Lecture 18-1

Linear classifiers and neural networks

1950s Age of the Perceptron

1957 The Perceptron (Rosenblatt)

1969 Perceptrons (Minsky, Papert)

1980s Age of the Neural Network

1986 Back propagation (Hinton)

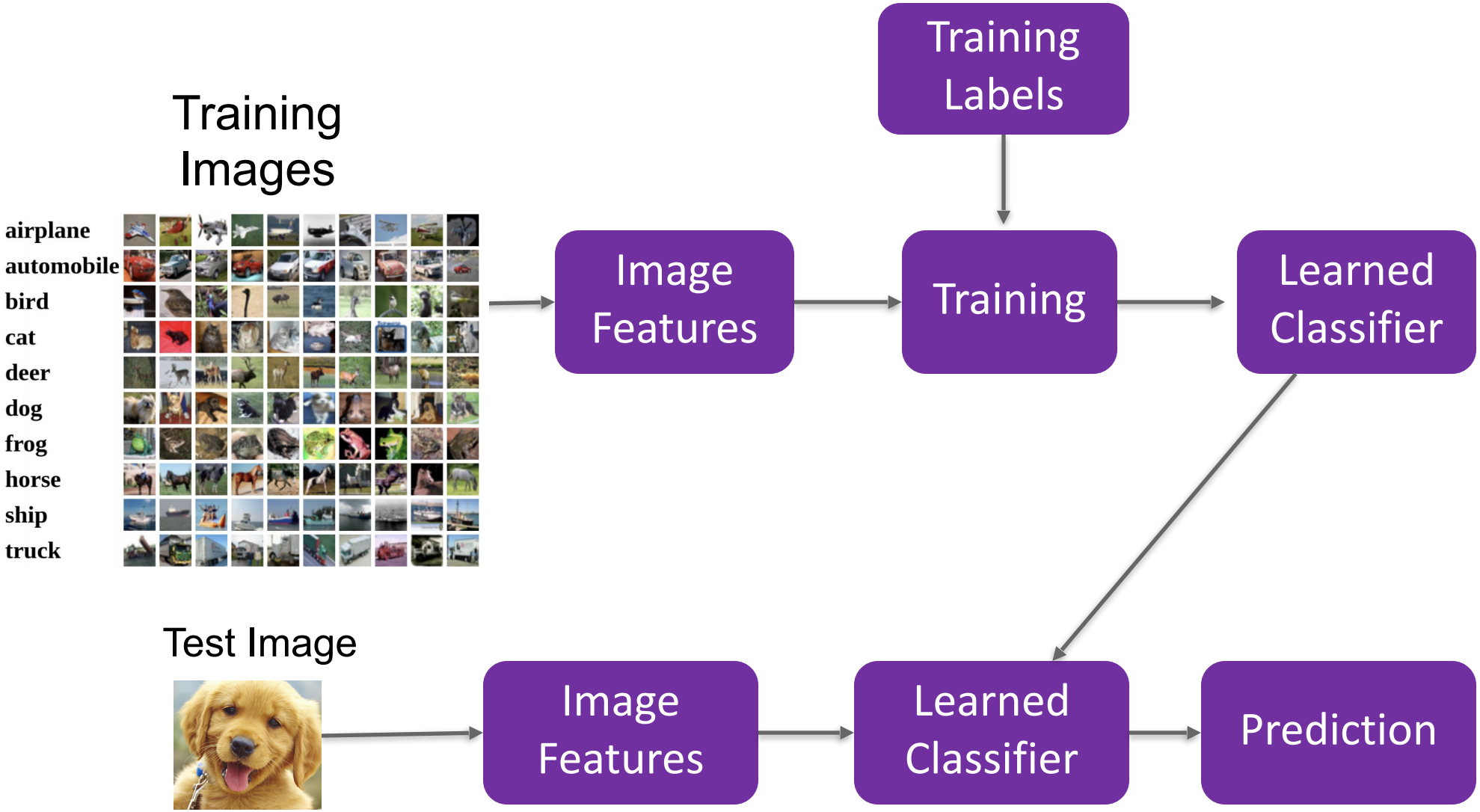
1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

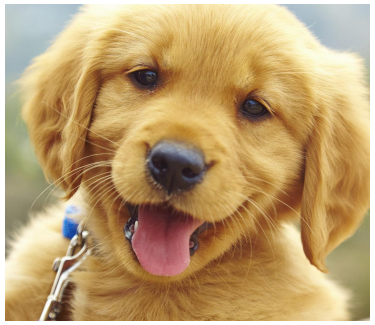
2010s Age of the Deep Network

deep learning = known algorithms + computing power + big data

Let's revisit our simple recognition pipeline to explain where perceptrons fit in



Recall: we can featurize images into a vector



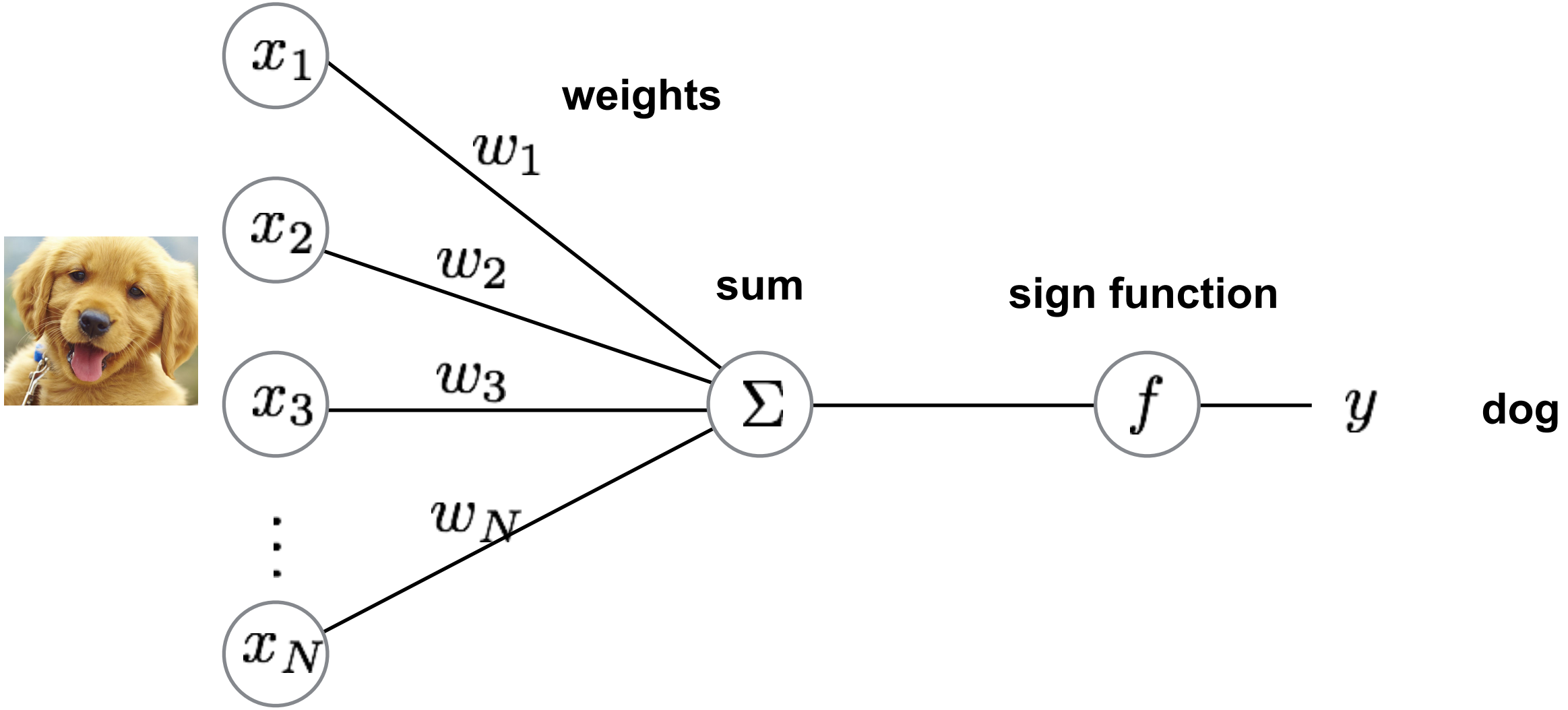
Raw pixels
Raw pixels + (x,y)
PCA
BoW
BoW + spatial pyramids

Image Vector

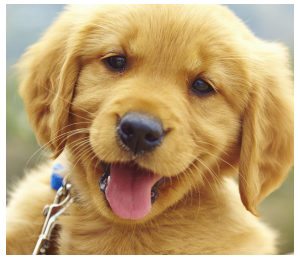
x_1
x_2
x_3
...
...
...
...
...
...
...
...
...
...

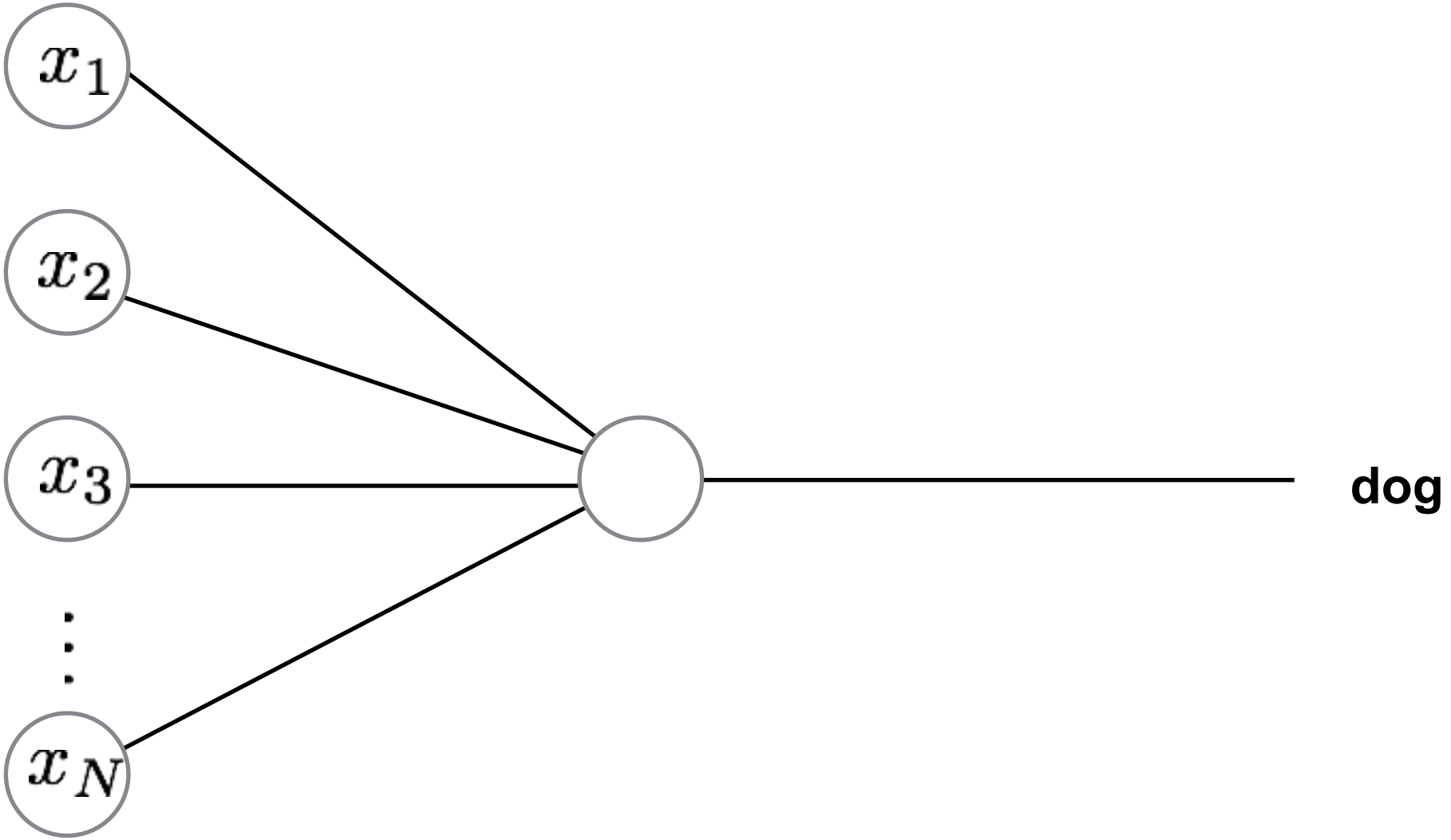
x_n

Perceptrons are a simple transformation that converts feature vectors into recognition scores

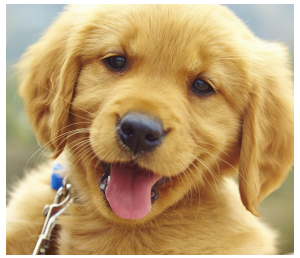


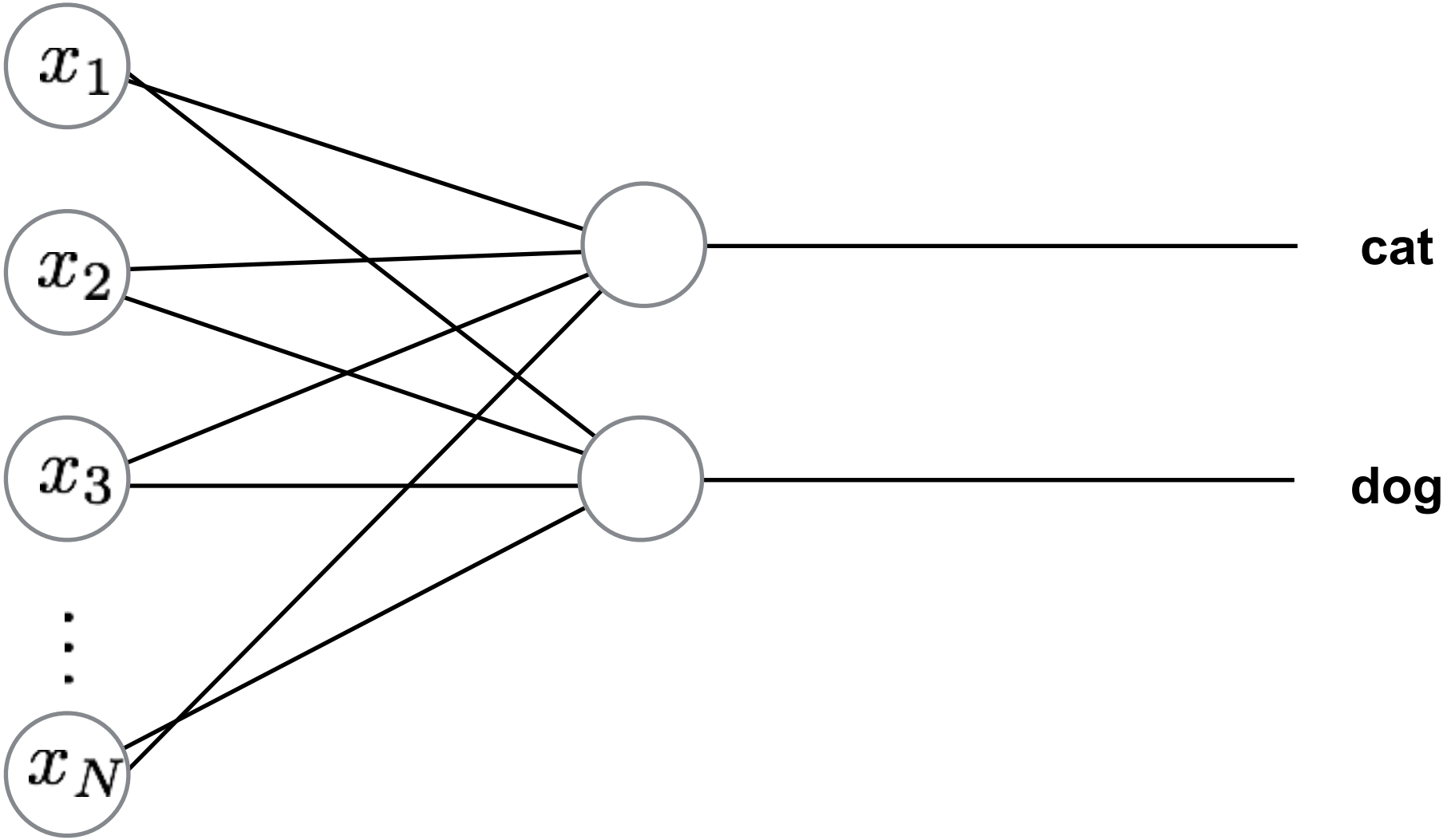
Perceptron: simplified view with one perceptron (produces 1 score for one category)

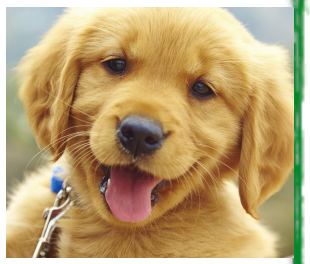




Perceptron: simplified view with two perceptrons (produces 2 scores with 2 categories)







x_1

x_2

x_3

x_N

cat

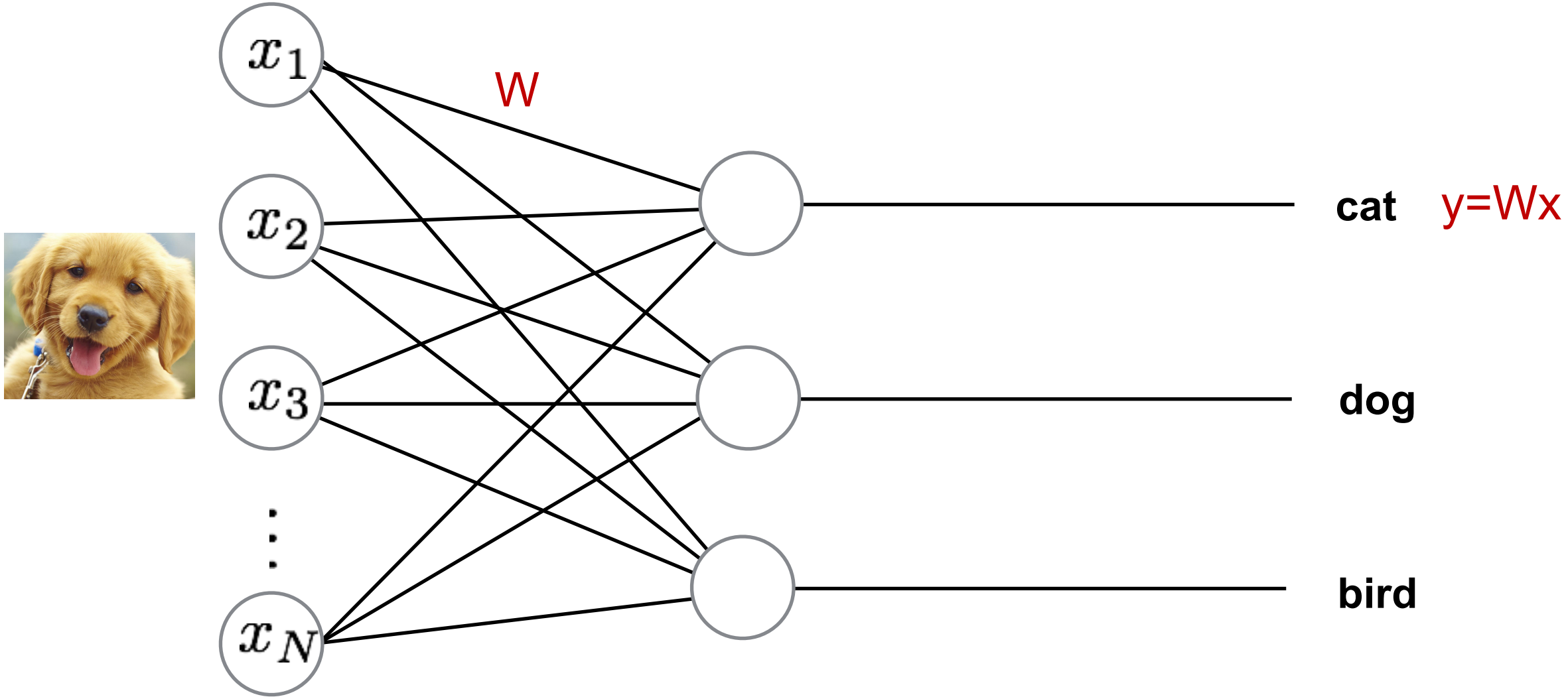
dog

bird

‘input’ layer

‘output’ layer

Linear classifier is a set of perceptrons produces one score for every category



Linear classifier: mathematical formulation with RGB features

(32x32x3)
3072 dimensional vector

x_1

x_2

x_3

x_N

$f(x, W)$

f(*x*, *W*) = *Wx*

x = 3072 x 1

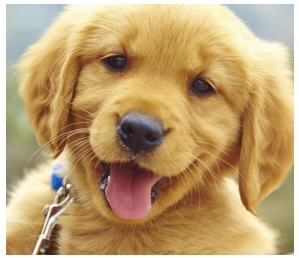
W = ?

Q. What is the shape of W?

W
weights or parameters

10 numbers giving class scores

Linear classifier: mathematical formulation with RGB features



(32x32x3)
3072 dimensional vector

x_1

x_2

x_3

x_N

f(*x*, *W*) = *W*x

x = 3072 x 1

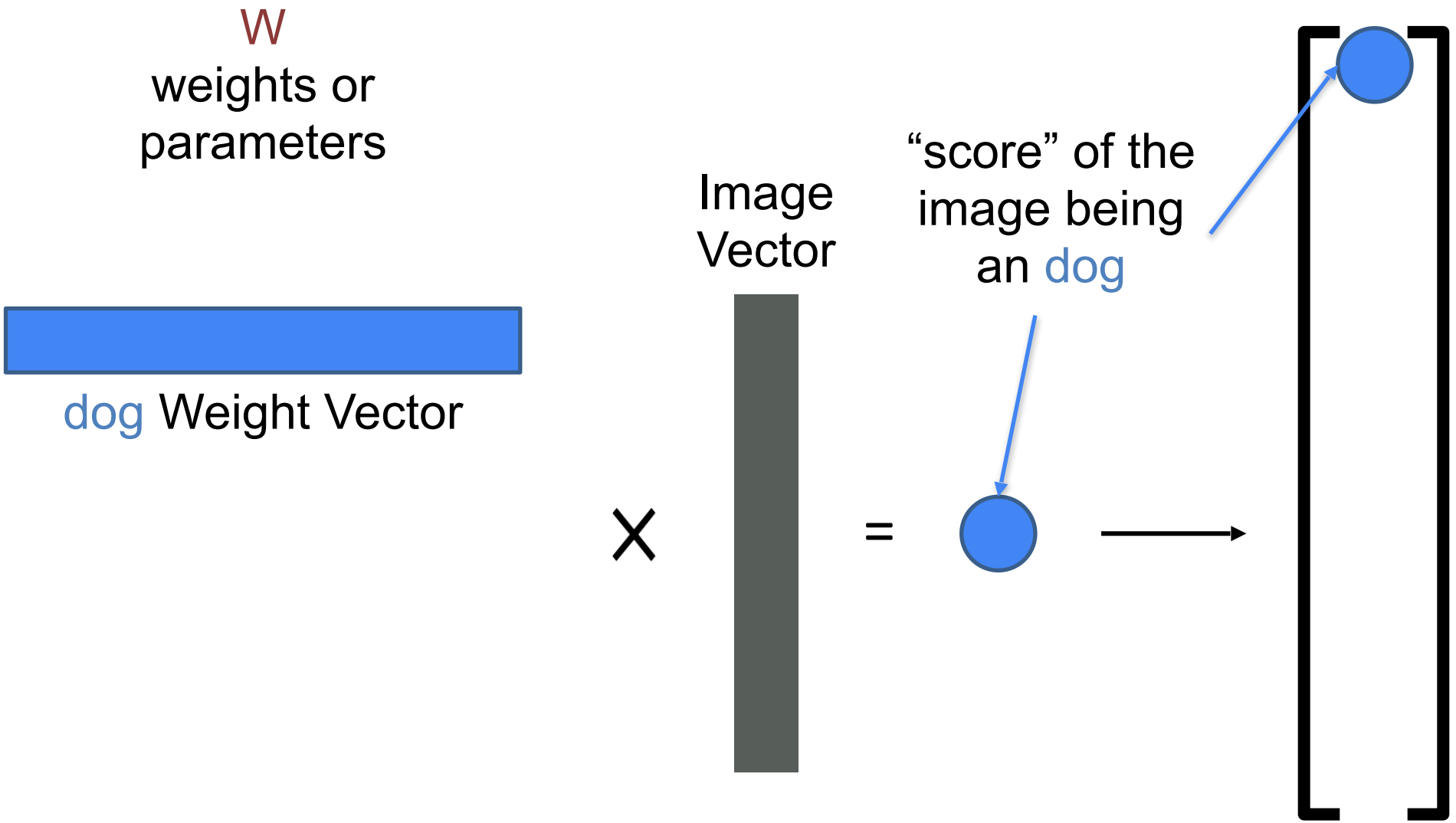
W = 10 x 3072

$f(x, W)$

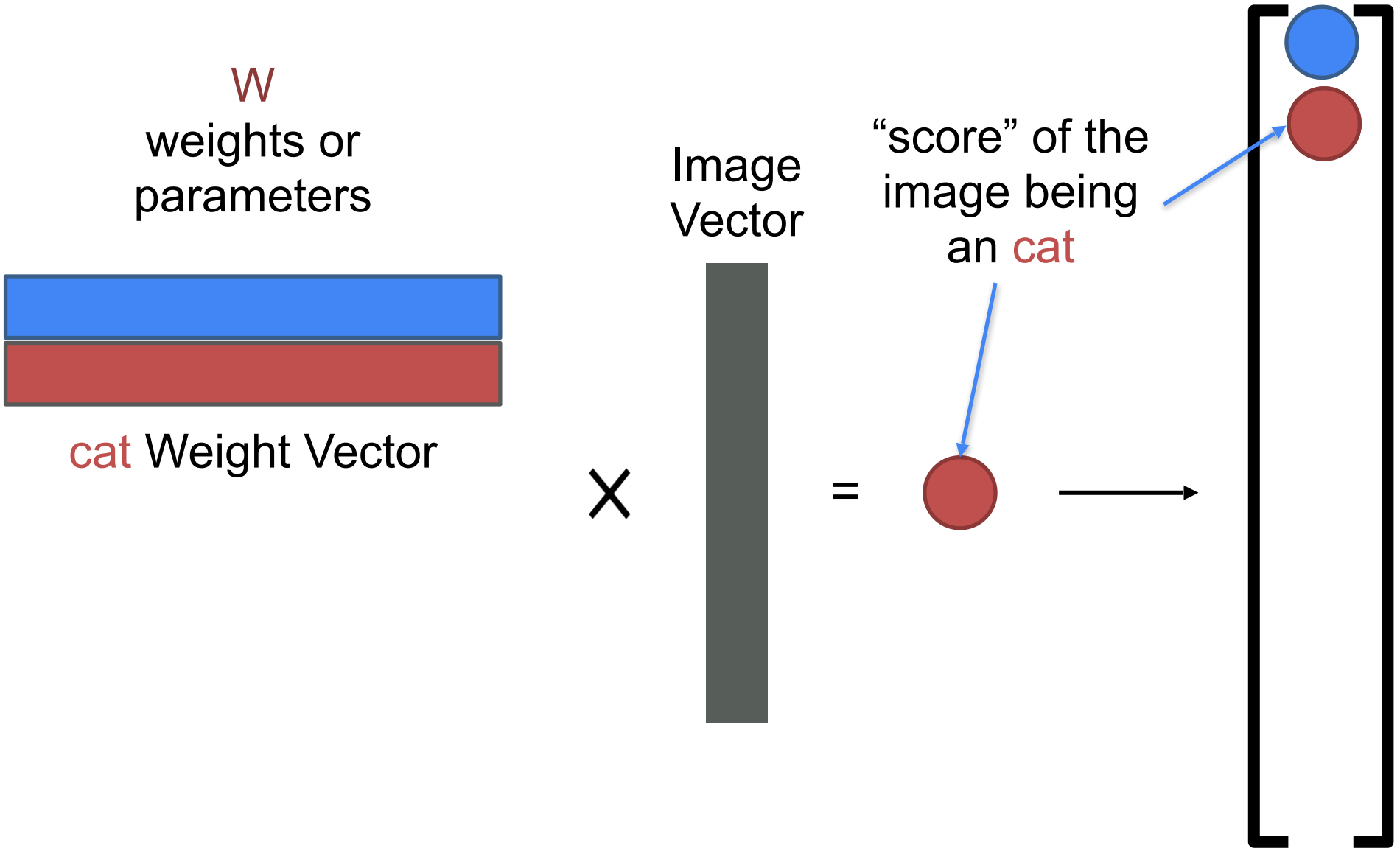
W
weights or parameters

10 numbers giving class scores

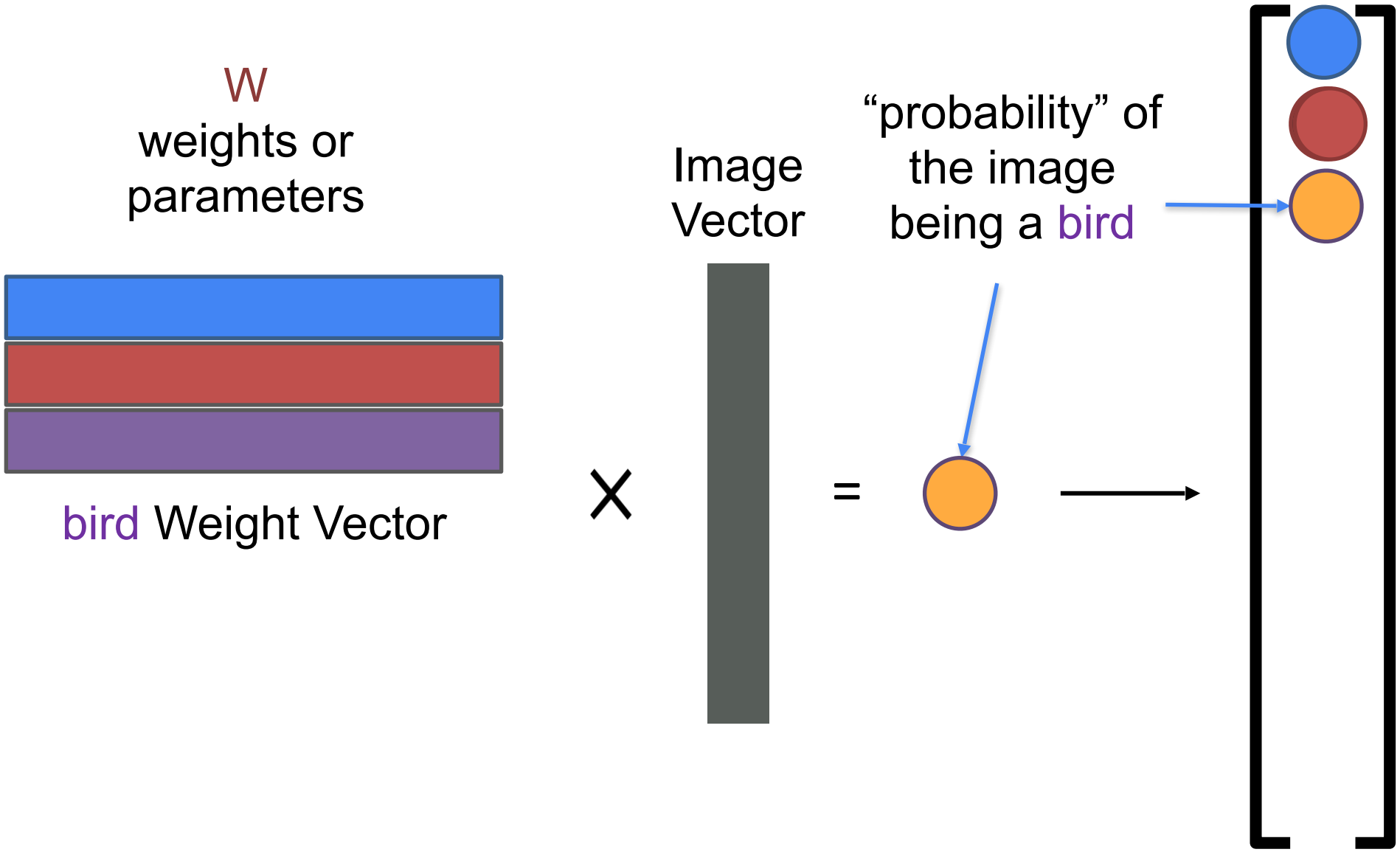
Linear classifier: function visualized



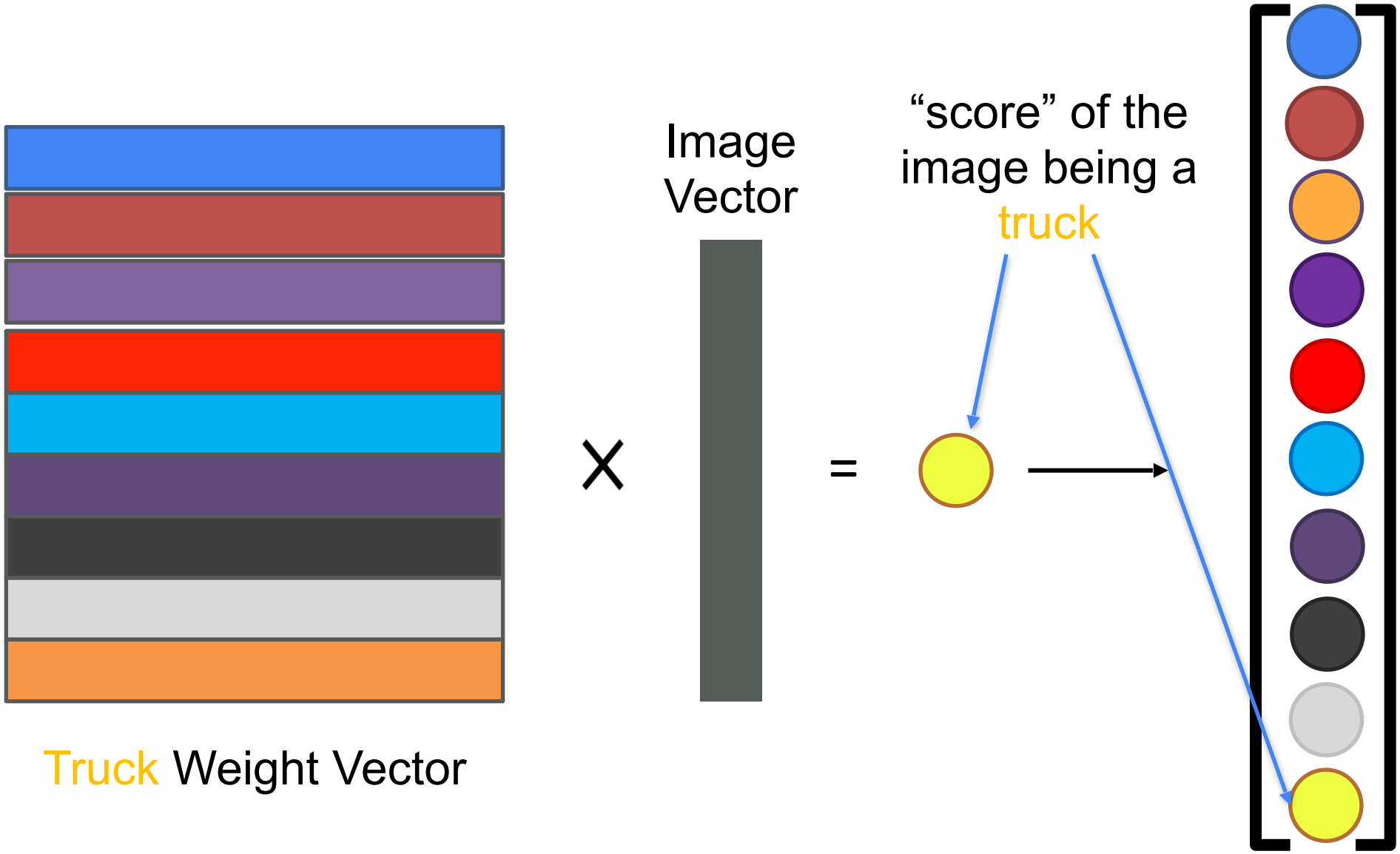
Linear classifier: function visualized



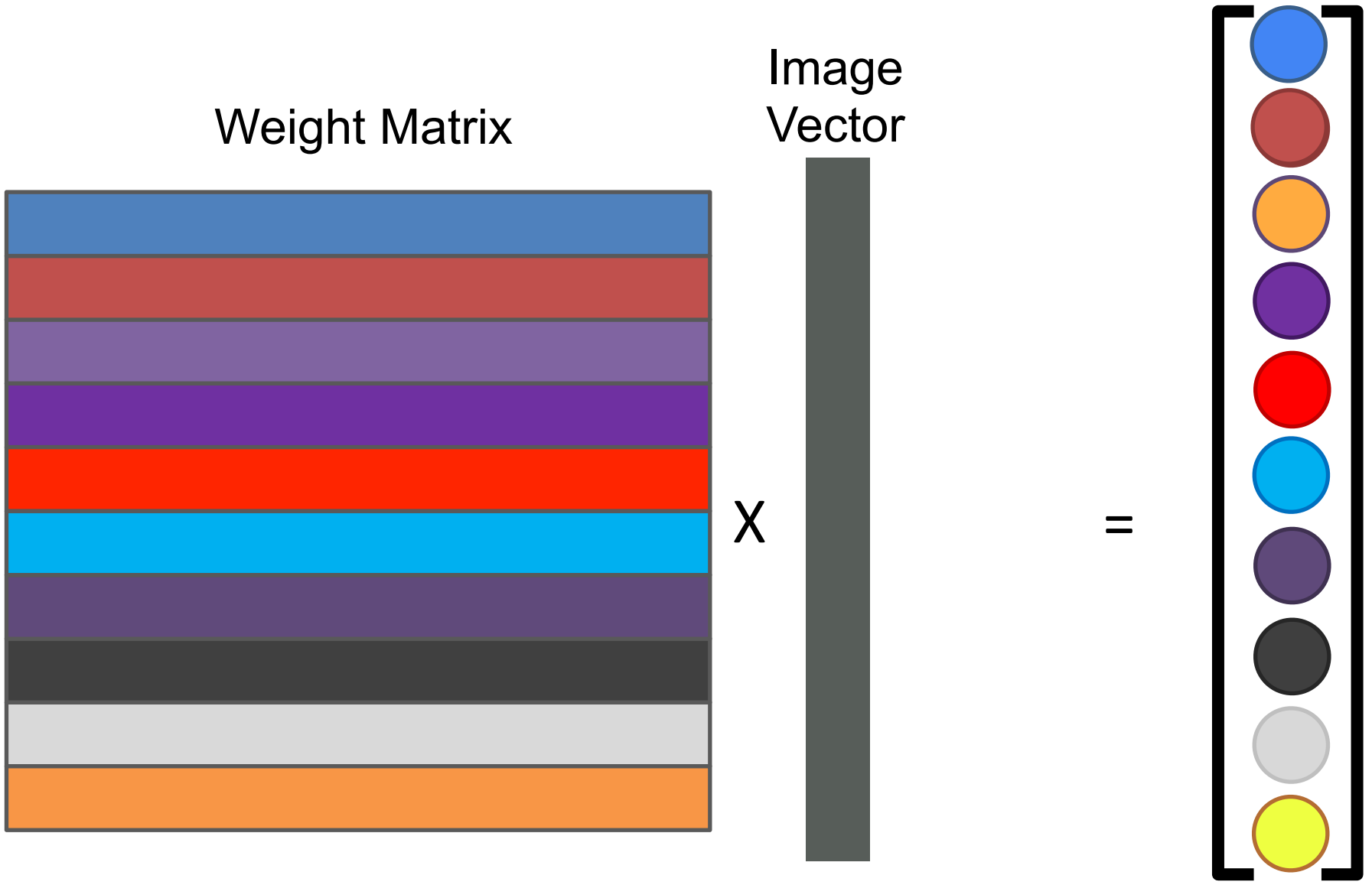
Linear classifier: function visualized



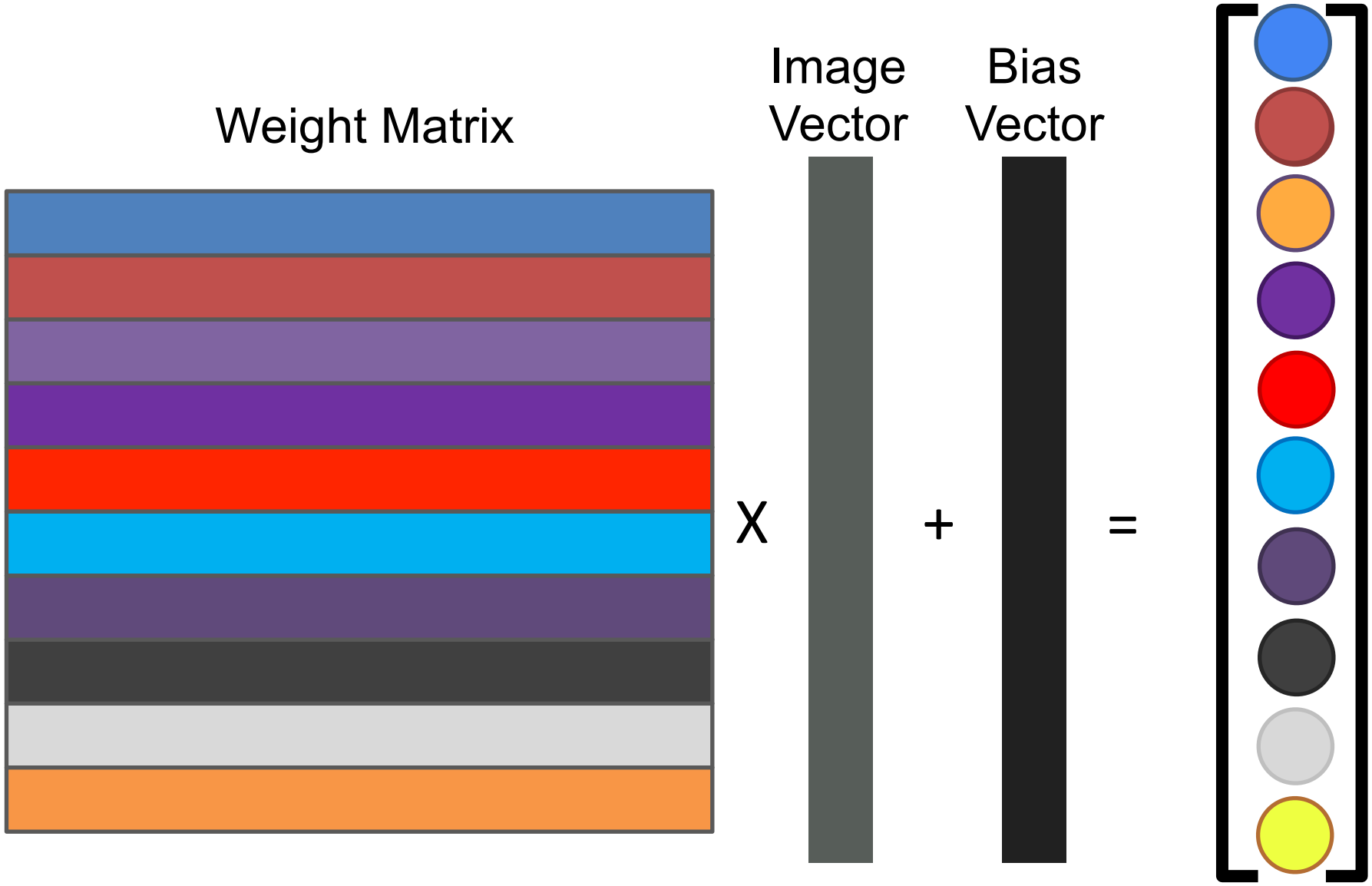
Linear classifier: function visualized



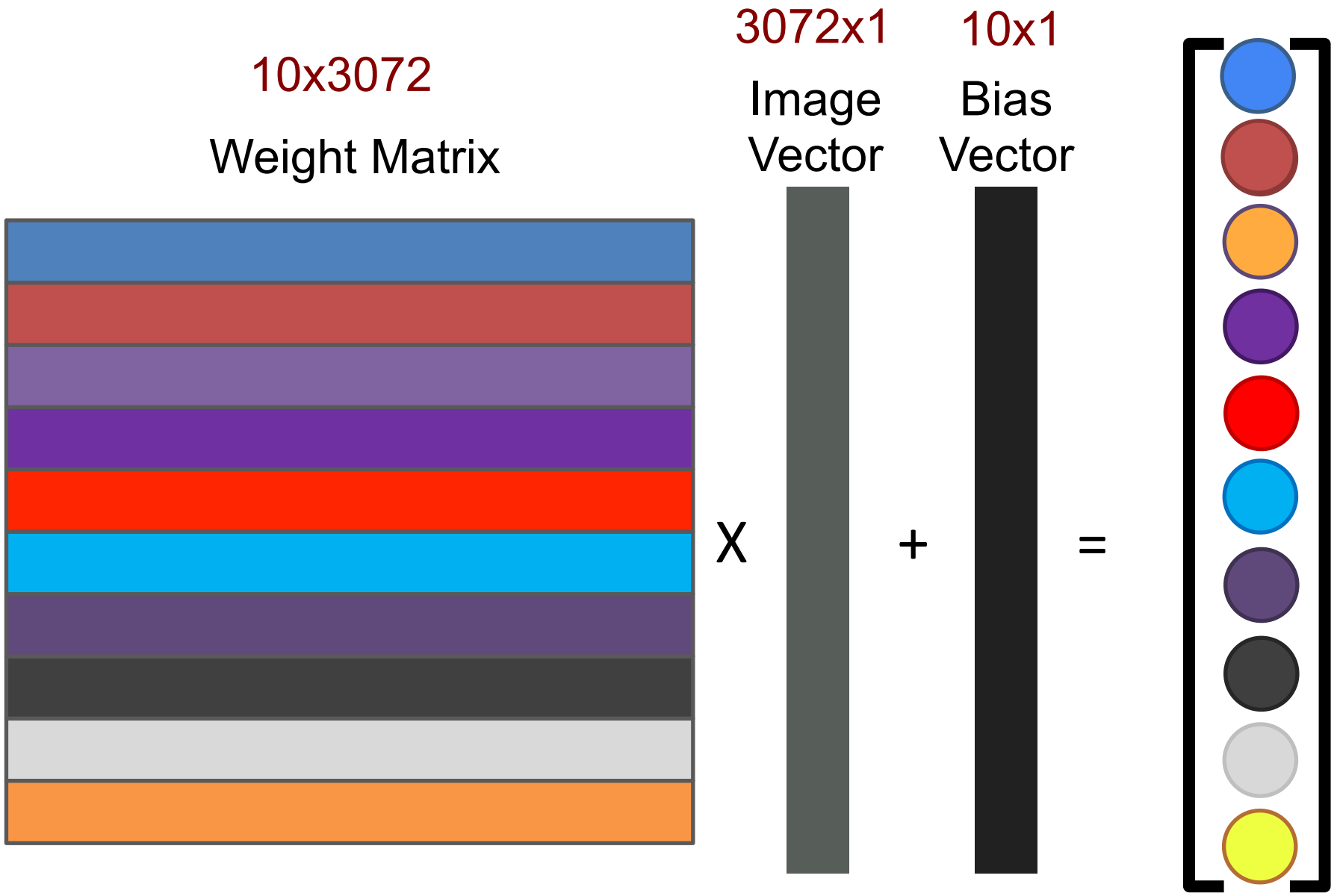
Linear classifier: function visualized



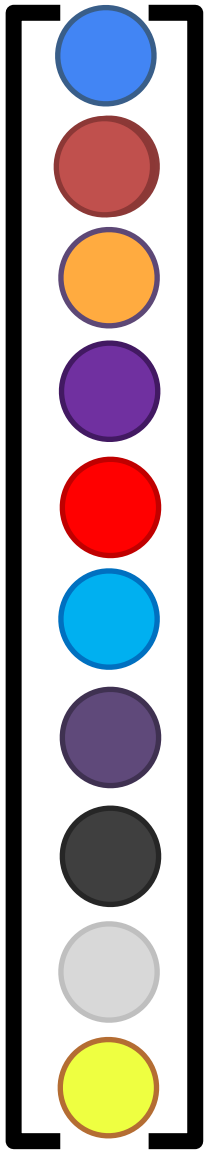
Linear classifier: bias vector



Linear classifier: size



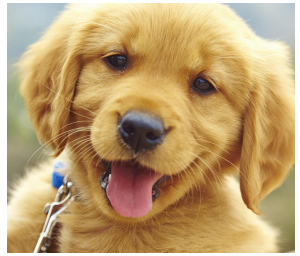
Linear classifier: Making a classification



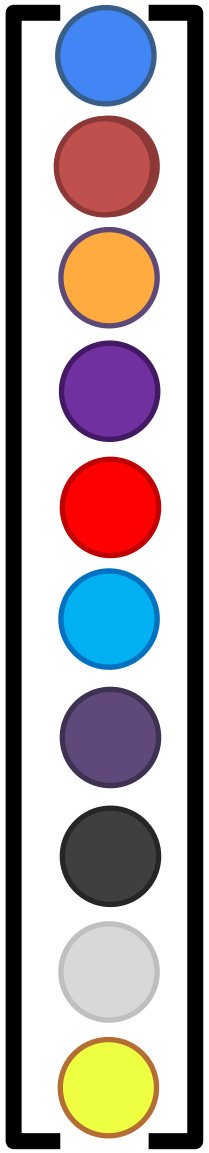
23.4
9.4
-10.4
2.2
1
2.7
-1.3
-1.2
9.9
-8.2

$argmax$

dog



Interpret the Output as Probability



23.4	0.48
9.4	0.18
-10.4	≈0
2.2	≈0
1	≈0
2.7	≈0
-1.3	≈0
-1.2	≈0
9.9	0.29
-8.2	0.05

- Every output is normalized into [0, 1] and sum to be 1
- Softmax Transformation:

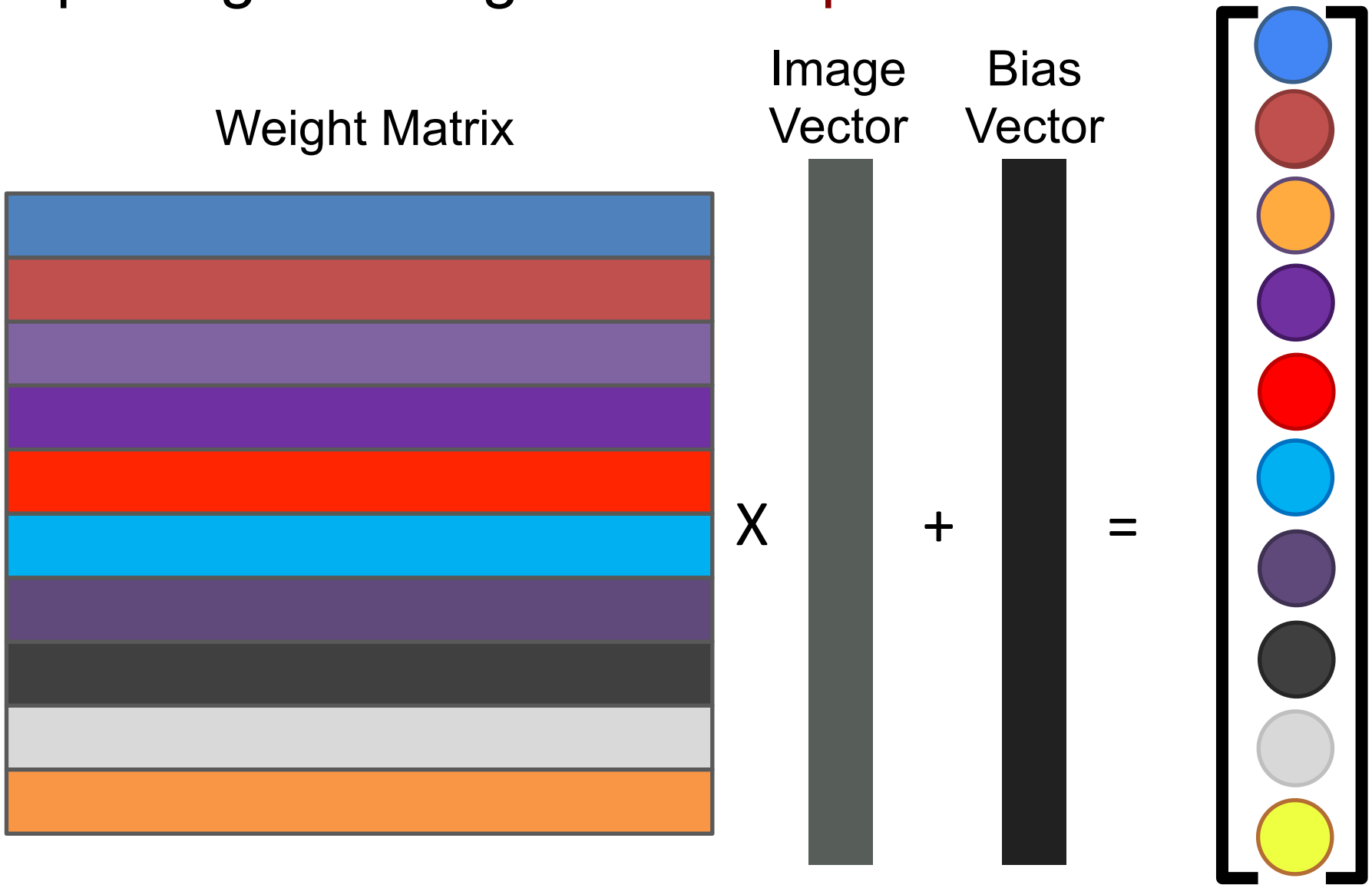
$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

- There are other transformations (e.g. Sparsemax)

Interpreting the weights

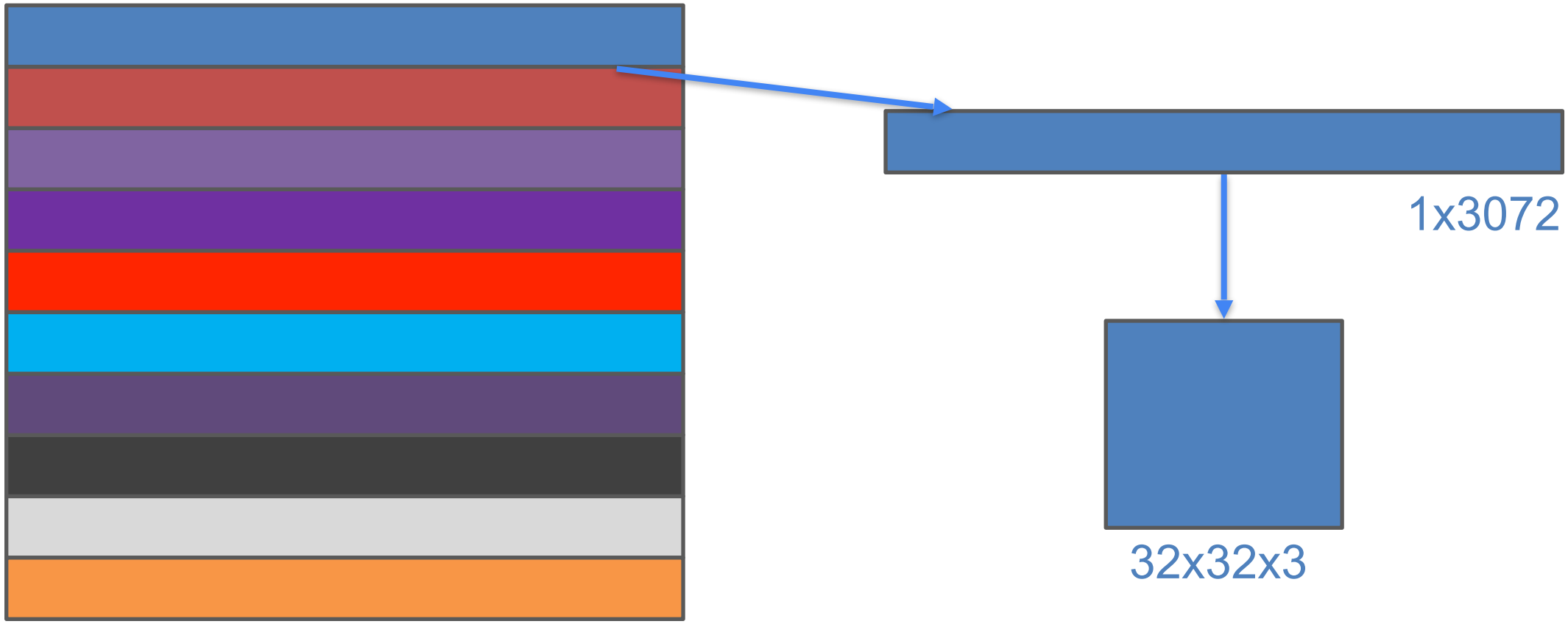
- Assume our weights are trained on the CIFAR 10 dataset with raw pixels:

Interpreting the weights as templates



Interpreting the weights as templates

We can reshape the vector back in to the shape of an image



Let's visualize what the templates look like

We can reshape the row back to the shape of an image

Today's agenda

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

Training linear classifiers

We need to learn how to pick the weights in the first place.

Formally, we need to find *W* such that

$$\min_{W} \text{Loss}(y, \hat{y})$$

Where y is the true label, $\hat{y}$ is the model's predicted label.

All we have to do is define a loss function!

- When the classifier predicts correctly, the loss should be low
- When the classifier makes mistakes, the loss should be high

Properties of a loss function

Given several training examples: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$

and a perceptron: $\hat{y} = wx$

where x is image and y is (integer) label (0 for dog, 1 for cat, etc)
Loss over the entire dataset is an average of loss over examples

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(y_i, \hat{y}_i)$$

How do we choose the loss function L_i ?

YOU get to chose the loss function!

(some are better than others depending on what you want to do)

Softmax Classifier (Multinomial Logistic Regression)

- Recall: we can treat the outputs of a model as probabilities for each class
- common way of measuring distance between probability distributions is Kullback-Leibler (KL) divergence:

$$D_{KL} = \sum_{y} P(y) \log \frac{P(y)}{Q(y)}$$

- where *P* is the ground truth distribution and *Q* is the model's output score distribution

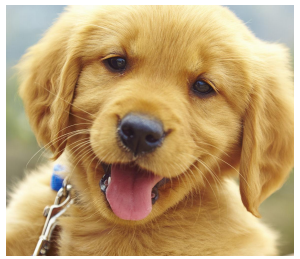
Softmax Classifier (Multinomial Logistic Regression)

KL divergence:

$$D_{KL} = \sum_{y} P(y) \log \frac{P(y)}{Q(y)}$$

In our case, *P* is only non-zero for correct class

For example, consider the case we only have 3 classes:



$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

dog
cat
bird

correct outputs

Softmax Classifier (Multinomial Logistic Regression)

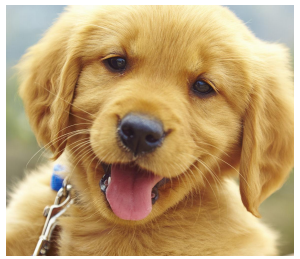
KL divergence:

$$D_{KL} = \sum_y P(y) \, log \frac{P(y)}{Q(y)}$$

$$= -\log Q(y) \text{ when } y = dog$$

$$= -\log Prob[f(x_i, W) = y_i]$$

(It's also called Cross Entropy)



$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

dog
cat
bird

correct outputs

Softmax Classifier (Multinomial Logistic Regression)

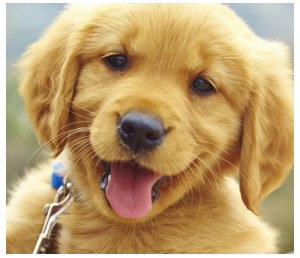
$$L_i = -\log Prob[f(x_i, W) == y_i]$$

We need a mechanism to convert or normalize the output into probability range [0, 1]

Recall:

SOFTMAX

$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$



$$\begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix}$$

model outputs

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

dog
cat
bird

correct outputs

Softmax Classifier (Multinomial Logistic Regression)

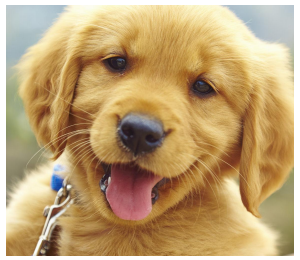
$$L_i = -\log Prob[f(x_i, W) == y_i]$$

We need a mechanism to convert or normalize the output into probability range [0, 1]

Recall:

SOFTMAX

$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$



$$\begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix} \xrightarrow{\exp} \begin{bmatrix} 24.5 \\ 164 \\ 0.18 \end{bmatrix}$$

model outputs

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

dog
cat
bird

loss

correct outputs

Softmax Classifier (Multinomial Logistic Regression)

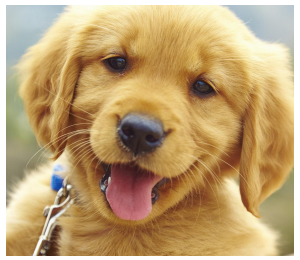
$$L_i = -\log Prob[f(x_i, W) == y_i]$$

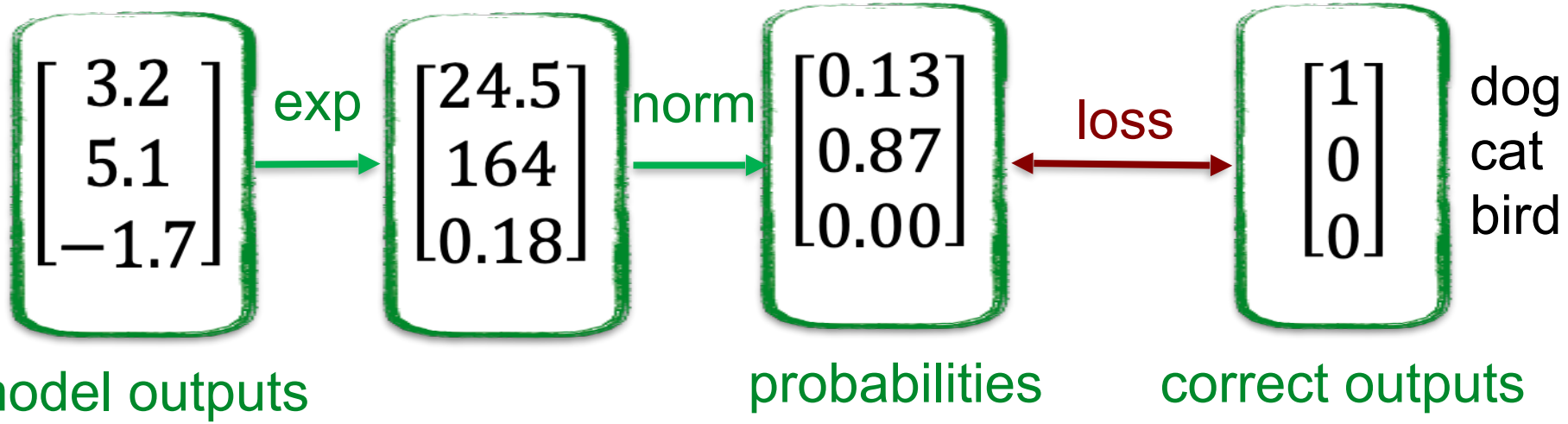
We need a mechanism to convert or normalize the output into probability range [0, 1]

Recall:

SOFTMAX

$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$



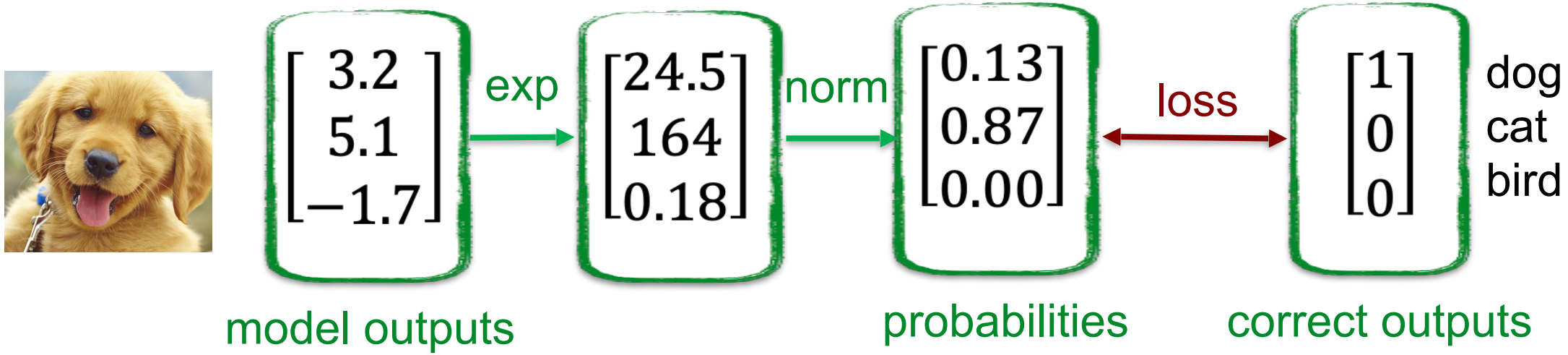


Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log Prob[f(x_i, W) == y_i]$$

In this case, what is the loss:

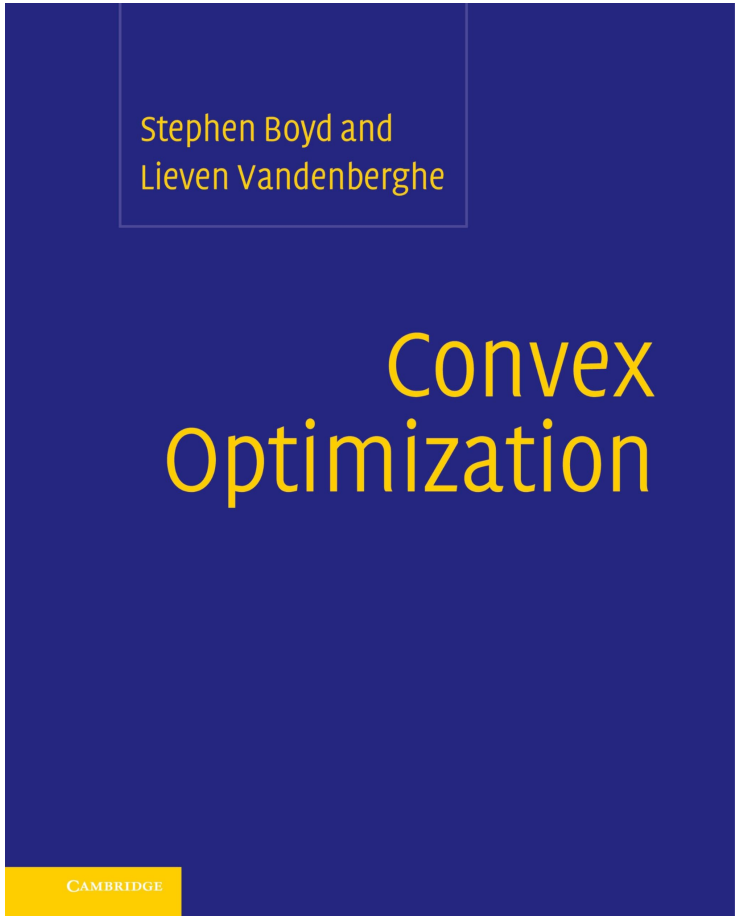
$$L_i = -\log(0.13) = 2.04$$



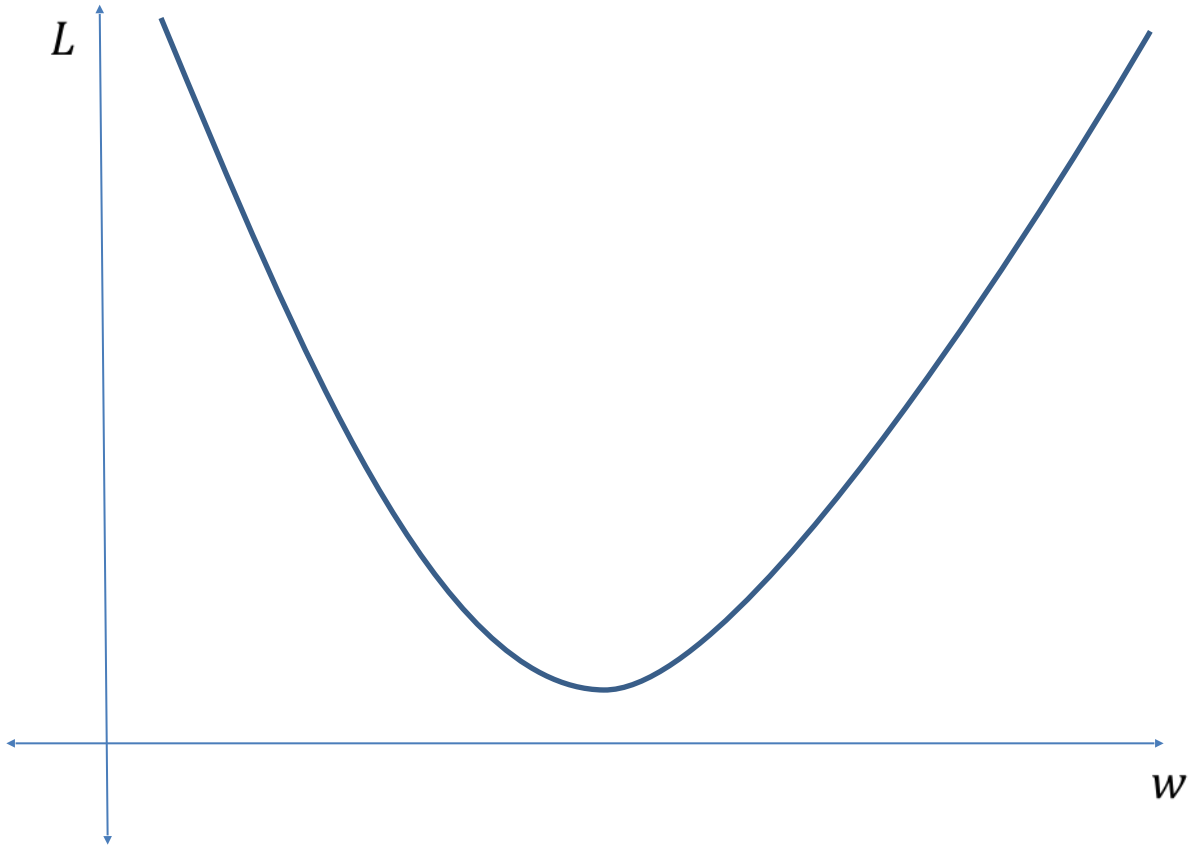
Today's agenda

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

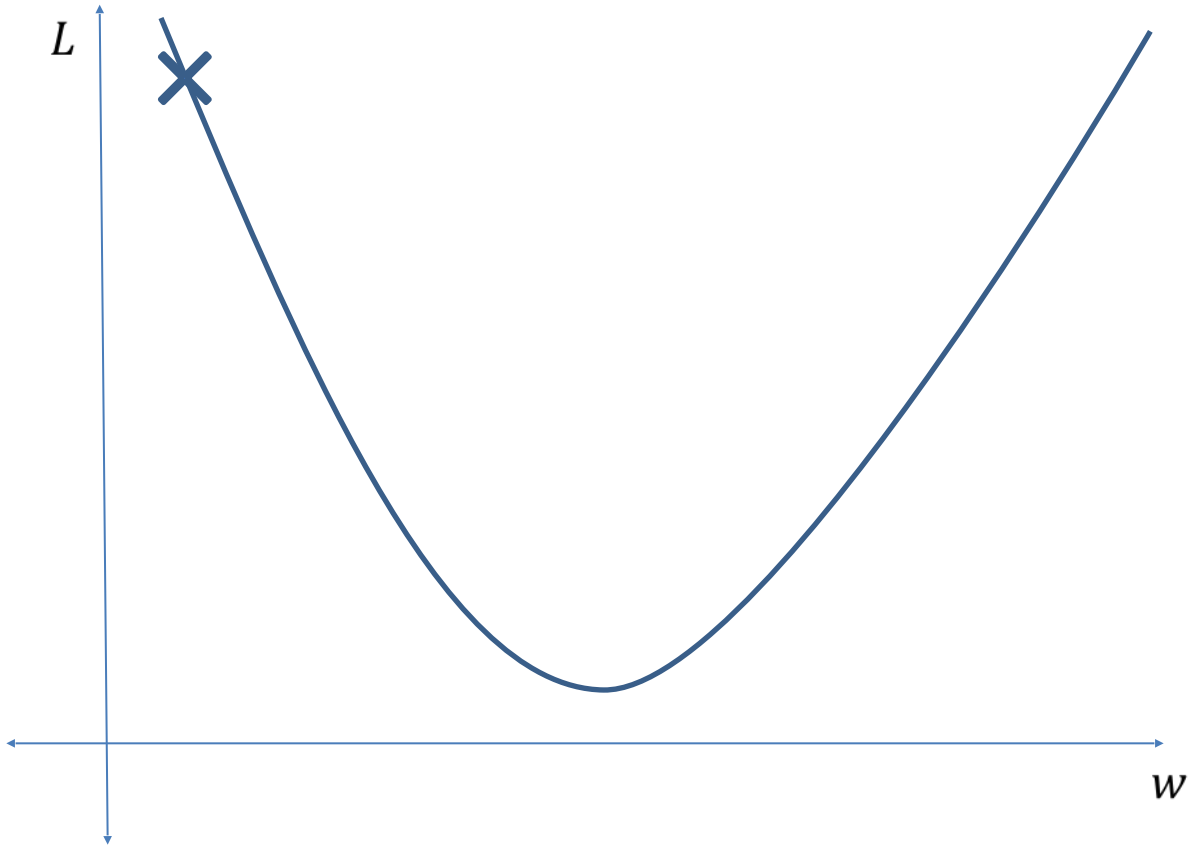
How do we find the weights that minimize the loss?



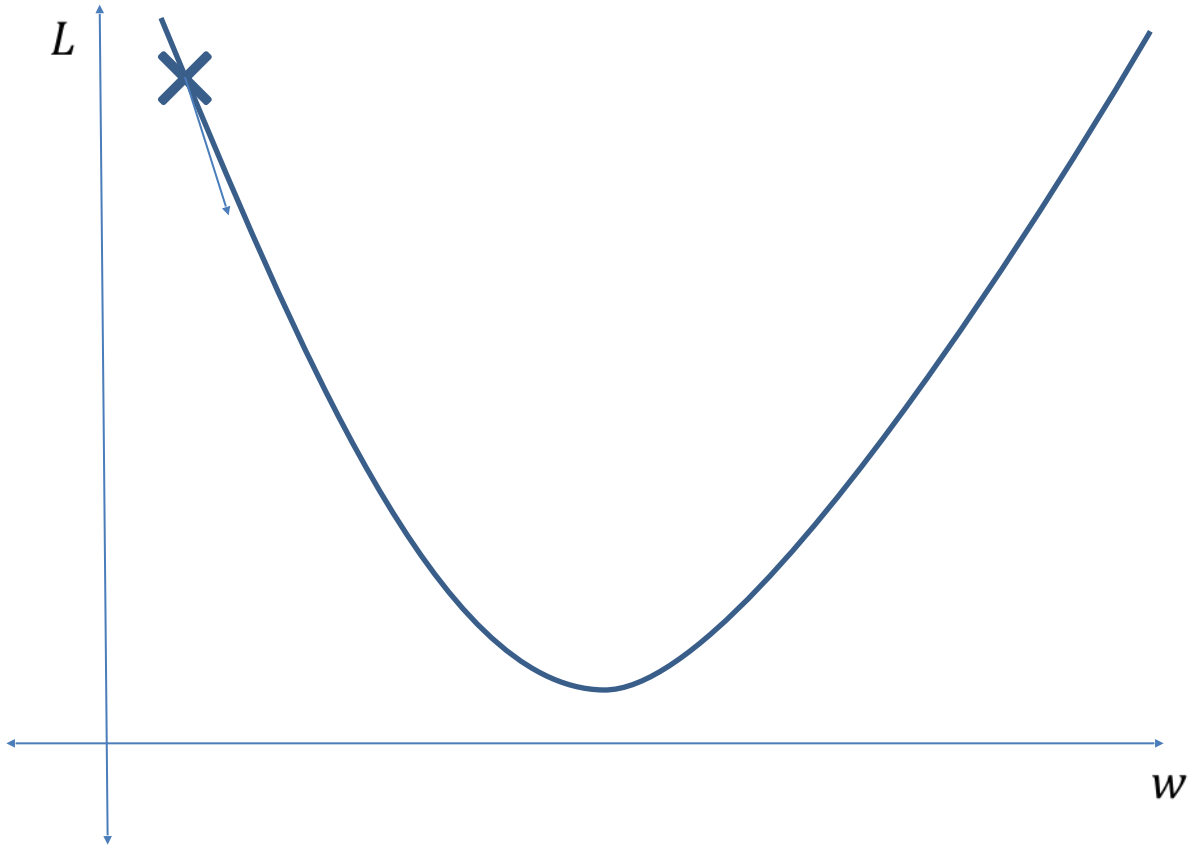
Gradient descent visualized: Minimizing loss



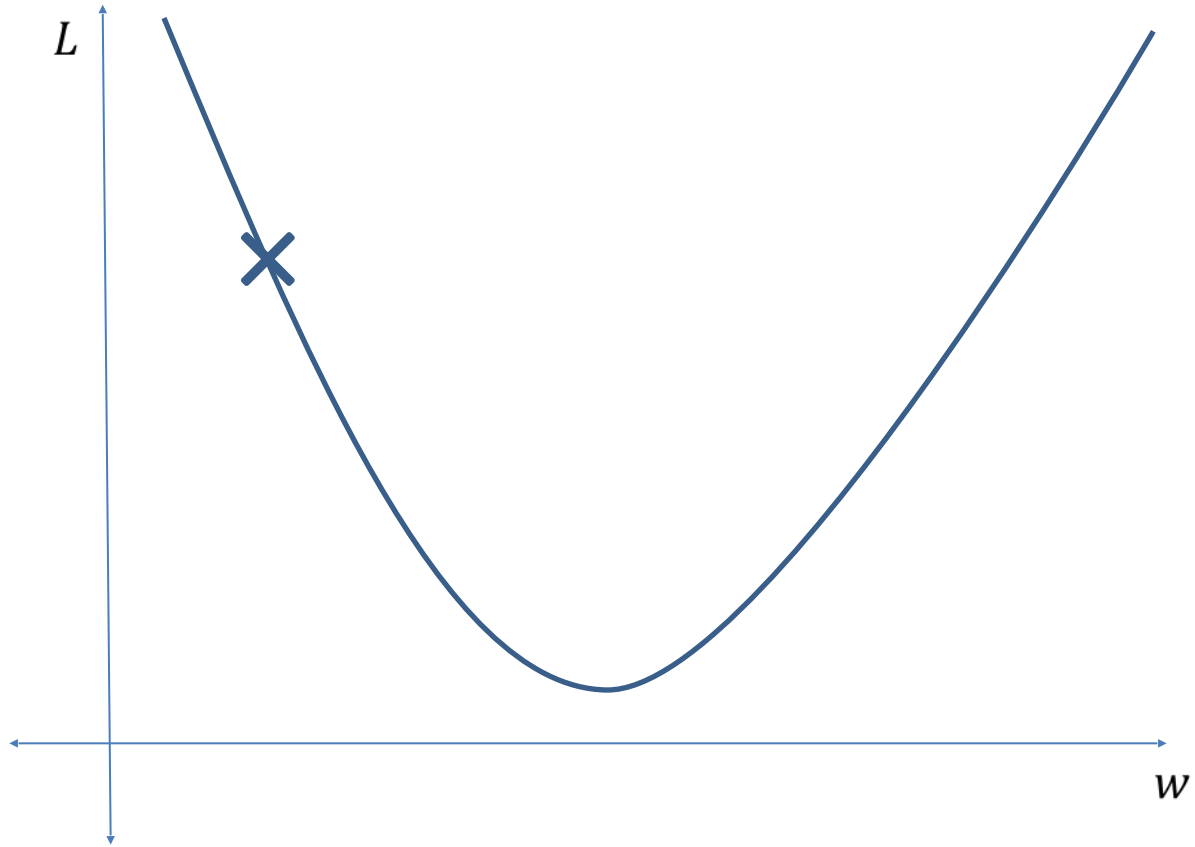
Gradient descent visualized: Minimizing loss



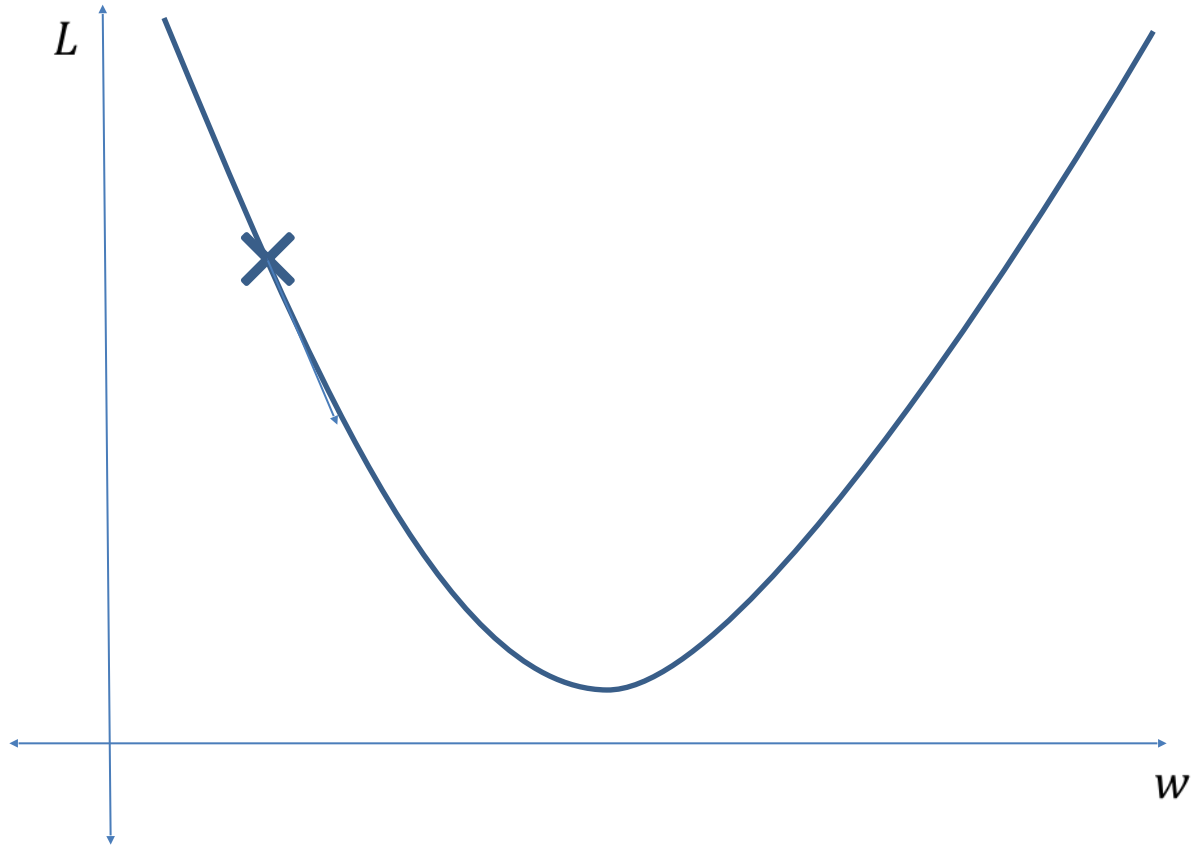
Gradient descent visualized: Minimizing loss



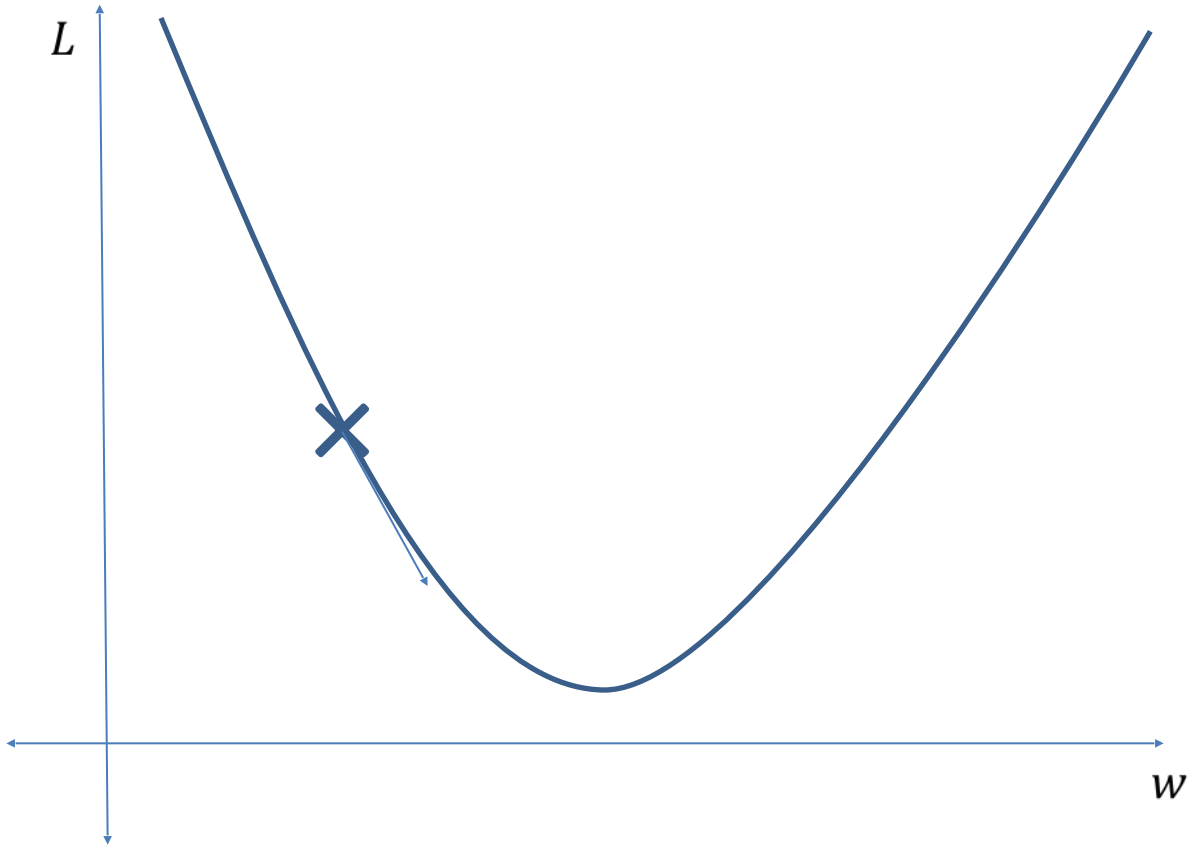
Gradient descent visualized: Minimizing loss



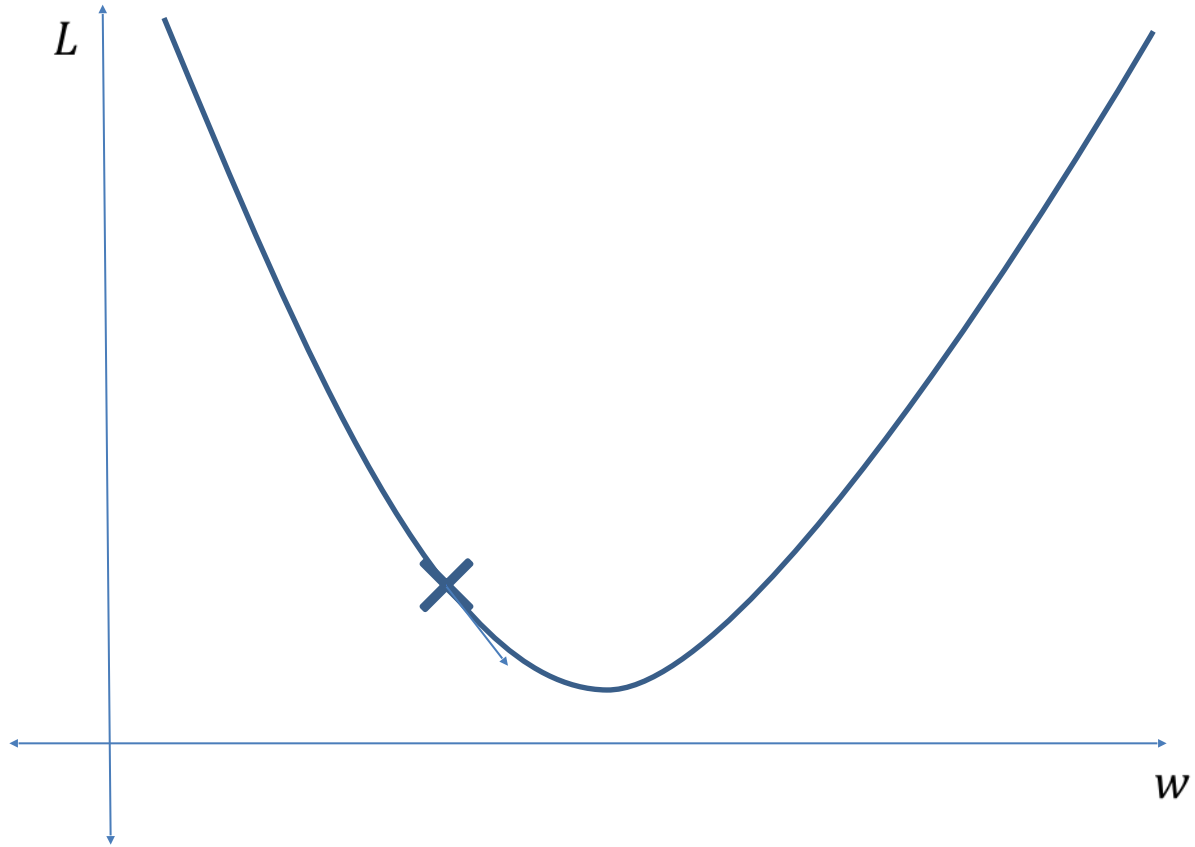
Gradient descent visualized: Minimizing loss



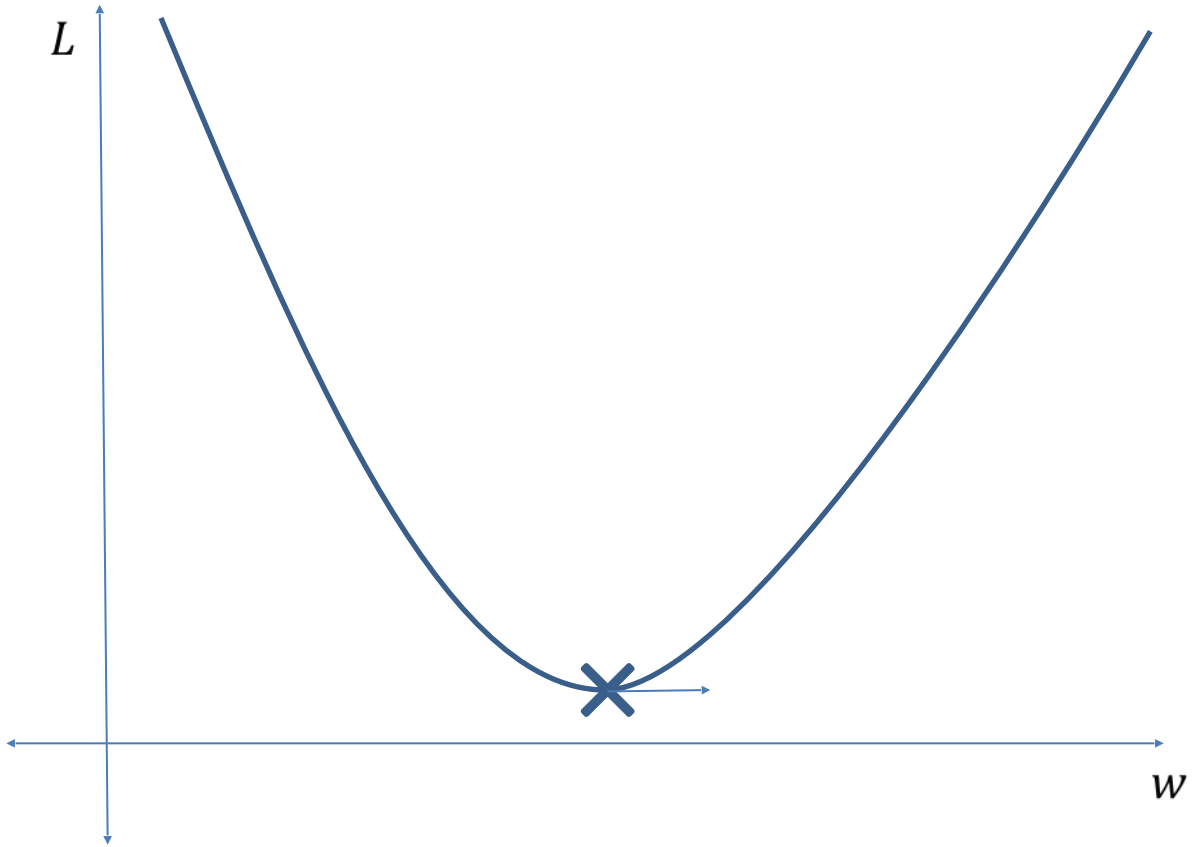
Gradient descent visualized: Minimizing loss



Gradient descent visualized: Minimizing loss



Gradient descent visualized: Minimizing loss



Gradient Descent Pseudocode

```
for _ in {0,…,num_epochs}:
    L = 0
    for x_i, y_i in data:
        ŷ_i = f(x_i, W)
        L += L_i(y_i, ŷ_i)
    dL/dW = ???
    W := W − α dL/dW
```

Small step x Gradient

Gradient Descent Pseudocode

```
for _ in {0,…,num_epochs}:
    L = 0
    for x_i, y_i in data:
        ŷ_i = f(x_i, W)
        L += L_i(y_i, ŷ_i)
    dL/dW = ???
    W := W − α dL/dW
```

Small step x Gradient

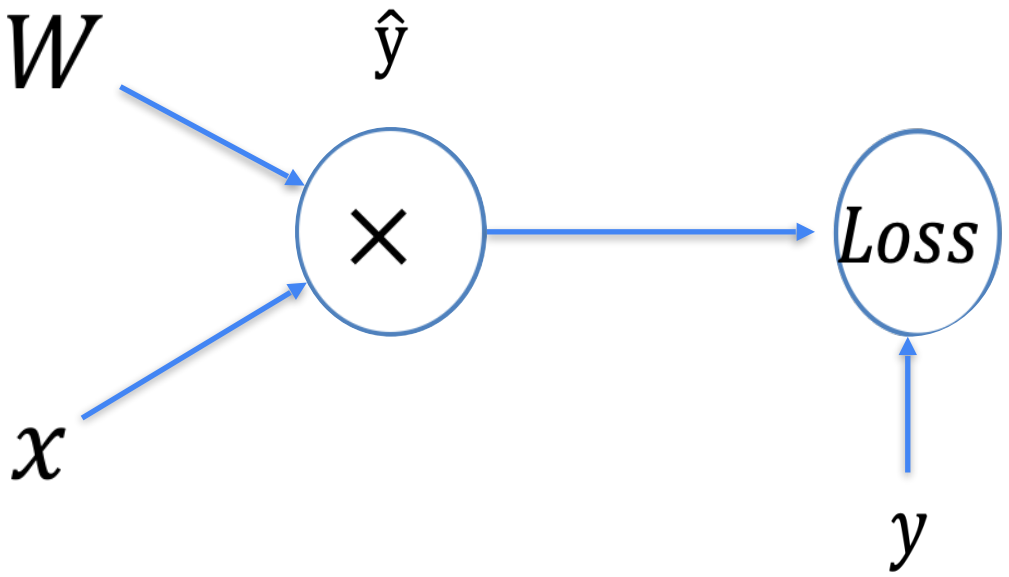
Exercise on linear classification:

$$\hat{y} = Wx$$

$$\text{Loss} = -\hat{y}_k + \log \sum_j e^{\hat{y}_j}$$

$$\frac{dL}{dW} = \cdot \begin{bmatrix} \frac{e^{\hat{y}_0}}{\sum_j e^{\hat{y}_j}} \\ \cdots \\ \cdots \\ -1 + \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}} \\ \cdots \\ \cdots \\ \frac{e^{\hat{y}_{3071}}}{\sum_j e^{\hat{y}_j}} \end{bmatrix} x$$

Backprop – another way of computing gradients

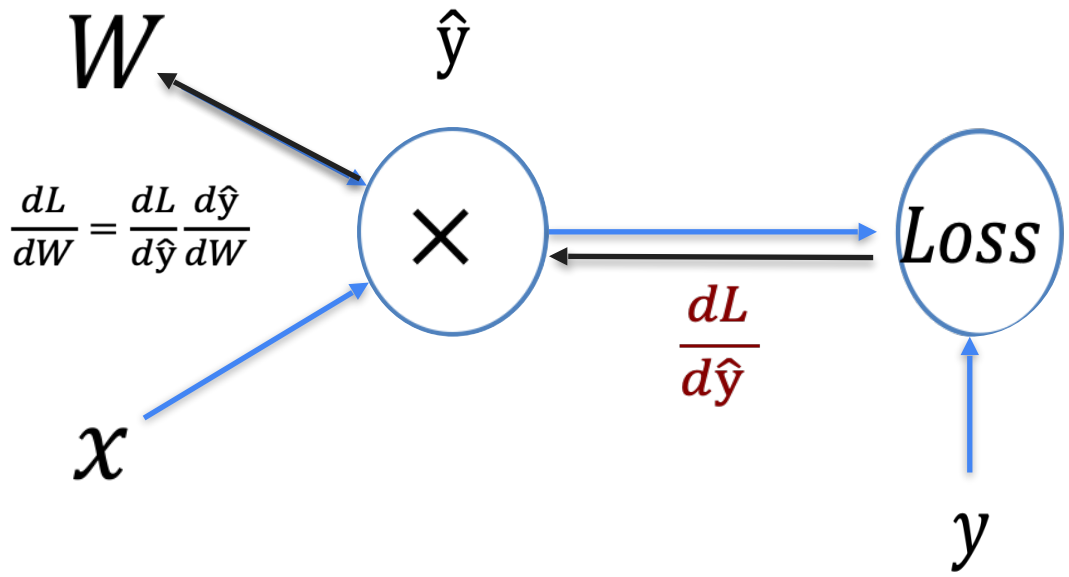


$$\hat{y} = Wx$$
$$L = Loss(\hat{y}, y)$$

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW}$$

Calculating the gradient is hard, but we can use the chain rule to make it simpler

Backprop – a way of computing gradients



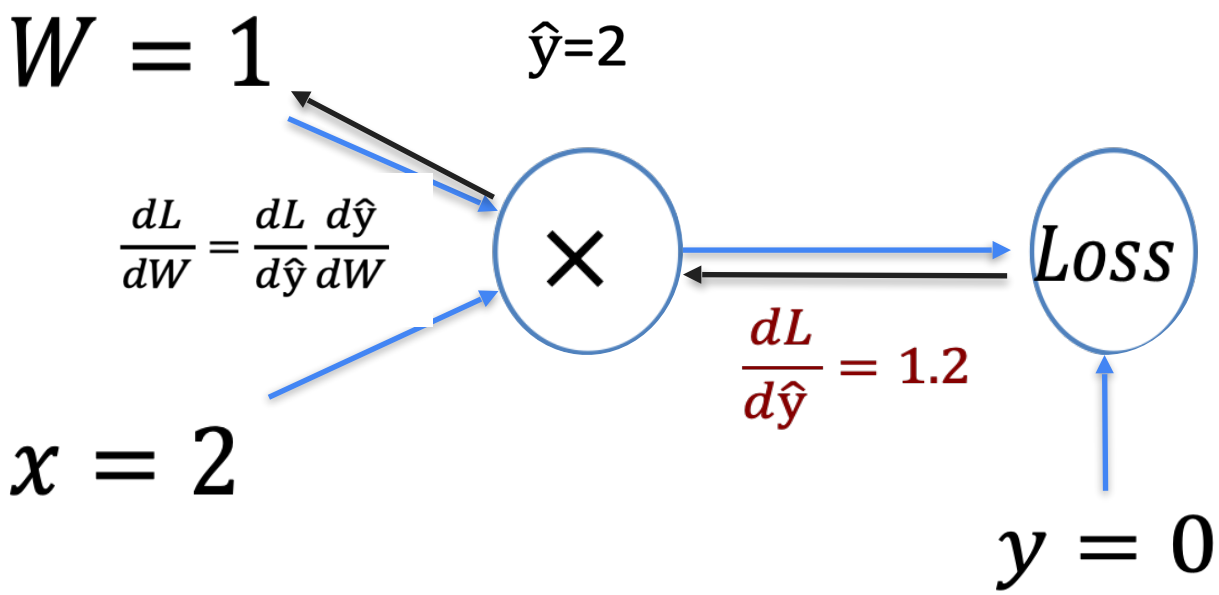
$$\hat{y} = Wx$$
$$L = Loss(\hat{y}, y)$$

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW}$$

Key Insight:

- visualize the computation as a graph flow
- Compute the forward pass to calculate the loss.
- Compute all gradients for each pair of nodes backwards

Backprop example in 1D:



We know the chain rule

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW}$$
$$= \frac{dL}{d\hat{y}}x$$
$$= 1.2x$$
$$= 1.2 \times 2$$
$$= 2.4$$