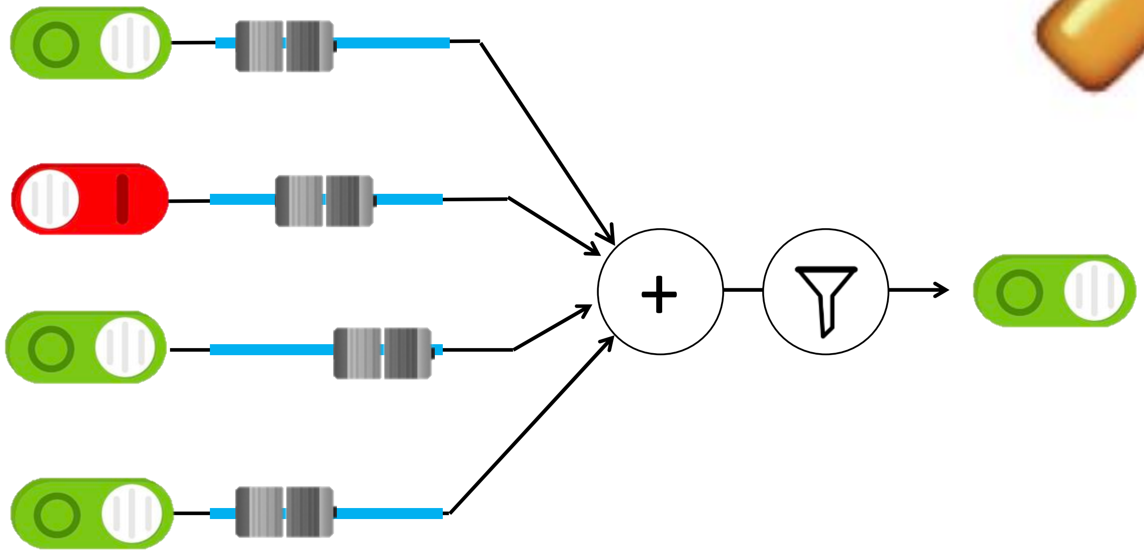


Dataset likelihood:



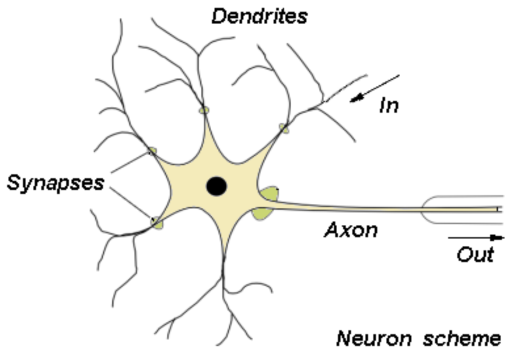
Artificial Neurons

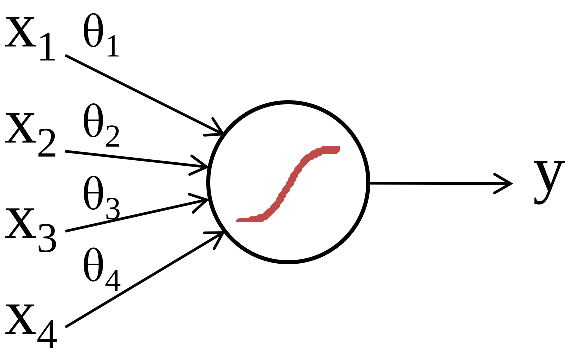




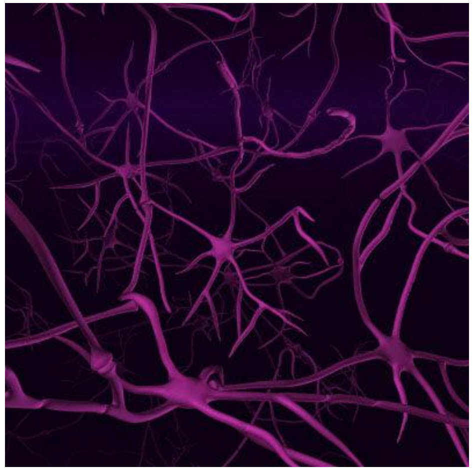
Biological Basis for Neural Networks

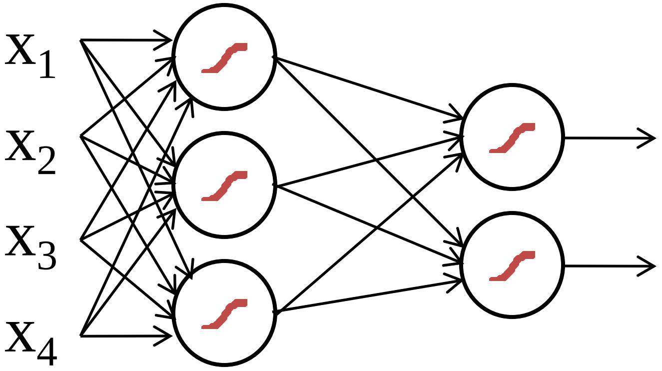
A neuron





Your brain

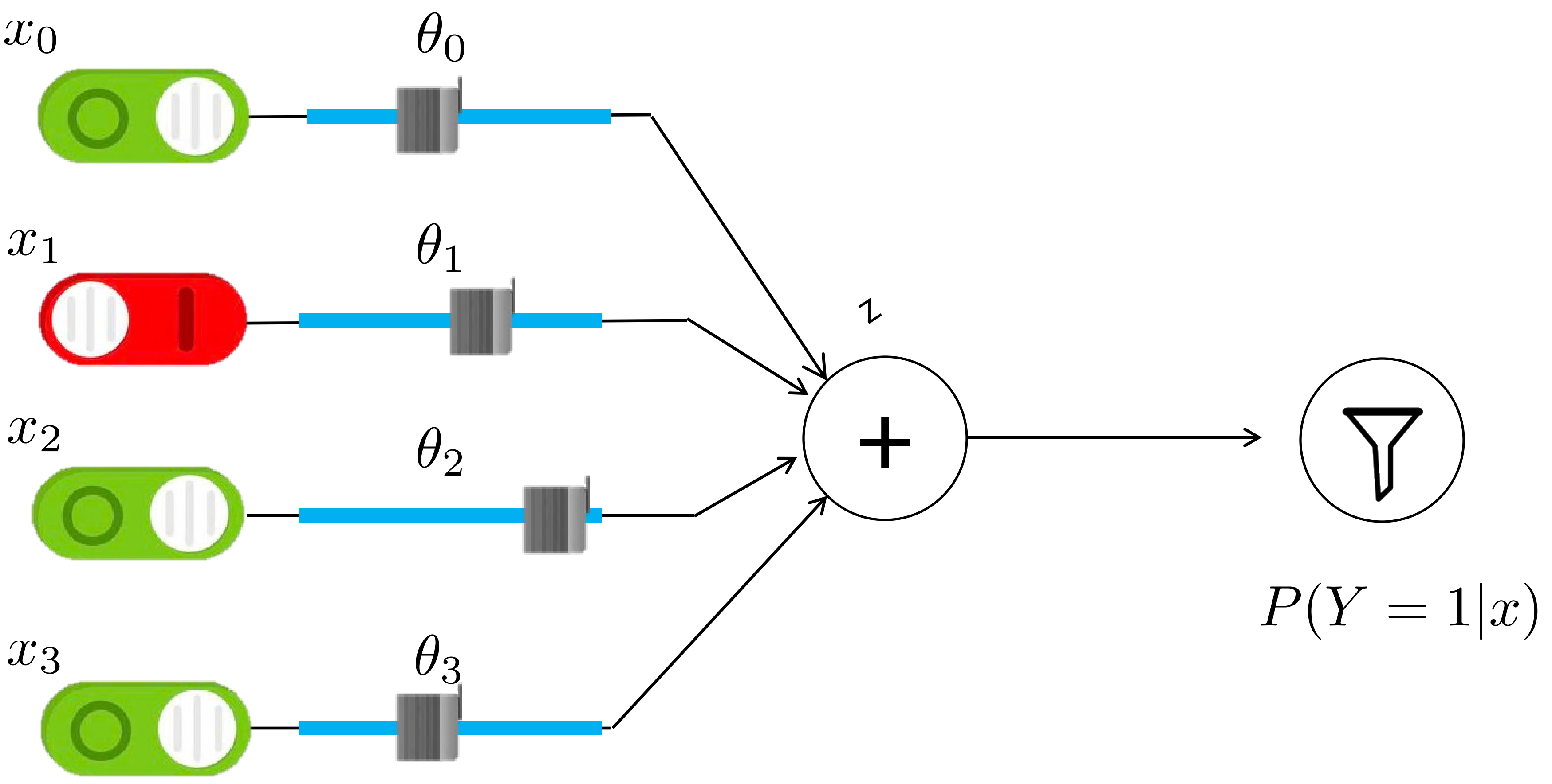




Actually, it's probably someone else's brain

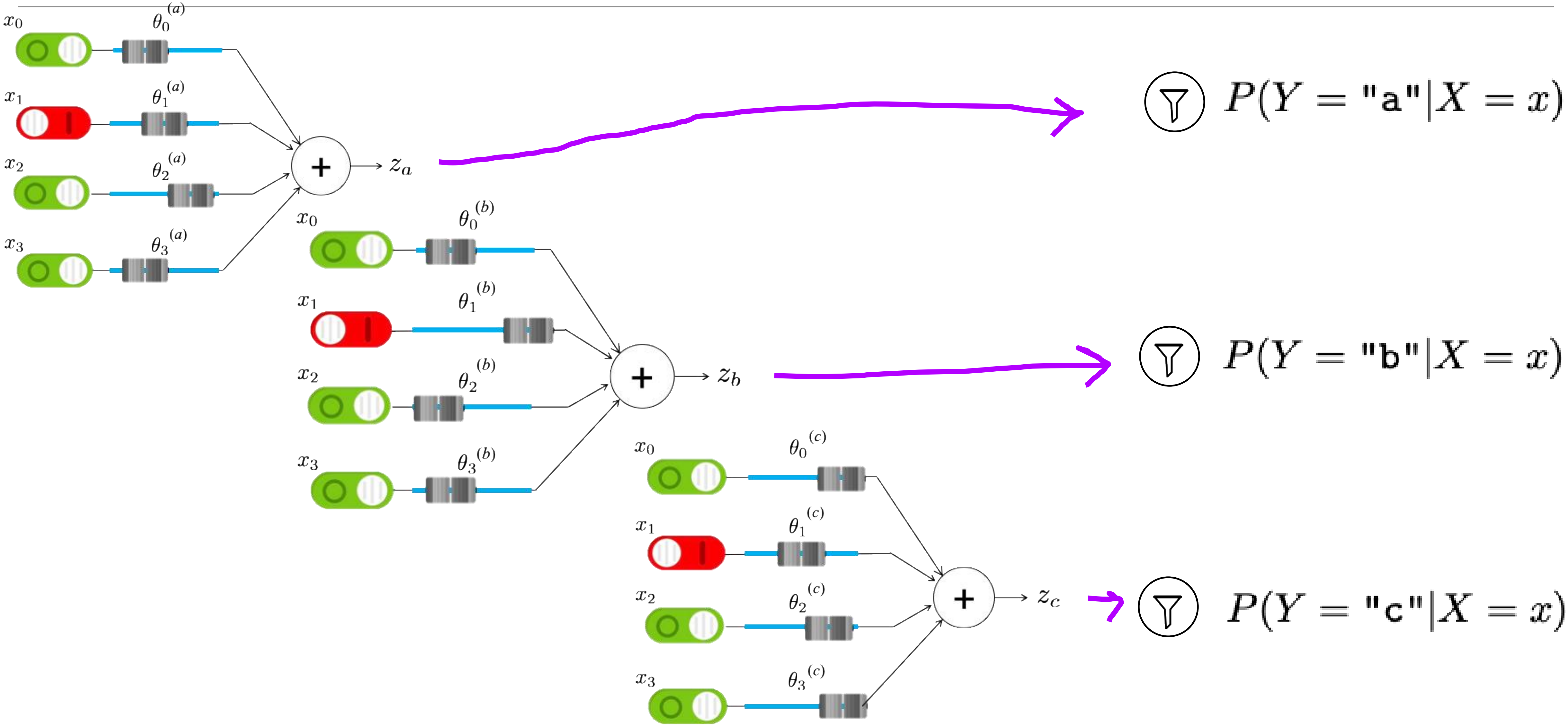
Beyond Classification

Binary Prediction

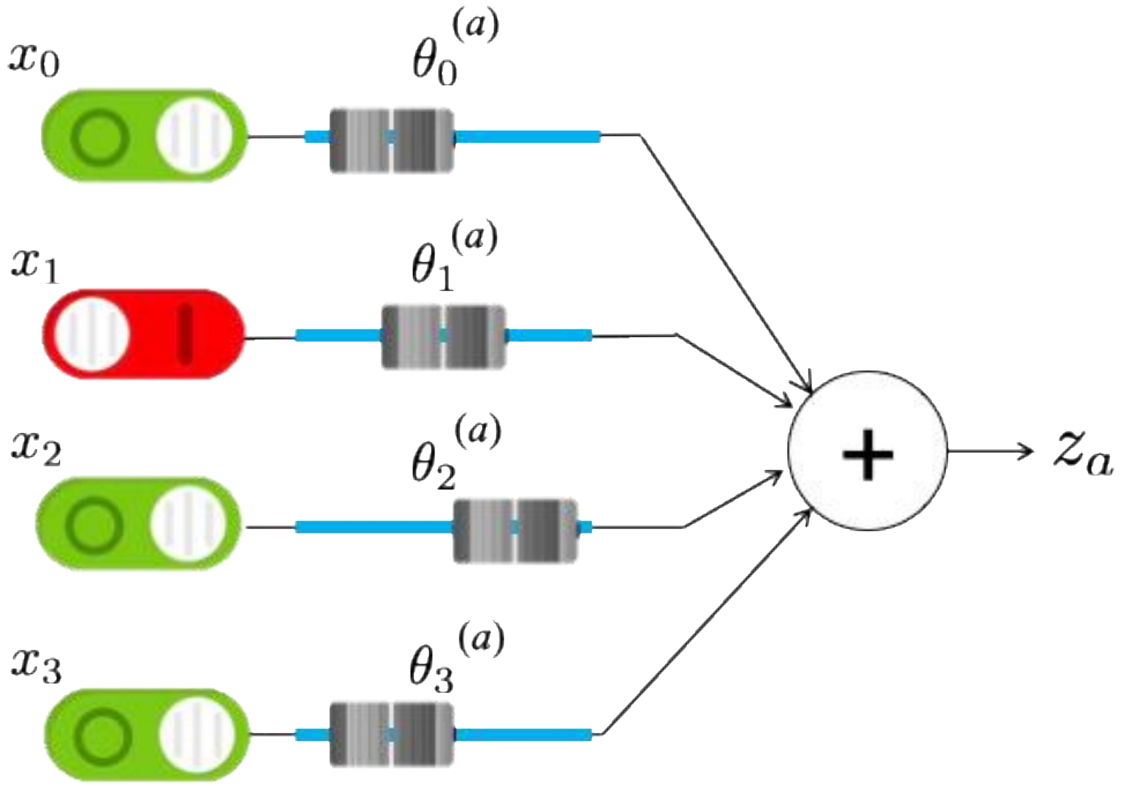


$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma\left(\sum_i \theta_i x_i\right)$$

Predict a Categorical



Predict a Real Value (aka Regression)



y is distributed as a normal with this mean

$$\underset{\theta}{\operatorname{argmax}}\ LL(\theta) = \underset{\theta}{\operatorname{argmax}}\ -\sum_{i=1}^{n}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

Hey it's the sum of squared errors!

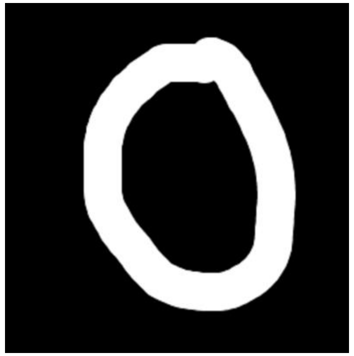
Core idea behind the revolution in AI

(aka Neural Networks)

Deep learning is (at its core) many logistic regression pieces stacked on top of each other.

Digit Recognition Example

Let's make feature vectors from pictures of numbers

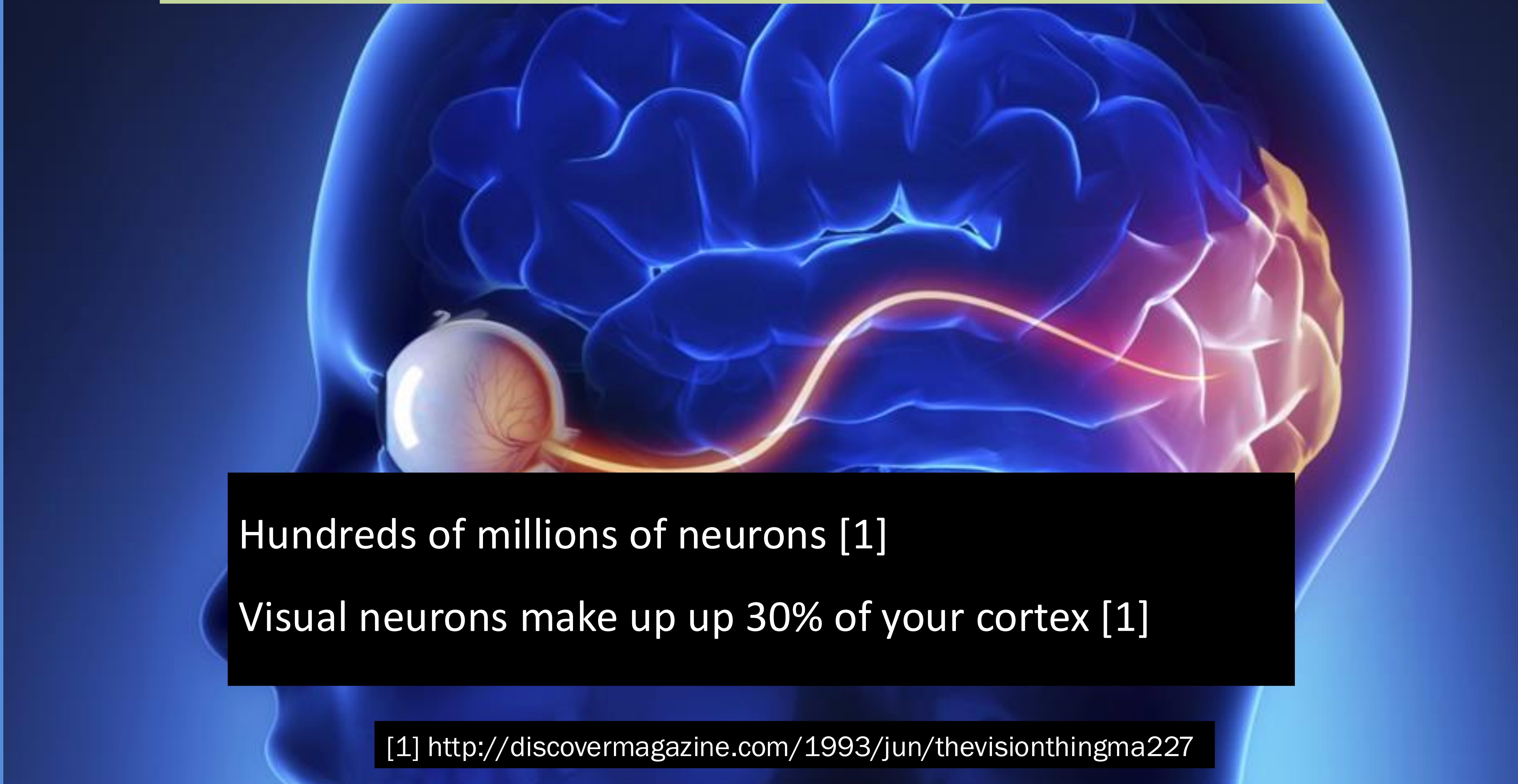


$$\mathbf{x}^{(i)} = [0, 0, 0, 0, \ldots, 1, 0, 0, 1, \ldots 0, 0, 1, 0]$$
$$y^{(i)} = 0$$

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

Computer Vision

Vision in your Brain

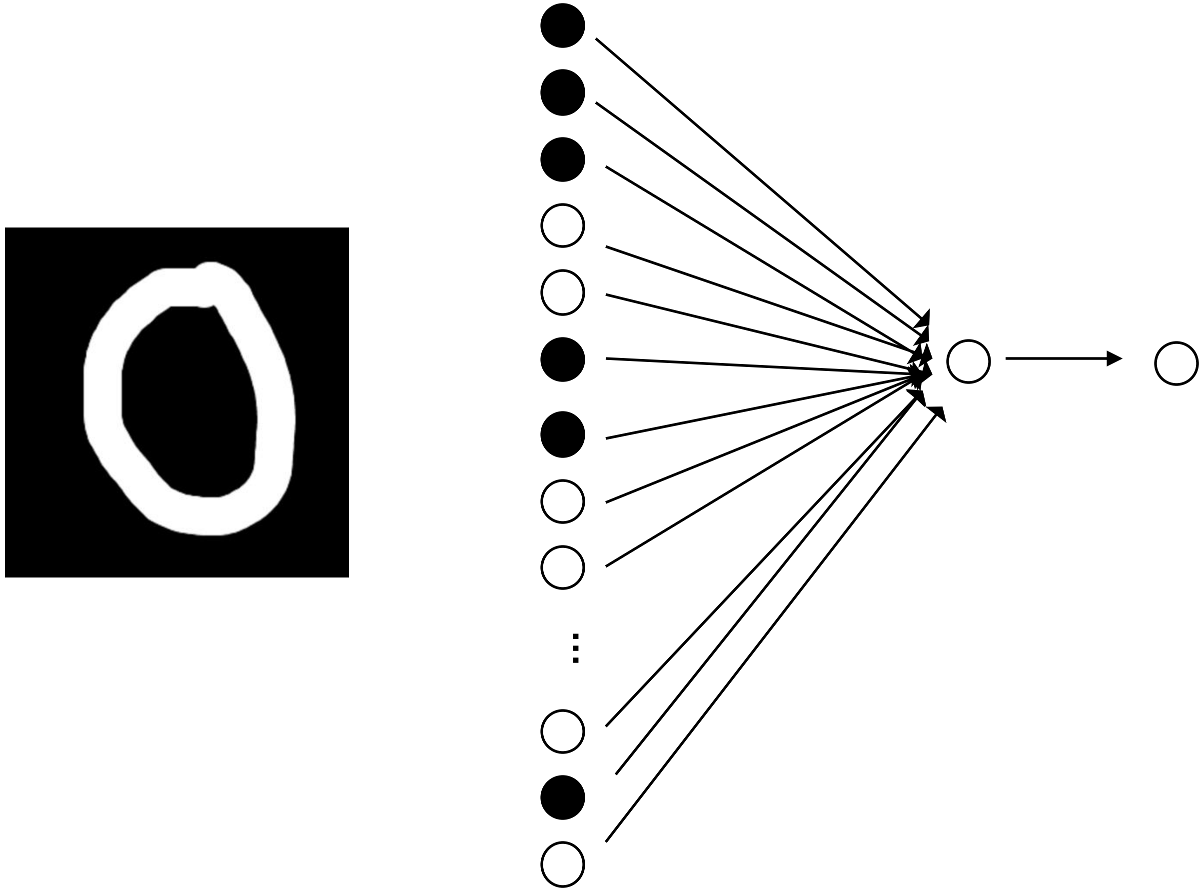


Hundreds of millions of neurons [1]

Visual neurons make up up 30% of your cortex [1]

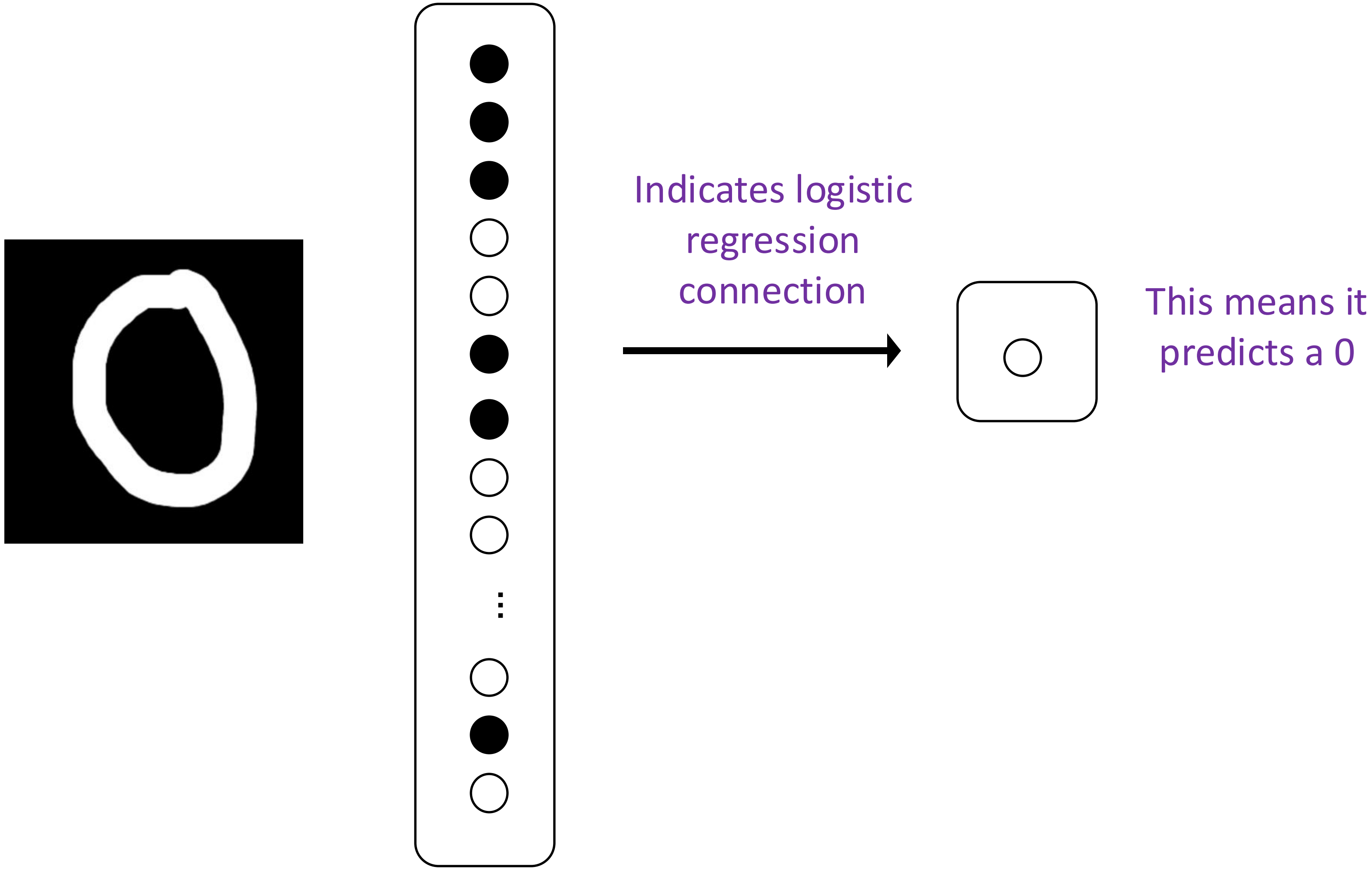
[1] http://discovermagazine.com/1993/jun/thevisionthingma227

Logistic Regression

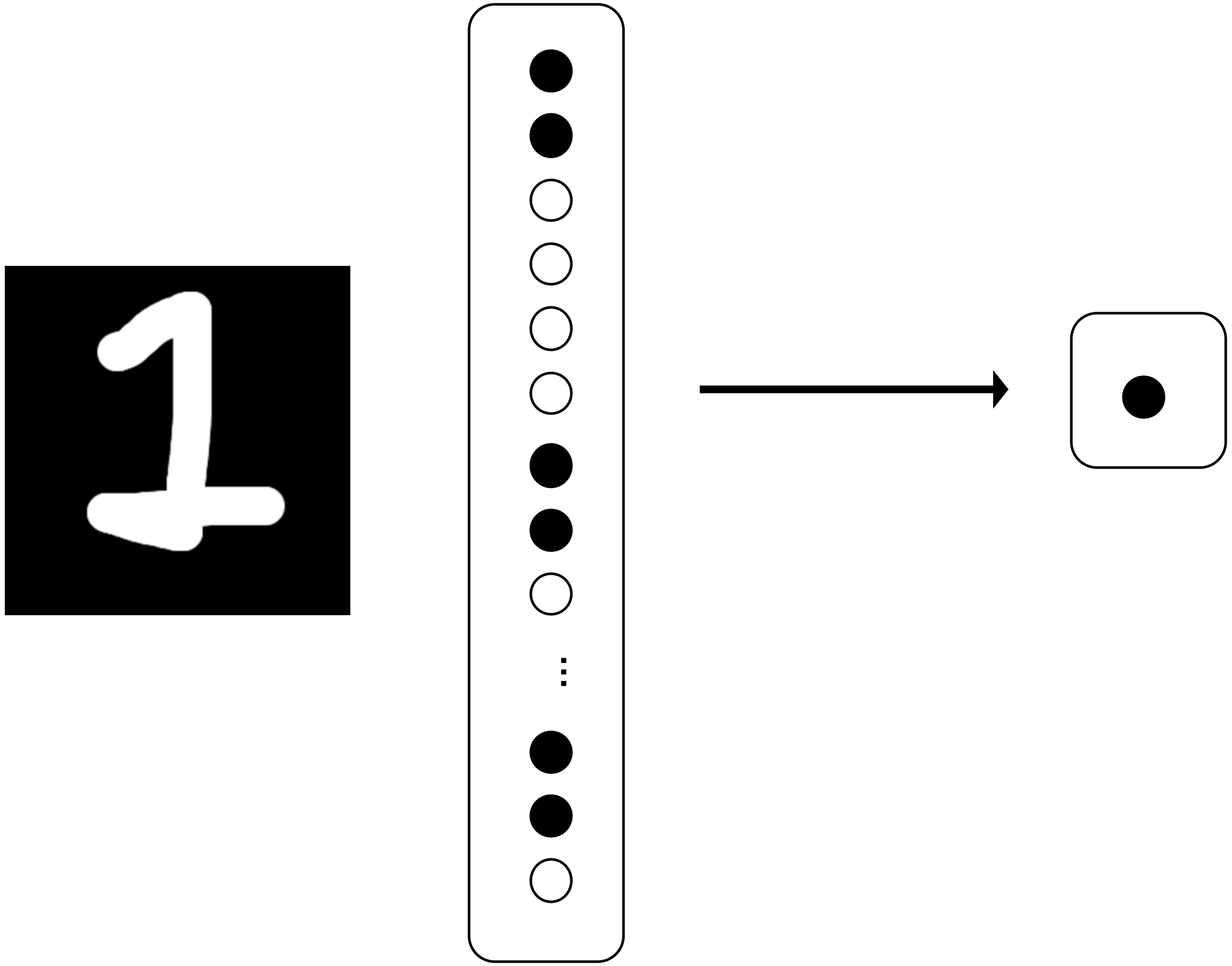


This means it predicts a 0

Logistic Regression

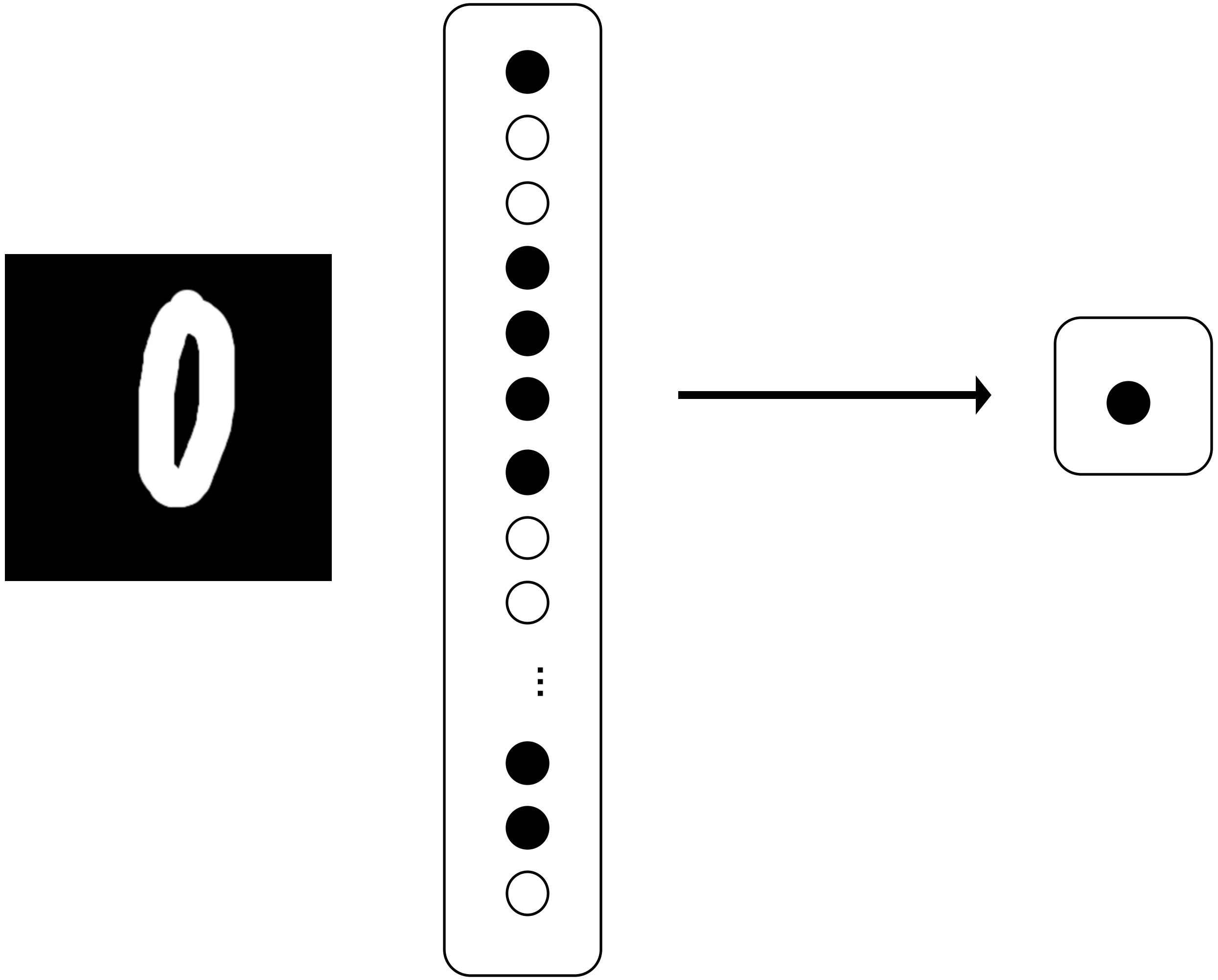


Logistic Regression



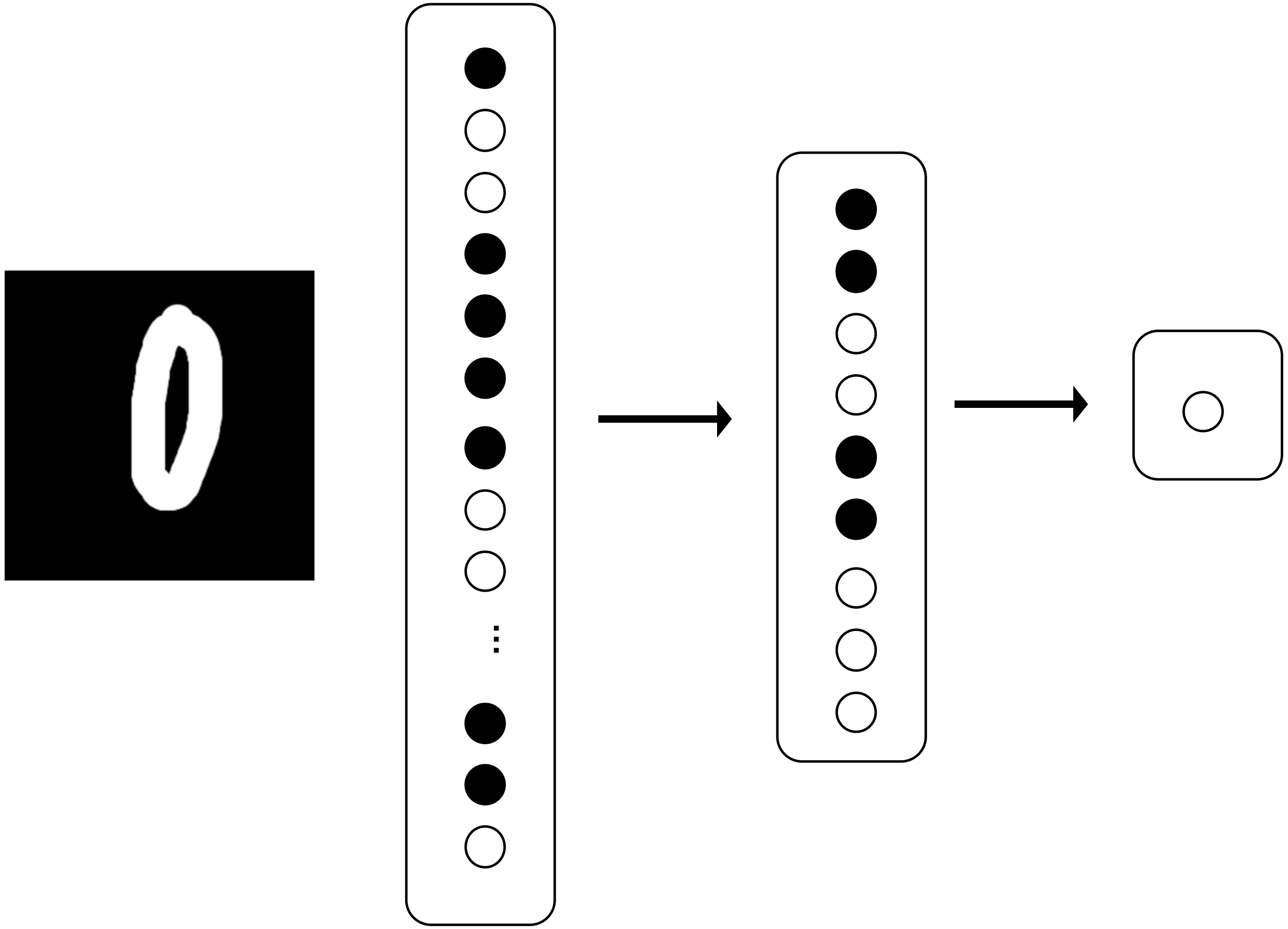
This means it
predicts a 1

Not So Good



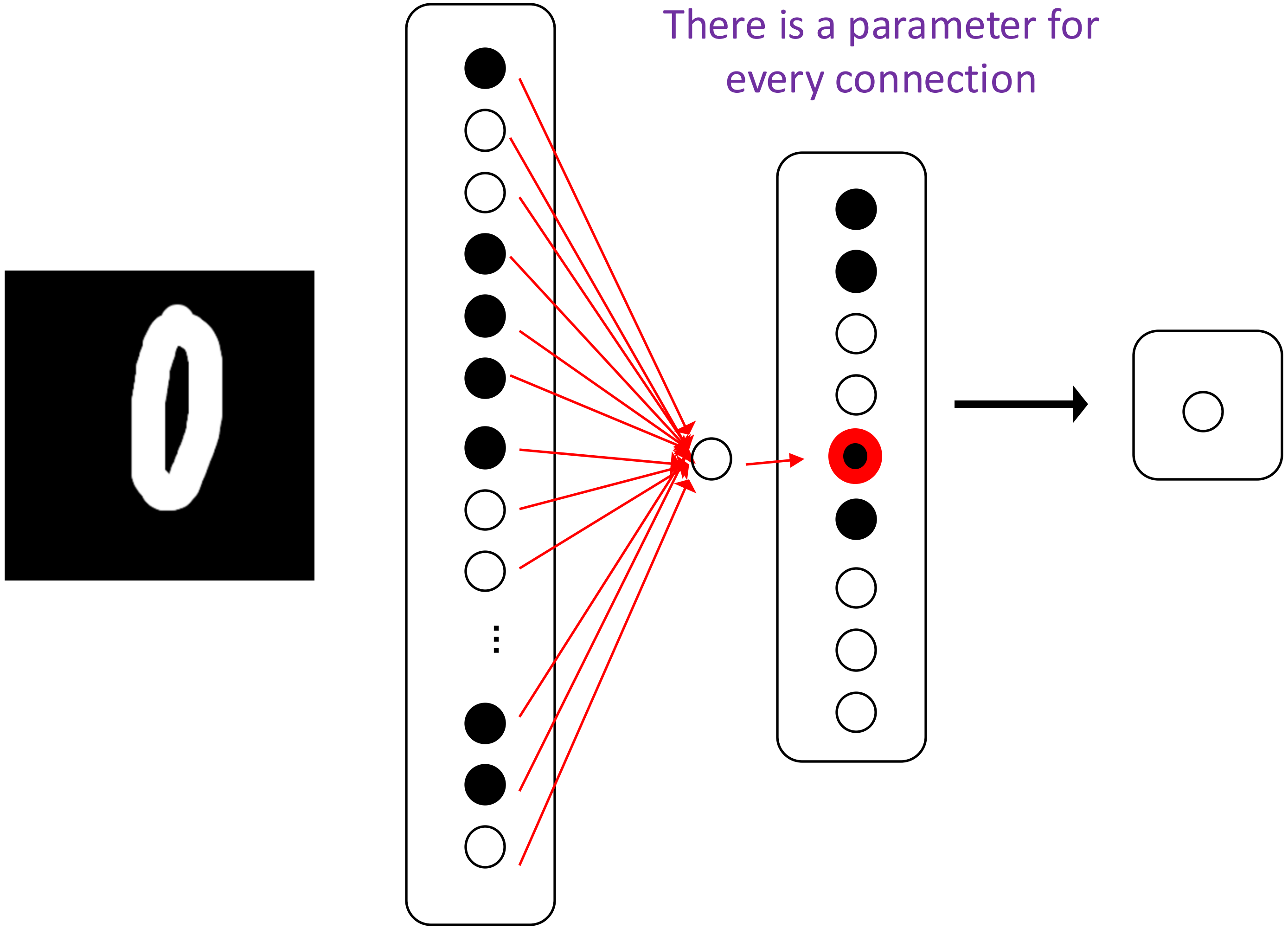
This means it predicts a 1

We Can Put Neurons Together



This means it predicts a 0

We Can Put Neurons Together

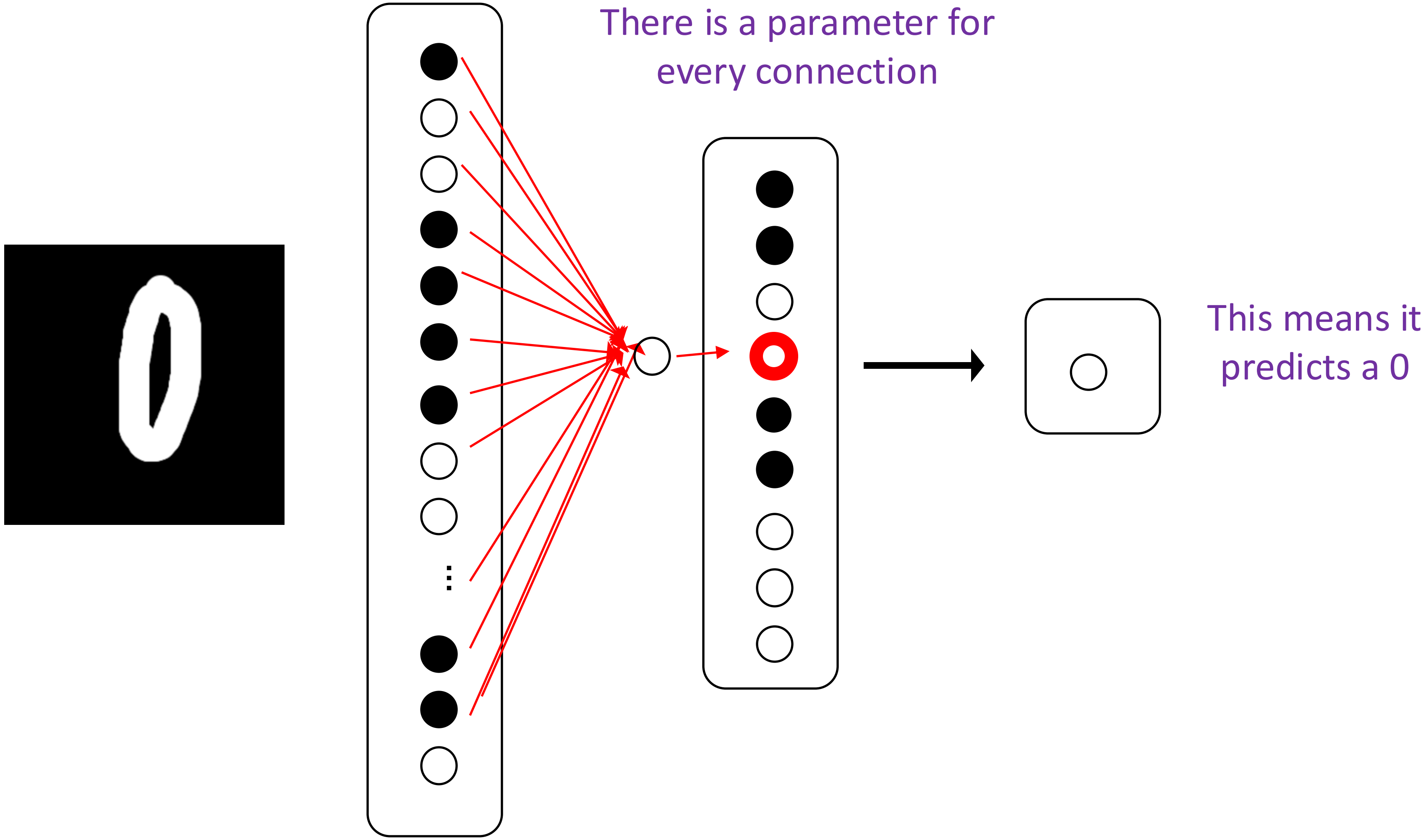


There is a parameter for every connection

This means it predicts a 0

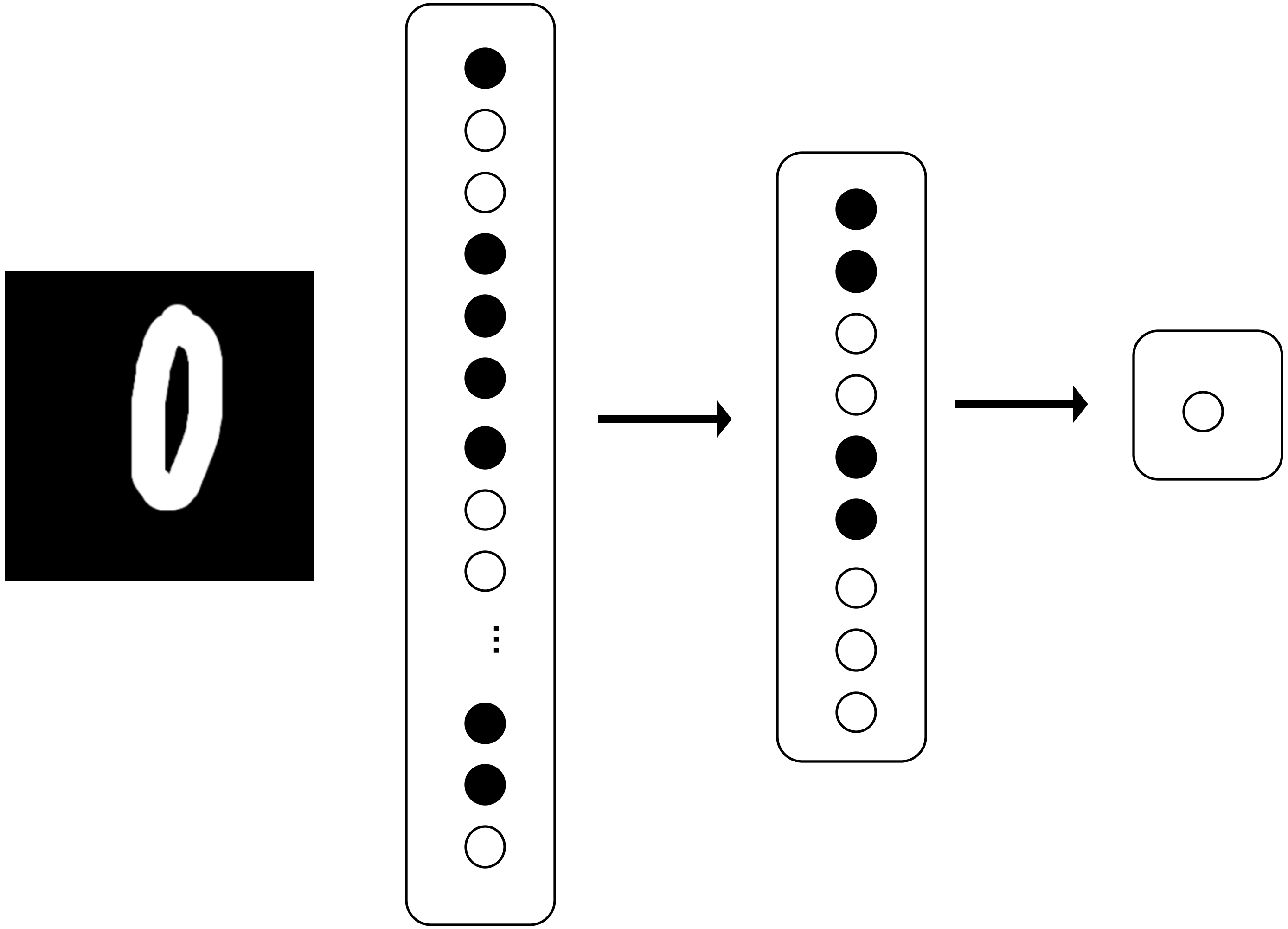
Look at a single “hidden” neuron

We Can Put Neurons Together



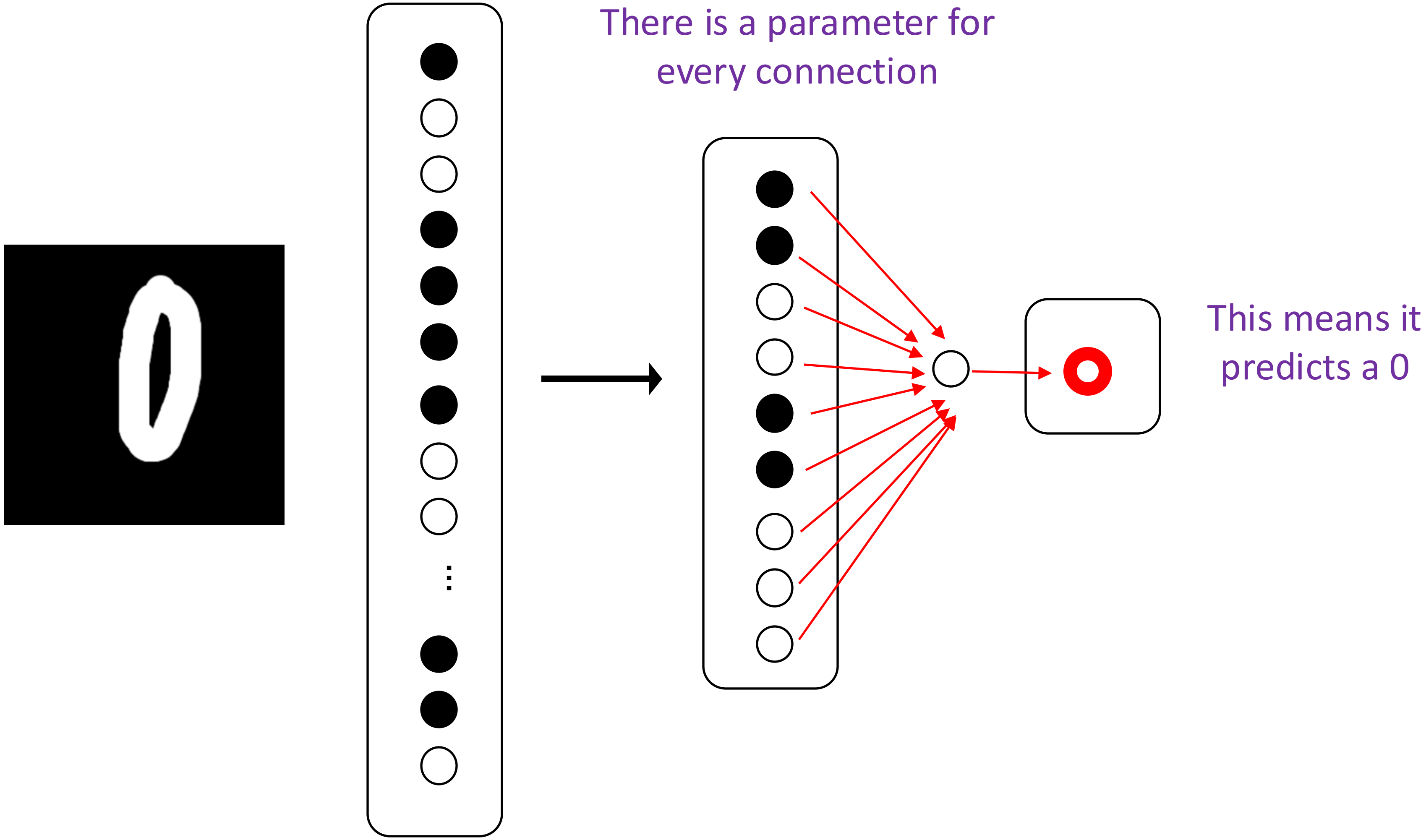
Look at another "hidden" neuron

We Can Put Neurons Together



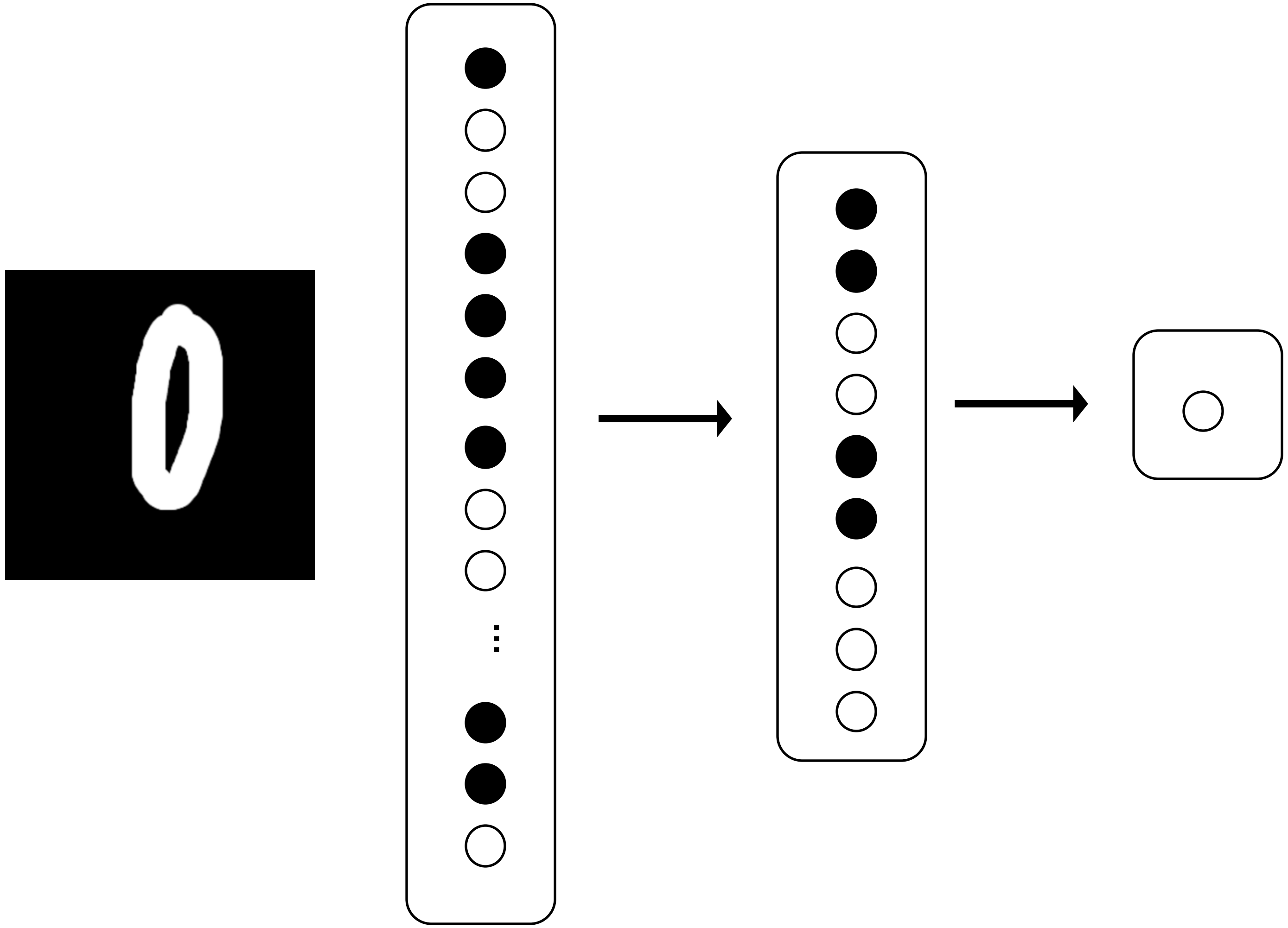
This means it
predicts a 0

We Can Put Neurons Together



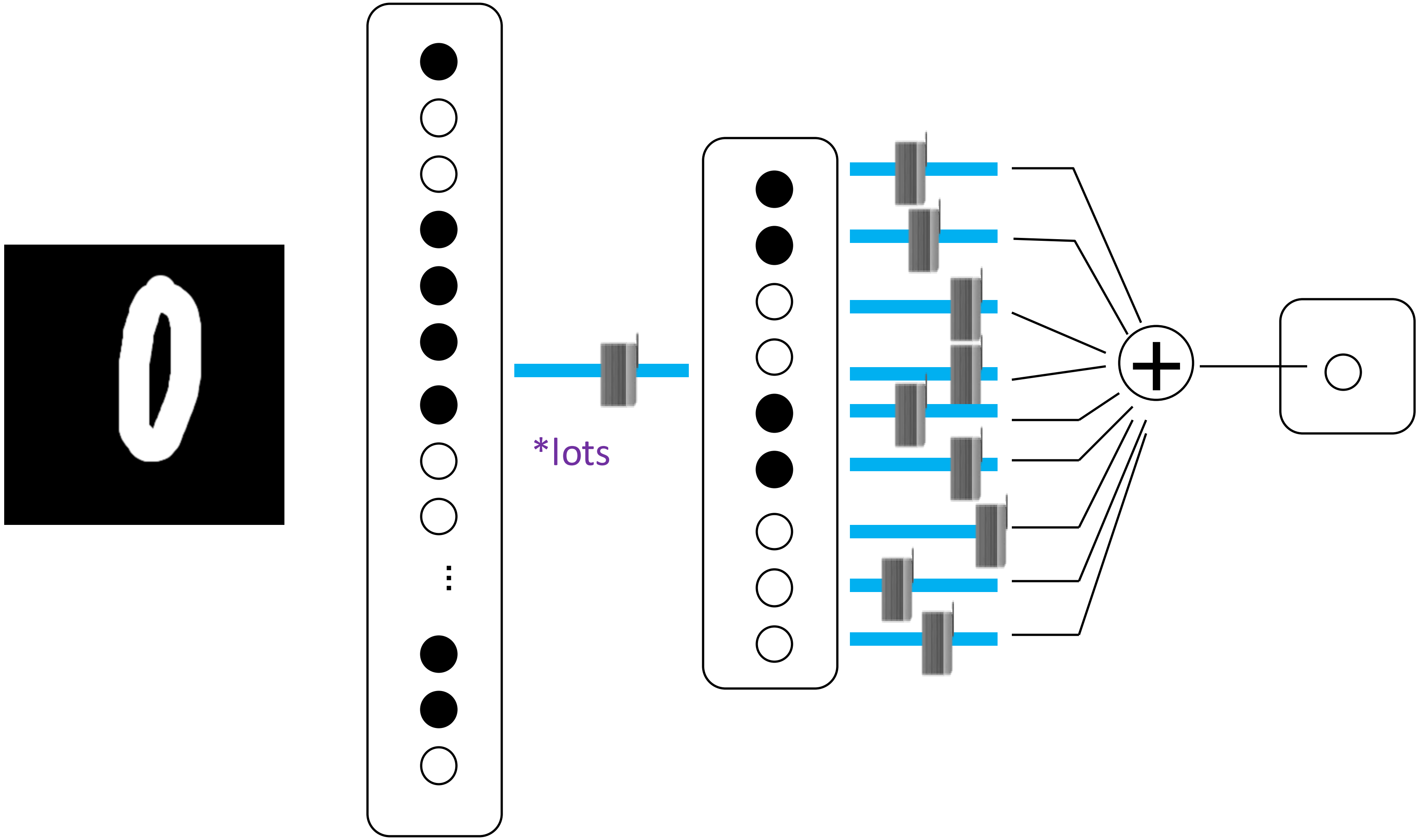
Look at another neuron

We Can Put Neurons Together

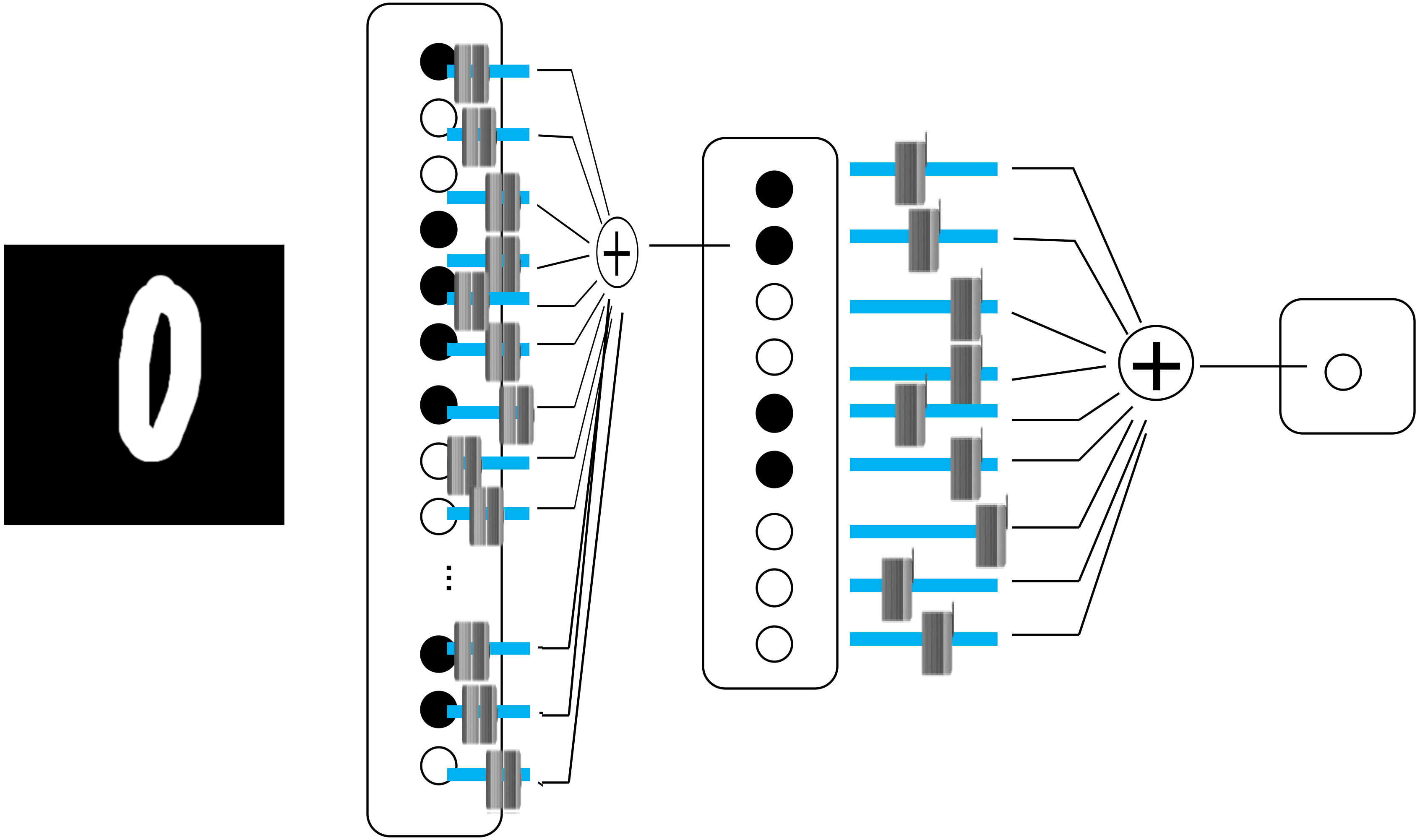


This means it
predicts a 0

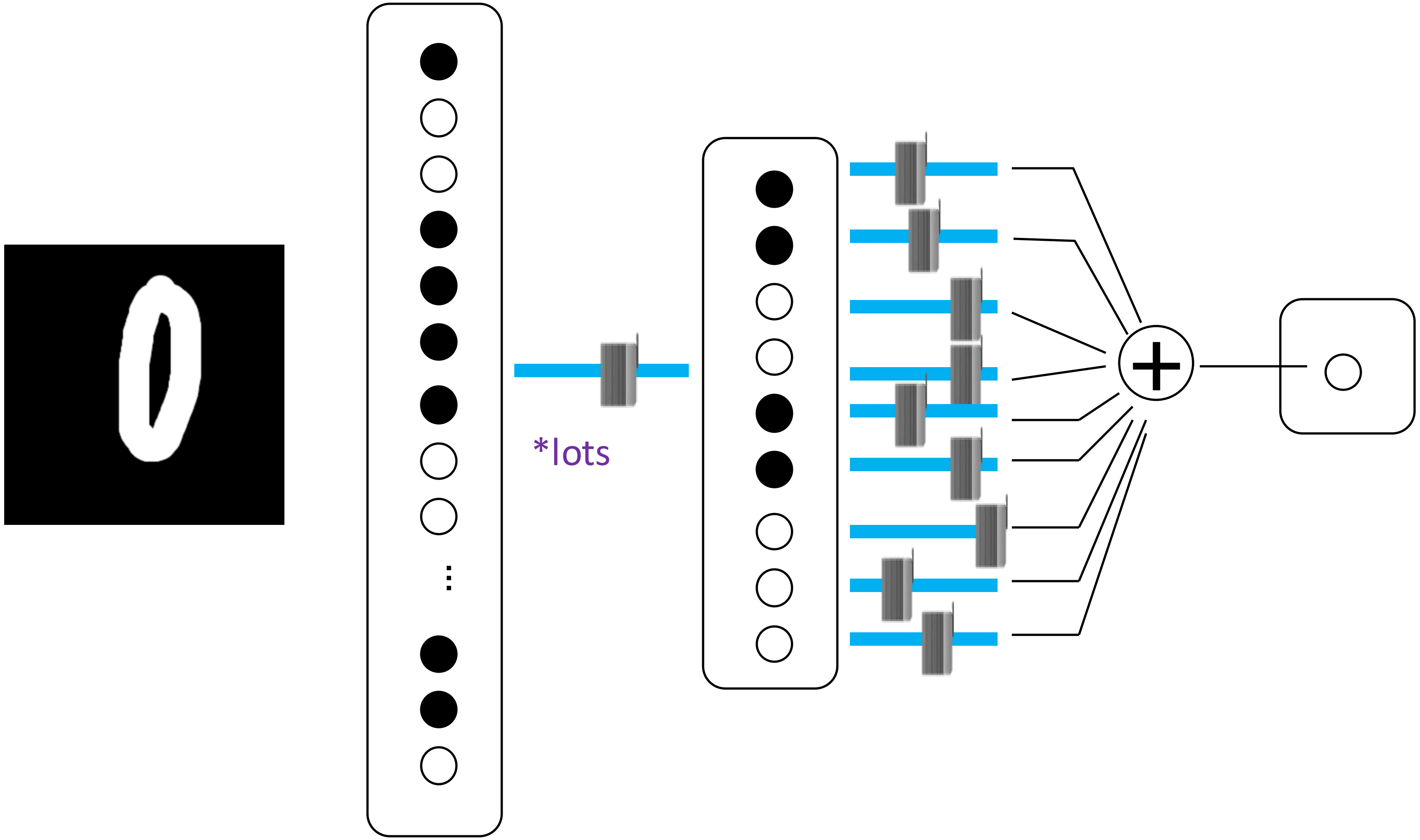
We Can Put Neurons Together



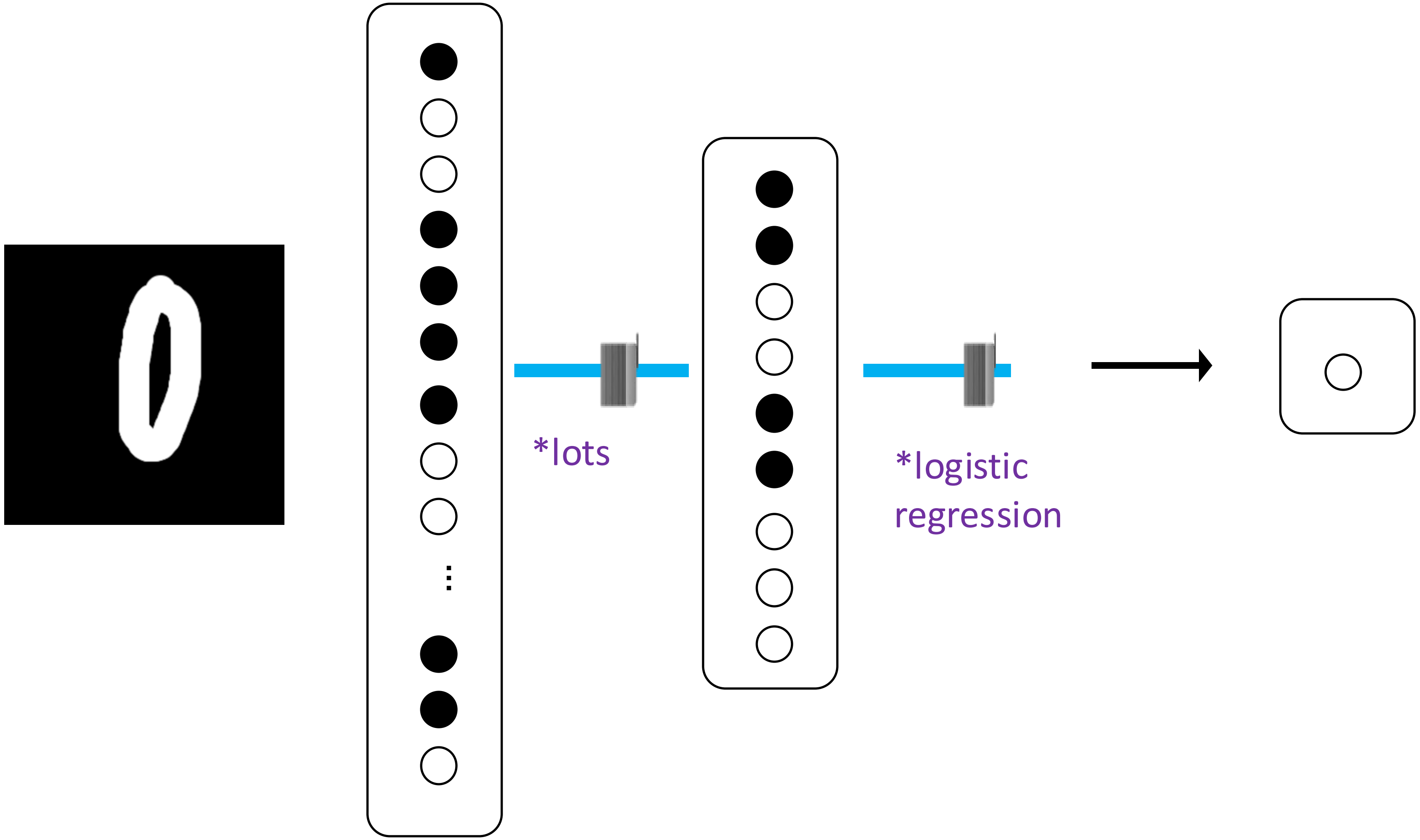
We Can Put Neurons Together



We Can Put Neurons Together



We Can Put Neurons Together



Deep learning

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

def A **neural network** is
(at its core) many logistic regression pieces stacked on top of each other.

$[1,0,\ldots,1]$
$\boldsymbol{x}$, input

LOL

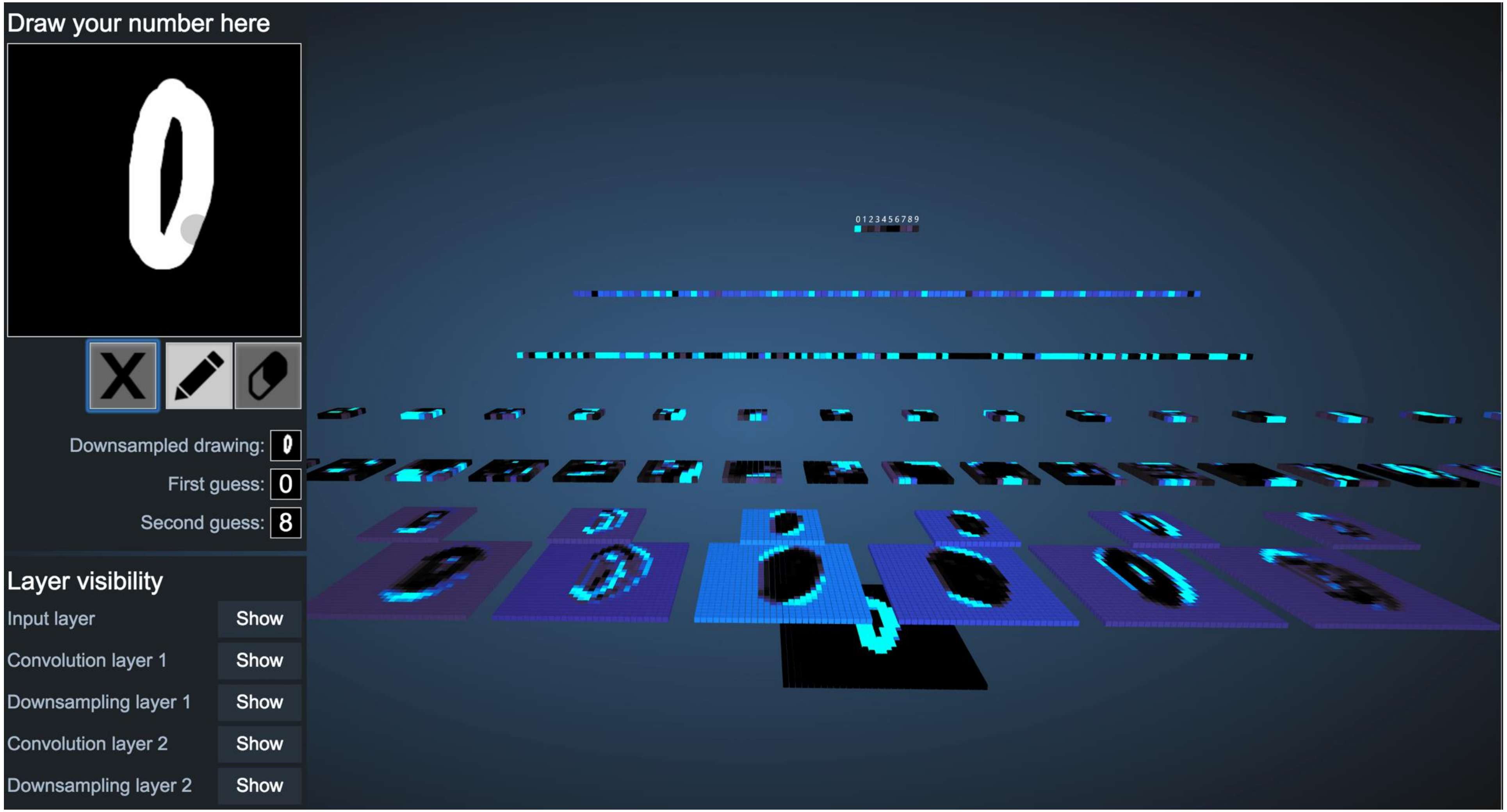
Lots of Logistic (regressions)

$\hat{y}$, output
$P(Y|\boldsymbol{X}=\boldsymbol{x})$

> 0.5?

Yes.
Predict 1

Demonstration



https://adamharley.com/nn_vis/cnn/2d.html

Stanford University

Deep learning gets its ***intelligence*** from its thetas (aka its parameters) train.py

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt

def main():
    # get the data
    train_loader, test_loader = download_data()
    print(f"Training examples: {len(train_loader.dataset)}")
    print(f"Test examples: {len(test_loader.dataset)}")

    # a very simple and fast nn
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28*28, 1024),
        nn.Sigmoid(),
        nn.Linear(1024, 1024),
        nn.Sigmoid(),
        nn.Linear(1024, 10)
    )

    # define the loss and optimizer ONCE outside run_train
    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
```

TERMINAL

```
Untrained accuracy: 9.58%
Training ...
Epoch 1, Test Accuracy: 94.16%
Epoch 2, Test Accuracy: 95.87%
Epoch 3, Test Accuracy: 96.80%
Epoch 4, Test Accuracy: 97.53%
Epoch 5, Test Accuracy: 97.64%
Epoch 6, Test Accuracy: 97.42%
Epoch 7, Test Accuracy: 97.93%
Epoch 8, Test Accuracy: 97.95%
```

How do we train?

MLE of Thetas!

First: Learning Goals…

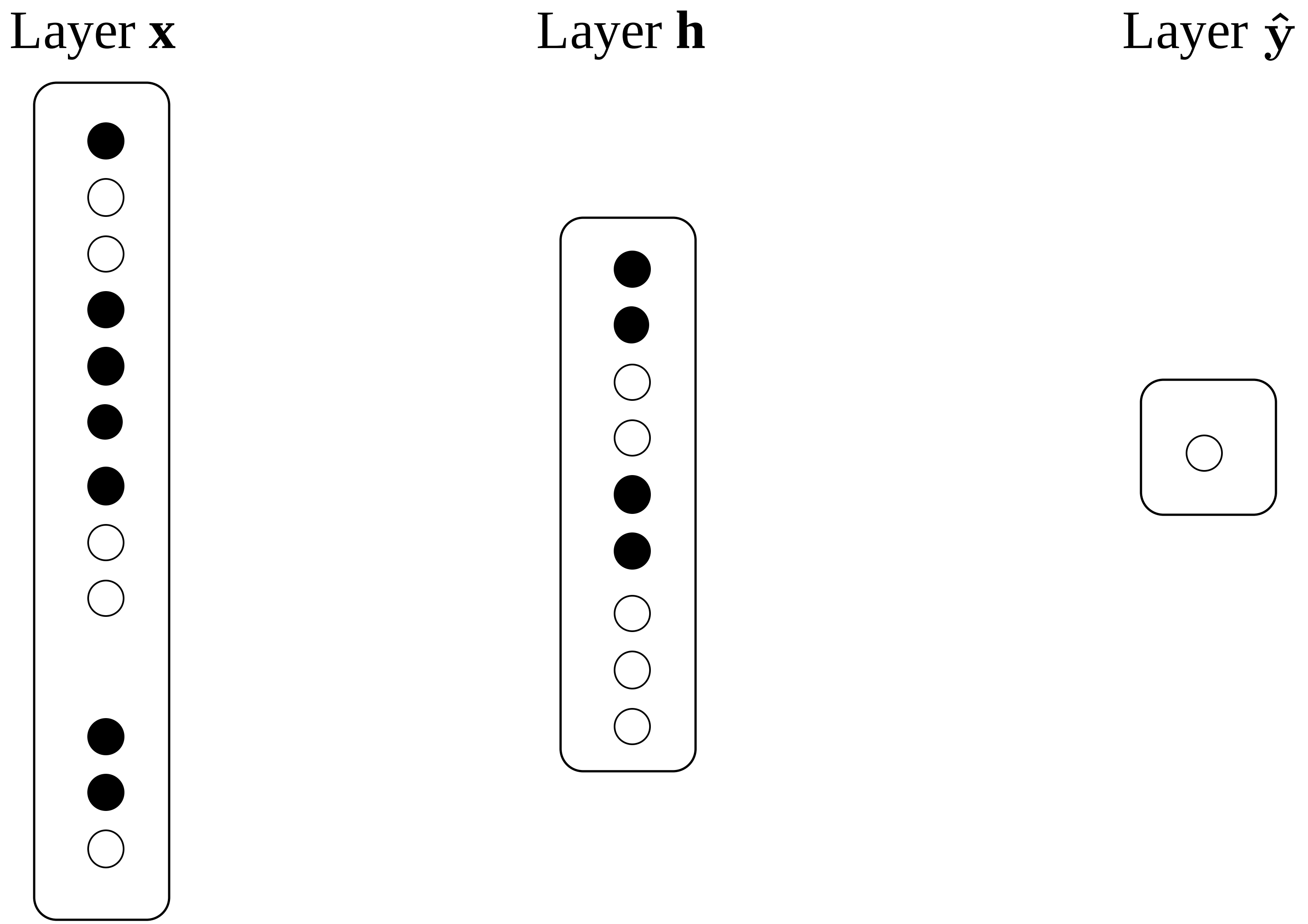
1. Understand Chain Rule as ♡ of Deep Learning

2. Demystify:
Deep Learning is MLE

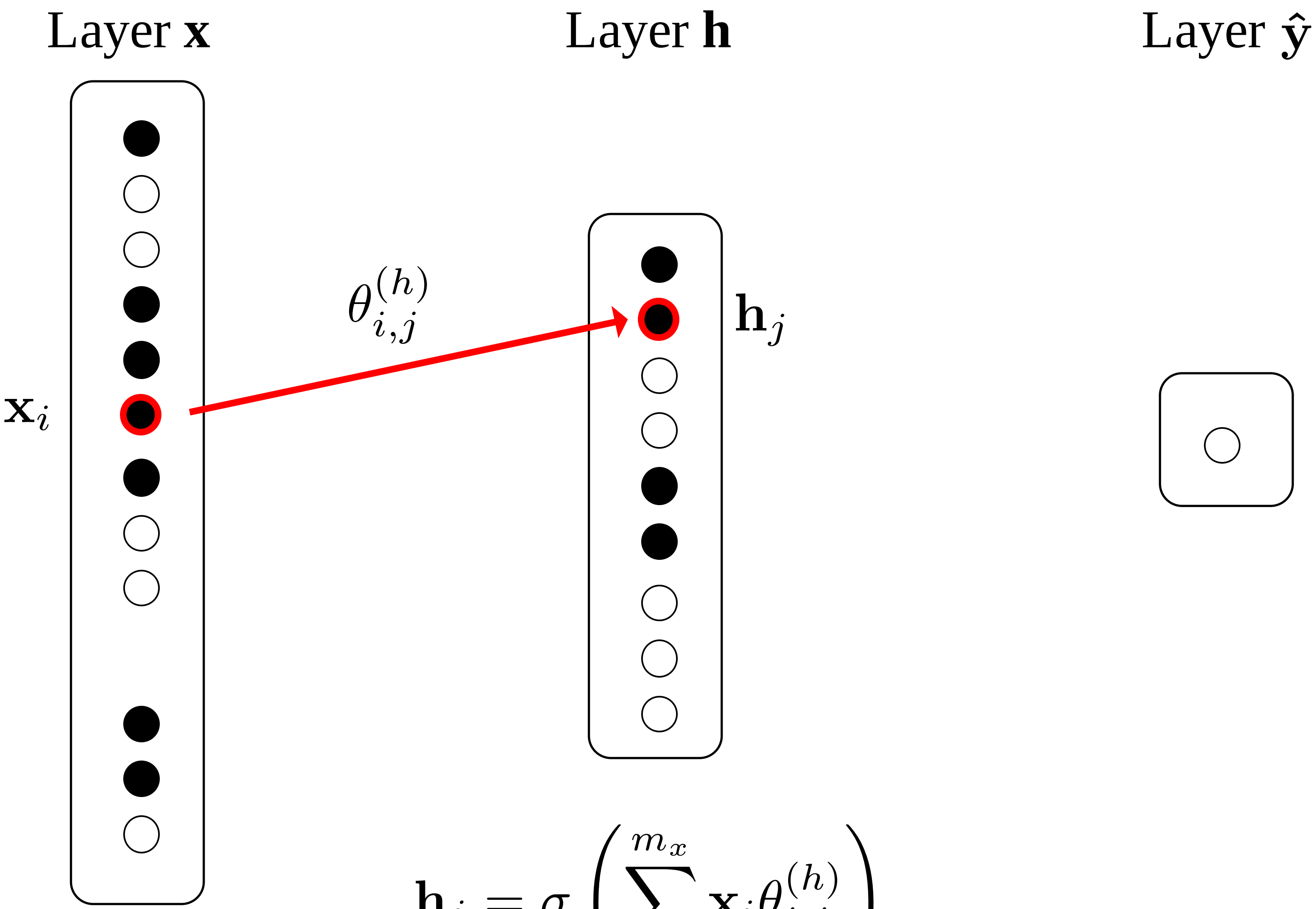
3. Become experts of logistic regression

Math worth knowing:

New Notation

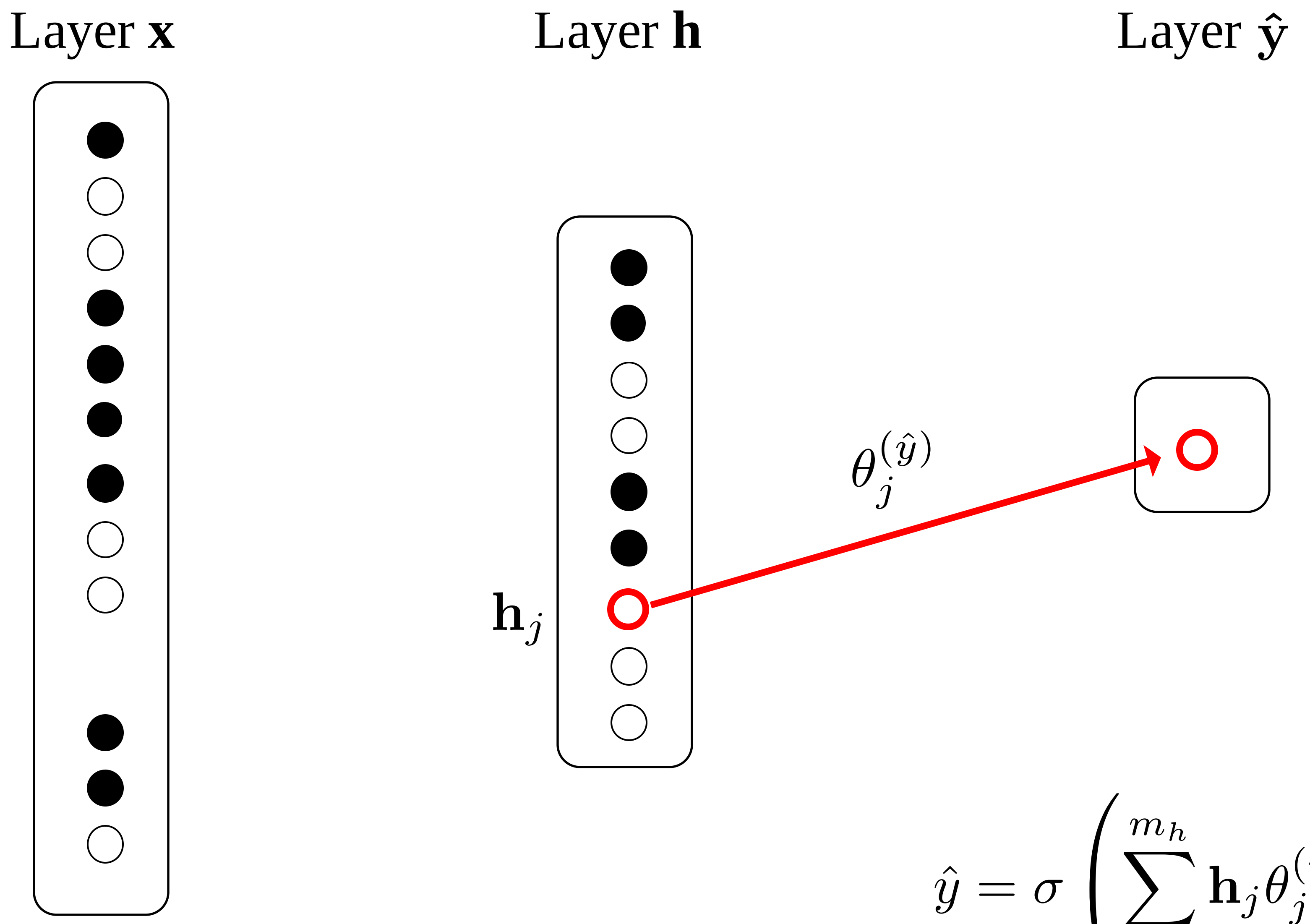


New Notation



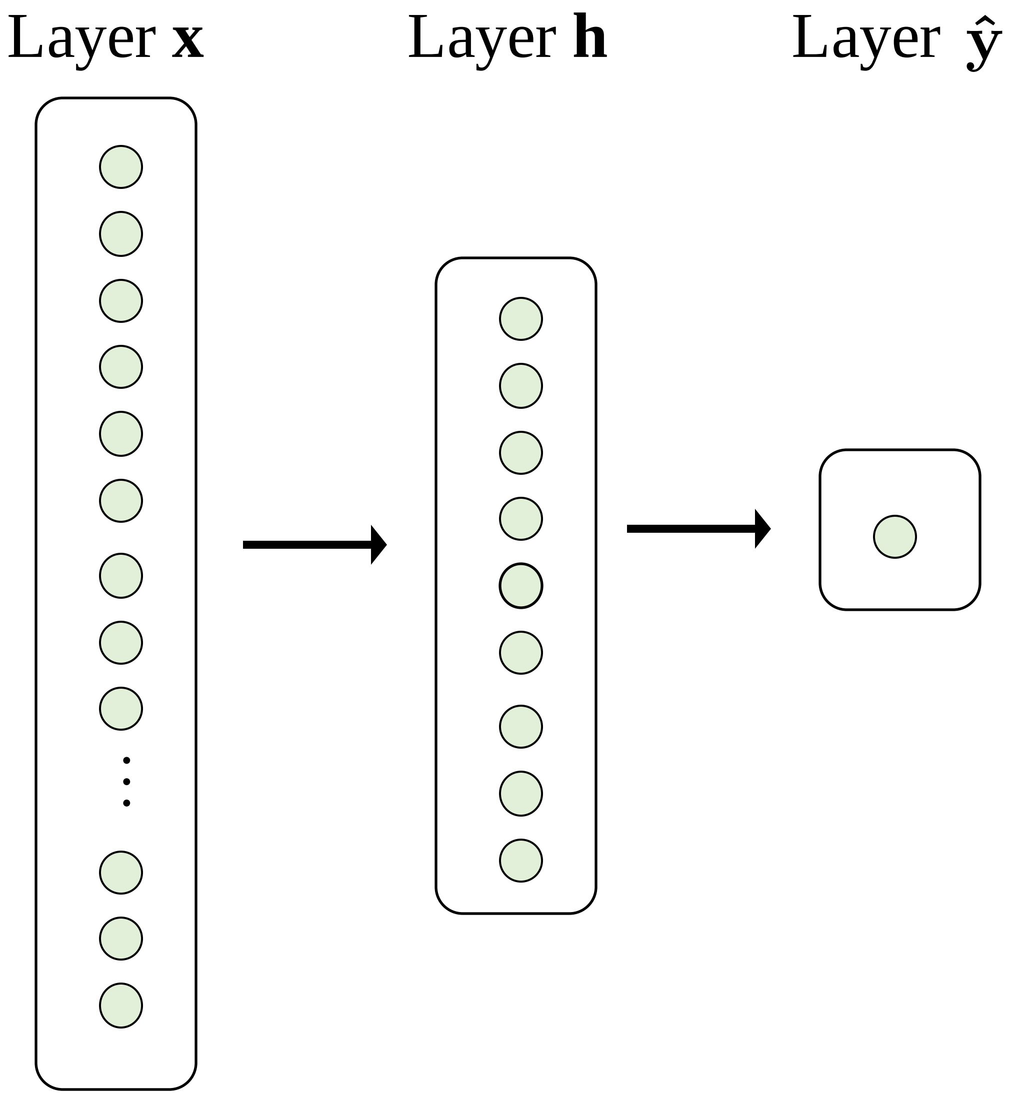
$$\mathbf{h}_j = \sigma\left(\sum_{i=0}^{m_x} \mathbf{x}_i \theta^{(h)}_{i,j}\right)$$

New Notation

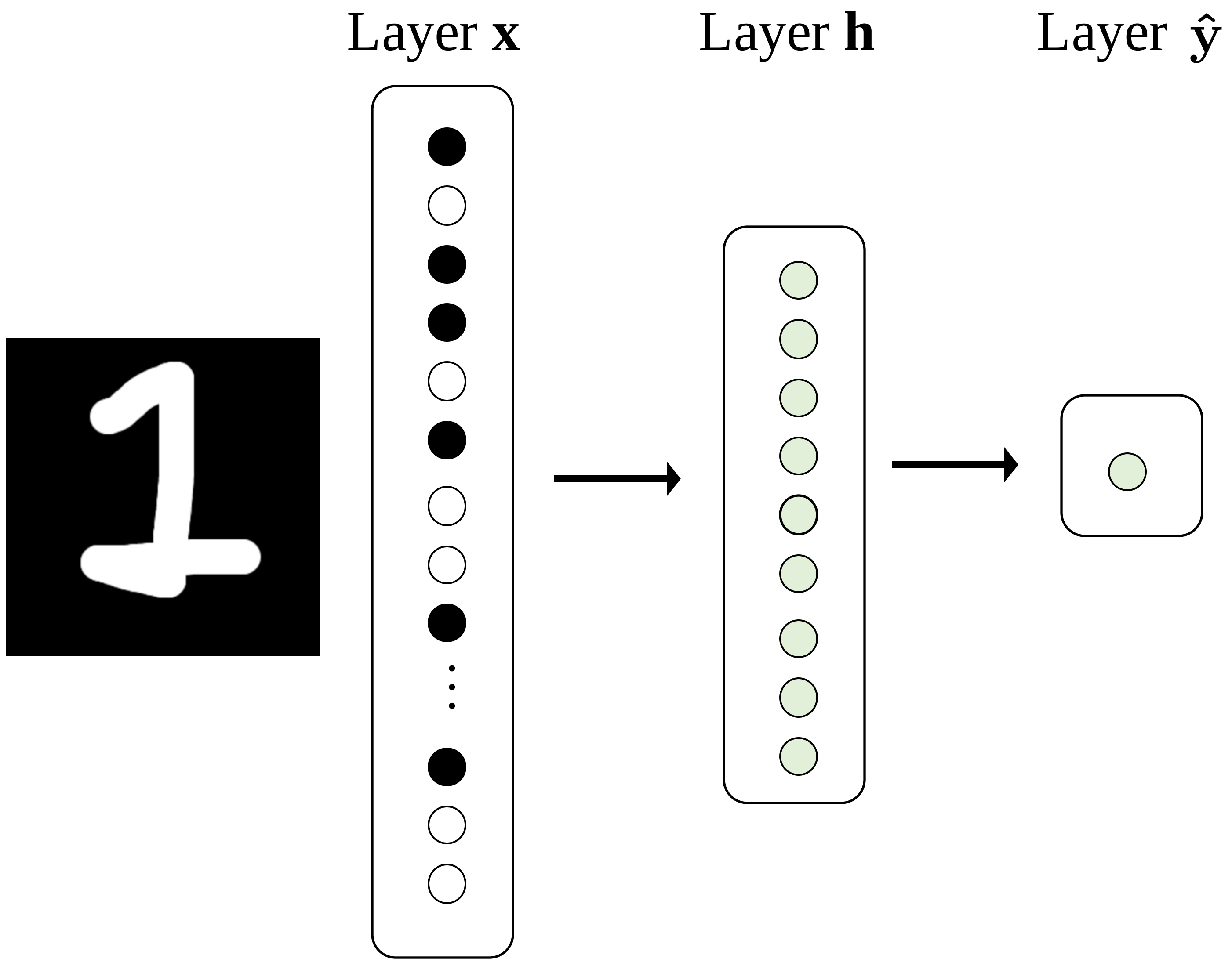


$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})}\right)$$

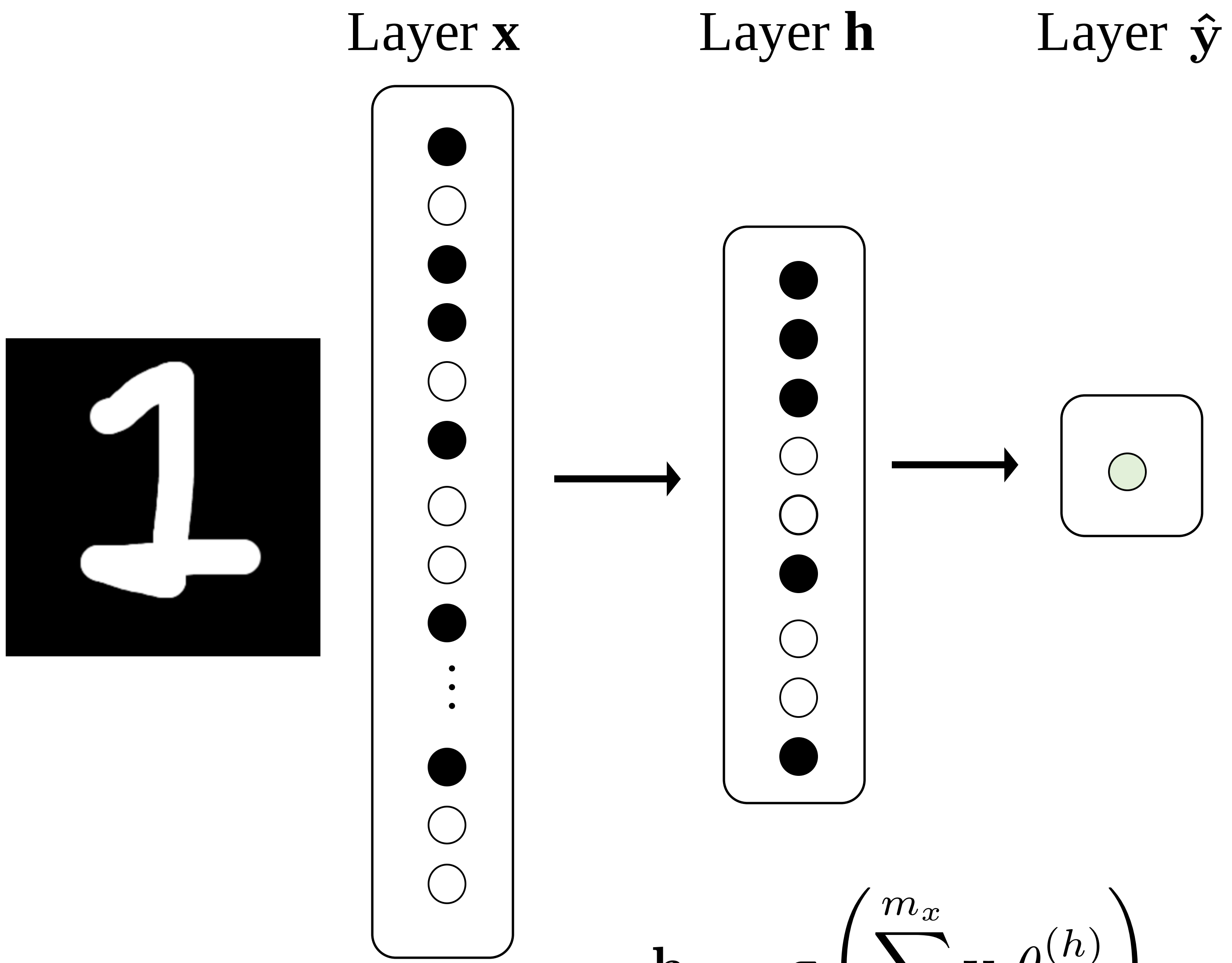
Forward Pass



Forward Pass

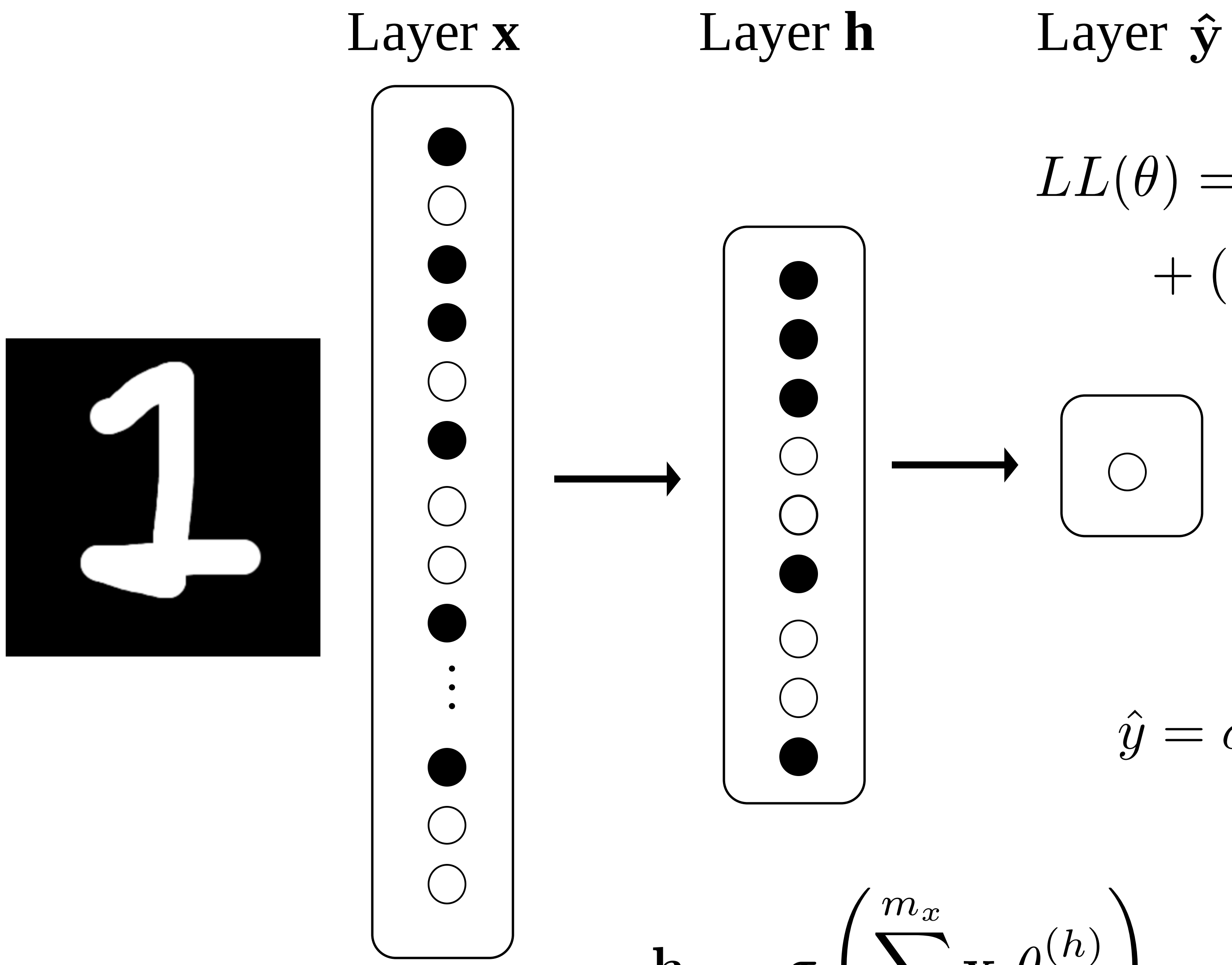


Forward Pass



$$\mathbf{h}_j = \sigma\left(\sum_{i=0}^{m_x} \mathbf{x}_i \theta_{i,j}^{(h)}\right)$$

Forward Pass

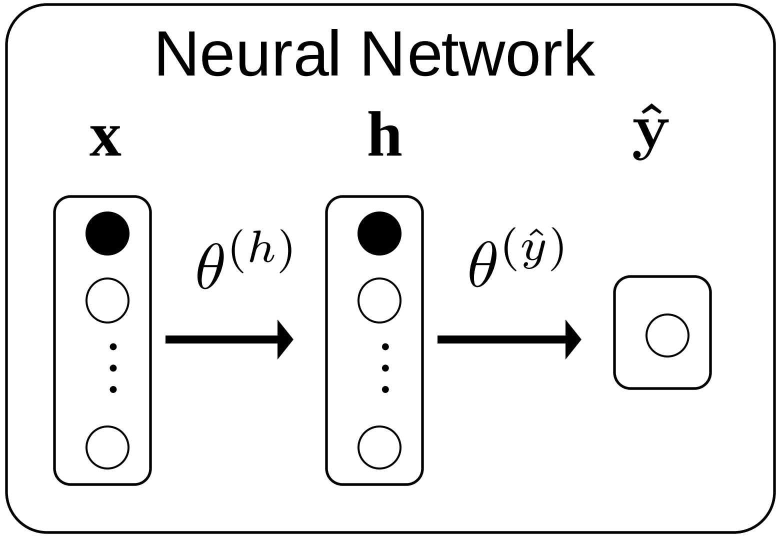


$$LL(\theta) = y \log \hat{y} + (1 - y) \log[1 - \hat{y}]$$

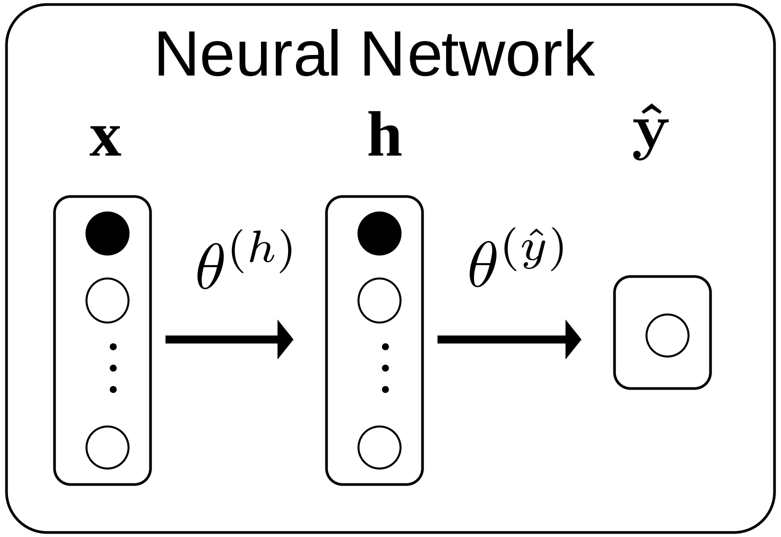
$$\hat{y} = \sigma \left(\sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})} \right)$$

$$\mathbf{h}_j = \sigma \left(\sum_{i=0}^{m_x} \mathbf{x}_i \theta_{i,j}^{(h)} \right)$$

All Together



Smoke Check 1



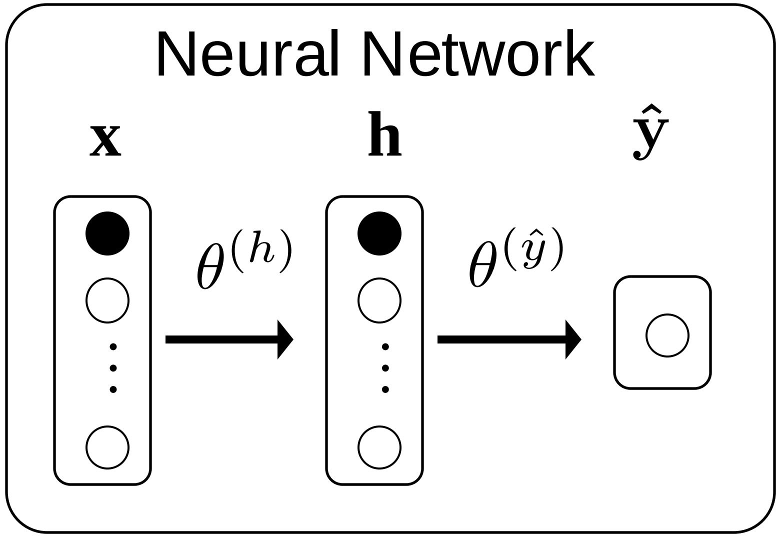
$|\mathbf{x}| = 40$

$|\mathbf{h}| = 20$

How many parameters in $\theta^{(\hat{y})}$?

a) 2 b) 20 c) 40 d) 800

Smoke Check 2



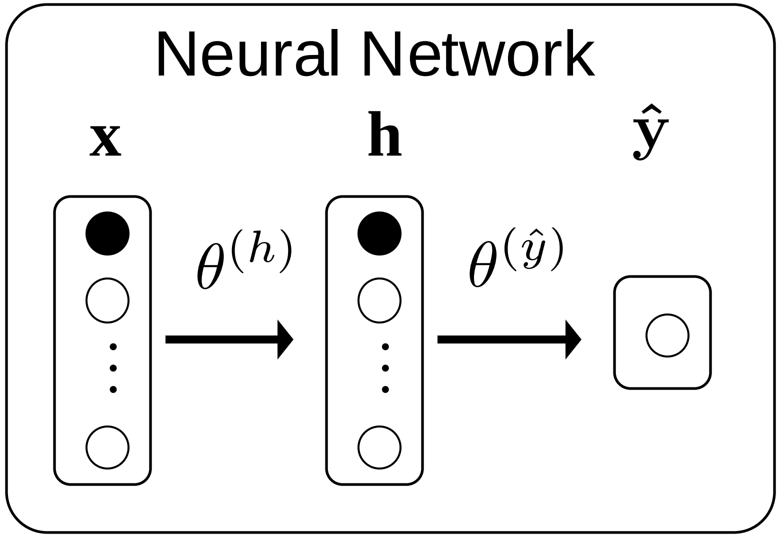
$|\mathbf{x}| = 40$

$|\mathbf{h}| = 20$

How many parameters in $\theta^{(h)}$?

a) 2 b) 20 c) 40 d) 800

Smoke Check 3



$|\mathbf{x}| = 40$

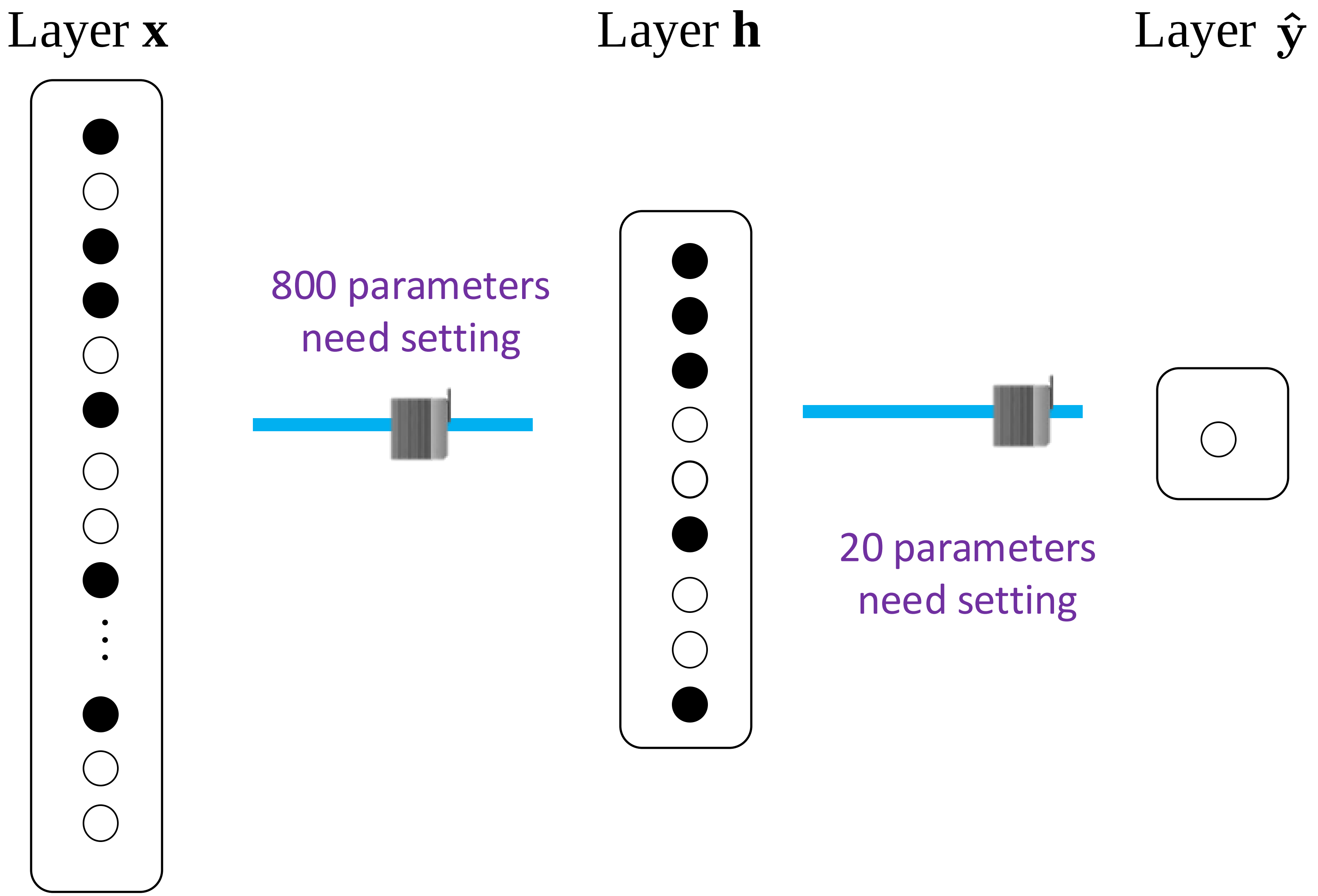
$|\mathbf{h}| = 20$

How many parameters in total?

a) 800 b) 20 c) 820 d) 16000

Today: Do Something Brave

Forward Pass



Only Have to Do Three Things

1. Make deep learning assumption

2. Calculate the log probability for all data

3. Get partial derivative of log likelihood with respect to each theta

Smoke Check

3 Get partial derivative of log likelihood with respect to each theta

Why?

Why We Calculate Partial Derivatives

A deep learning model gets its **intelligence** by having **useful thetas**.

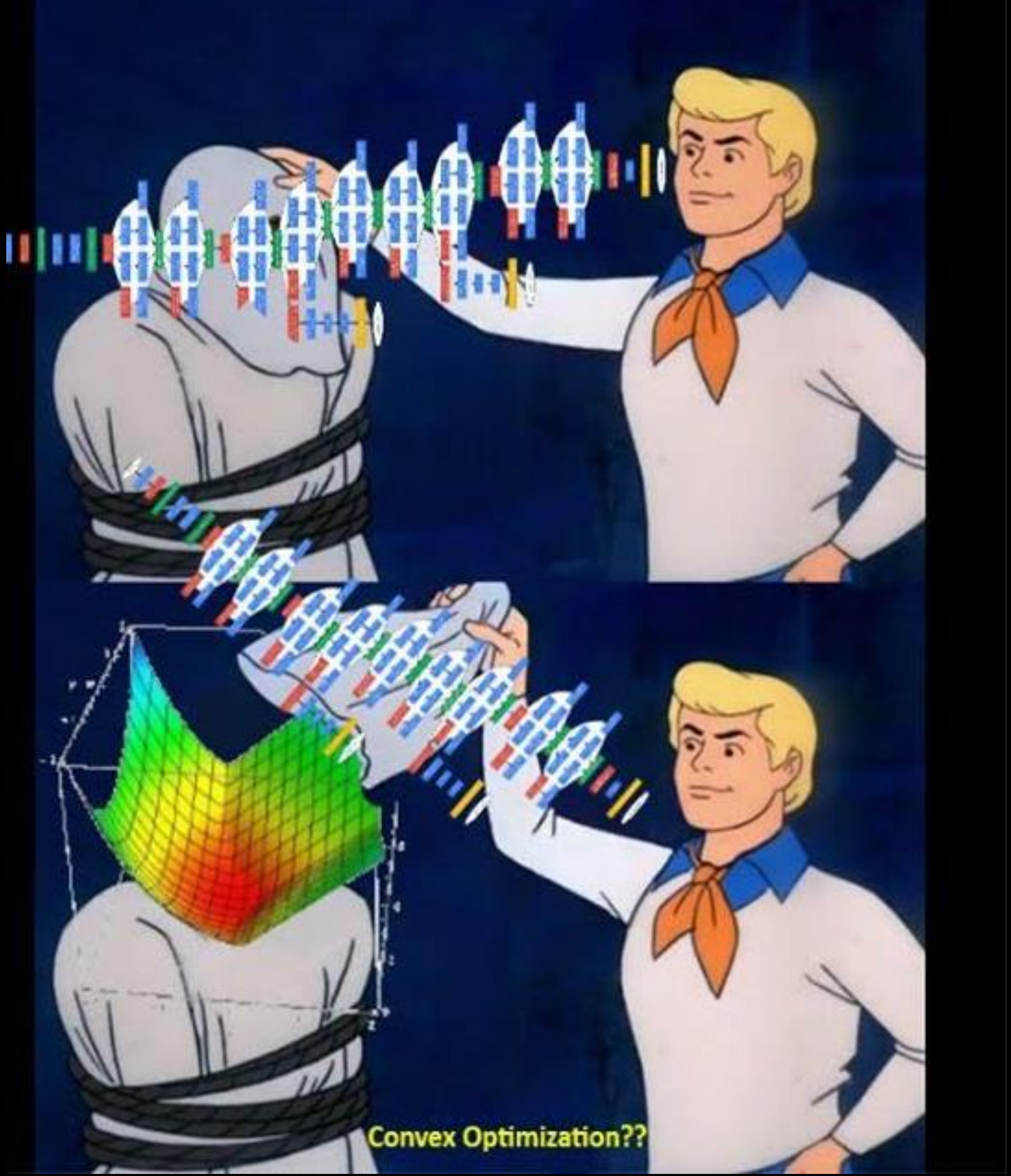
We can find **useful thetas**, by searching for ones that **maximize likelihood** of our training data

We can **maximize likelihood** using **optimization techniques** (such as gradient ascent).

In order to use **optimization techniques**, we need to calculate the **partial derivative** of likelihood with respect to thetas.

Basically MLE is hard because it has so many details

Thanks to Keith Eicher


Convex Optimization??

Only Have to Do Three Things

1. Make deep learning assumption

$$P(Y = 1|X = \mathbf{x}) = \hat{y}$$
$$P(Y = 0|X = \mathbf{x}) = 1 - \hat{y}$$

2. Calculate the log probability for all data

Same Assumption, Same LL

$$P(Y=1|X=\mathbf{x}) = \hat{y}$$

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})}\right)$$

$$\mathbf{h}_j = \sigma\left(\sum_{i=0}^{m_x} \mathbf{x}_i \theta_{i,j}^{(h)}\right)$$

For one datum

$$P(Y=y|\mathbf{X}=\mathbf{x}) = (\hat{y})^y (1-\hat{y})^{1-y}$$

Feel the Bern!

$$Y \sim \text{Bern}(\hat{y})$$

For IID data

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

$$= \prod_{i=1}^{n} (\hat{y}^{(i)})^{y^{(i)}} \cdot \left[1-(\hat{y}^{(i)})\right]^{(1-y^{(i)})}$$

Take the log

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

Only Have to Do Three Things

1. Make deep learning assumption

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})}\right)$$

$$P(Y = 1 | X = \mathbf{x}) = \hat{y}$$

$$P(Y = 0 | X = \mathbf{x}) = 1 - \hat{y}$$

2. Calculate the log probability for all data

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log[1 - \hat{y}^{(i)}]$$

3. Get partial derivative of log likelihood with respect to each theta

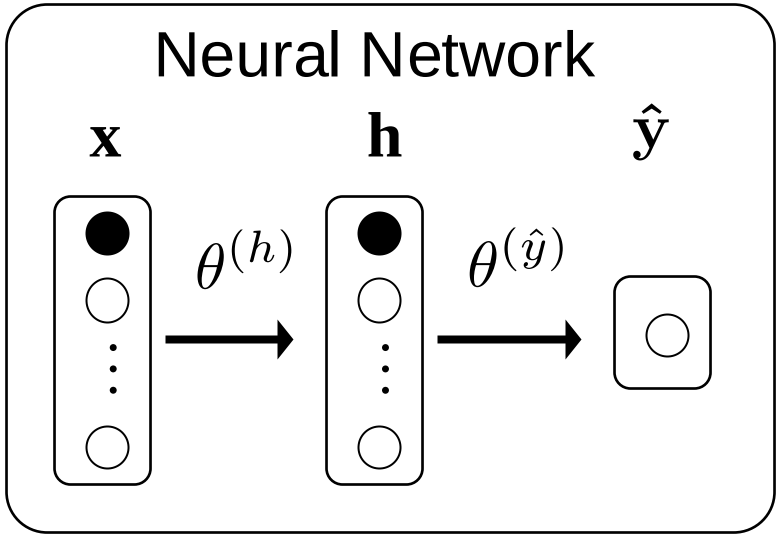
Derivative Goals

Loss with respect to output layer params

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}}$$

Loss with respect to hidden layer params

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$



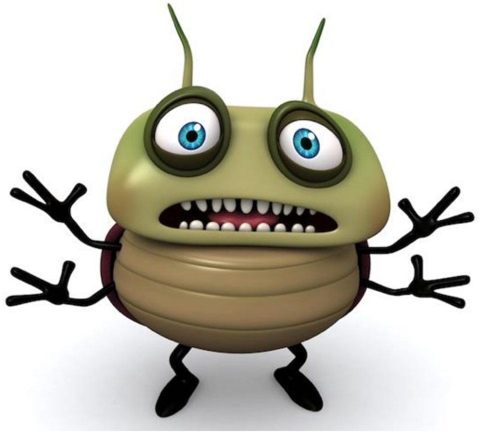
Bad Approach

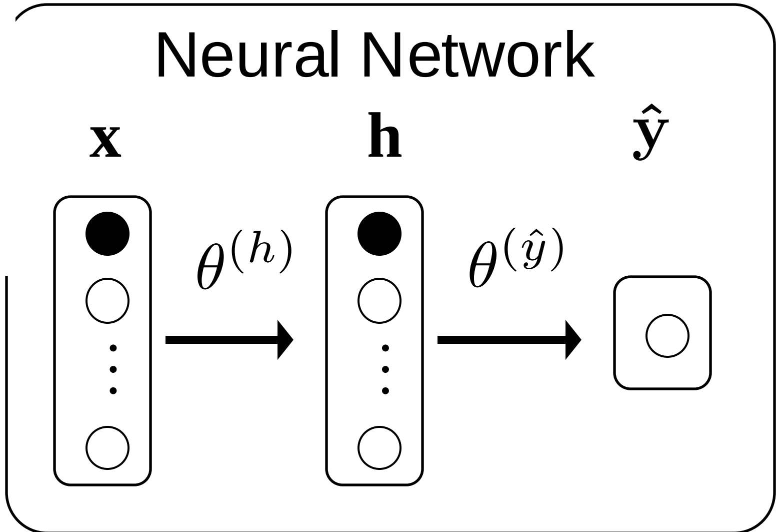
$$LL(\theta) = y \log \hat{y} + (1 - y) \log[1 - \hat{y}]$$

$$\hat{y} = \sigma\left(\sum_{i=0}^{m_h} \mathbf{h}_i \theta_i^{(\hat{\mathbf{y}})}\right)$$

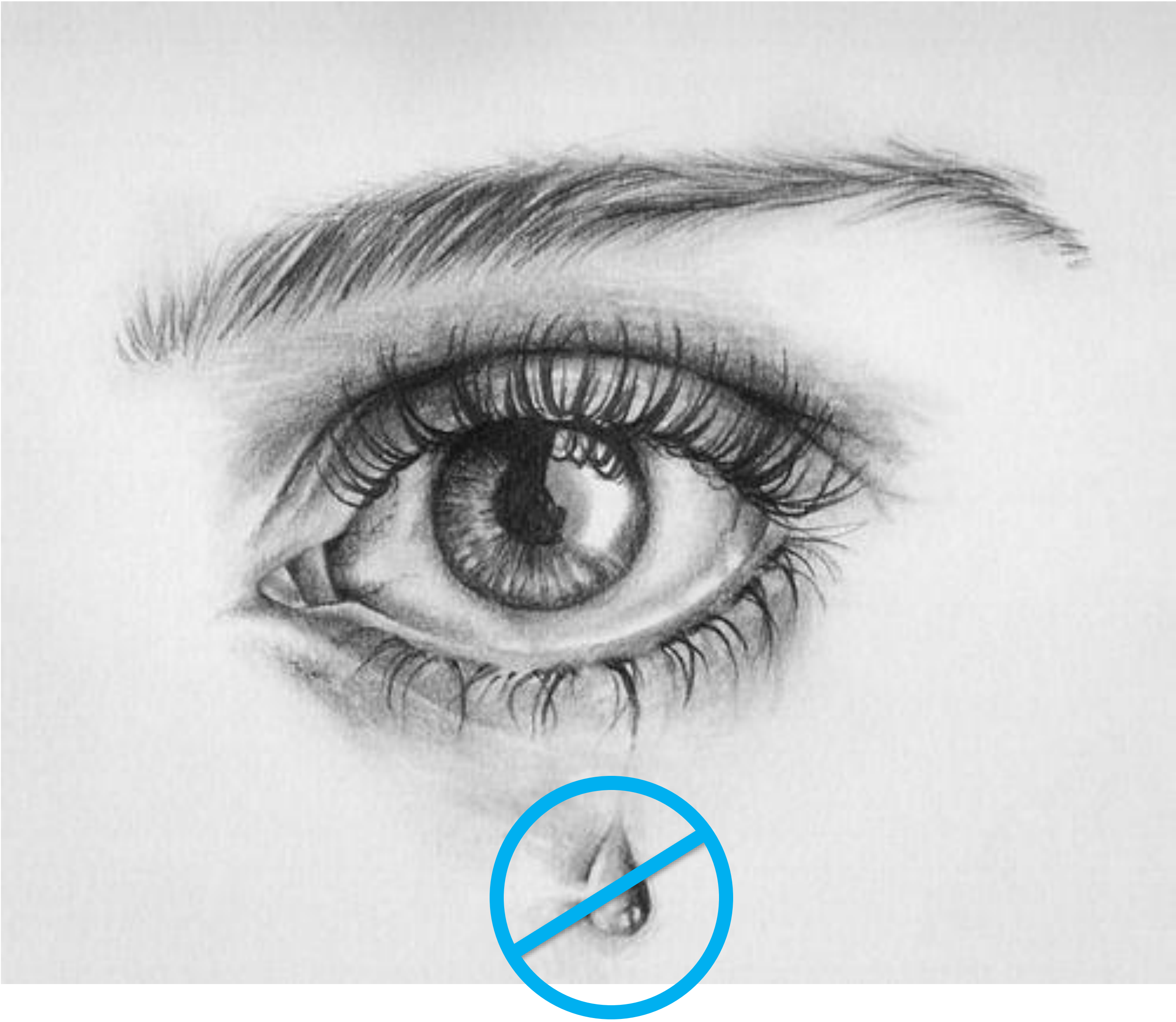
Math bug

$$= \sigma\left(\sum_{i=0}^{m_h} \left[\sigma\left(\sum_{i=0}^{m_x} \mathbf{x}_i \theta_{i,j}^{(\mathbf{h})}\right)\right] \theta_i^{(\hat{\mathbf{y}})}\right)$$





Derivatives Without Tears



Big Idea #1: Chain Rule

Woah Mr Blanton, you were right.
Chain rule is useful!

$$\frac{\partial f(z)}{\partial x} = \frac{\partial f(z)}{\partial z} \cdot \frac{\partial z}{\partial x}$$

First use:

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_i^{(\hat{y})}}$$

Big Idea #2: Sigmoid Derivative

True fact about sigmoid functions

$$\frac{\partial}{\partial z}\sigma(z) = \sigma(z)[1 - \sigma(z)]$$

Big Idea #3: Derivative of Sum

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log[1 - \hat{y}^{(i)}]$$

We only need to calculate the gradient for one training example!

$$\frac{\partial}{\partial x} \sum f(x) = \sum \frac{\partial}{\partial x} f(x)$$

We will pretend we only have one example

$$LL(\theta) = y \log \hat{y} + (1 - y) \log[1 - \hat{y}]$$

We can sum up the gradients of each example to get the correct answer

Recall

Sigmoid has a Beautiful Slope

$$\frac{\partial}{\partial \theta_j} \sigma(\theta^T x)?$$

$$\frac{\partial}{\partial z} \sigma(z) = \sigma(z)[1 - \sigma(z)]$$

where $z = \theta^T x$

$$\frac{\partial}{\partial \theta_j} \sigma(\theta^T x) = \frac{\partial}{\partial z} \sigma(z) \cdot \frac{\partial z}{\partial \theta_j}$$

Chain rule!

$$\frac{\partial}{\partial \theta_j} \sigma(\theta^T x) = \sigma(\theta^T x)[1 - \sigma(\theta^T x)] x_j$$

Plug and chug

Sigmoid, you should be a ski hill

This is ~~Sparta~~!!!!!

Stanford

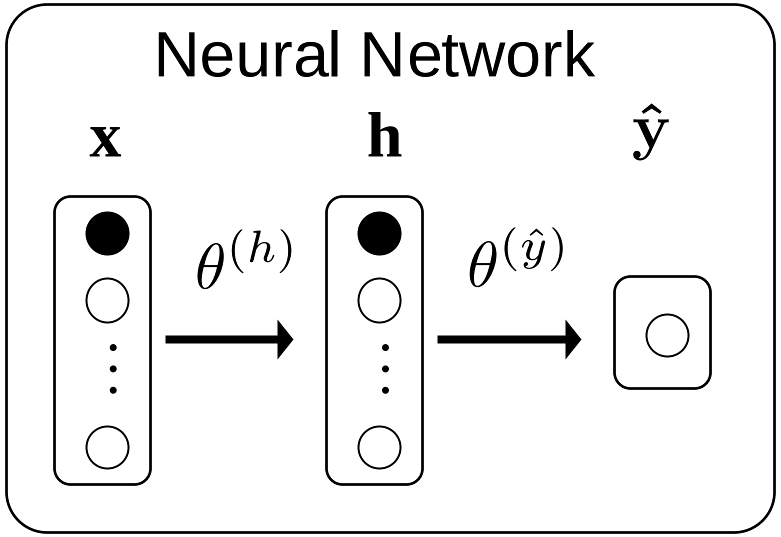
Derivative Goals

Loss with respect to output layer params

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}}$$

Loss with respect to hidden layer params

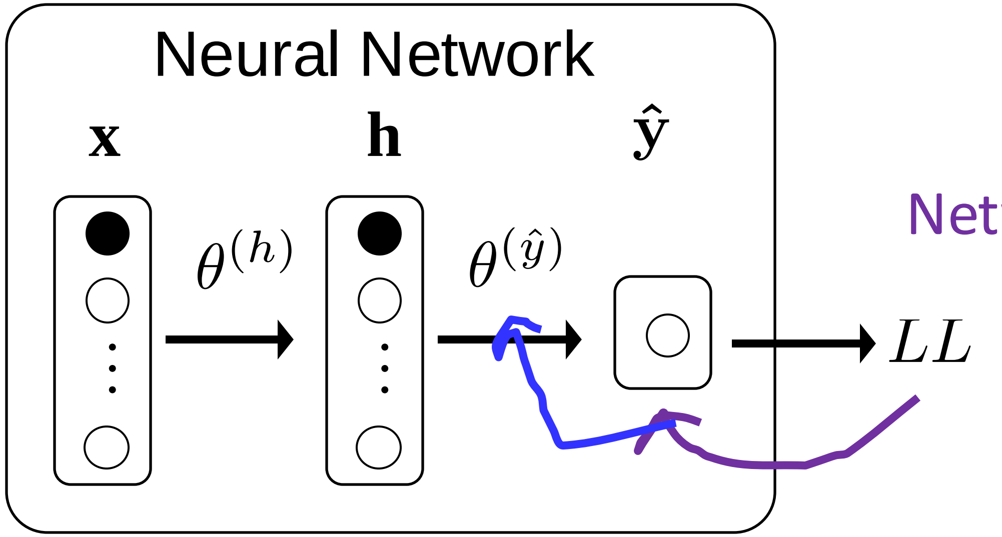
$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$



Chain Rule Example 1

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}}$$

Goal



Network

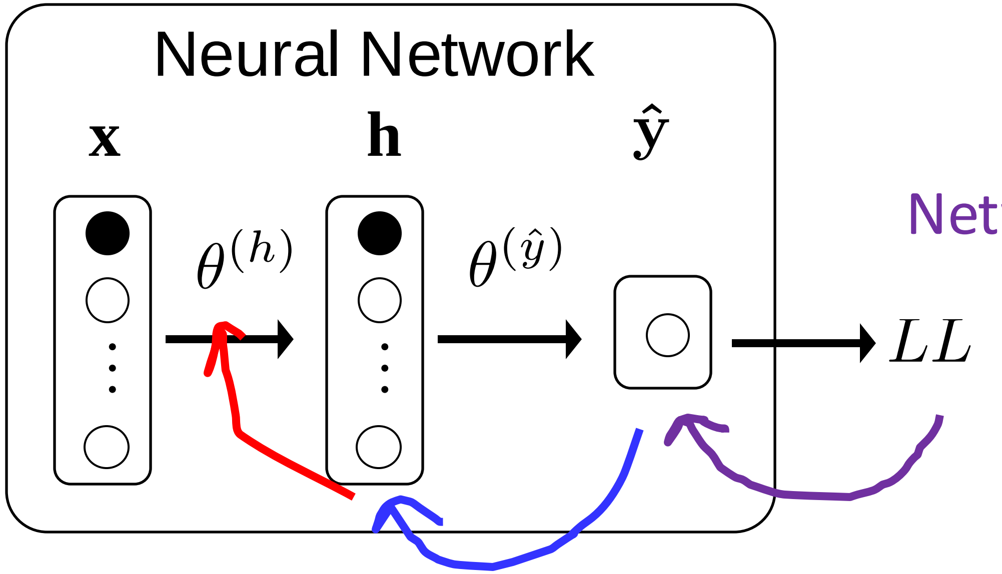
$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_i^{(\hat{y})}}$$

Decomposition

Chain Rule Example 2

$$\frac{\partial LL(\theta)}{\partial \theta^{(h)}_{i,j}}$$

Goal



Network

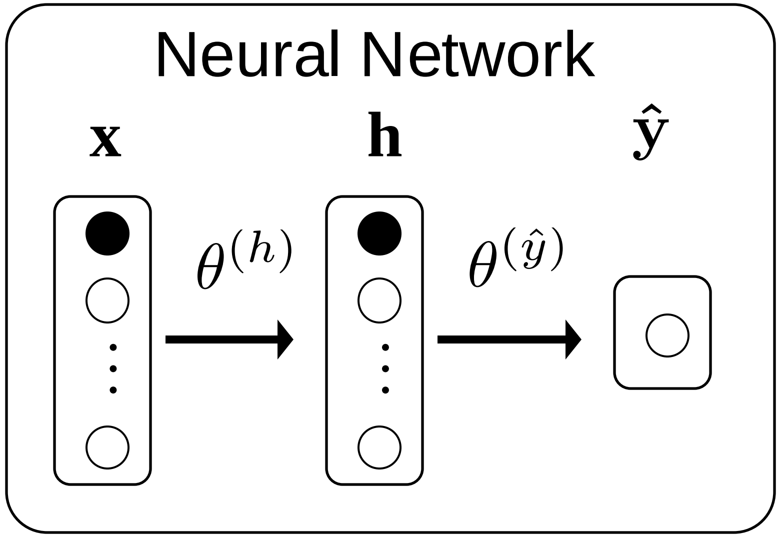
$$\frac{\partial LL(\theta)}{\partial \theta^{(h)}_{i,j}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{h}_j} \cdot \frac{\partial \mathbf{h}_j}{\partial \theta^{(h)}_{i,j}}$$

Decomposition

Decomposition

Gradient of output layer params

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_i^{(\hat{y})}}$$



Gradient of output layer params

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_i^{(\hat{y})}}$$

$$LL(\theta) = y \log \hat{y} + (1-y) \log[1-\hat{y}]$$

$$\frac{\partial LL(\theta)}{\partial \hat{y}} = \frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial (1-\hat{y})}{\partial \hat{y}}$$

$$\frac{\partial LL(\theta)}{\partial \hat{y}} = \frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})}$$

Gradient of output layer params

$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_i^{(\hat{y})}}$$

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})}\right) = \sigma(z) \quad \text{where} \quad z = \sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial \theta_i^{(\hat{y})}} = \hat{y}[1-\hat{y}] \cdot \frac{\partial}{\partial \theta_i^{(\hat{y})}} \sum_{j=0}^{m_h} \mathbf{h}_j \theta_j^{(\hat{y})}$$

$$= \hat{y}[1-\hat{y}] \cdot h_i$$

What! That's not scary!

Make it Simple

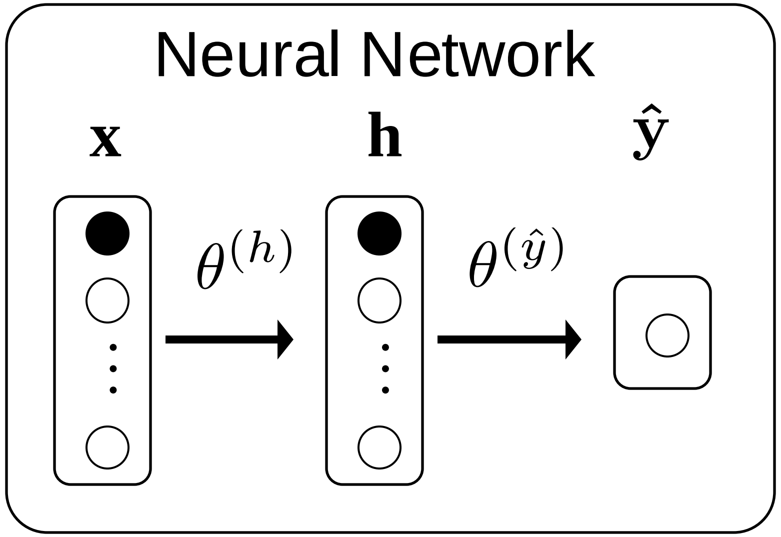
$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}} =$$

$$= \frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})}$$

$$= \hat{y}[1-\hat{y}] \cdot h_i$$

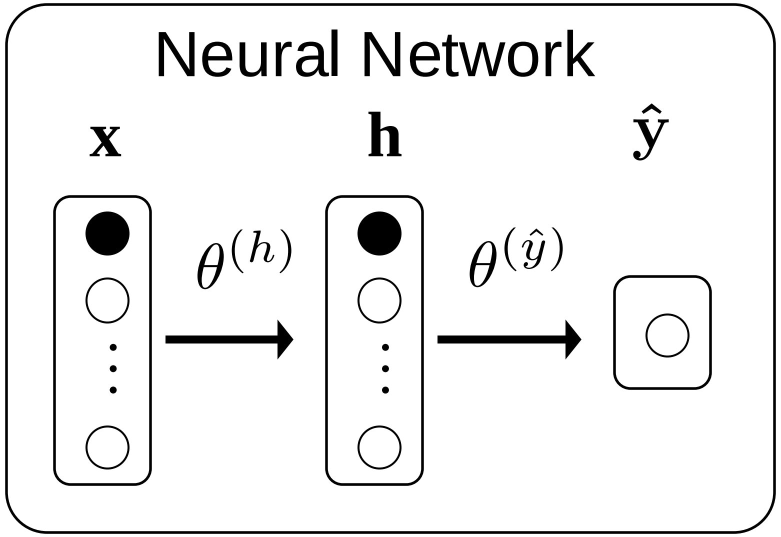
Boom!

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$



Gradient of hidden layer params

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{h}_j} \cdot \frac{\partial \mathbf{h}_j}{\partial \theta_{i,j}^{(h)}}$$



Gradient of hidden layer params

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{h}_j} \cdot \frac{\partial \mathbf{h}_j}{\partial \theta_{i,j}^{(h)}}$$

$$\hat{y} = \sigma\left(\sum_{i=0}^{m_h} \mathbf{h}_i \theta_i^{(\hat{y})}\right)$$

$$\frac{\partial \hat{y}}{\partial \mathbf{h}_j} = \hat{y}[1 - \hat{y}]\theta_j^{(\hat{y})}$$

Wait is it over?

Gradient of hidden layer params

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \mathbf{h}_j} \cdot \frac{\partial \mathbf{h}_j}{\partial \theta_{i,j}^{(h)}}$$

$$\mathbf{h}_j = \sigma\left(\sum_{k=0}^{m_x} \mathbf{x}_k \theta_{k,j}\right)$$

$$\frac{\partial \mathbf{h}_j}{\partial \theta_{i,j}^{(h)}} = \mathbf{h}_j[1 - \mathbf{h}_j]\mathbf{x}_i$$

That one too?

Make it Simple

$$\frac{\partial LL(\theta)}{\partial \theta^{(h)}_{i,j}} =$$   

 $$= \frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})}$$

 $$= \hat{y}[1-\hat{y}]\theta_j^{(\hat{y})}$$

 $$= \mathbf{h}_j[1-\mathbf{h}_j]\mathbf{x}_j$$

Congrats. You now know Backpropagation

Moment of silence

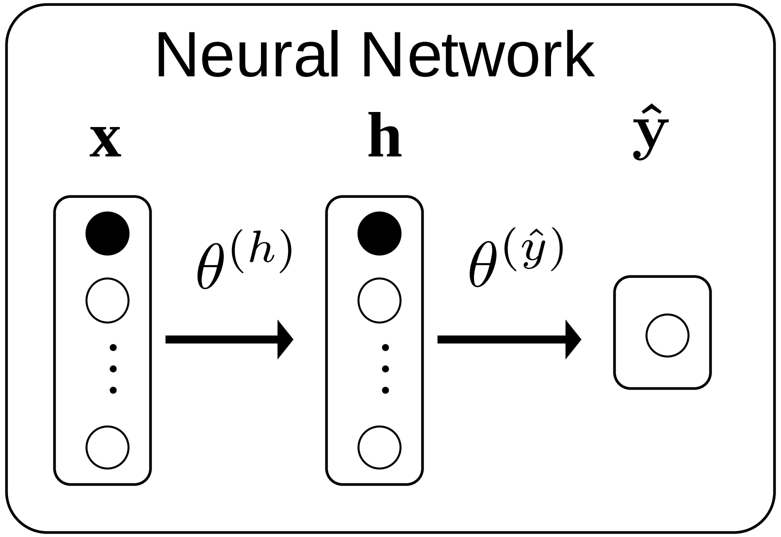
Summary: Simple Calculations For

Loss with respect to output layer params

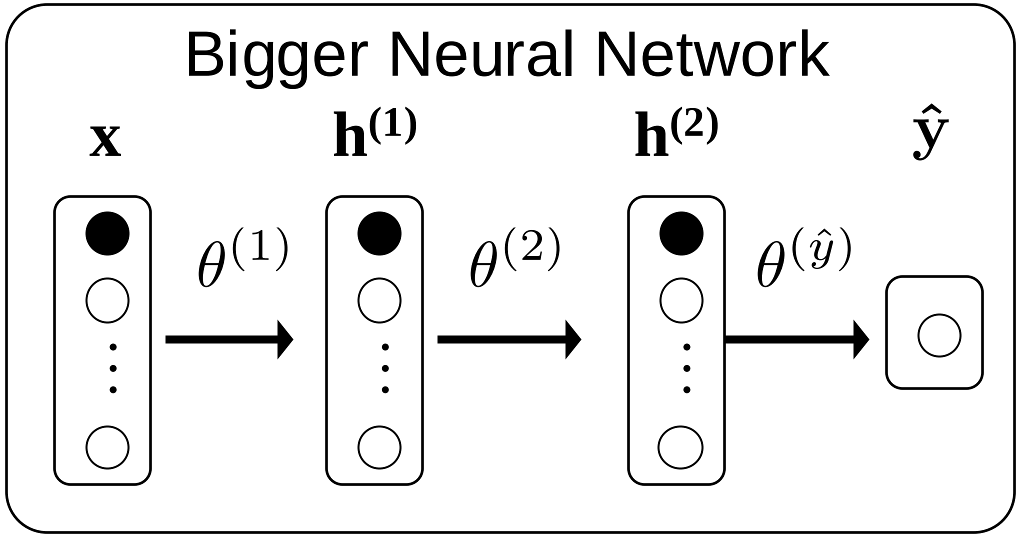
$$\frac{\partial LL(\theta)}{\partial \theta_i^{(\hat{y})}}$$

Loss with respect to hidden layer params

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$



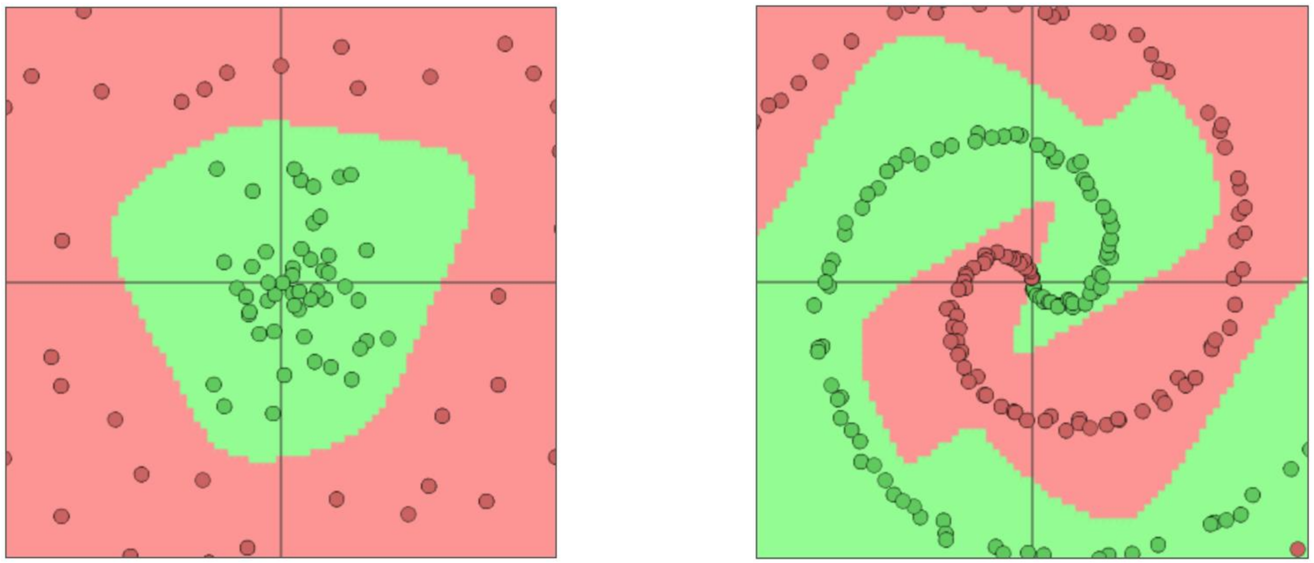
What Would You Do Here?



Chain rule:
Game changer for
artificial intelligence

Neural Networks Can Learn Complex Functions

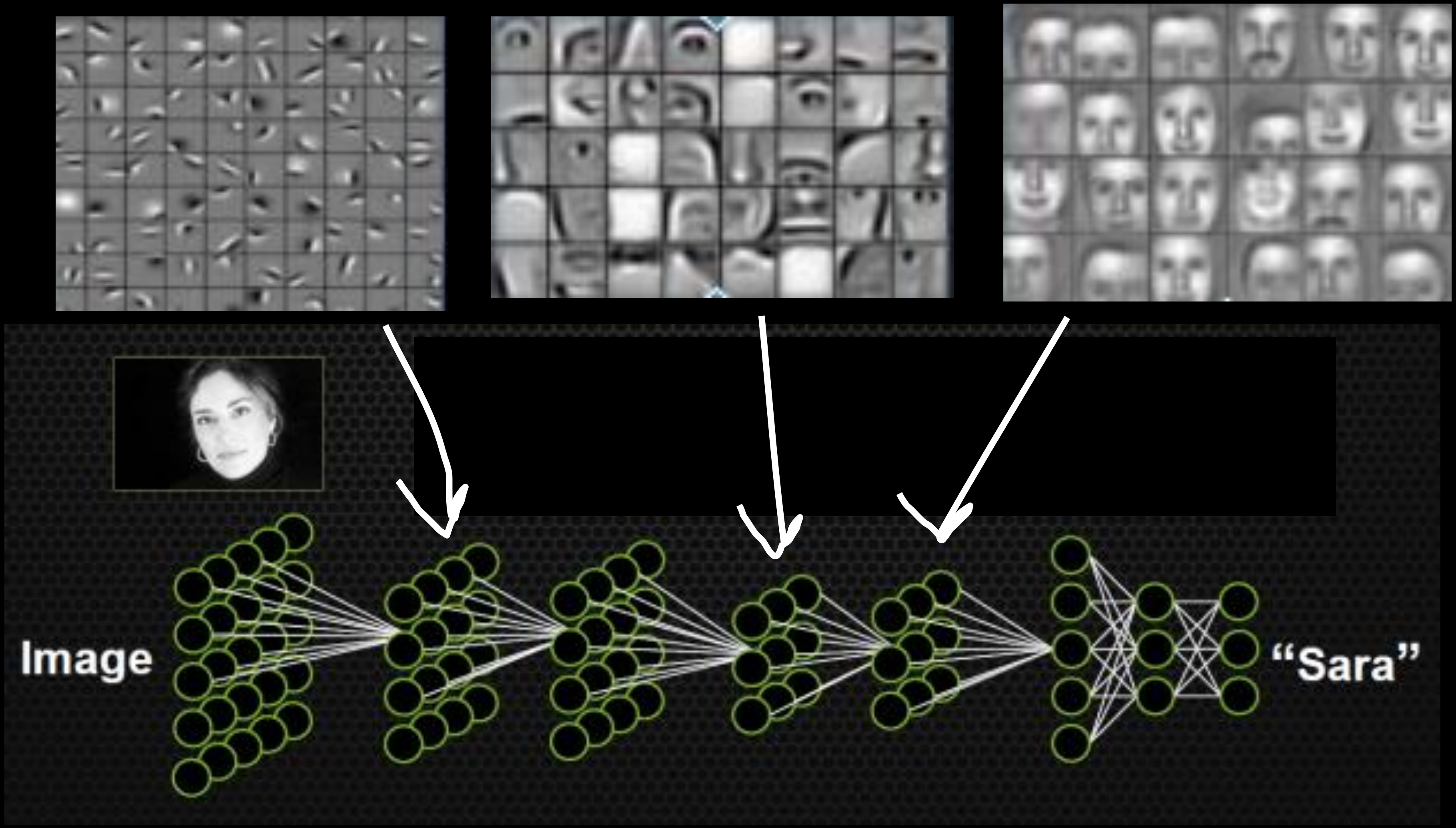
- Some data sets/functions are not separable



  - These are classifiers learned by neural networks

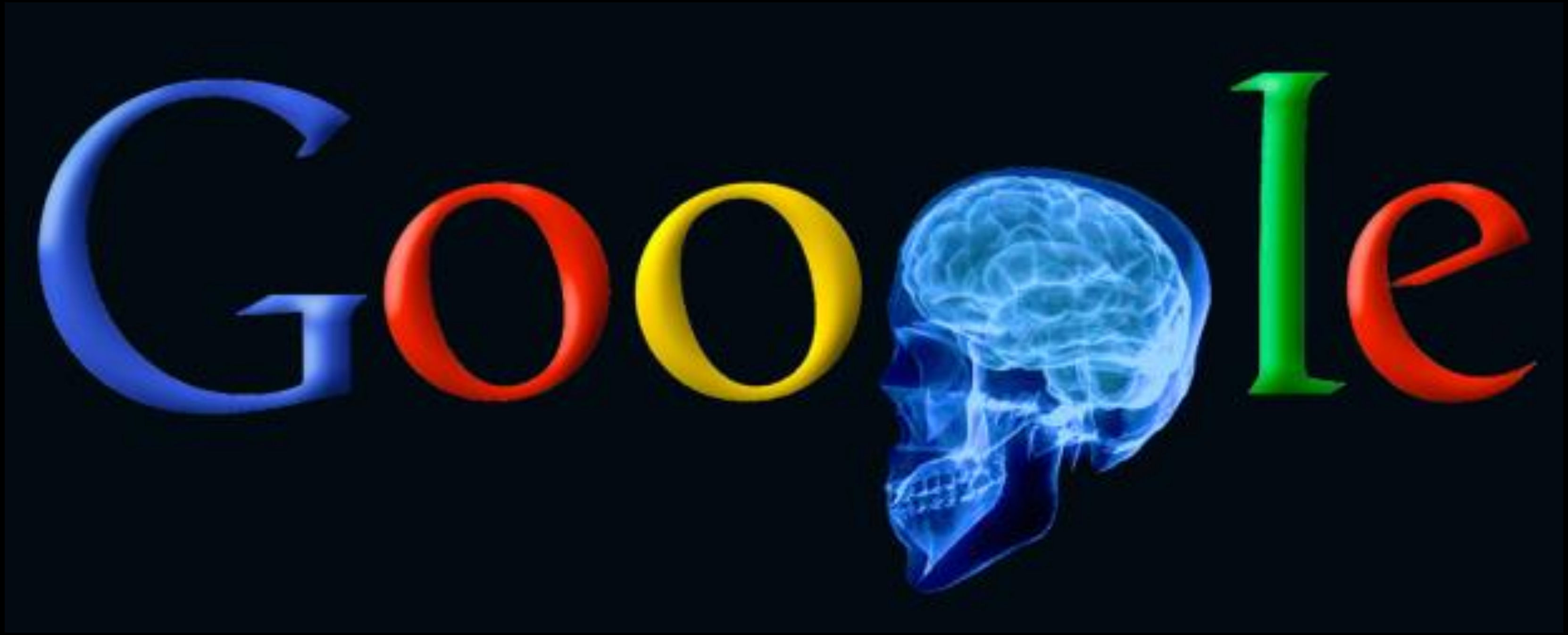
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Works for any number of layers



x a b c d e f g LL

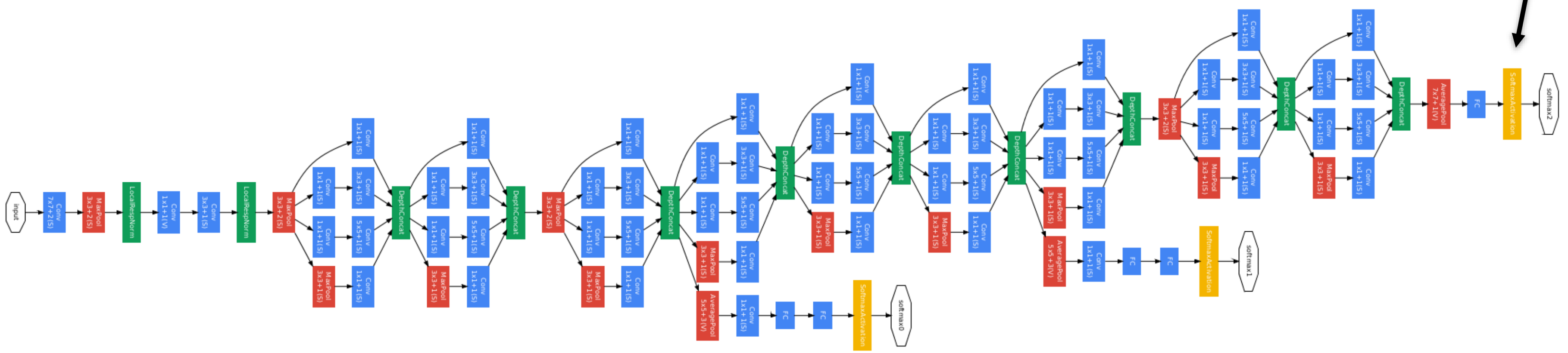
GoogLeNet Brain



1 Trillion Artificial Neurons

GoogLeNet Brain

Multiple,
Multi class output



22 layers deep

The Cat Neuron

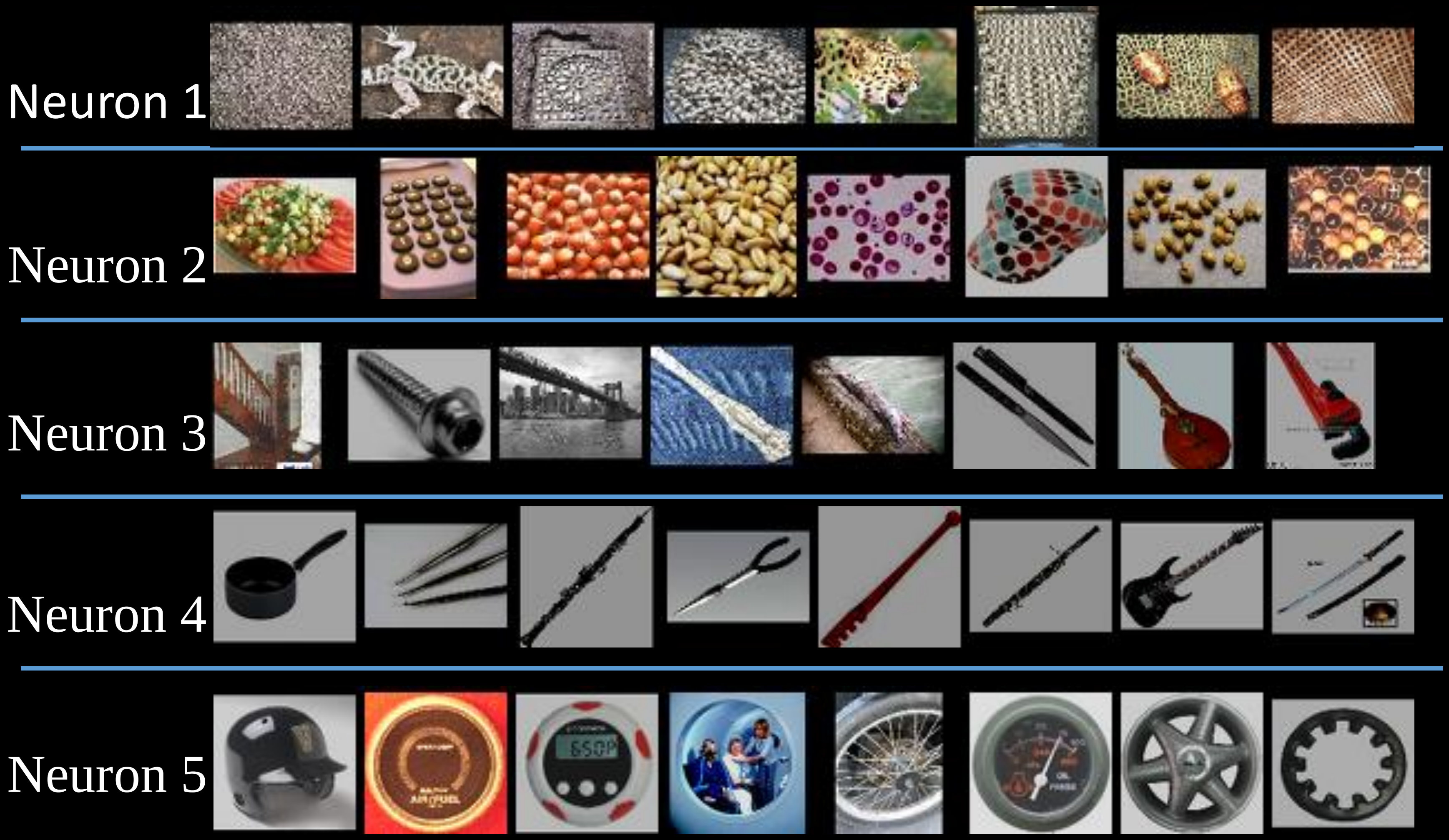
Top stimuli from the test set

Optimal stimulus
by numerical optimization

Le, et al., *Building high-level features using large-scale unsupervised*

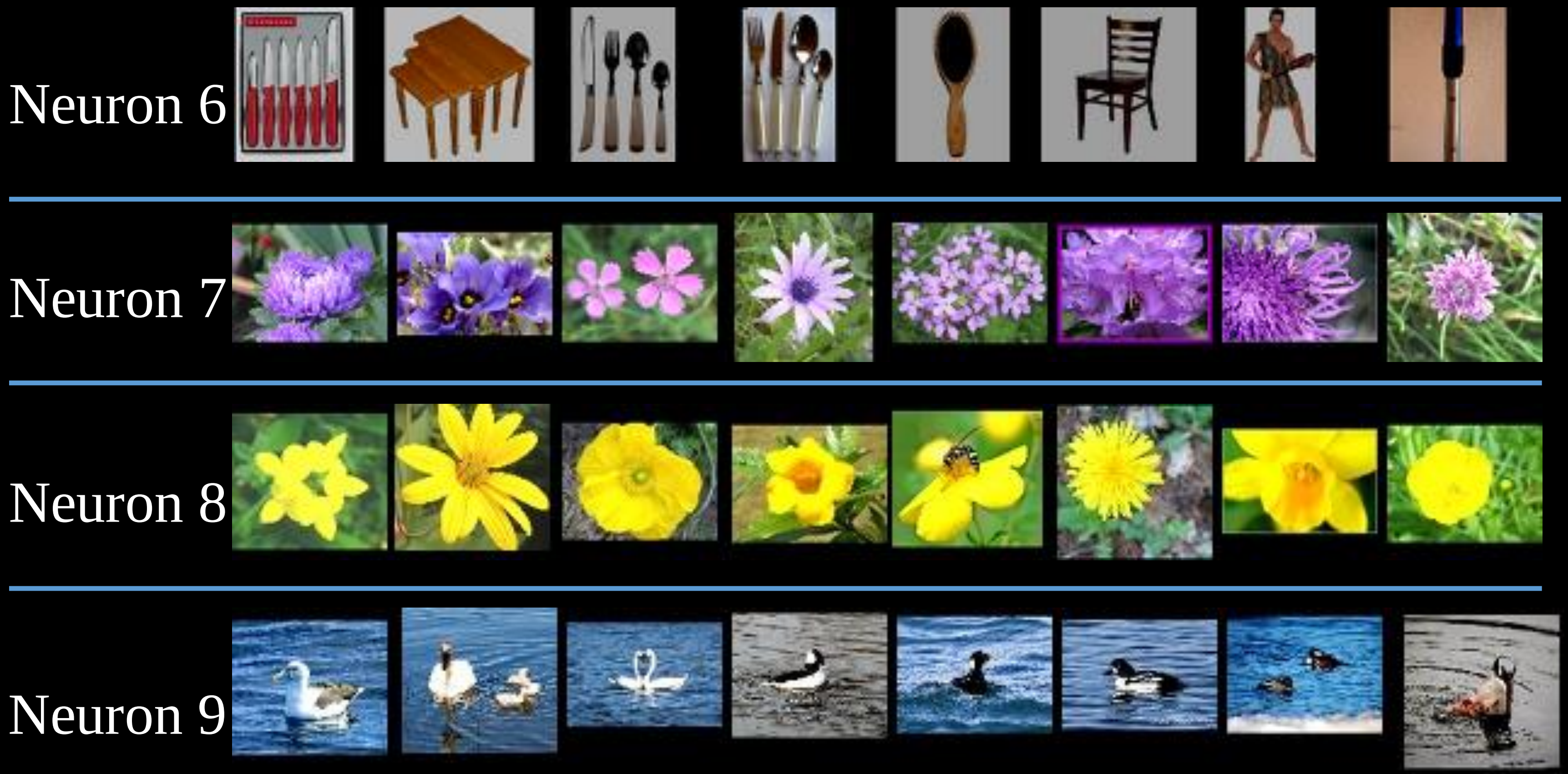

Hire the smartest people in the world
Invent cat detector

Best Neuron Stimuli



Le, et al., *Building high-level features using large-scale unsupervised*

Best Neuron Stimuli



Le, et al., *Building high-level features using large-scale unsupervised*

Best Neuron Stimuli

Le, et al., *Building high-level features using large-scale unsupervised*

ImageNet Classification

22,000 categories

14,000,000 images

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

Le, et al., *Building high-level features using large-scale unsupervised*

22,000 is a lot!

…
smoothhound, smoothhound shark, Mustelus mustelus
American smooth dogfish, Mustelus canis
Florida smoothhound, Mustelus norrisi
whitetip shark, reef whitetip shark, Triaenodon obseus
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
…

0.005%	1.5%	?
Random guess	Pre Neural Networks	GoogLeNet

Le, et al., *Building high-level features using large-scale unsupervised*

0.005% Random guess

1.5% Pre Neural Networks

43.9% GoogLeNet

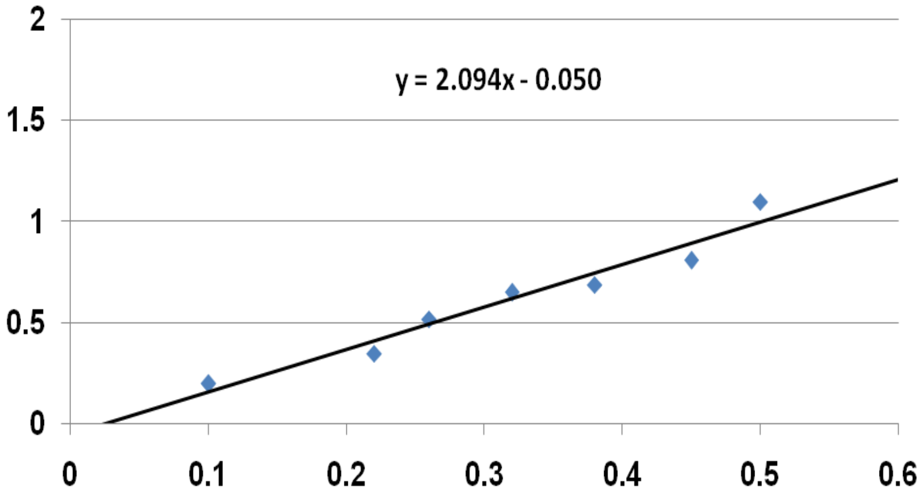
Szegedy et al, Going Deeper With Convolutions, CVPR 2015

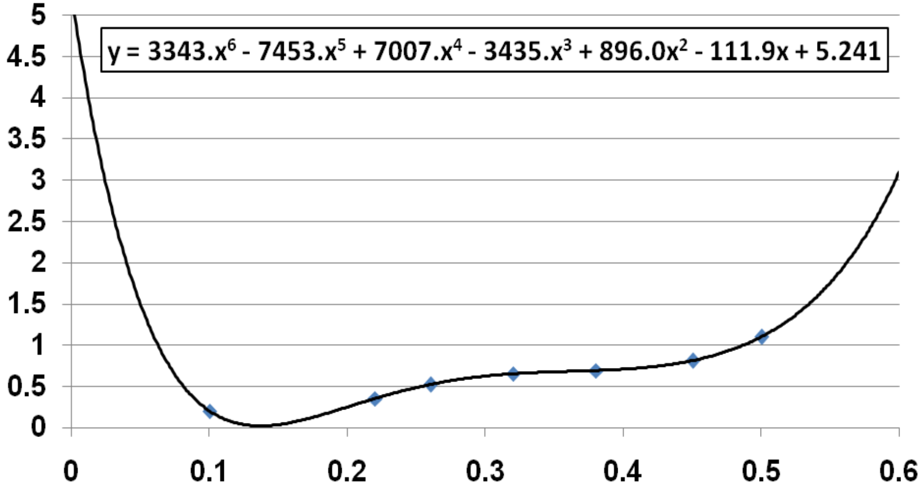
0.005%	1.5%	95.1%
Random guess	Pre Neural Networks	SE-ResNet

How many parameters
is too many?

Good ML = Generalization

- Goal of machine learning: build models that ***generalize*** well to predicting new data
  - "Overfitting": fitting the training data too well, so we lose generality of model

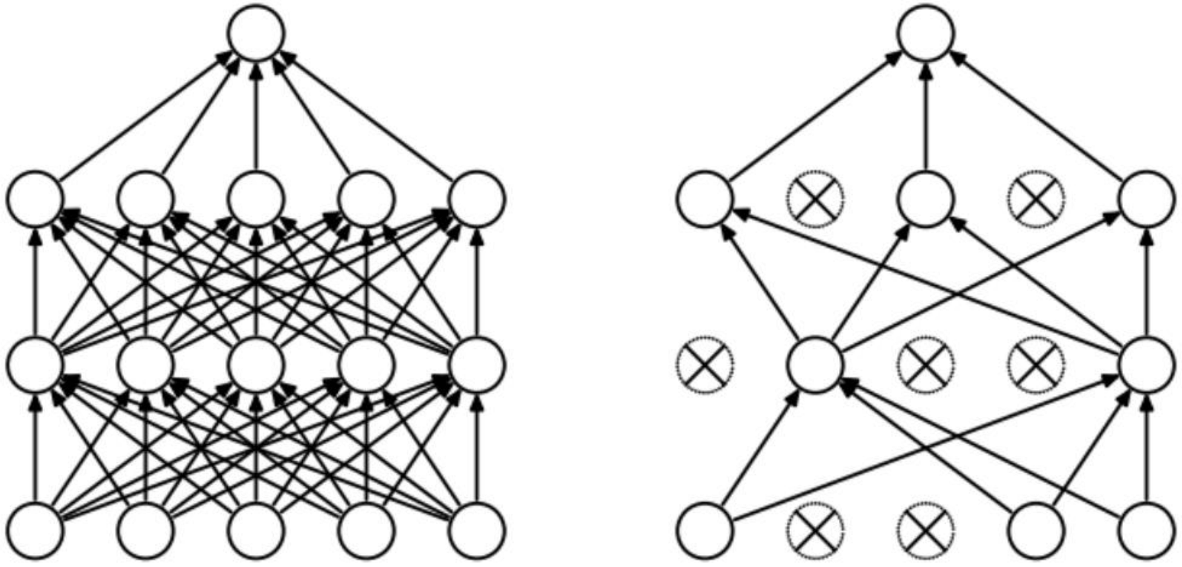




  - Polynomial on the right fits training data perfectly!
  - Which would you rather use to predict a new data point?

Prevent Overfitting?

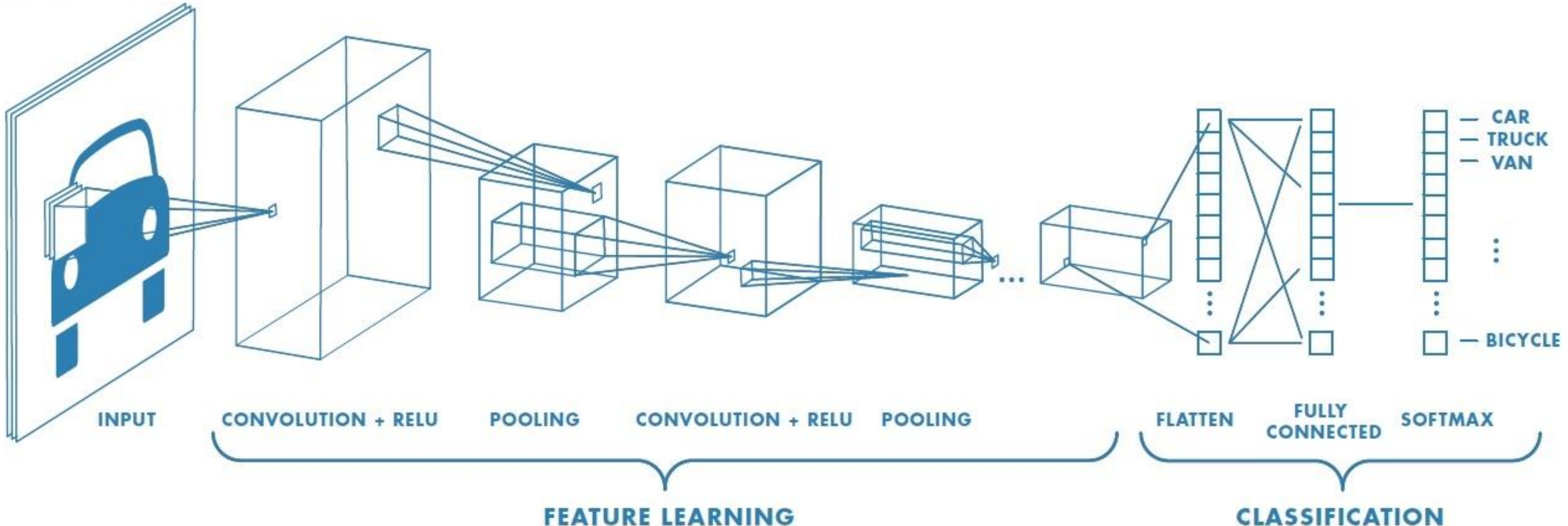
Dropout when your model is training, randomly turn off your neurons with probability 0.5. It will make your network more robust.



(a) Standard Neural Net (b) After applying dropout.

Shared Weights?

Convolution it turns out if you want to force some of your weights to be shared for different neurons, the math isn't that much harder. This is used a lot for vision (CNN).

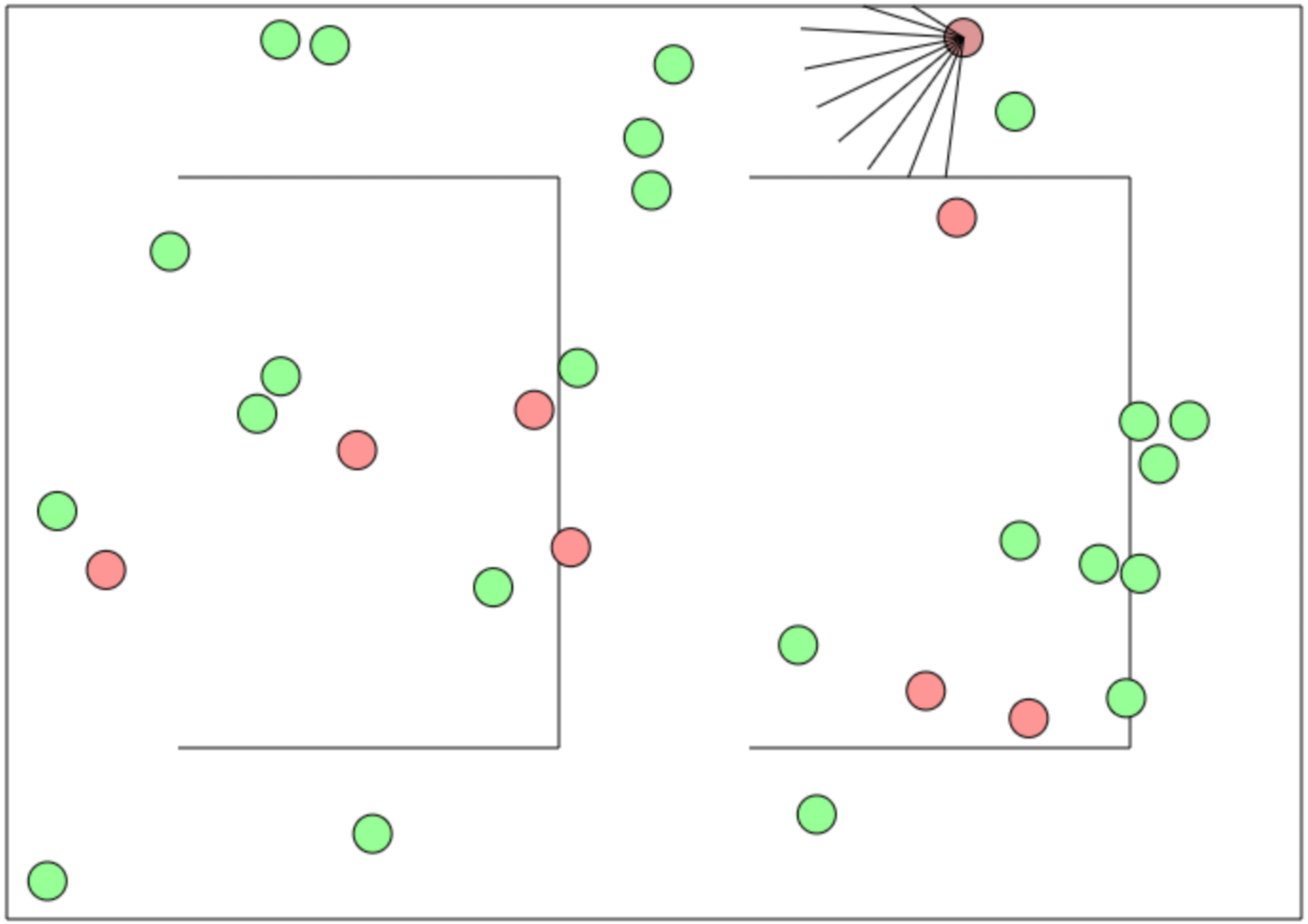


Not everything is classification

Making Decisions?

Deep Reinforcement Learning
Instead of having the output of a model be a probability you can make it an expectation.

Deep Reinforcement Learning

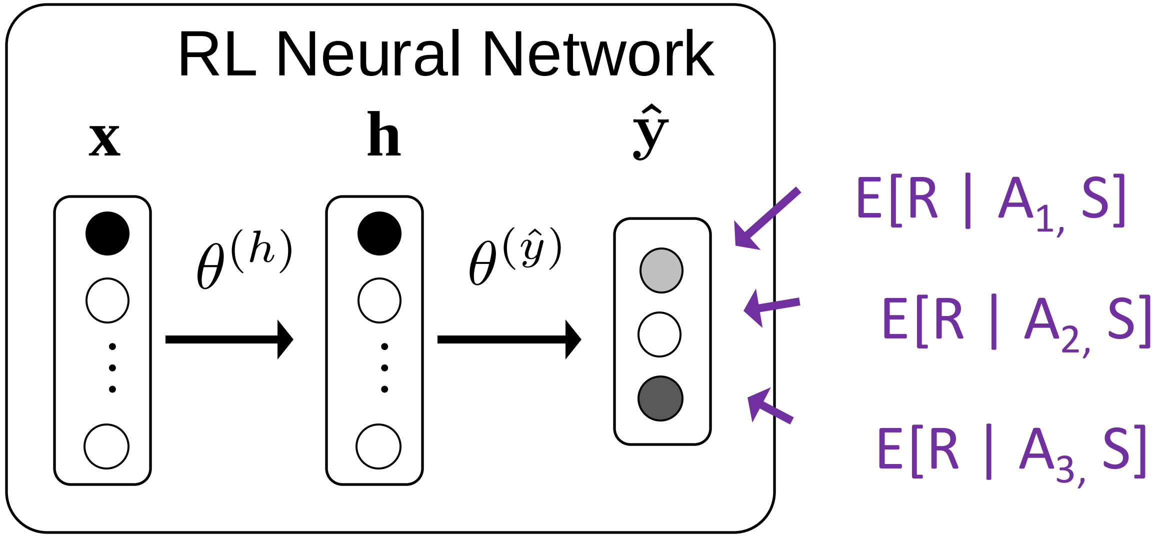


http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

Deep Reinforcement Learning

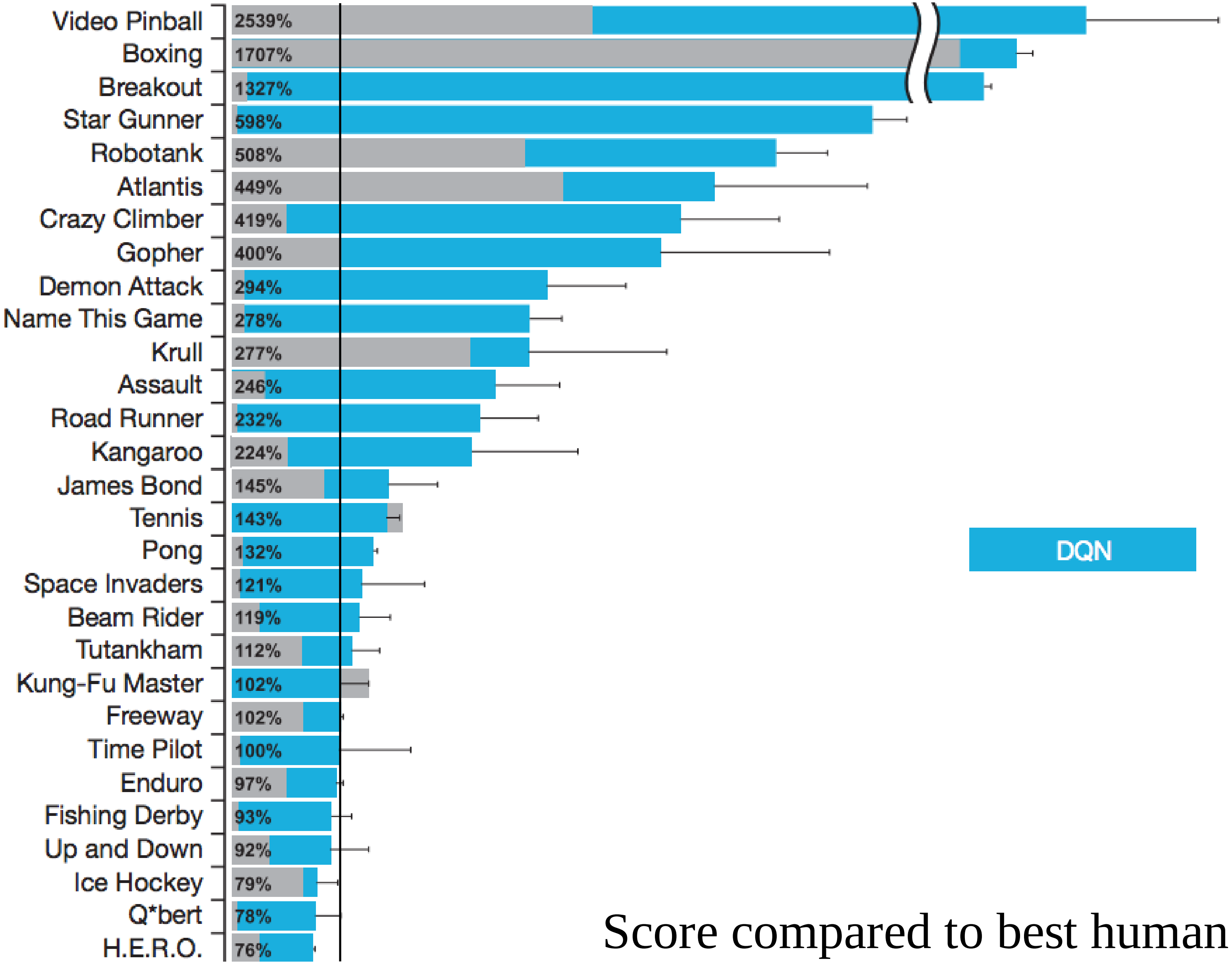
R is a reward and A_i is a legal action

Input is a representation of current state (S)



Interpret outputs as expected reward for a given action

Deep Mind Atari Games



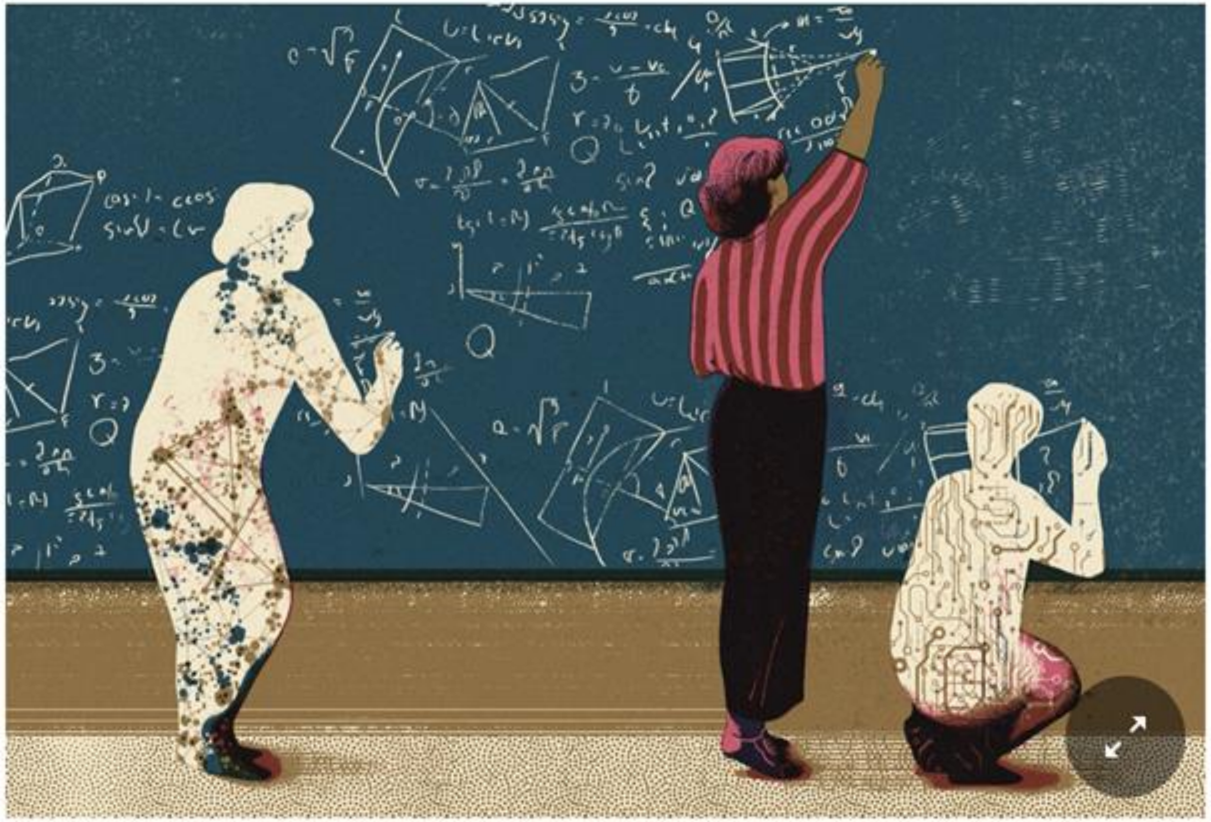
Score compared to best human

The New York Times

Stanford | News

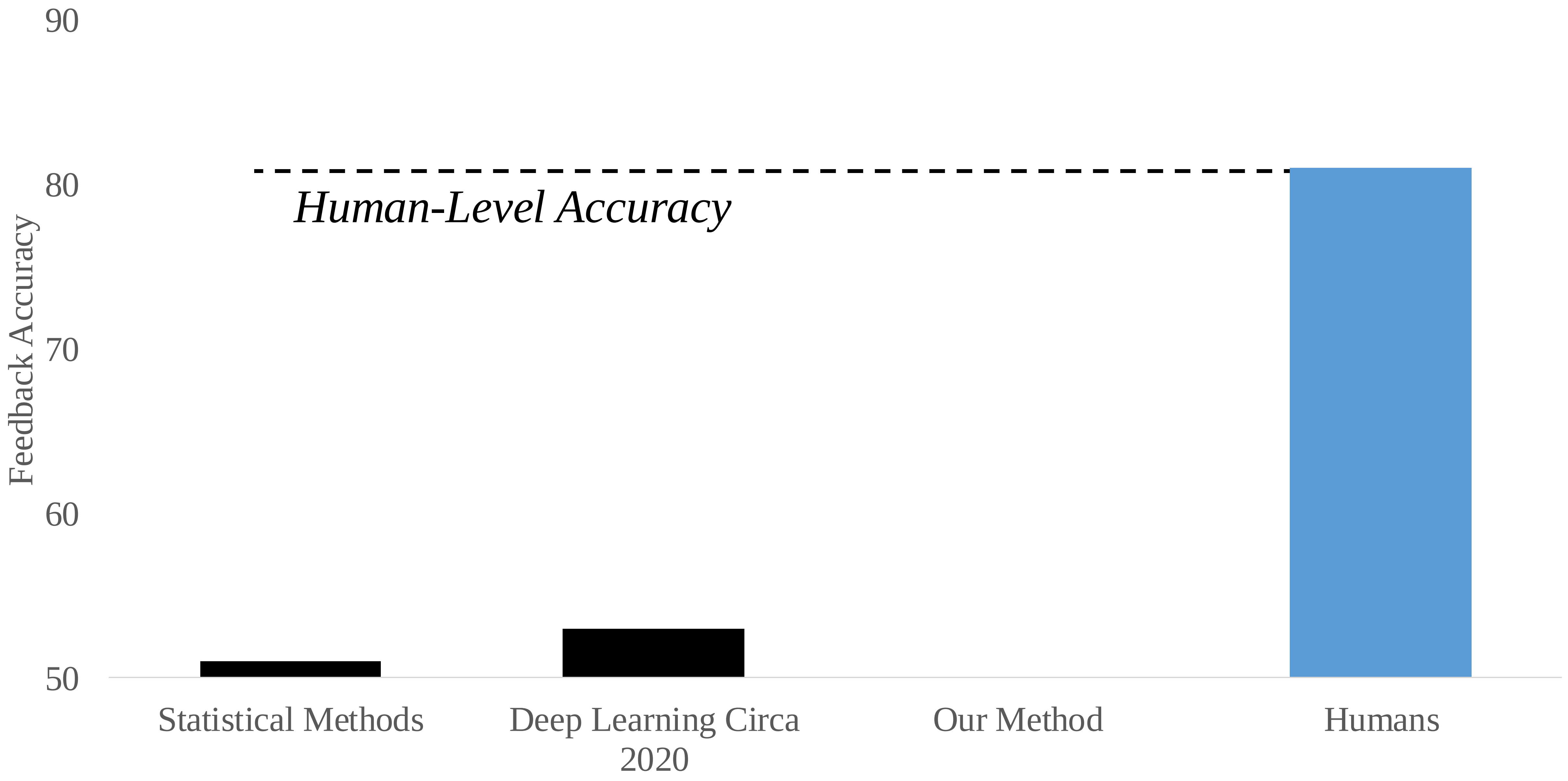
Can A.I. Grade Your Next Test?

Neural networks could give online education a boost by providing automated feedback to students.

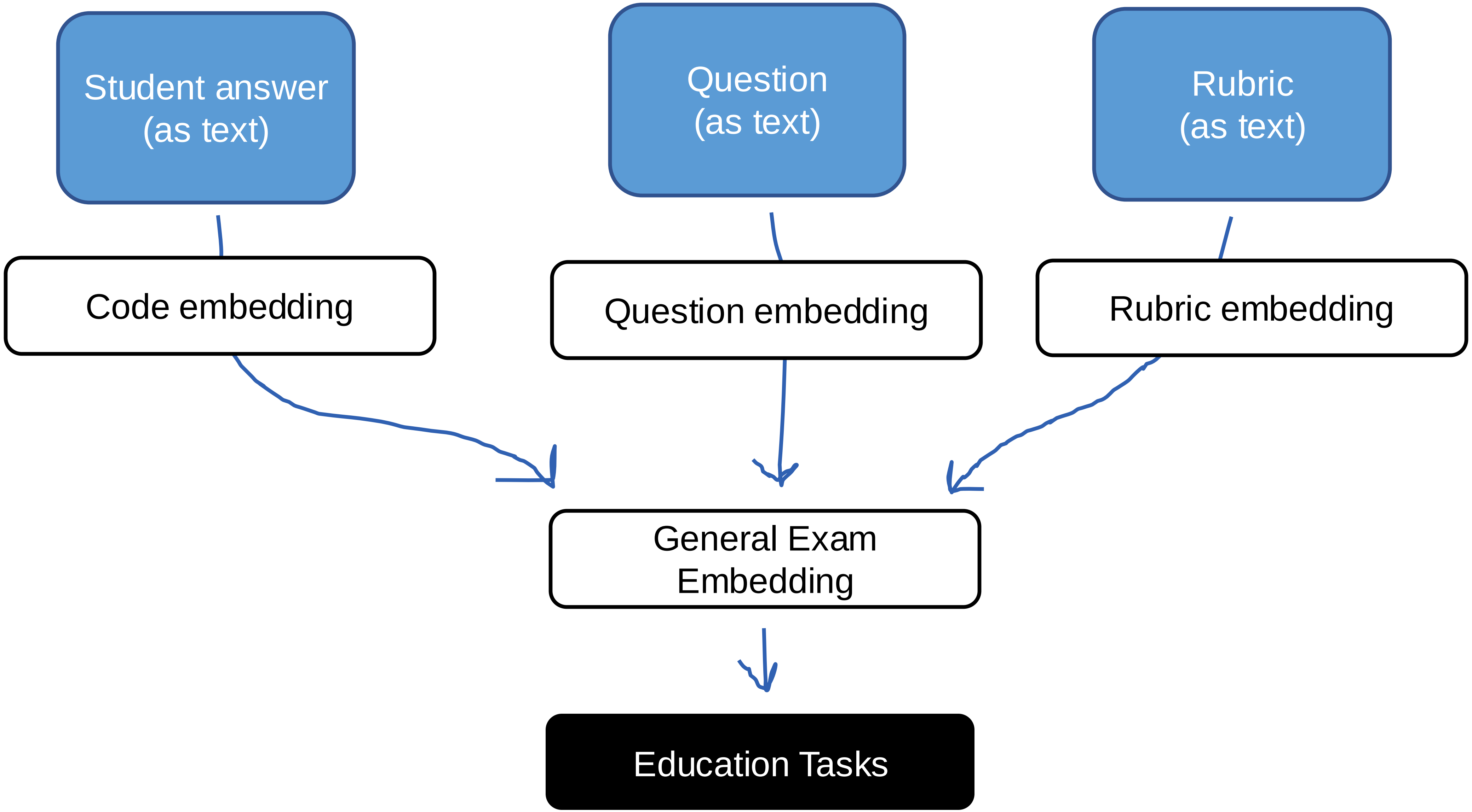


https://www.nytimes.com/2021/07/20/technology/ai-education-neural-networks.html

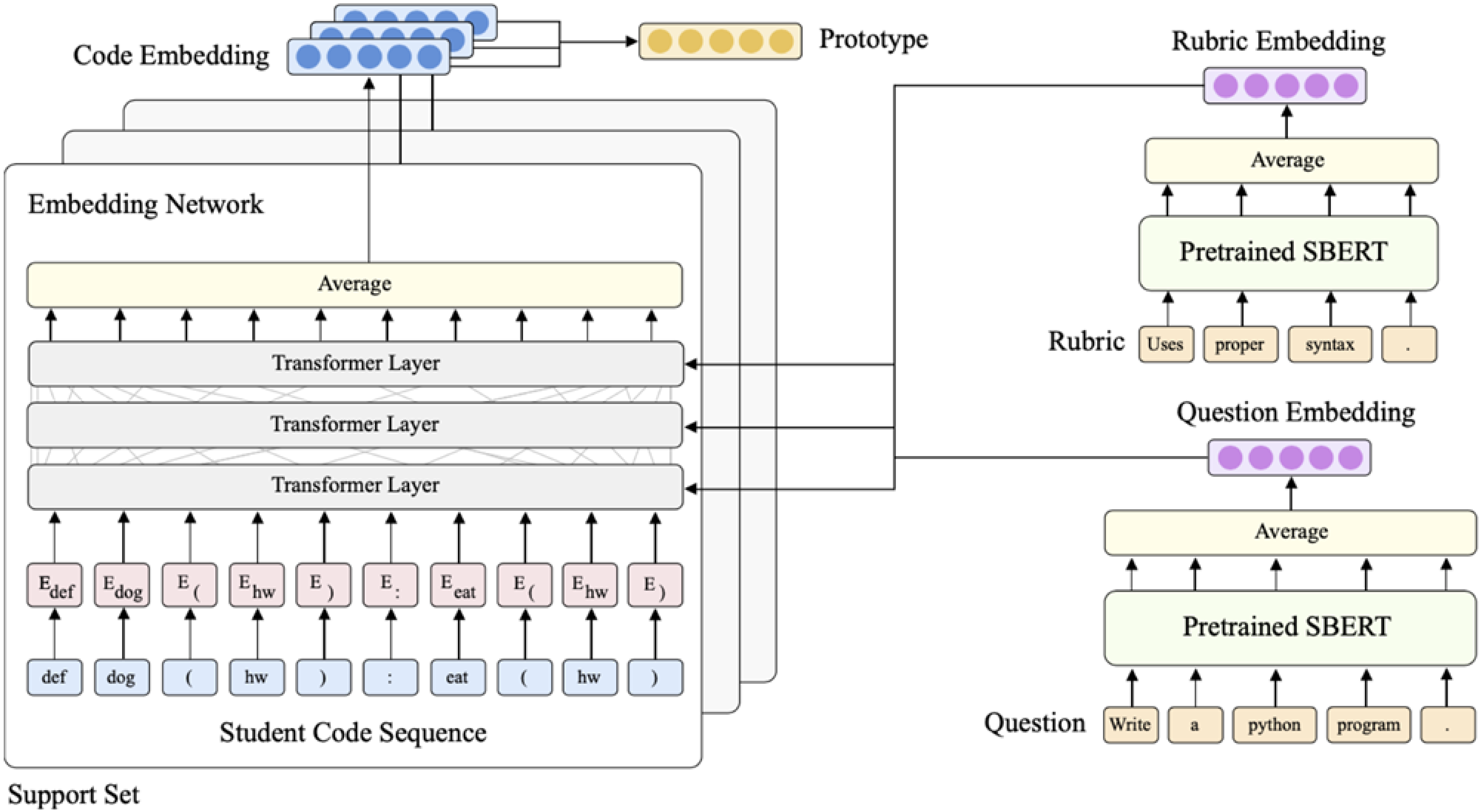
Rubric Level Accuracy on Few-Shot Grading a Novel Question



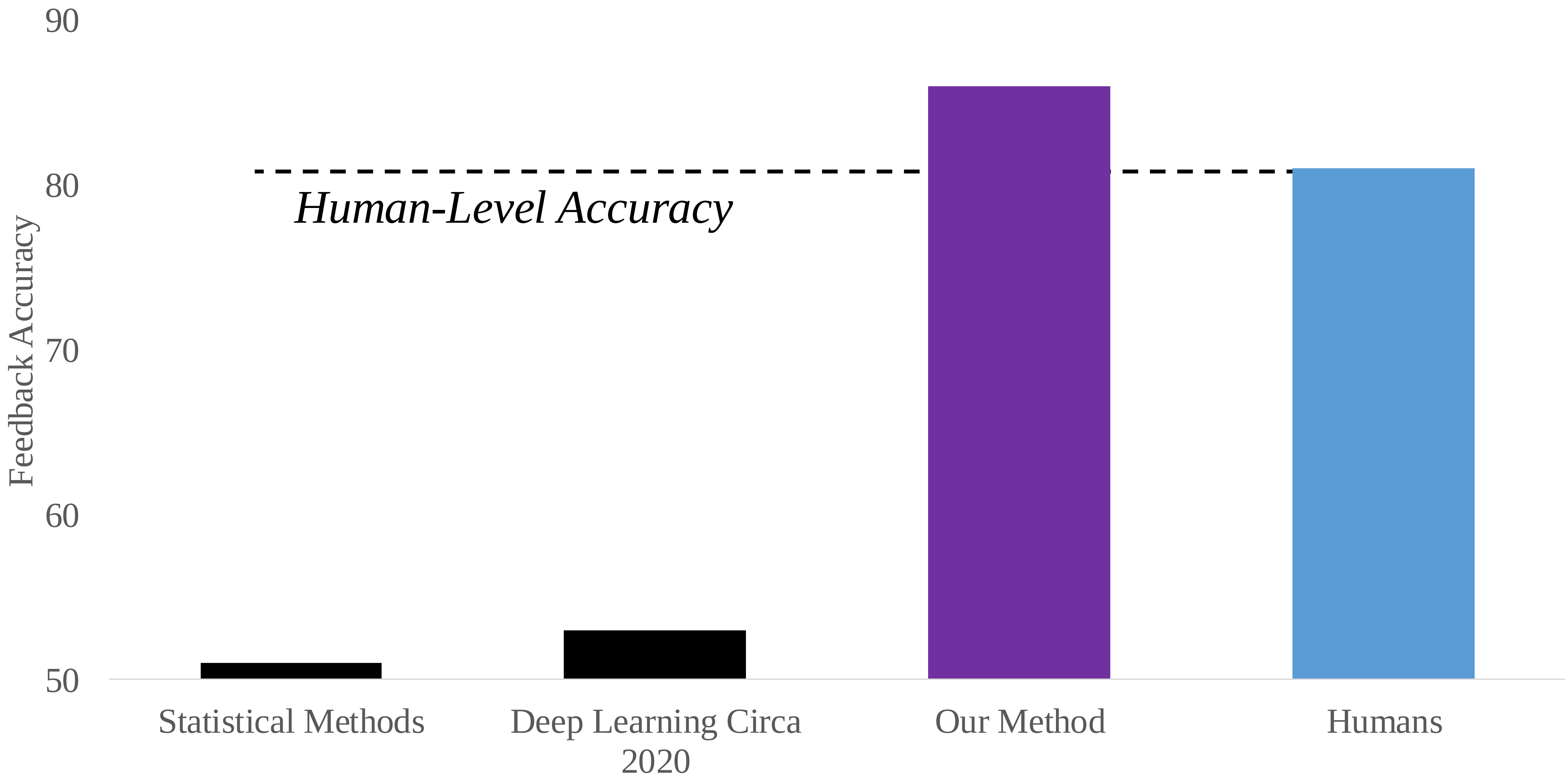
General Exam Grading Model



Invented the Proto-Transformer

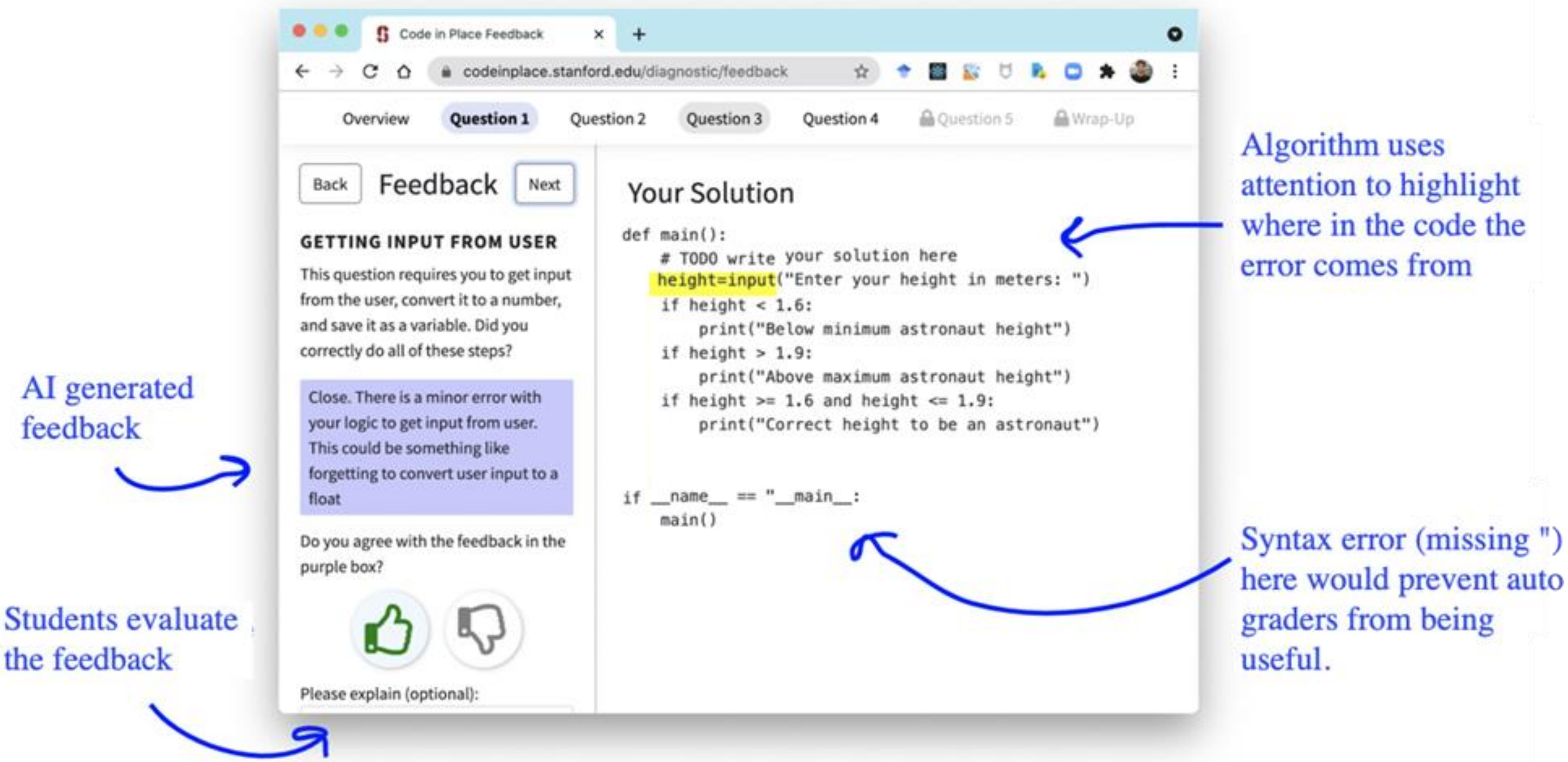


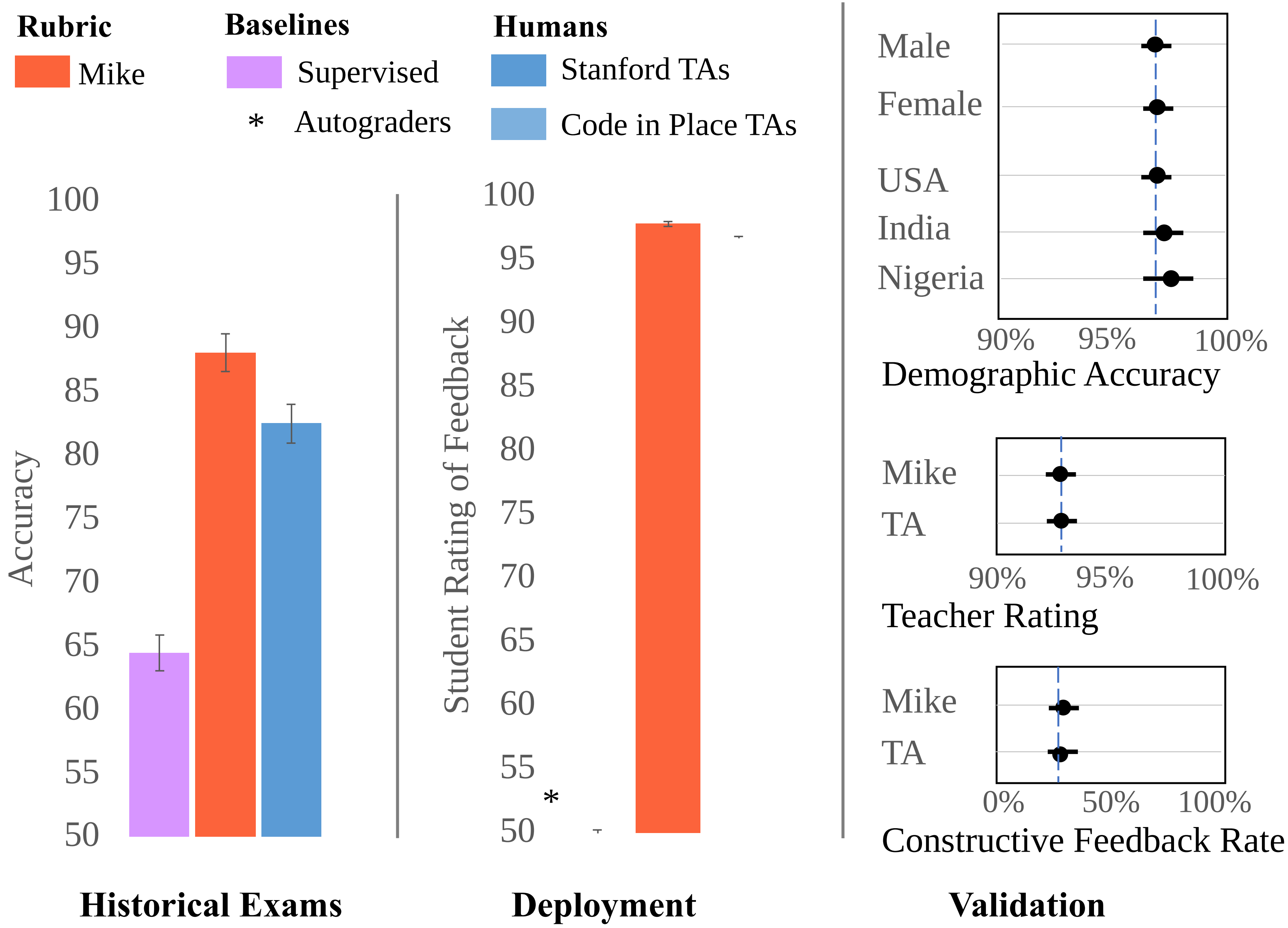
Rubric Level Accuracy on Few-Shot Grading a Novel Question



Gave Feedback to 3,500 Real Students

Do you agree? AI feedback **97.9%**. Human feedback 96.7%





Rubric
Mike
Baselines
Supervised
* Autograders
Humans
Stanford TAs
Code in Place TAs
Accuracy
100
95
90
85
80
75
70
65
60
55
50
Student Rating of Feedback
*
Male
Female
USA
India
Nigeria
90%
95%
100%
Demographic Accuracy
Mike
TA
90%
95%
100%
Teacher Rating
Mike
TA
0%
50%
100%
Constructive Feedback Rate
Historical Exams
Deployment
Validation

But what about interactive, creative assignments?

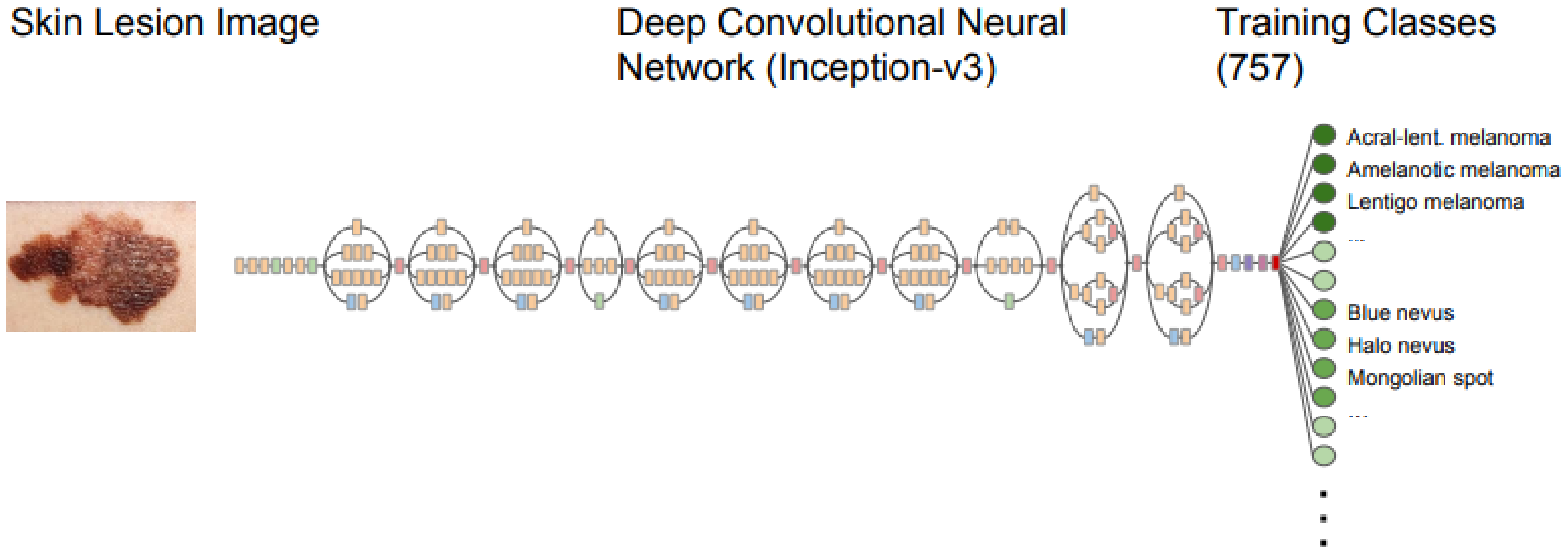
1M ungraded code.org assignments.

The AI is shown a brand new student game. Does it work?

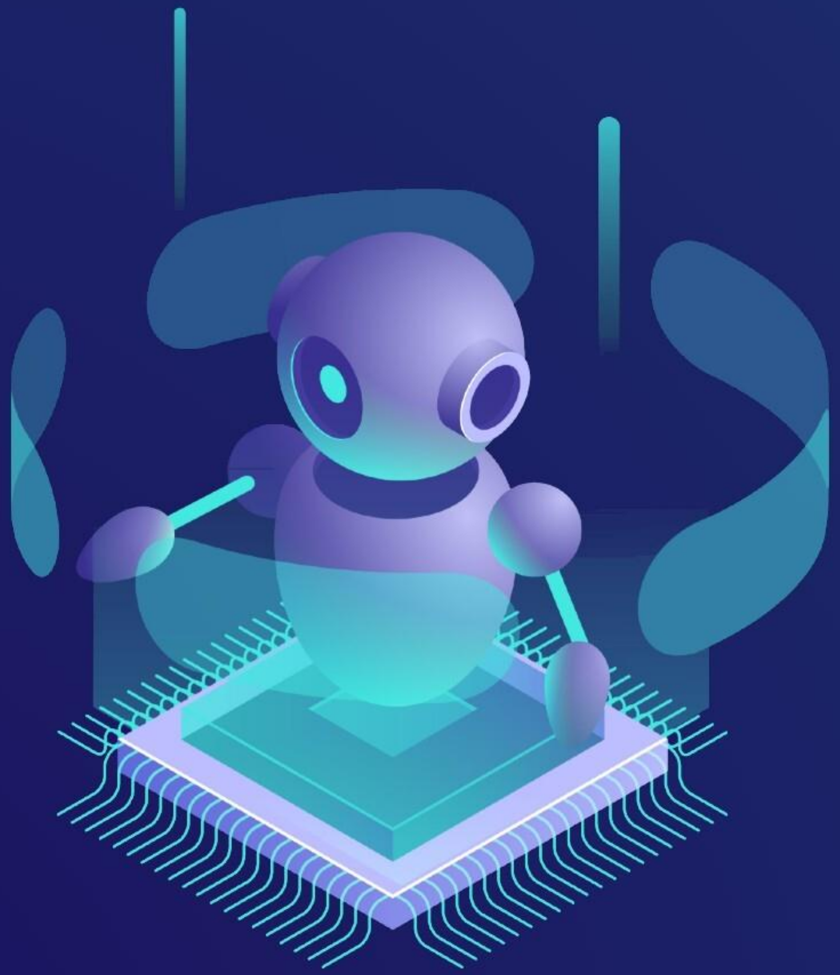
Simultaneously learn to grade and play to grade.

Majority class: 50%
Code-as-text: 67%
Play-to-grade: **94%**

Detecting skin cancer



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



GPT-3