Many Many Models for Classification

Introduction to Data Science

Kigali, Rwanda

July 10th, 2019

Outline

1. Polynomial Logistic Regression
2. Decision Trees
3. Bias and Variance
4. Ensembling:
    - Random Forests
    - AdaBoost
5. Neural Networks
6. Image Data

Logistic Regression with Polynomial Boundaries

Linear Decision Boundaries

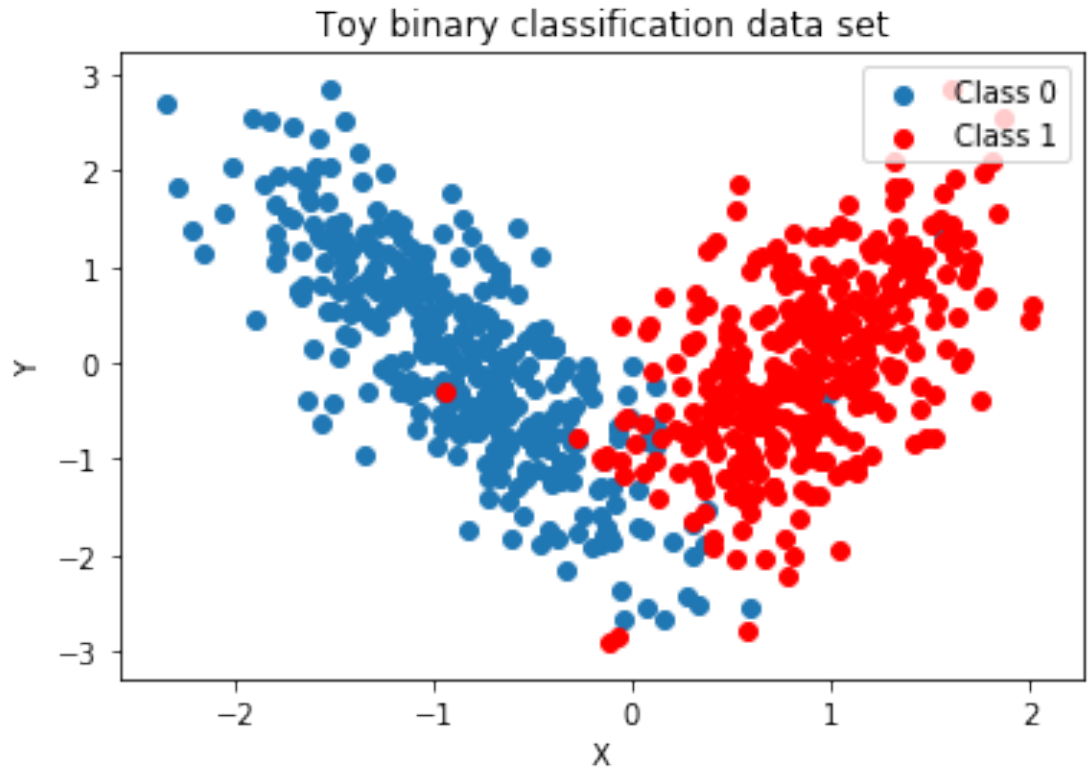
When the decision boundary is linear, it is defined by the equation

$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

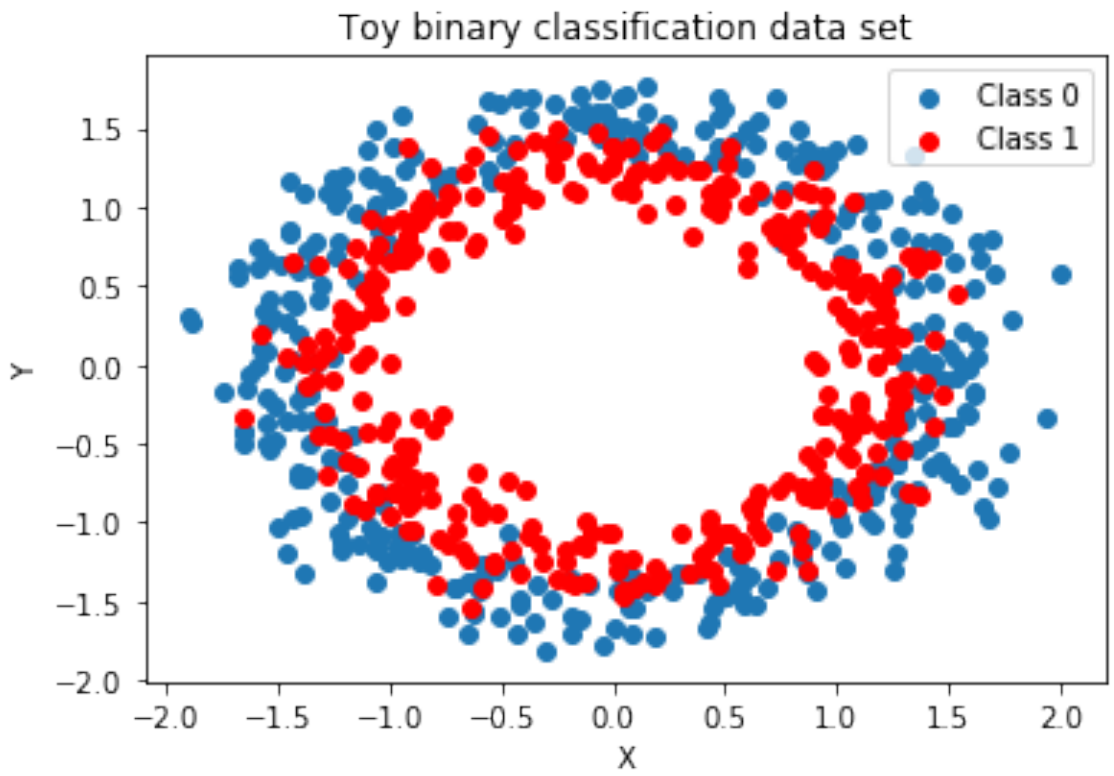
$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

where $x_0 = 1 x_0 = 1$. Often we write $b = w_0 x_0 \, b = w_0 x_0$ and we call $b b$ the ***bias*** term.

The vector $\mathbf{w} \mathbf{w}$ allow us to gauge the 'distance' of a point from the decision boundary



How would you parametrize a ellipitical decision boundary?



We can say that the decision boundary is given by a ***quadratic function*** of the input:

$$w_1 x_1^2 + w_2 x_2^2 + w_3 = 0$$

$$w_1 x_1^2 + w_2 x_2^2 + w_3 = 0$$

We say that we can fit such a decision boundary using logistic regression with degree 2 polynomial features

Logistic Regression with Quadratic Decision Boundary

Recall that polynomial regression is simply a linear regression fit on polynomial features of the inputs `x`:

1. transform xx into [1, xx, x^2x^2, x^3x^3, x^4x^4, ... etc]
2. fit linear regression on the polynomial features [1, xx, x^2x^2, x^3x^3, x^4x^4, ... etc]

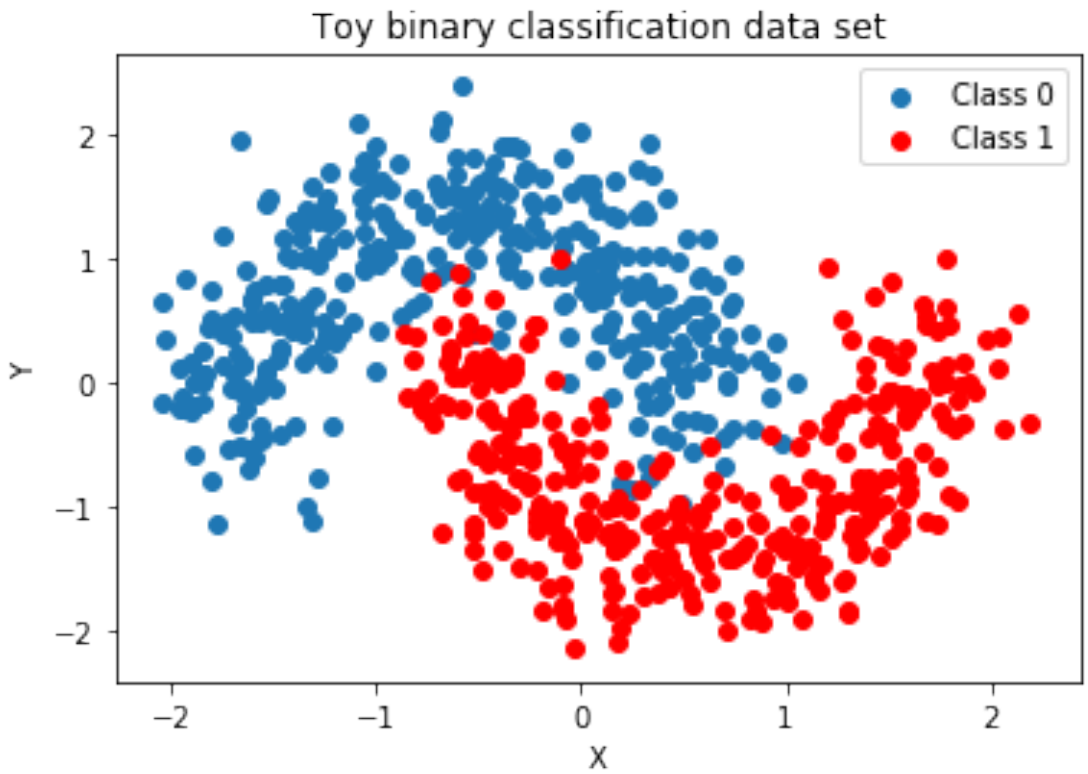
Similary, if we want to fit a logistic regression with quadratic features, we:

1. transform xx into [1, xx, x^2x^2]
2. fit logistic regression on the quadratic features [1, xx, x^2x^2]

Logistic Regression with Polynomial Boundary Implementation in `sklearn`

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

#transform your training and testing data into polynomial features
polynomial_features = PolynomialFeatures(2)
polynomial_features.fit(x)
x_poly = polynomial_features.transform(x)

#fit a logistic regression on top of your polynomial features
logistic_poly = LogisticRegression(solver='lbfgs', max_iter=1000)
logistic_poly.fit(x_poly, y)
```

How would you parametrize an arbitrary complex decision boundary?



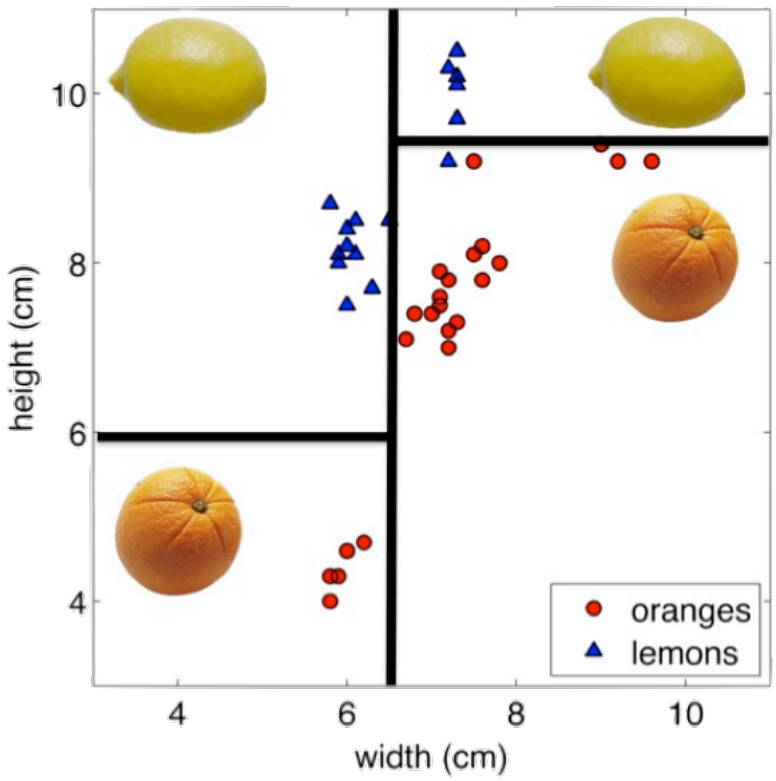
It's not easy to think of a function $g(x)g(x)$ can capture this decision boundary.

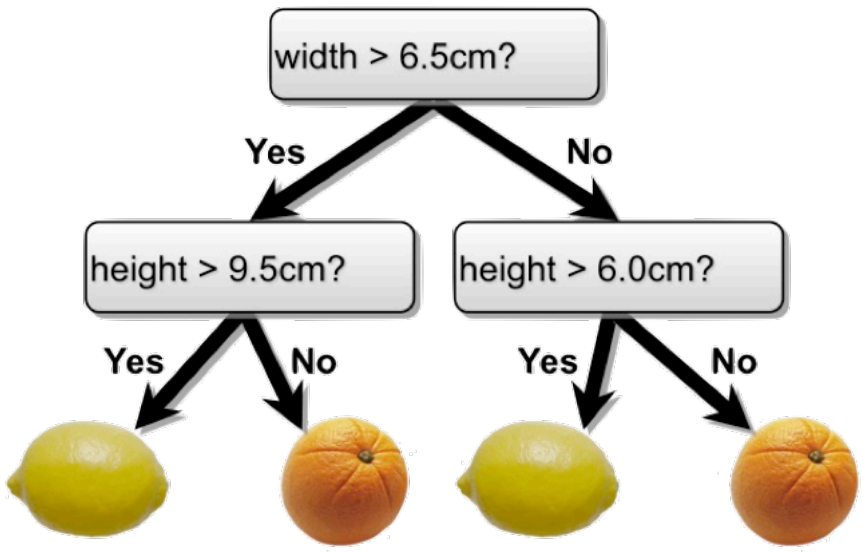
GOAL: Find models that can capture *arbitrarily complex* decision boundaries.

Decision Trees

Decision Tree Models

People in every walk of life have long been using ***decision trees*** (flow charts) for differentiating between classes of objects and phenomena:





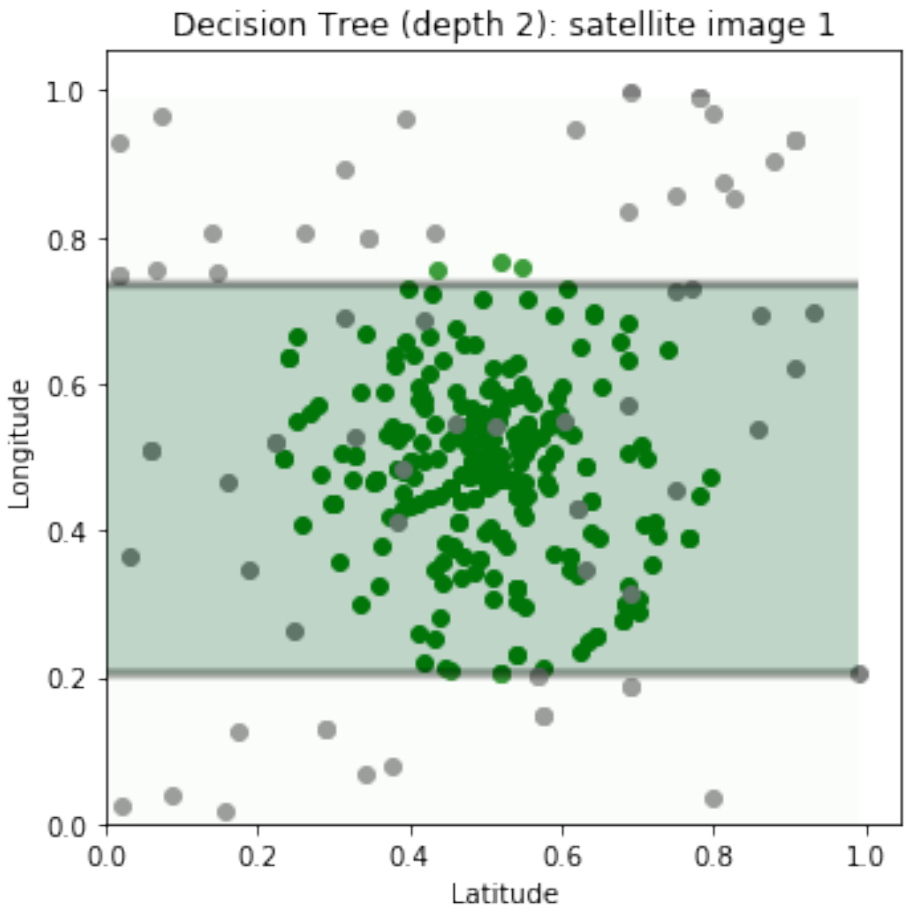
Every flow chart tree corresponds to a partition of the input space by axis aligned lines or (hyper) planes.

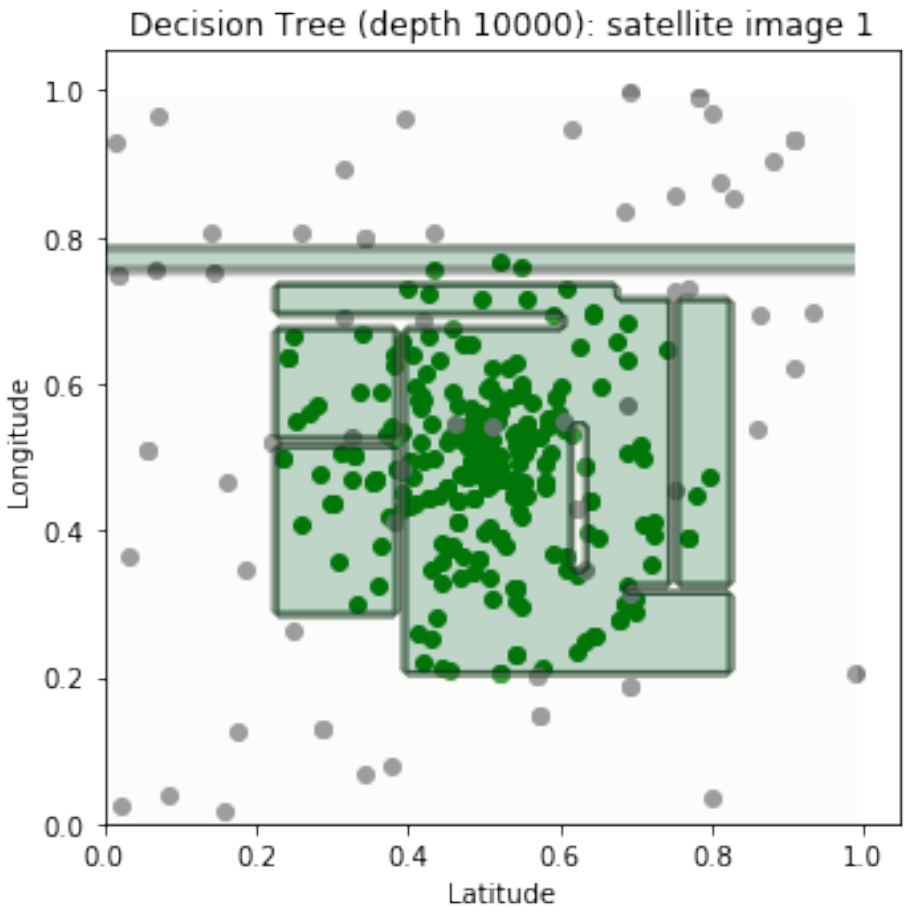
Conversely, every such partition can be written as a flow chart tree.

Shallow vs Deep Trees: The Bias Variance Trade-Off

Model 1: `train_accuracy = 0.653 , test_accuracy = 0.60`

Model 2: `train_accuracy = 0.993 , test_accuracy = 0.7219`





Regularization: Increase Bias, Decrease Variance

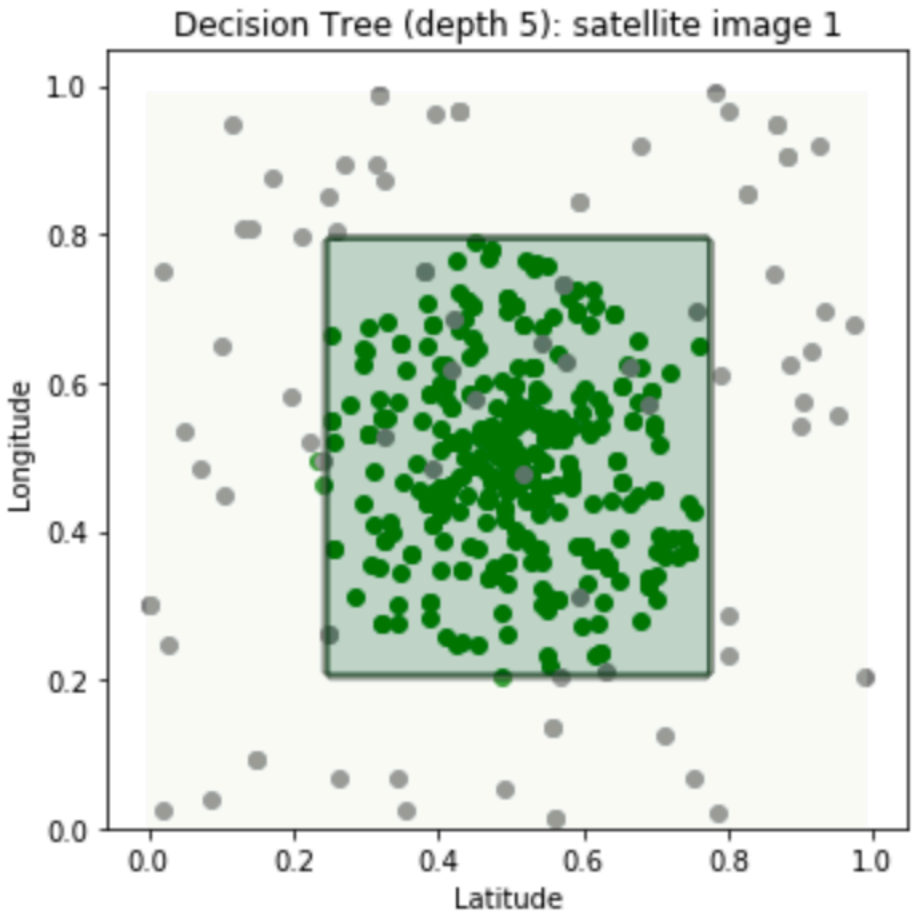
Regularization for Parametric Models

Penalize the parameters of the model for being too large.

1. Regression
   - Linear regression with ℓ_1 ℓ_1 regularization is called the `Lasso` regression in `sklearn`.
   - Linear regression with ℓ_2 ℓ_2 regularization is called the `Ridge` regression in `sklearn`.
2. Classification
   - Set the `C` parameter in `linear_model.LogisticRegression(C=1.)`

Regularization for Trees

Limit the depth of the tree.



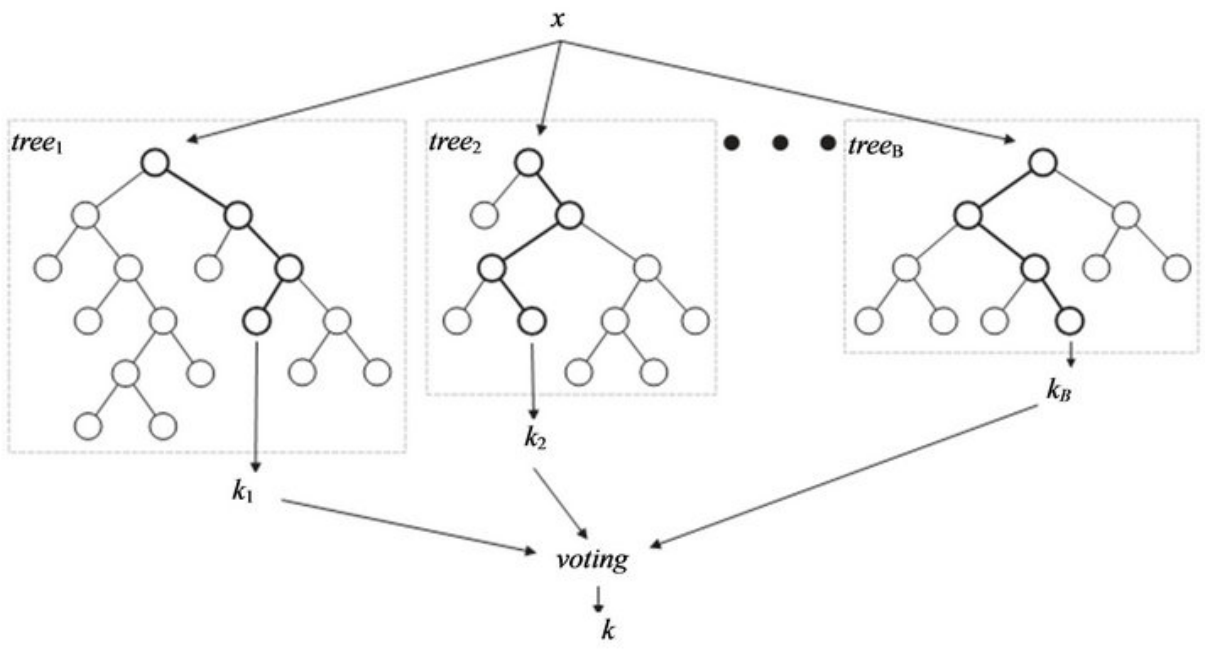
Ensembling: How to Have Your Cake and Eat It Too

Bagging: Random Forest

As we've seen:

1. a shallow decision tree can't capture complex decision boundaries (**high bias**)
2. a deep decision tree is too sensitive to the noise in the data leading to overfitting (**high variance**)

A compromise for reducing variance is to fit a large number of sensitive models (deep trees) on the training data and then average the results (reducing the variance).



A **random forest** is the 'averaged model' of collection of deep decision trees each learned on a subset of the training data (each branch in each tree is also trained on a randomized set of input dimensions to reduce correlation between trees).

Boosting

Instead of 'averaging' over a large number of complex models (to reduce variance or overfitting), we can aggregate a large number of simple (low variance) models into a complex model (to overcome high bias).

Each model T_h T_h might be a poor fit for the data, but a linear combination of the ensemble

$$T = \sum_h \lambda_h T_h$$

$$T = \sum_h \lambda_h T_h$$

can be expressive.

Gradient Boosting

Gradient boosting is a method for iteratively building a complex regression model T T by adding simple models. Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble.

1. We start by fitting a simple model $T^{(0)}$ $T^{(0)}$ on the training data, $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ $\{(x_1, y_1), \ldots, (x_N, y_N)\}$.
2. Compute the ***residuals*** or errors $\{r_1, \ldots, r_N\}$ $\{r_1, \ldots, r_N\}$ for T T
3. Fit a simple model $T^{(i)}$ $T^{(i)}$ to models the errors of T T
4. Set $T \leftarrow T + \lambda T^i$ $T \leftarrow T + \lambda T^i$

Intuitively, with each addition of $T^{(i)}$ $T^{(i)}$, the error is reduced

Note that gradient boosting has a hyper-parameter, $\lambda\lambda$. This is called the ***learning rate***.

Overwhelming Choices

Now that you know so many models for classification, which model should you choose? Suppose that the classes are balanced in the below.

	AdaBoost	RF	logistic	svm	tree
train score	0.764286	0.997143	0.658571	0.662857	0.962857
test score	0.693333	0.653333	0.716667	0.680000	0.620000

Remember Class Imbalance

Your model must address class imbalance!

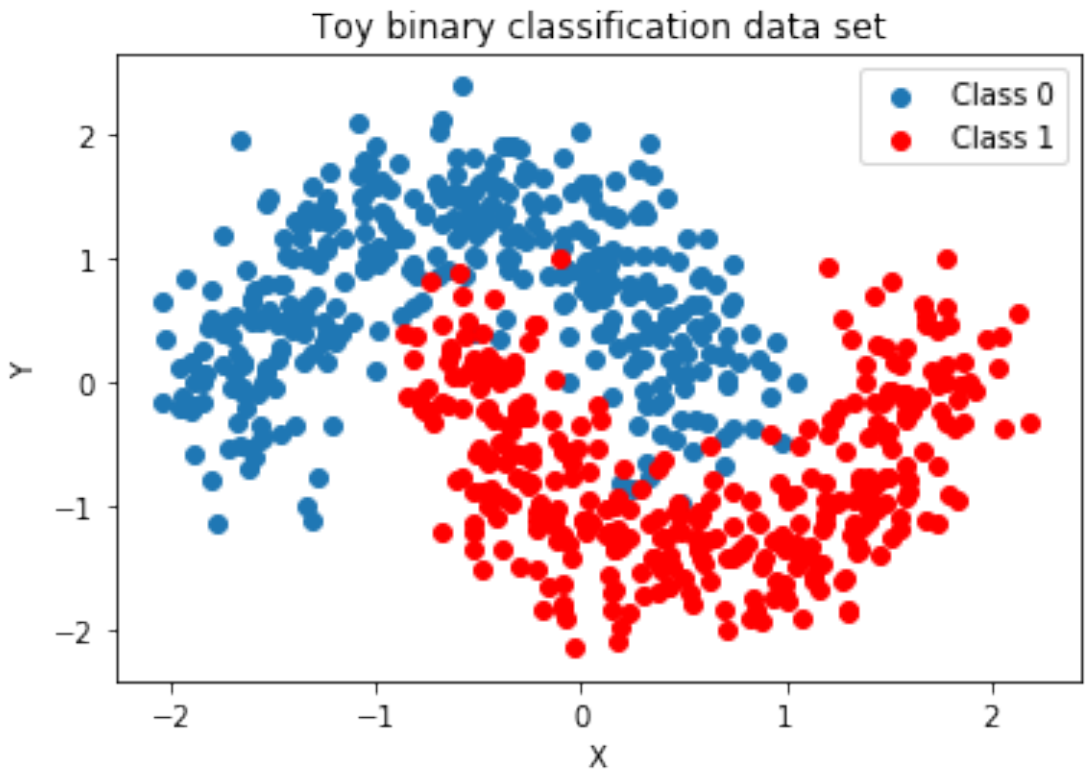
Every `sklearn` classification model has a `class_weights` parameter you can set!

Remember Computation Considerations

1. Your model might need to run on small inefficient computers.
2. Your prediction might need to be made in a brief period of time.
3. You may need to make constant changes to your model.
4. You may need to work with large quantities of data.

Neural Networks

How would you parametrize an arbitrary complex decision boundary?



It's not easy to think of a function $g(x)g(x)$ can capture this decision boundary.

GOAL: Find models that can capture *arbitrarily complex* decision boundaries.

Approximating Arbitrarily Complex Decision Boundaries

Given an exact parametrization, we could learn the functional form, gg, of the decision boundary directly.

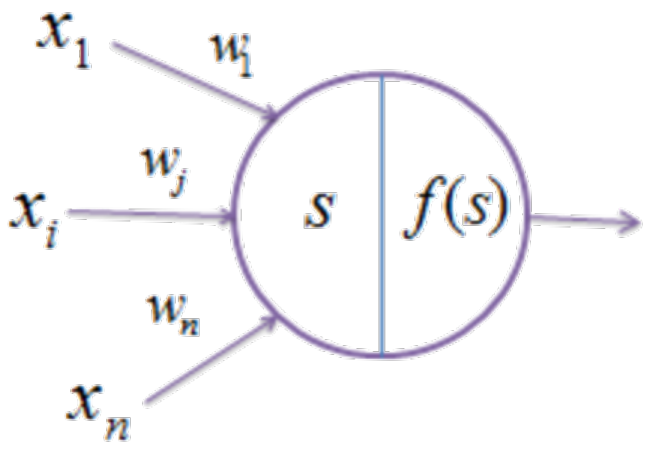
However, assuming an exact form for gg is restrictive.

Rather, we can build increasingly good approximations, $\widehat{g}\hat{g}$, of gg by composing simple functions.

What is a Neuron?

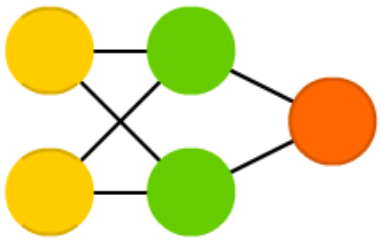
Goal: build a good approximation $\widehat{g}\hat{g}$ of a complex function gg by composing simple functions.

For example, let the following picture represents $f\left(\sum_i w_i x_i\right)f\left(\sum_i w_i x_i\right)$, where ff is a non-linear transform:



What is a Neural Network?

Translate the following graphical representation of a neural network into a functional form, $g(x_1, x_2) =?$
$g(x_1, x_2) = \ ?$

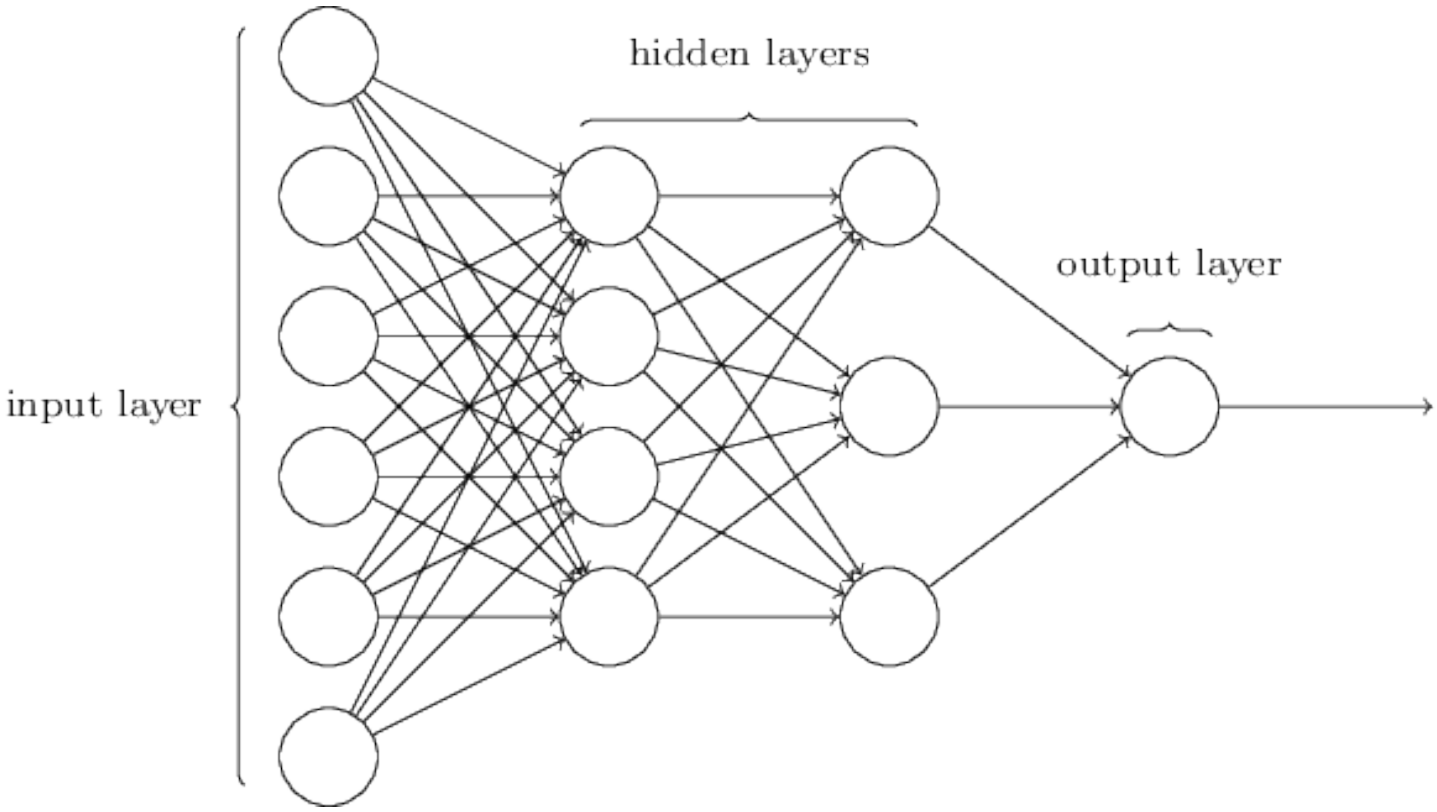


Use the following labels:

1. the input nodes $x_1 x_1$ and $x_2 x_2$
2. the hidden nodes $h_1 h_1$ and $h_2 h_2$
3. the output node yy
4. the weight from $x_i x_i$ to $h_j h_j$ as $w_j^i w_j^i$
5. the weight from $h_j h_j$ to yy as $w^j w^j$
6. the activation function $f(x) = e^{-0.5x^2} f(x) = e^{-0.5x^2}$

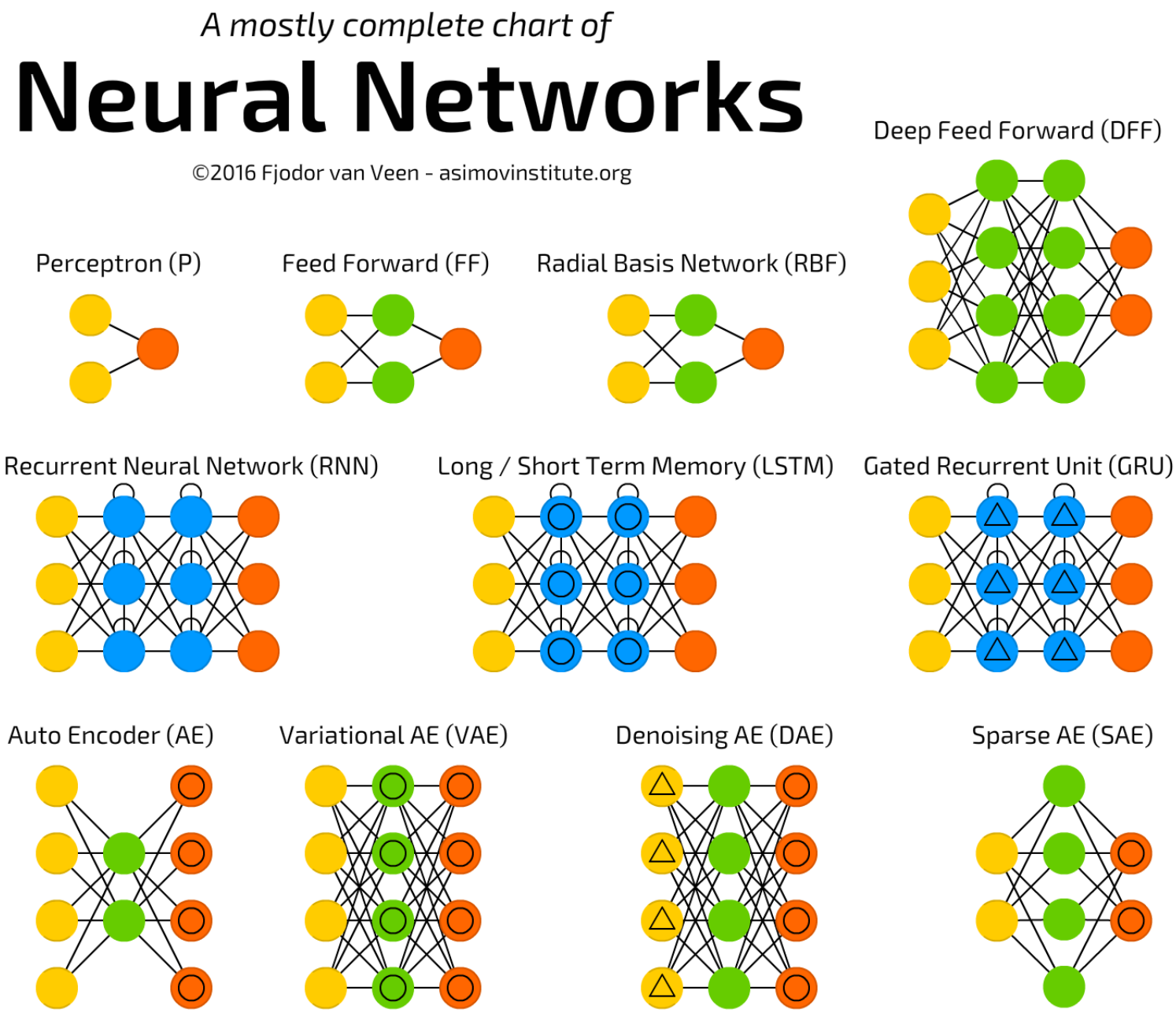
Neural Networks as Function Approximators

Then we can define the approximation $\widehat{g}\hat{g}$ with a graphical schema representing a complex series of compositions and sums of the form, $f\left(\sum_i w_i x_i\right)f\left(\sum_i w_i x_i\right)$



This is a ***neural network***. We denote the weights of the neural network collectively by $\mathbf{W}\mathbf{W}$.

A Flexible Framework for Function Approximation



Neural Networks for Prediction

1. **Regression:** use features `x_train` to predict real-valued labels `y_train` .

   Neural Network Model: `y_predict` $= g_{\mathbf{W}}($ = $g_{\mathbf{W}}($ `x_train` $))$, where $g_{\mathbf{W}}$ $g_{\mathbf{W}}$ is a neural network with parameters $\mathbf{W}$ $\mathbf{W}$.

   Training Objective: find $\mathbf{W}$ $\mathbf{W}$ to minimize the Mean Square Error, $\min_{\mathbf{W}} \text{MSE}(\mathbf{W})$ $\min_{\mathbf{W}} \text{MSE}(\mathbf{W})$.

   Optimizing the Training Objective: How do we take the gradient of a neural network? Can we solve for the zeros of this gradient?

2. **Classification:** use features `x_train` to predict categorical labels `y_train` .

   Neural Network Model: $p(y = 1 | p(y = 1 |$ `x_train` $) = \text{sigmoid}(g_{\mathbf{W}}() = \text{sigmoid}(g_{\mathbf{W}}($ `x_train` $))))$, where $g_{\mathbf{W}}$ $g_{\mathbf{W}}$ is a neural network with parameters $\mathbf{W}$ $\mathbf{W}$.

   Training Objective: find $\mathbf{W}$ $\mathbf{W}$ to minimize the **probability** of predicting wrong.

   Optimizing the Training Objective: How do we take the gradient of a neural network? Can we solve for the zeros of this gradient?

Gradient Descent for Training Neural Networks:

The intuition behind various flavours of gradient descent is as follows:

Gradient Descent: the Algorithm

1. start at random place: $W_0 \leftarrow \textbf{random}$ $W_0 \leftarrow \textbf{random}$
2. until (stopping condition satisfied):

   a. compute gradient: gradient = ∇ ∇ loss_function(W_t W_t)

   b. take a step in the negative gradient direction: $W_{t+1} \leftarrow W_t$ $W_{t+1} \leftarrow W_t$ - eta * gradient

Here *eta* is called the ***learning rate***.

Neural Networks in `python`

`keras` : a Python Library for Neural Networks

`keras` is a `python` library that provides intuitive api's for build neural networks quickly.

```
#keras model for feedforward neural networks
from keras.models import Sequential

#keras model for layers in feedforward networks
from keras.layers import Dense

#keras model for optimizing training objectives
from keras import optimizers
```

Building a Neural Network for Classification in `keras`

```
#instantiate a feedforward model
model = Sequential()

#add layers sequentially

#input layer: 2 input dimensions
model.add(Dense(3, input_dim=2, activation='relu'))
#hidden layer: 2 nodes
model.add(Dense(2, activation='relu'))

#output layer: 1 output dimension
model.add(Dense(1, activation='sigmoid'))

#configure the model: specify training objective and training algorithm
adam = optimizers.Adam(lr=0.01)
model.compile(optimizer=adam,
              loss='binary_crossentropy')
```

Diagnosing Problems with Your Neural Networks

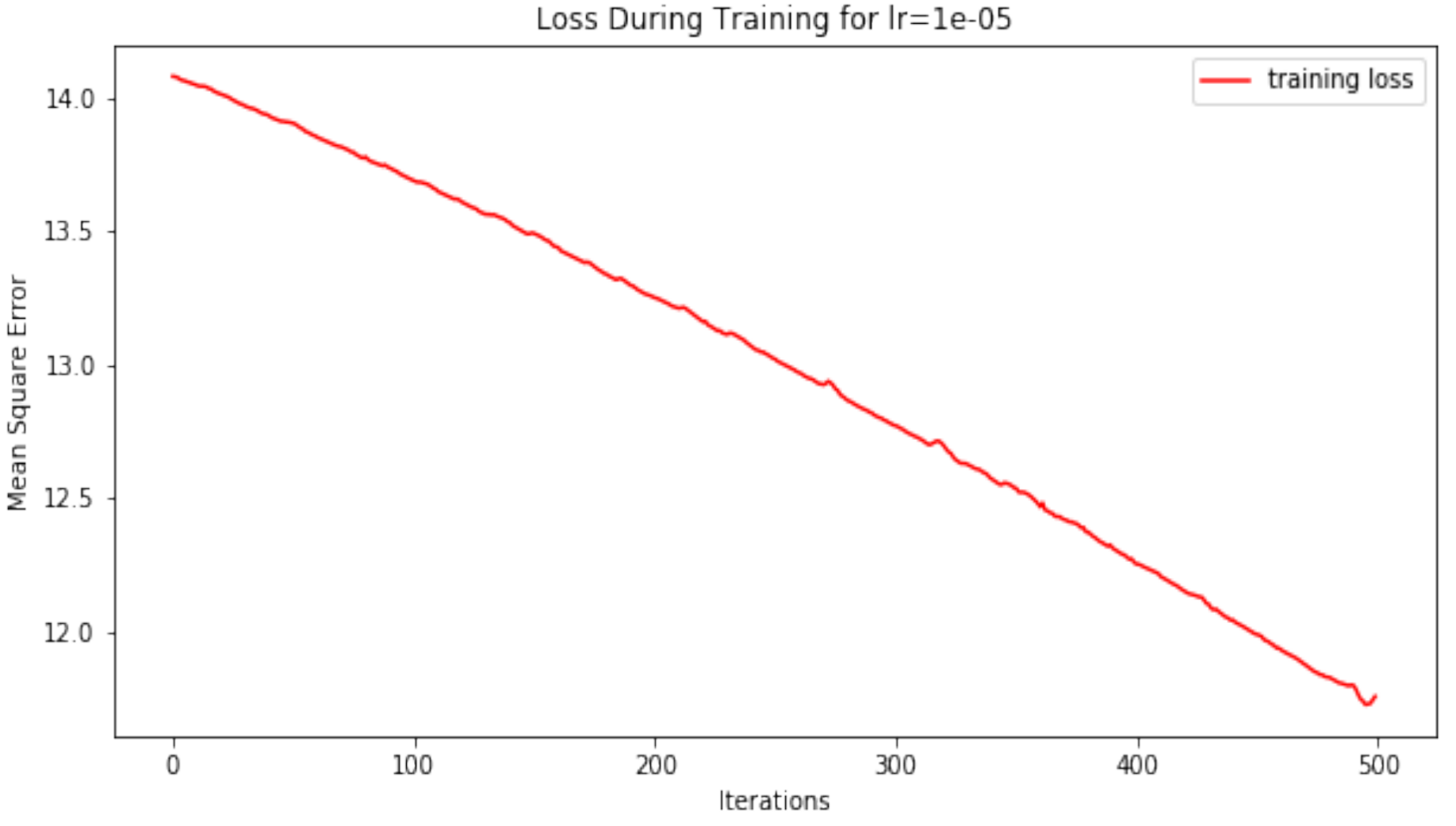
Monitoring Neural Network Training

Visualize the mean square error over the training, this is called the training ***trajectory***.

```
#fit the model and return the mean squared error during training
history = model.fit(x_train, y_train, batch_size=20, shuffle=True, epochs=5
00, verbose=0)

# Plot the loss function and the evaluation metric over the course of train
ing
fig, ax = plt.subplots(1, 1, figsize=(10, 5))

ax.plot(np.array(history.history['loss']), color='blue', label='training lo
ss function')

plt.show()
```

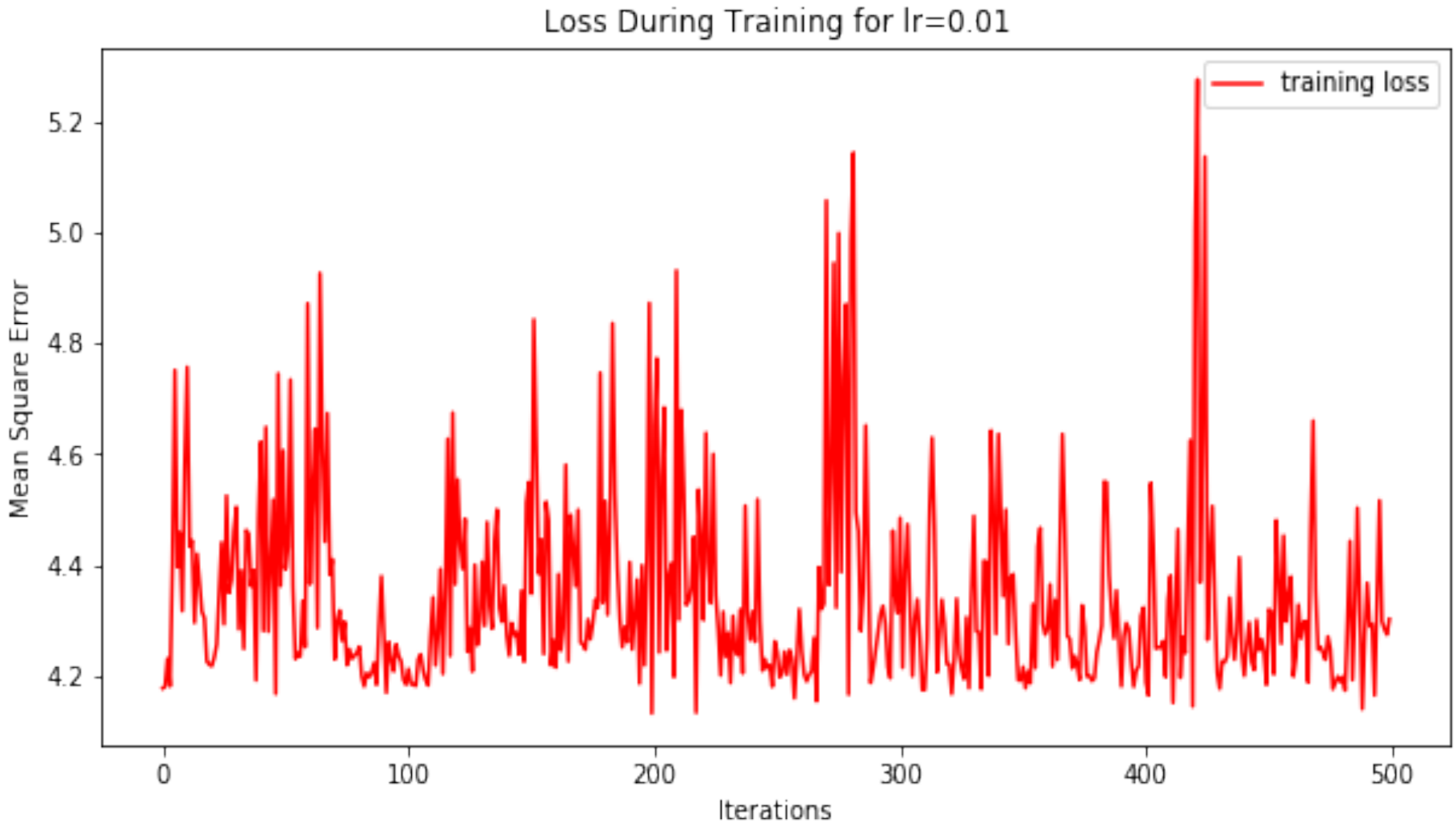
Diagnosing Issues with the Trajectory

If this is your objective function during training, what can you conclude about your step-size?



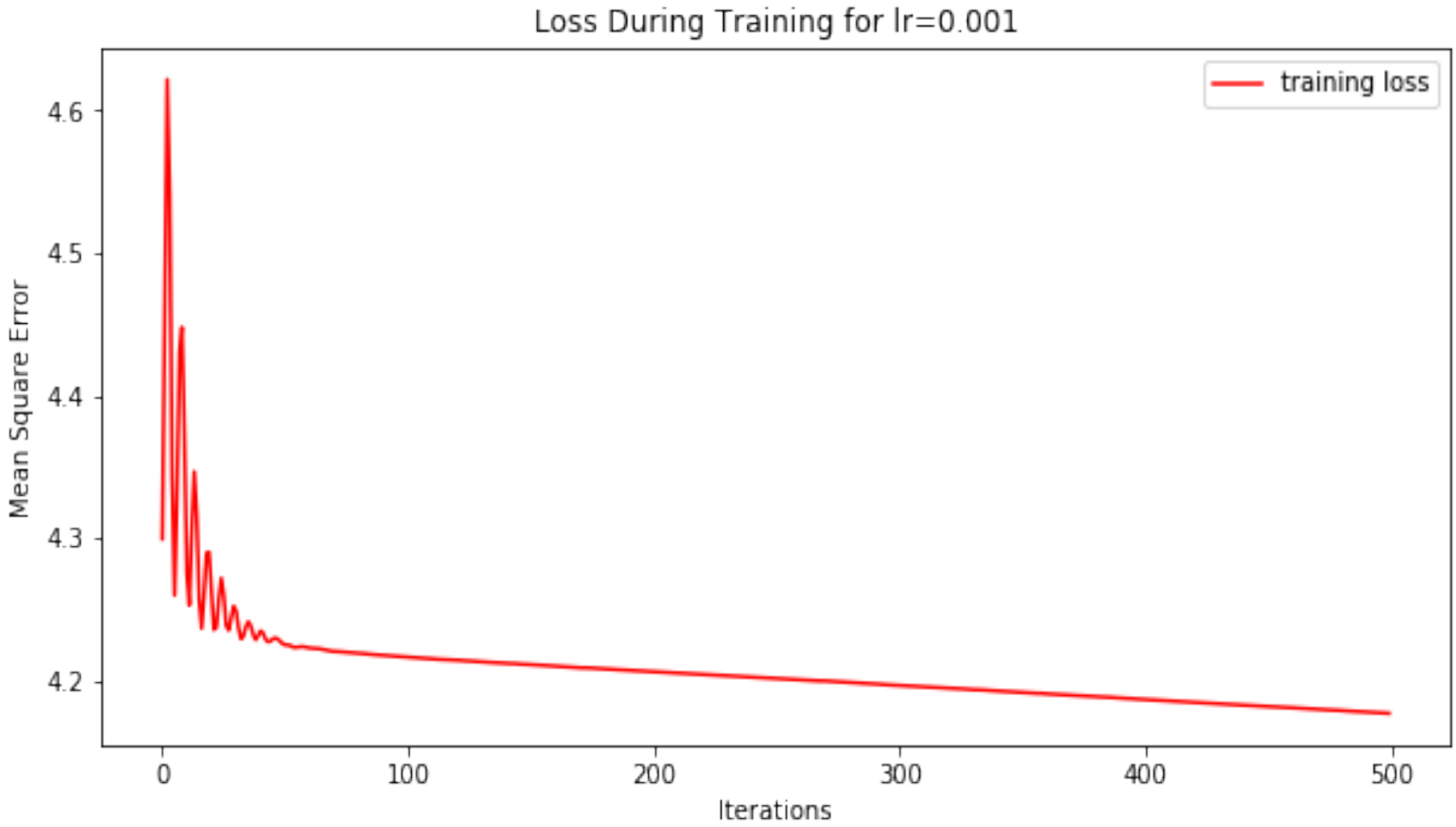
Diagnosing Issues with the Trajectory

If this is your objective function during training, what can you conclude about your step-size?

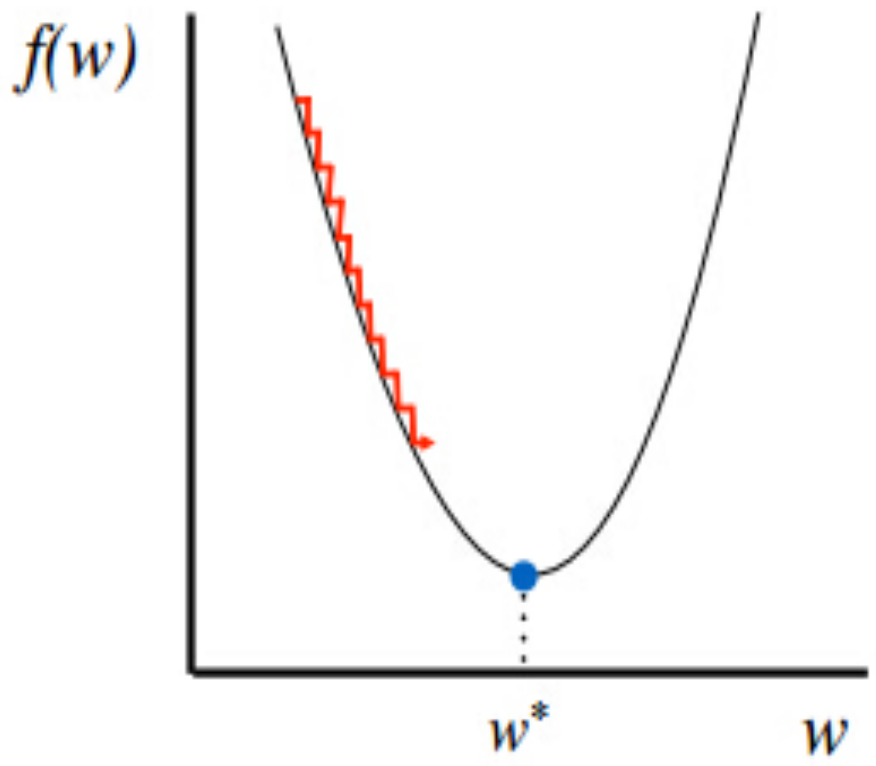


Diagnosing Issues with the Trajectory

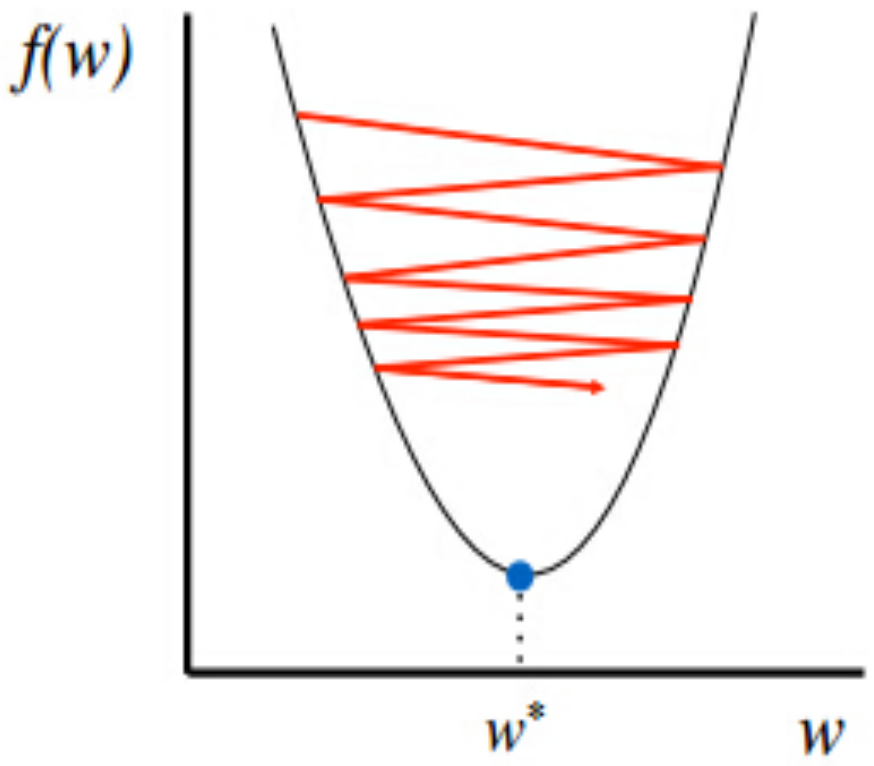
If this is your objective function during training, what can you conclude about your step-size?



Training Your NN: Optimization Choices Matter



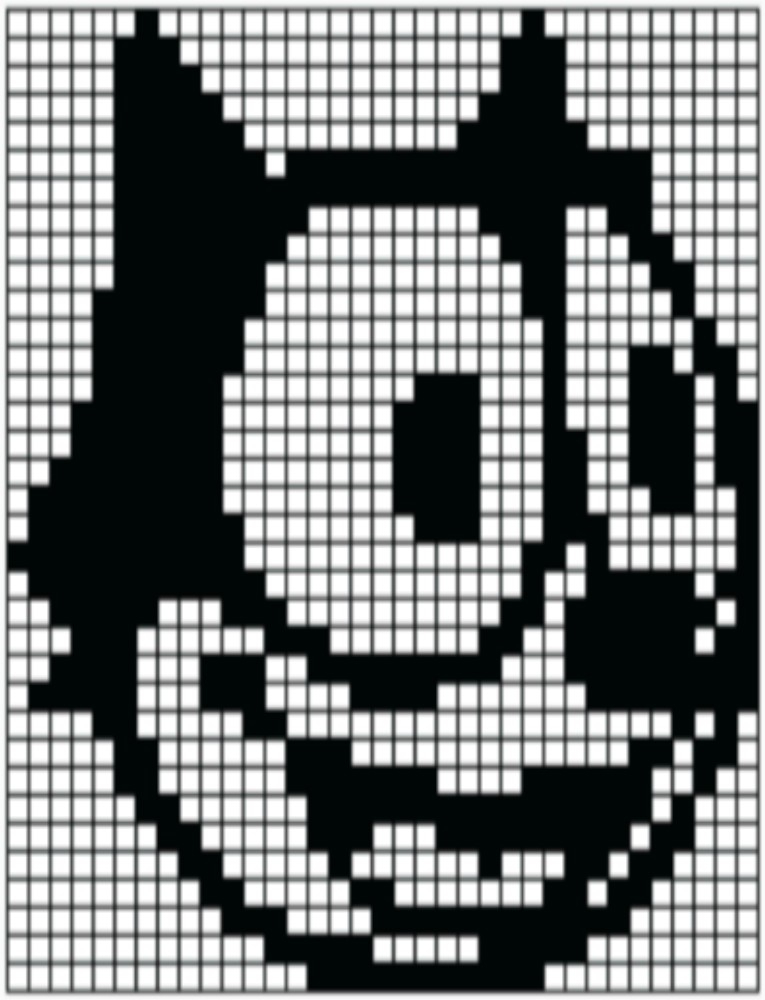
Too small: converge very slowly

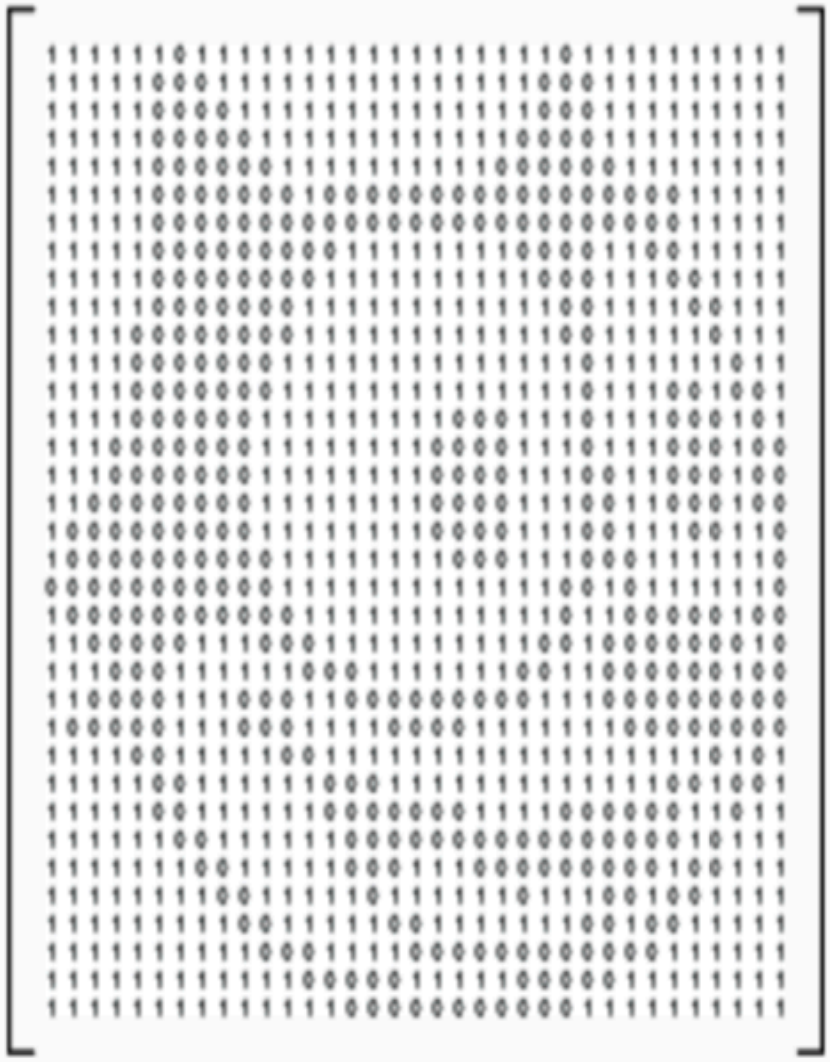


Too big: overshoot and even diverge

Dealing with Image Data

How Do we Represent Gray-scale Images as Numerical Data?





How Do We Represent an Image as a Vector?

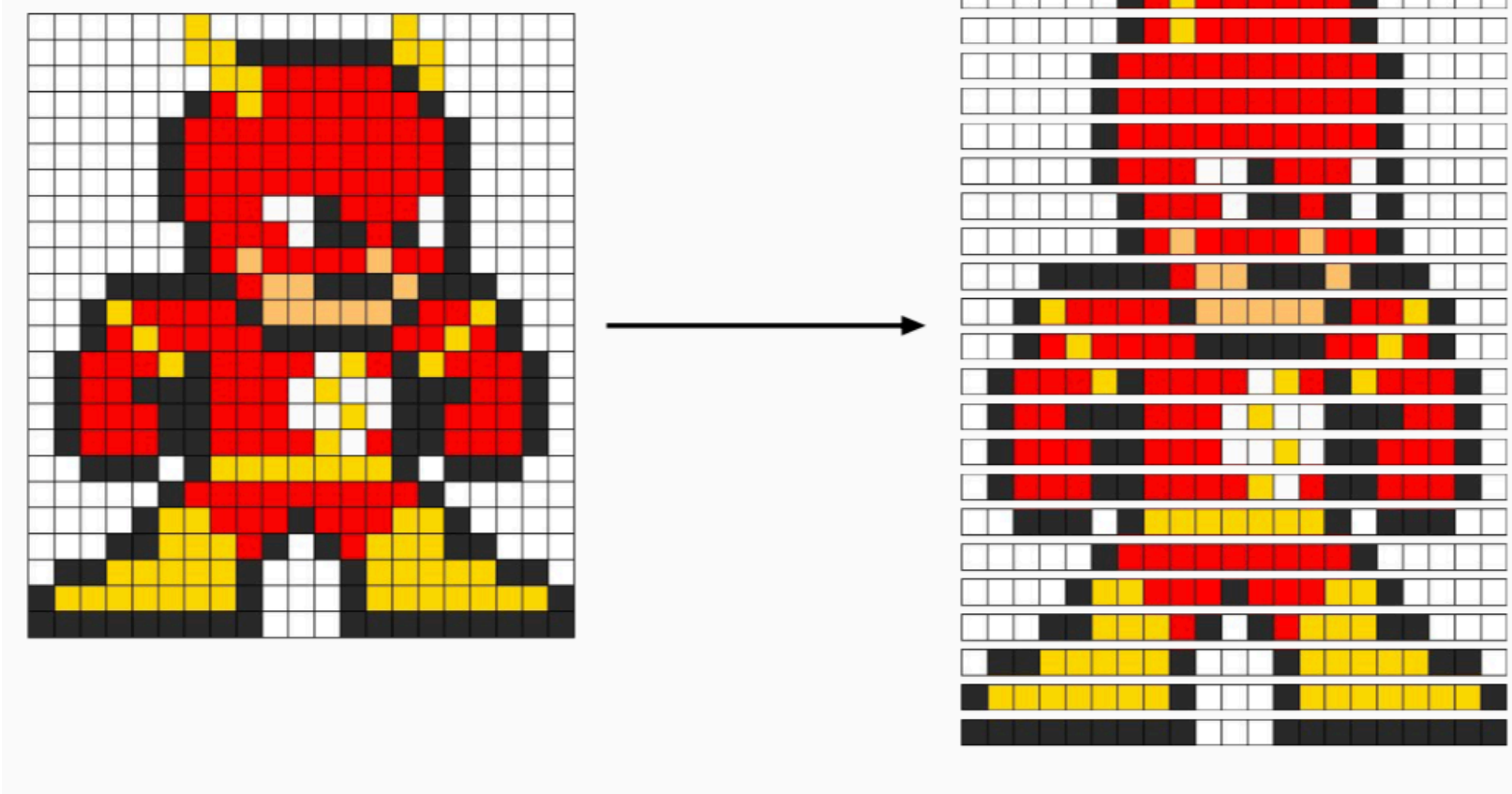
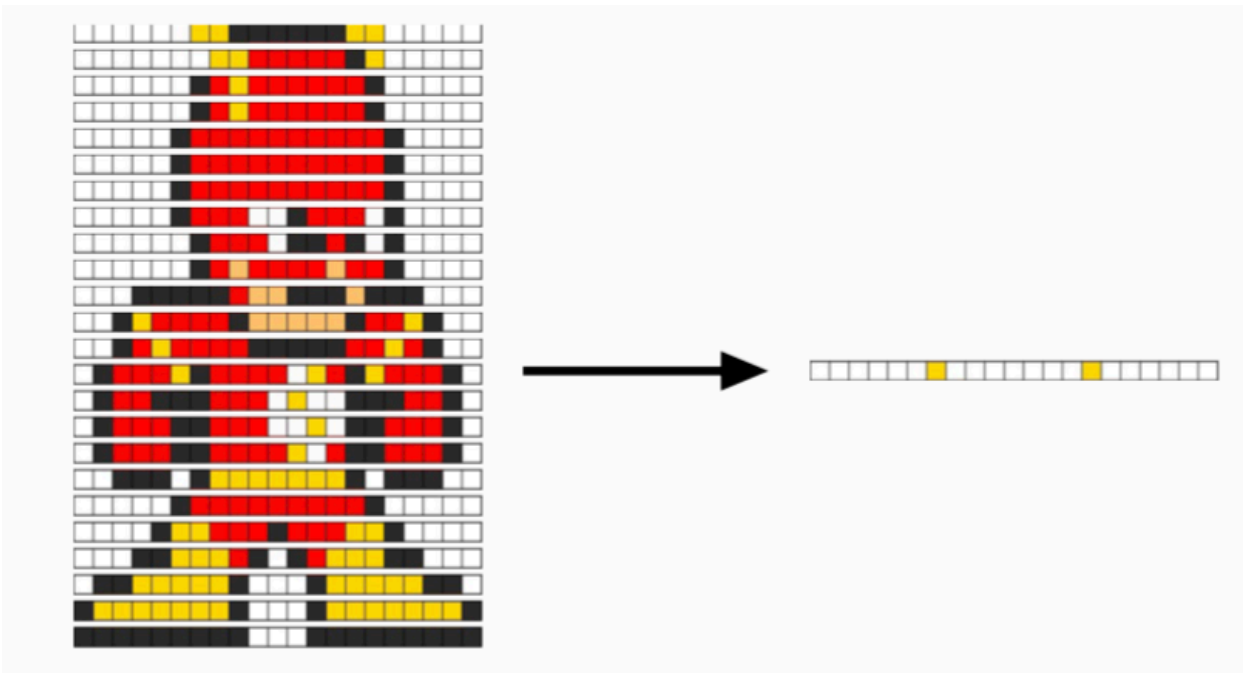


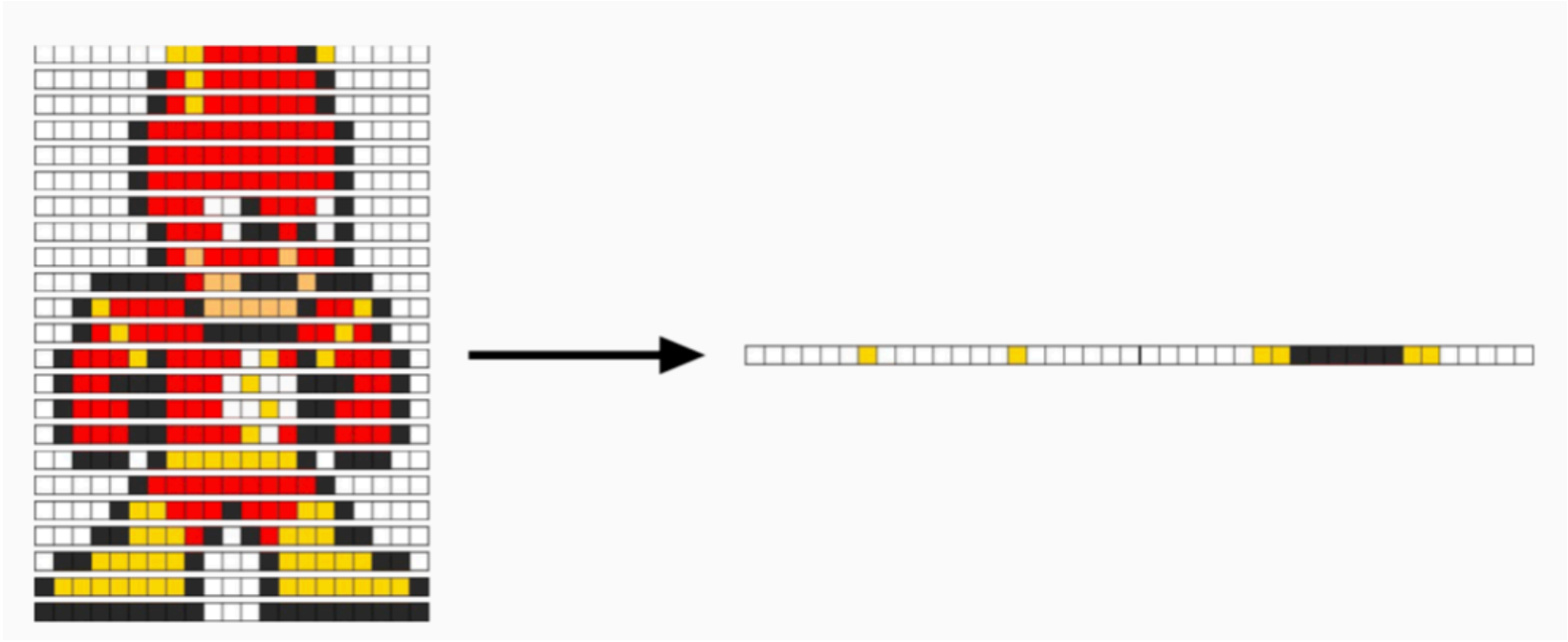
Image as matrix of pixels

Image matrix split along rows

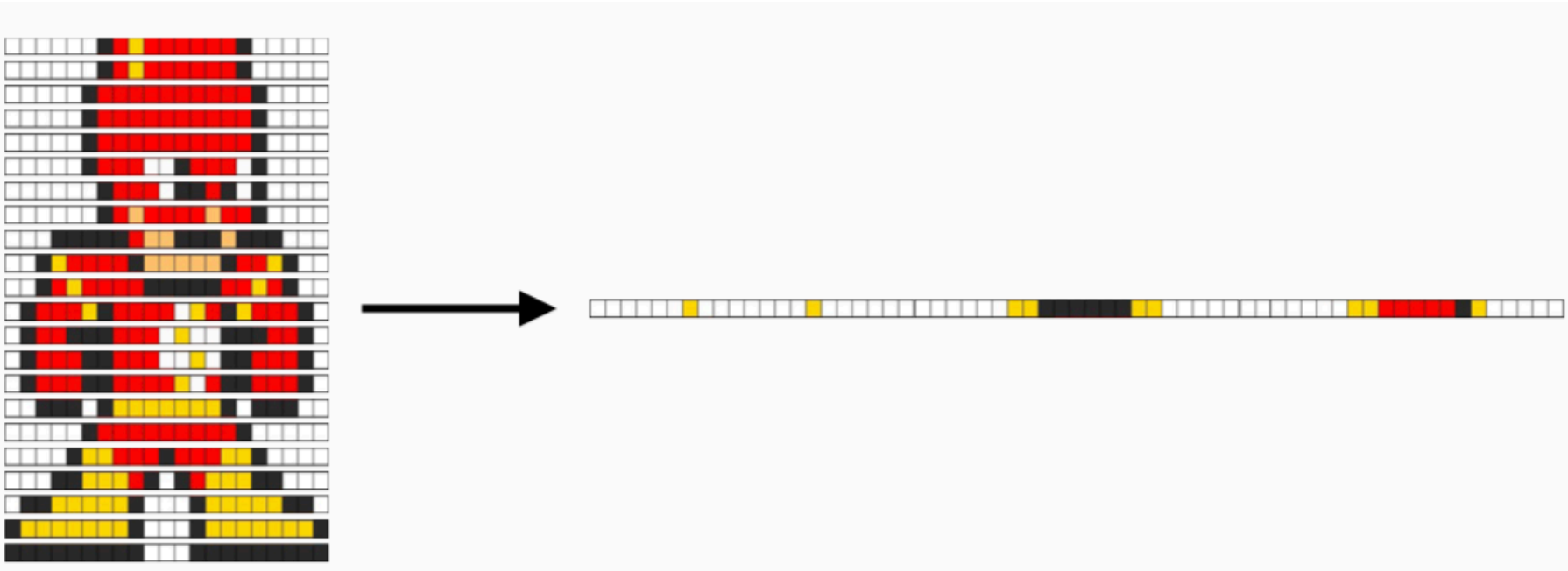
How Do We Represent an Image as a Vector?



How Do We Represent an Image as a Vector?

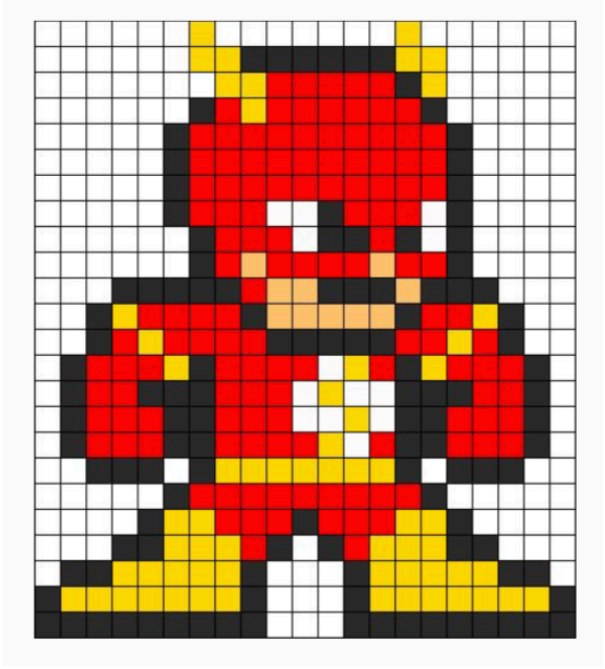


How Do We Represent an Image as a Vector?

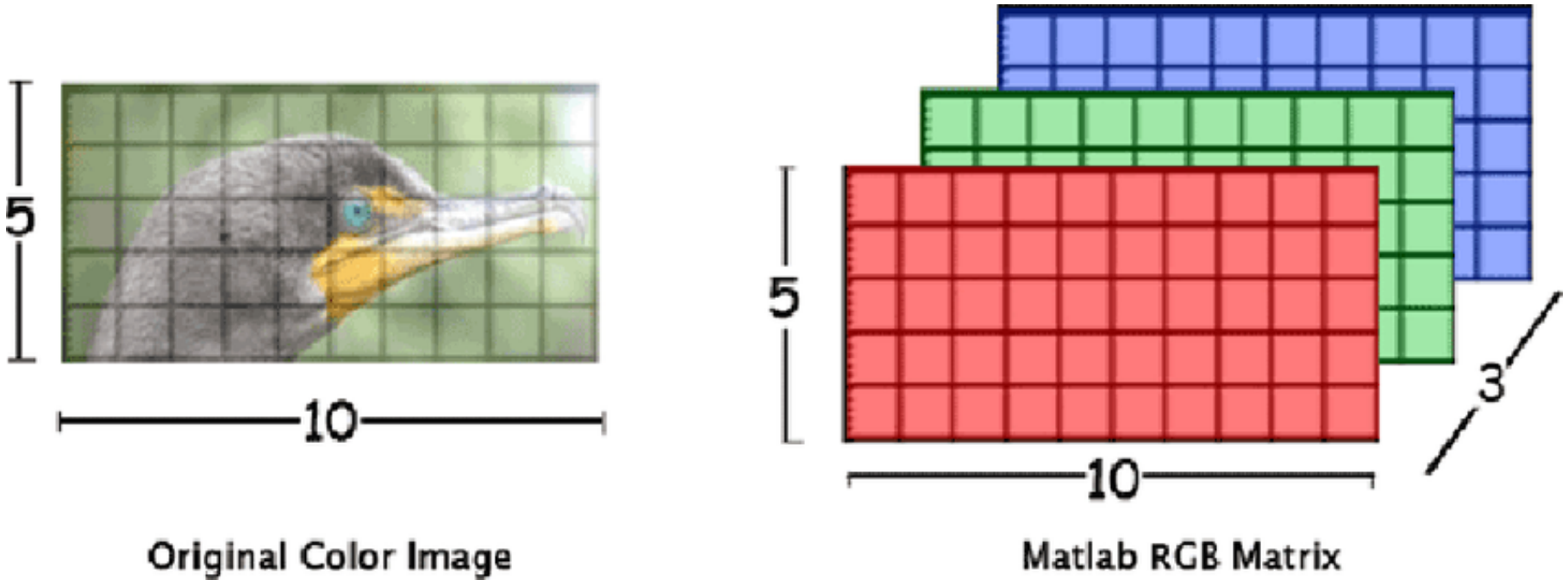


How Do We Represent an Image as a Vector?

This image, when flattened, is represented as a numpy array of shape `(441, )`.



How Do We Represent a Color Image Numerically?



Exercise:

Many Many Models for Classification

Introduction to Data Science

Kigali, Rwanda

July 10th, 2019

Outline

1. Polynomial Logistic Regression
2. Decision Trees
3. Bias and Variance
4. Ensembling:
    - Random Forests
    - AdaBoost
5. Neural Networks
6. Image Data

Logistic Regression with Polynomial Boundaries

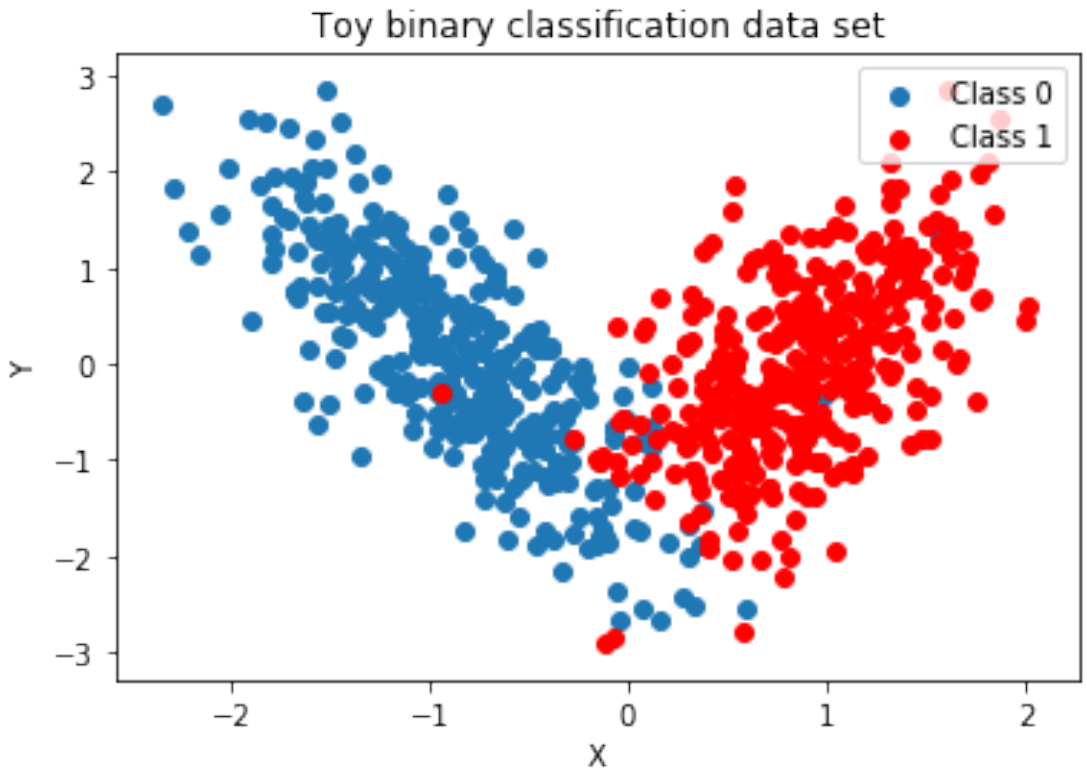
Linear Decision Boundaries

When the decision boundary is linear, it is defined by the equation

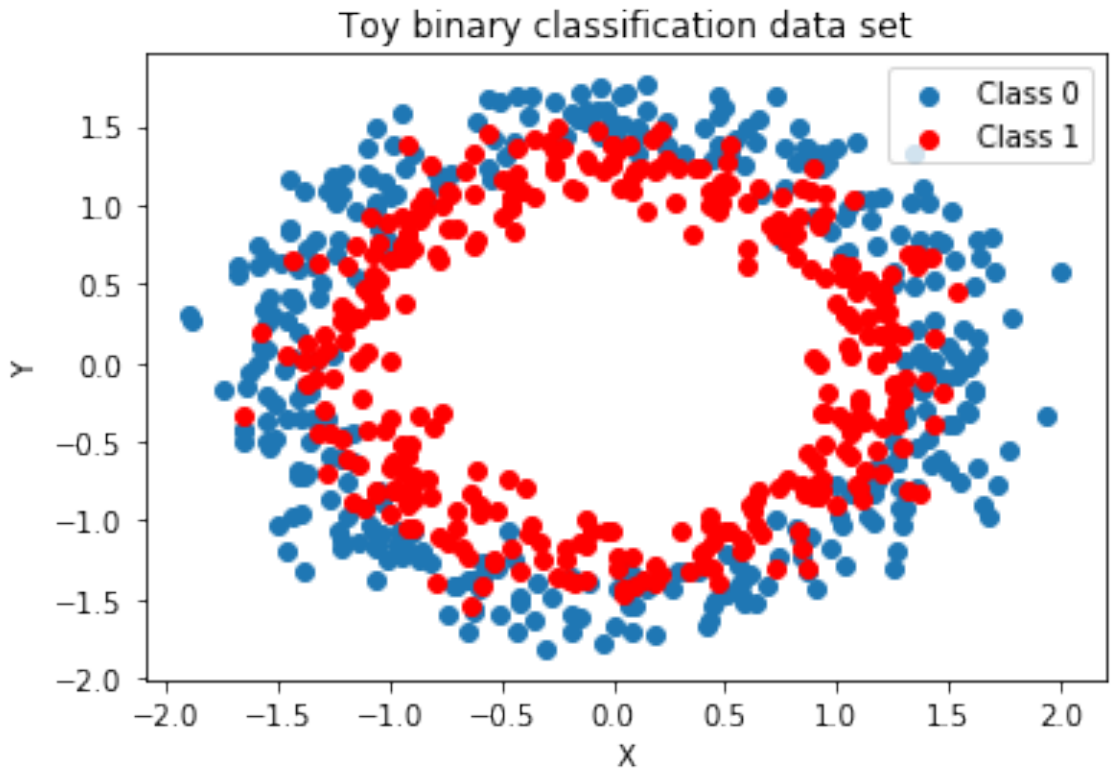
$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

where $x_0 = 1$. Often we write $b = w_0 x_0$ and we call b the ***bias*** term.

The vector $\mathbf{w}$ allow us to gauge the 'distance' of a point from the decision boundary



How would you parametrize a ellipitical decision boundary?



We can say that the decision boundary is given by a ***quadratic function*** of the input:

$$w_1 x_1^2 + w_2 x_2^2 + w_3 = 0$$

We say that we can fit such a decision boundary using logistic regression with degree 2 polynomial features

Logistic Regression with Quadratic Decision Boundary

Recall that polynomial regression is simply a linear regression fit on polynomial features of the inputs `x`:

1. transform x into [1, x, x^2, x^3, x^4, ... etc]
2. fit linear regression on the polynomial features [1, x, x^2, x^3, x^4, ... etc]

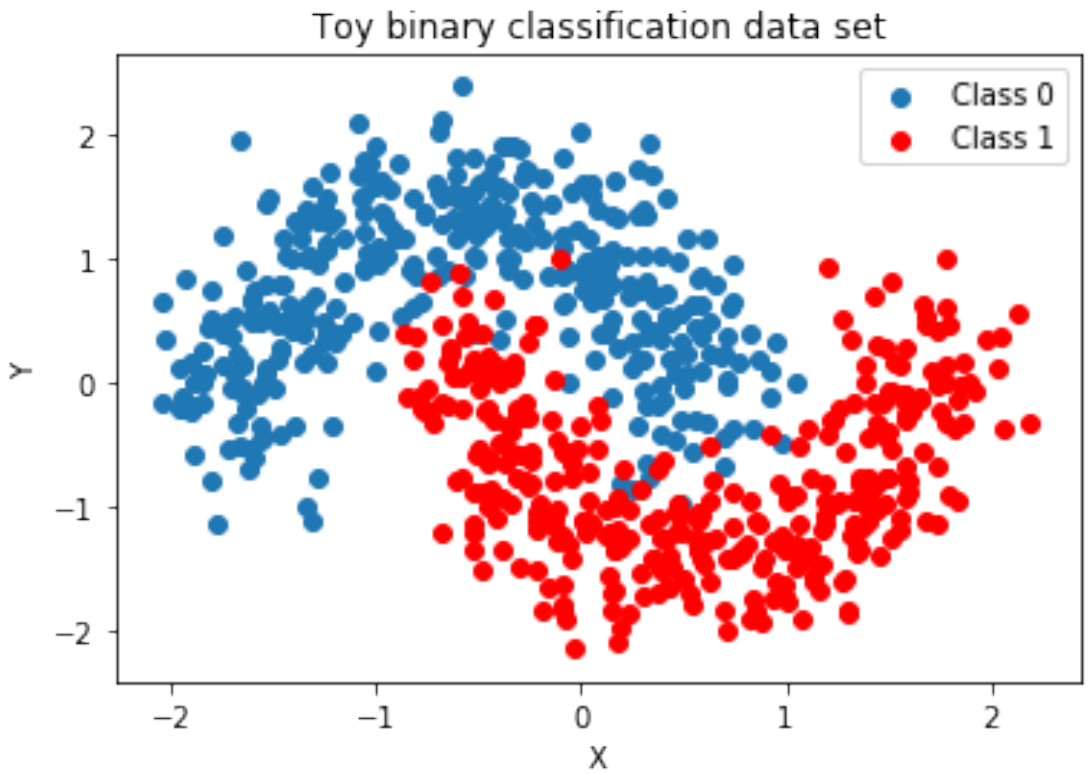
Similary, if we want to fit a logistic regression with quadratic features, we:

1. transform x into [1, x, x^2]
2. fit logistic regression on the quadratic features [1, x, x^2]

Logistic Regression with Polynomial Boundary Implementation in `sklearn`

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

#transform your training and testing data into polynomial features
polynomial_features = PolynomialFeatures(2)
polynomial_features.fit(x)
x_poly = polynomial_features.transform(x)

#fit a logistic regression on top of your polynomial features
logistic_poly = LogisticRegression(solver='lbfgs', max_iter=1000)
logistic_poly.fit(x_poly, y)
```

How would you parametrize an arbitrary complex decision boundary?



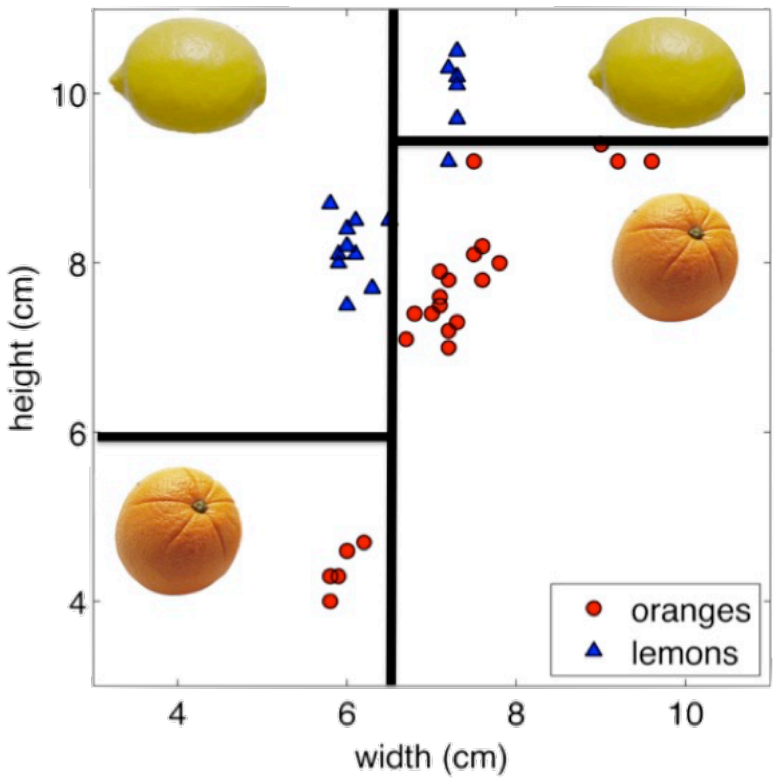
It's not easy to think of a function $g(x)$ can capture this decision boundary.

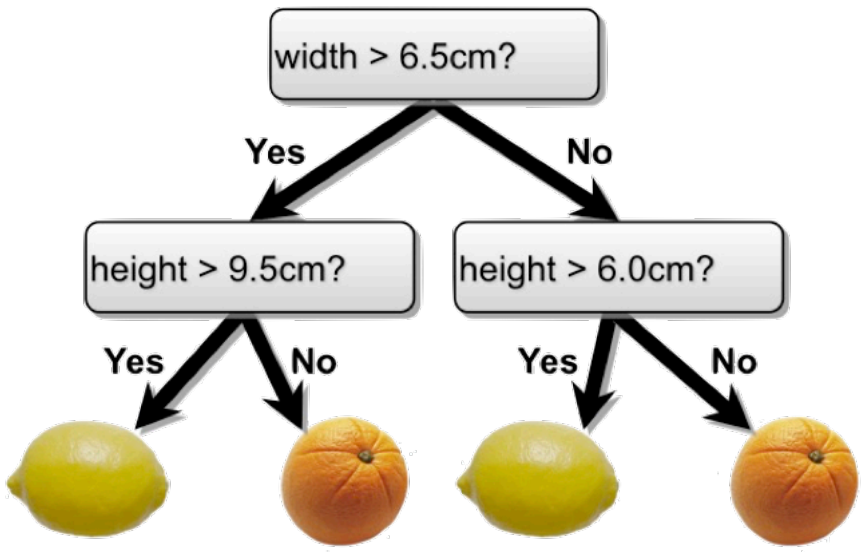
GOAL: Find models that can capture *arbitrarily complex* decision boundaries.

Decision Trees

Decision Tree Models

People in every walk of life have long been using ***decision trees*** (flow charts) for differentiating between classes of objects and phenomena:





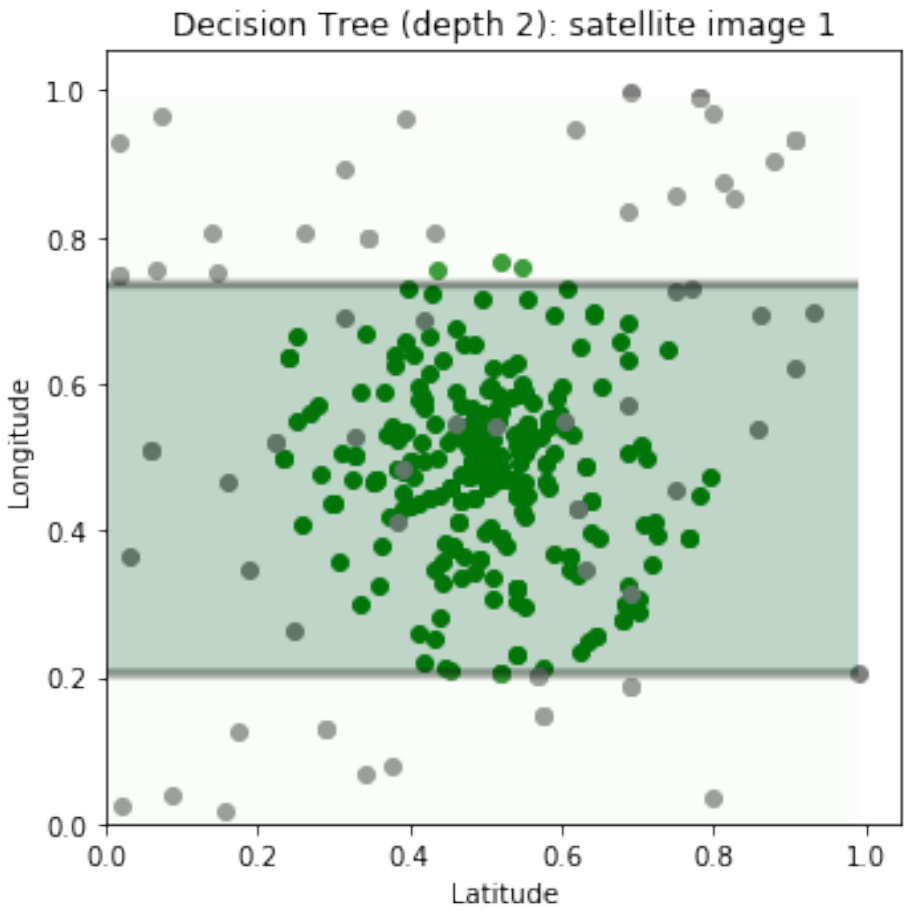
Every flow chart tree corresponds to a partition of the input space by axis aligned lines or (hyper) planes.

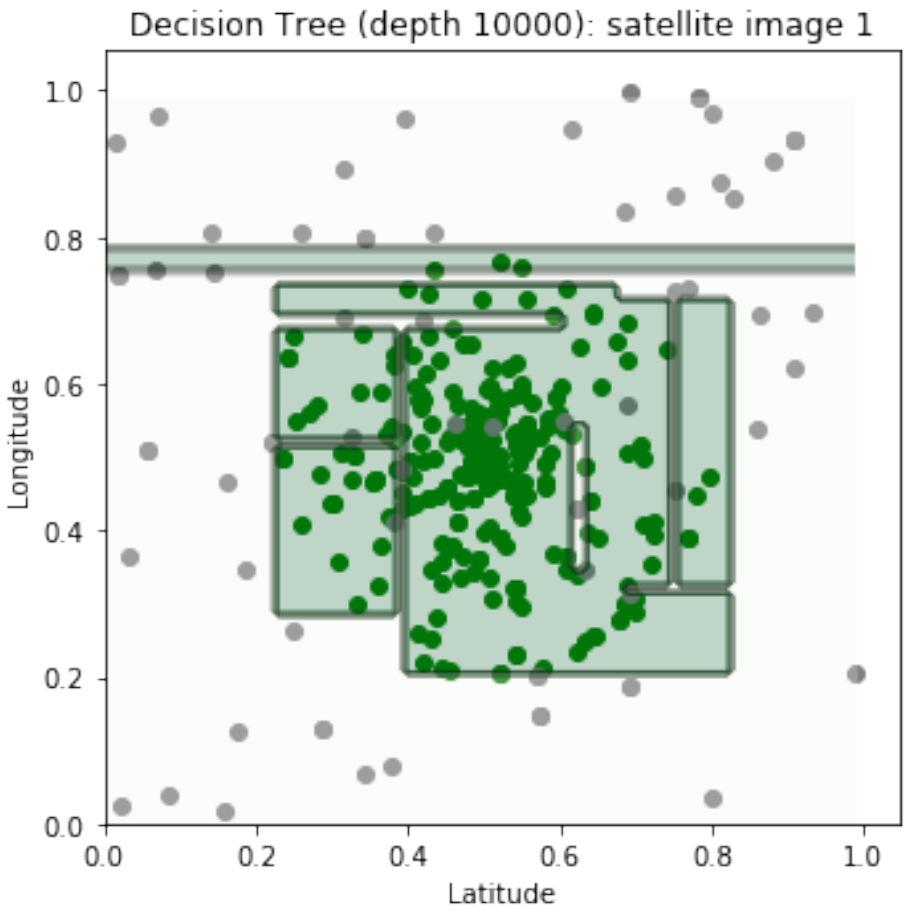
Conversely, every such partition can be written as a flow chart tree.

Shallow vs Deep Trees: The Bias Variance Trade-Off

Model 1: `train_accuracy = 0.653 , test_accuracy = 0.60`

Model 2: `train_accuracy = 0.993 , test_accuracy = 0.7219`





Regularization: Increase Bias, Decrease Variance

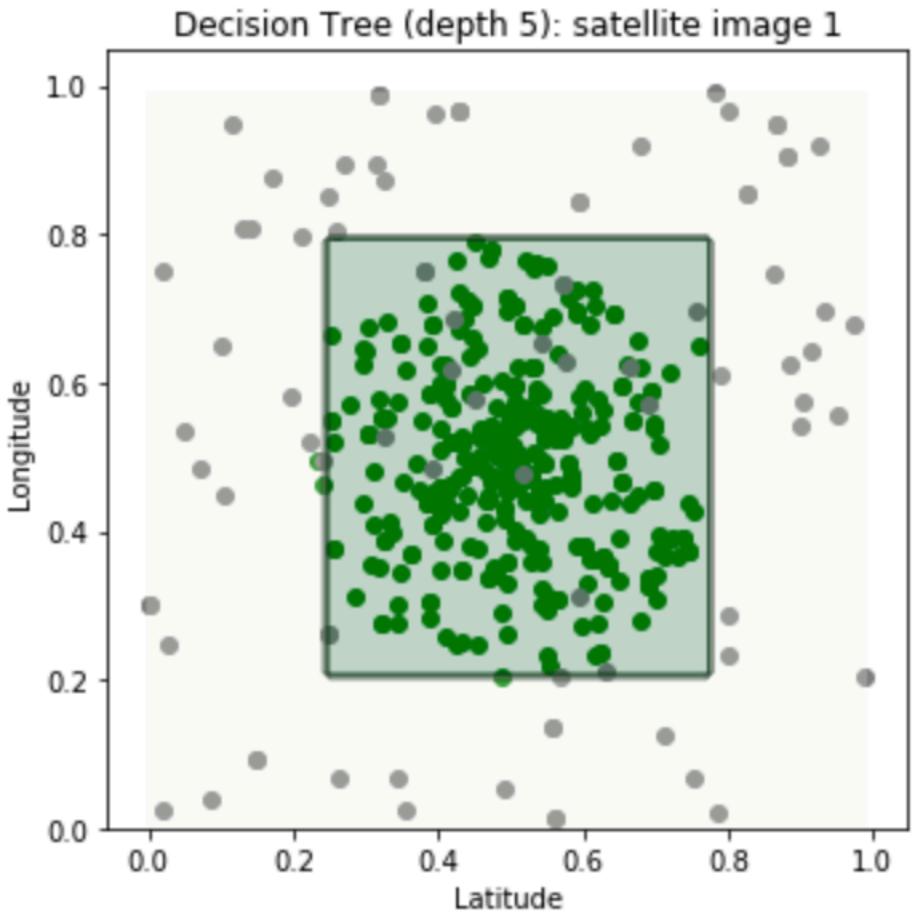
Regularization for Parametric Models

Penalize the parameters of the model for being too large.

1. Regression
   - Linear regression with ℓ_1 regularization is called the `Lasso` regression in `sklearn`.
   - Linear regression with ℓ_2 regularization is called the `Ridge` regression in `sklearn`.
2. Classification
   - Set the `C` parameter in `linear_model.LogisticRegression(C=1.)`

Regularization for Trees

Limit the depth of the tree.



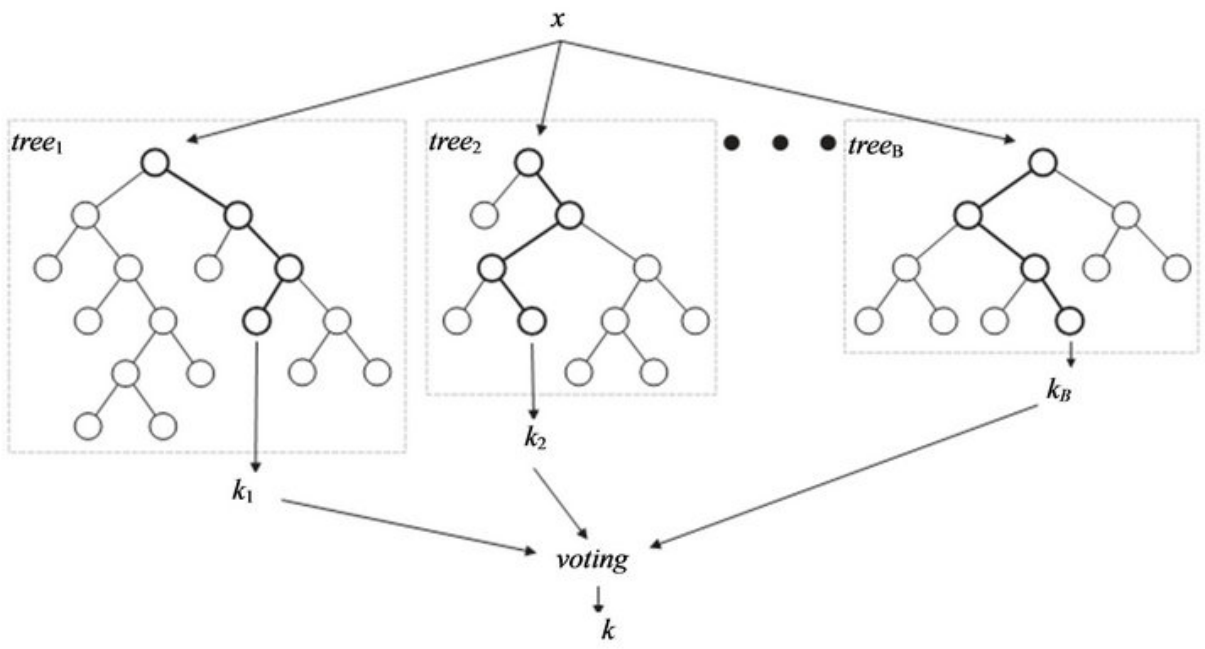
Ensembling: How to Have Your Cake and Eat It Too

Bagging: Random Forest

As we've seen:

1. a shallow decision tree can't capture complex decision boundaries (**high bias**)
2. a deep decision tree is too sensitive to the noise in the data leading to overfitting (**high variance**)

A compromise for reducing variance is to fit a large number of sensitive models (deep trees) on the training data and then average the results (reducing the variance).



A **random forest** is the 'averaged model' of collection of deep decision trees each learned on a subset of the training data (each branch in each tree is also trained on a randomized set of input dimensions to reduce correlation between trees).

Boosting

Instead of 'averaging' over a large number of complex models (to reduce variance or overfitting), we can aggregate a large number of simple (low variance) models into a complex model (to overcome high bias).

Each model T_h might be a poor fit for the data, but a linear combination of the ensemble

$$T = \sum_h \lambda_h T_h$$

can be expressive.

Gradient Boosting

Gradient boosting is a method for iteratively building a complex regression model T by adding simple models. Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble.

1. We start by fitting a simple model $T^{(0)}$ on the training data, $\{(x_1, y_1), \ldots, (x_N, y_N)\}$.
2. Compute the ***residuals*** or errors $\{r_1, \ldots, r_N\}$ for T
3. Fit a simple model $T^{(i)}$ to models the errors of T
4. Set $T \leftarrow T + \lambda T^i$

Intuitively, with each addition of $T^{(i)}$, the error is reduced

Note that gradient boosting has a hyper-parameter, λ. This is called the ***learning rate***.

Overwhelming Choices

Now that you know so many models for classification, which model should you choose? Suppose that the classes are balanced in the below.

	AdaBoost	RF	logistic	svm	tree
train score	0.764286	0.997143	0.658571	0.662857	0.962857
test score	0.693333	0.653333	0.716667	0.680000	0.620000

Remember Class Imbalance

Your model must address class imbalance!

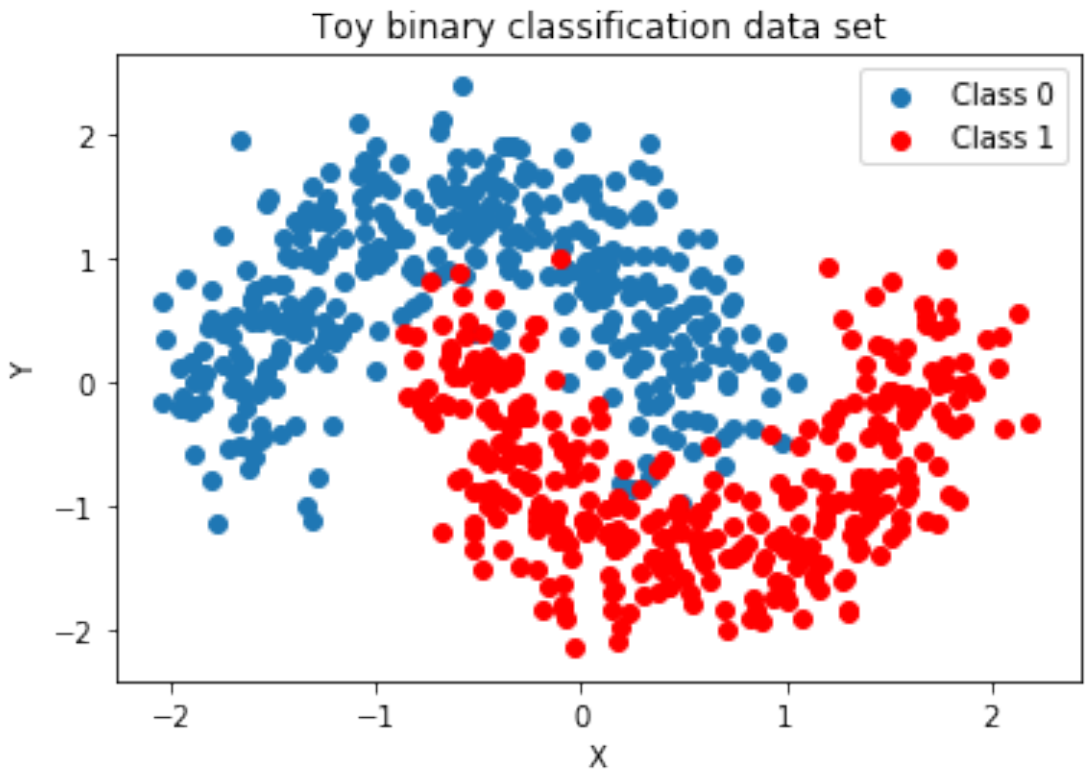
Every `sklearn` classification model has a `class_weights` parameter you can set!

Remember Computation Considerations

1. Your model might need to run on small inefficient computers.
2. Your prediction might need to be made in a brief period of time.
3. You may need to make constant changes to your model.
4. You may need to work with large quantities of data.

Neural Networks

How would you parametrize an arbitrary complex decision boundary?



It's not easy to think of a function $g(x)$ can capture this decision boundary.

GOAL: Find models that can capture *arbitrarily complex* decision boundaries.

Approximating Arbitrarily Complex Decision Boundaries

Given an exact parametrization, we could learn the functional form, g, of the decision boundary directly.

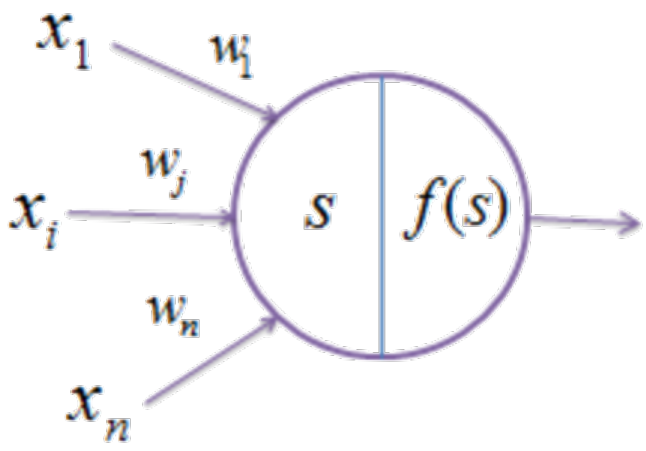
However, assuming an exact form for g is restrictive.

Rather, we can build increasingly good approximations, $\hat{g}$, of g by composing simple functions.

What is a Neuron?

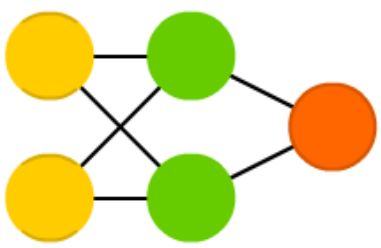
Goal: build a good approximation $\hat{g}$ of a complex function g by composing simple functions.

For example, let the following picture represents $f\left(\sum_i w_i x_i\right)$, where f is a non-linear transform:



What is a Neural Network?

Translate the following graphical representation of a neural network into a functional form, $g(x_1, x_2) = ?$

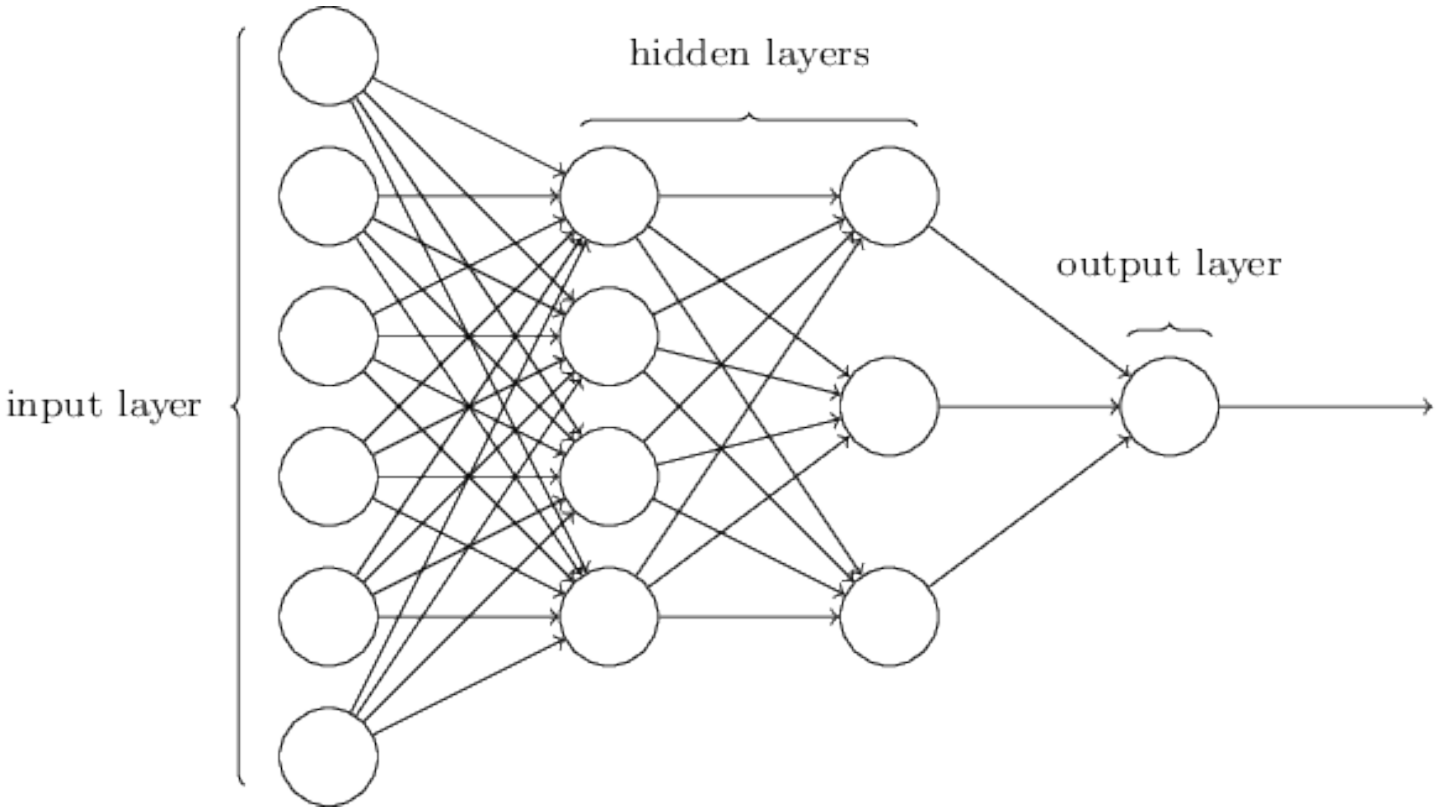


Use the following labels:

1. the input nodes x_1 and x_2
2. the hidden nodes h_1 and h_2
3. the output node y
4. the weight from x_i to h_j as w_j^i
5. the weight from h_j to y as w^j
6. the activation function $f(x) = e^{-0.5x^2}$

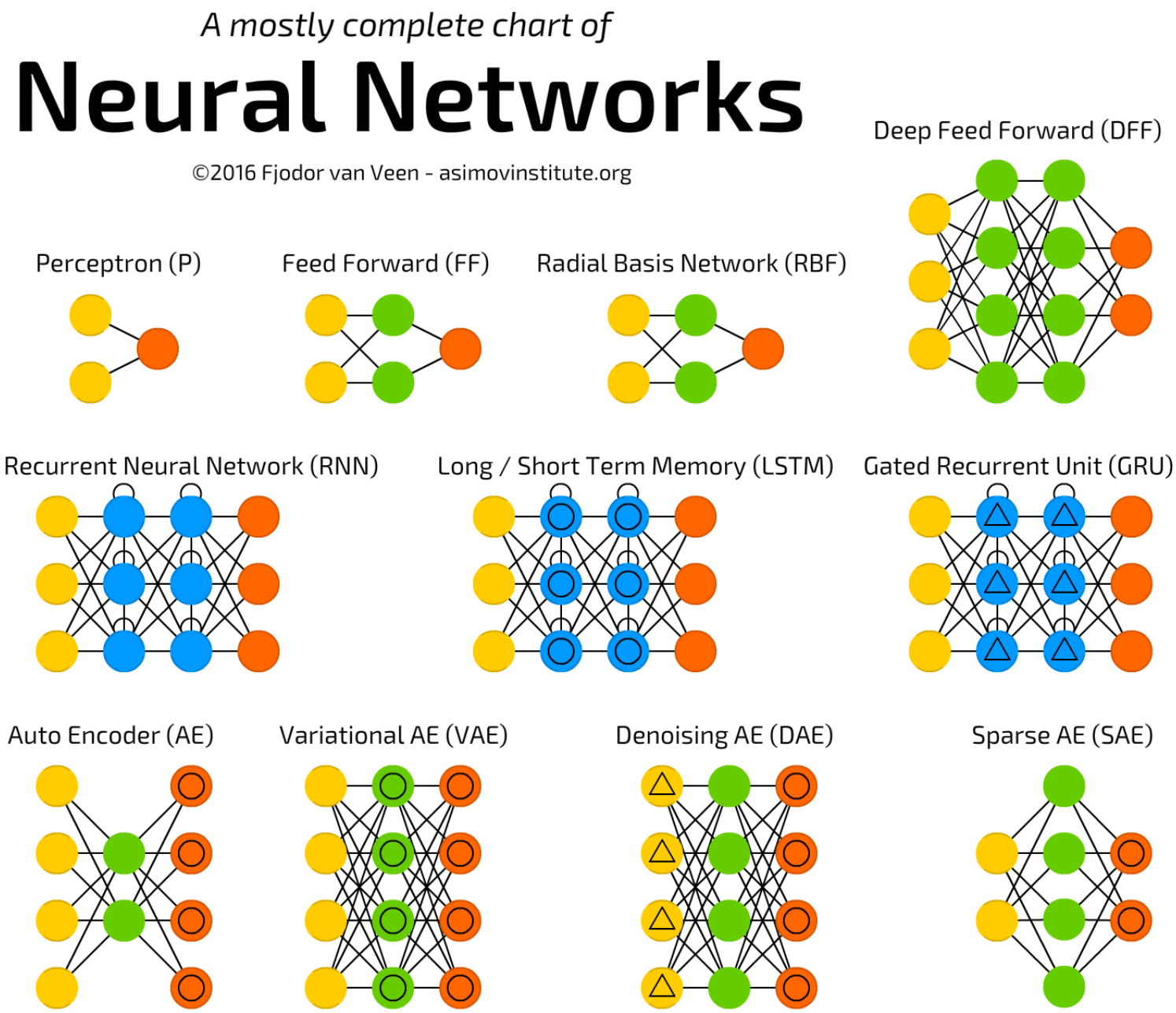
Neural Networks as Function Approximators

Then we can define the approximation $\hat{g}$ with a graphical schema representing a complex series of compositions and sums of the form, $f\left(\sum_i w_i x_i\right)$



This is a ***neural network***. We denote the weights of the neural network collectively by $\mathbf{W}$.

A Flexible Framework for Function Approximation



Neural Networks for Prediction

1. **Regression:** use features `x_train` to predict real-valued labels `y_train`.

   Neural Network Model: `y_predict` $= g_{\mathbf{W}}($ `x_train` $)$, where $g_{\mathbf{W}}$ is a neural network with parameters $\mathbf{W}$.

   Training Objective: find $\mathbf{W}$ to minimize the Mean Square Error, $\min_{\mathbf{W}} \text{MSE}(\mathbf{W})$.

   Optimizing the Training Objective: How do we take the gradient of a neural network? Can we solve for the zeros of this gradient?

2. **Classification:** use features `x_train` to predict categorical labels `y_train`.

   Neural Network Model: $p(y = 1 \mid$ `x_train` $) = \text{sigmoid}(g_{\mathbf{W}}($ `x_train` $))$, where $g_{\mathbf{W}}$ is a neural network with parameters $\mathbf{W}$.

   Training Objective: find $\mathbf{W}$ to minimize the **probability** of predicting wrong.

   Optimizing the Training Objective: How do we take the gradient of a neural network? Can we solve for the zeros of this gradient?

Gradient Descent for Training Neural Networks:

The intuition behind various flavours of gradient descent is as follows:

Gradient Descent: the Algorithm

1. start at random place: $W_0 \leftarrow \mathbf{random}$
2. until (stopping condition satisfied):

   a. compute gradient: gradient = ∇ loss_function(W_t)

   b. take a step in the negative gradient direction: $W_{t+1} \leftarrow W_t$ - eta * gradient

Here *eta* is called the ***learning rate***.

Neural Networks in `python`

`keras` : a Python Library for Neural Networks

`keras` is a `python` library that provides intuitive api's for build neural networks quickly.

```
#keras model for feedforward neural networks
from keras.models import Sequential

#keras model for layers in feedforward networks
from keras.layers import Dense

#keras model for optimizing training objectives
from keras import optimizers
```

Building a Neural Network for Classification in `keras`

```
#instantiate a feedforward model
model = Sequential()

#add layers sequentially

#input layer: 2 input dimensions
model.add(Dense(3, input_dim=2, activation='relu'))
#hidden layer: 2 nodes
model.add(Dense(2, activation='relu'))

#output layer: 1 output dimension
model.add(Dense(1, activation='sigmoid'))

#configure the model: specify training objective and training algorithm
adam = optimizers.Adam(lr=0.01)
model.compile(optimizer=adam,
              loss='binary_crossentropy')
```

Diagnosing Problems with Your Neural Networks

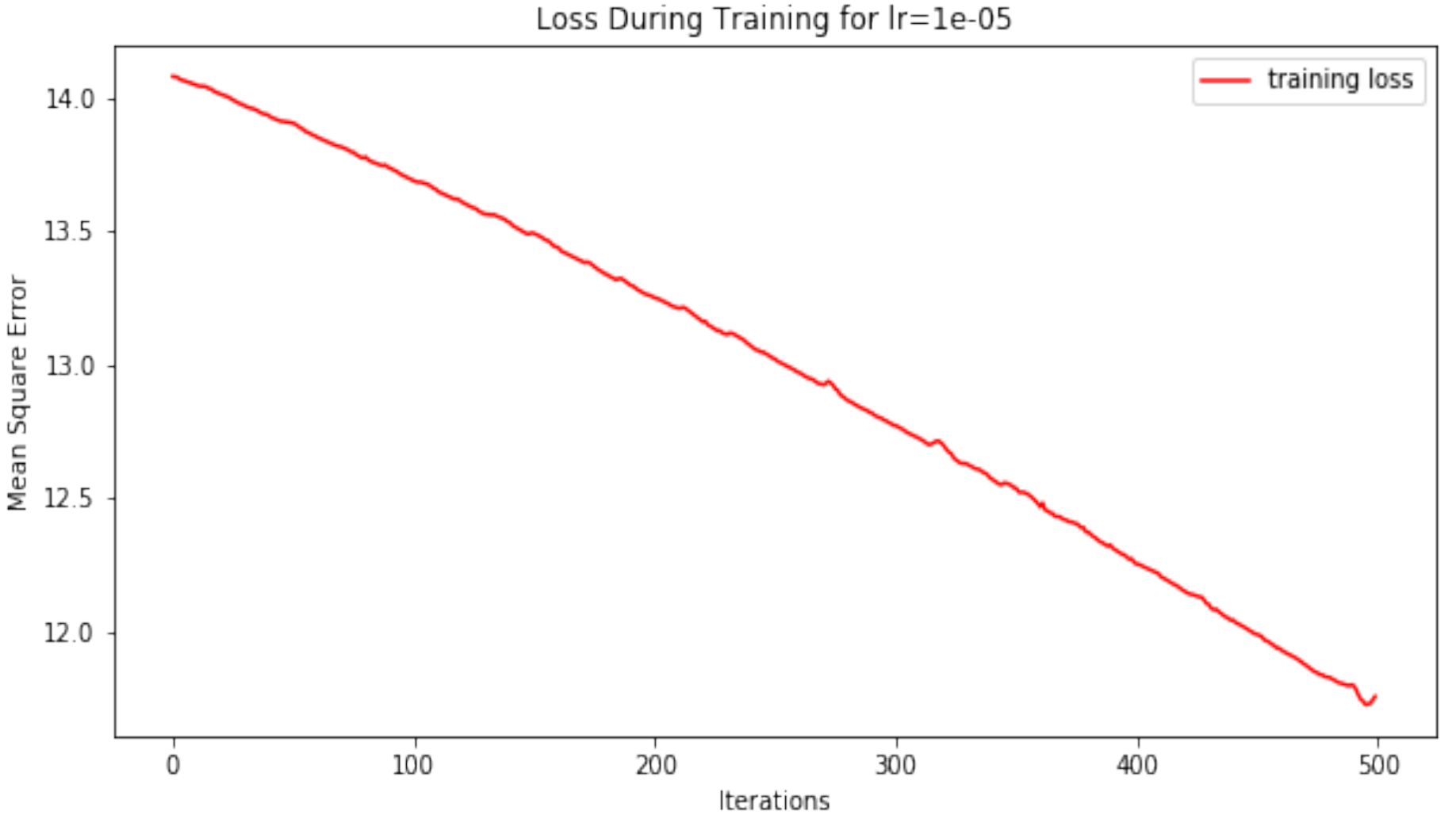
Monitoring Neural Network Training

Visualize the mean square error over the training, this is called the training ***trajectory***.

```
#fit the model and return the mean squared error during training
history = model.fit(x_train, y_train, batch_size=20, shuffle=True, epochs=5
00, verbose=0)

# Plot the loss function and the evaluation metric over the course of train
ing
fig, ax = plt.subplots(1, 1, figsize=(10, 5))

ax.plot(np.array(history.history['loss']), color='blue', label='training lo
ss function')

plt.show()
```

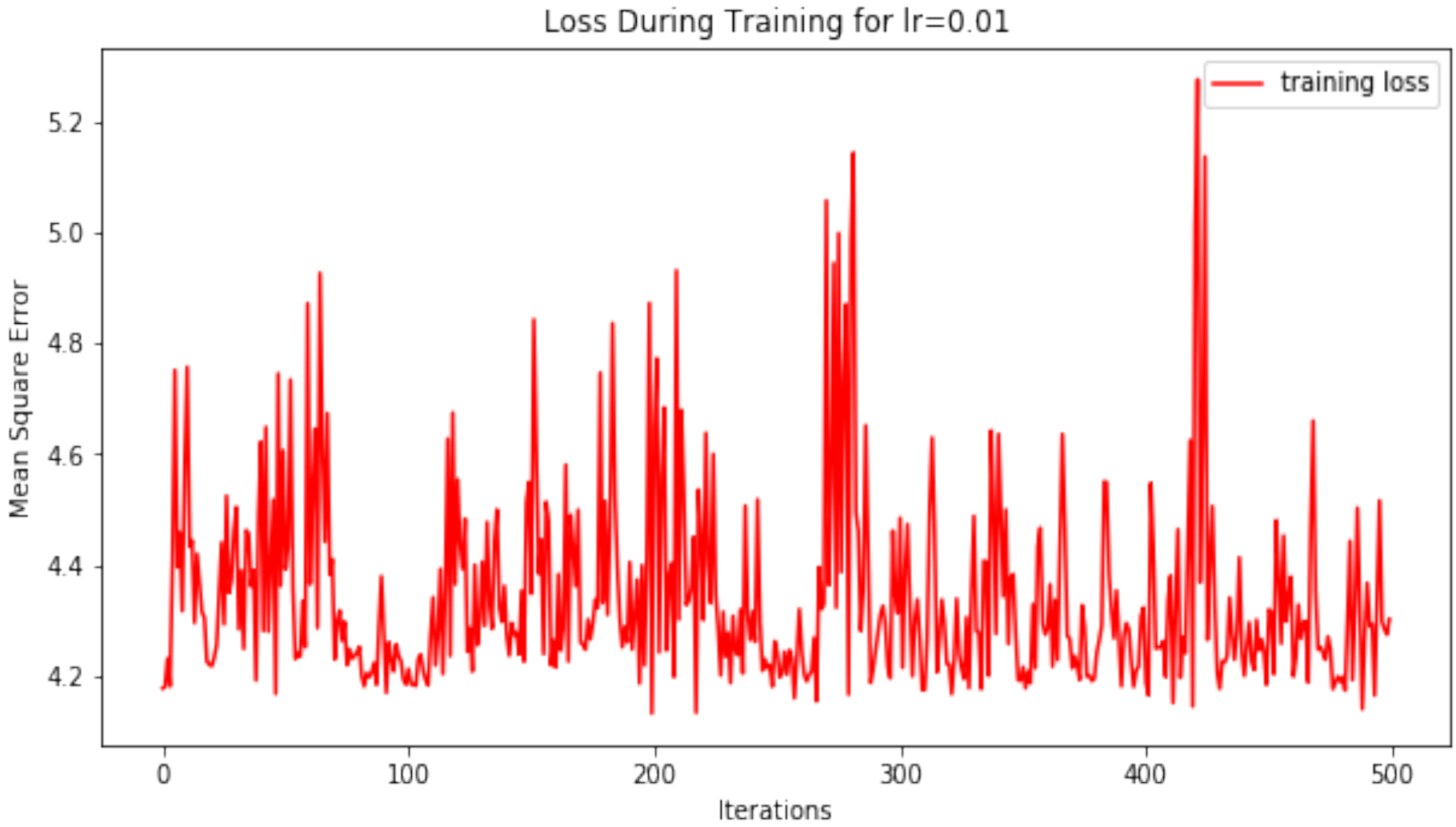
Diagnosing Issues with the Trajectory

If this is your objective function during training, what can you conclude about your step-size?



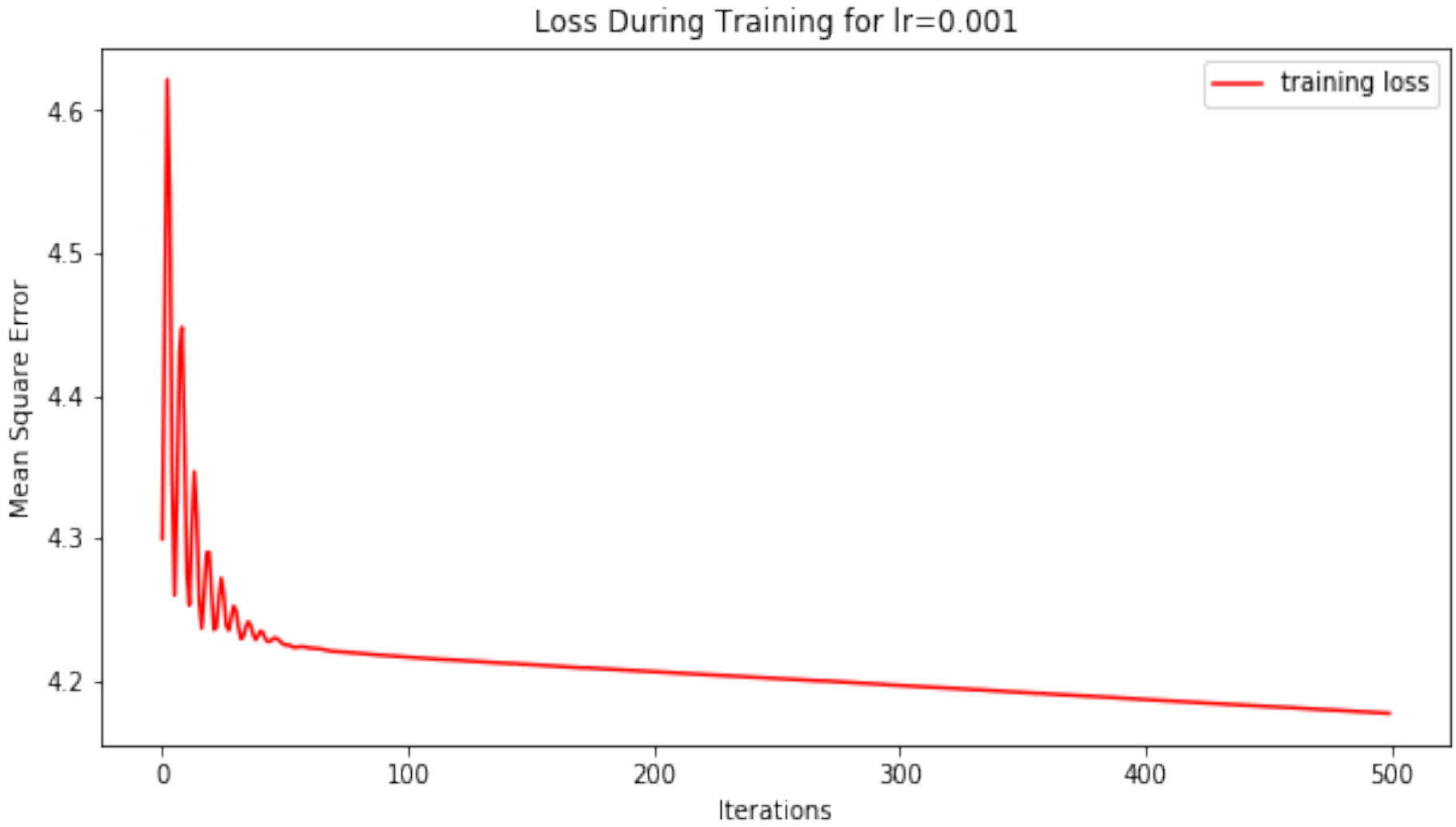
Diagnosing Issues with the Trajectory

If this is your objective function during training, what can you conclude about your step-size?

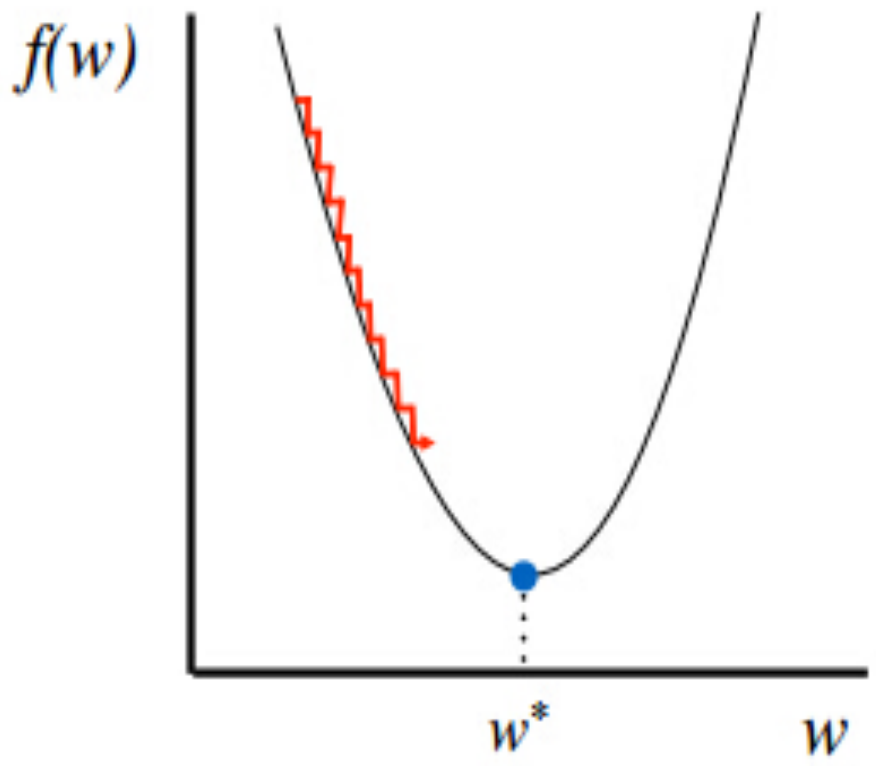


Diagnosing Issues with the Trajectory

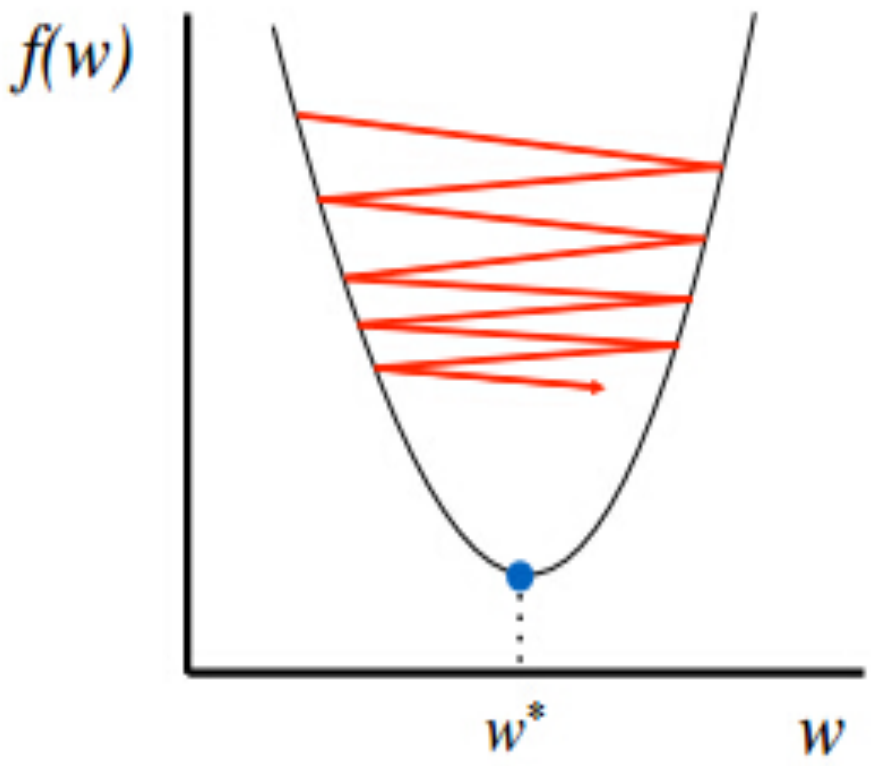
If this is your objective function during training, what can you conclude about your step-size?



Training Your NN: Optimization Choices Matter



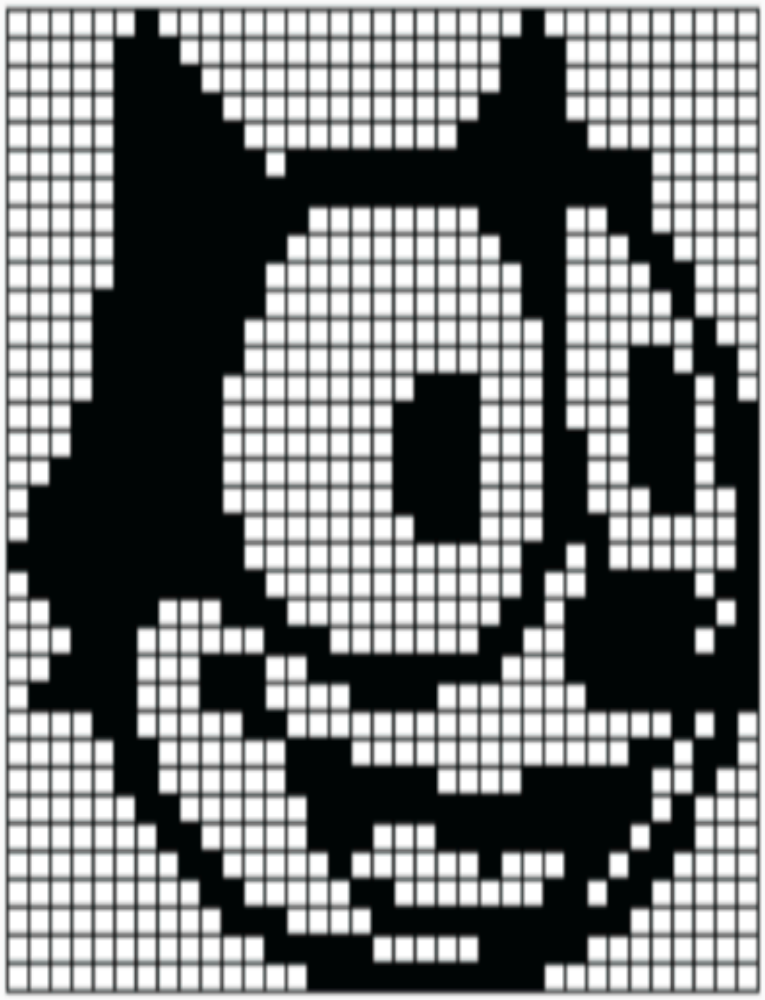
Too small: converge very slowly

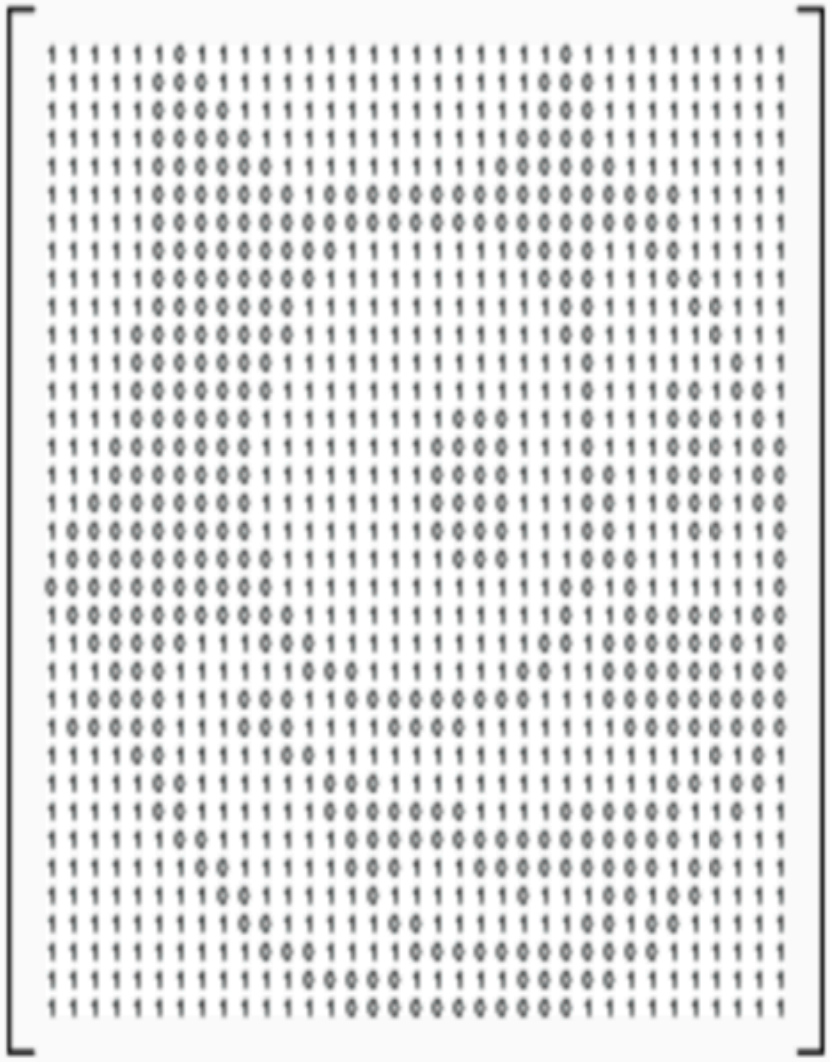


Too big: overshoot and even diverge

Dealing with Image Data

How Do we Represent Gray-scale Images as Numerical Data?





How Do We Represent an Image as a Vector?

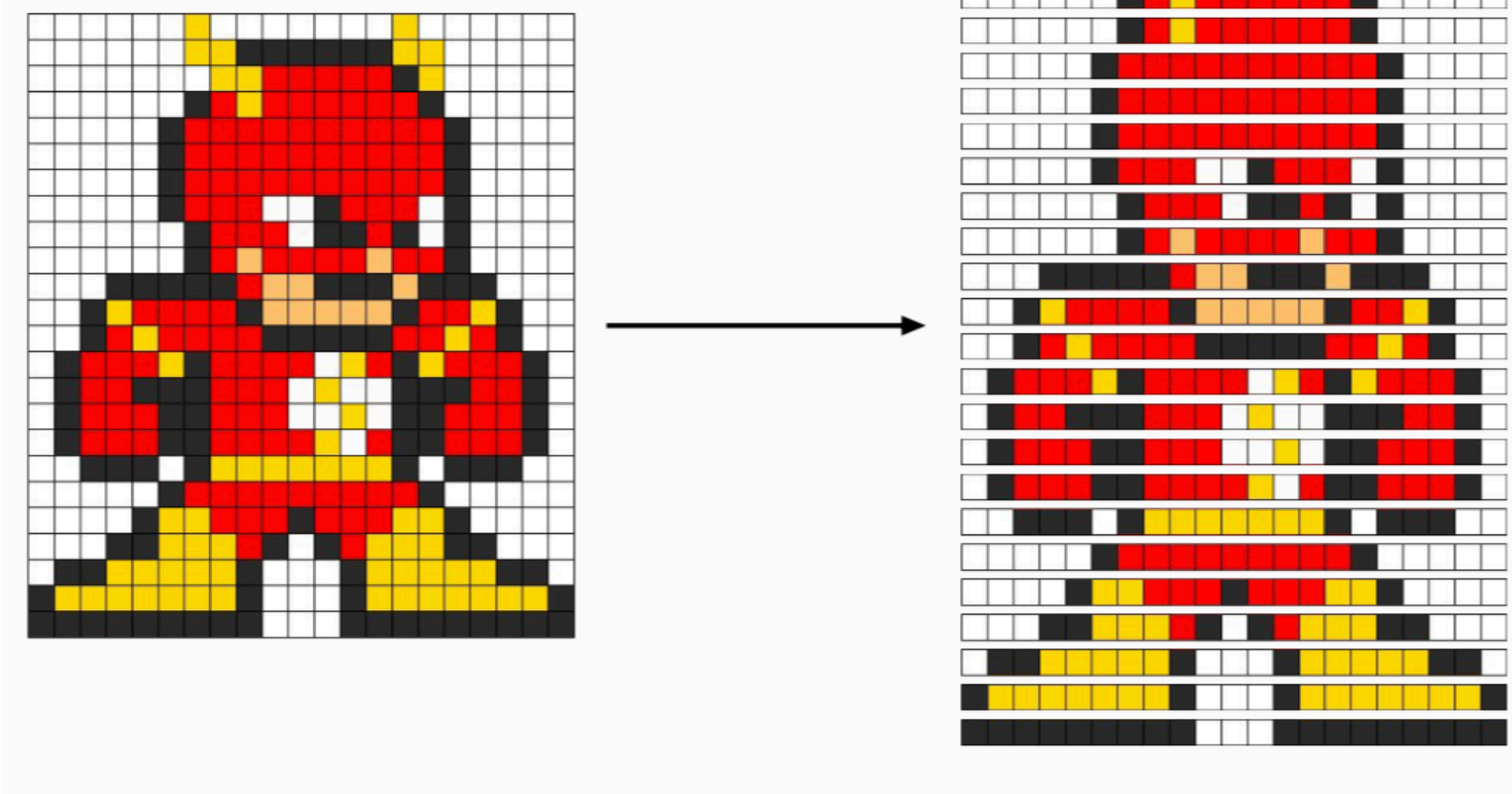
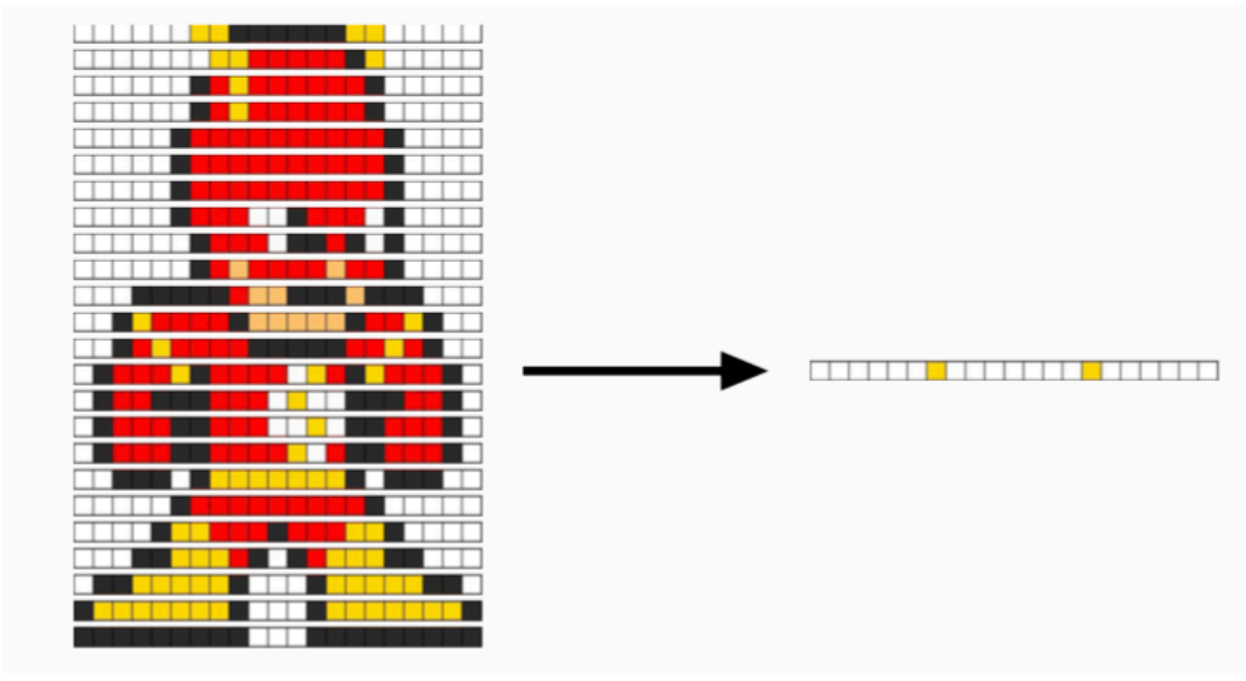


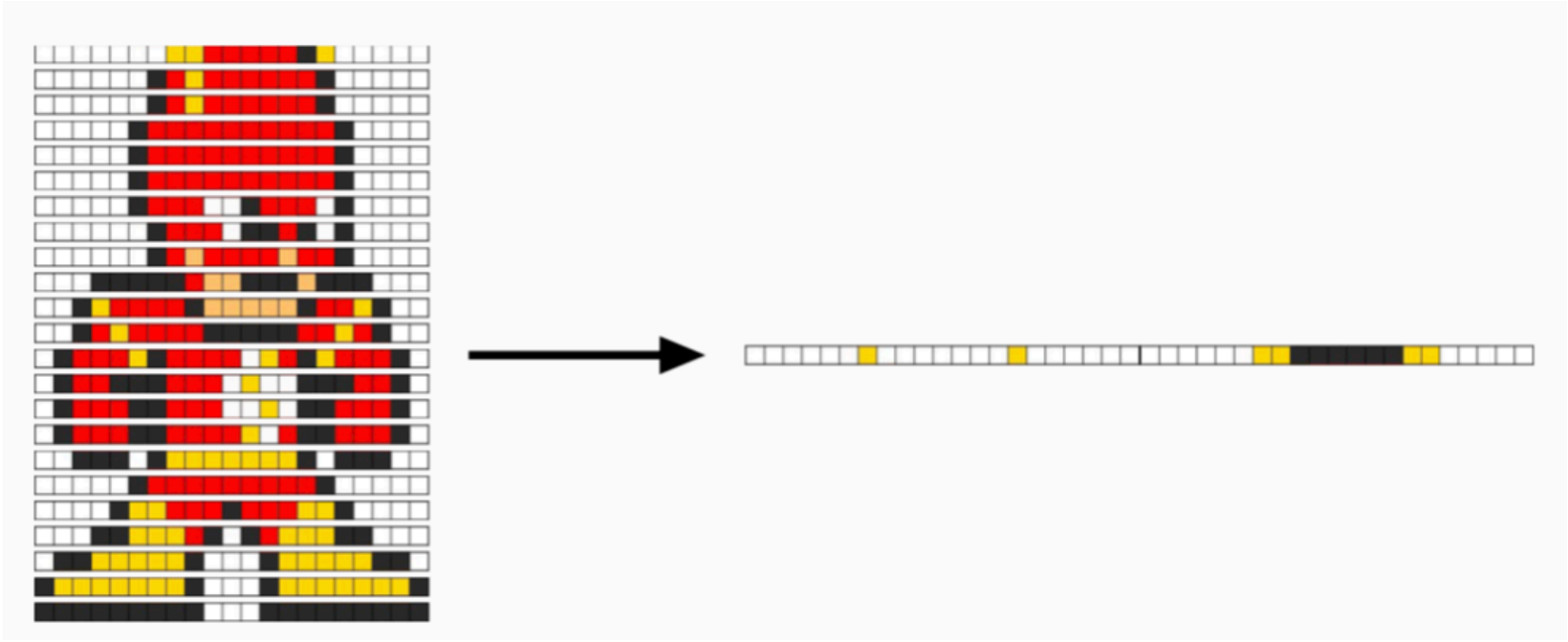
Image as matrix of pixels

Image matrix split along rows

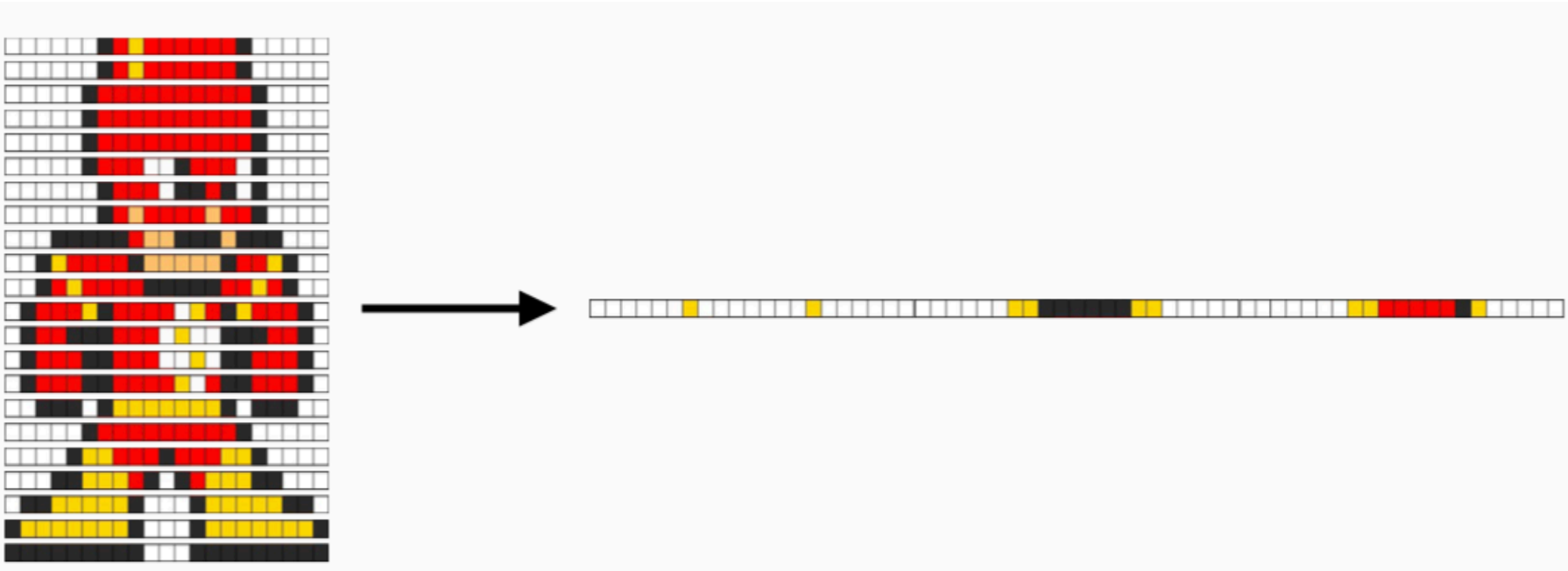
How Do We Represent an Image as a Vector?



How Do We Represent an Image as a Vector?

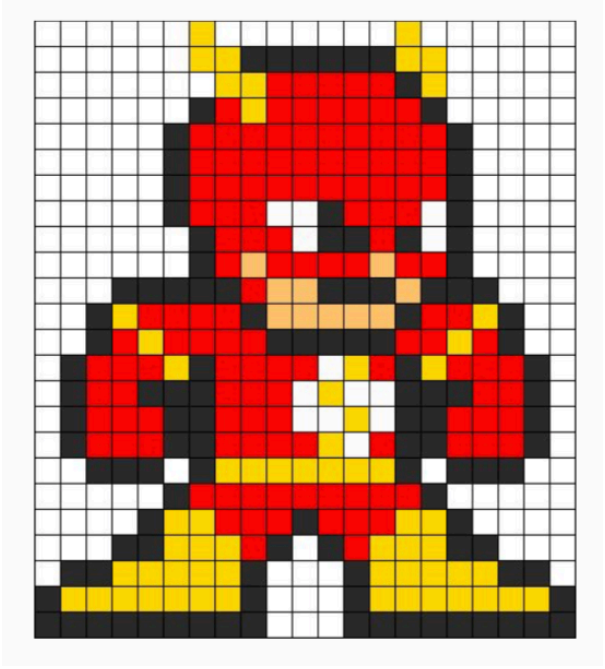


How Do We Represent an Image as a Vector?

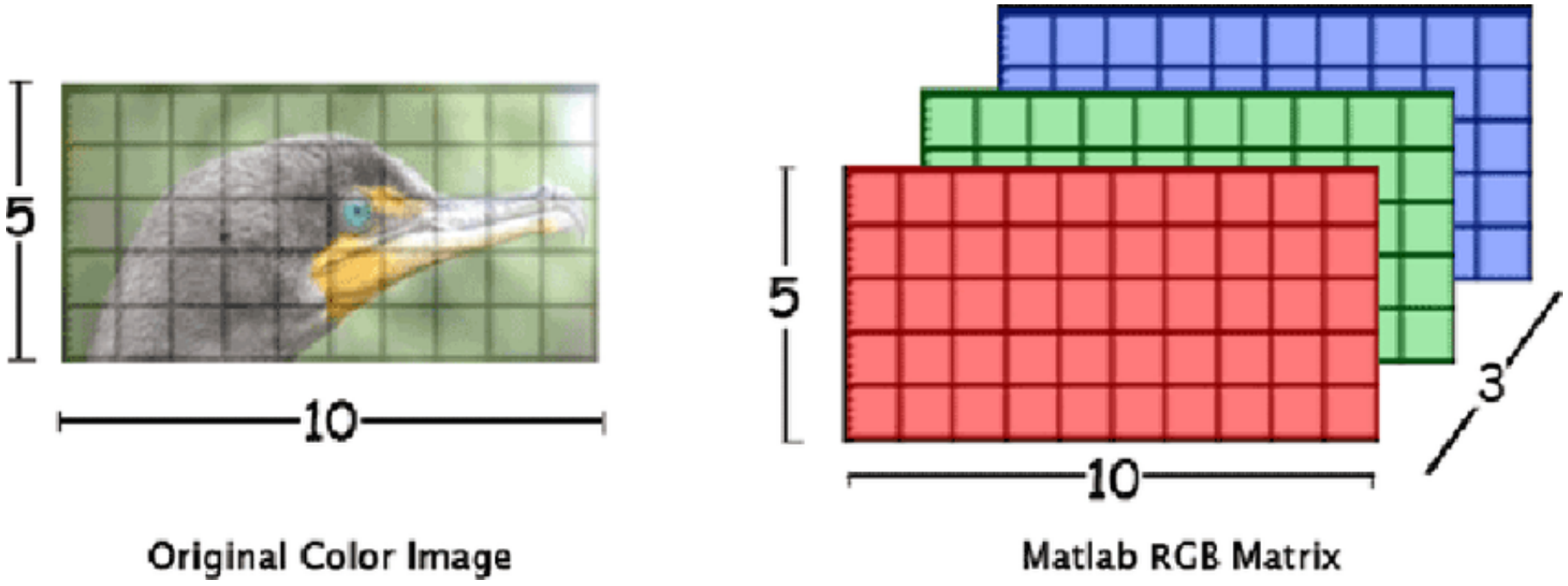


How Do We Represent an Image as a Vector?

This image, when flattened, is represented as a numpy array of shape `(441, )`.



How Do We Represent a Color Image Numerically?



Exercise: