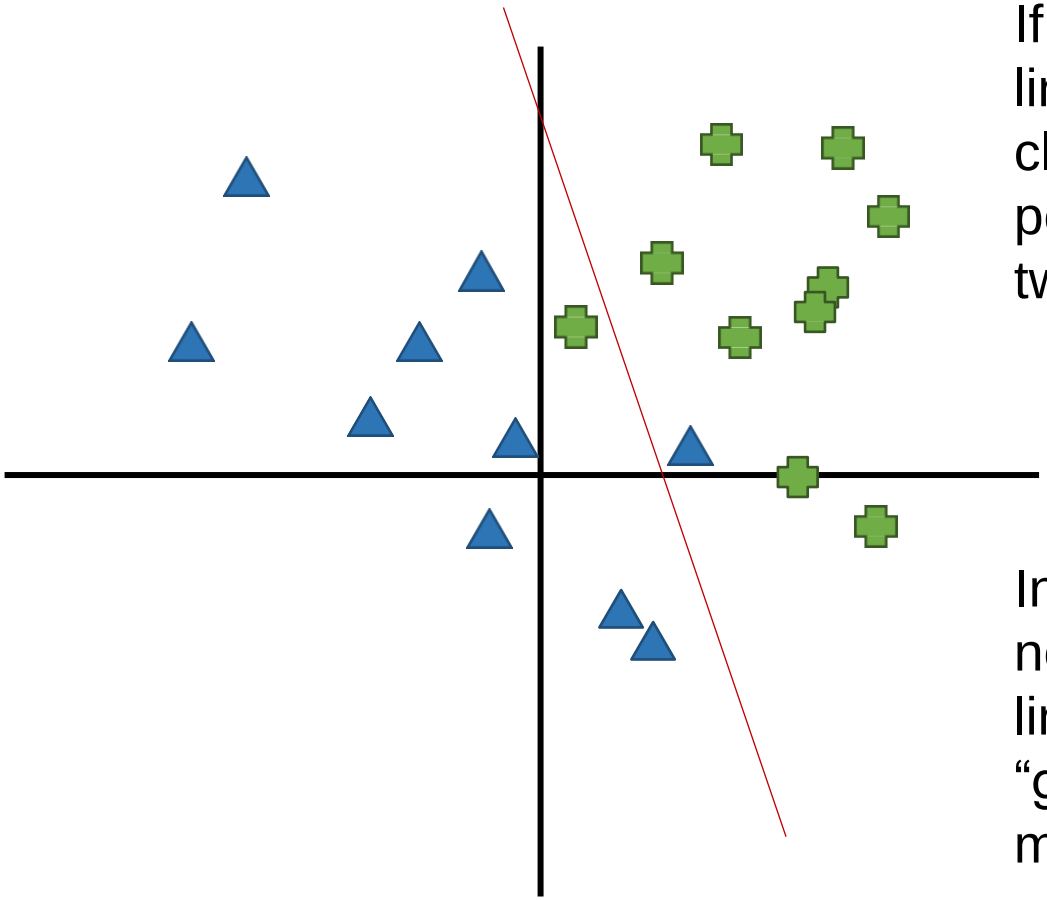
Nonlinear Classification

Linear Classification

Most classifiers we've seen use **linear** functions to separate classes:

- Perceptron
- Logistic regression
- Support vector machines
  (unless kernelized)

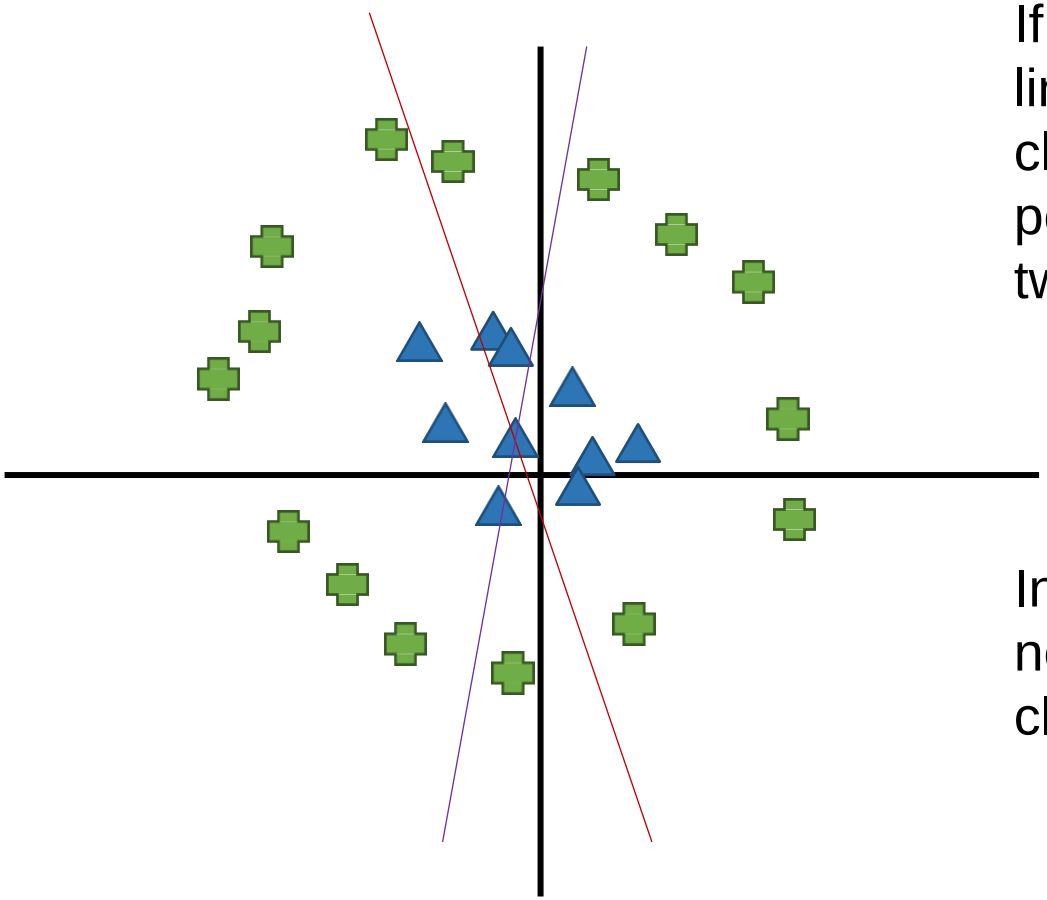
Linear Classification



If the data are not linearly separable, a linear classification cannot perfectly distinguish the two classes.

In many datasets that are not linearly separable, a linear classifier will still be “good enough” and classify most instances correctly.

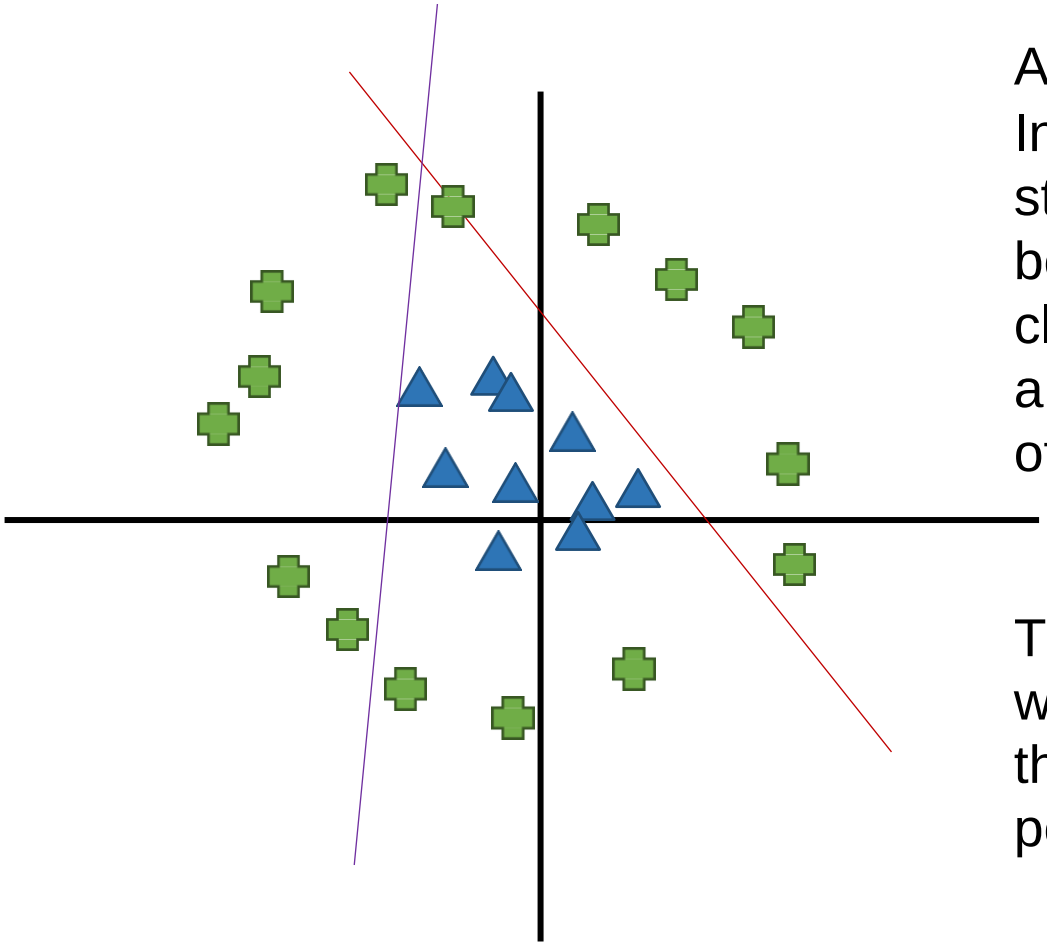
Linear Classification



If the data are not linearly separable, a linear classification cannot perfectly distinguish the two classes.

In some datasets, there is no way to learn a linear classifier that works well.

Linear Classification



Aside:
In datasets like this, it might still be possible to find a boundary that isolates one class, even if the classes are mixed on the other side of the boundary.

This would yield a classifier with decent *precision* on that class, despite having poor overall *accuracy*.

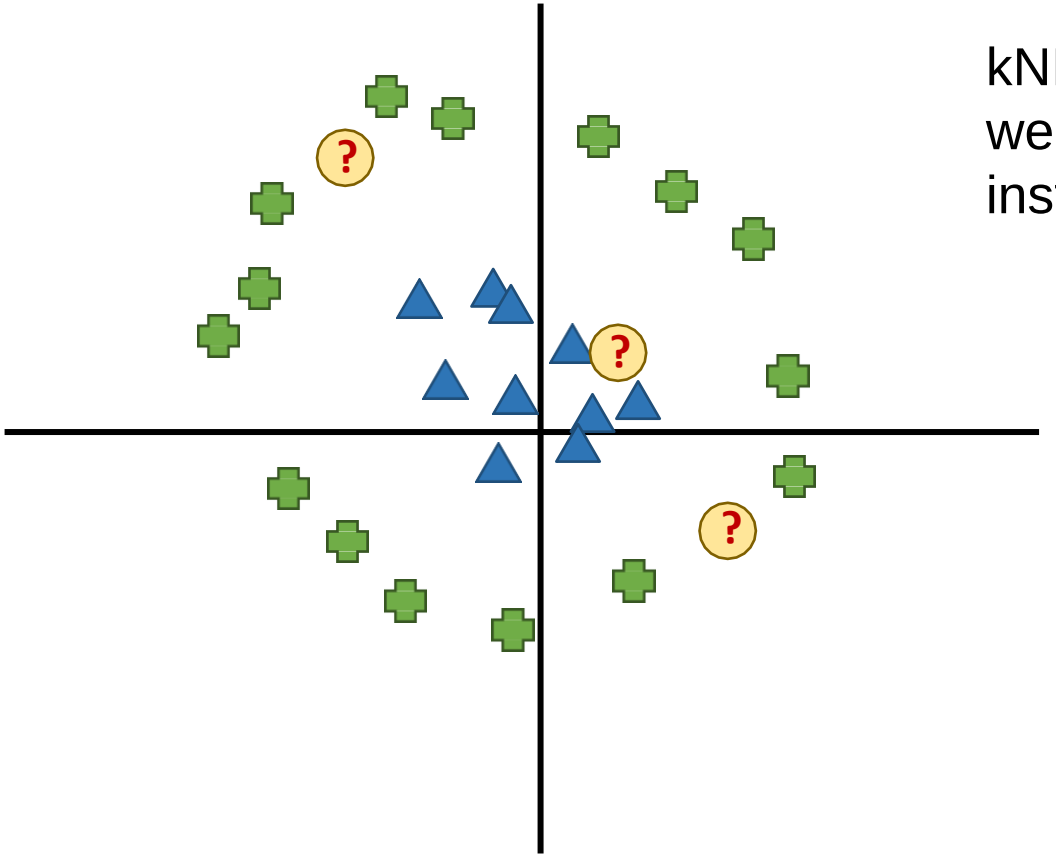
Nonlinear Classification

Nonlinear functions can be used to separate instances that are not linearly separable.

We've seen two nonlinear classifiers:

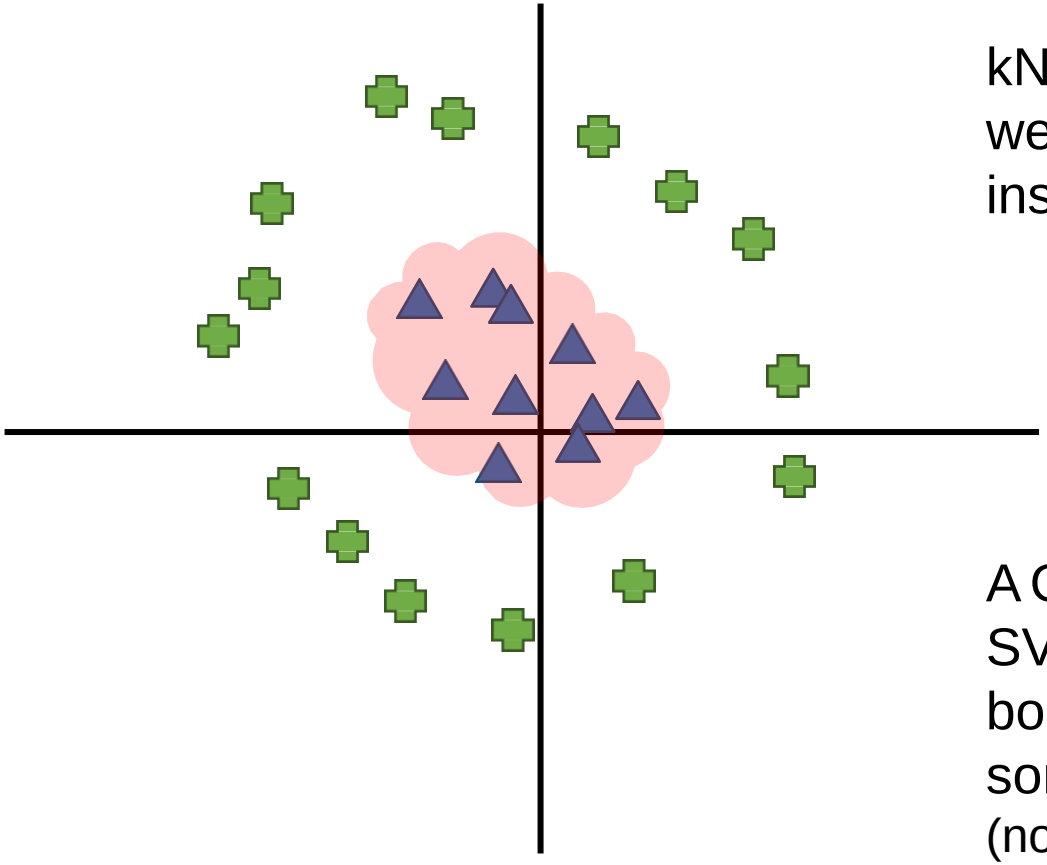
- k-nearest-neighbors (kNN)
- Kernel SVM
  - Kernel SVMs are still implicitly learning a linear separator in a higher dimensional space, but the separator is nonlinear in the original feature space.

Nonlinear Classification



kNN would probably work well for classifying these instances.

Nonlinear Classification



kNN would probably work well for classifying these instances.

A Gaussian/RBF kernel SVM could also learn a boundary that looks something like this.
(not exact; just an illustration)

Nonlinear Classification

Both kNN and kernel methods use the concept of distance/similarity to training instances

Next, we'll see two nonlinear classifiers that make predictions based on features instead of distance/similarity:

- Decision tree
- Multilayer perceptron
  - A basic type of *neural network*

Decision Trees

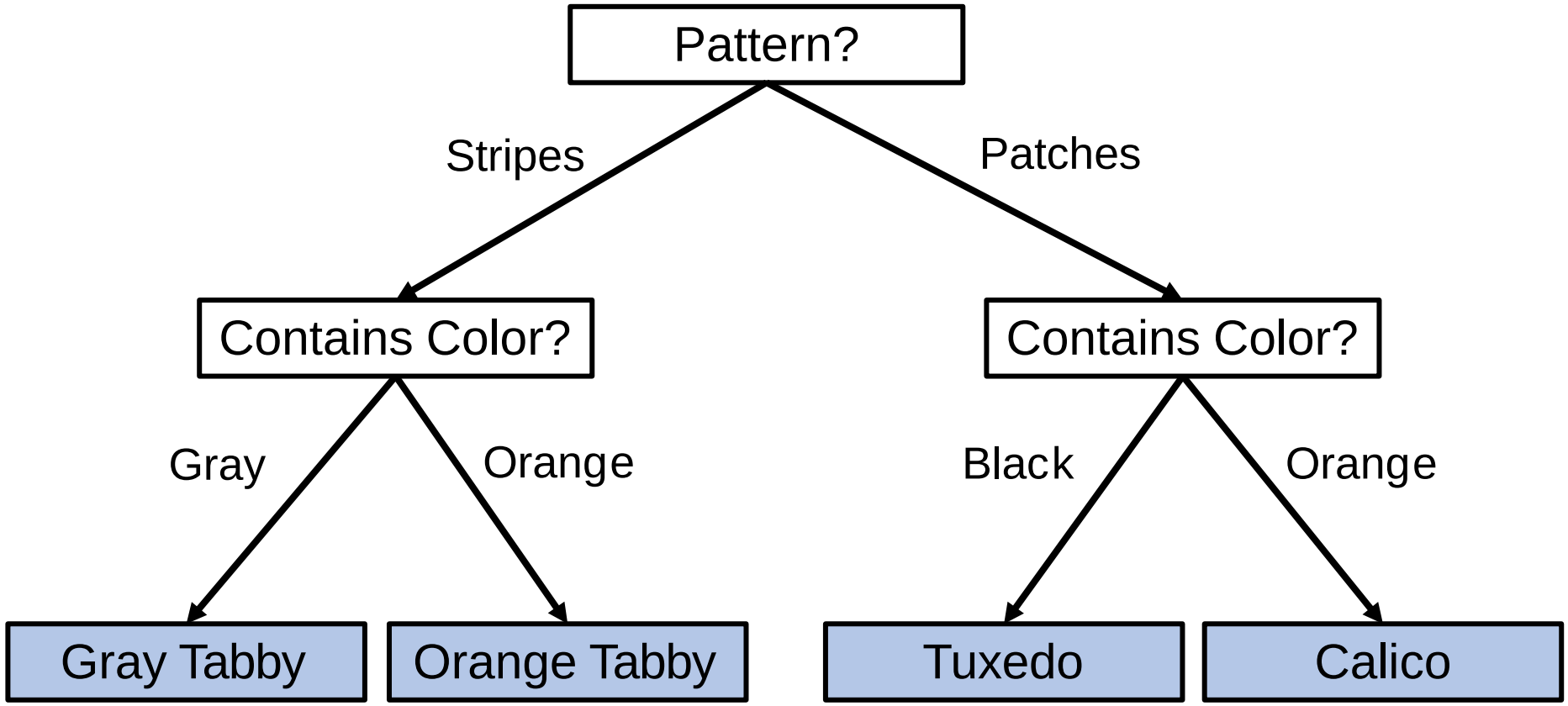
What color is the cat in this photo?

Calico

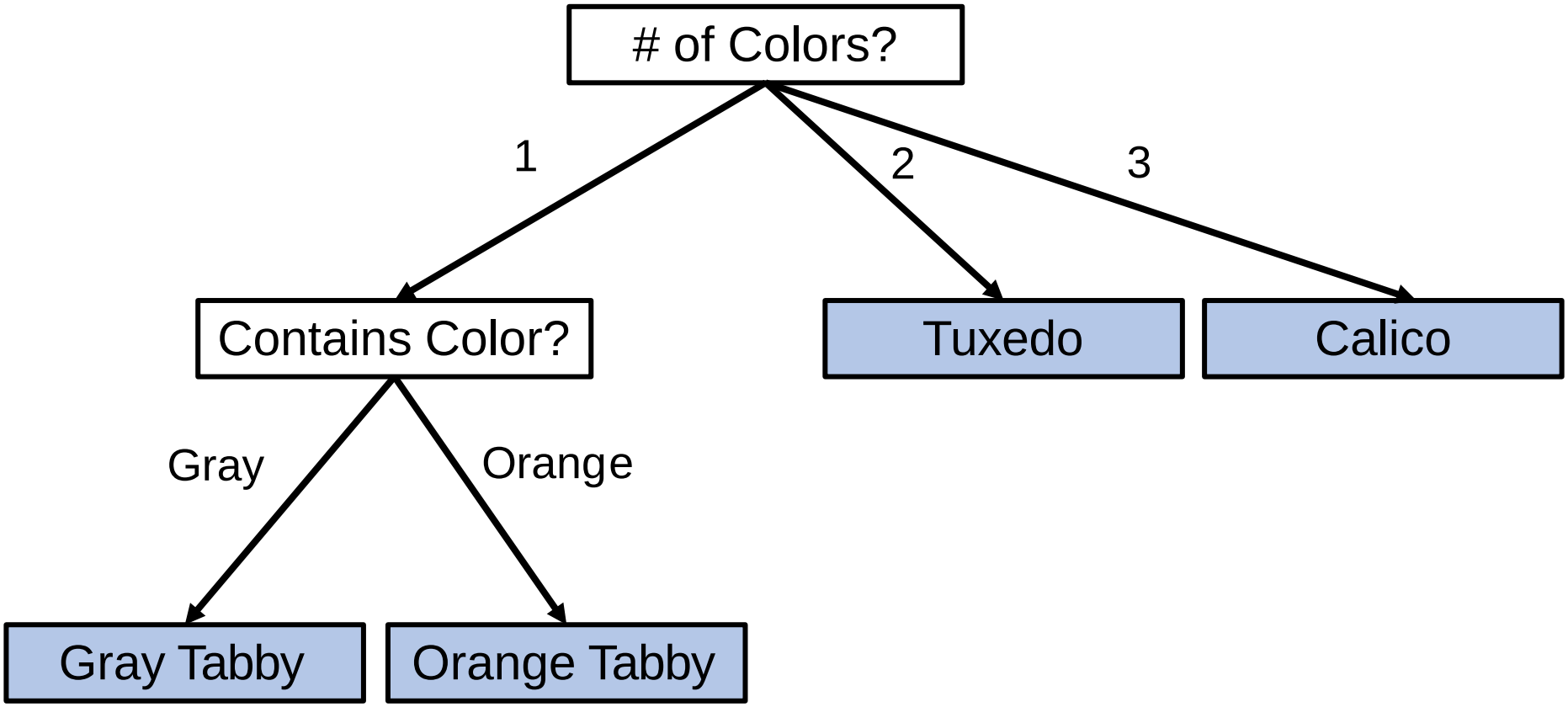
Orange Tabby

Tuxedo

Decision Trees



Decision Trees



Decision Trees

Decision tree classifiers are structured as a **tree**, where:

- *nodes* are features
- *edges* are feature values
  - If the values are numeric, an edge usually corresponds to a range of values (e.g., $x < 2.5$)
- *leaves* are classes

To classify an instance:

Start at the root of the tree, and follow the branches based on the feature values in that instance. The final node is the final prediction.

Decision Trees

We won't cover how to *learn* a decision tree in detail in this class (see book for more detail)

General idea:

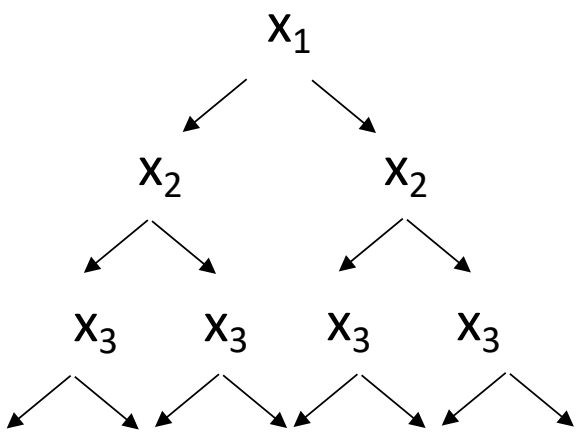
1. Pick the feature that best distinguishes classes
   - If you group the instances based on their value for that feature, some classes should become more likely
   - The distribution of classes should have low *entropy* (high entropy means the classes are evenly distributed)
2. Recursively repeat for each group of instances
3. When all instances in a group have the same label, set that class as the final node.

Decision Trees

Decision trees can easily overfit.

Without doing anything extra, they will literally memorize training data!

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



A tree can encode all possible combinations of feature values.

Decision Trees

Two common techniques to avoid overfitting:

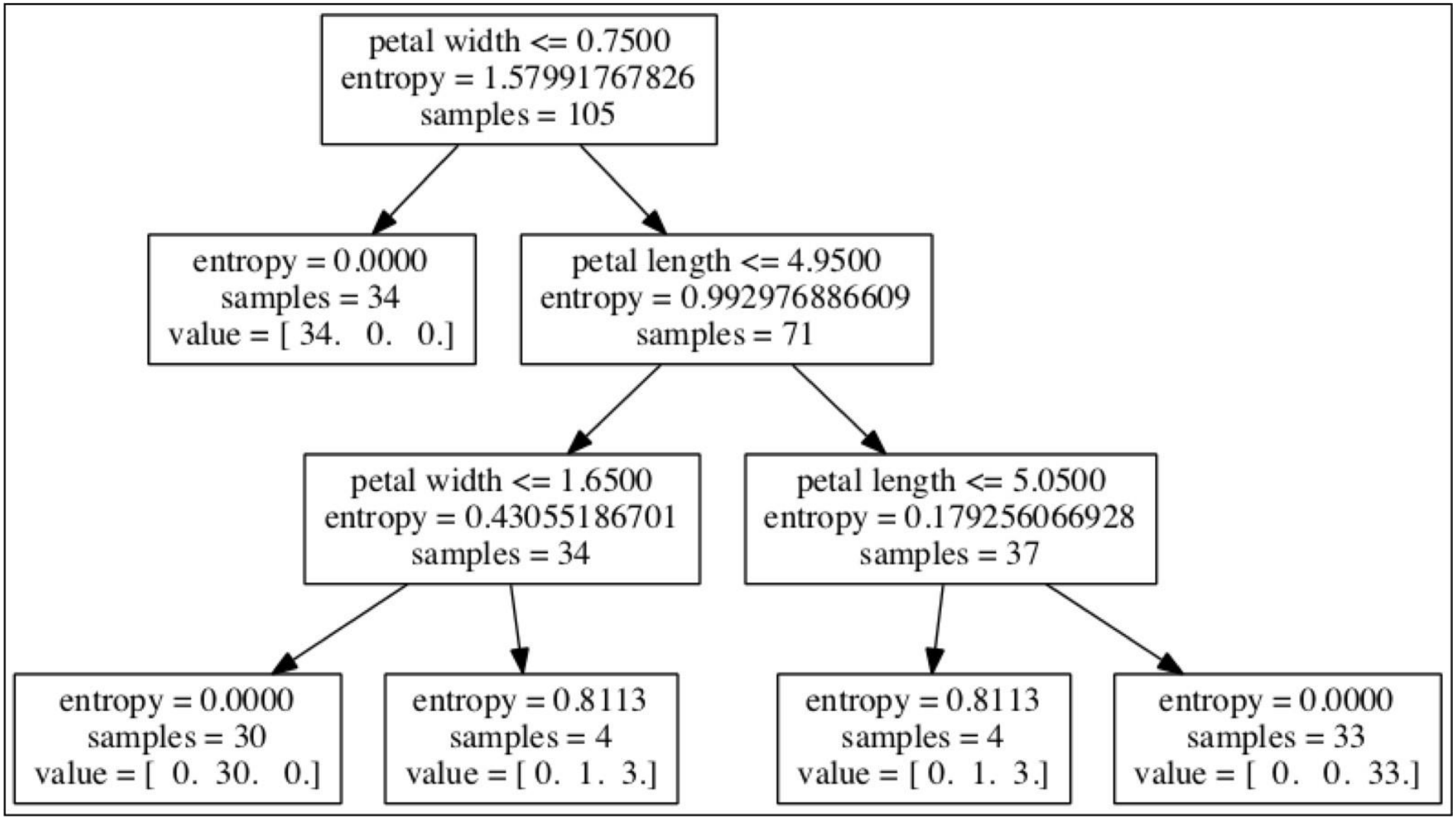
- Restrict the maximum depth of the tree
- Restrict the minimum number of instances that must be remaining before splitting further

If you stop creating a deeper tree before all instances at that node belong to the same class, use the majority class within that group as the final class for the leaf node.

Decision Trees

One reason decision trees are popular is because the algorithm is relatively easy to understand, and the classifiers are relatively **interpretable**.

- That is, it is possible to see how a classifier makes a decision.
- This stops being true if your decision tree becomes large; trees of depth 3 or less are easiest to visualize.


petal width <= 0.7500
entropy = 1.57991767826
samples = 105
entropy = 0.0000
samples = 34
value = [34. 0. 0.]
petal length <= 4.9500
entropy = 0.992976886609
samples = 71
petal width <= 1.6500
entropy = 0.43055186701
samples = 34
petal length <= 5.0500
entropy = 0.179256066928
samples = 37
entropy = 0.0000
samples = 30
value = [0. 30. 0.]
entropy = 0.8113
samples = 4
value = [0. 1. 3.]
entropy = 0.8113
samples = 4
value = [0. 1. 3.]
entropy = 0.0000
samples = 33
value = [0. 0. 33.]

Decision Trees

Decision trees are powerful because they automatically use *conjunctions* of features (e.g., *Pattern*=“striped” AND *Color*=“Orange”)

This gives *context* to the feature values.

- In a decision tree, *Color*=“Orange” can lead to a different prediction depend on the value of the *Pattern* feature.
- In a linear classifier, only one weight can be given to *Color*=“Orange”; it can’t be associated with different classes in different contexts

Decision Trees

Decision trees naturally handle **multiclass** classification without making any modifications

Decision trees can also be used for **regression** instead of classification

- Common implementation: final prediction is the average value of all instances at the leaf

Random Forests

Random forests are a type of **ensemble** learning with decision trees (a *forest* is a set of trees)

We'll revisit this later in the semester, but it's useful to know that random forests:

- are one of the most successful types of ensemble learning
- avoid overfitting better than individual decision trees

Neural Networks

Recall from the book that perceptron was inspired by the way neurons work:

- a perceptron “fires” only if the inputs sum above a threshold (that is, a perceptron outputs a positive label if the score is above the threshold; negative otherwise)

Also recall that a perceptron is also called an *artificial neuron*.

Neural Networks

An **artificial neural network** is a collection of artificial neurons that interact with each other

- The outputs of some are used as inputs for others

A **multilayer perceptron** is one type of neural network which combines multiple perceptrons

- Multiple *layers* of perceptrons, where each layer's output "feeds" into the next layer as input
- Called a *feed-forward* network

Perceptron Learning Algorithm

Perceptron

Perceptron was introduced by Frank Rosenblatt in 1957.
He proposed a Perceptron learning rule based on the original MCP neuron.
A Perceptron is an algorithm for supervised learning of binary classifiers.
This algorithm enables neurons to learn and processes elements in the training set one at a time.

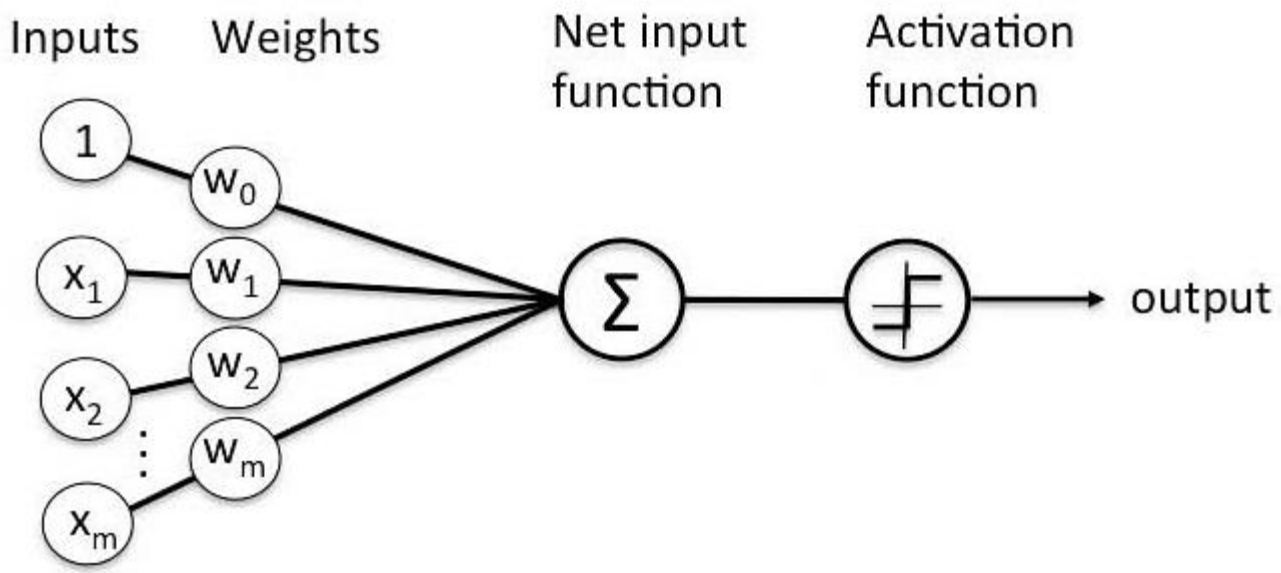


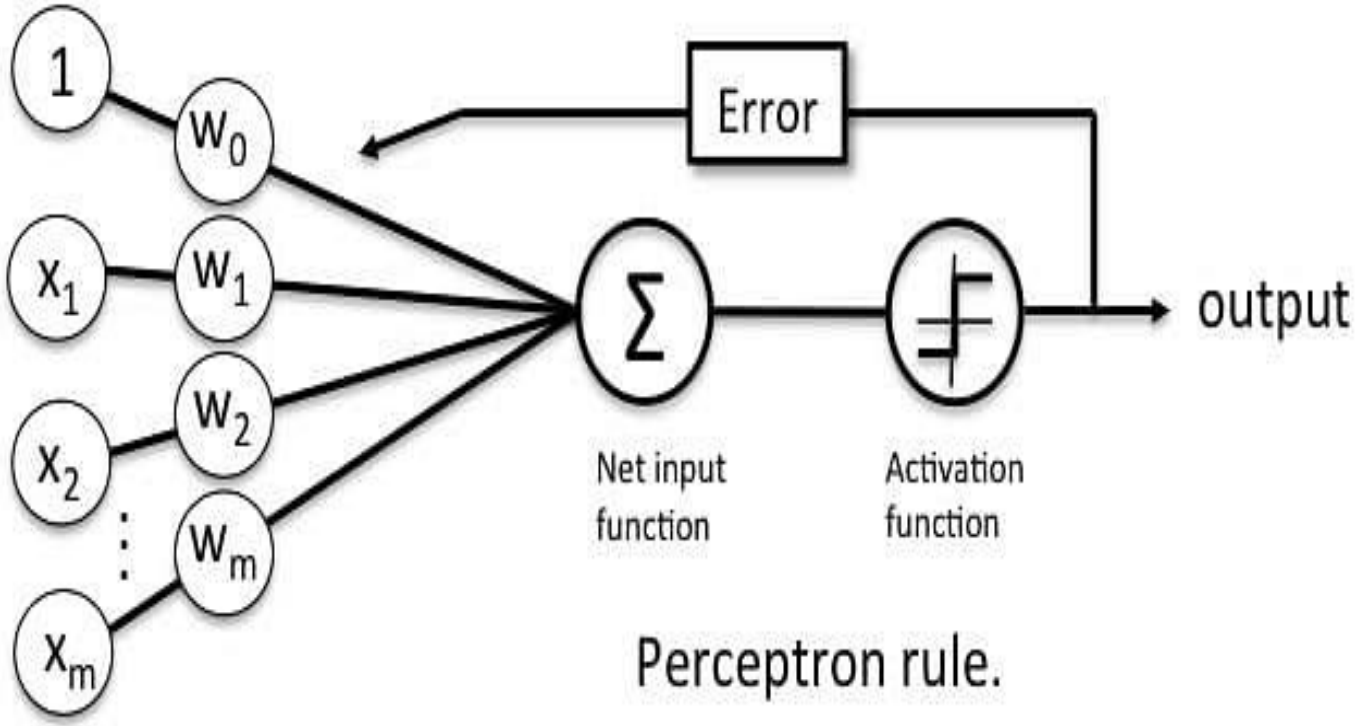
Figure : Perceptron

There are two types of Perceptrons: Single layer and Multilayer.

Single layer - Single layer perceptrons can learn only linearly separable patterns
Multilayer - Multilayer perceptrons or feedforward neural networks with two or more layers have the greater processing power

Perceptron Learning Rule

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients.
The input features are then multiplied with these weights to determine if a neuron fires or not.
The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output.
In the context of supervised learning and classification, this can then be used to predict the class of a sample.



Perceptron rule.

Perceptron Function

Perceptron is a function that maps its input “x,” which is multiplied with the learned weight coefficient; an output value ”f(x)”is generated.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

In the equation given above:
“w” = vector of real-valued weights
“b” = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
“x” = vector of input x values $\sum_{i=1}^{m} w_i x_i$

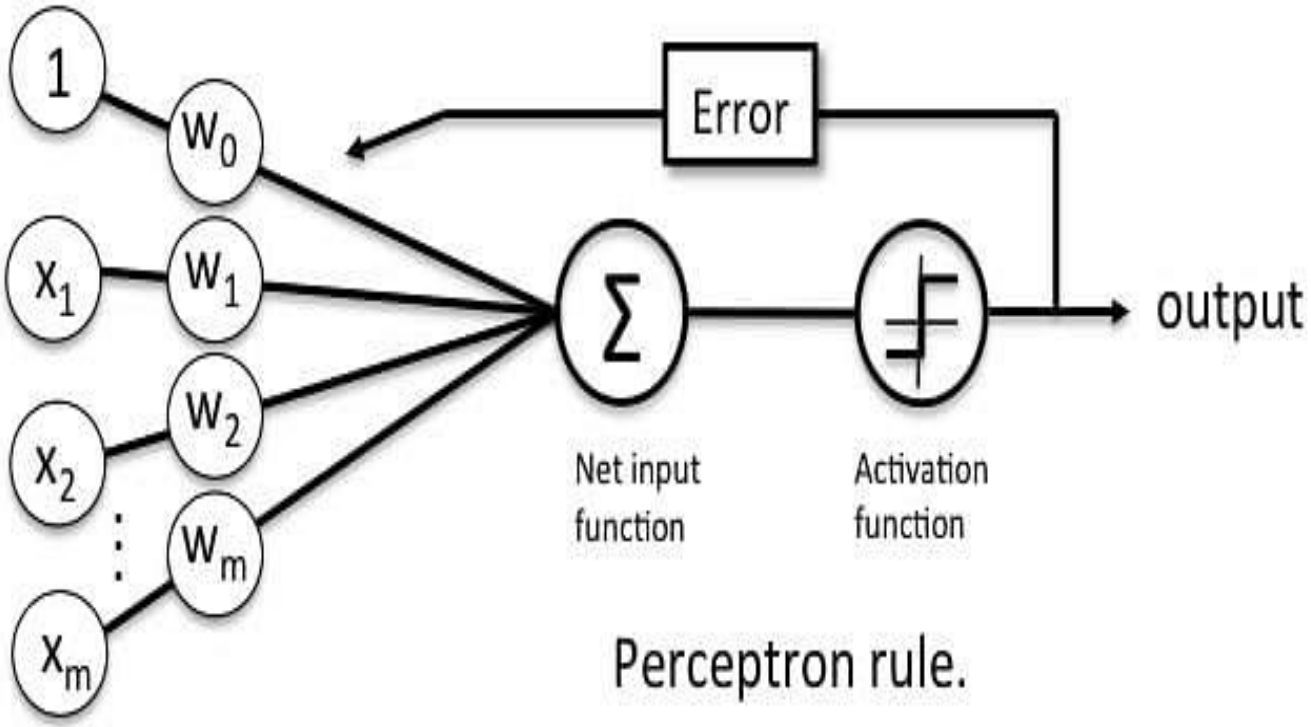
“m” = number of inputs to the Perceptron
The output can be represented as “1” or “0.” It can also be represented as “1” or “-1” depending on which activation function is used.

Inputs of a Perceptron

- A Perceptron accepts inputs, moderates them with certain weight values, then applies the transformation function to output the final result.
- A Boolean output is based on inputs such as salaried, married, age, past credit profile, etc. It has only two values: Yes and No or True and False. The summation function "∑" multiplies all inputs of "x" by weights "w" and then adds them up as follows:

$$w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

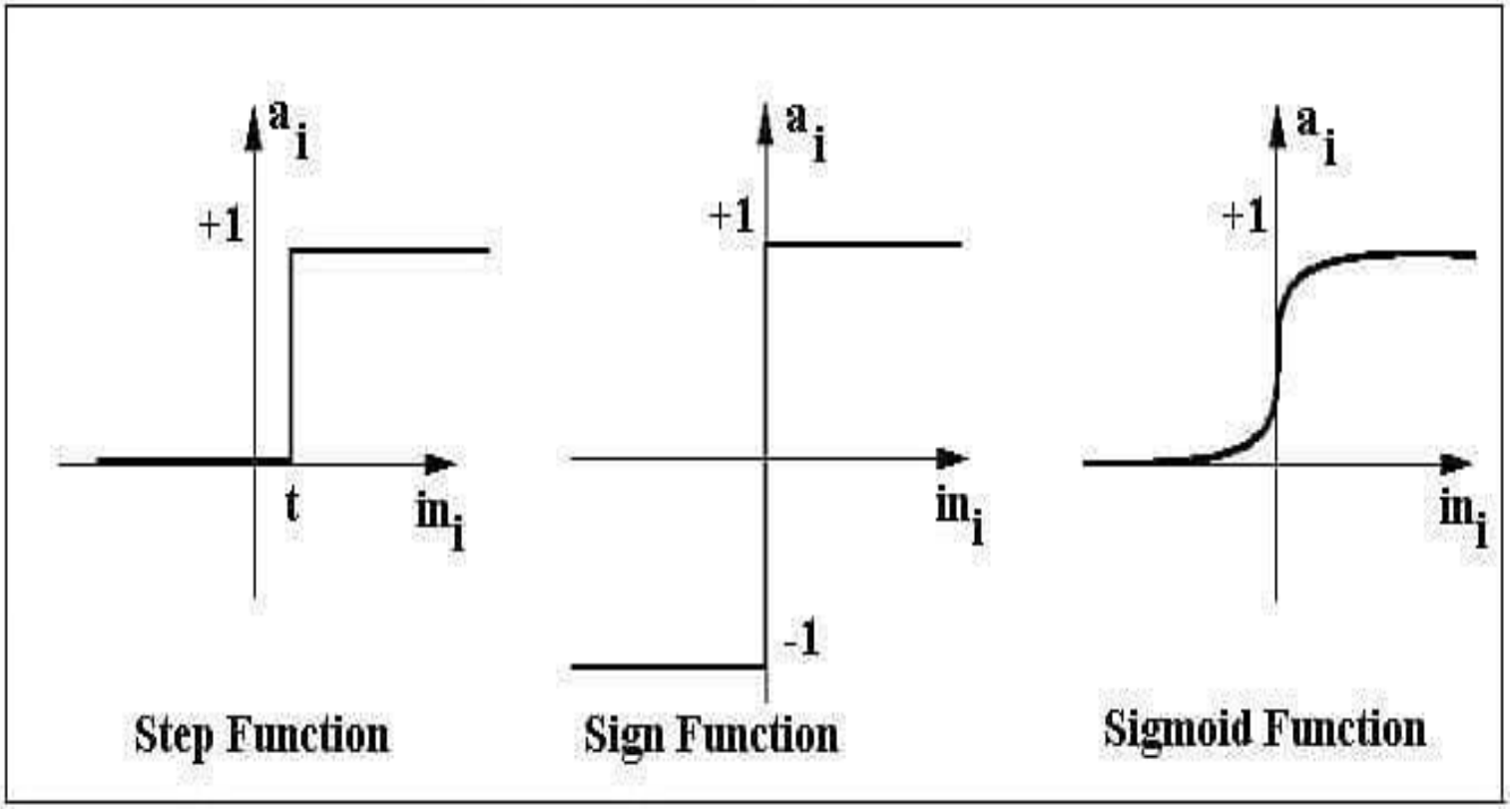
Perceptron with a Boolean output



Activation Functions of Perceptron

The activation function applies a step rule (convert the numerical output into +1 or -1) to check if the output of the weighting function is greater than zero or not.
For example:
If $\sum$ wixi> 0 => then final output “o” = 1 (issue bank loan)
Else, final output “o” = -1 (deny bank loan)
Step function gets triggered above a certain value of the neuron output; else it outputs zero. Sign Function outputs +1 or -1 depending on whether neuron output is greater than zero or not. Sigmoid is the S-curve and outputs a value between 0 and 1.

Activation Functions of Perceptron



Output of Perceptron

Perceptron with a Boolean output:
Inputs: x1…xn
Output: o(x1….xn)
Weights: wi=> contribution of input xi to the Perceptron output; w0=> bias or threshold
If ∑w.x > 0, output is +1, else -1. The neuron gets triggered only when weighted input reaches a certain threshold value.

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

An output of +1 specifies that the neuron is triggered. An output of -1 specifies that the neuron did not get triggered. “sgn” stands for sign function with output +1 or -1.

Error in Perceptron

In the Perceptron Learning Rule, the predicted output is compared with the known output.
If it does not match, the error is propagated backward to allow weight adjustment to happen.

Perceptron: Decision Function

A decision function φ(z) of Perceptron is defined to take a linear combination of x and w vectors

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

The value z in the decision function is given by:

$$z = w_1 x_1 + \ldots + w_m x_m$$

The decision function is +1 if z is greater than a threshold θ, and it is -1 otherwise.

$$\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

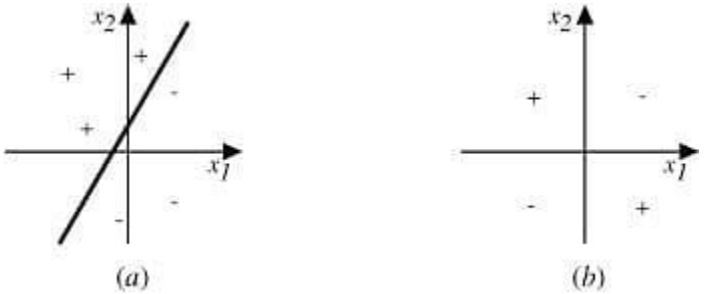
This is the Perceptron algorithm.

Implement Logic Gates with Perceptron

Perceptron - Classifier Hyperplane
The Perceptron learning rule converges if the two classes can be separated by the linear hyper plane. However, if the classes cannot be separated perfectly by a linear classifier, it could give rise to errors.

$$\vec{w} \cdot \vec{x} = 0$$

As discussed in the previous topic, the classifier boundary for a binary output in a Perceptron is represented by the equation given below:
The diagram above shows the decision surface represented by a two-input Perceptron.



Observation:
In Fig(a) above, examples can be clearly separated into positive and negative values; hence, they are linearly separable. This can include logic gates like AND, OR, NOR, NAND.
Fig (b) shows examples that are not linearly separable (as in an XOR gate).
Diagram (a) is a set of training examples and the decision surface of a Perceptron that classifies them correctly.
Diagram (b) is a set of training examples that are not linearly separable, that is, they cannot be correctly classified by any straight line.
X1 and X2 are the Perceptron inputs.
In the next section, let us talk about logic gates.

What is Logic Gate?

Logic gates are the building blocks of a digital system, especially neural networks. In short, they are the electronic circuits that help in addition, choice, negation, and combination to form complex circuits. Using the logic gates, Neural Networks can learn on their own without you having to manually code the logic. Most logic gates have two inputs and one output.
Each terminal has one of the two binary conditions, low (0) or high (1), represented by different voltage levels. The logic state of a terminal changes based on how the circuit processes data.

Based on this logic, logic gates can be categorized into seven types:
AND
NAND
OR
NOR
NOT
XOR
XNOR

Implementing Basic Logic Gates With Perceptron

1. AND

If the two inputs are TRUE (+1), the output of Perceptron is positive, which amounts to TRUE. This is the desired behavior of an AND gate.

x1= 1 (TRUE), x2= 1 (TRUE)

w0 = -.8, w1 = 0.5, w2 = 0.5

=> o(x1, x2) => -.8 + 0.5*1 + 0.5*1 = 0.2 > 0

Implementing Basic Logic Gates With Perceptron

2. OR

If either of the two inputs are TRUE (+1), the output of Perceptron is positive, which amounts to TRUE. This is the desired behavior of an OR gate.

x1 = 1 (TRUE), x2 = 0 (FALSE)

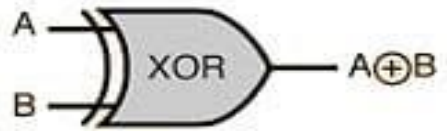
w0 = -.3, w1 = 0.5, w2 = 0.5

=> o(x1, x2) => -.3 + 0.5*1 + 0.5*0 = 0.2 > 0

Implementing Basic Logic Gates With Perceptron

3. XOR

A XOR gate, also called as Exclusive OR gate, has two inputs and one output.



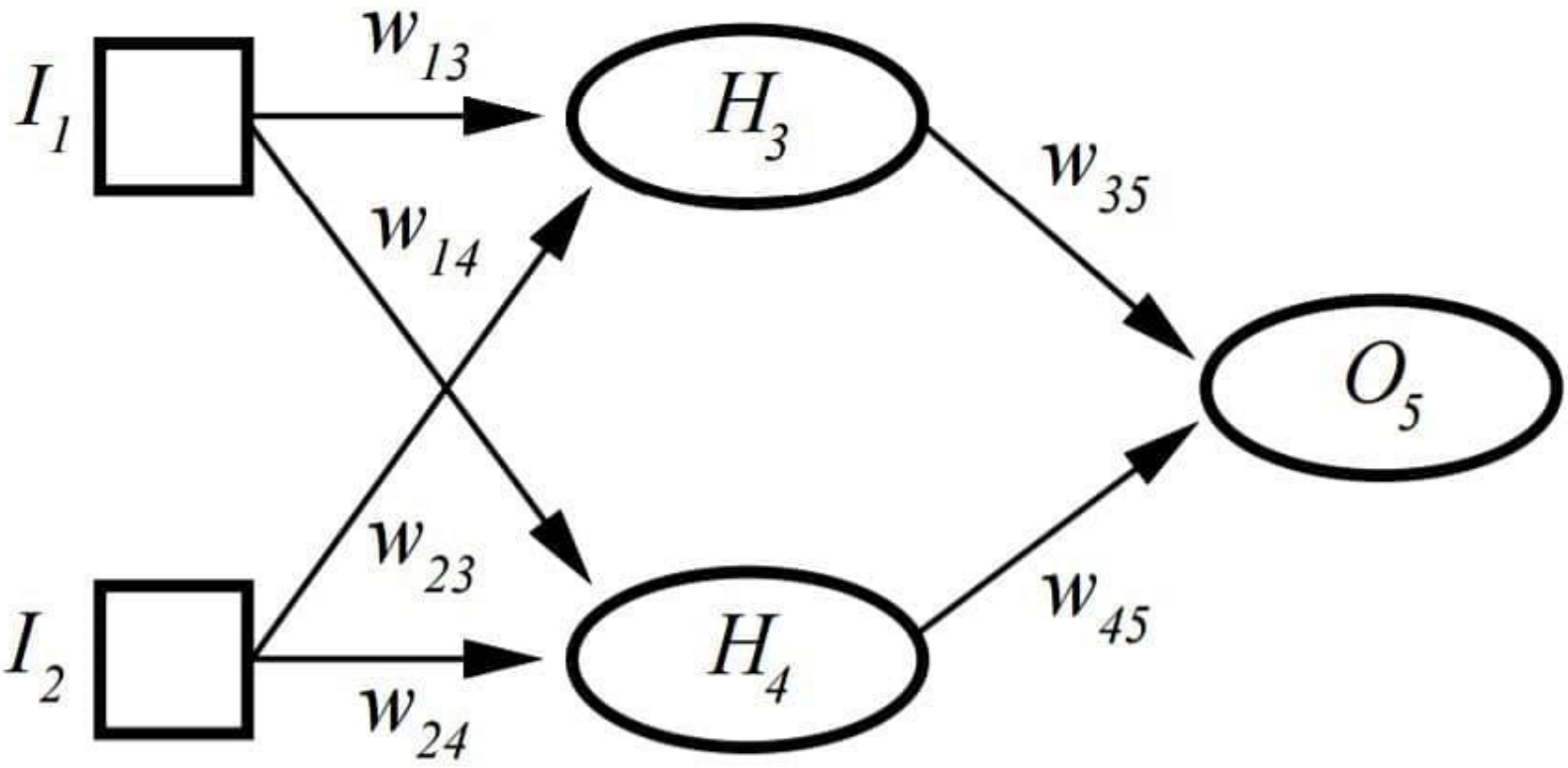
The gate returns a TRUE as the output if and ONLY if one of the input states is true.

XOR Truth Table

Input	Output	
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

XOR Gate with Neural Networks

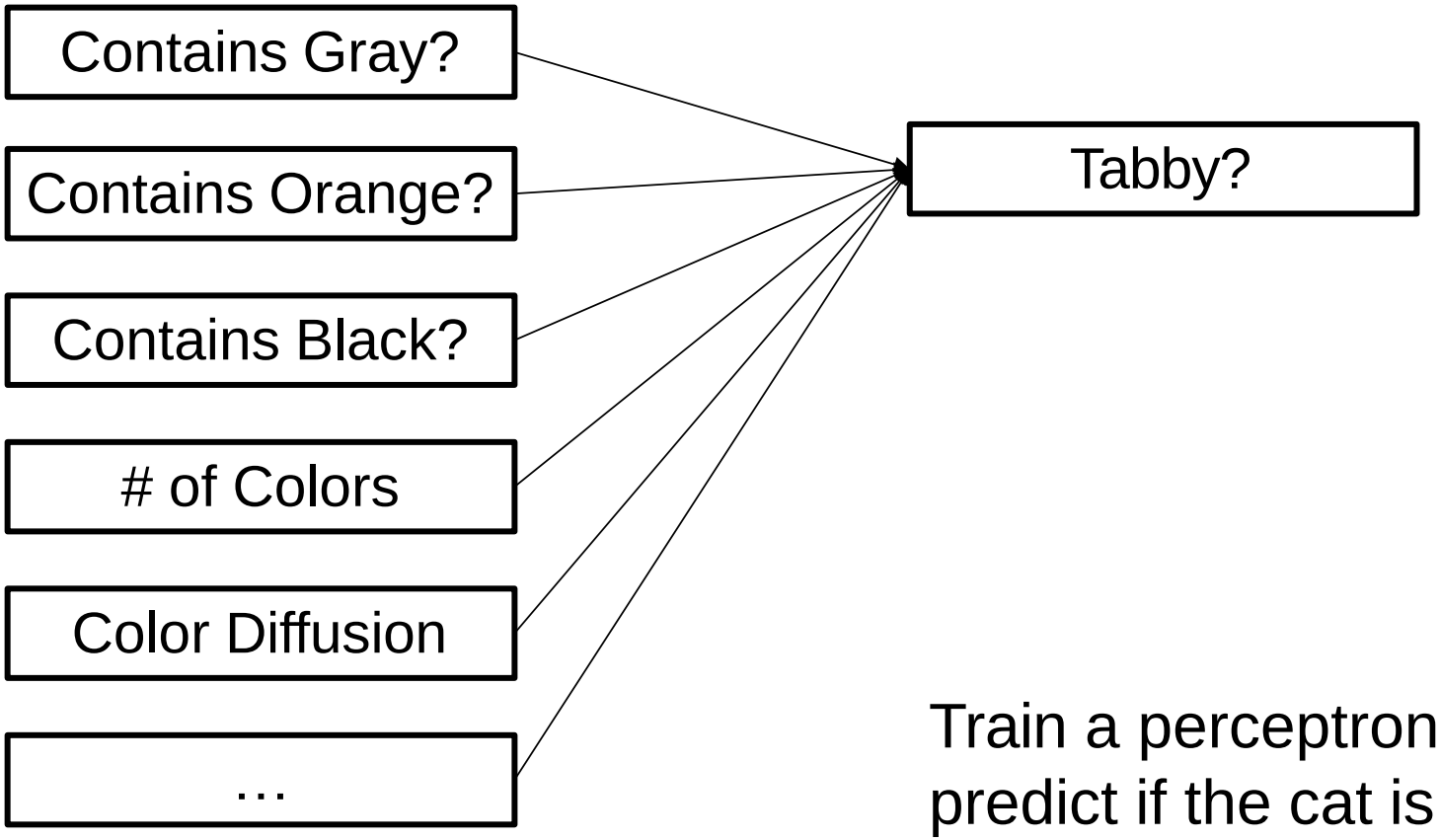
Unlike the AND and OR gate, an XOR gate requires an intermediate hidden layer for preliminary transformation in order to achieve the logic of an XOR gate.
An XOR gate assigns weights so that XOR conditions are met. It cannot be implemented with a single layer Perceptron and requires Multi-layer Perceptron or MLP.
H represents the hidden layer, which allows XOR implementation.
I1, I2, H3, H4, O5are 0 (FALSE) or 1 (TRUE)
t3= threshold for H3; t4= threshold for H4; t5= threshold for O5
H3= sigmoid (I1*w13+ I2*w23–t3); H4= sigmoid (I1*w14+ I2*w24–t4)
O5= sigmoid (H3*w35+ H4*w45–t5);


I_1
I_2
H_3
H_4
O_5
w_{13}
w_{14}
w_{23}
w_{24}
w_{35}
w_{45}

Multilayer Perceptron

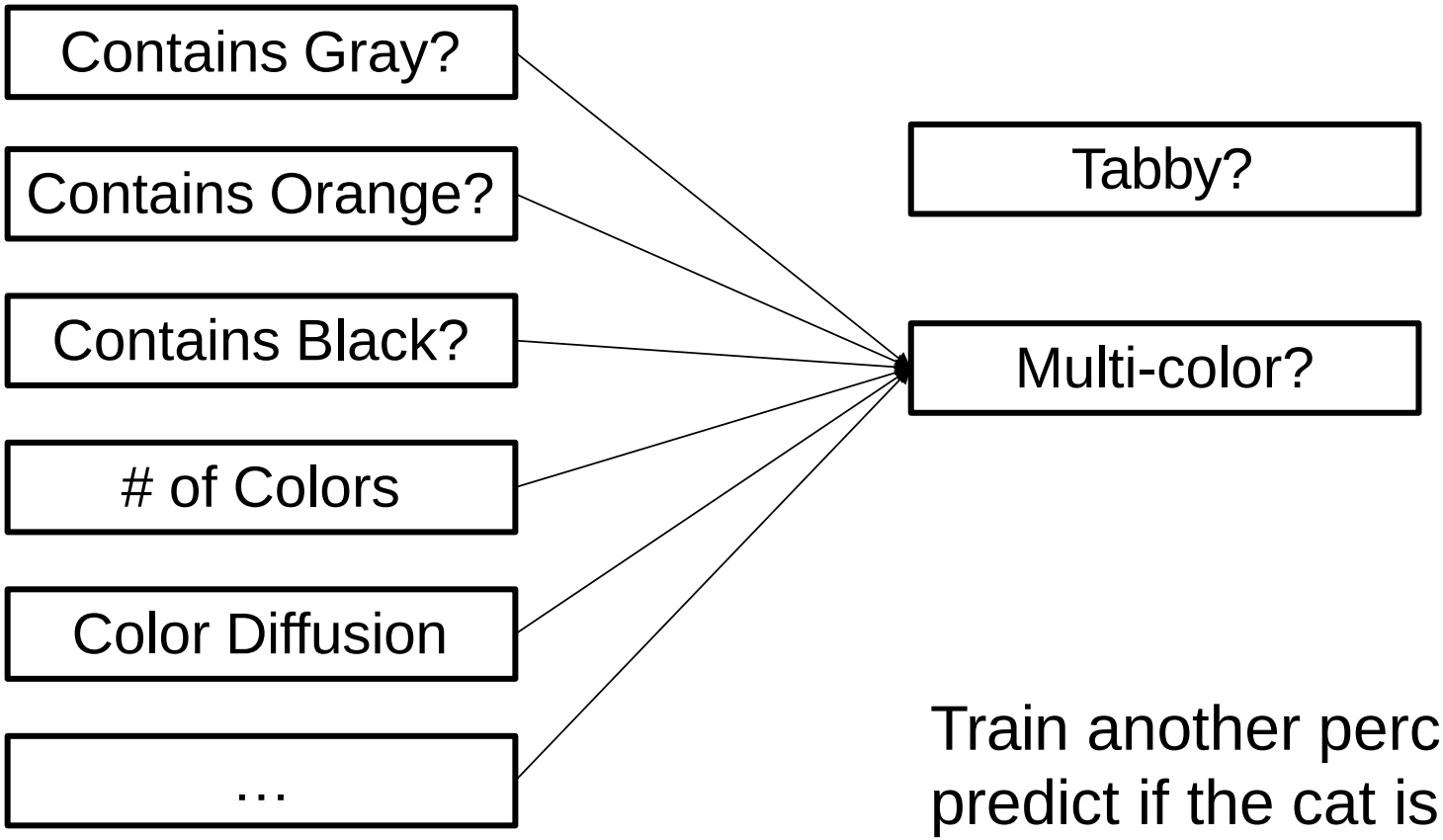
Let’s start with a cartoon about how a multilayer perceptron (MLP) works conceptually.

Multilayer Perceptron



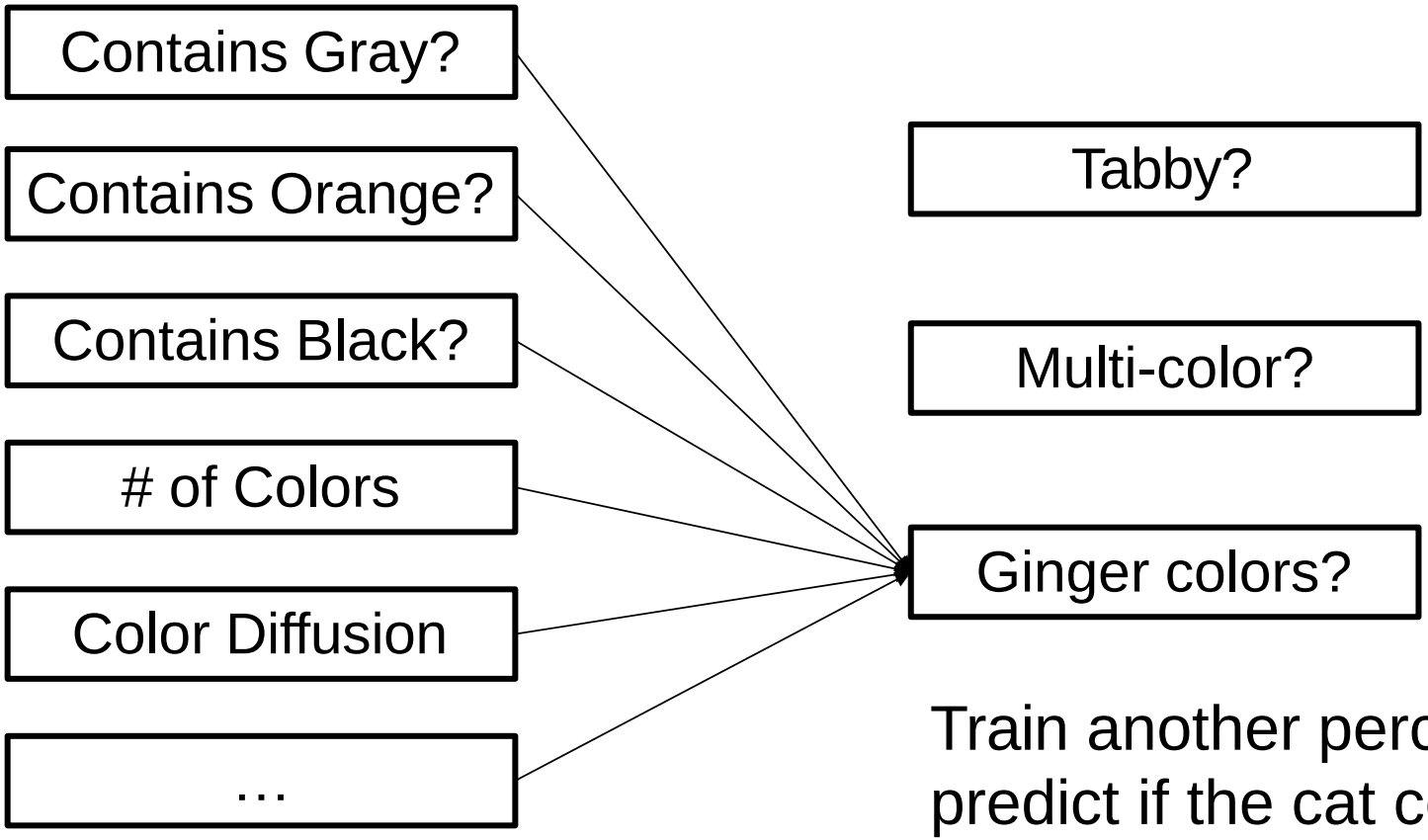
Train a perceptron to predict if the cat is a tabby

Multilayer Perceptron



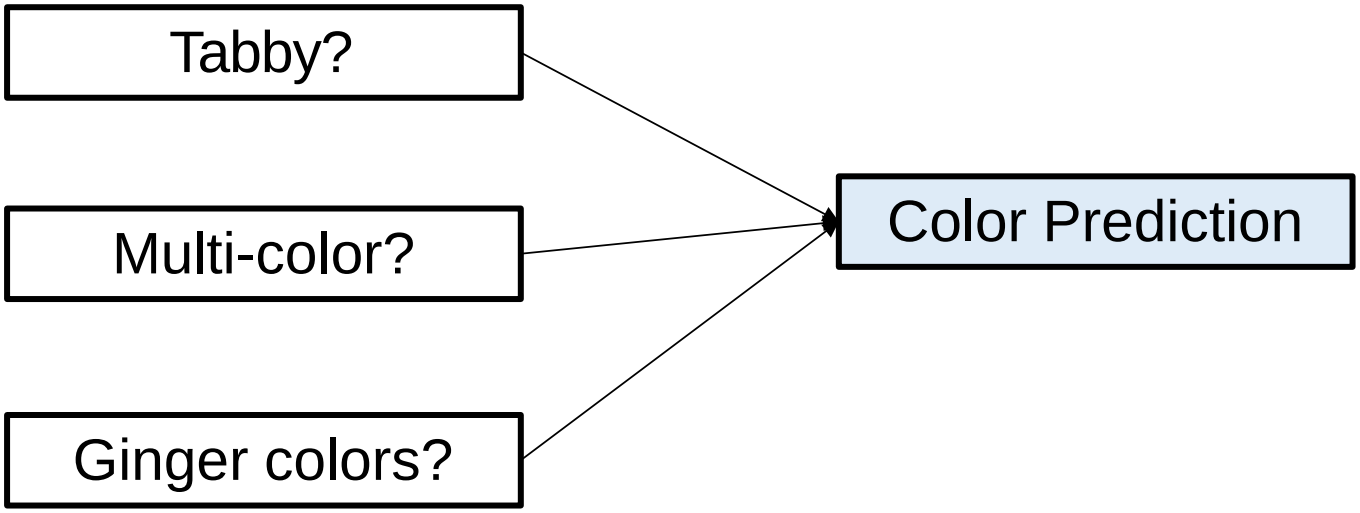
Train another perceptron to predict if the cat is bi-color or tri-color

Multilayer Perceptron



Train another perceptron to predict if the cat contains orange/red/brown colors

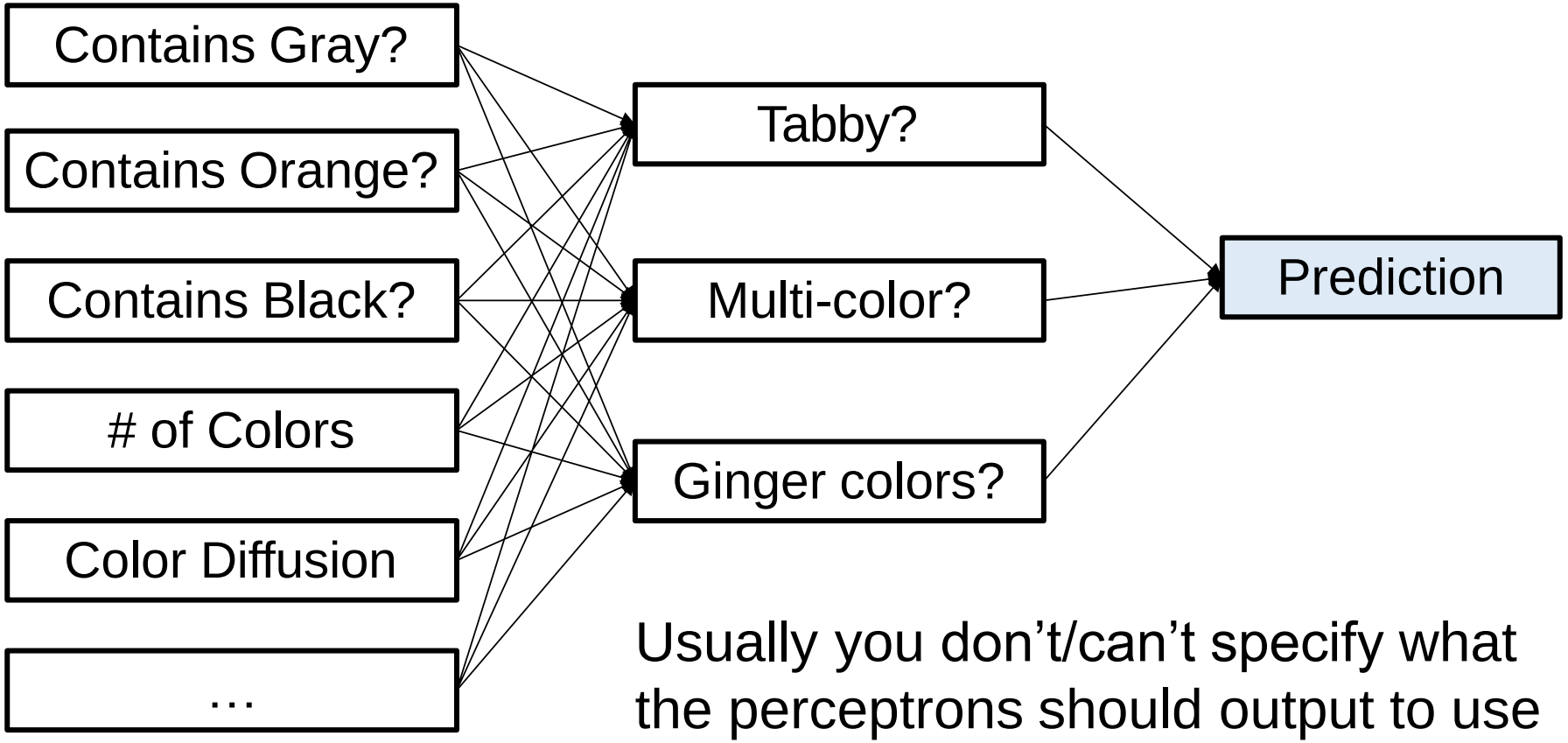
Multilayer Perceptron



Treat the outputs of your perceptrons as new features

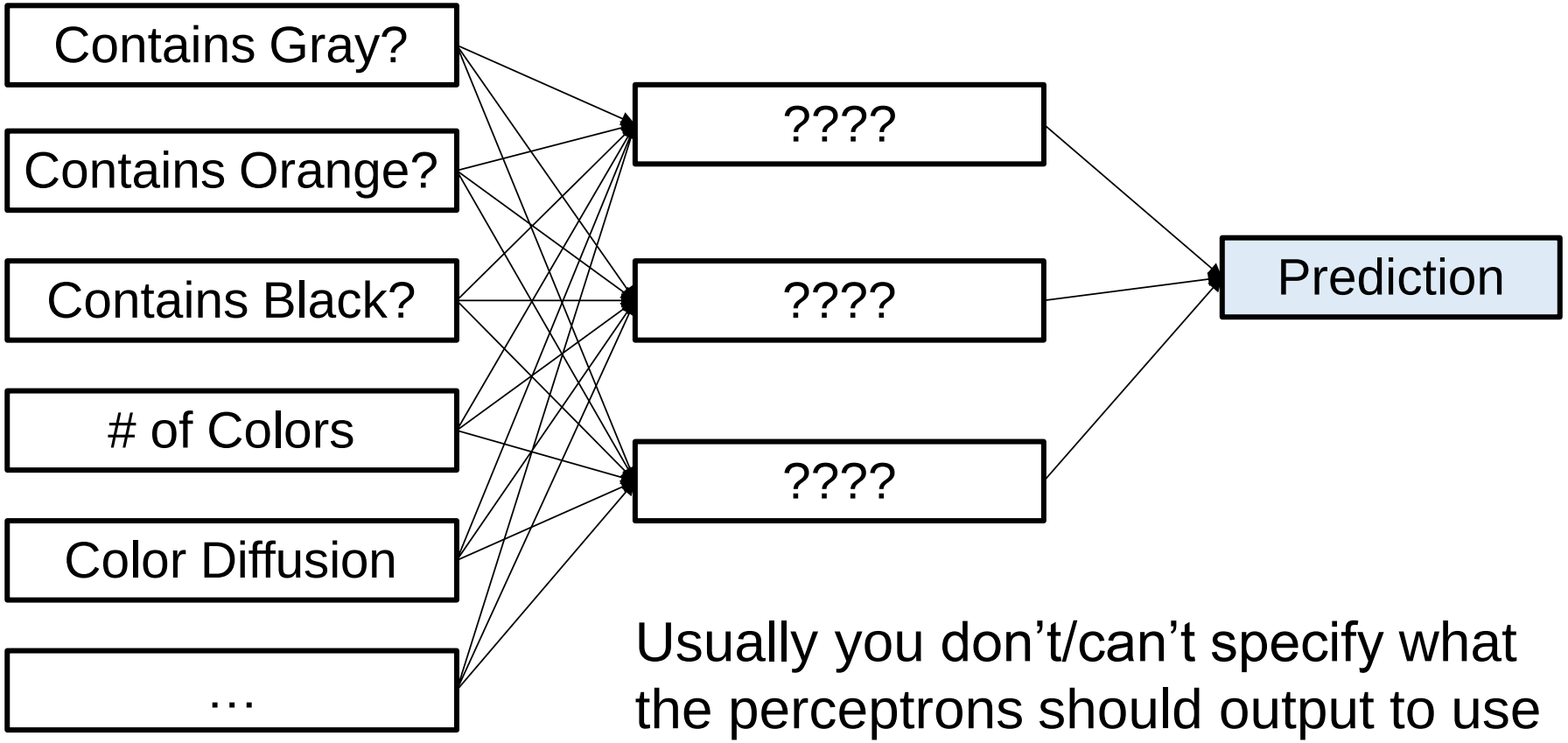
Train another perceptron on these new features

Multilayer Perceptron



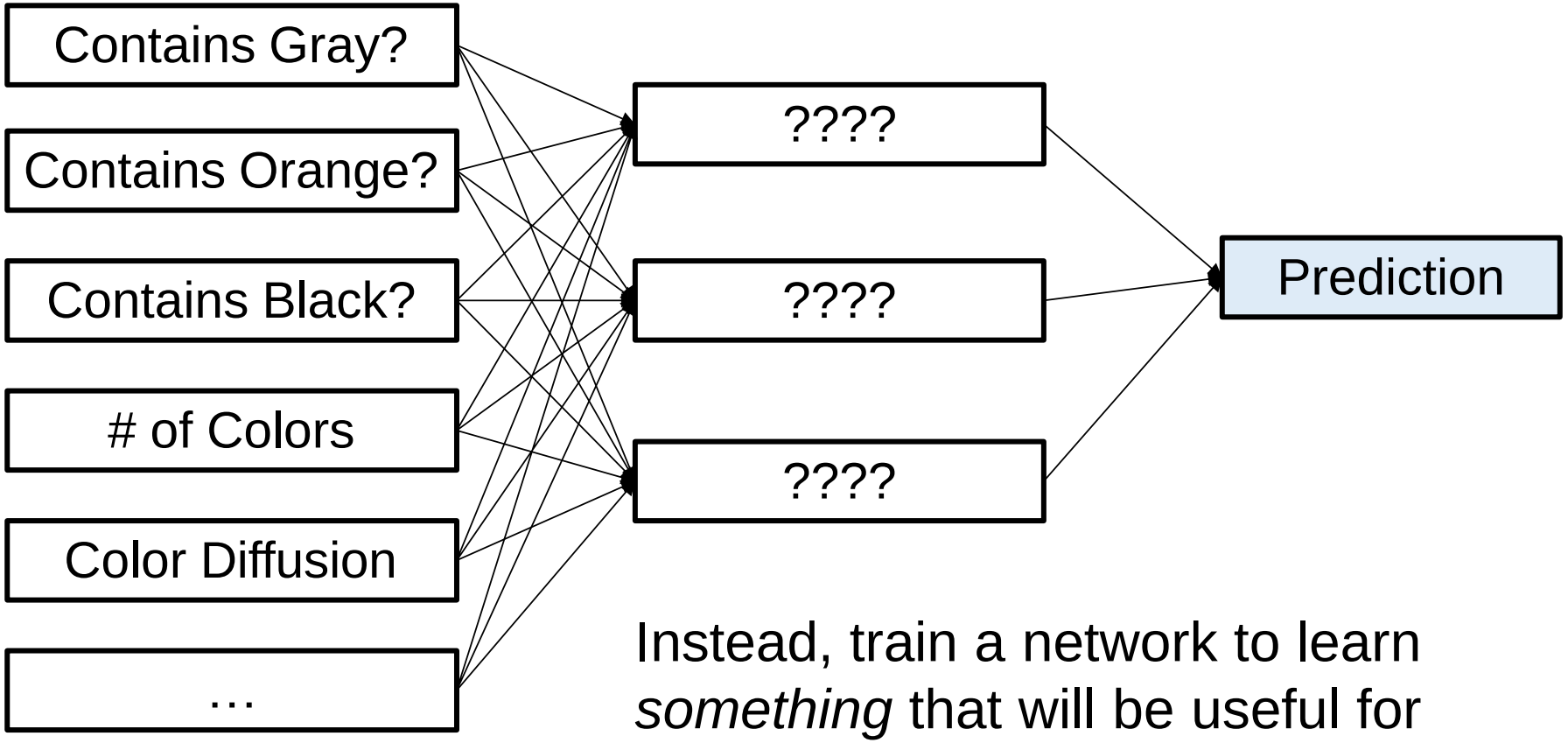
Usually you don’t/can’t specify what the perceptrons should output to use as new features in the next layer.

Multilayer Perceptron



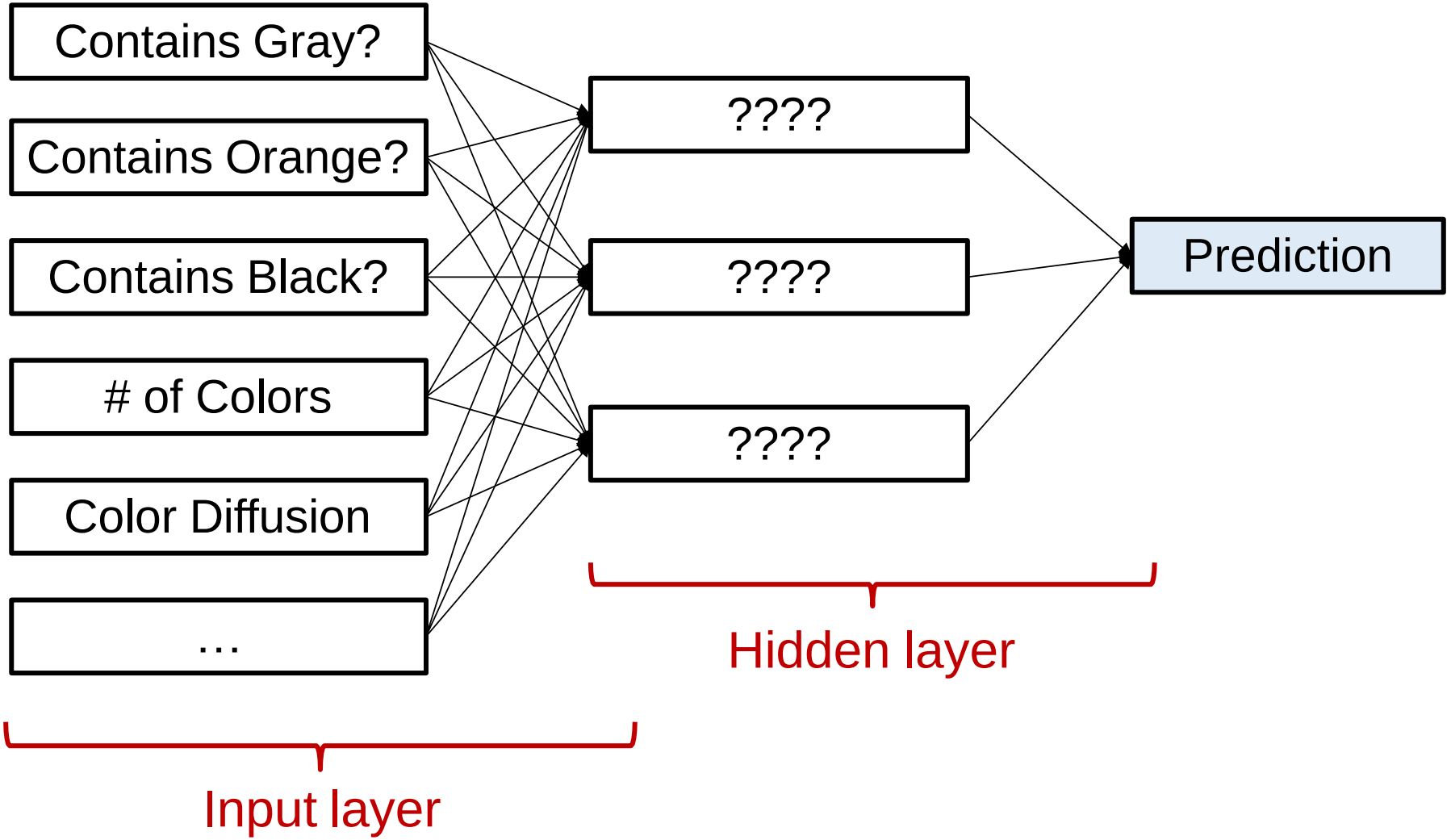
Usually you don’t/can’t specify what the perceptrons should output to use as new features in the next layer.

Multilayer Perceptron



Instead, train a network to learn *something* that will be useful for prediction.

Multilayer Perceptron



Multilayer Perceptron

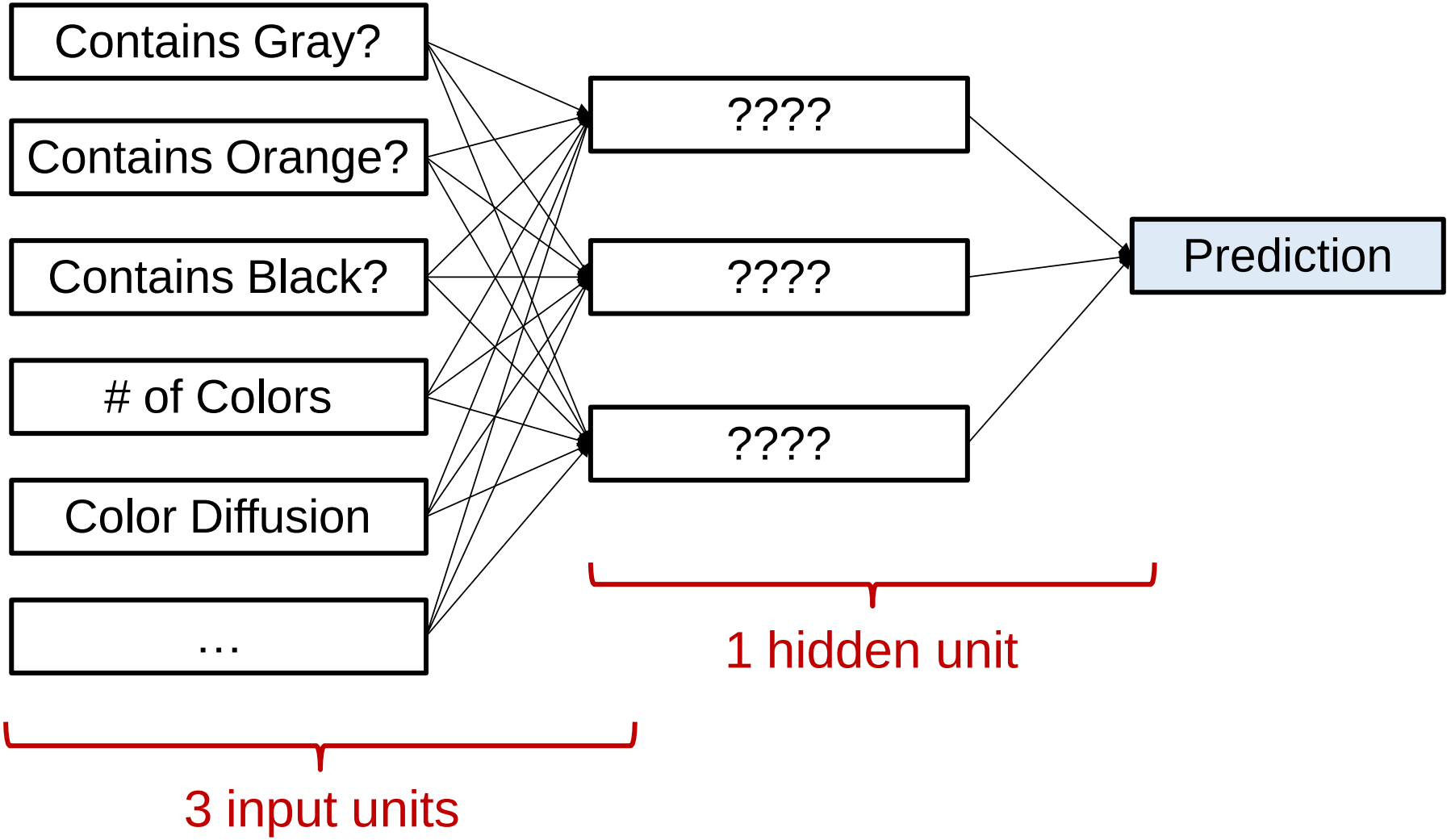
The **input layer** is the first set of perceptrons which output positive/negative based on the observed features in your data.

A **hidden layer** is a set of perceptrons that uses the outputs of the previous layer as inputs, instead of using the original data.

- There can be multiple hidden layers!
- The final hidden layer is also called the **output layer.**

Each perceptron in the network is called a **unit.**

Multilayer Perceptron



Activation Functions

Remember, perceptron defines a *score* $\mathbf{w}^T\mathbf{x}$, then the score is input into an *activation function* which converts the score into an output:

$\phi(\mathbf{w}^T\mathbf{x})$ = 1 if above 0, -1 otherwise

Logistic regression used the logistic function to convert the score into an output between 0 and 1:

$\phi(\mathbf{w}^T\mathbf{x}) = 1 / (1 + \exp(-\mathbf{w}^T\mathbf{x}))$

Activation Functions

Neural networks usually use also use the logistic function (or another sigmoid function) as the activation function

This is true even in multilayer perceptron

- Potentially confusing terminology:
  The “units” in a multilayer perceptron aren’t technically perceptrons!

The reason is that calculating the perceptron threshold is not differentiable, so can’t calculate the gradient for learning.

Multilayer Perceptron

The final classification can be written out in full:

$$\phi\left(w_{211}(\phi(\mathbf{w_{11}}^T\mathbf{x})) + w_{212}(\phi(\mathbf{w_{12}}^T\mathbf{x})) + w_{213}(\phi(\mathbf{w_{13}}^T\mathbf{x}))\right)$$

Equivalent to writing:

$\phi(w_{21}^T\mathbf{y})$, where $\mathbf{y} = <\phi(\mathbf{w_{11}}^T\mathbf{x}), \phi(\mathbf{w_{12}}^T\mathbf{x}), \phi(\mathbf{w_{13}}^T\mathbf{x})>$

(above, the dot product has been expanded out)

Multilayer Perceptron

The final classification can be written out in full:

$$\phi\left(w_{211}(\phi(\mathbf{w_{11}}^T\mathbf{x})) + w_{212}(\phi(\mathbf{w_{12}}^T\mathbf{x})) + w_{213}(\phi(\mathbf{w_{13}}^T\mathbf{x}))\right)$$

Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\phi(w_{211}(\phi(\mathbf{w_{11}}^T\mathbf{x})) + w_{212}(\phi(\mathbf{w_{12}}^T\mathbf{x})) + w_{213}(\phi(\mathbf{w_{13}}^T\mathbf{x})))$$

Outputs of the three “perceptron” units in the first layer (passing the three scores through the activation function)

Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\phi\left(w_{211}\left(\phi\left(\mathbf{w_{11}}^{T}\mathbf{x}\right)\right) + w_{212}\left(\phi\left(\mathbf{w_{12}}^{T}\mathbf{x}\right)\right) + w_{213}\left(\phi\left(\mathbf{w_{13}}^{T}\mathbf{x}\right)\right)\right)$$

Score of the one “perceptron” unit in the second layer
(which uses the three outputs from the last layer as “features”)

Outputs of the three “perceptron” units in the first layer

Scores of the three “perceptron” units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\phi\left(w_{211}\left(\phi\left(\mathbf{w_{11}}^{T}\mathbf{x}\right)\right) + w_{212}\left(\phi\left(\mathbf{w_{12}}^{T}\mathbf{x}\right)\right) + w_{213}\left(\phi\left(\mathbf{w_{13}}^{T}\mathbf{x}\right)\right)\right)$$

Final output (passing the final score through the activation function)

Score of the one "perceptron" unit in the second layer

Outputs of the three "perceptron" units in the first layer

Scores of the three "perceptron" units in the first layer

Multilayer Perceptron

The final classification can be written out in full:

$$\phi\left(w_{211}(\phi(\mathbf{w_{11}}^T\mathbf{x})) + w_{212}(\phi(\mathbf{w_{12}}^T\mathbf{x})) + w_{213}(\phi(\mathbf{w_{13}}^T\mathbf{x}))\right)$$

Can then define a loss function L(**w**) based on the difference between classifications and true labels

- Can minimize with gradient descent (or related methods)
  - Algorithm called **backpropagation** makes gradient calculations more efficient
- Loss function is **non-convex**
  - Lots of local minima make it hard to find good solution

Multilayer Perceptron

How many hidden layers should there be?

How many units should there be in each layer?

You have to specify these when you run the algorithm.

- You can think of these as yet more *hyperparameters* to tune.

Nonlinear Example: XOR

Consider two binary features x_1 and x_2 and you want to learn to output the function, x_1 XOR x_2.

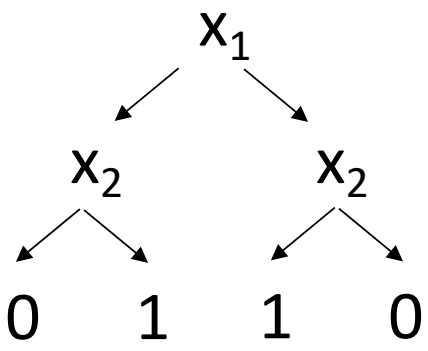
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

A linear classifier would not be able to learn this, but decision trees and neural networks can.

Nonlinear Example: XOR

A decision tree can simply learn that:

- if x_1=0, output 0 if x_2=0 and output 1 if x_2=1
- if x_1=1, output 1 if x_2=0 and output 0 if x_2=1



Nonlinear Example: XOR

We can learn this with a MLP with 2 input units and 1 hidden unit.

Input units:

w_{11} = <0.6, 0.6>, b_{11} = -1.0

w_{12} = <1.1, 1.1>, b_{12} = -1.0

- Unit 1 will only output 1 if both x_1=1 and x_2=1
- Unit 2 will only output 1 if either x_1=1 or x_2=1

Nonlinear Example: XOR

Input units:

- Unit 1 will only output 1 if both x_1=1 and x_2=1
- Unit 2 will only output 1 if either x_1=1 or x_2=1

Hidden unit:

w_2 = <-2.0, 1.1>, b_2 = -1.0

- If feature 1 (Unit 1 output) is 1, this will output 0
- If both features (Units 1 and 2) are 0, this will output 0
- If only feature 2 (Unit 2 output) is 1, this will output 1