Carnegie Mellon University
Heinz College

# 95-865 Australia Lecture 5: Introduction to Predictive Data Analytics, Neural Nets, and Deep Learning

George Chen

# Announcements

- No the quiz hasn't been graded yet

- Please make sure you have AWS set up with AWS Educate credits (bug Erick)

- Python 3.7 currently has some compatibility issues with Keras and Tensorflow — please downgrade your Python to version 3.6 if you're using 3.7!!!

```
conda install python=3.6
conda install keras
```

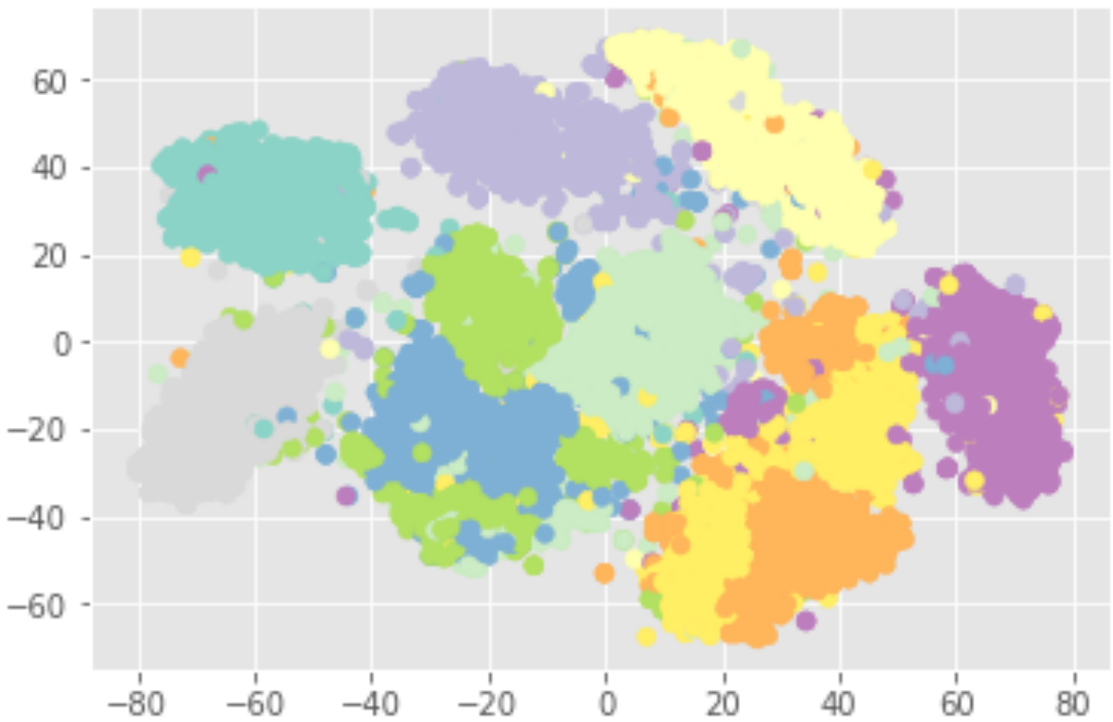
Disclaimer: unfortunately "***k***"
means many things

# Previous Lecture: Topic Modeling

- There are actually *many* topic models, not just LDA
  - Correlated topic models, Pachinko allocation, biterm topic models, anchor word topic models, …

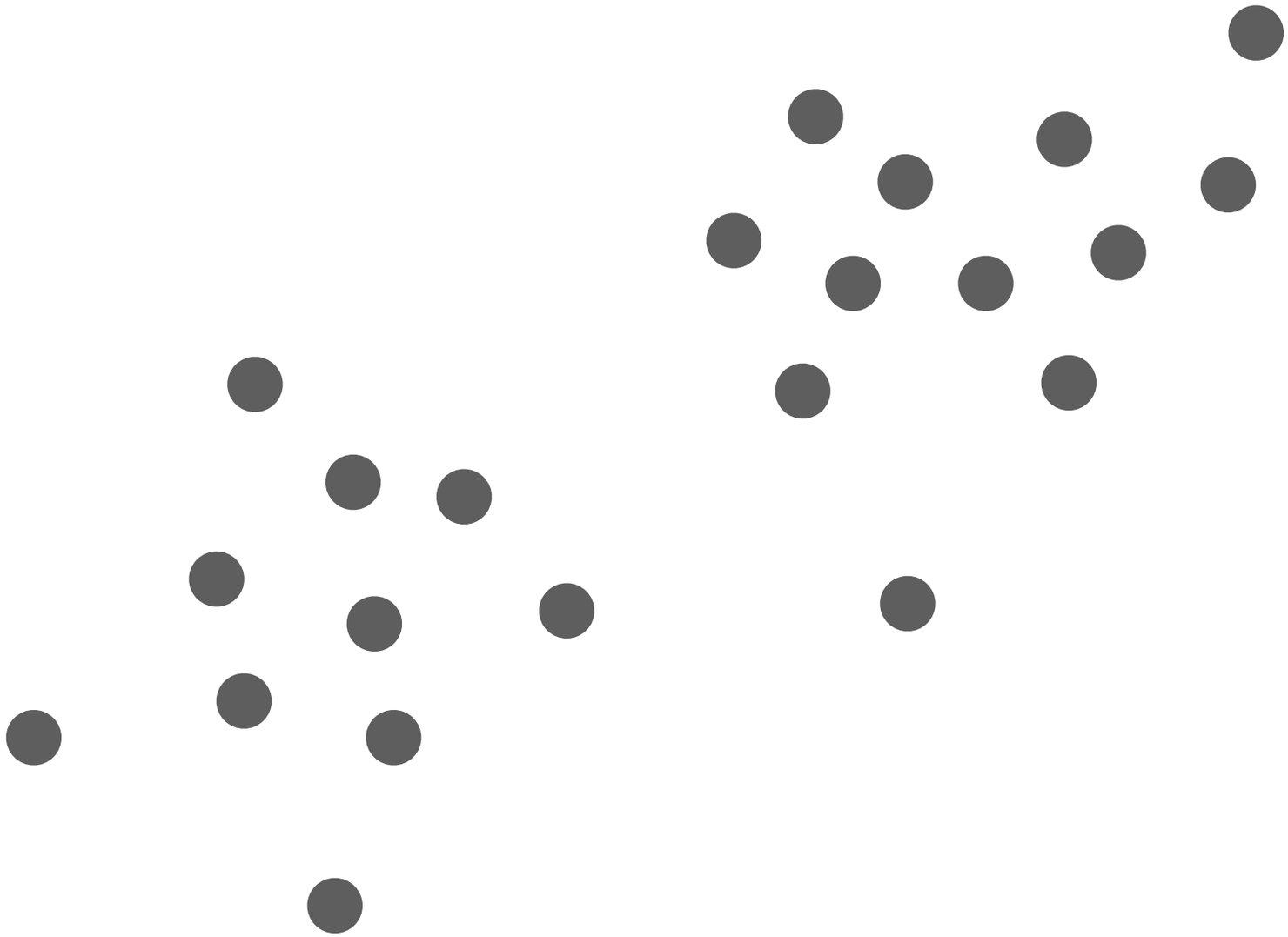
- Dynamic topic models: tracks how topics change *over time*
  - This sort of idea could be used to figure out how user tastes change over time in a recommendation system
    - Could try to see if there are existing patterns for how certain topics become really popular

# What if we have labels?




Example: MNIST handwritten digits have known labels

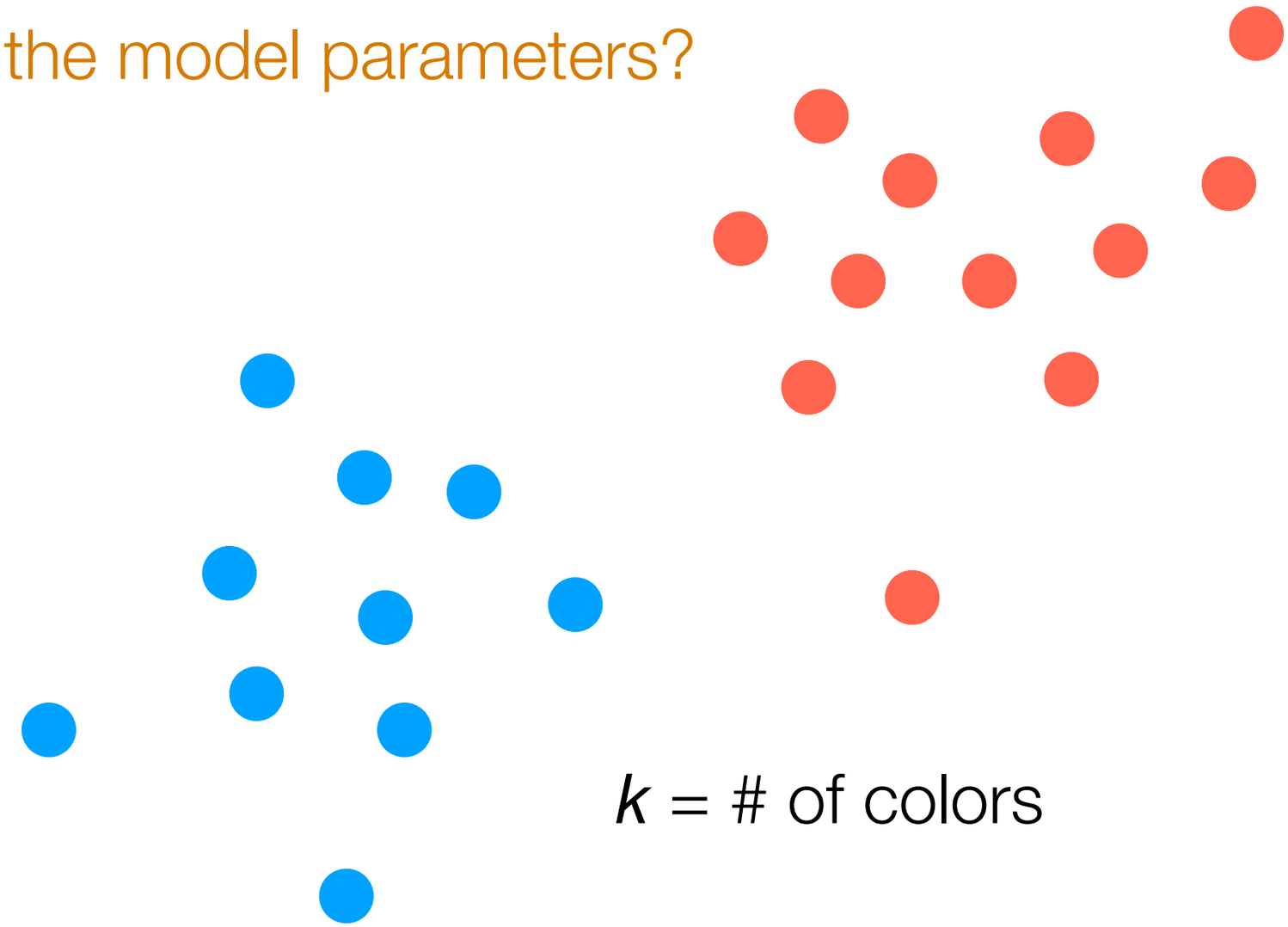
If the labels are known…



If the labels are known…

And we assume data generated by GMM…

What are the model parameters?



$k$ = # of colors

We can directly estimate cluster means, covariances

# Flashback: Learning a GMM

Don’t need this top part if we know the labels!

Step 0: Pick *k*

Step 1: Pick guesses for **cluster means and covariances**

**Repeat until convergence:**

Step 2: Compute probability of each point belonging to each of the *k* clusters

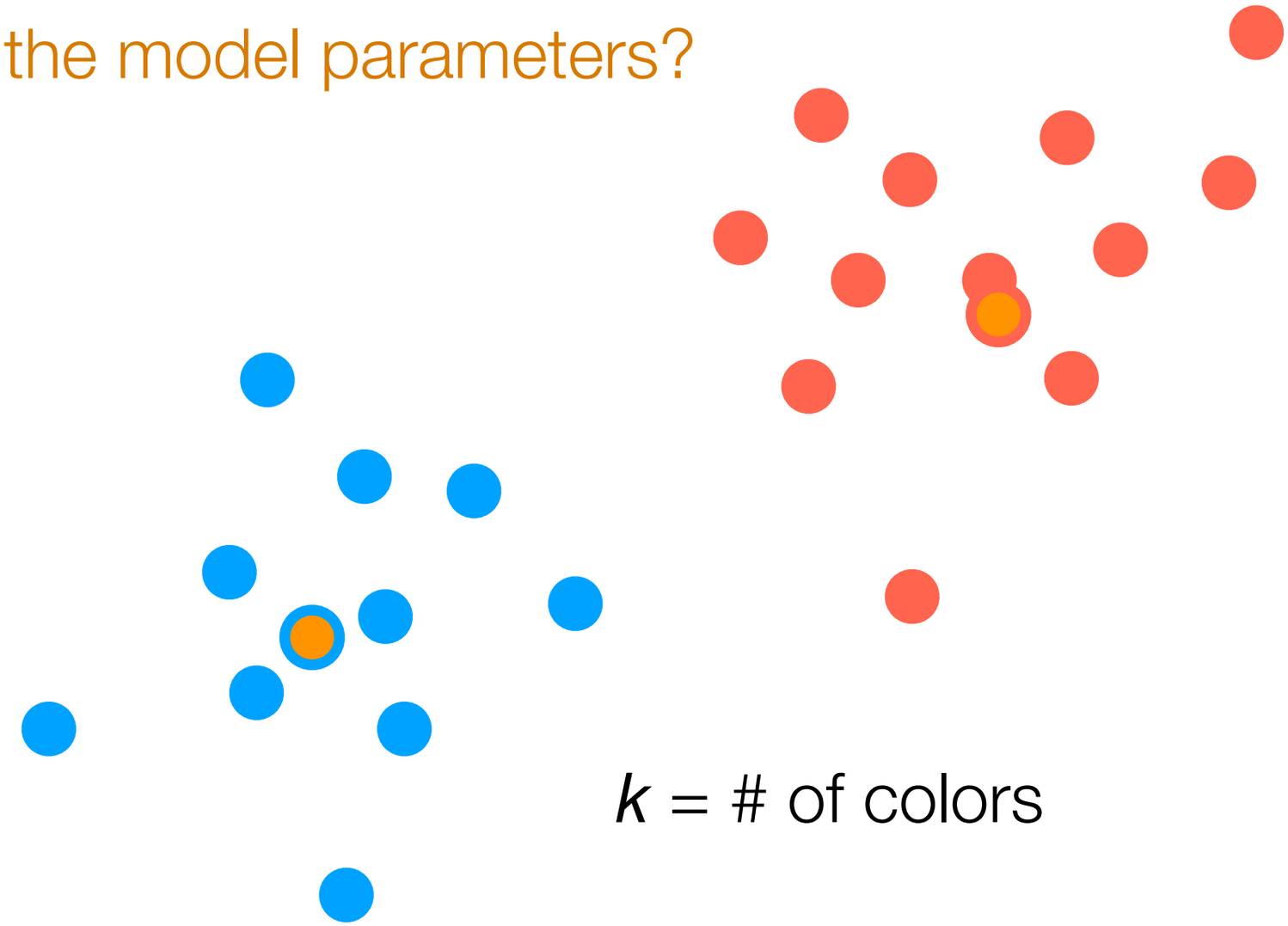
Step 3: Update **cluster means and covariances** carefully accounting for probabilities of each point belonging to each of the clusters

We don’t need to repeat until convergence

If the labels are known…

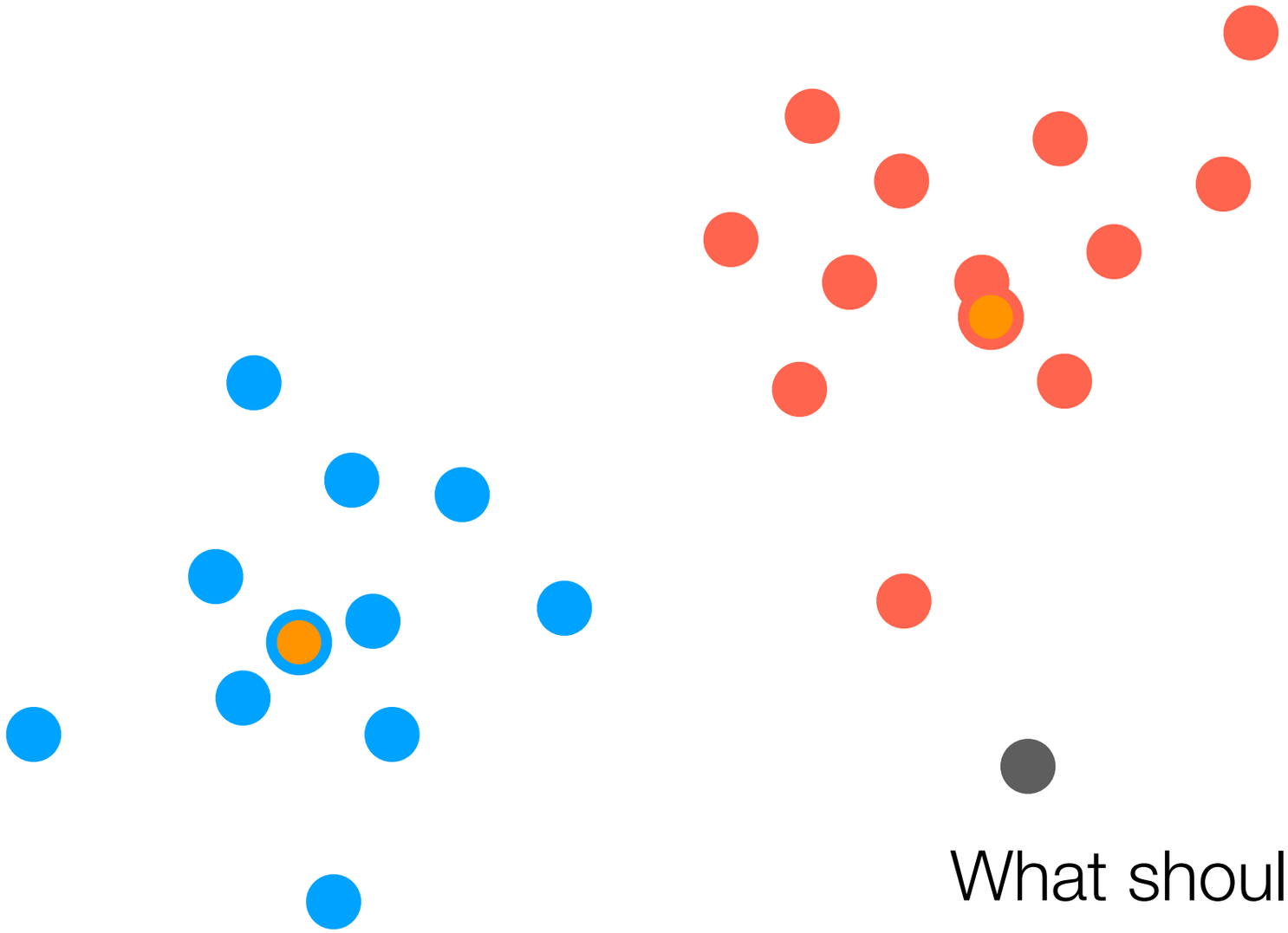
And we assume data generated by GMM…

What are the model parameters?



$k$ = # of colors

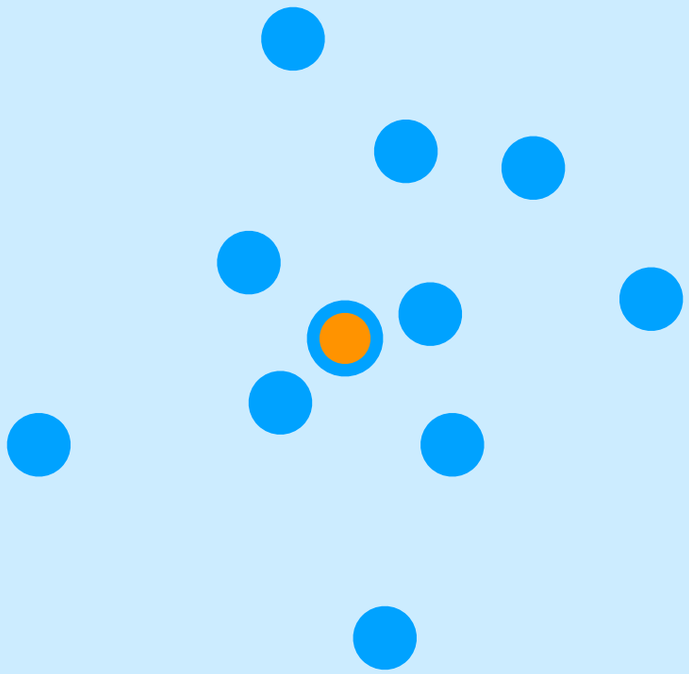
We can directly estimate cluster means, covariances



What should the label of this new point be?

Whichever cluster has higher probability!

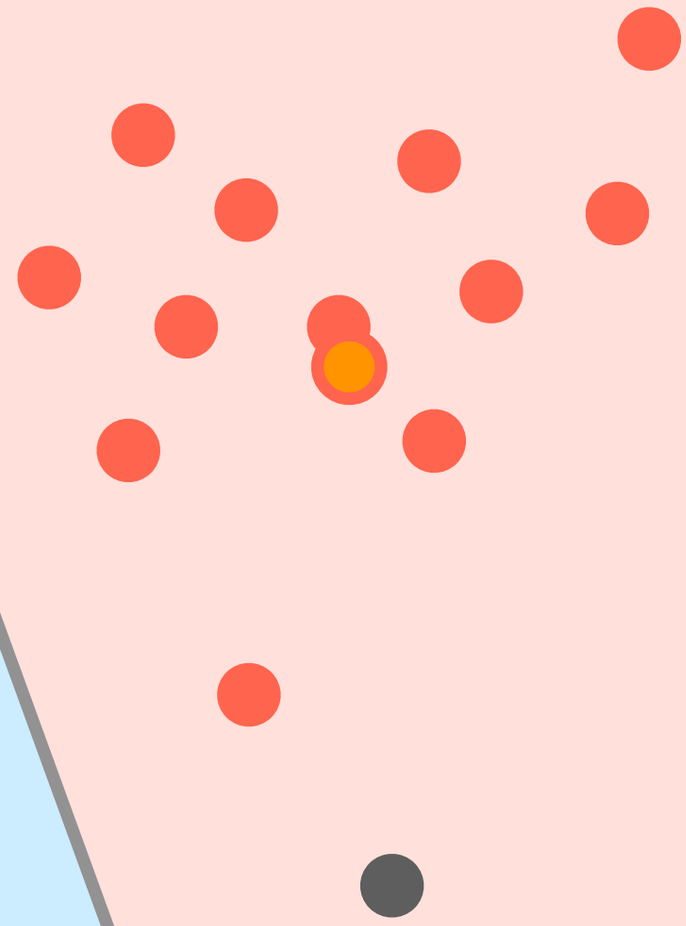
Decision boundary



This classifier we've created assumes a *generative model*

We just created a **classifier**

(a procedure that given a new data point tells us what "class" it belongs to)



What should the label of this new point be?

Whichever cluster has higher probability!

# You've seen generative models before for prediction

Linear regression!

Model parameters: slope *m*, intercept *b*

*y*

Label
(1D in this case)

*x*

Feature vector
(1D in this case)

Label
(1D in this case)

*y*

Model parameters: slope *m*, intercept *b*

*x*

Feature vector
(1D in this case)

For specific value of *x*,
assume *y* drawn from
Gaussian with mean
*mx*+*b*, standard dev $\sigma$

# Predictive Data Analysis

Training data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

Goal: Given new feature vector $x$, predict label $y$

- $y$ is discrete (such as colors red and blue)
  ➔ prediction method is called a **classifier**

- $y$ is continuous (such as a real number)
  ➔ prediction method is called a **regressor**

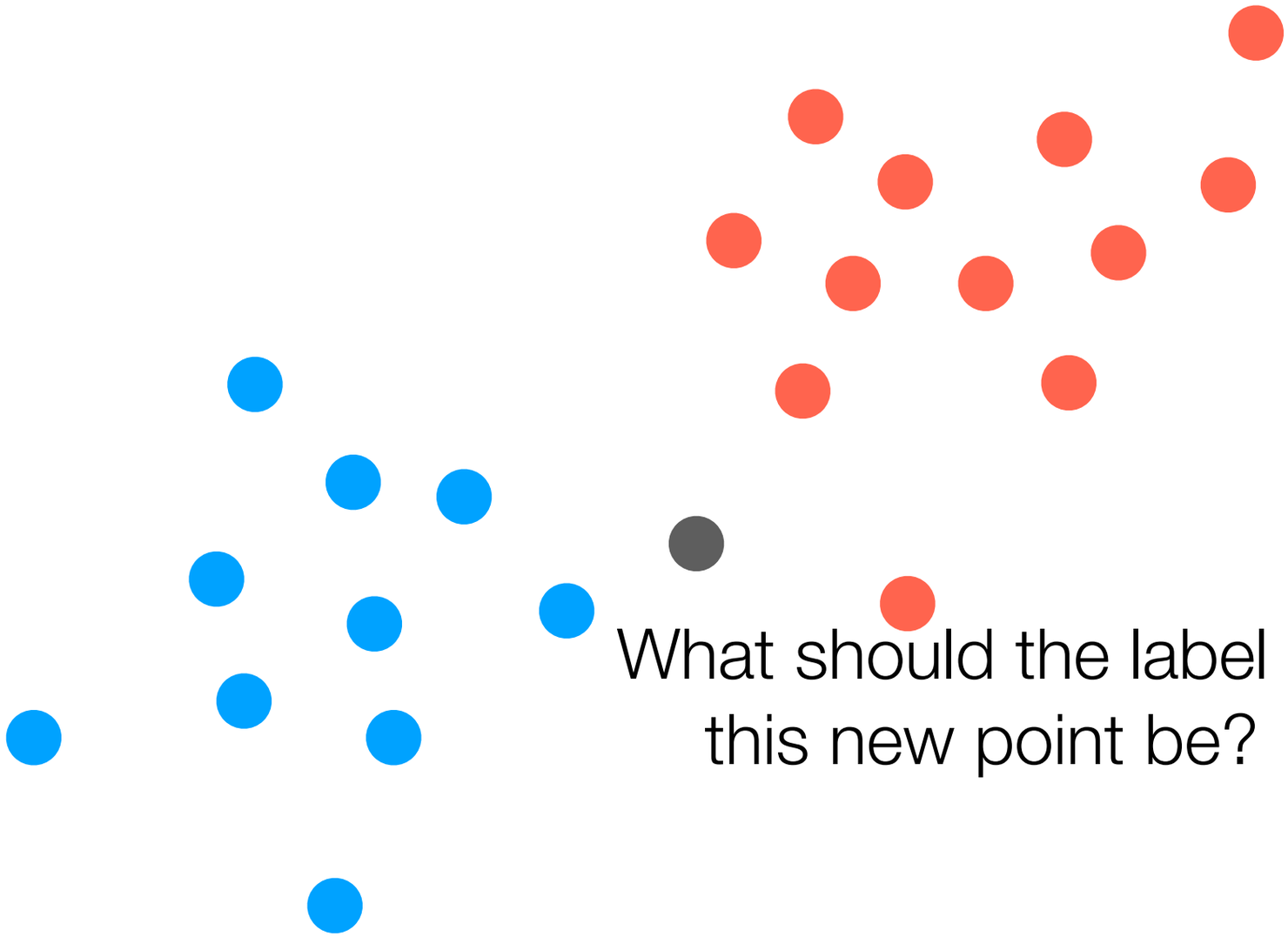
A giant zoo of methods

# Generative Models

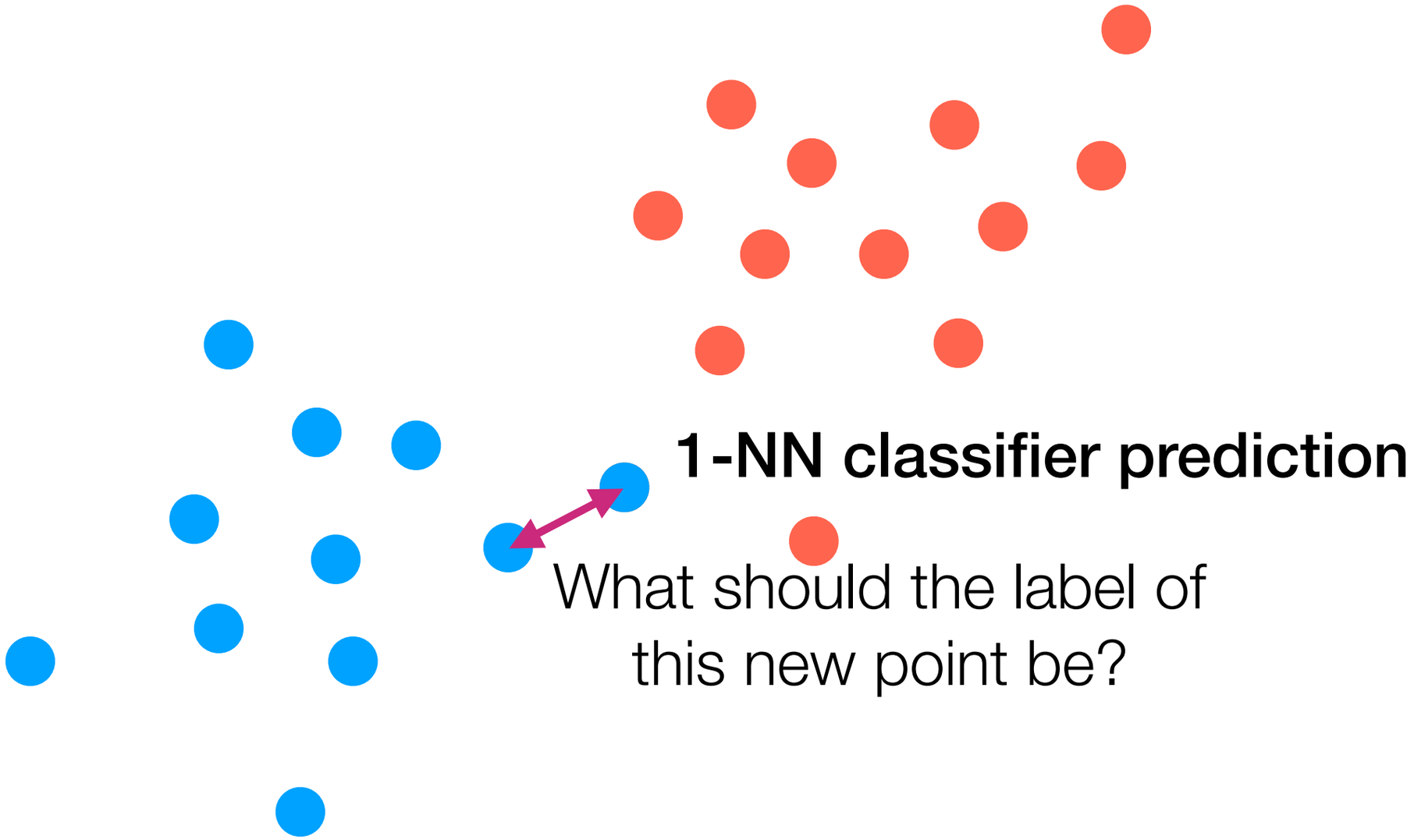
- Hypothesize a specific way in which data are generated
- After learning a generative model:
    - We can generate new synthetic data from the model
        - Usually generative models are probabilistic and we can evaluate probabilities for a new data point
- In contrast to generative models, there are *discriminative* methods that just care about learning a prediction rule

# Example of a Discriminative Method: *k*-NN Classification

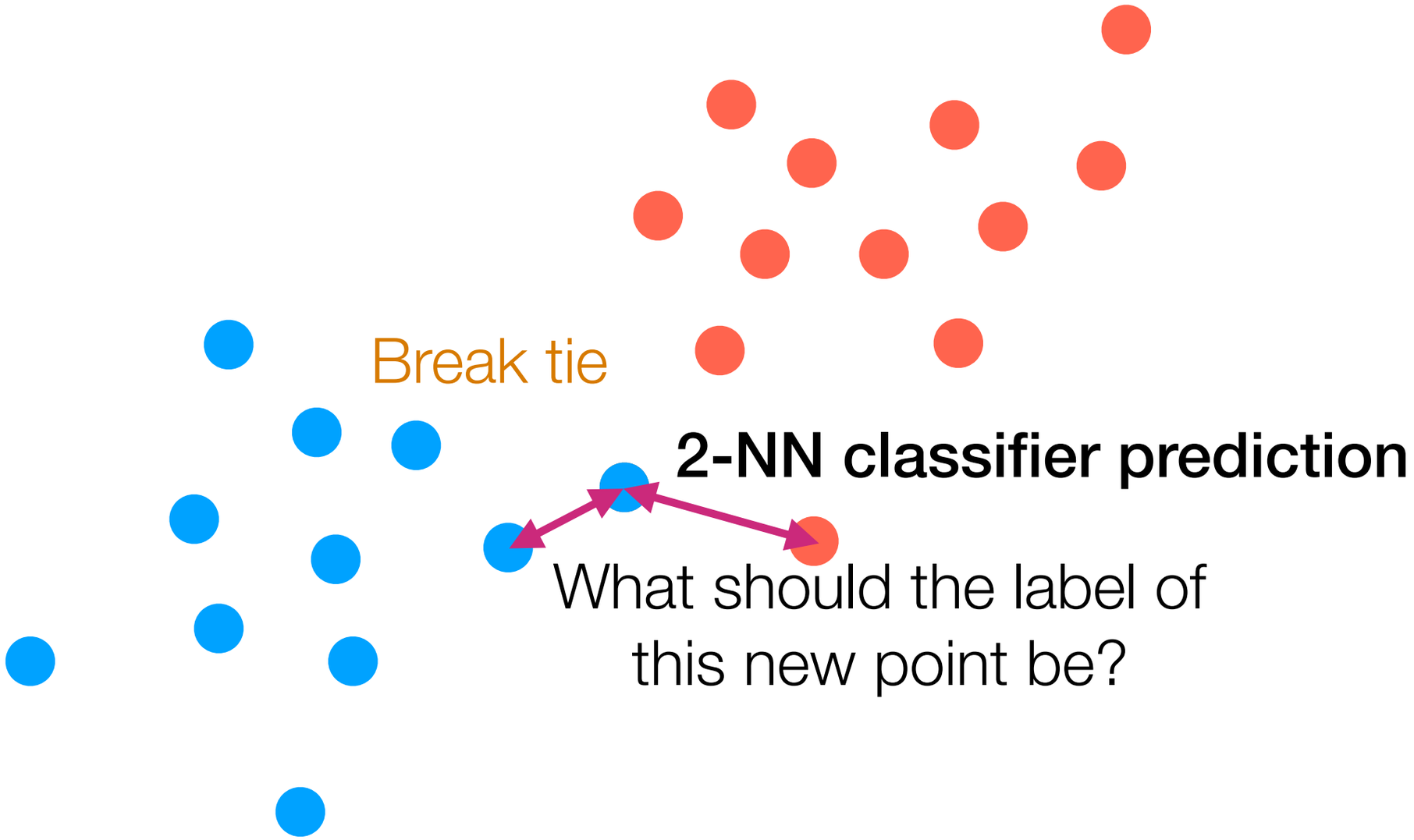
# Example: *k*-NN Classification



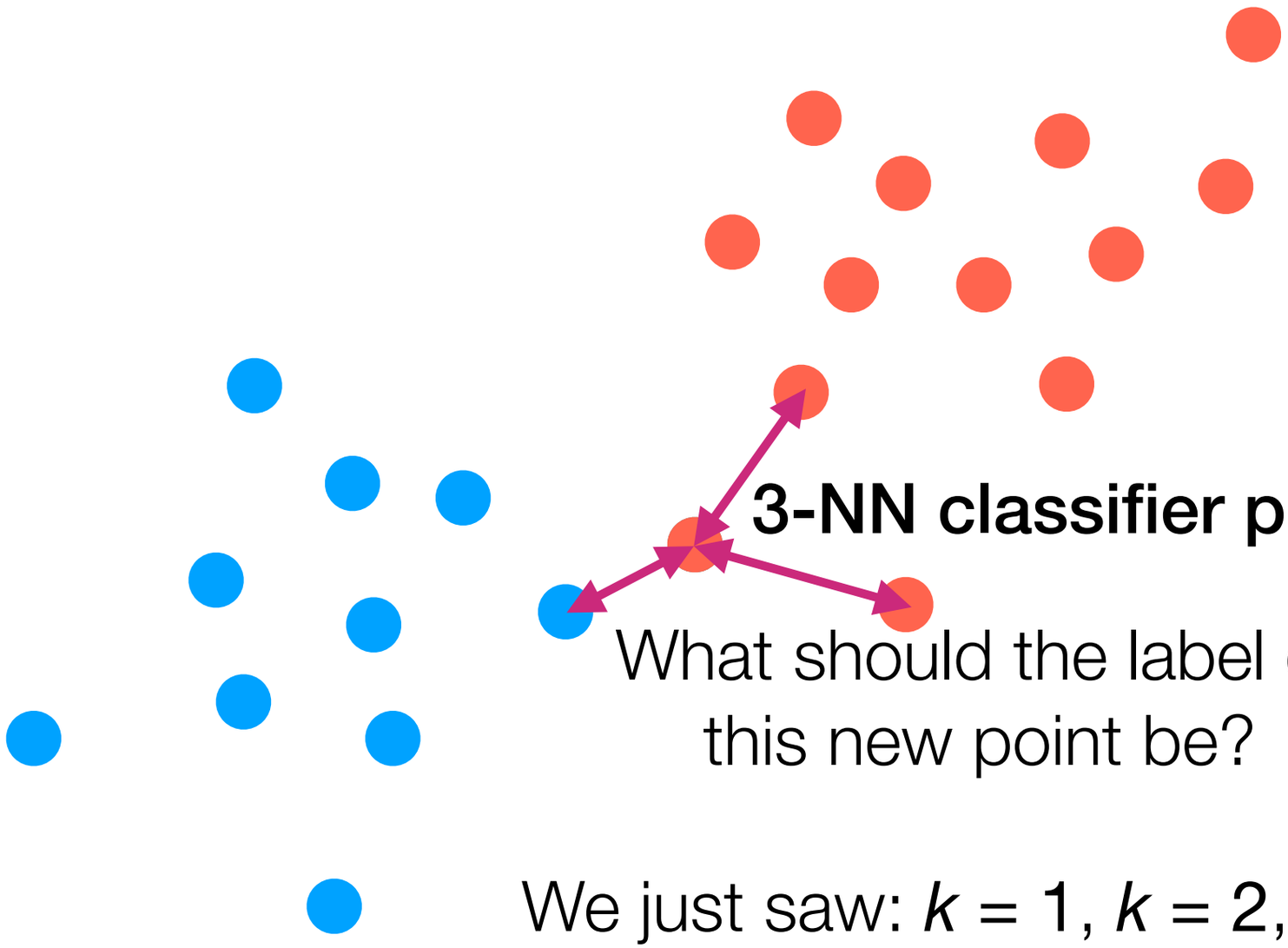
# Example: *k*-NN Classification



# Example: *k*-NN Classification



# Example: *k*-NN Classification



**3-NN classifier prediction**

What should the label of this new point be?

We just saw: $k = 1$, $k = 2$, $k = 3$

What happens if $k = n$?

# How do we choose *k*?

What I'll describe next can be used to select hyperparameter(s) for any prediction method

First: How do we assess how good a prediction method is?

# Hyperparameters vs. Parameters

- We fit a model's parameter to training data (terminology: we "learn" the parameters)
- We pick values of hyperparameters and they do *not* get fit to training data

- Example: Gaussian mixture model
  - Hyperparameter: number of clusters *k*
  - Parameters: cluster probabilities, means, covariances

- Example: *k*-NN classification
  - Hyperparameter: number of nearest neighbors *k*
  - Parameters: N/A

Training data

Training data point

Training data point

Training data point

Training data point

Training data point

Training data point

Training data point

Training data point

Training data point

Training data point

Example: Each data point is an email and we know whether it is spam/ham

Want to classify these points correctly

Test data point

Test data point

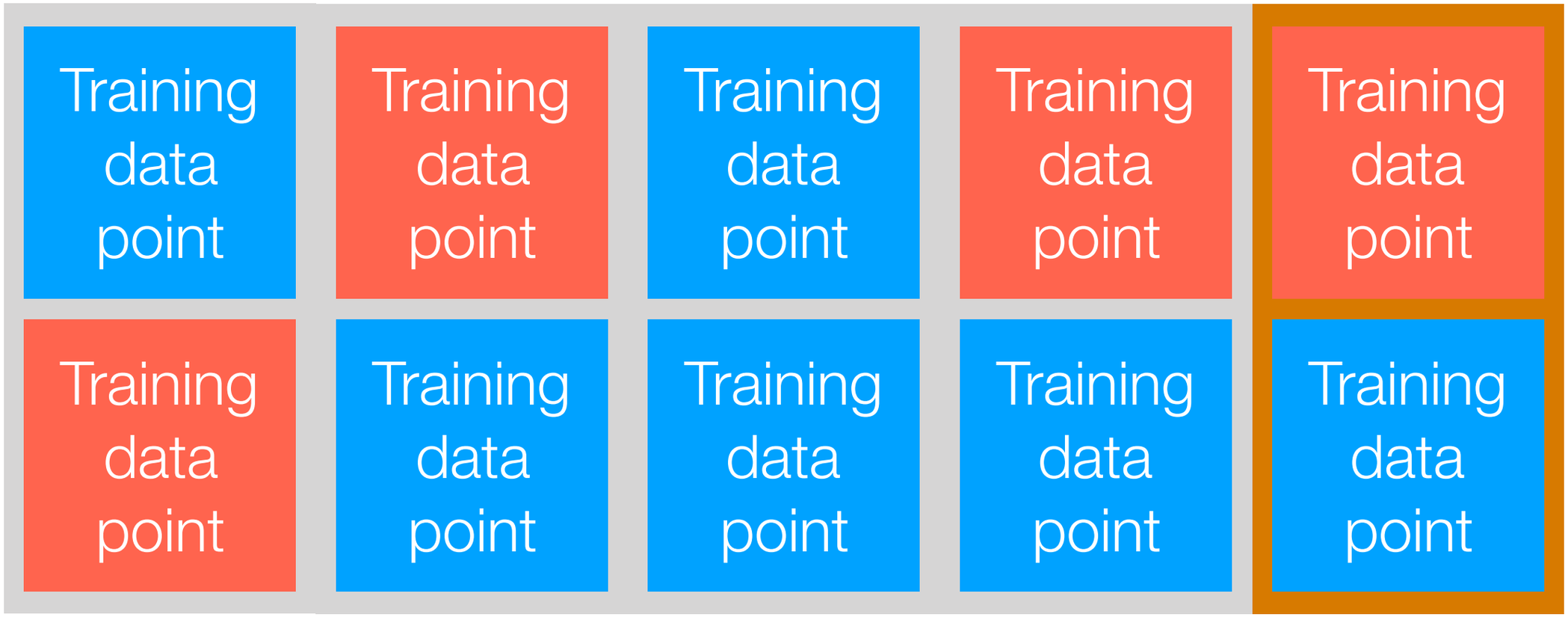
Test data point

Test data point

Test data point

Example: future emails to classify as spam/ham

## Predicted labels

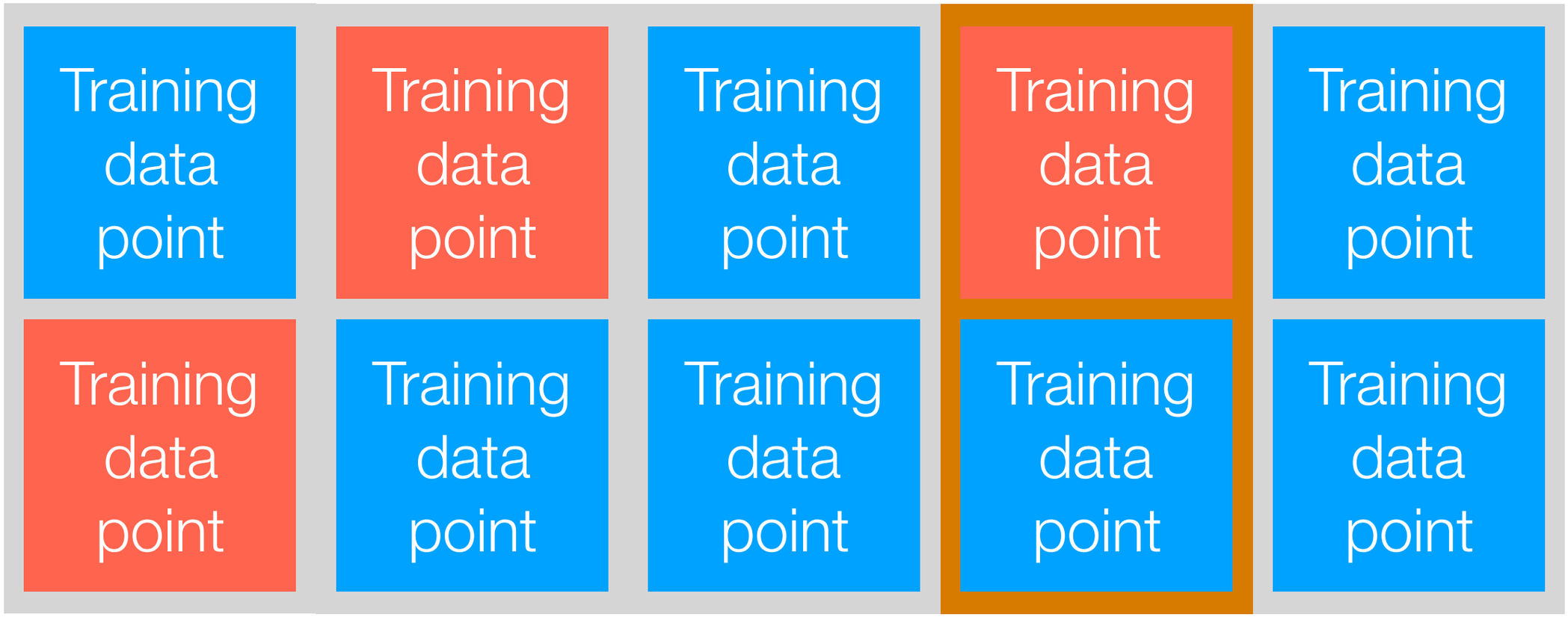



Train method on data in gray

Predict on data in orange

Compute prediction error

50%



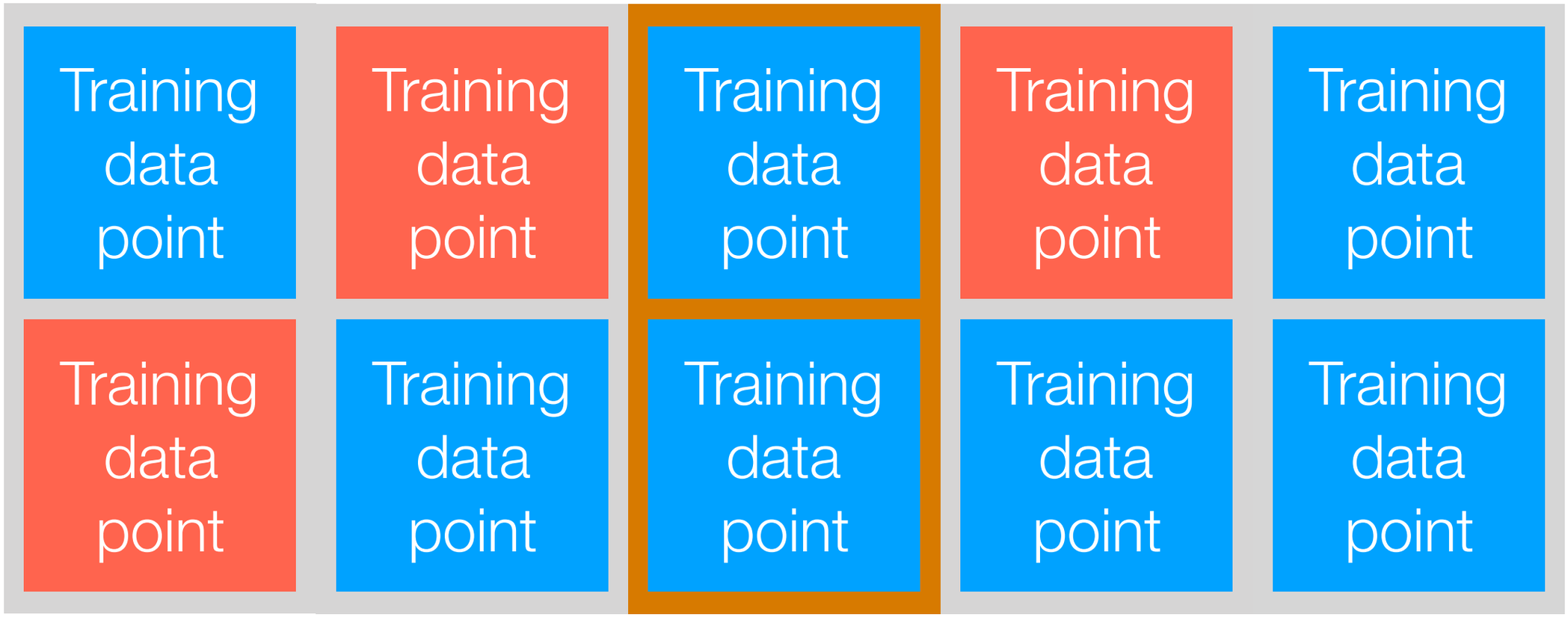

Train method on data in gray

Predict on data in orange

Compute prediction error

0%

50%




Train method on data in gray

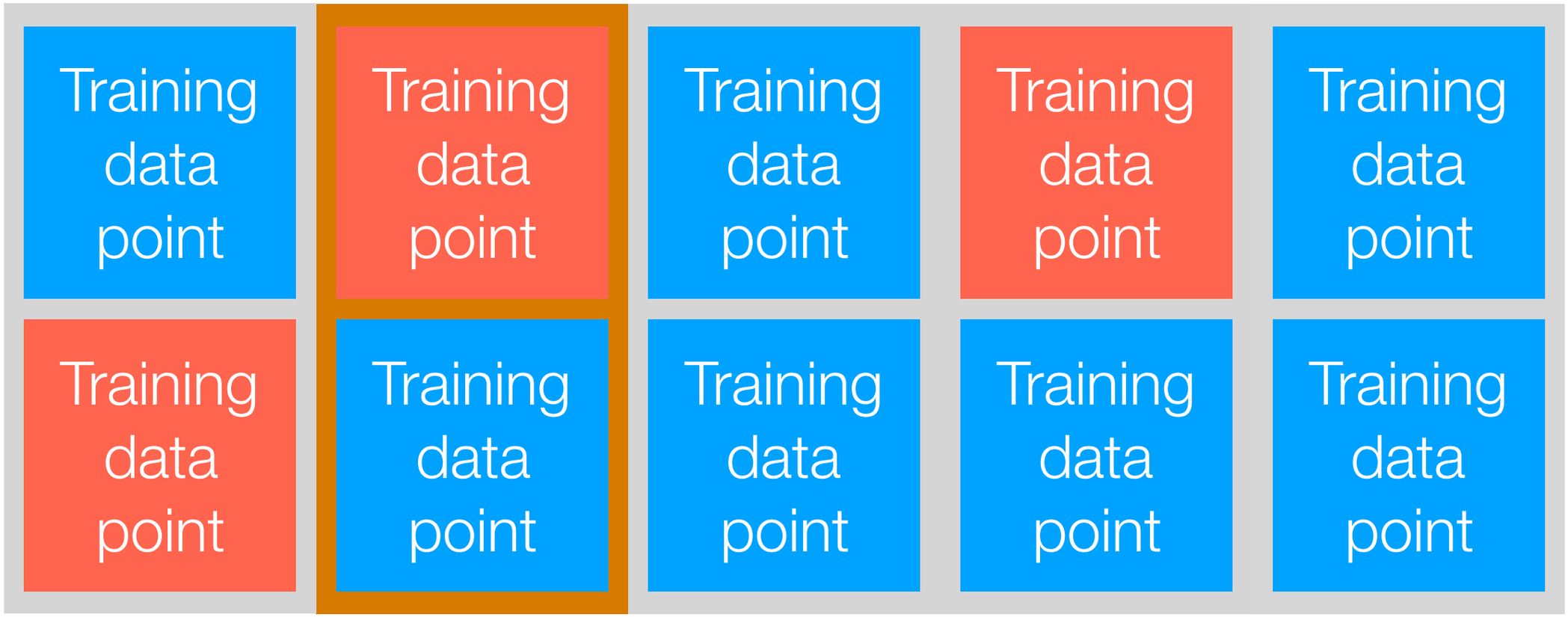
Predict on data in orange

Compute prediction error

50%

0%

50%




Train method on data in gray

Predict on data in orange

Compute prediction error

0%

50%

0%

50%

| | | | | |
|---|---|---|---|---|
| Training data point | Training data point | Training data point | Training data point | Training data point |
| Training data point | Training data point | Training data point | Training data point | Training data point |

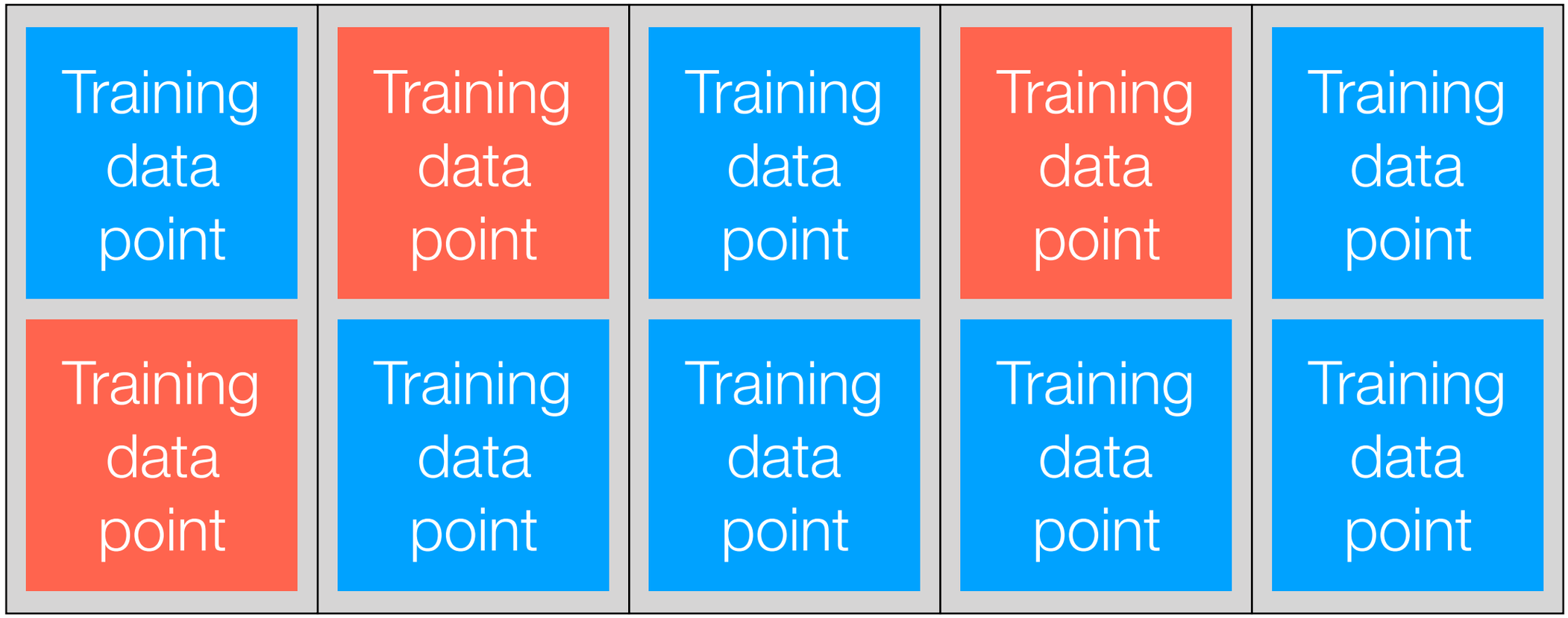
Train method on data in gray

Predict on data in orange

Compute prediction error

0% 0% 50% 0% 50%

Average error: (0+0+50+0+50)/5 = 20%




1. Shuffle data and put them into “folds” (5 folds in this example)
2. For each fold (which consists of its own train/validation sets):
   (a) Train on fold’s training data, test on fold’s validation data
   (b) Compute prediction error
3. Compute average prediction error across the folds

not the same *k* as in *k*-means or *k*-NN classification

# *k*-fold Cross Validation

| | | | | |
|---|---|---|---|---|
| Training data point | Training data point | Training data point | Training data point | Training data point |
| Training data point | Training data point | Training data point | Training data point | Training data point |

1. Shuffle data and put them into “folds” (*k*=5 folds in this example)
2. For each fold (which consists of its own train/validation sets):
   (a) Train on fold’s training data, test on fold’s validation data
   (b) Compute prediction error
3. Compute average prediction error across the folds

not the same *k* as in *k*-means or *k*-NN classification

# *k*-fold Cross Validation

| | | | | |
|---|---|---|---|---|
| Training data point | Training data point | Training data point | Training data point | Training data point |
| Training data point | Training data point | Training data point | Training data point | Training data point |

1. Shuffle data and put them into "folds" (*k*=5 folds in this example)
2. For each fold (which consists of its own train/validation sets):
   (a) Train on fold's training data, test on fold's validation data
   (b) Compute **some sort of prediction score**
3. Compute **average prediction score** across the folds

"cross validation score"

# Choosing *k* in *k*-NN Classification

Note: *k*-NN classifier has a single hyperparameter *k*

For each *k* = 1, 2, 3, …, the maximum *k* you are willing to try:

Compute 5-fold cross validation score using *k*-NN classifier as prediction method

Use whichever *k* has the best cross validation score

# Automatic Hyperparameter Selection

Suppose the prediction algorithm you're using has hyperparameters $\theta$

For each hyperparameter setting $\theta$ you are willing to try:

Compute 5-fold cross validation score using your algorithm with hyperparameters $\theta$

Use whichever $\theta$ has the best cross validation score

Why 5?

People have found using 10 folds or 5 folds to work well in practice but it's just empirical — there's no deep reason

**Important:** the cross validation score is trying to predict what the prediction quality will be on the unseen test data

Our earlier example had a cross validation score of 20% error

*This is a guess at how well the prediction method should perform on test data*

**This guess is <u>not</u> always accurate**

Example: Each data point is an email and we know whether it is spam/ham

Want to classify these points correctly

Test data point

Test data point

Test data point

Test data point

Test data point

Example: future emails to classify as spam/ham

# Different Ways to Measure Accuracy

Simplest way:

- **Raw error rate:** fraction of predicted labels that are wrong (this was in our cross validation example earlier)

In “binary” classification (there are 2 labels such as spam/ham) when 1 label is considered “positive” and the other “negative”:

- **Precision:** among data points predicted to be “positive”, what fraction of these predictions is correct?
- **Recall:** among data points that are actually “positive”, what fraction of these points is predicted correctly as “positive”? (also called **true positive rate**)
- **F1 score:**

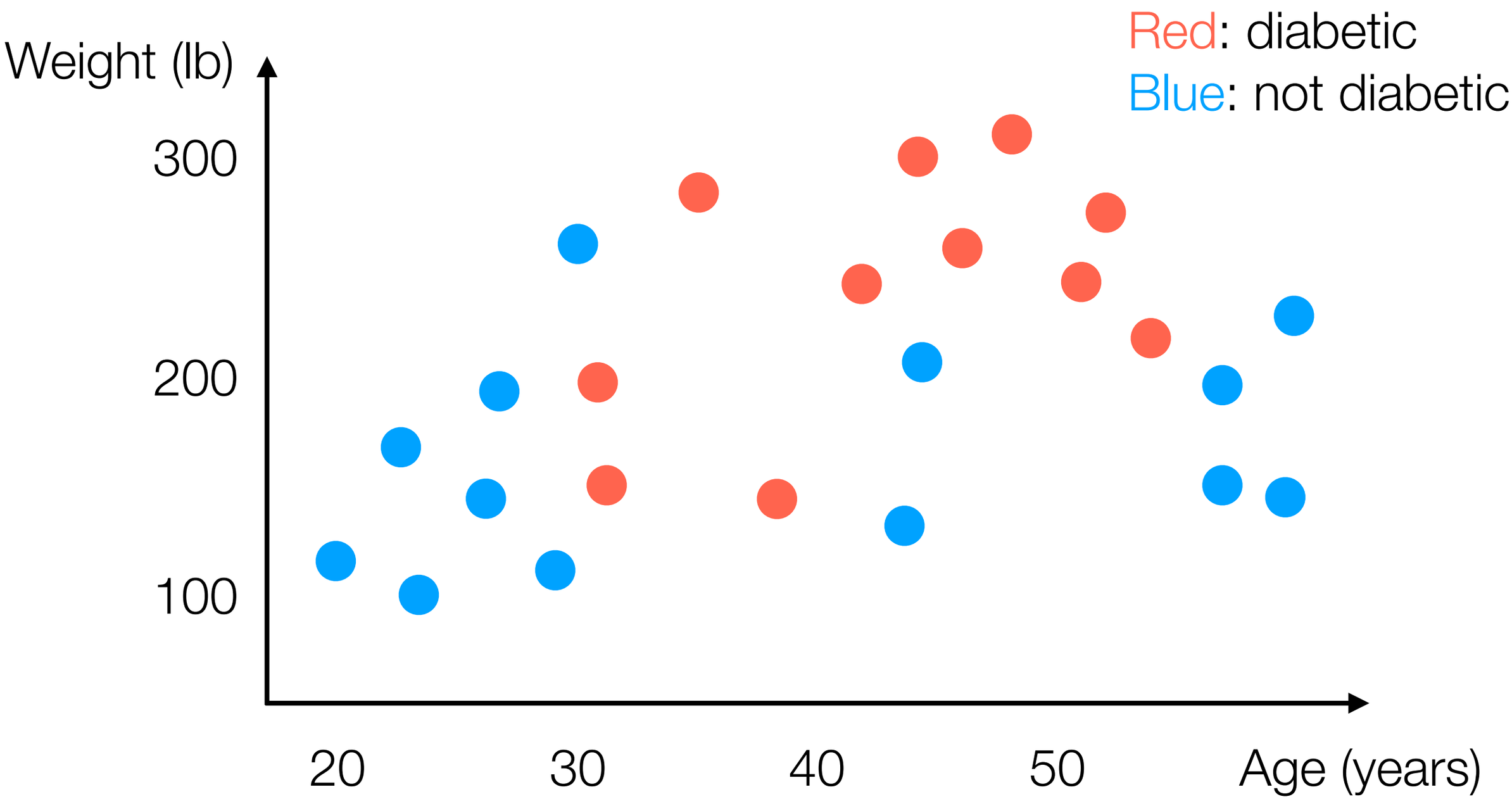
$$\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

# Prediction and Model Validation

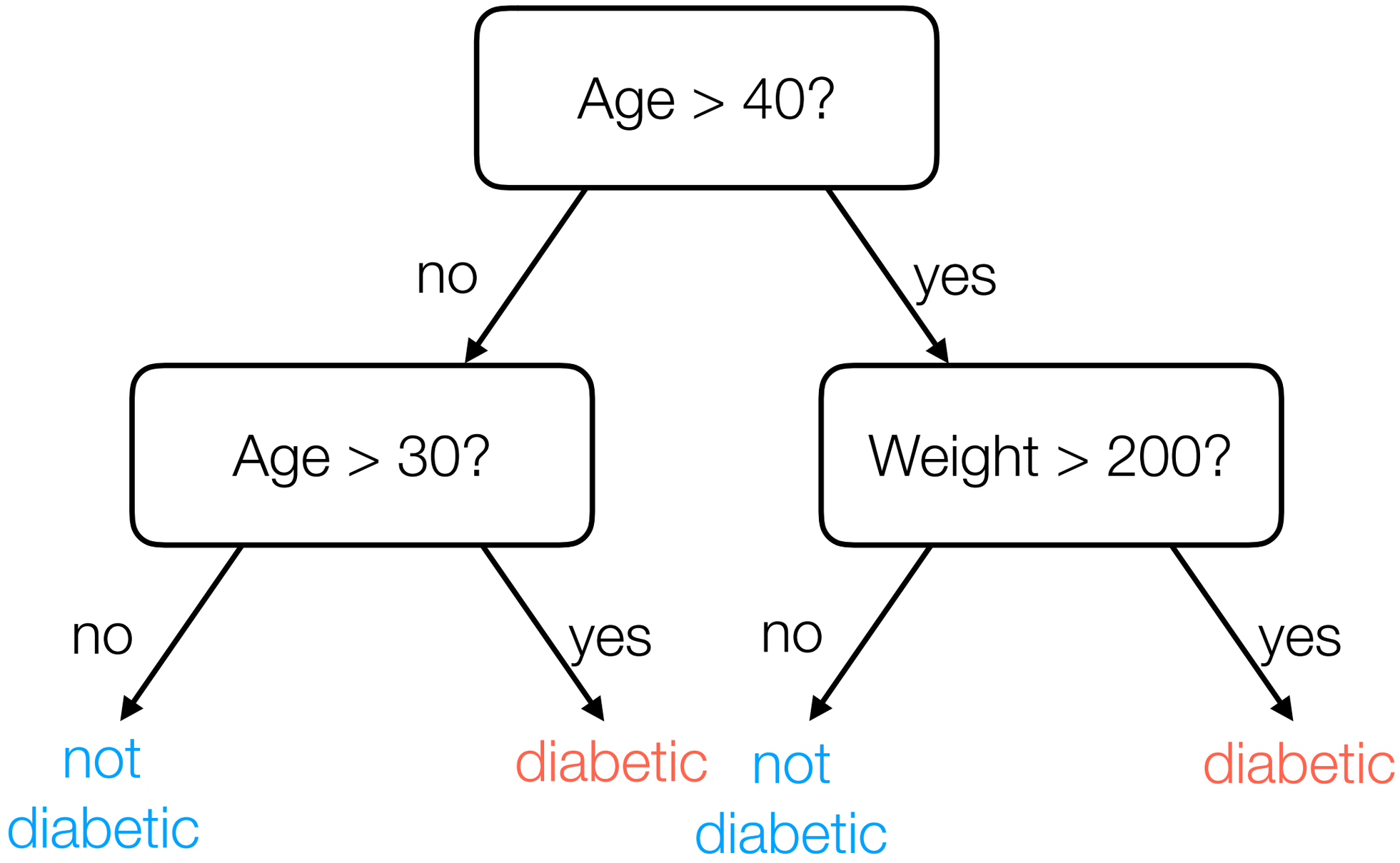
Demo

# Decision Trees

# Example Made-Up Data



# Example Decision Tree

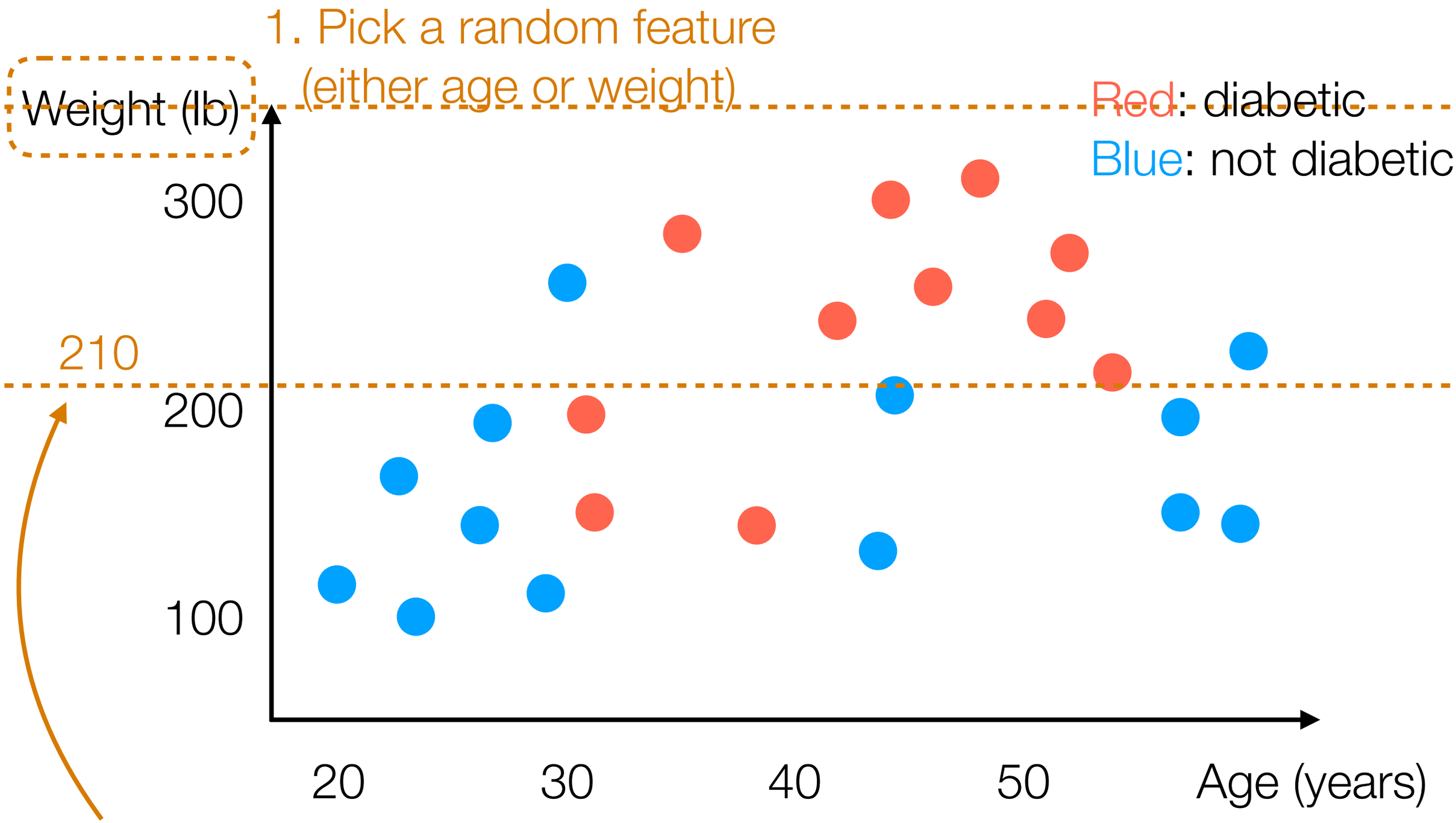


# Learning a Decision Tree

- Many ways: general approach actually looks a lot like divisive clustering *but accounts for label information*

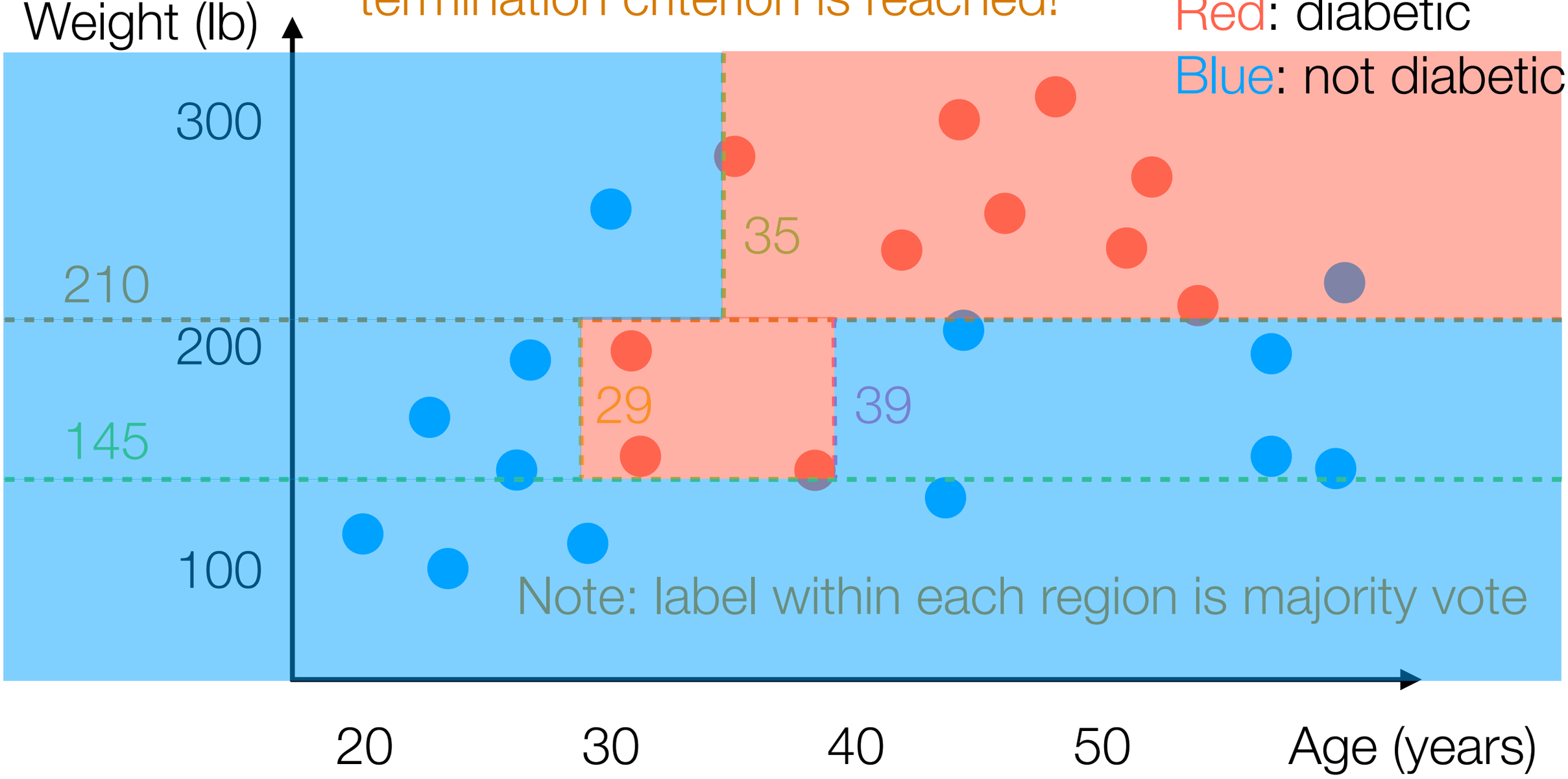
- I'll show one way (that nobody actually uses in practice) but it's easy to explain

# Learning a Decision Tree



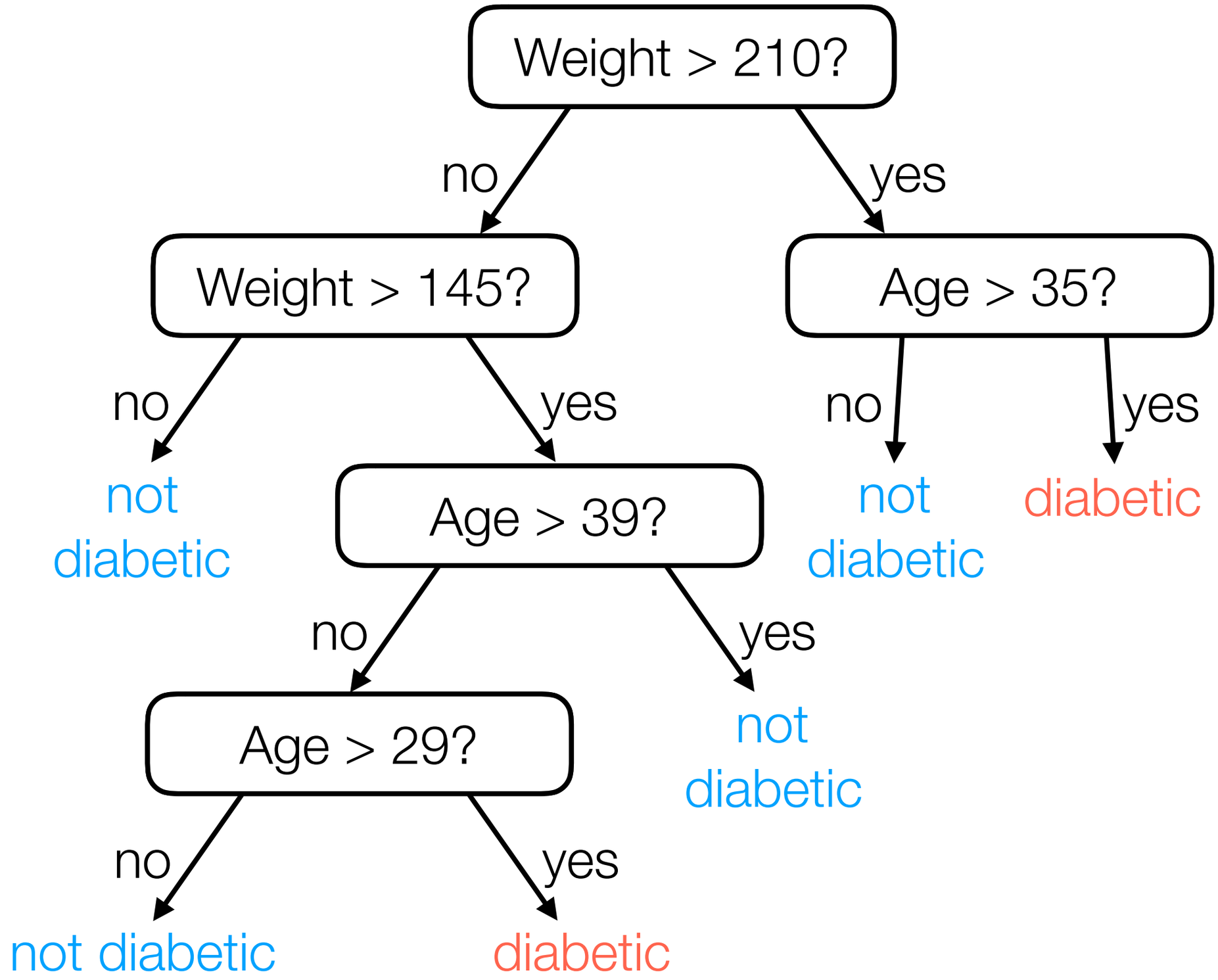
# Learning a Decision Tree

Within each side, recurse until a termination criterion is reached!




Example termination criteria: ≥90% points within region has same label, number of points within region is <5

# Decision Tree Learned



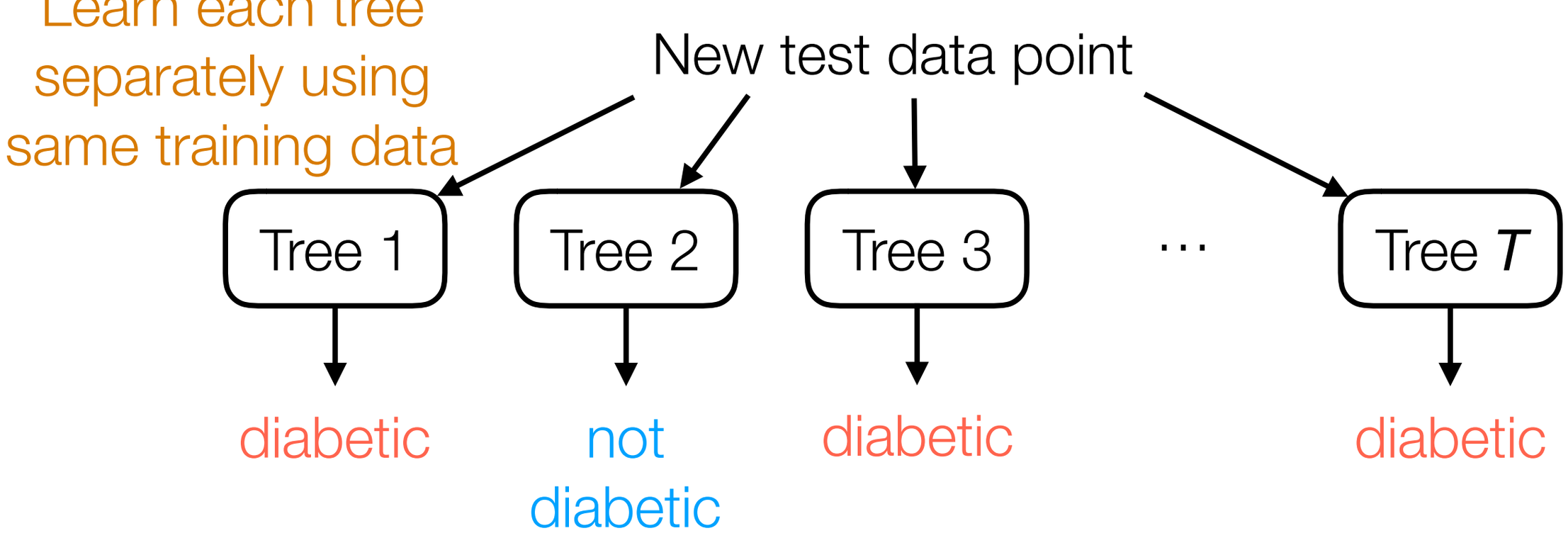

For a new person with feature vector (age, weight), easy to predict!

# Decision Forest for Classification

- Typically, a decision tree is learned with randomness (e.g., we randomly chose which feature to threshold)

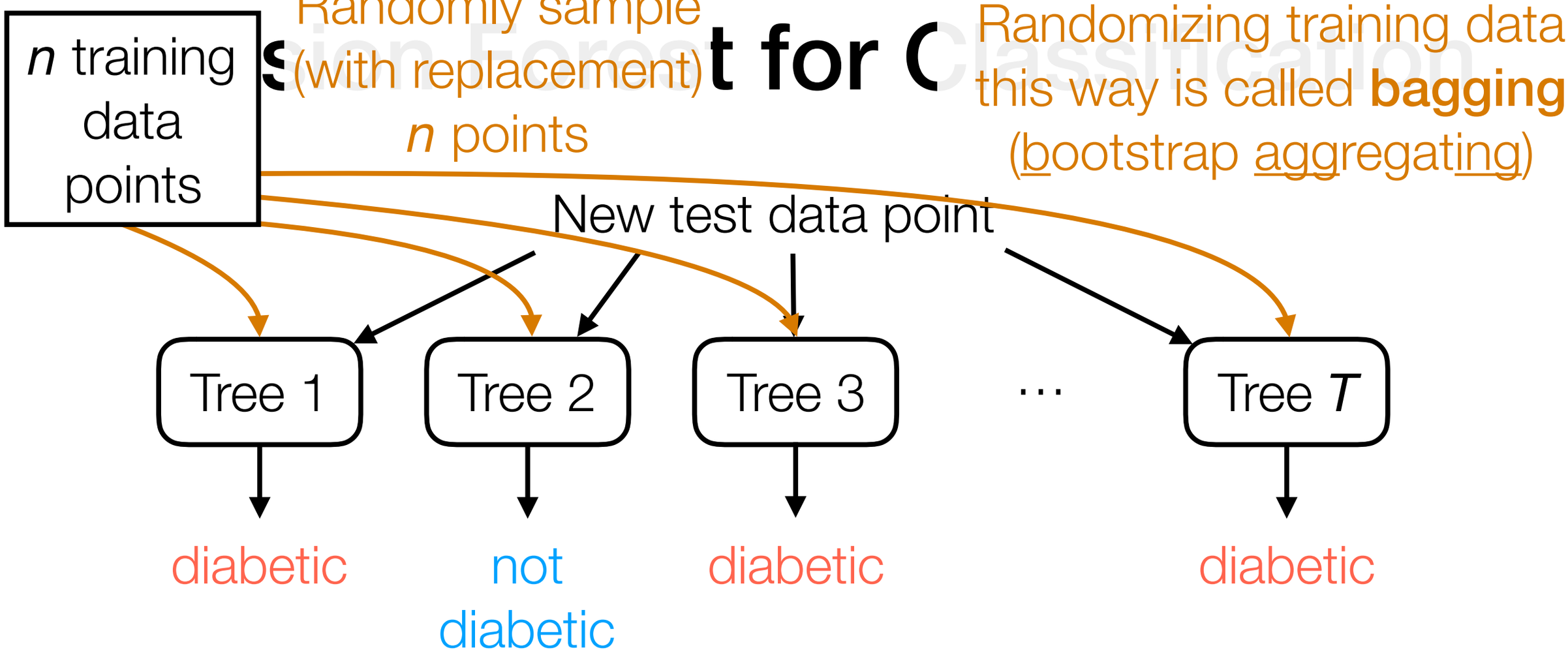
  ➔ by re-running the same learning procedure, we can get different decision trees that make different predictions!

- For a more stable prediction, use many decision trees




**Final prediction:** majority vote of the different trees' predictions

# Random Forest for Classification




**Question:** What happens if all the trees are the same?

*Adding randomness can make trees more different!*

- **Random Forest:** in addition to randomly choosing features to threshold, also randomize training data used for each tree
- **Extremely randomized trees:** further randomize thresholds rather than trying to pick clever thresholds

# Back to the demo

# Neural Nets and Deep Learning

IMAGENET

Over 10 million images, 1000 object classes

2011: Traditional computer vision achieves accuracy ~74%

2012: Initial deep neural network approach accuracy ~84%

2015 onwards: Deep learning achieves accuracy 96%+

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

# Deep Learning Takeover

Academia:

- Top computer vision conferences (CVPR, ICCV, ECCV) are now nearly all about deep learning
- Top machine learning conferences (ICML, NeurIPS) have *heavily* been taken over by deep learning

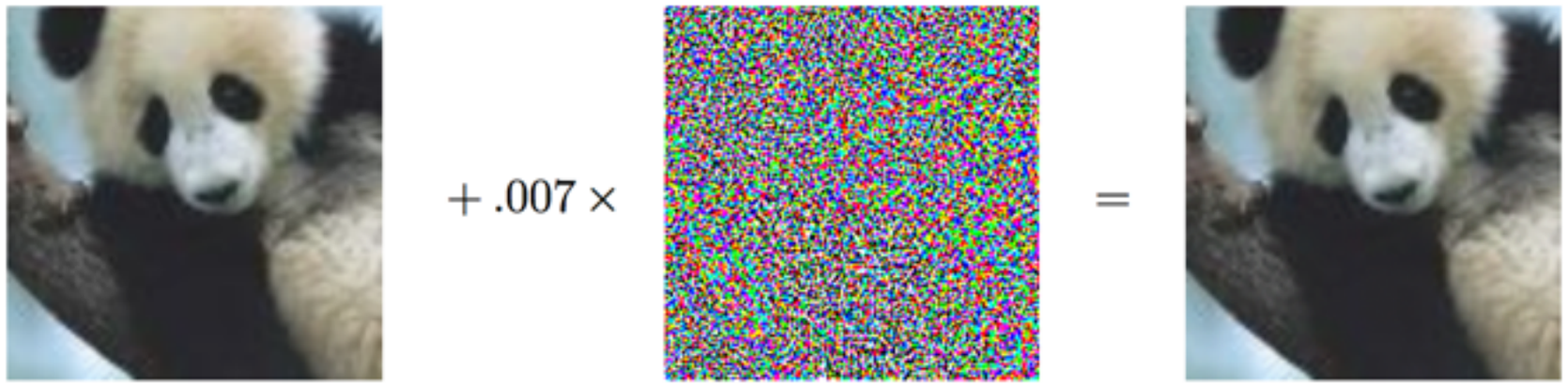
Heavily dominated by industry now!

Extremely useful in practice:


- Near human level image classification (including handwritten digit recognition)
- Near human level speech recognition
- Improvements in machine translation, text-to-speech
- Self-driving cars
- *Better* than humans at playing Go

Google DeepMind’s AlphaGo vs Lee Sedol, 2016

# Is it all hype?

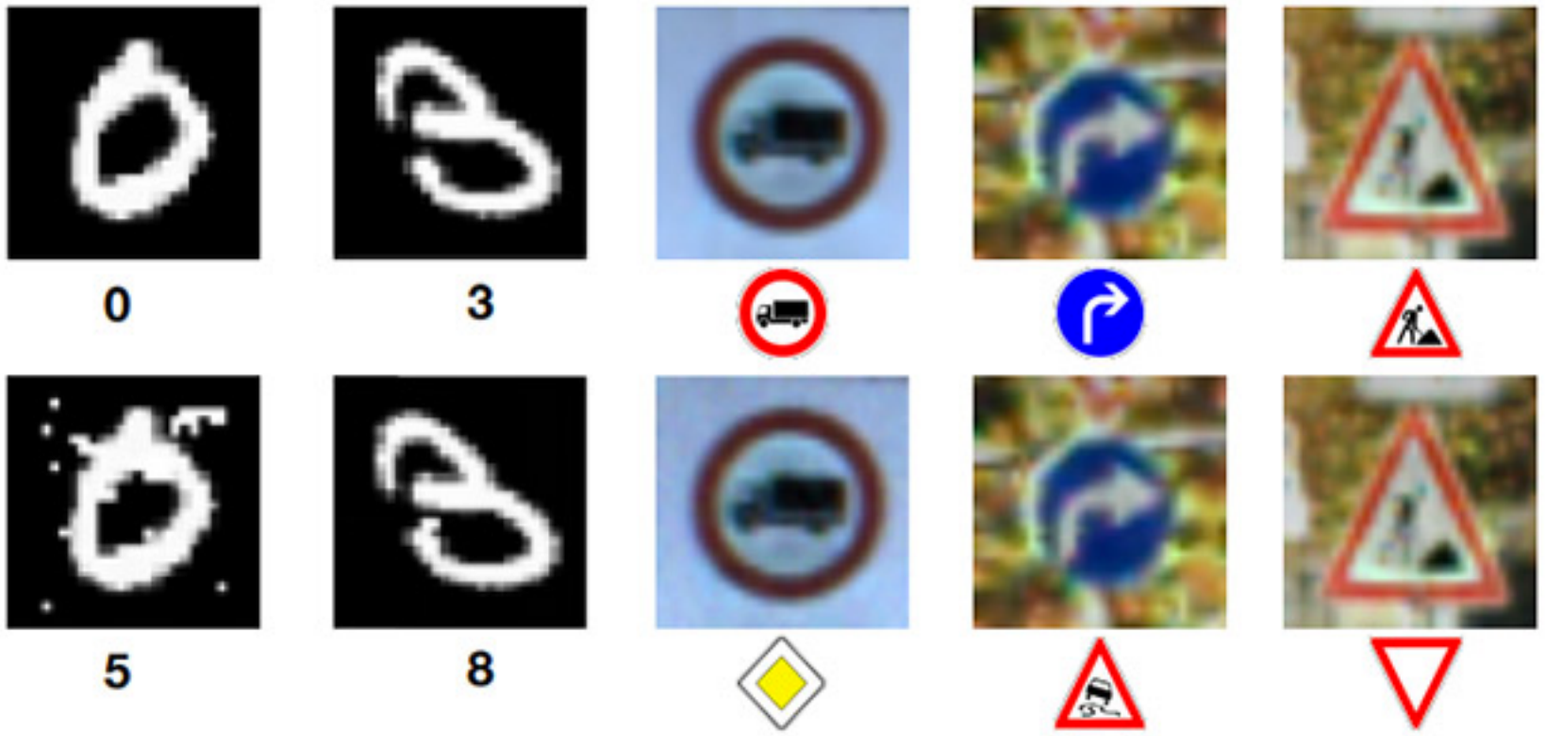



panda
~58% confidence

adversarial
noise

gibbon
~99% confidence

Source: Goodfellow, Shlens, and Szegedy. Explaining and Harnessing Adversarial Examples. ICLR 2015.




Source: Papernot et al. Practical Black-Box Attacks against Machine Learning. Asia Conference on Computer and Communications Security 2017.

# Fooling Neural Networks in the Physical World with 3D Adversarial Objects

31 Oct 2017 · 3 min read — shared on Hacker News, Lobsters, Reddit, Twitter

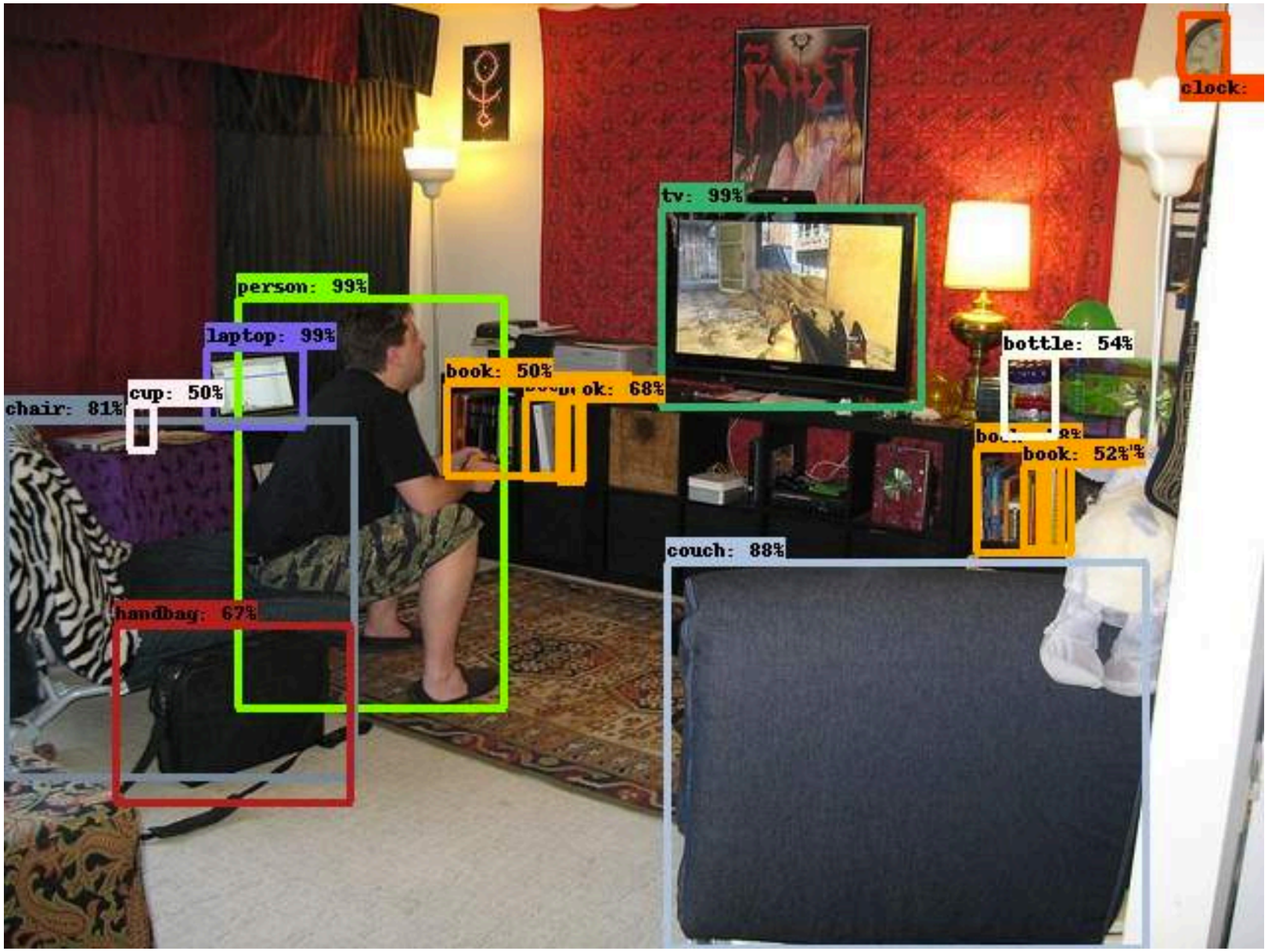
We've developed an approach to generate *3D adversarial objects* that reliably fool neural networks in the real world, no matter how the objects are looked at.


Neural network based classifiers reach near-human performance in many tasks, and they're used in high risk, real world systems. Yet, these same neural networks are particularly vulnerable to *adversarial examples*, carefully perturbed inputs that cause

Source: labsix

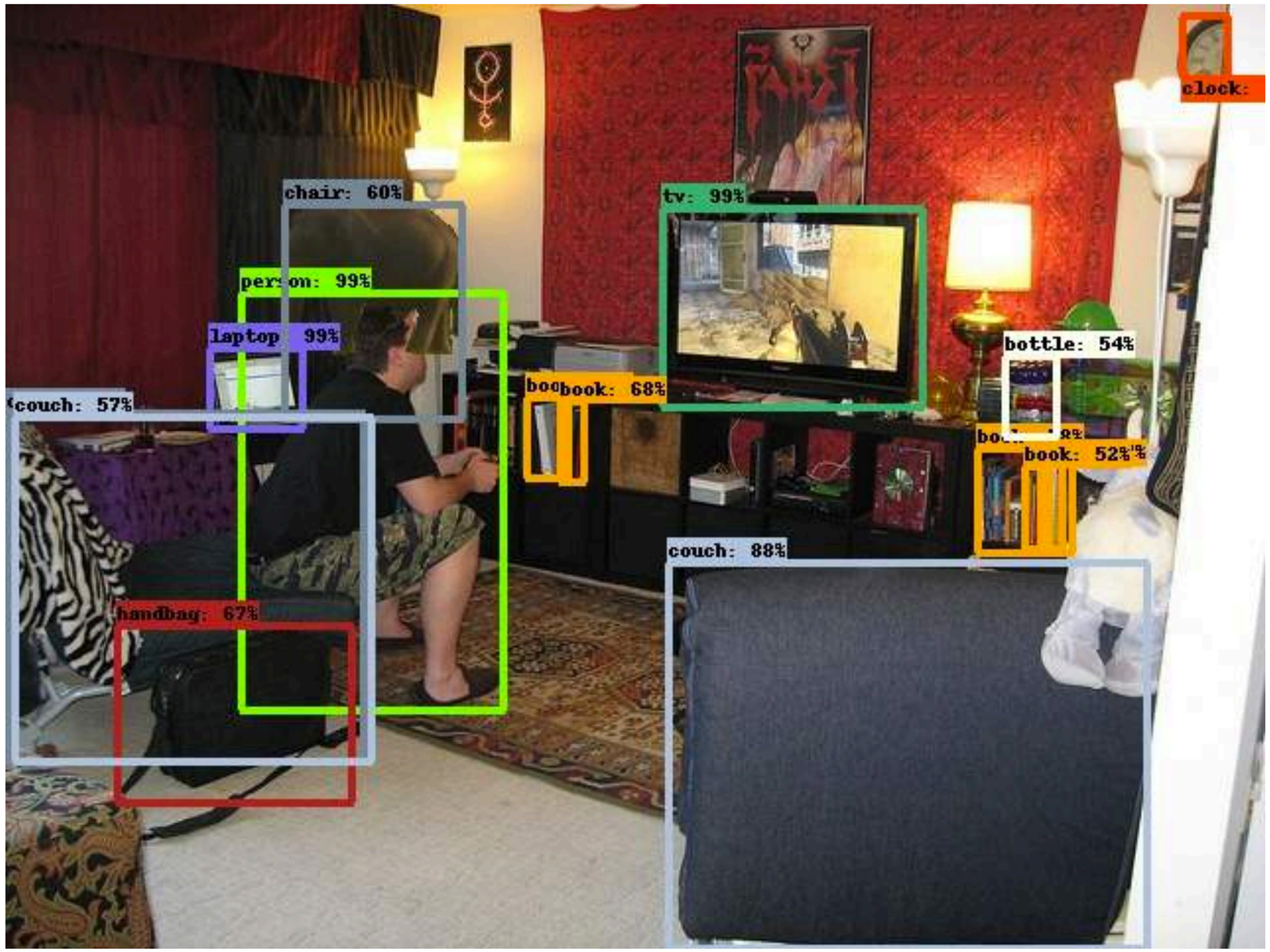

Source: Gizmodo article “This Neural Network's Hilariously Bad Image Descriptions Are Still Advanced AI”. September 16, 2015. (They’re using the NeuralTalk image-to-caption software.)

# Slightly modifying an image results in different prediction results




Source: Quanta Magazine article “Machine Learning Confronts the Elephant in the Room”. September 20, 2018.

# Slightly modifying an image results in different prediction results




Source: Quanta Magazine article “Machine Learning Confronts the Elephant in the Room”. September 20, 2018.

# Another AI Winter?

~1970's: First AI winter over symbolic AI

~1980's: Second AI winter over "expert systems"

Every time: Lots of hype, explosion in funding, then bubble bursts

Michael Jordan Follow

Michael I. Jordan is a Professor in the Department of Electrical Engineering and Computer Sciences and the Department of Statistics at UC Berkeley.

Apr 18 · 16 min read

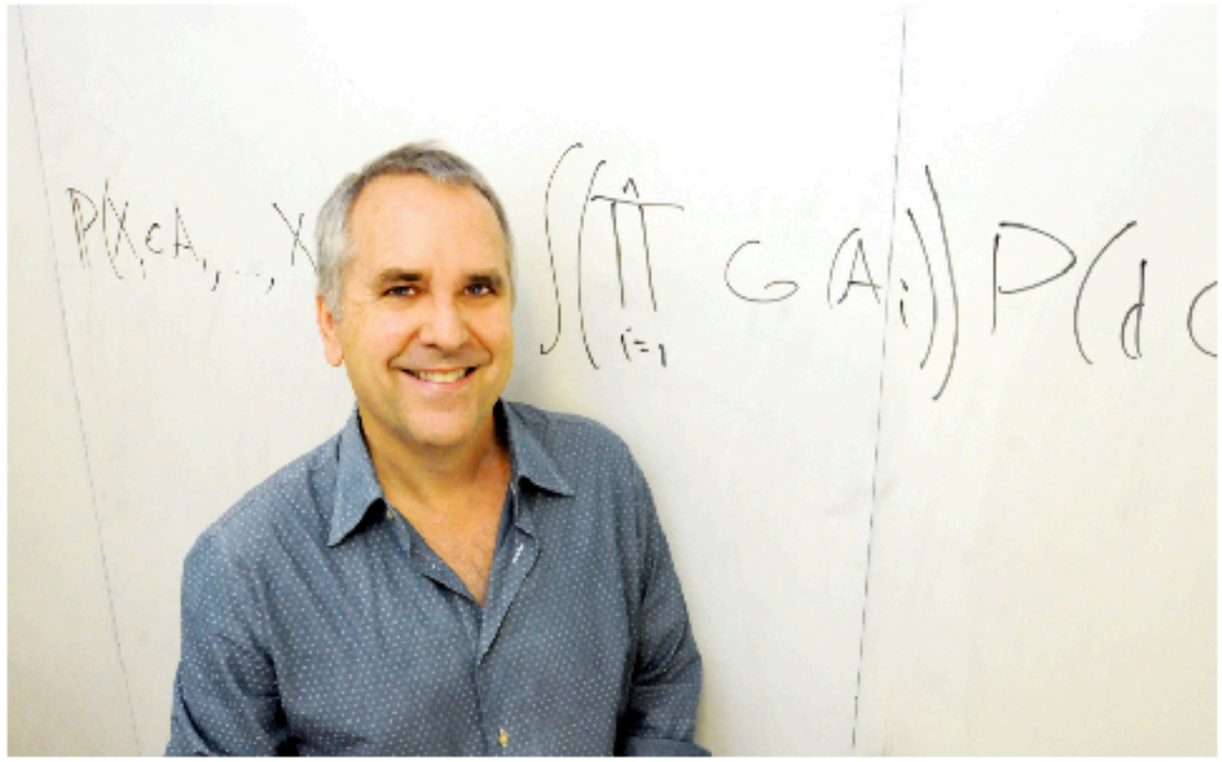


Photo credit: Peg Skorpinski

# Artificial Intelligence—The Revolution Hasn't Happened Yet

Artificial Intelligence (AI) is the mantra of the current era. The phrase is intoned by technologists, academicians, journalists and venture capitalists

https://medium.com/@mijordan3/artificial-intelligence-the-revolution-hasnt-happened-yet-5e1d5812e1e7

TECHNOLOGY

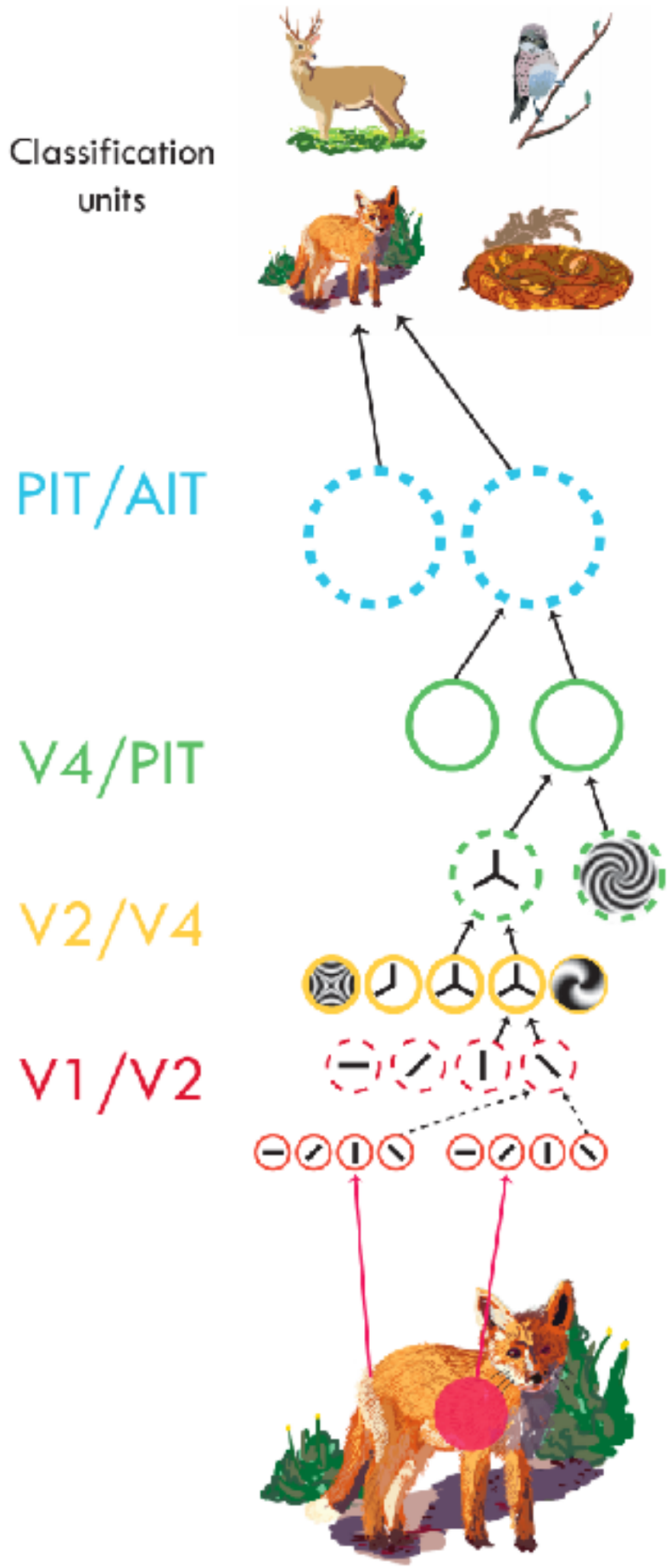
# How a Pioneer of Machine Learning Became One of Its Sharpest Critics

Judea Pearl helped artificial intelligence gain a strong grasp on probability, but laments that it still can't compute cause and effect.

KEVIN HARTNETT AND QUANTA MAY 19, 2018

https://www.theatlantic.com/technology/archive/2018/05/machine-learning-is-stuck-on-asking-why/560675/?single_page=true

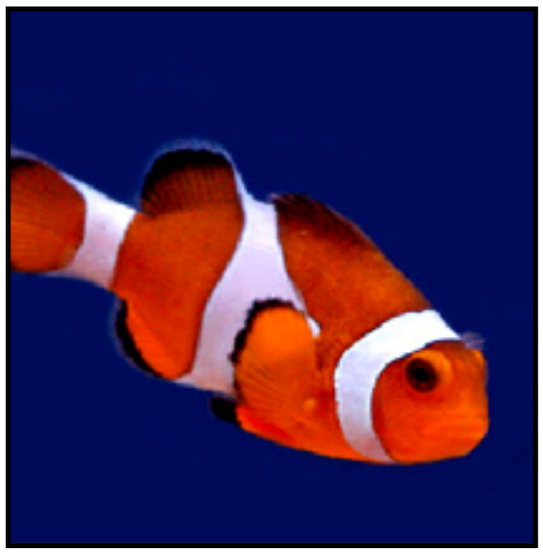
# What is deep learning?




Slide by Phillip Isola

Serre, 2014

# Basic Idea

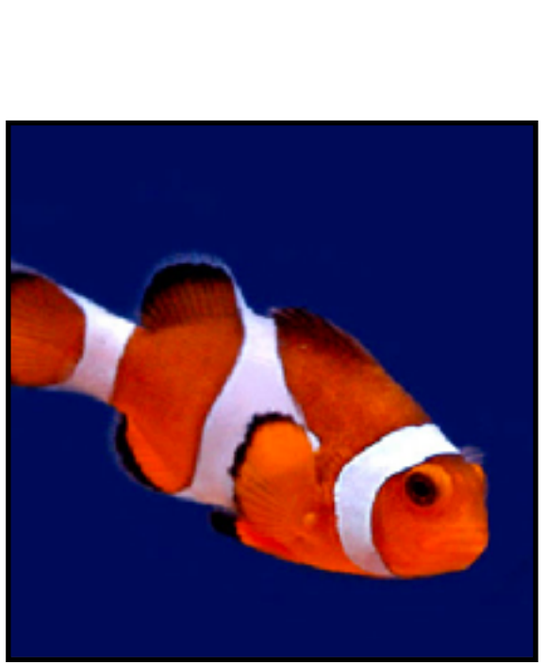


Brain/Machine

“clown fish”

Slide by Phillip Isola

# Object Recognition



Edges

Texture

Colors

Segments

Parts

“clown fish”

Feature extractors

Classifier

Slide by Phillip Isola

# Object Recognition

Learned

Edges

Texture

Colors

Segments

Parts

“clown fish”

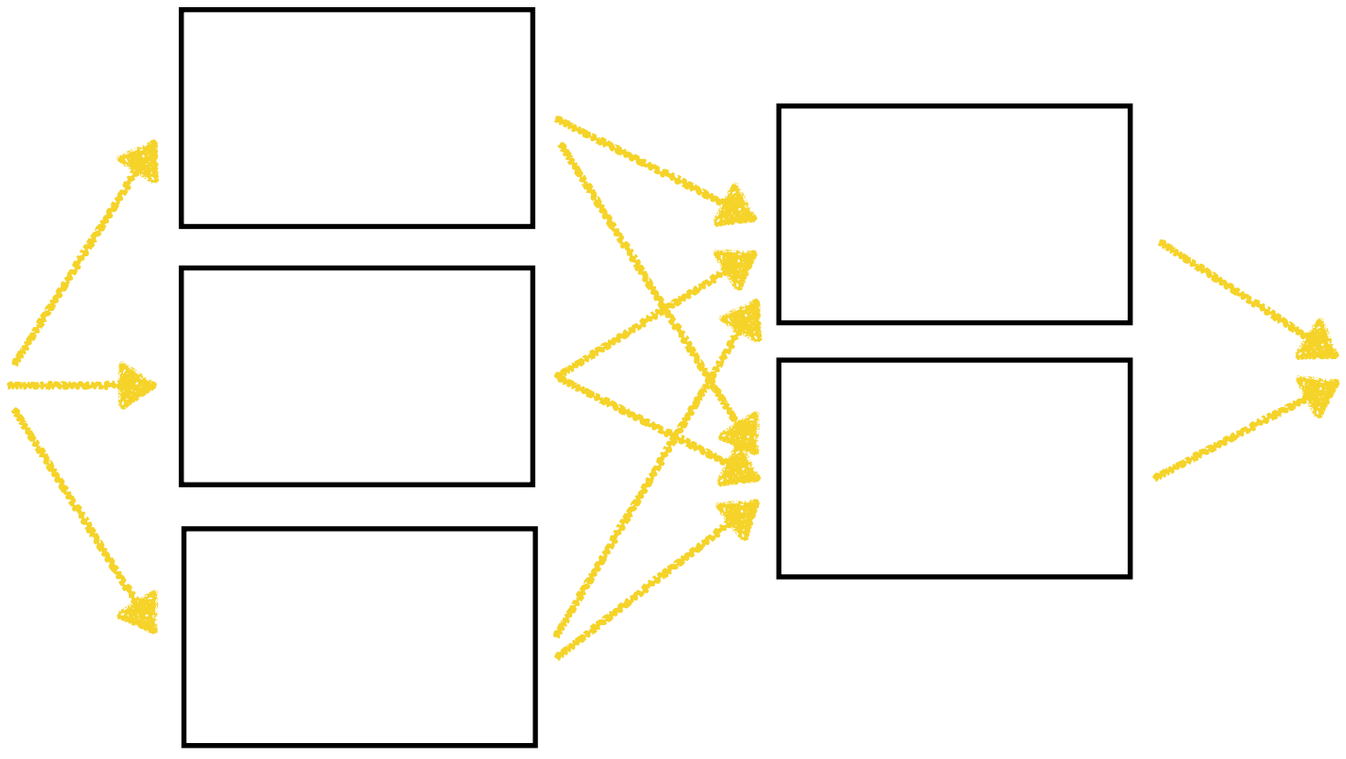
Feature extractors

Classifier

Slide by Phillip Isola

# Neural Network

Learned

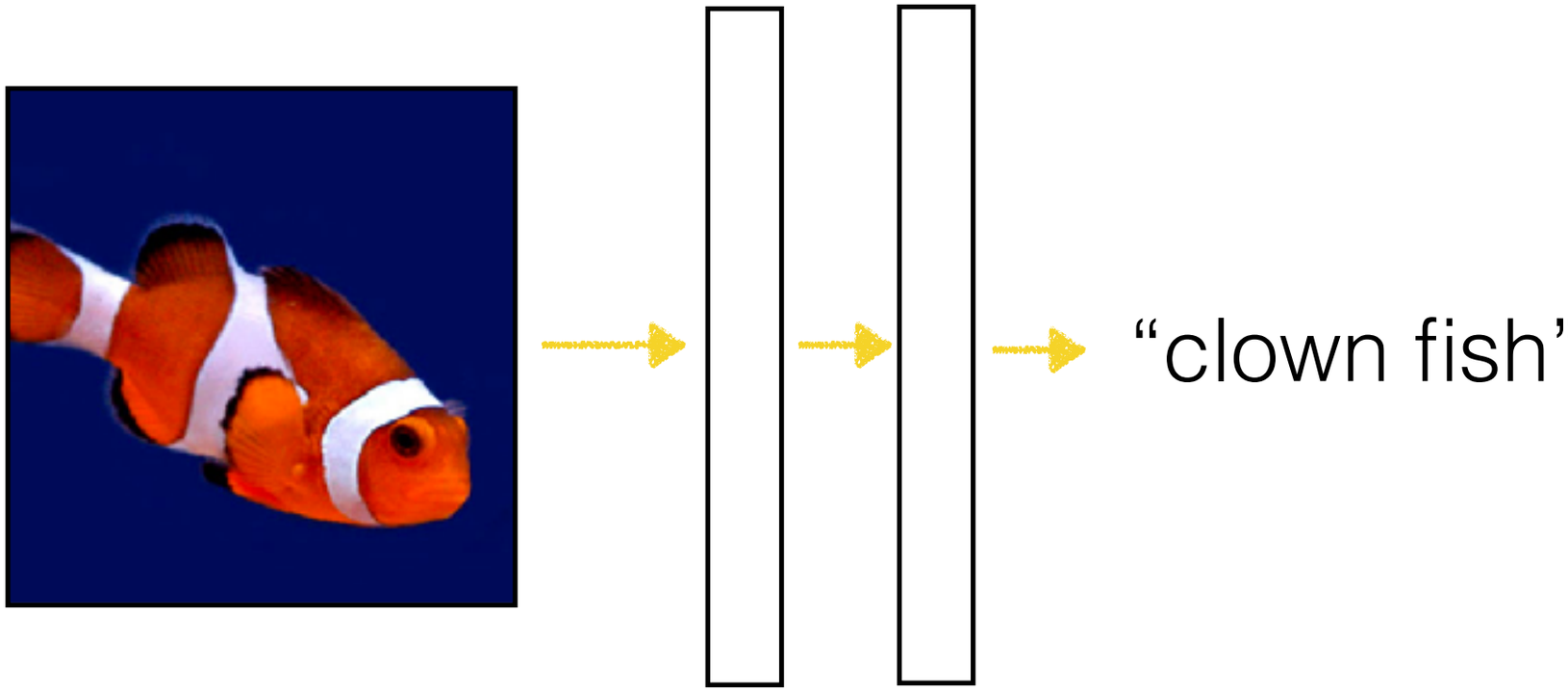


“clown fish”

Slide by Phillip Isola

# Neural Network

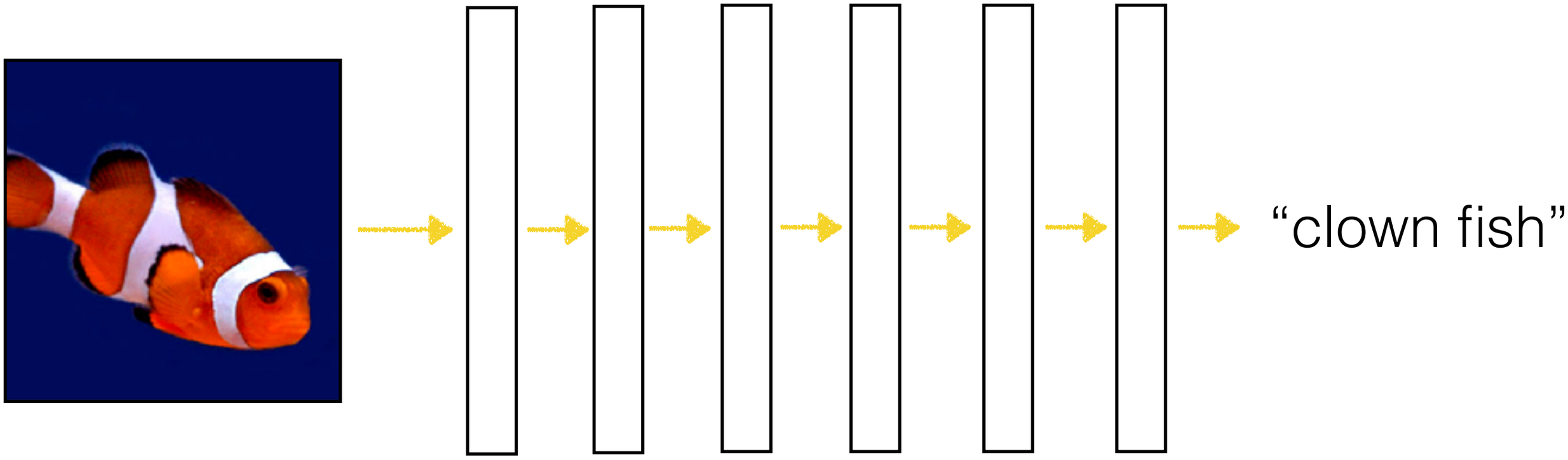
Learned




Slide by Phillip Isola

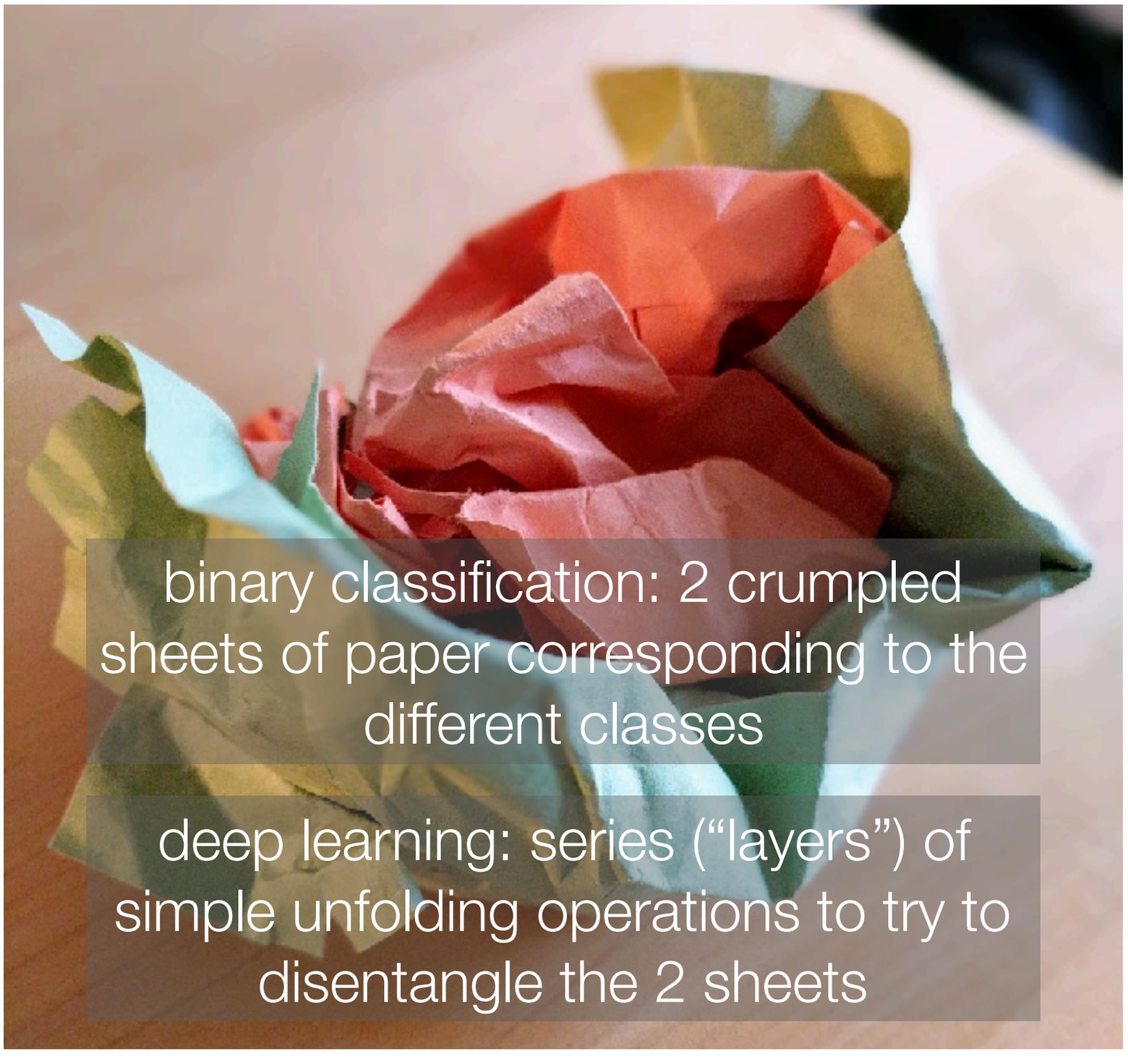
# Deep Neural Network

Learned




Slide by Phillip Isola

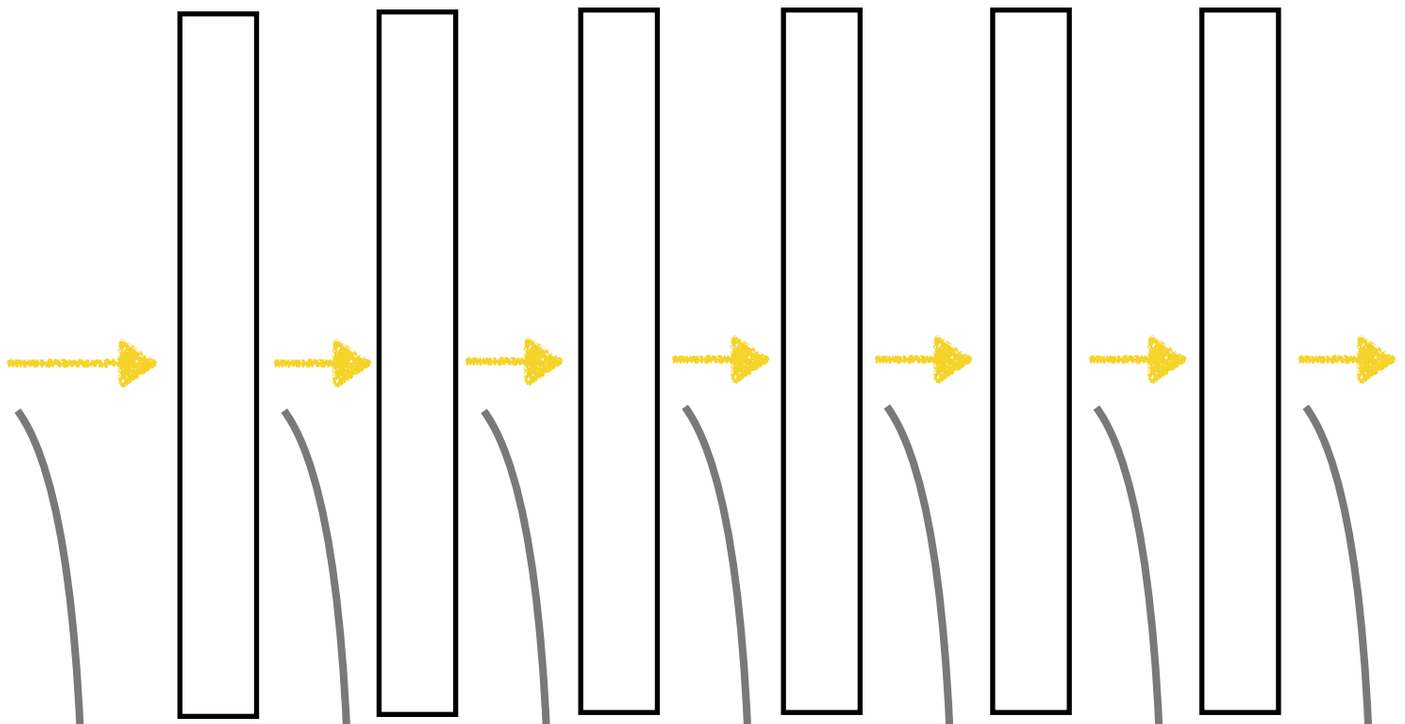
# Crumpled Paper Analogy




Analogy: Francois Chollet, photo: George Chen

# Representation Learning

Each layer's output is *another way we could represent the input data*



Learned

"clown fish"

Visualize (e.g., t-SNE)

Visualize

Visualize

Visualize

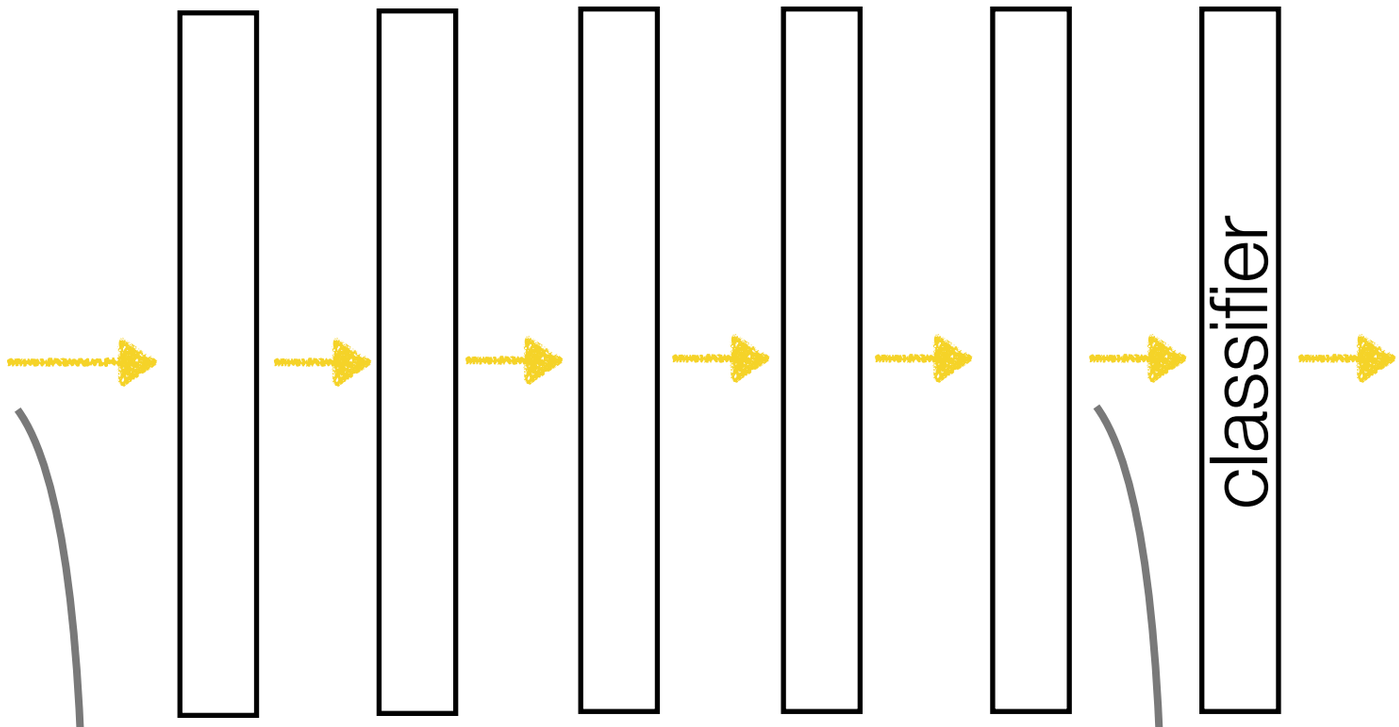
Visualize

Visualize

Visualize

# Representation Learning

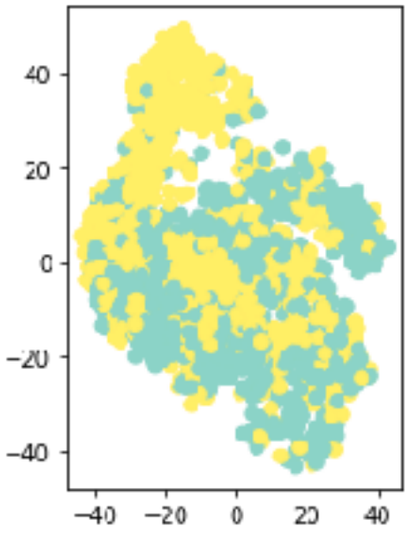
Each layer's output is *another way we could represent the input data*



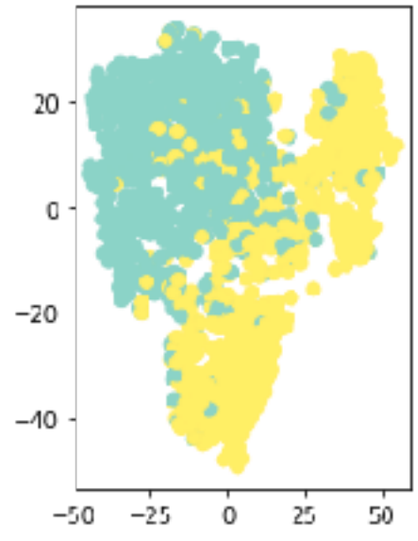

Learned

"clown fish"

Visualize (e.g., t-SNE)




Visualize



# Why Does Deep Learning Work?

Actually the ideas behind deep learning are old (~1980's)

- Big data

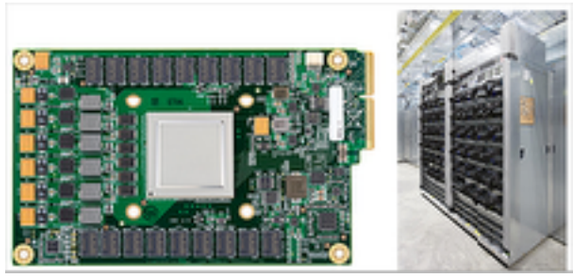
amazon.com NETFLIX fitbit lyft UPMC LIFE CHANGING MEDICINE

- Better hardware

intel AMD

CPU's
& Moore's law


GPU's



TPU's

- Better algorithms

# Structure Present in Data Matters

Neural nets aren't doing black magic

- **Image analysis:** convolutional neural networks (convnets) neatly incorporates basic image processing structure

- **Time series analysis:** recurrent neural networks (RNNs) incorporates ability to remember and forget things over time

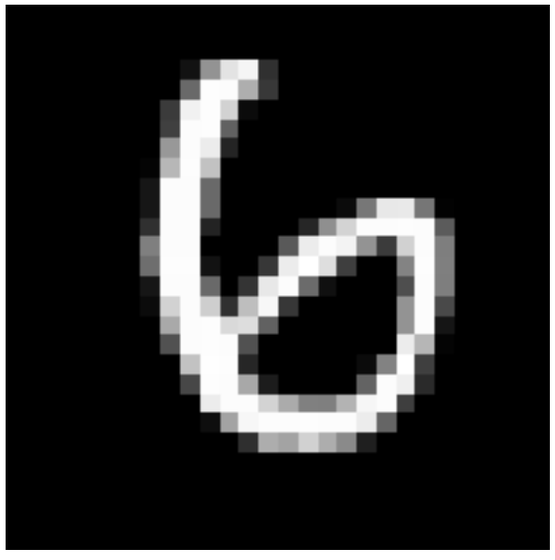
  - Note: text is a time series

  - Note: video is a time series

# Handwritten Digit Recognition Example

Walkthrough of building a 1-layer and then a 2-layer neural net

# Handwritten Digit Recognition



28x28 image

flatten & treat as 1D vector

length 784 vector (784 input neurons)

weighted sums

(parameterized by a weight matrix ***W*** and a bias ***b***)

“dense” layer with 10 numbers

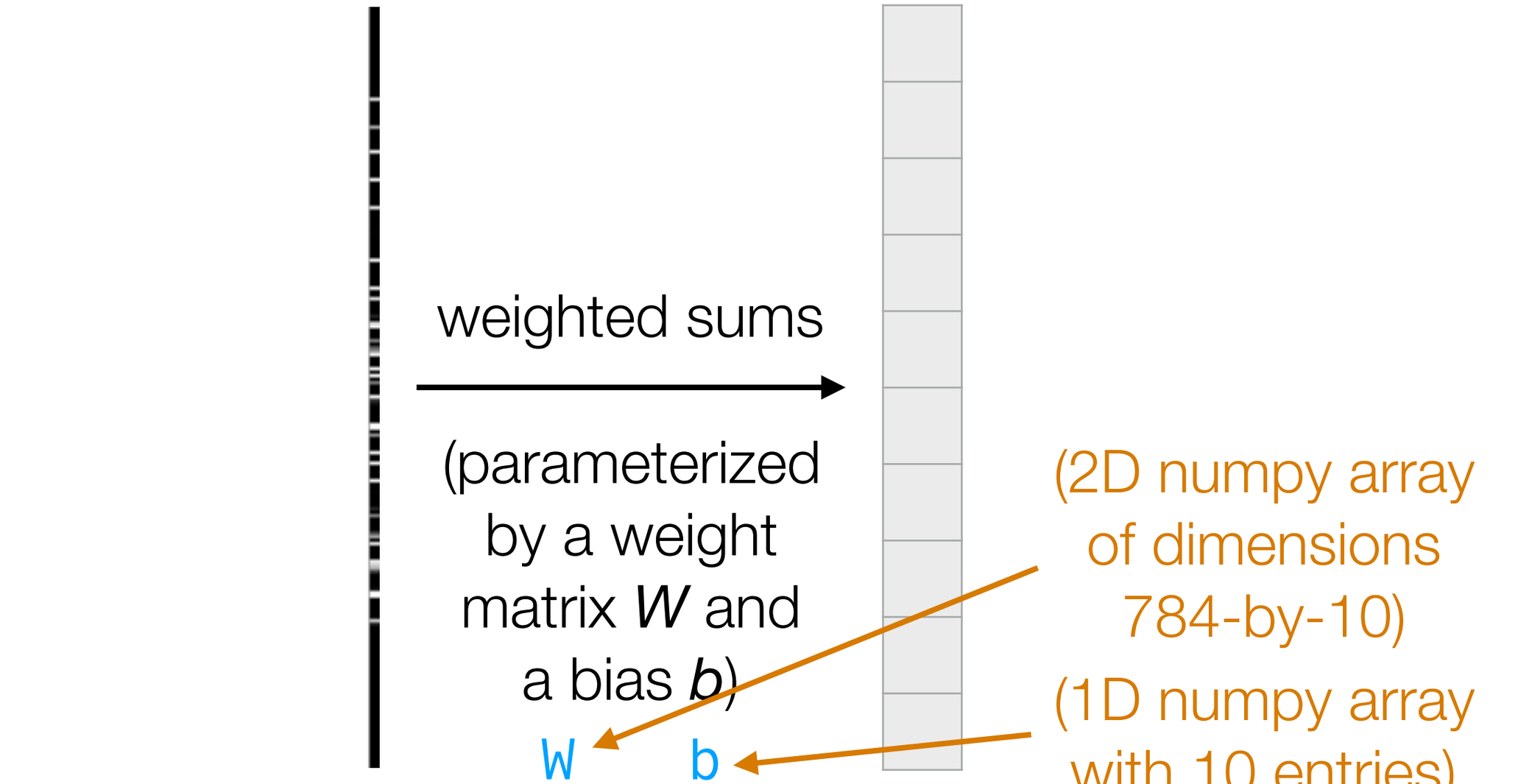
activation

(can be thought of as post-processing)

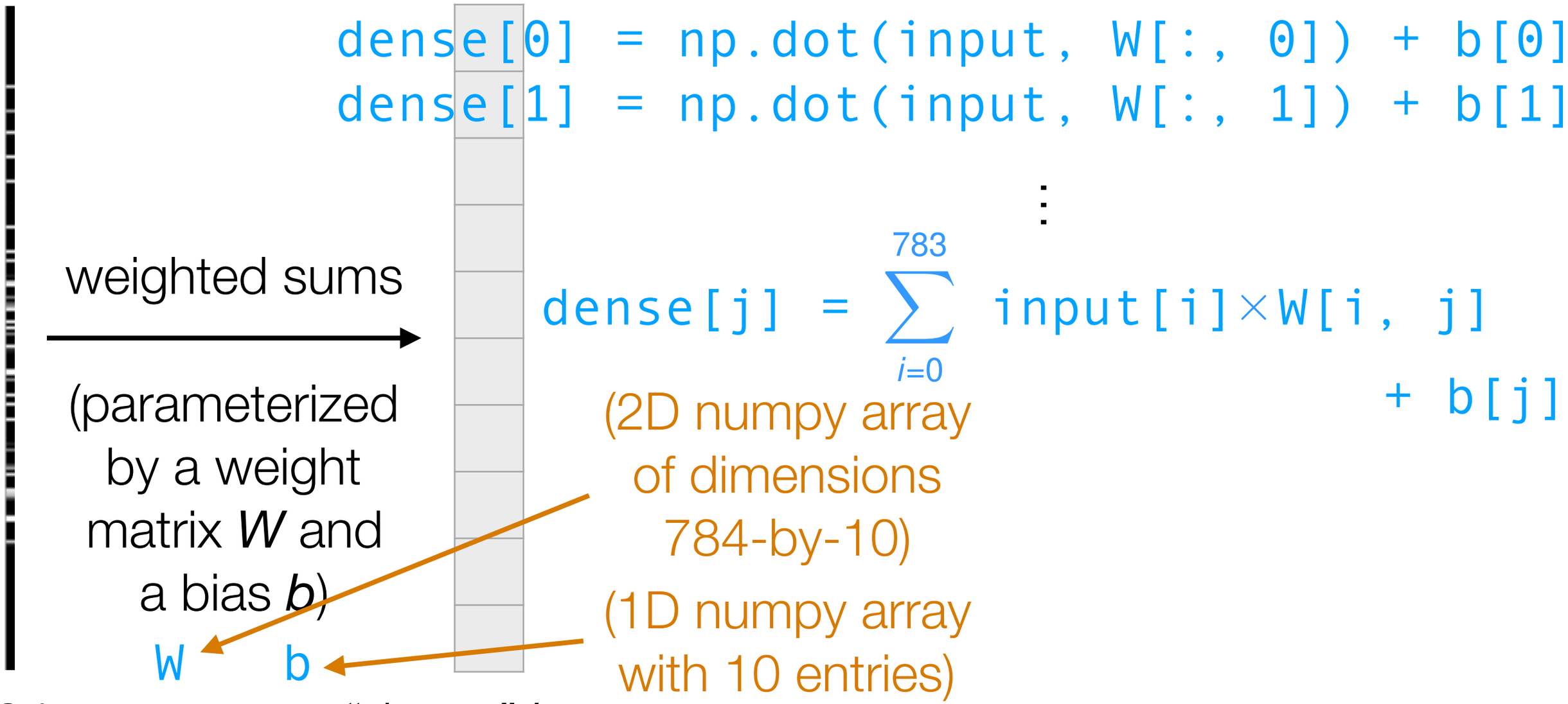
“dense” layer final output

single “dense” layer with 10 neurons

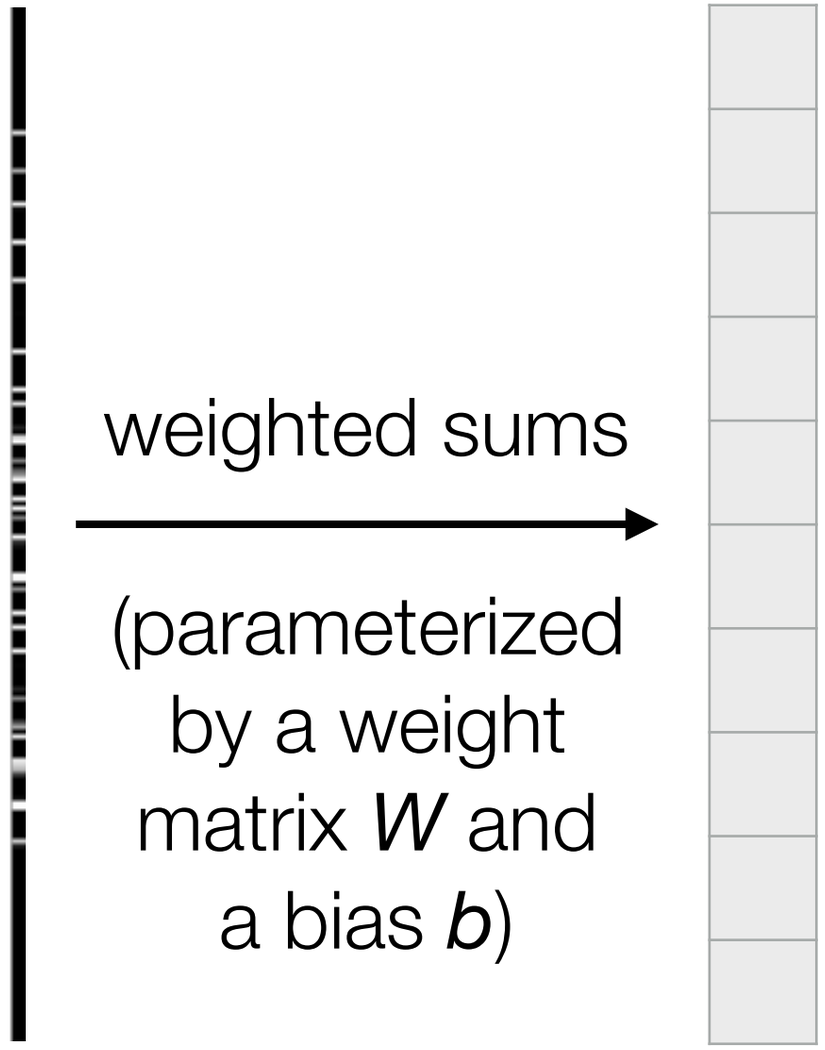
# Handwritten Digit Recognition



# Handwritten Digit Recognition



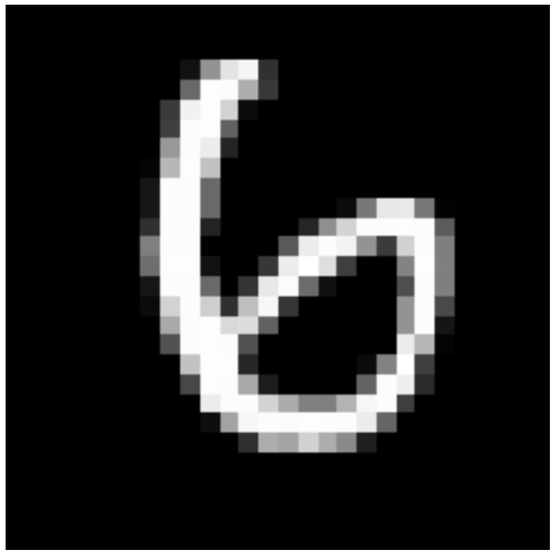
# Handwritten Digit Recognition




length 784 vector
(784 input neurons)

“dense” layer
with 10 numbers

# Handwritten Digit Recognition



28x28 image

flatten & treat as 1D vector

length 784 vector (784 input neurons)

weighted sums

(parameterized by a weight matrix ***W*** and a bias ***b***)

“dense” layer with 10 numbers

activation

(can be thought of as post-processing)

“dense” layer final output

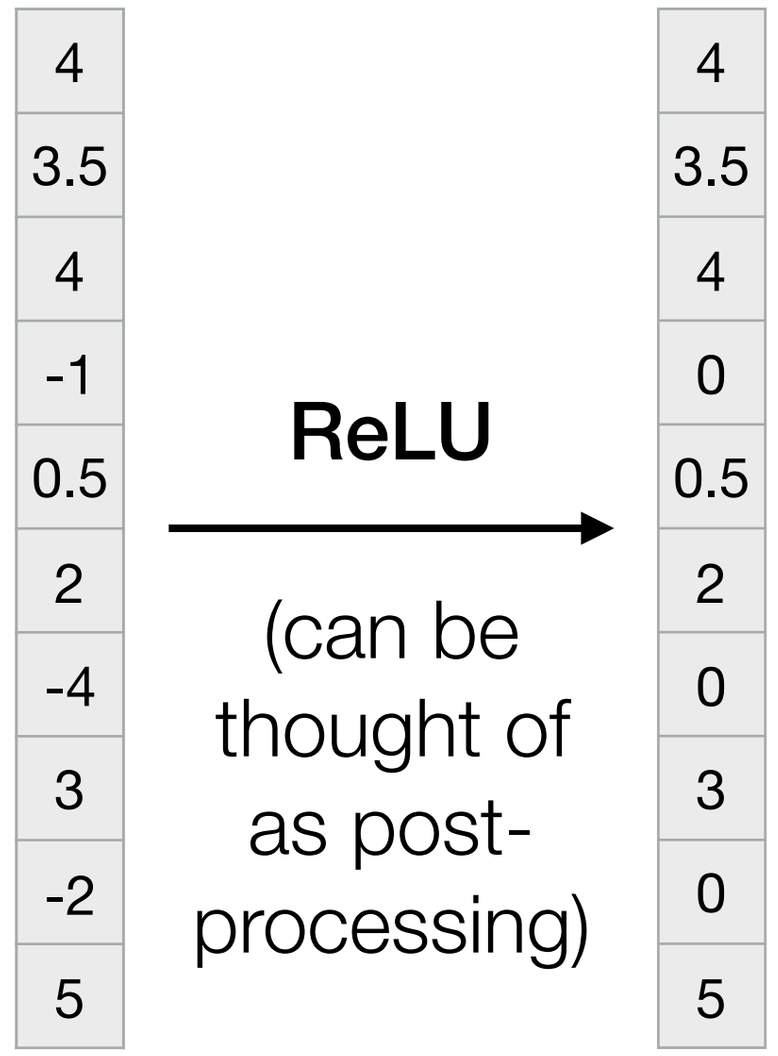
single “dense” layer with 10 neurons

# Handwritten Digit Recognition

Many different activation functions possible

Example: **Rectified linear unit (ReLU)**
zeros out entries that are negative

```
dense_final = np.maximum(0, dense)
```




“dense” layer with 10 numbers

`dense`

“dense” layer final output

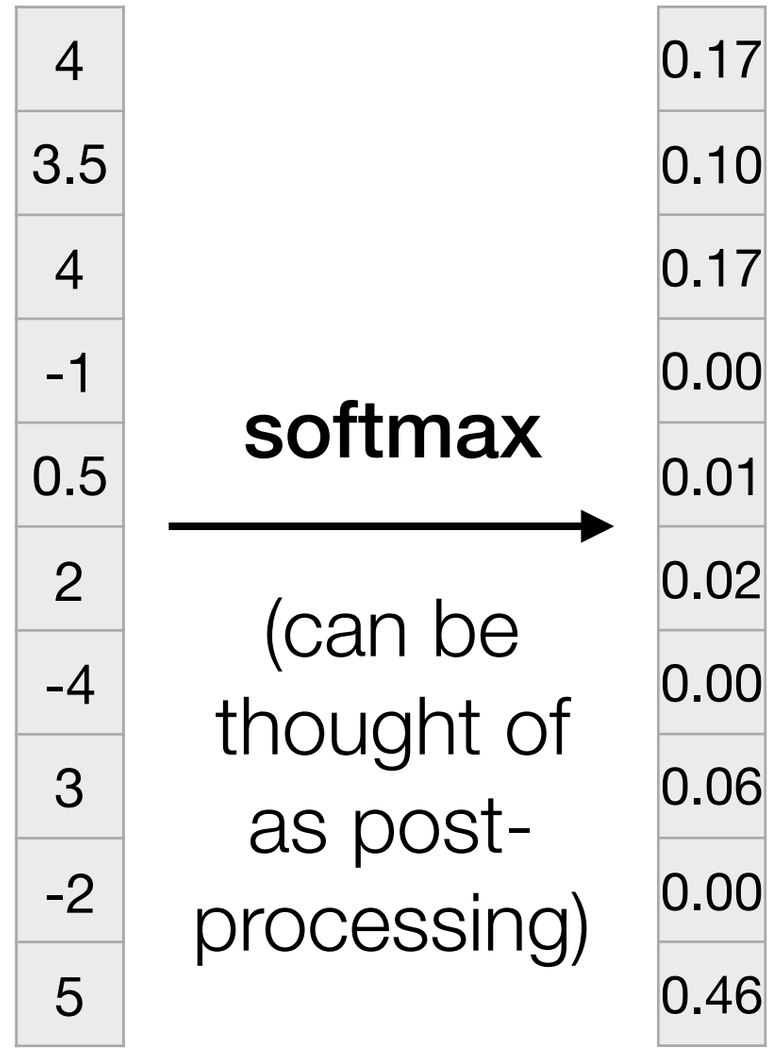
`dense_final`

# Handwritten Digit Recognition

Many different activation functions possible

Example: **softmax** turns the entries in the dense layer (prior to activation) into a probability distribution (using the “softmax” transformation)

```
dense_exp = np.exp(dense)
dense_exp /= np.sum(dense_exp)
dense_final = dense_exp
```



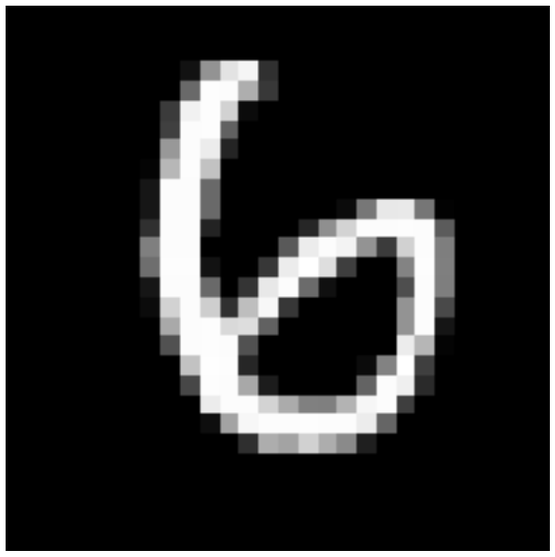

“dense” layer with 10 numbers

`dense`

“dense” layer final output

`dense_final`

# Handwritten Digit Recognition



28x28 image

flatten & treat as 1D vector

length 784 vector (784 input neurons)

weighted sums

(parameterized by a weight matrix *W* and a bias *b*)

"dense" layer with 10 numbers

**softmax**

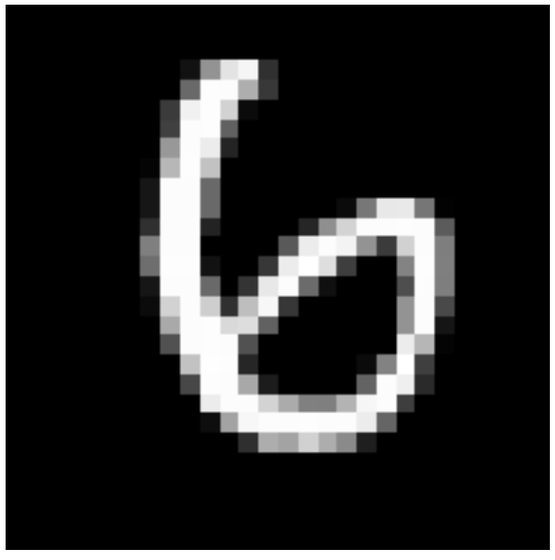
(can be thought of as post-processing)

Pr(digit 0)
Pr(digit 1)
Pr(digit 2)
Pr(digit 3)
Pr(digit 4)
Pr(digit 5)
Pr(digit 6)
Pr(digit 7)
Pr(digit 8)
Pr(digit 9)

"dense" layer final output

single "dense" layer with 10 neurons

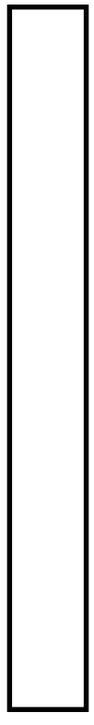
# Handwritten Digit Recognition



28x28 image

flatten &
treat as
1D vector

length 784 vector
(784 input neurons)



dense layer with
10 neurons,
softmax activation,
parameters $W$, $b$

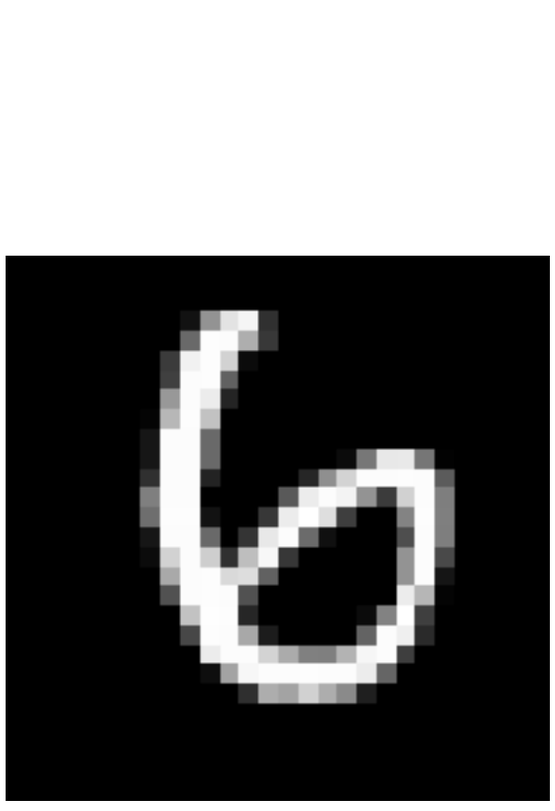
We want the output of the dense layer to encode probabilities for whether the input image is a 0, 1, 2, …, 9

*but as of now we aren't providing any sort of information to enforce this*

# Handwritten Digit Recognition

Demo part 1

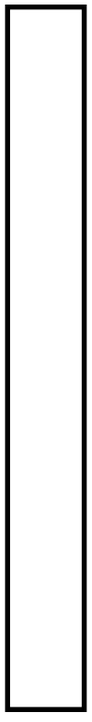
# Handwritten Digit Recognition



28x28 image

flatten &
treat as
1D vector

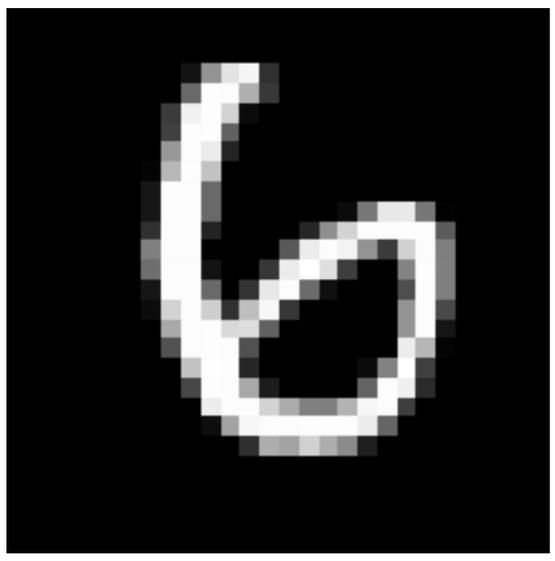
length 784 vector
(784 input neurons)



dense layer with
10 neurons,
softmax activation,
parameters $W$, $b$

# Handwritten Digit Recognition

Training label: 6

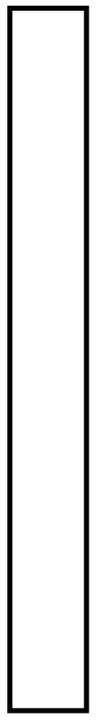


28x28 image

Learning this neural net means learning *W* and *b*

flatten & treat as 1D vector

length 784 vector (784 input neurons)



dense layer with 10 neurons, softmax activation, parameters *W*, *b*

Loss/“error” → error

Error is averaged across training examples

Popular loss function for classification (> 2 classes): **categorical cross entropy**

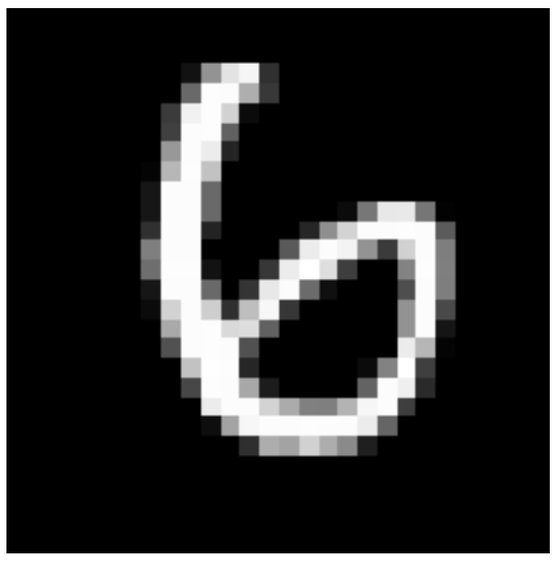
$$\log \frac{1}{\Pr(\text{digit 6})}$$

# Handwritten Digit Recognition

Demo part 2

# Handwritten Digit Recognition

Training label: 6

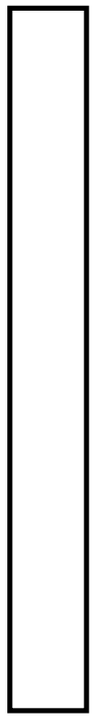


28x28 image

Learning this neural net means learning *W* and *b*

flatten & treat as 1D vector

length 784 vector (784 input neurons)



dense layer with 10 neurons, softmax activation, parameters *W*, *b*

Loss/“error” → error

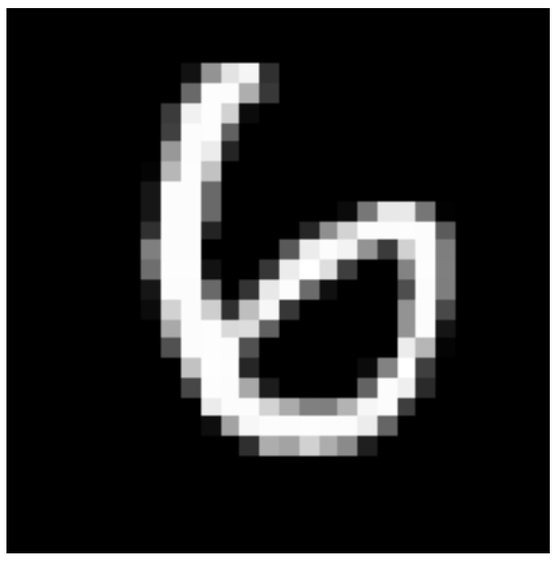
Error is averaged across training examples

Popular loss function for classification (> 2 classes): **categorical cross entropy**

$$\log \frac{1}{\Pr(\text{digit 6})}$$

# Handwritten Digit Recognition

Training label: 6



28x28 image

length 784 vector
(784 input neurons)

dense layer
with 512
neurons, ReLU
activation

dense layer with
10 neurons,
softmax activation

Learning this neural net means learning parameters of both dense layers!

Error is averaged across training examples

Loss/“error”

error

Popular loss function for classification (> 2 classes):
**categorical cross entropy**

$$\log \frac{1}{\Pr(\text{digit 6})}$$

# Handwritten Digit Recognition

Demo part 3