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)

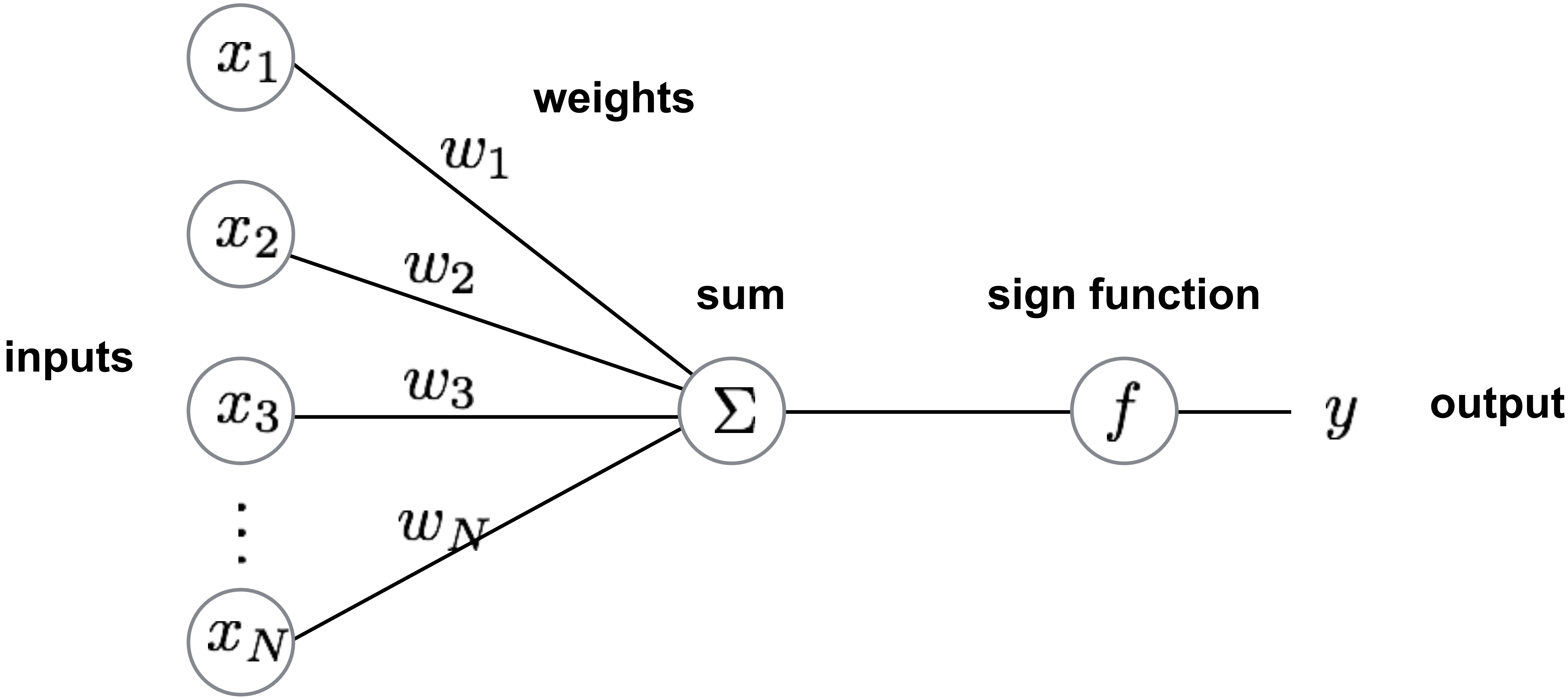
1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

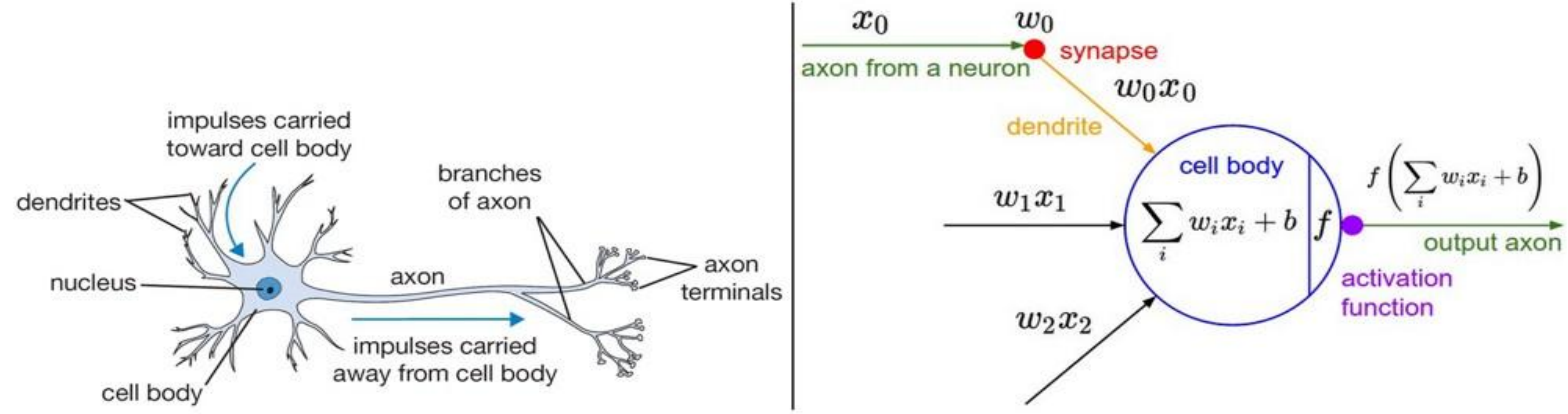
2010s Age of the Deep Network

deep learning = known algorithms + computing power + big data

Perceptron


x_1
x_2
x_3
x_N
weights
w_1
w_2
w_3
w_N
inputs
sum
Σ
sign function
f
y
output

Aside: Inspiration from Biology

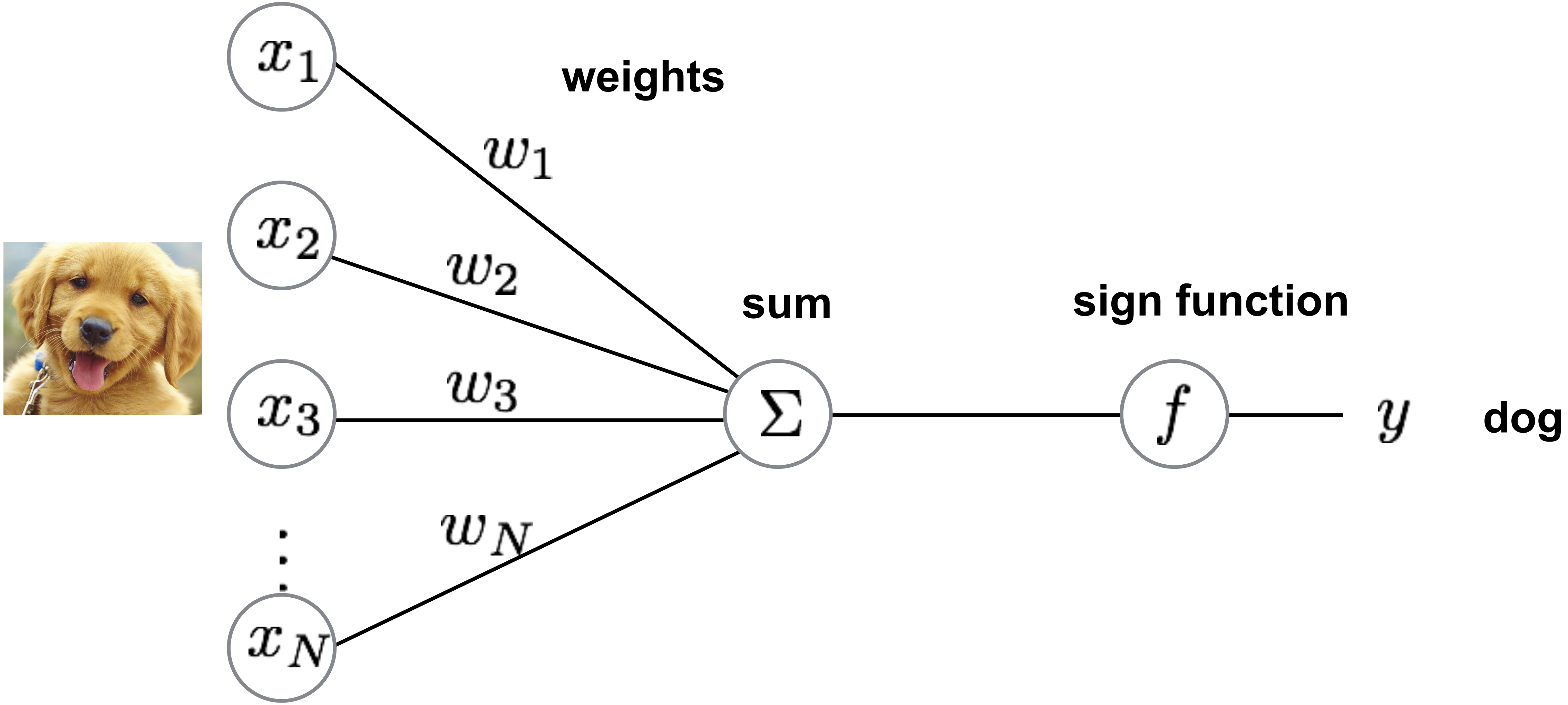


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

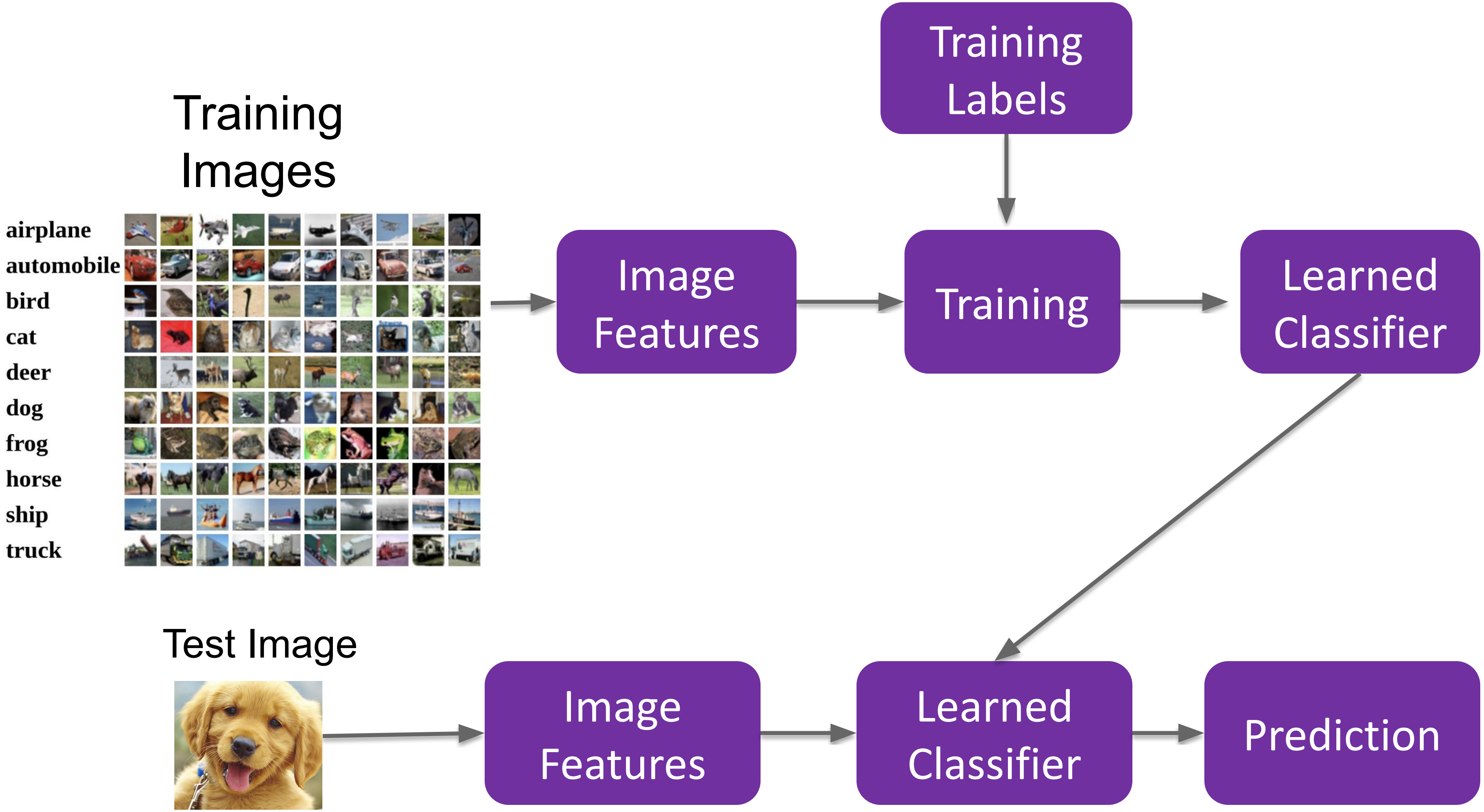
Neural nets/perceptrons are loosely inspired by biology.

But they are NOT how the brain works, or even how neurons work.

Perceptron: for image classification



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



Training Images

Training Labels

Image Features

Training

Learned Classifier

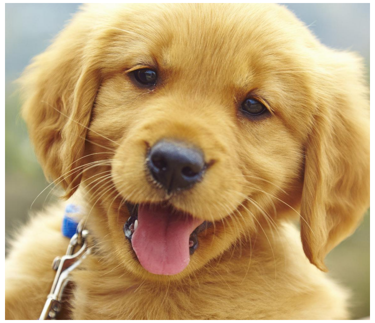
Test Image

Image Features

Learned Classifier

Prediction

Recall: we can featurize images into a vector



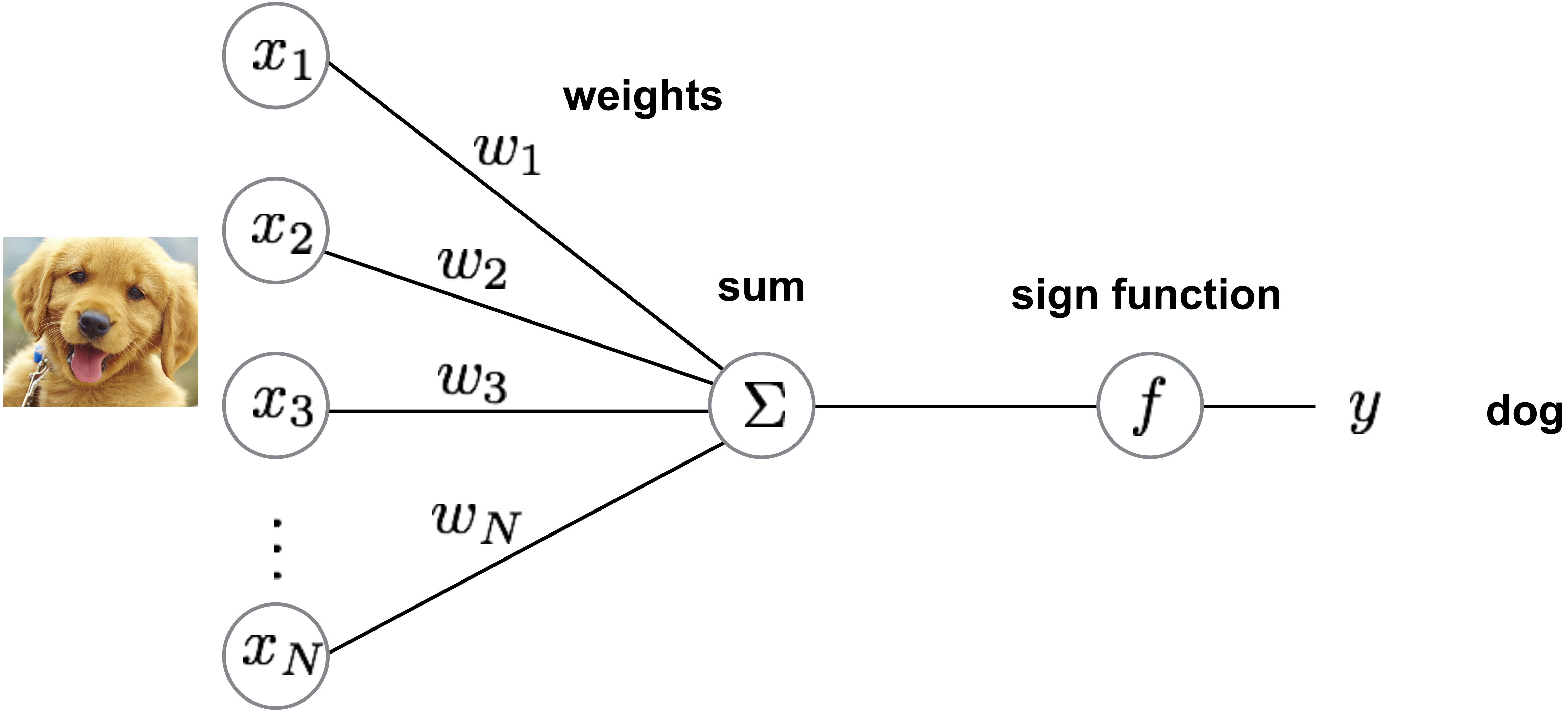
Raw pixels
Raw pixels + (x,y)
PCA
BoW
BoW + spatial pyramids

Image
Vector

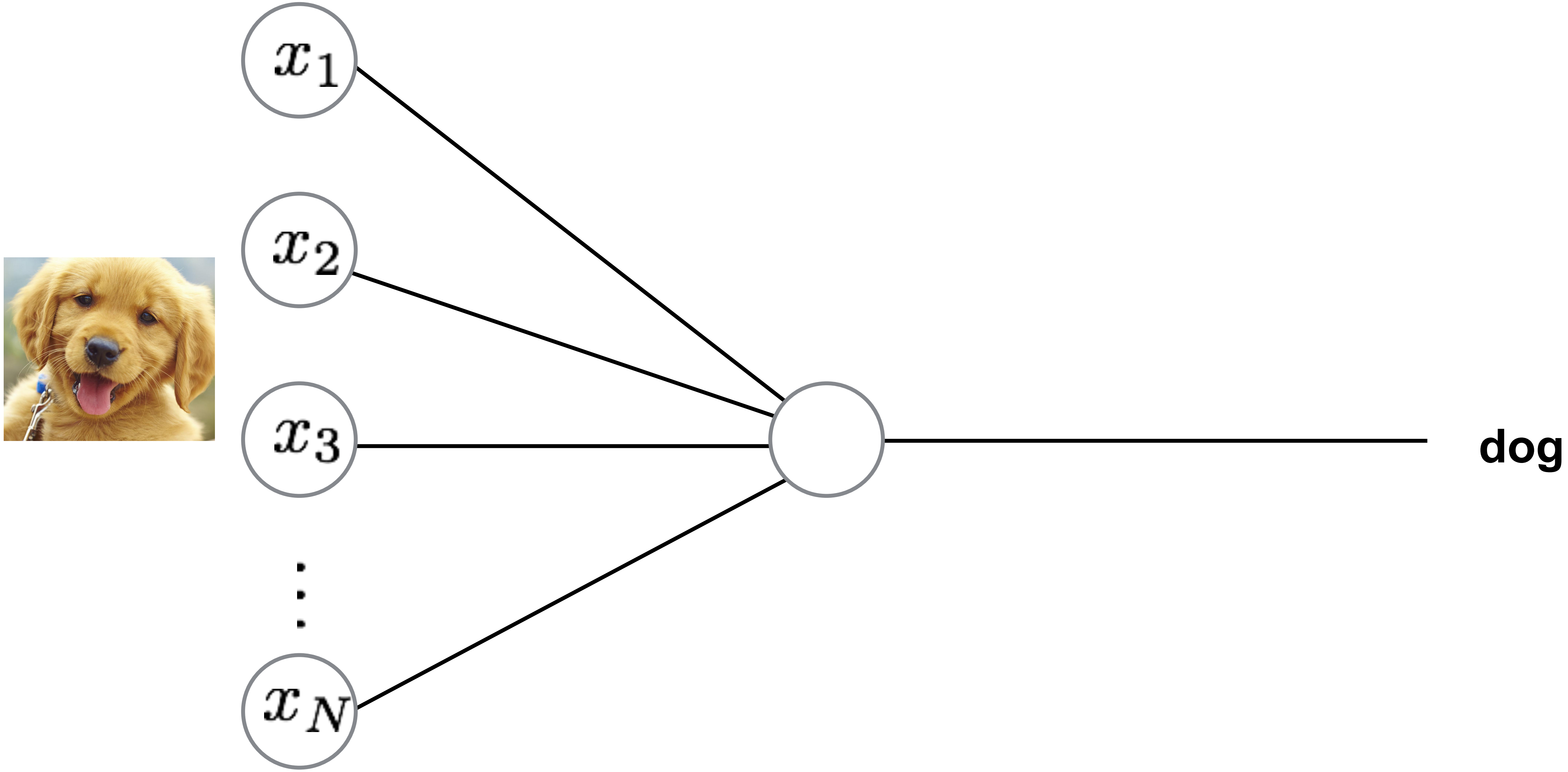
x_1
x_2
x_3
…
…
…
…
…
…
…
…
…

x_n

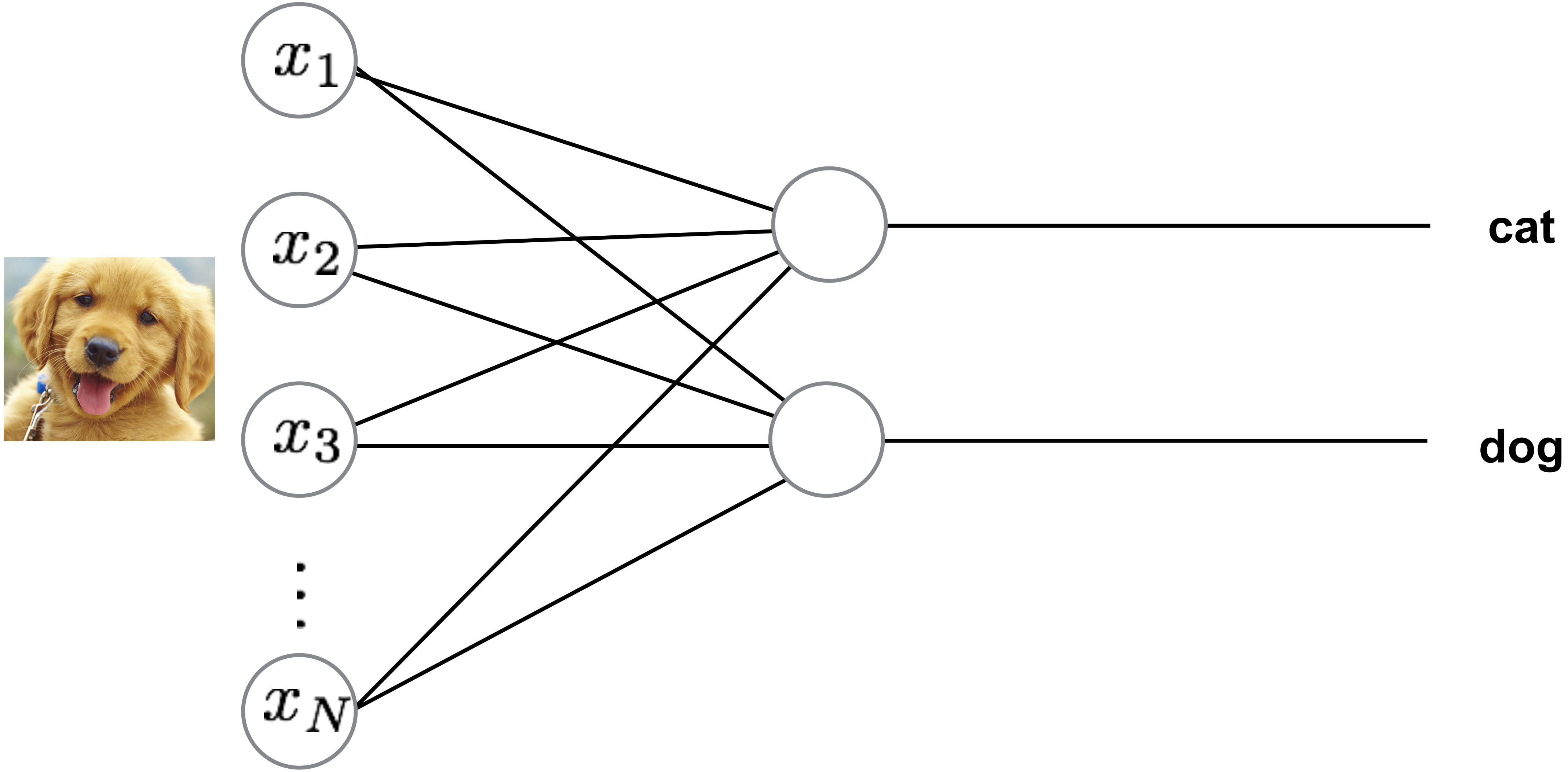
Perceptrons are a simple transformation that converts feature vectors into recognition scores

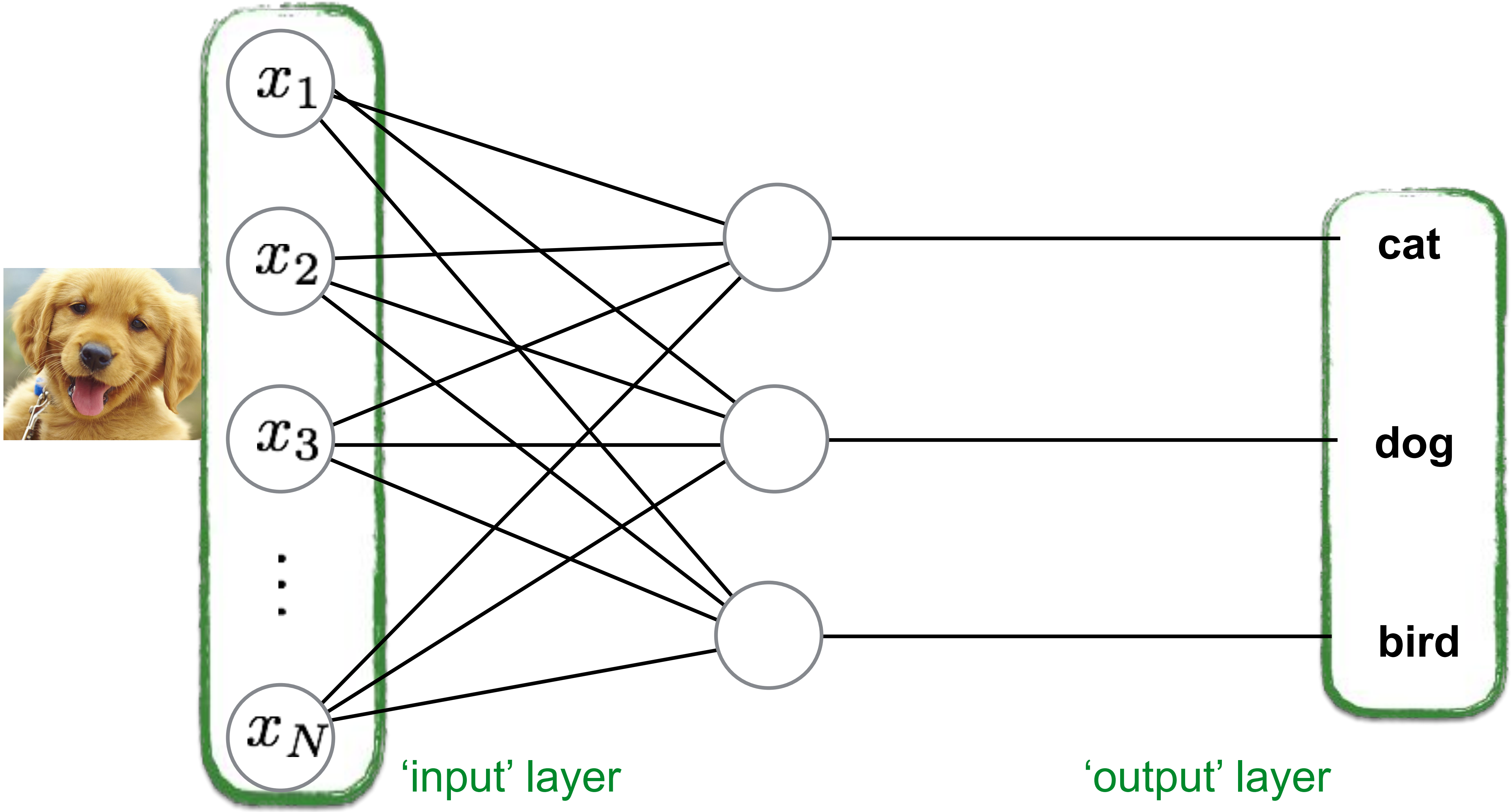


Perceptron: simplified view with one perceptron (produces 1 score for one category)



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





cat

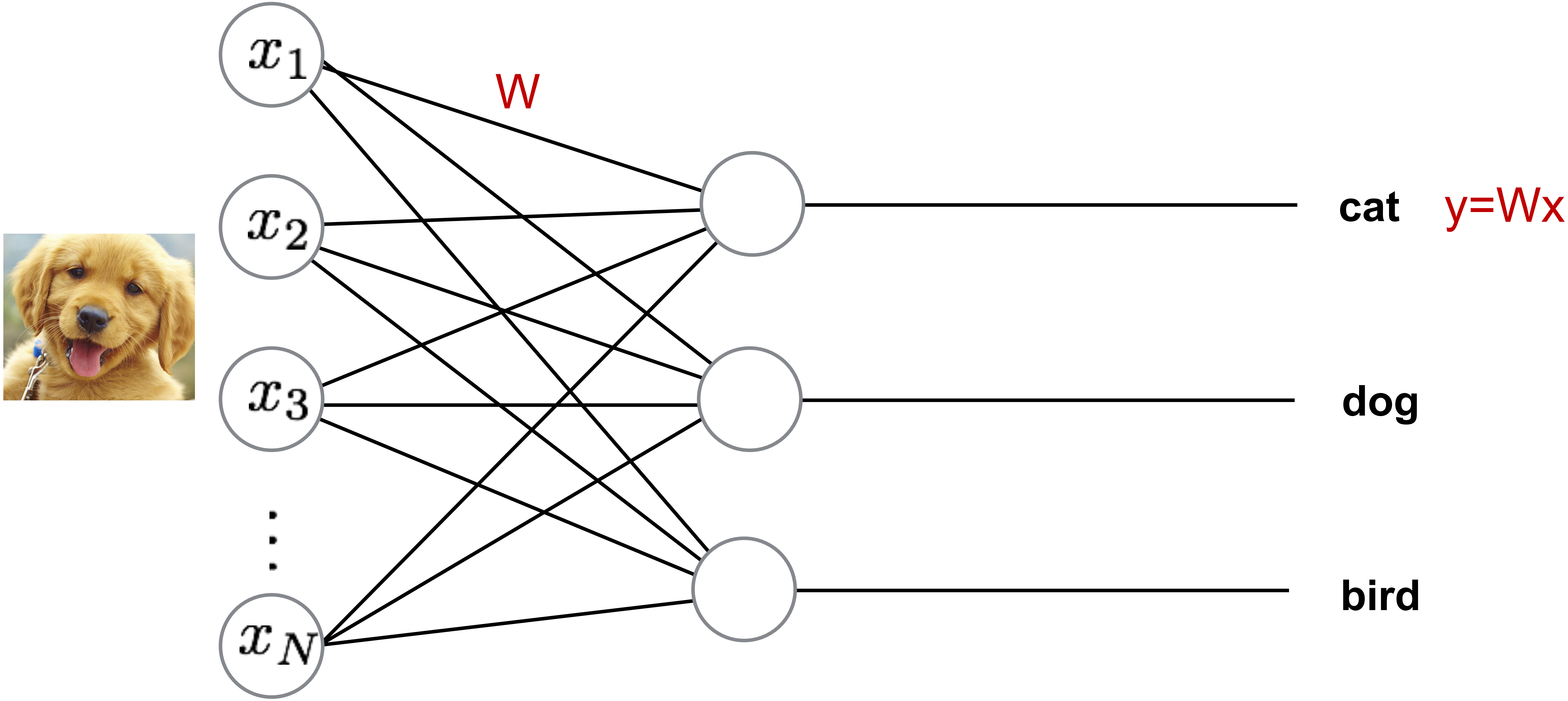
dog

bird

‘input’ layer

‘output’ layer

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



y=Wx

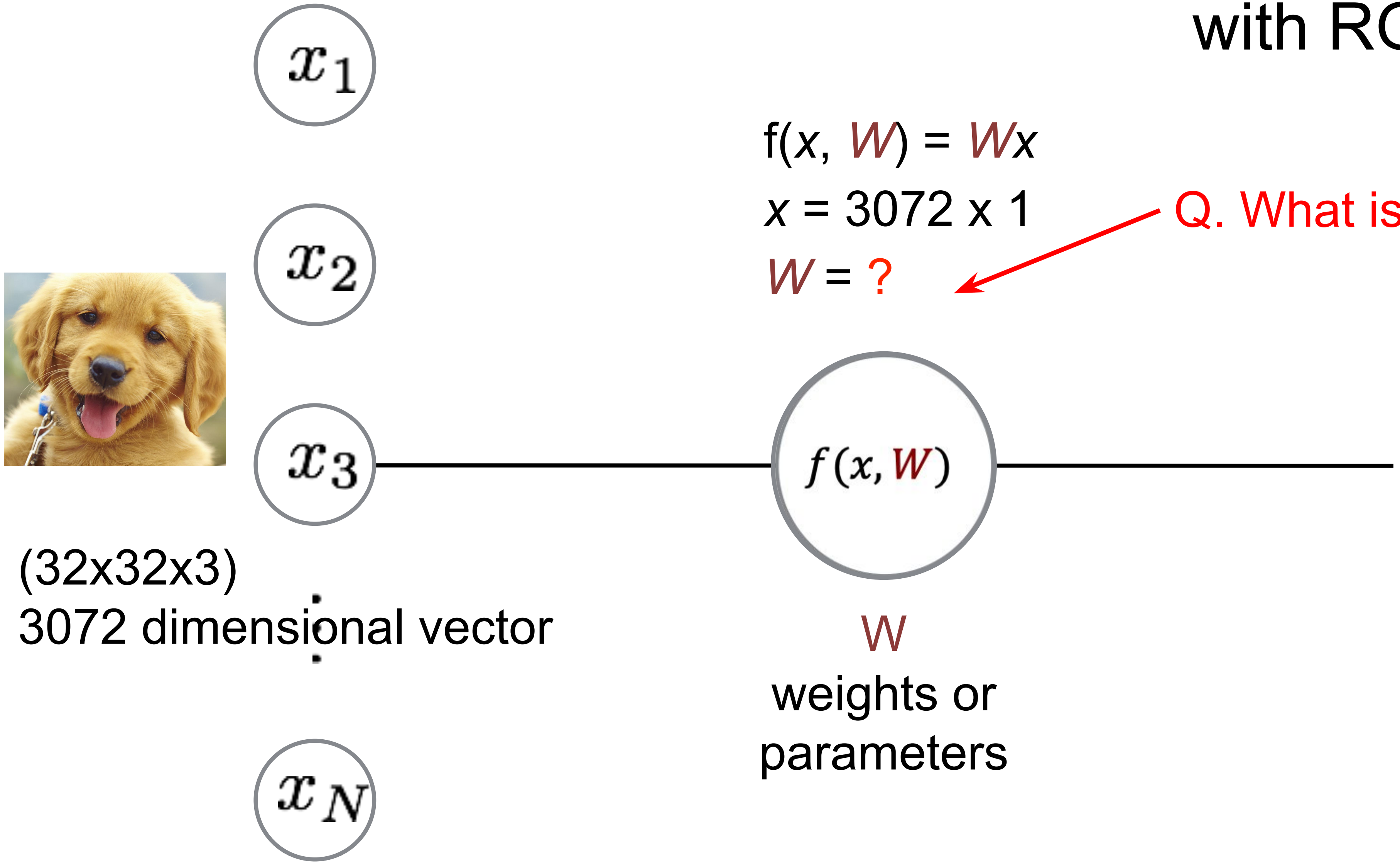
Linear classifier: mathematical formulation with RGB features

$f(x, W) = Wx$

$x = 3072 \times 1$

$W = ?$

Q. What is the shape of W?

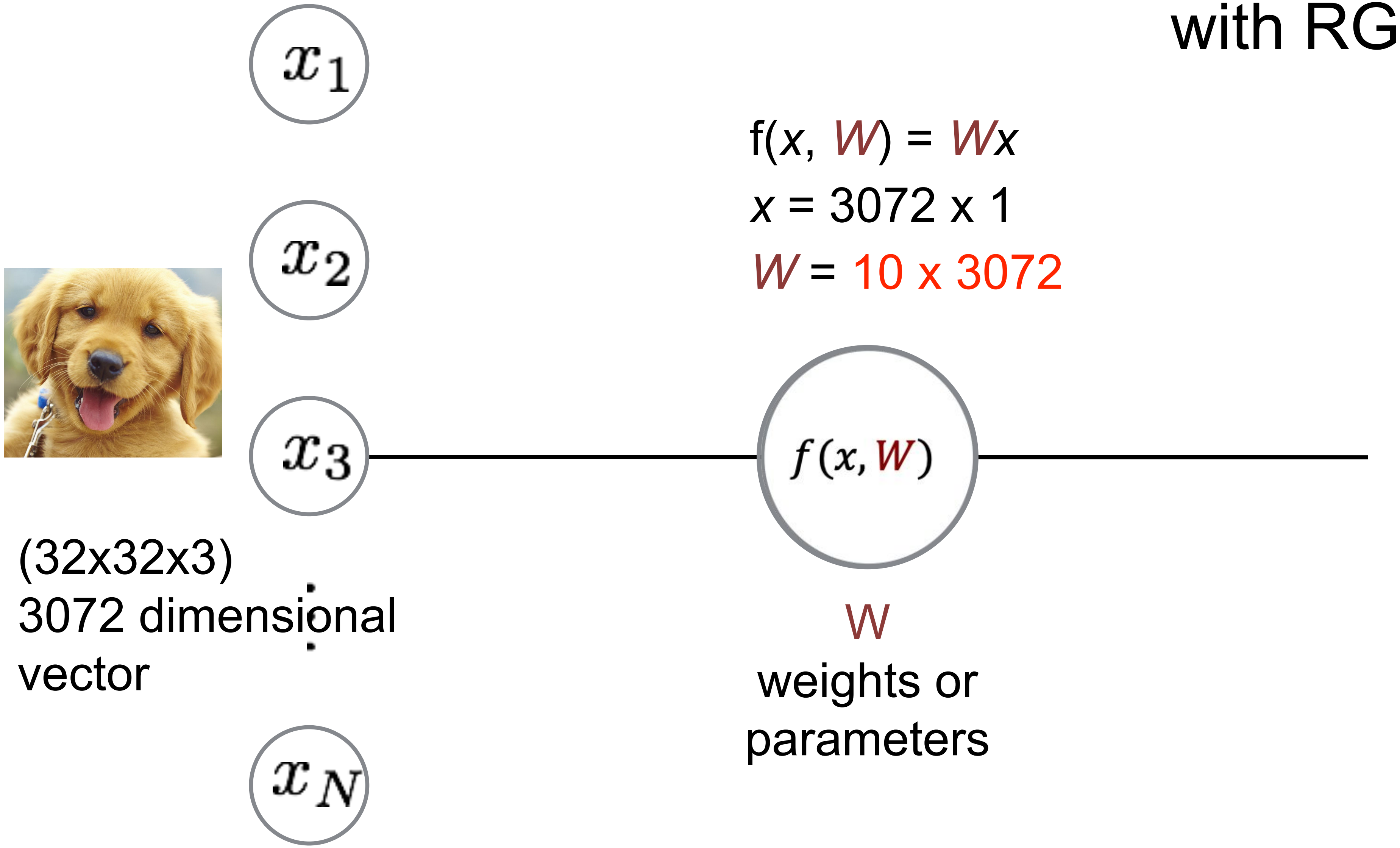


(32x32x3)
3072 dimensional vector

W
weights or
parameters

10 numbers
giving class
scores

Linear classifier: mathematical formulation with RGB features



$f(x, W) = Wx$

$x = 3072 \times 1$

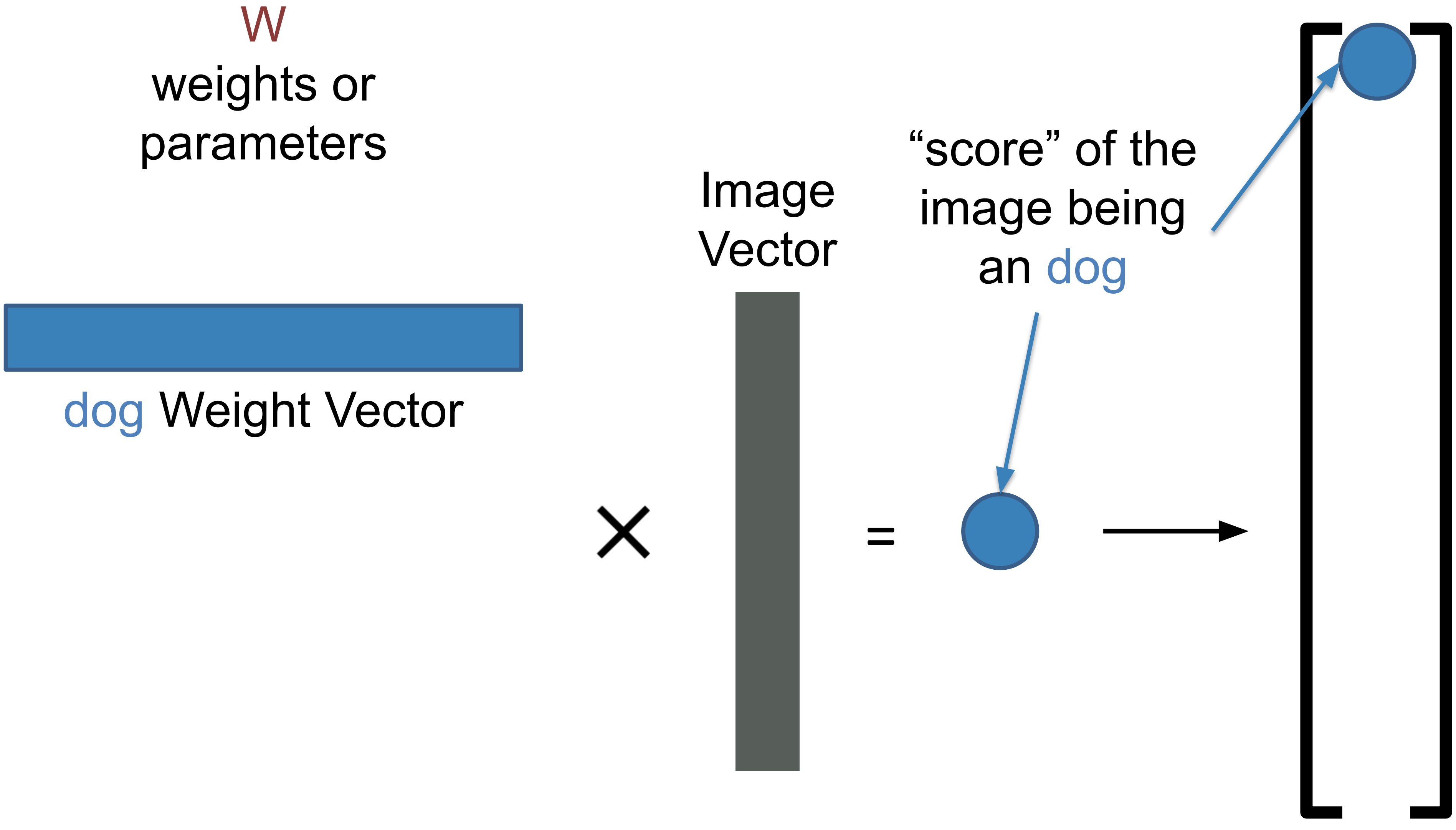
$W = 10 \times 3072$

(32x32x3)
3072 dimensional
vector

W
weights or
parameters

10 numbers
giving class
scores

Linear classifier: function visualized



W
weights or
parameters

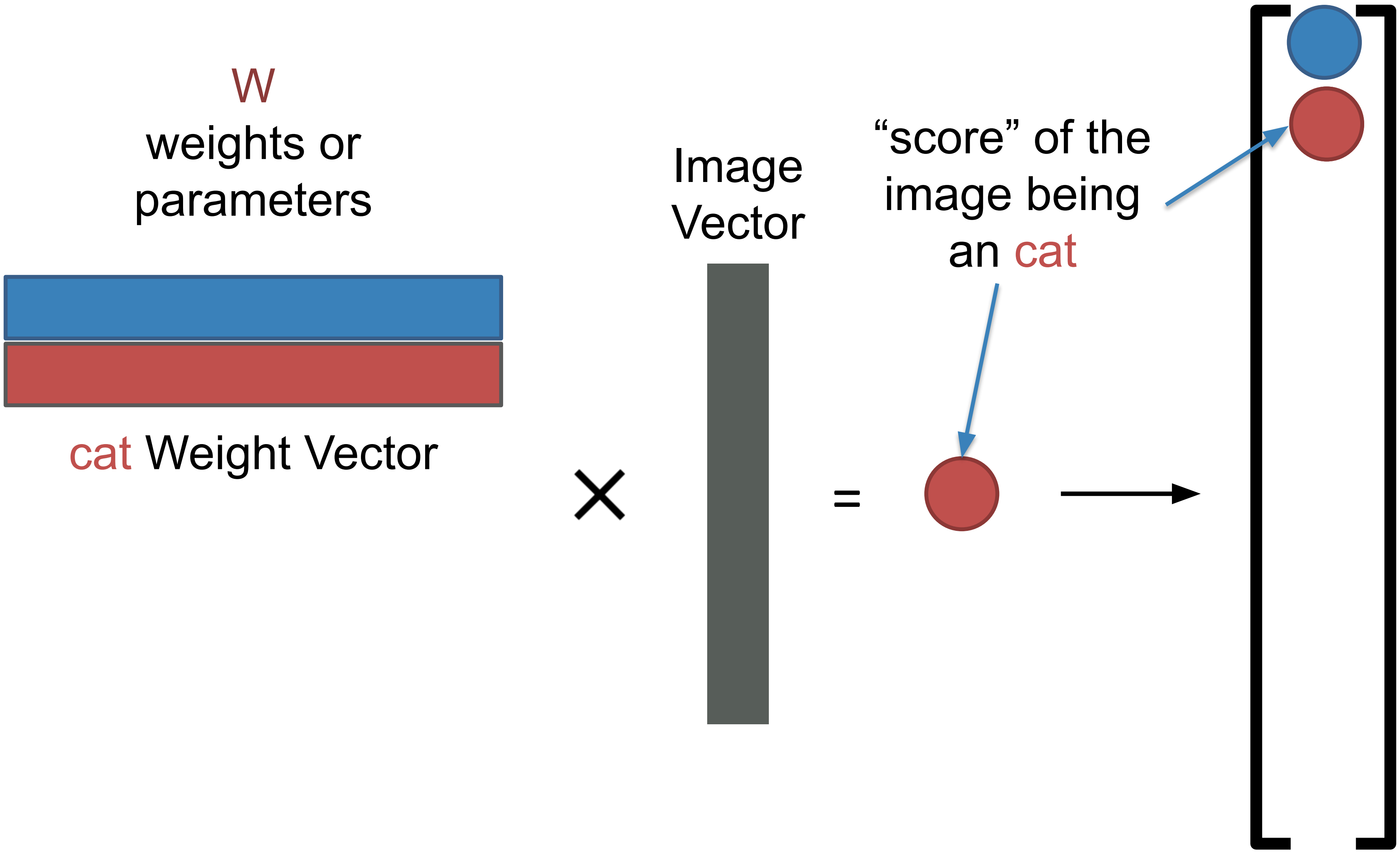
dog Weight Vector

Image
Vector

× =

"score" of the
image being
an dog

Linear classifier: function visualized



W

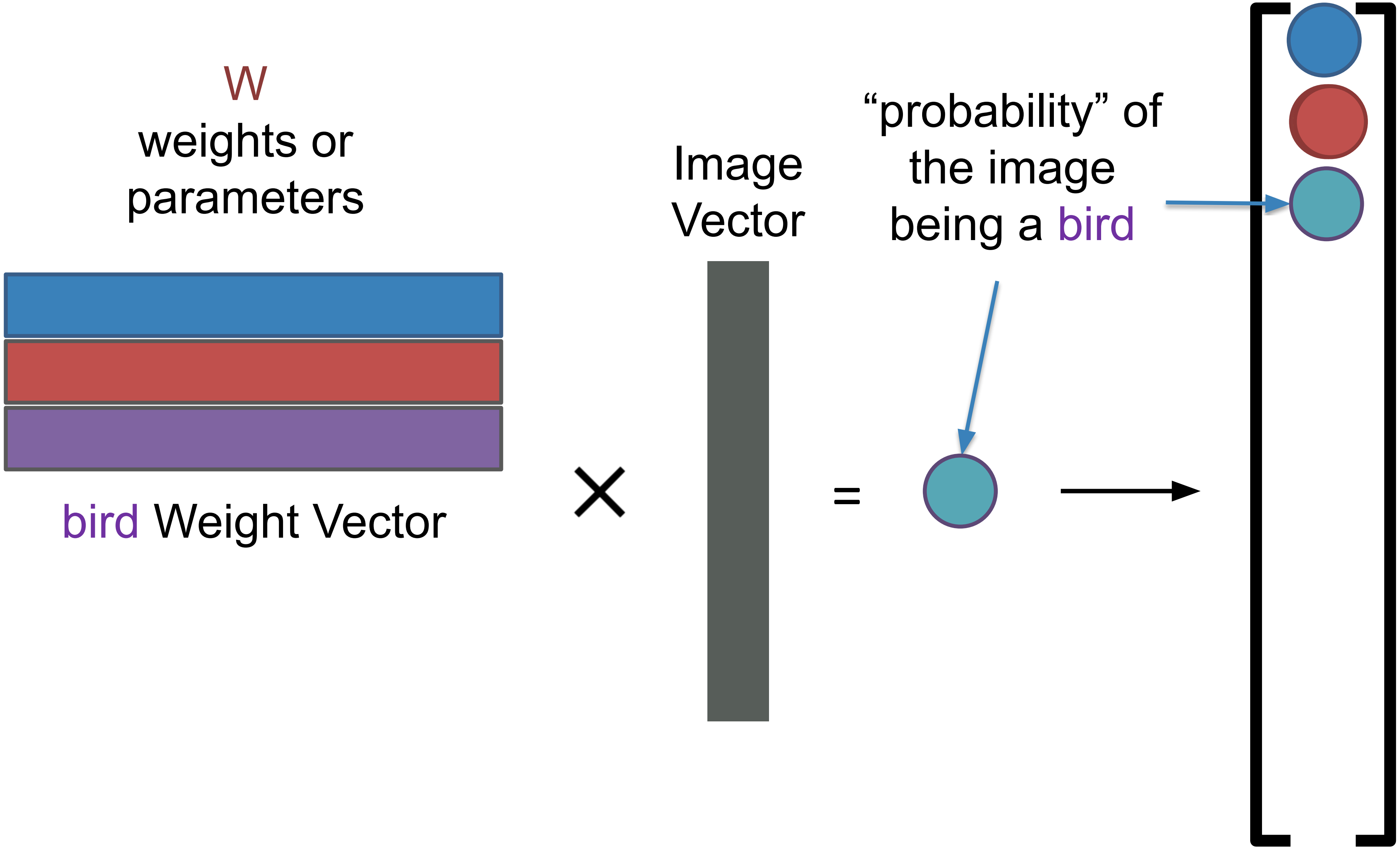
weights or parameters

cat Weight Vector

Image Vector

“score” of the image being an cat

Linear classifier: function visualized



W
weights or
parameters

bird Weight Vector

Image
Vector

“probability” of
the image
being a bird

Linear classifier: function visualized

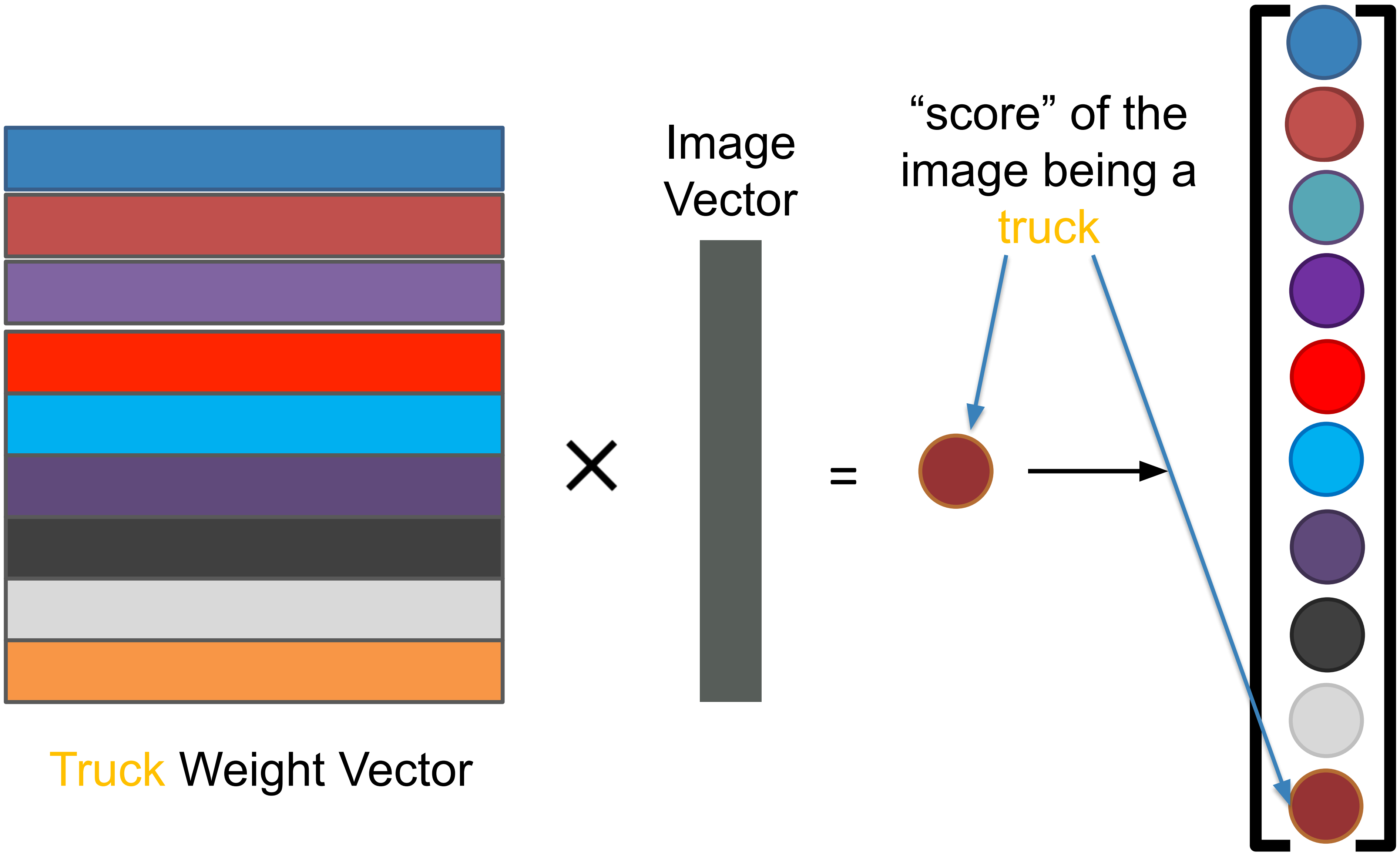
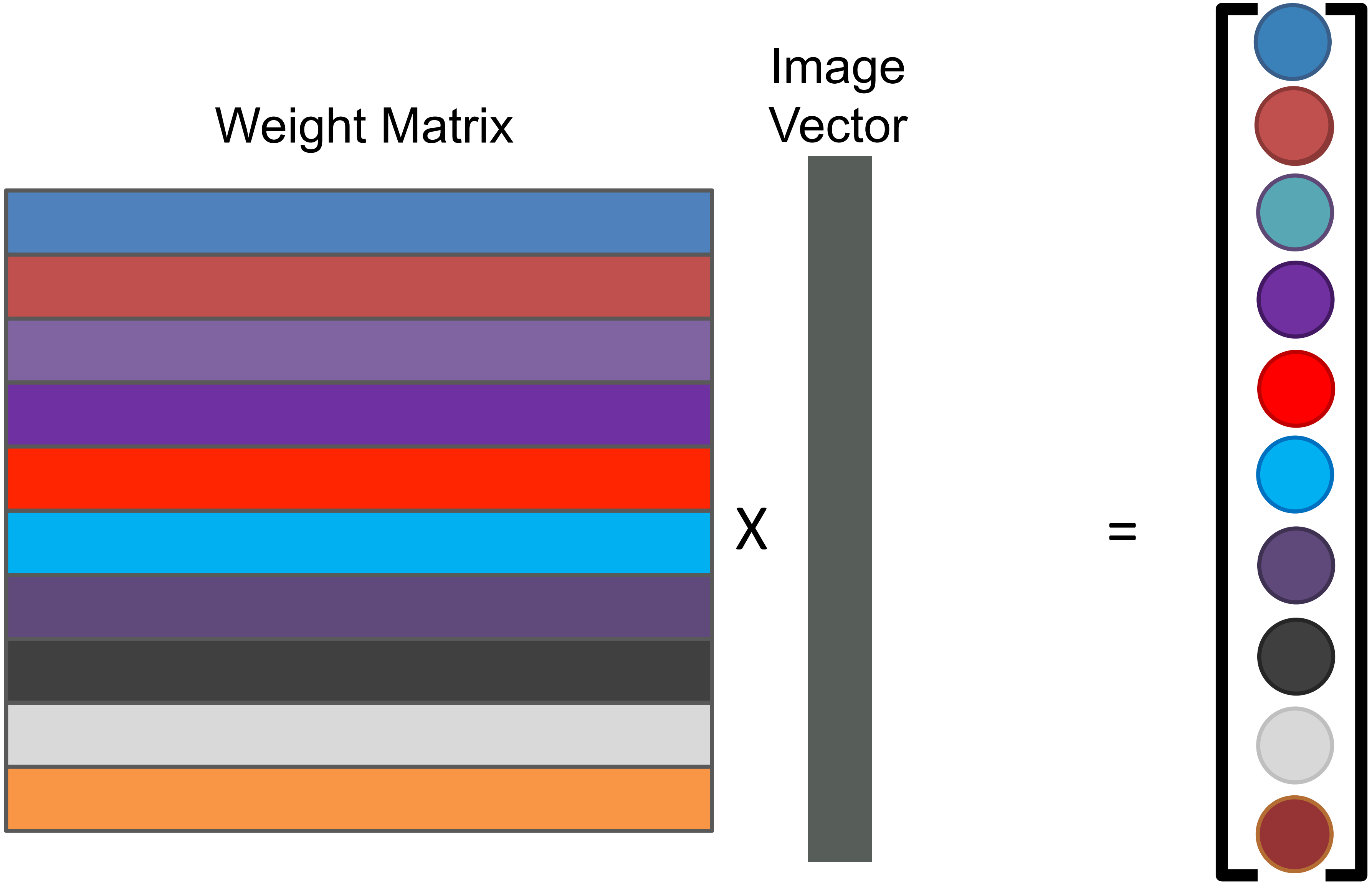


Image Vector

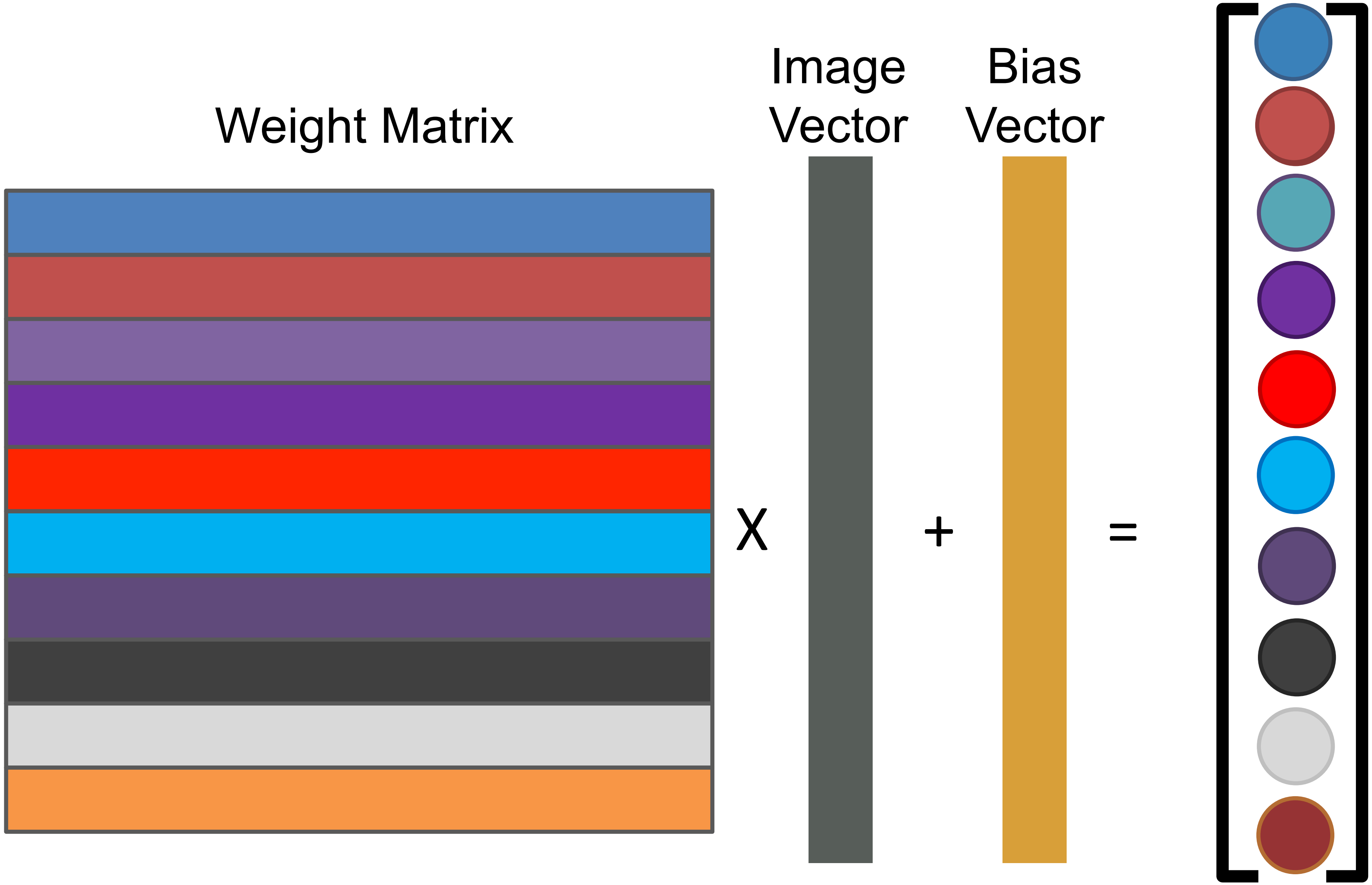
“score” of the image being a truck

Truck Weight Vector

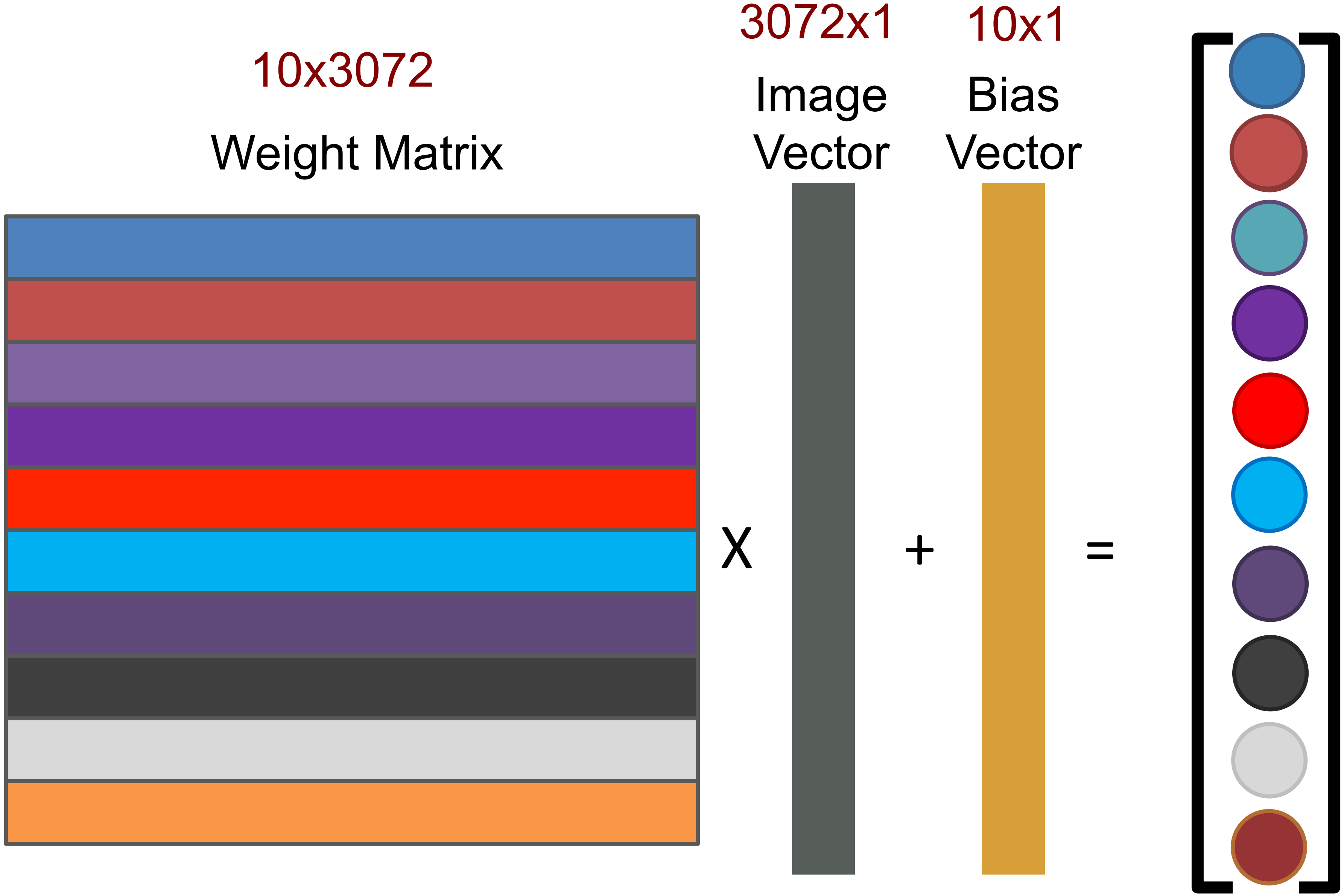
Linear classifier: function visualized



Linear classifier: bias vector



Linear classifier: size



10x3072

Weight Matrix

X

3072x1

Image Vector

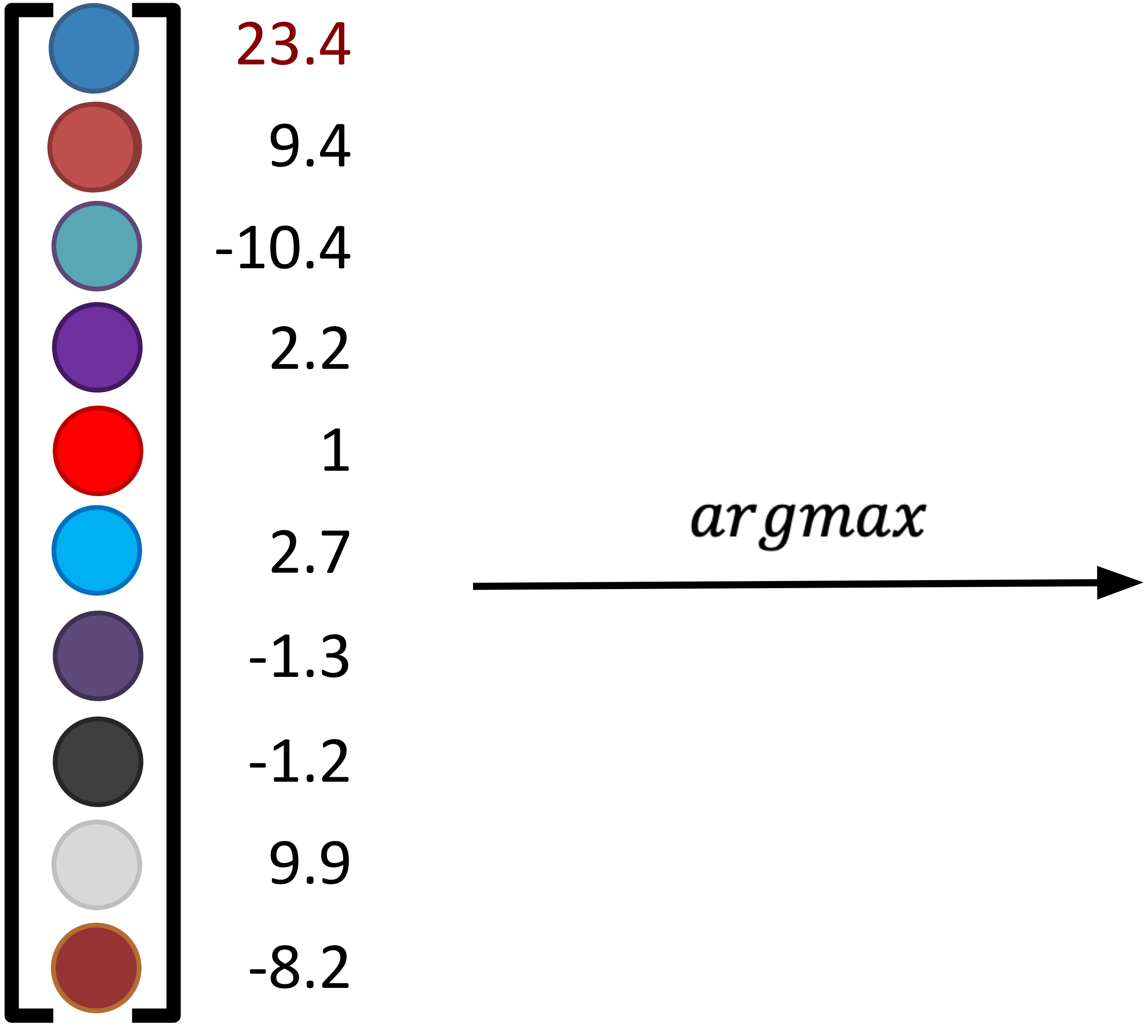
+

10x1

Bias Vector

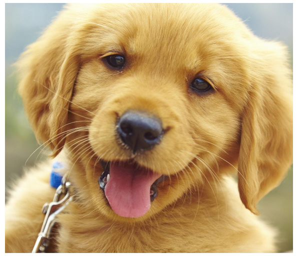
=

Linear classifier: Making a classification

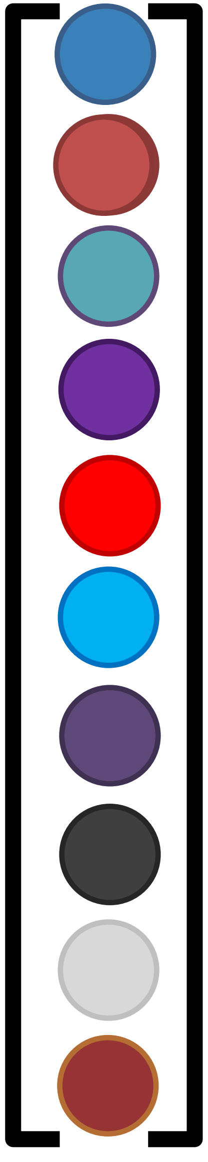


23.4
9.4
-10.4
2.2
1
2.7
-1.3
-1.2
9.9
-8.2

$\xrightarrow{argmax}$ **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:

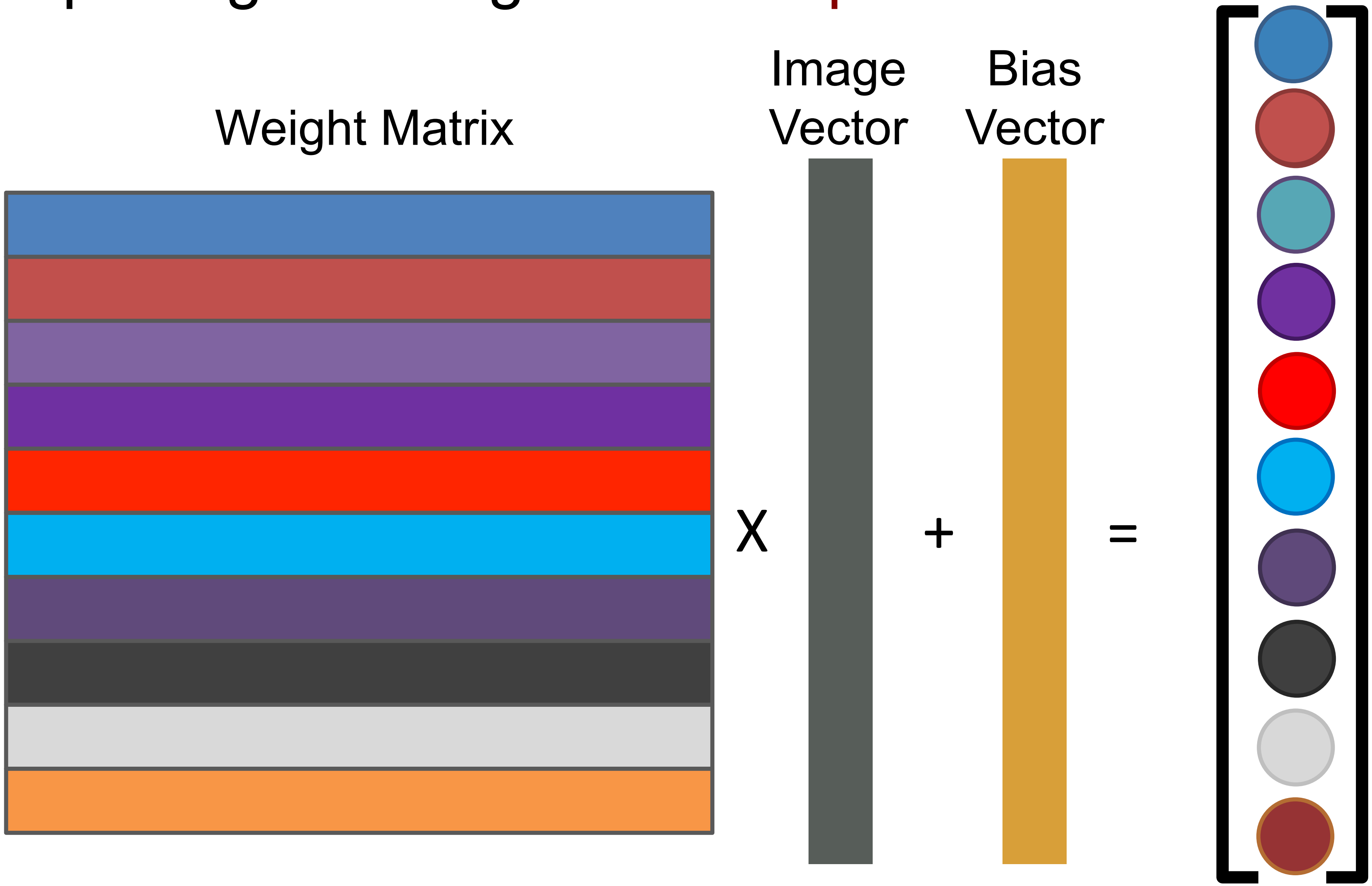
$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

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

Interpreting the weights

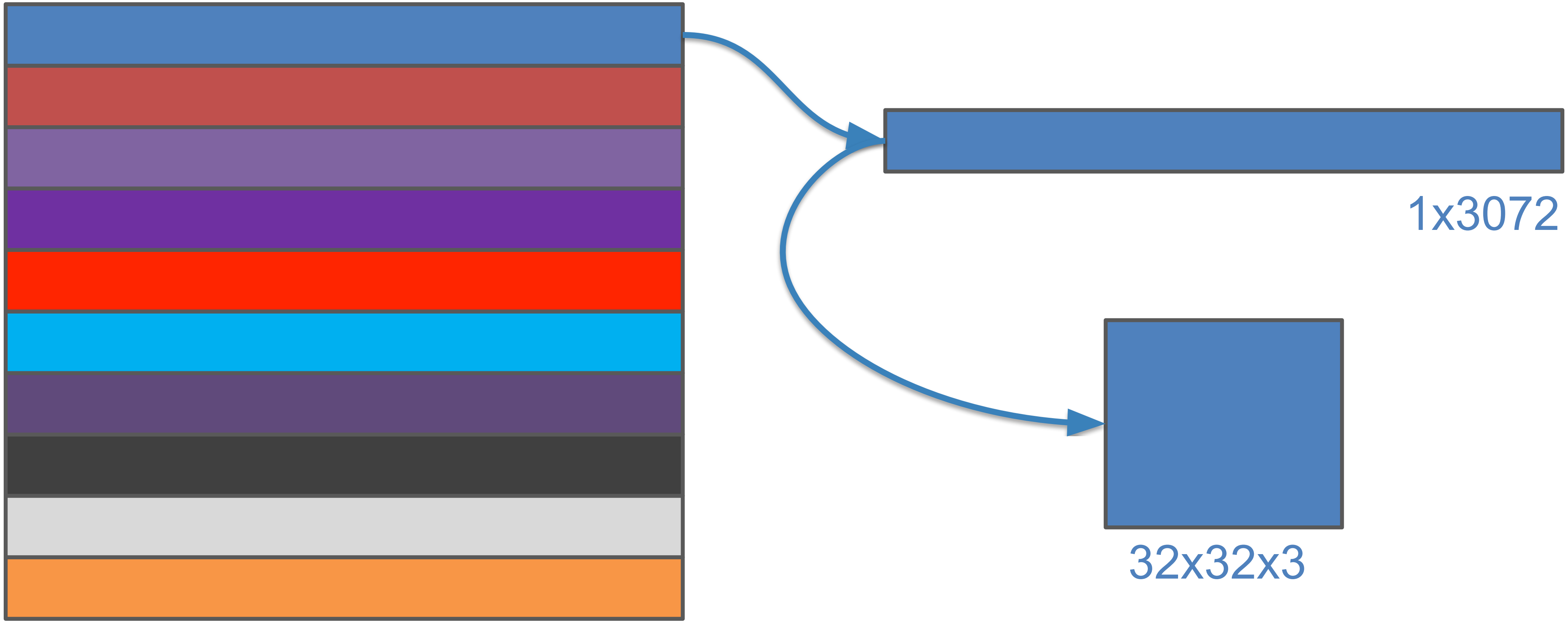
- Assume our weights are trained on the CIFAR 10 dataset with raw pixels:

Interpreting the weights as templates



Interpreting the weights as templates

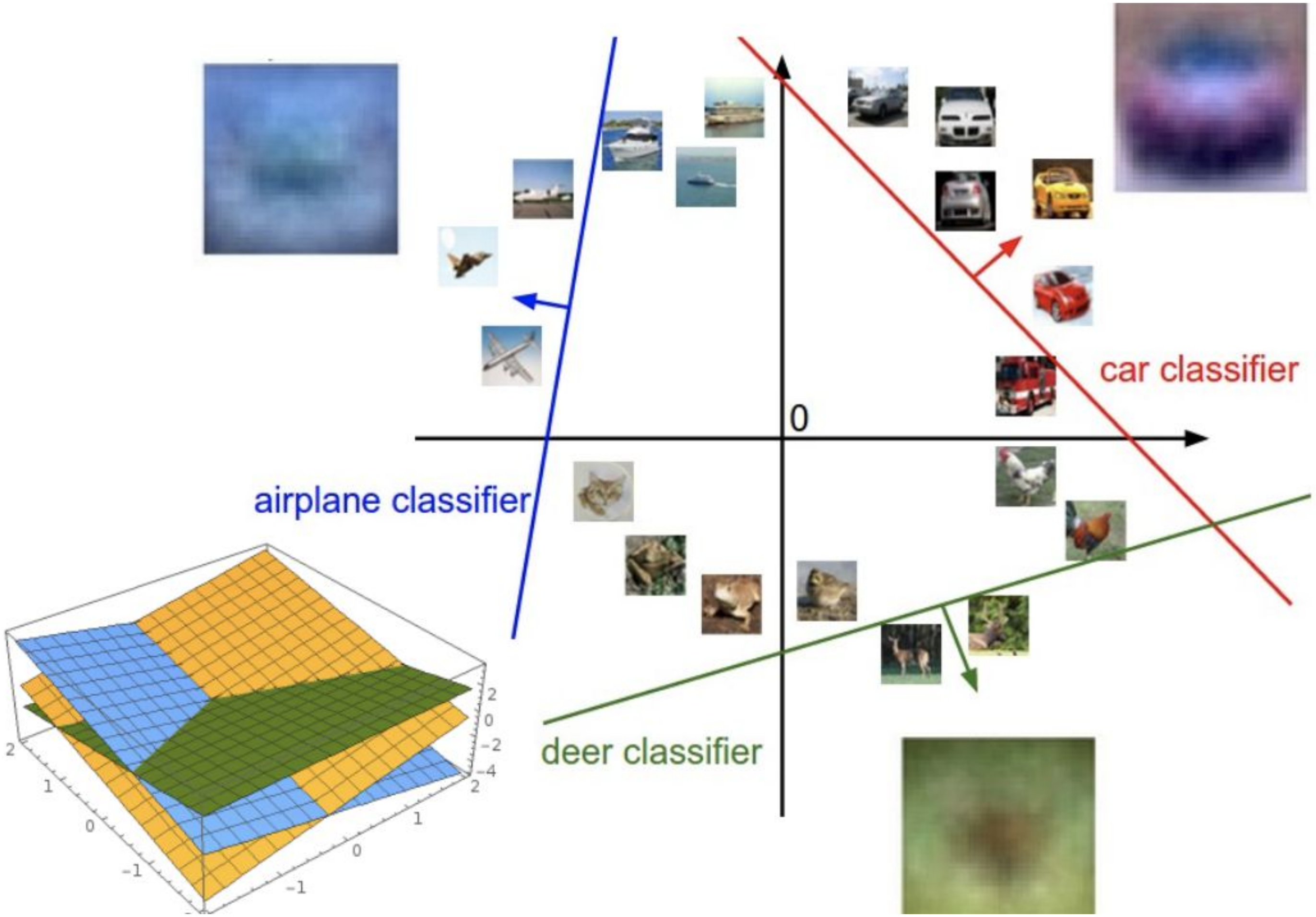
We can reshape the vector back in to the shape of an image



Let's visualize what the templates look like

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

Interpreting the weights geometrically



- Assume the image vectors are in 2D space to make it easier to visualize.

Plot created using Wolfram Cloud

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

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

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

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

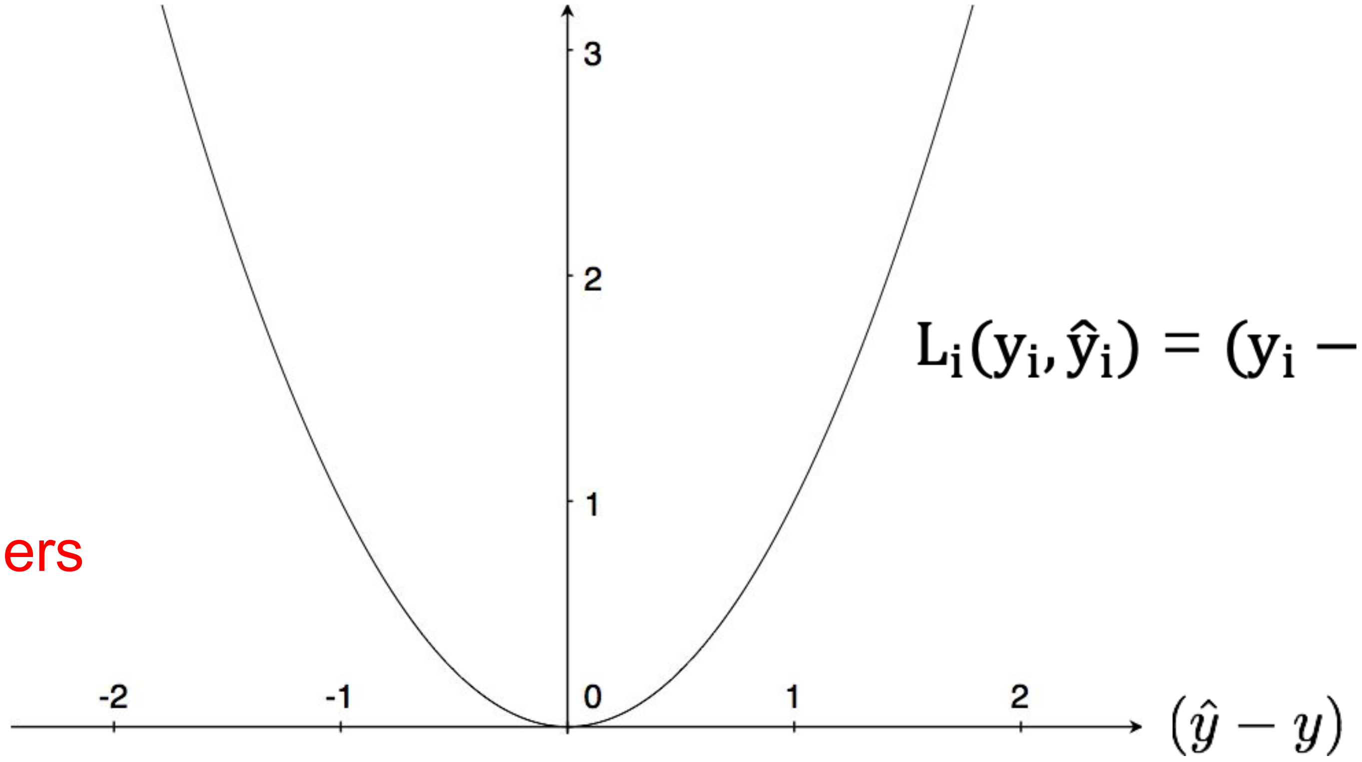
How do we choose the loss function L_i ?

YOU get to chose the loss function!

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

Squared Error (L2)

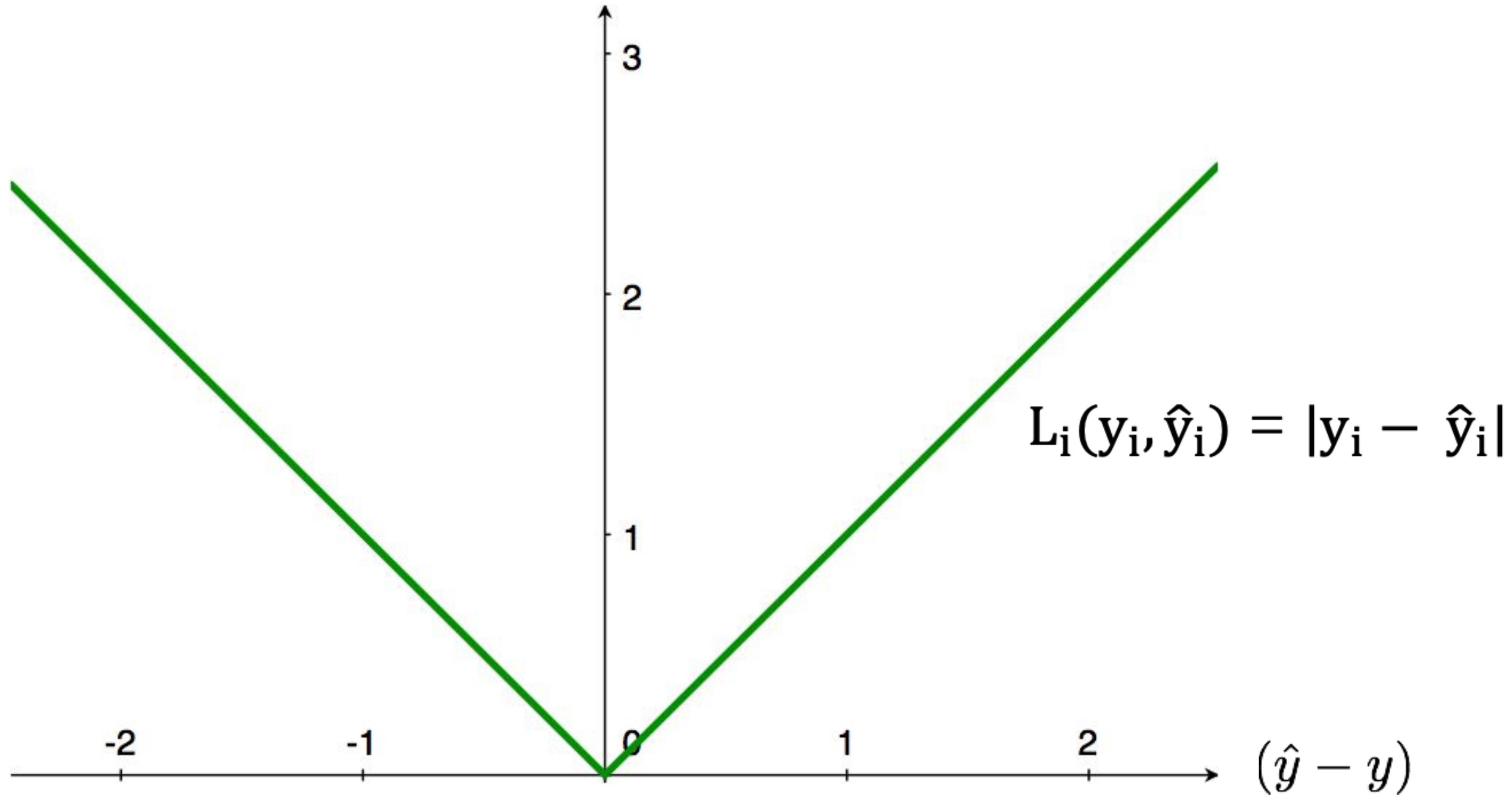
(a popular loss function) ((why?))



$$L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

Not robust to outliers

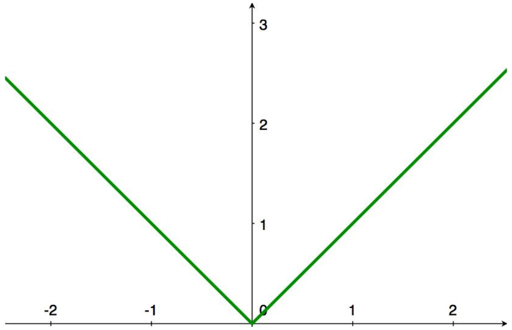
L1 loss



$$L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

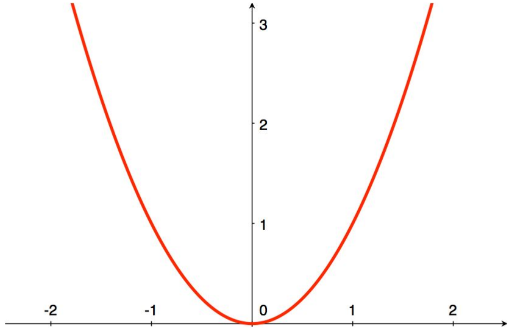
L1 Loss

$$L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$



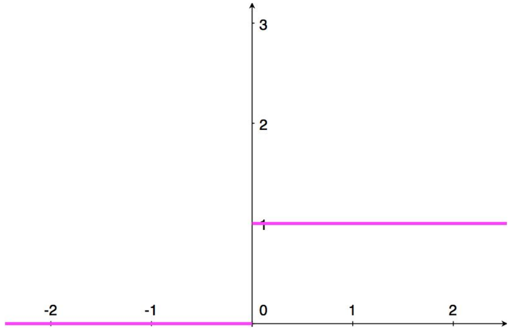
L2 Loss

$$L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$



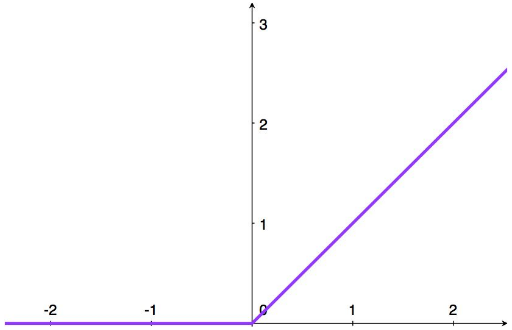
Zero-One Loss

$$L_i(y_i, \hat{y}_i) = 1||y_i \neq \hat{y}_i||$$



Hinge Loss (only if y ranges [0,1])

$$L_i(y_i, \hat{y}_i) = \max(0,\ 1 - y_i\,\hat{y}_i)$$



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:

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

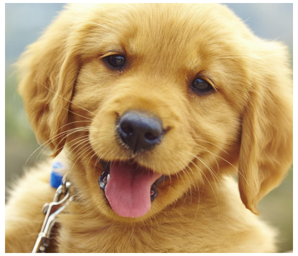
- where *P* is the ground truth distribution and *Q* is the model's output score distribution

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:



$$\boxed{\begin{array}{cl} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \begin{array}{l} \text{dog} \\ \text{cat} \\ \text{bird} \end{array} \end{array}}$$

correct outputs

Softmax Classifier (Multinomial Logistic Regression)

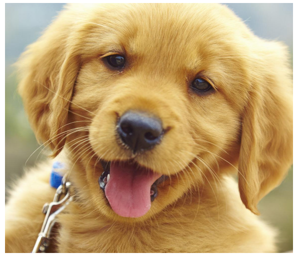
KL divergence:

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

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

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

(It’s also called Cross Entropy)



$$\boxed{\begin{bmatrix}1\\0\\0\end{bmatrix}}\begin{array}{l}\text{dog}\\ \text{cat}\\ \text{bird}\end{array}$$

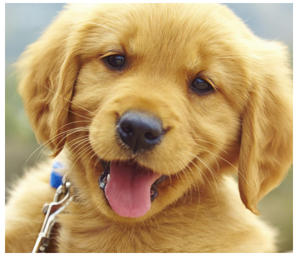
correct outputs

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

$$\begin{array}{cl} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \begin{array}{l} \text{dog} \\ \text{cat} \\ \text{bird} \end{array} \end{array}$$

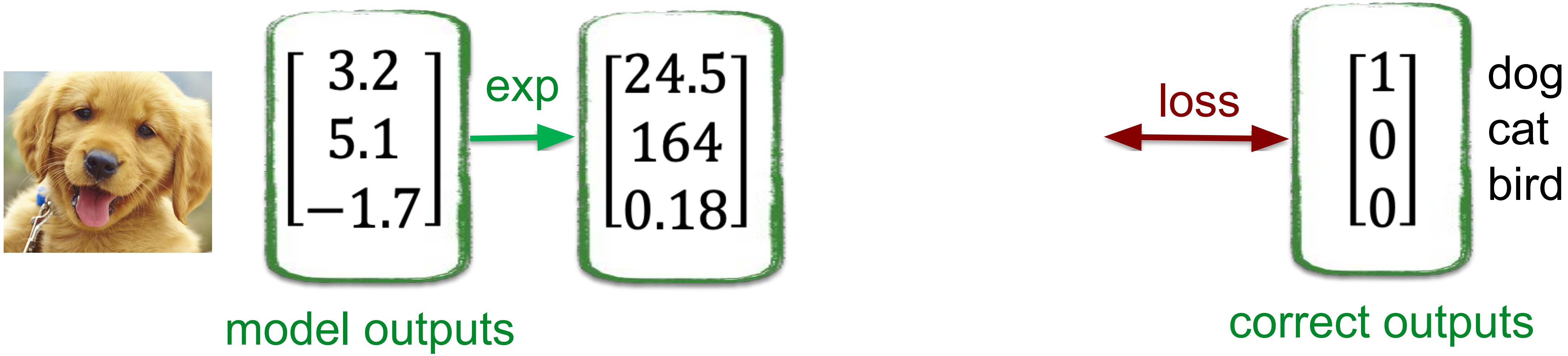
correct outputs

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



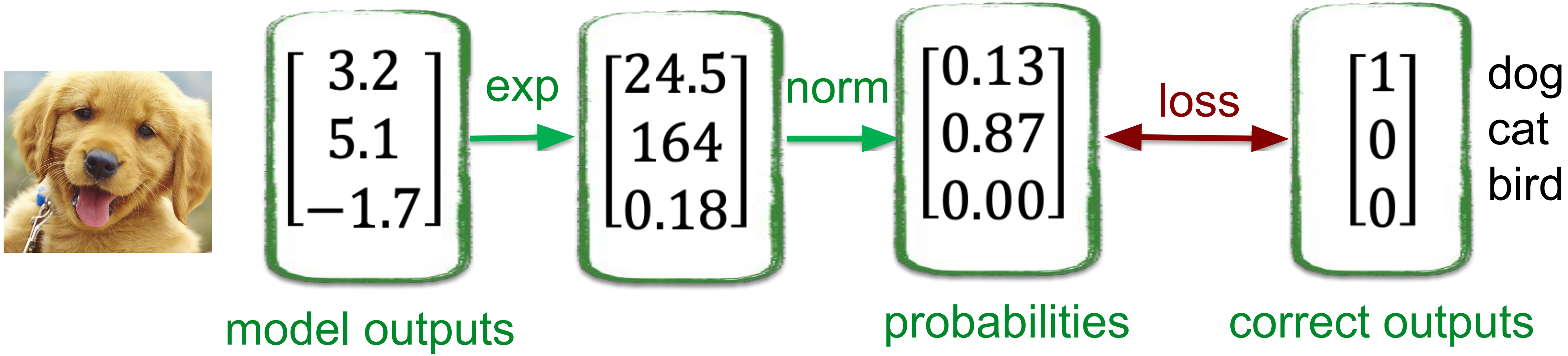
$$\begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix} \xrightarrow{exp} \begin{bmatrix} 24.5 \\ 164 \\ 0.18 \end{bmatrix} \qquad \xleftrightarrow{loss} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

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



$$\begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix} 24.5 \\ 164 \\ 0.18 \end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix} 0.13 \\ 0.87 \\ 0.00 \end{bmatrix} \xleftrightarrow{\text{loss}} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \text{dog} \\ \text{cat} \\ \text{bird} \end{matrix}$$

model outputs

probabilities

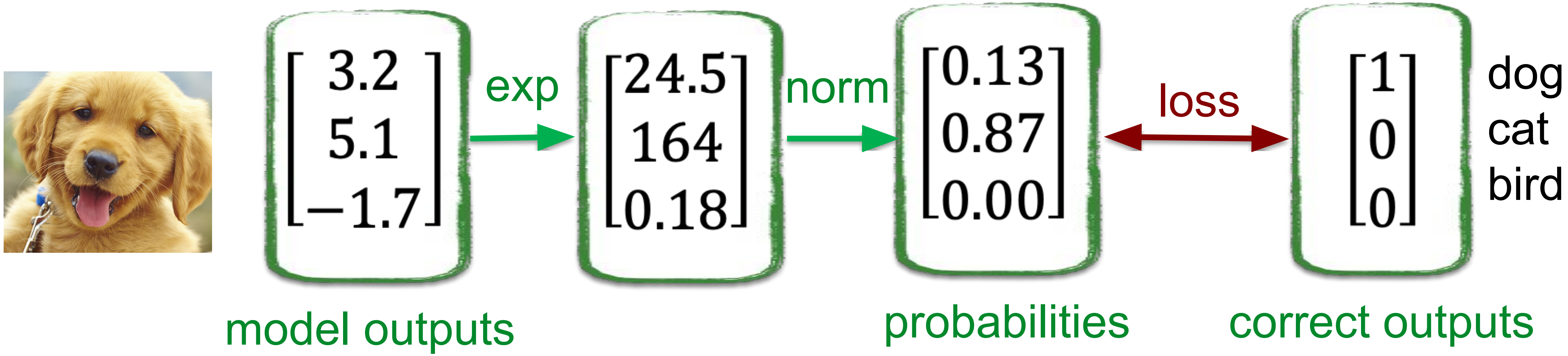
correct outputs

Softmax Classifier (Multinomial Logistic Regression)

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

In this case, what is the loss:

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



$$\begin{bmatrix}3.2\\5.1\\-1.7\end{bmatrix} \xrightarrow{\text{exp}} \begin{bmatrix}24.5\\164\\0.18\end{bmatrix} \xrightarrow{\text{norm}} \begin{bmatrix}0.13\\0.87\\0.00\end{bmatrix} \xleftrightarrow{\text{loss}} \begin{array}{cc}\begin{bmatrix}1\\0\\0\end{bmatrix} & \begin{array}{l}\text{dog}\\\text{cat}\\\text{bird}\end{array}\end{array}$$

model outputs

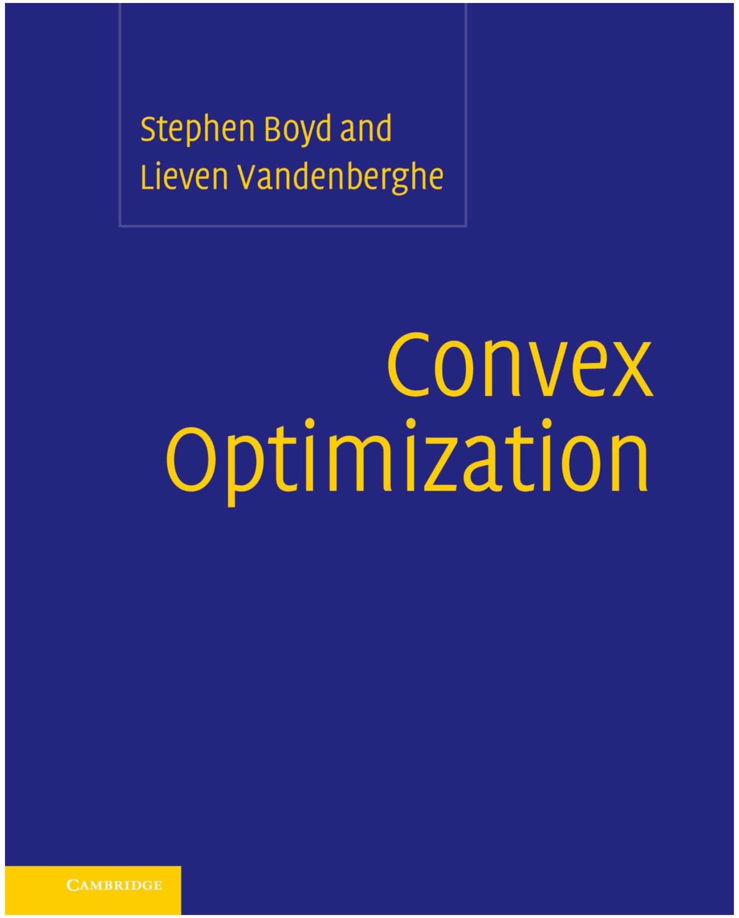
probabilities

correct outputs

Today's agenda

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

How do we find the weights that minimize the loss?

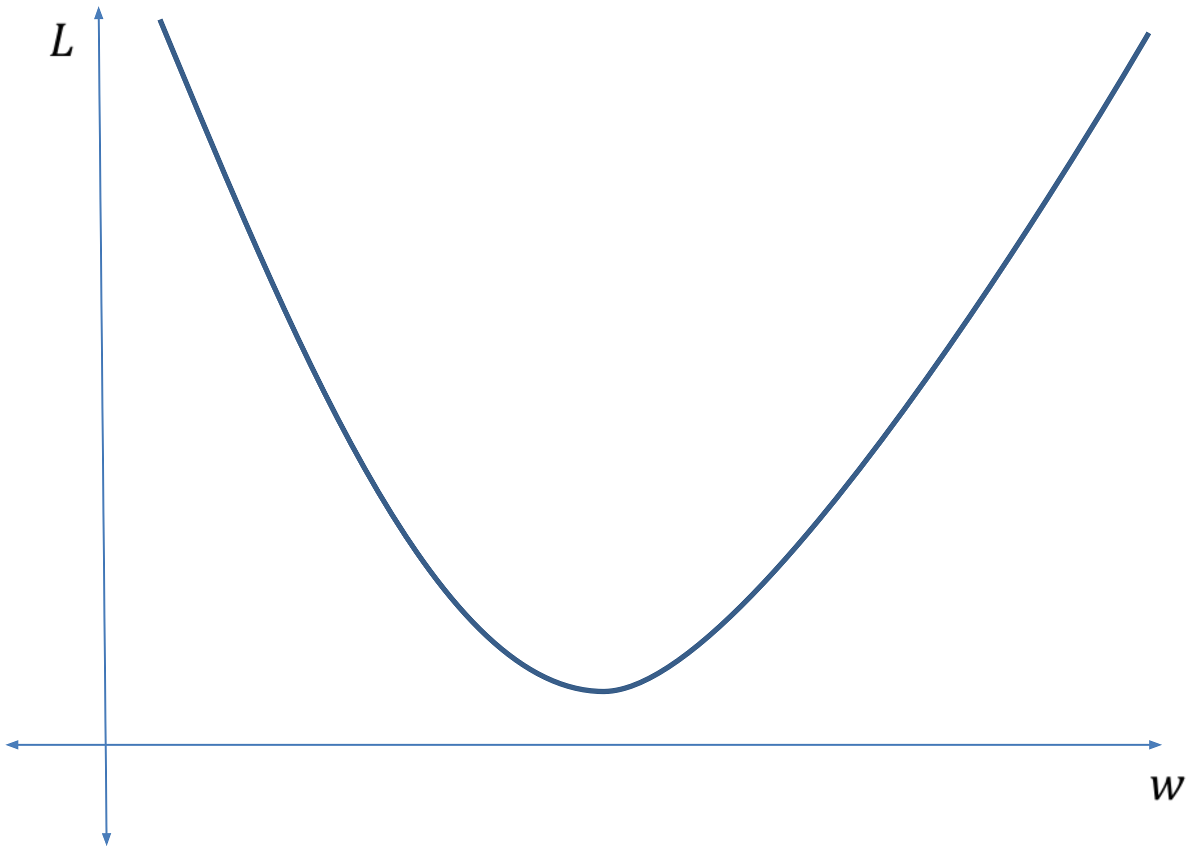


Stephen Boyd and
Lieven Vandenberghe

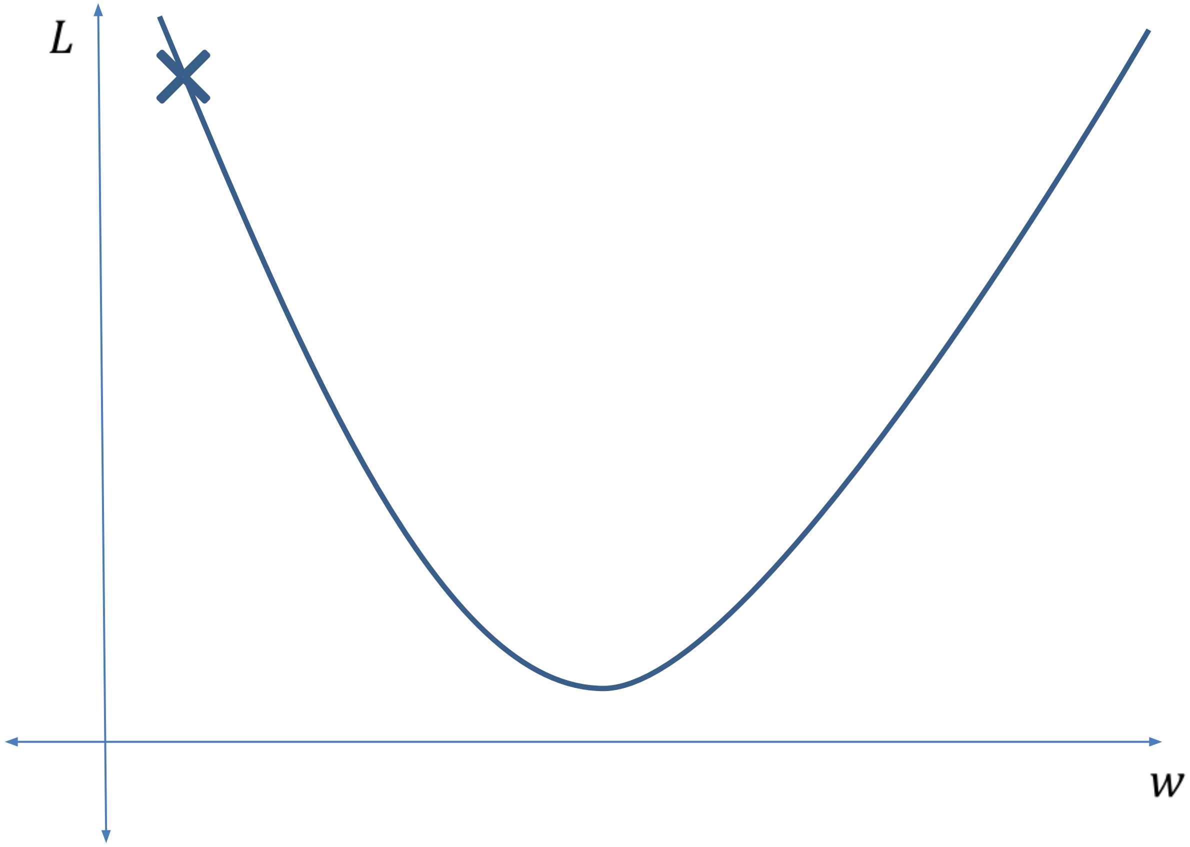
Convex
Optimization

CAMBRIDGE

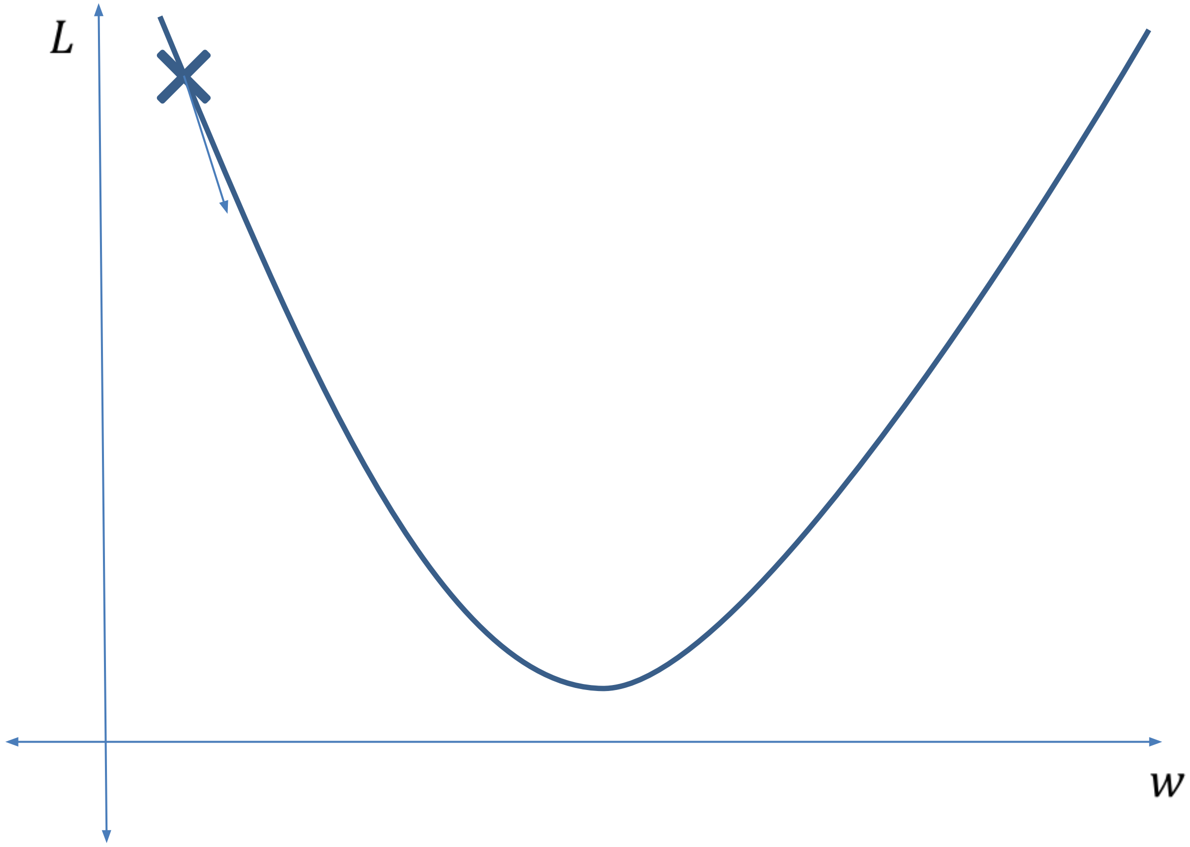
Gradient descent visualized: Minimizing loss



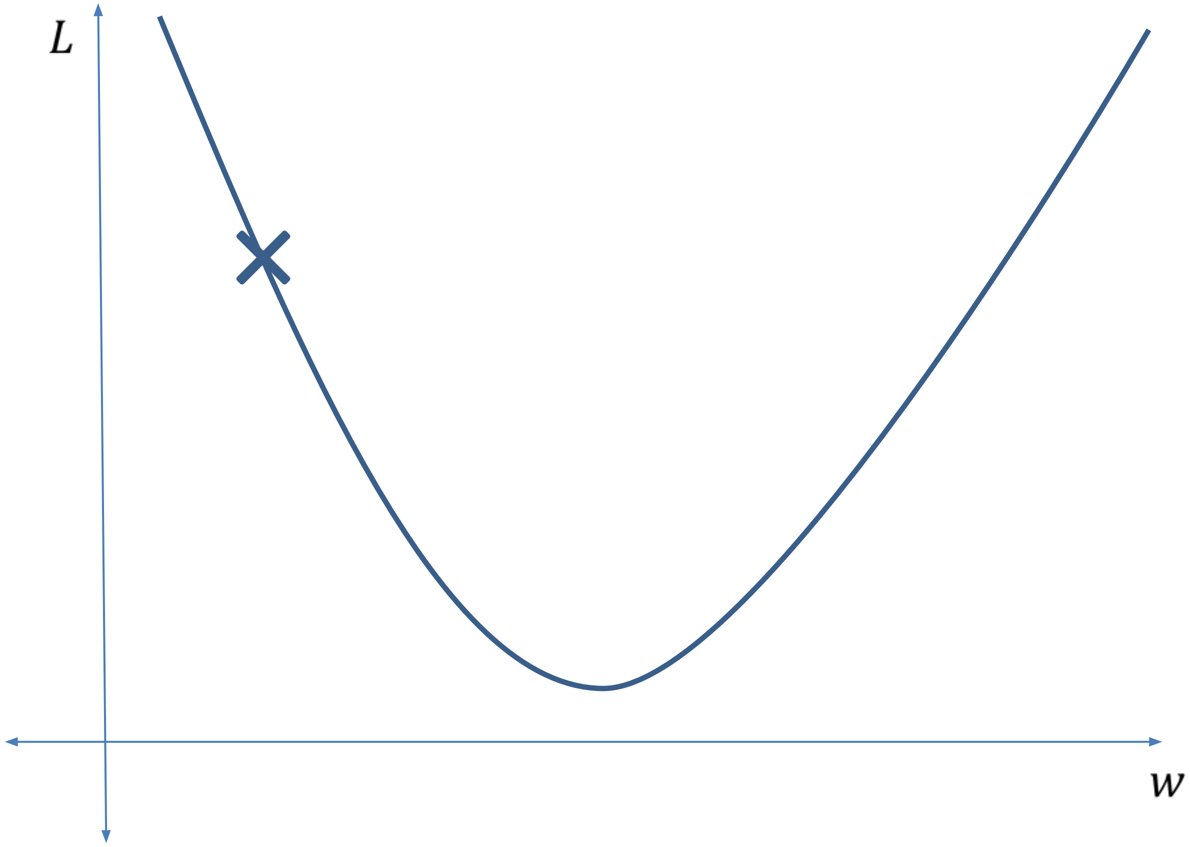
Gradient descent visualized: Minimizing loss



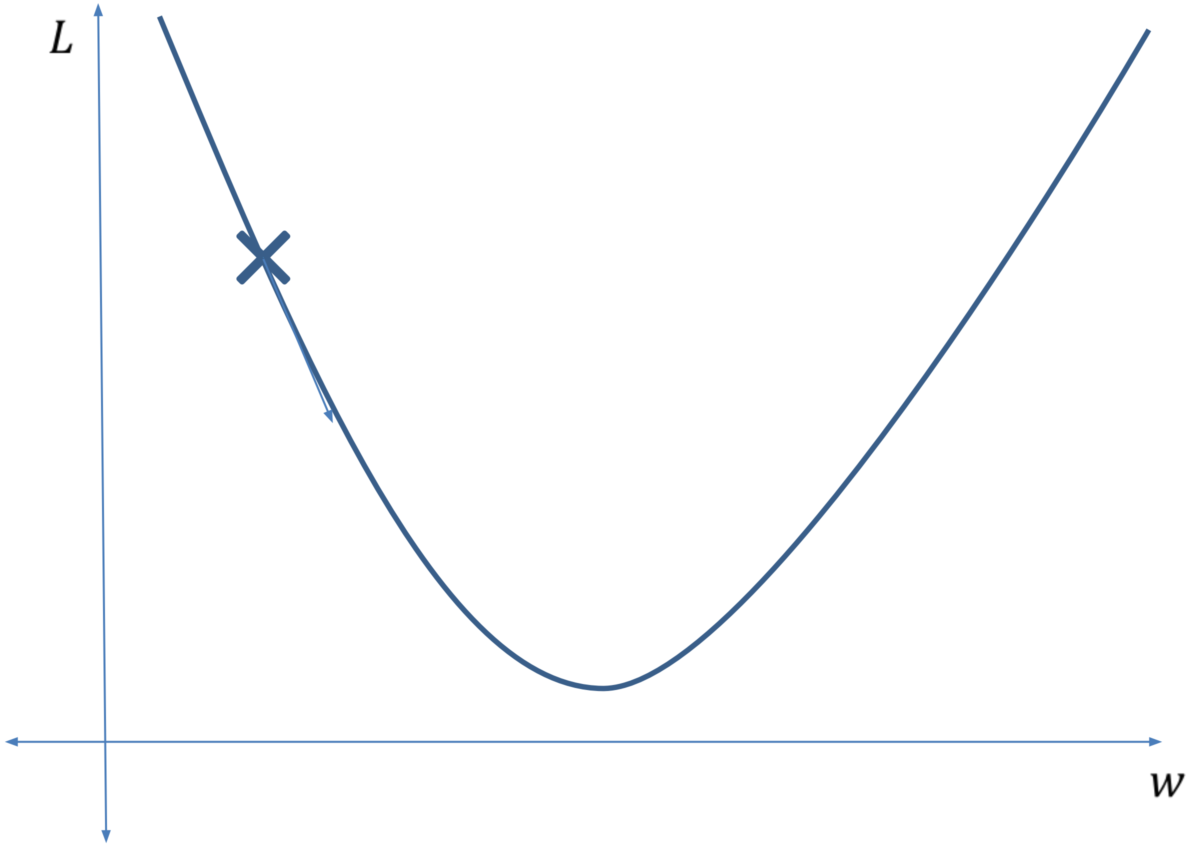
Gradient descent visualized: Minimizing loss



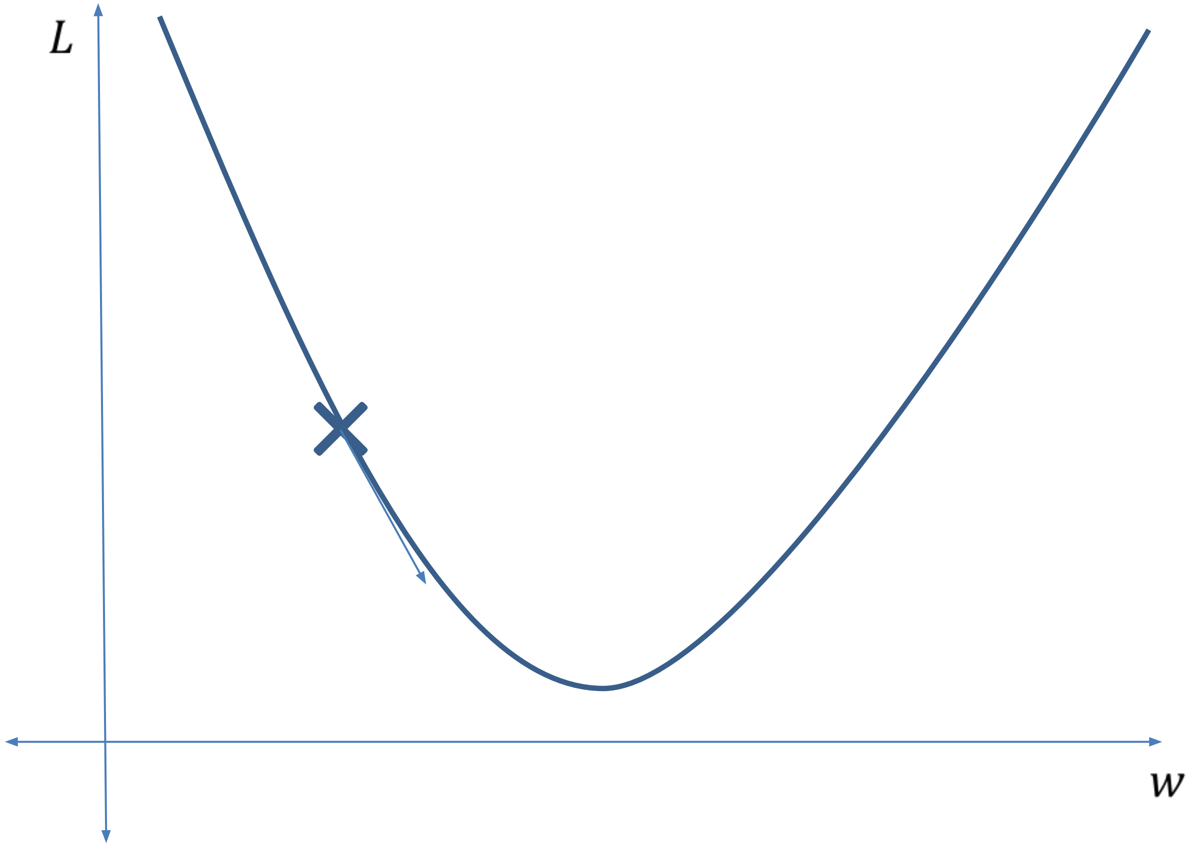
Gradient descent visualized: Minimizing loss



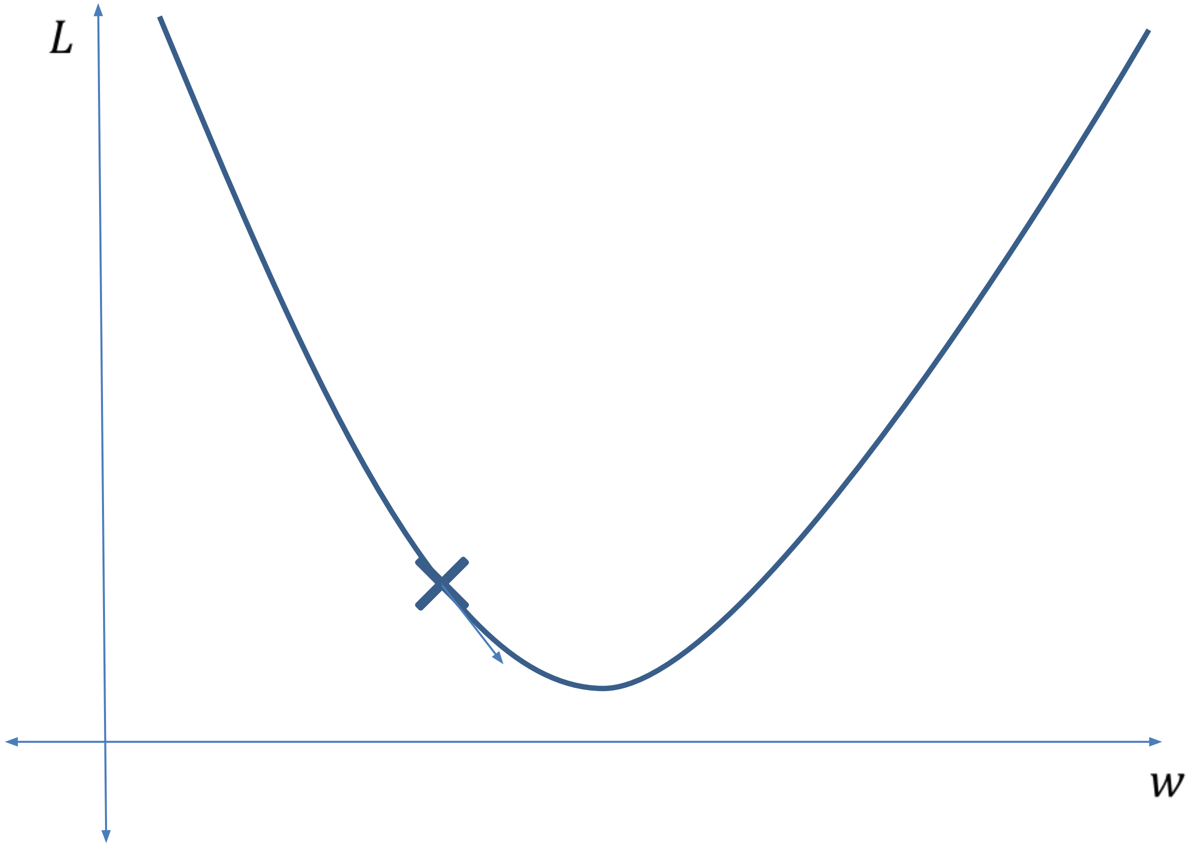
Gradient descent visualized: Minimizing loss



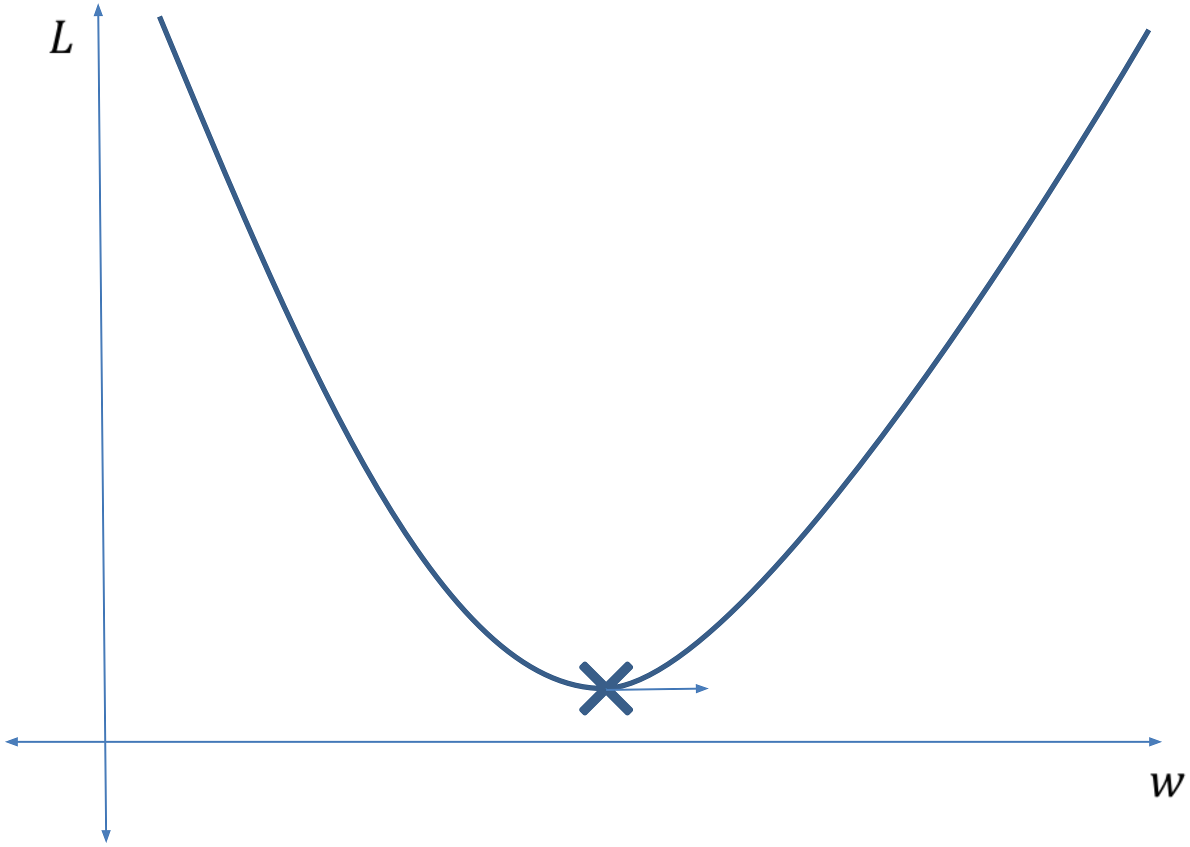
Gradient descent visualized: Minimizing loss



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:
        $\hat{y}_i = f(x_i, W)$
        $L \mathrel{+}= L_i(y_i, \hat{y}_i)$
    $\frac{dL}{dW} = ???$
    $W := W - \alpha \frac{dL}{dW}$
```

Small step x Gradient

Gradient Descent Pseudocode

```
for _ in {0,…,num_epochs}:
    $L = 0$
    for $x_i$, $y_i$ in data:
        $\hat{y}_i = f(x_i, W)$
        $L += L_i(y_i, \hat{y}_i)$
    $\frac{dL}{dW} = ???$
    $W := W - \alpha \frac{dL}{dW}$
```

Small step x Gradient

Exercise on linear classification:

$$\hat{y} = Wx$$

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

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

Partial derivative of loss to update weights

Given training data point (***x***, ***y***), the linear classifier formula is: $\hat{y} = Wx$

Let's assume that the correct label is class ***k***, implying ***y=k***

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

Now, we want to update the weights ***W*** by calculating the direction in which to change the weights to reduce the loss: $\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}$

we know that $\frac{d\hat{y}}{dW} = x$, but what about $\frac{dL}{d\hat{y}}$?

Partial derivative of loss to update weights

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

To calculate $\frac{dL}{d\hat{y}}$, we need to consider two cases:

Case 1:

$$\frac{dL}{d\hat{y}_k} = -1 + \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

Case 2:

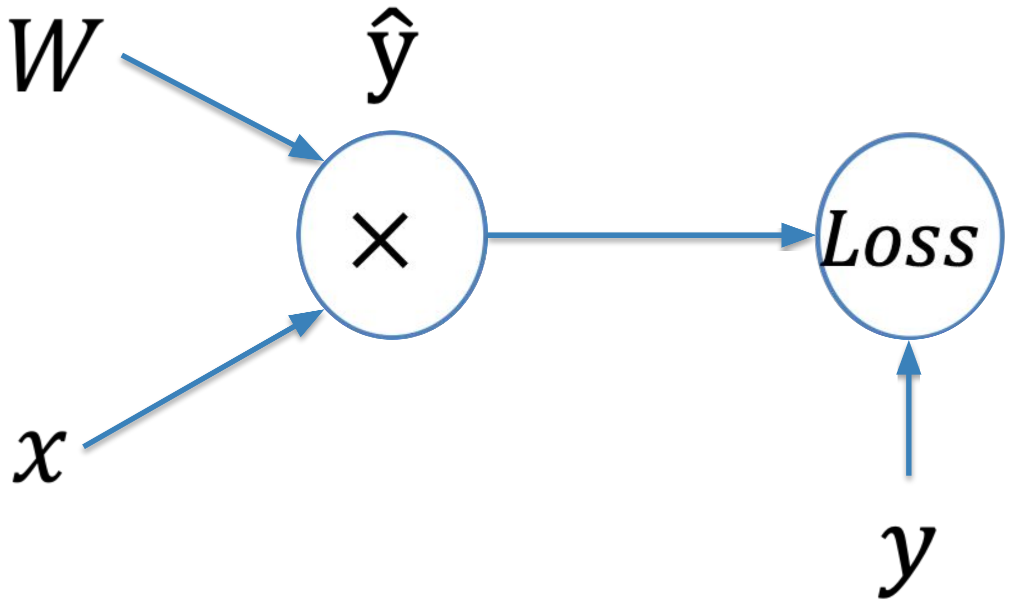
$$\frac{dL}{d\hat{y}_{l \neq k}} = \frac{e^{\hat{y}_l}}{\sum_j e^{\hat{y}_j}}$$

Exercise: Partial derivative of loss to update weights

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

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

Backprop – another way of computing gradients

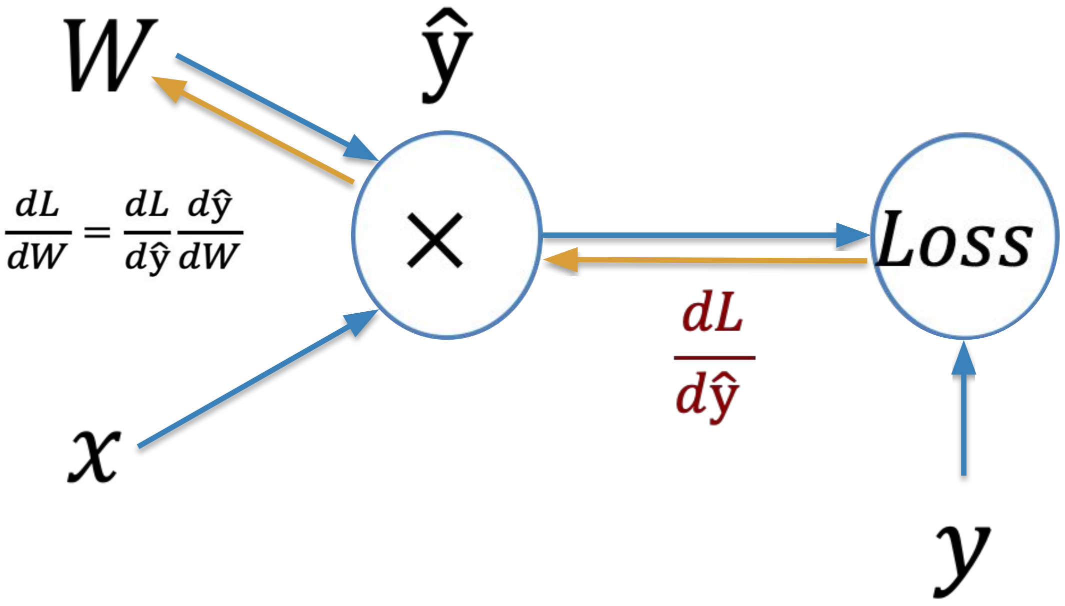


$$\hat{y} = Wx$$
$$L = Loss(\hat{y}, y)$$

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

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

Backprop – a way of computing gradients



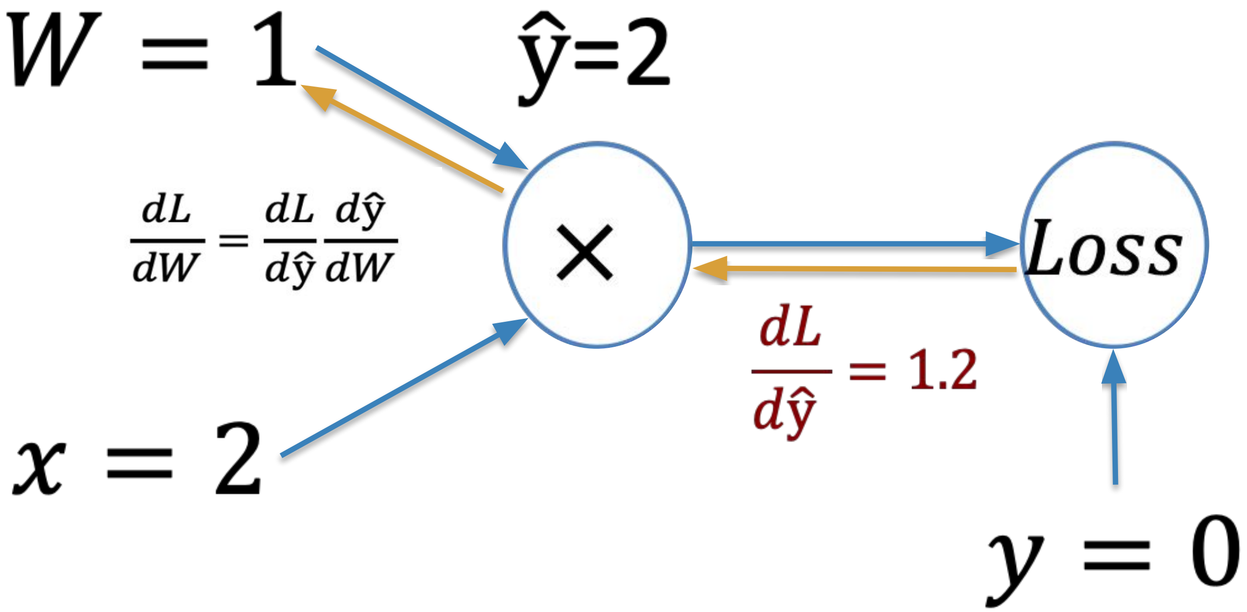
$$\hat{y} = Wx$$
$$L = Loss(\hat{y}, y)$$

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

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

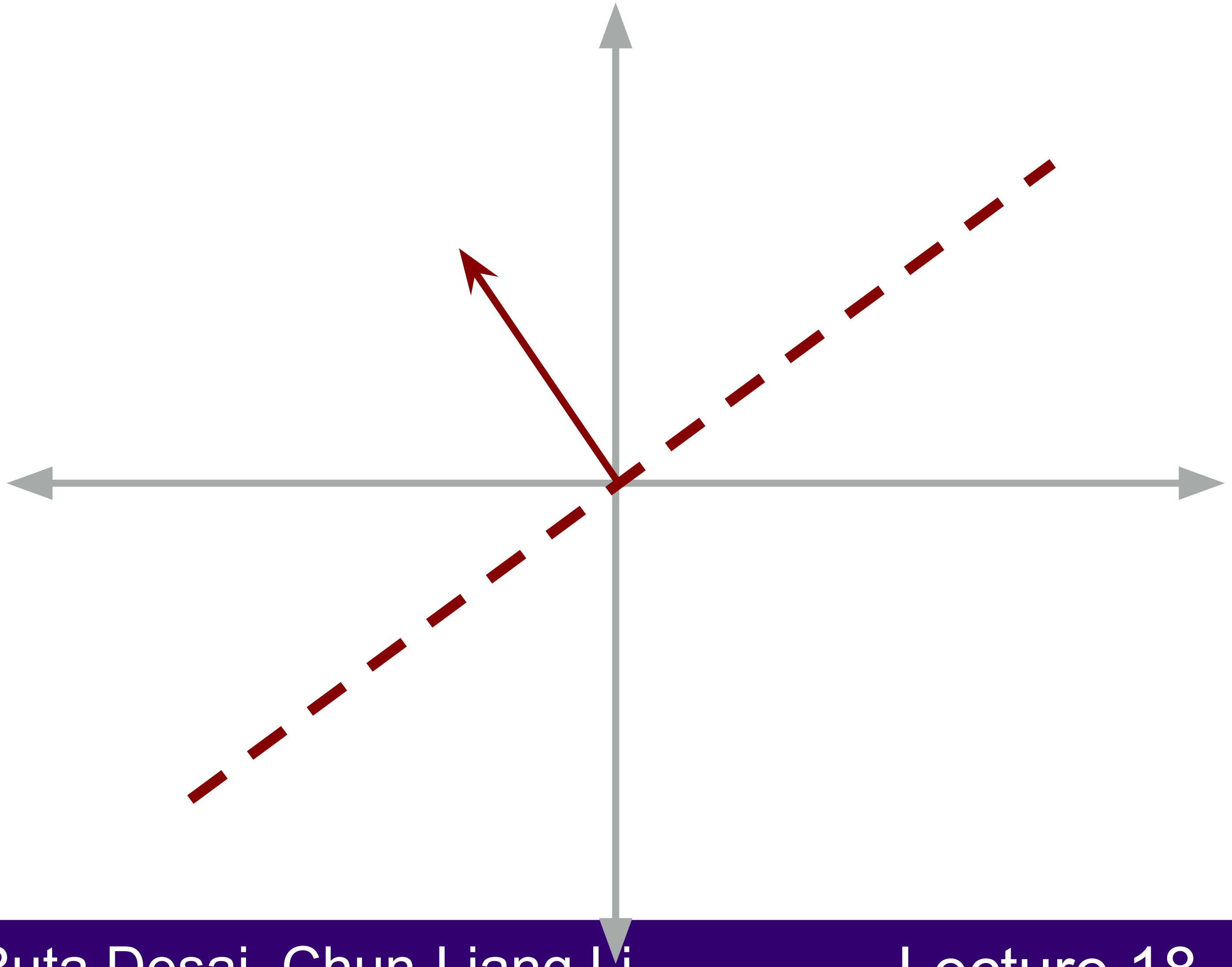
Backprop example in 1D:



We know the chain rule

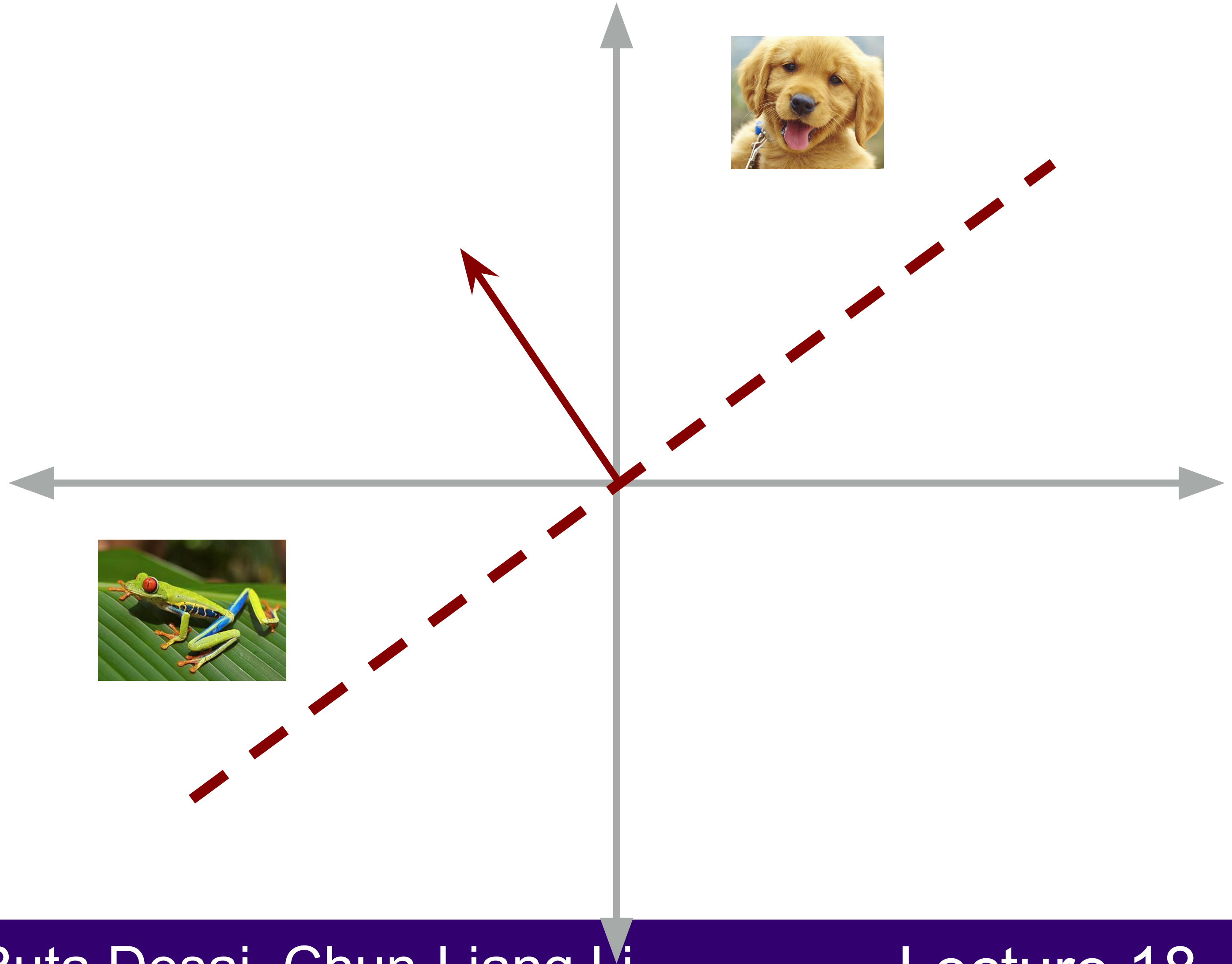
$$\frac{dL}{dW} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW}$$
$$= \frac{dL}{d\hat{y}}x$$
$$= 1.2x$$
$$= 1.2 \times 2$$
$$= 2.4$$

Interpreting the weights geometrically



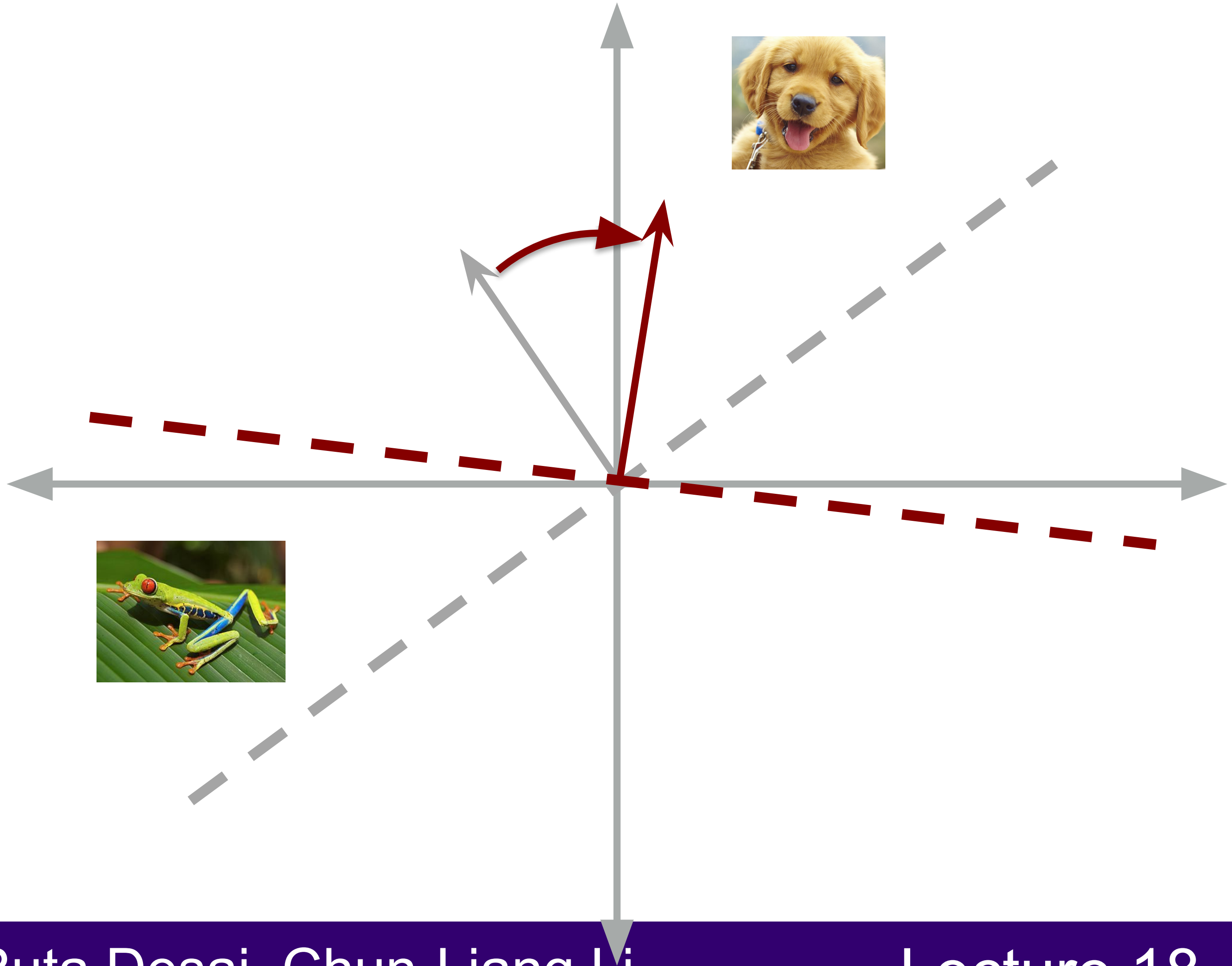
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly

Interpreting the weights geometrically



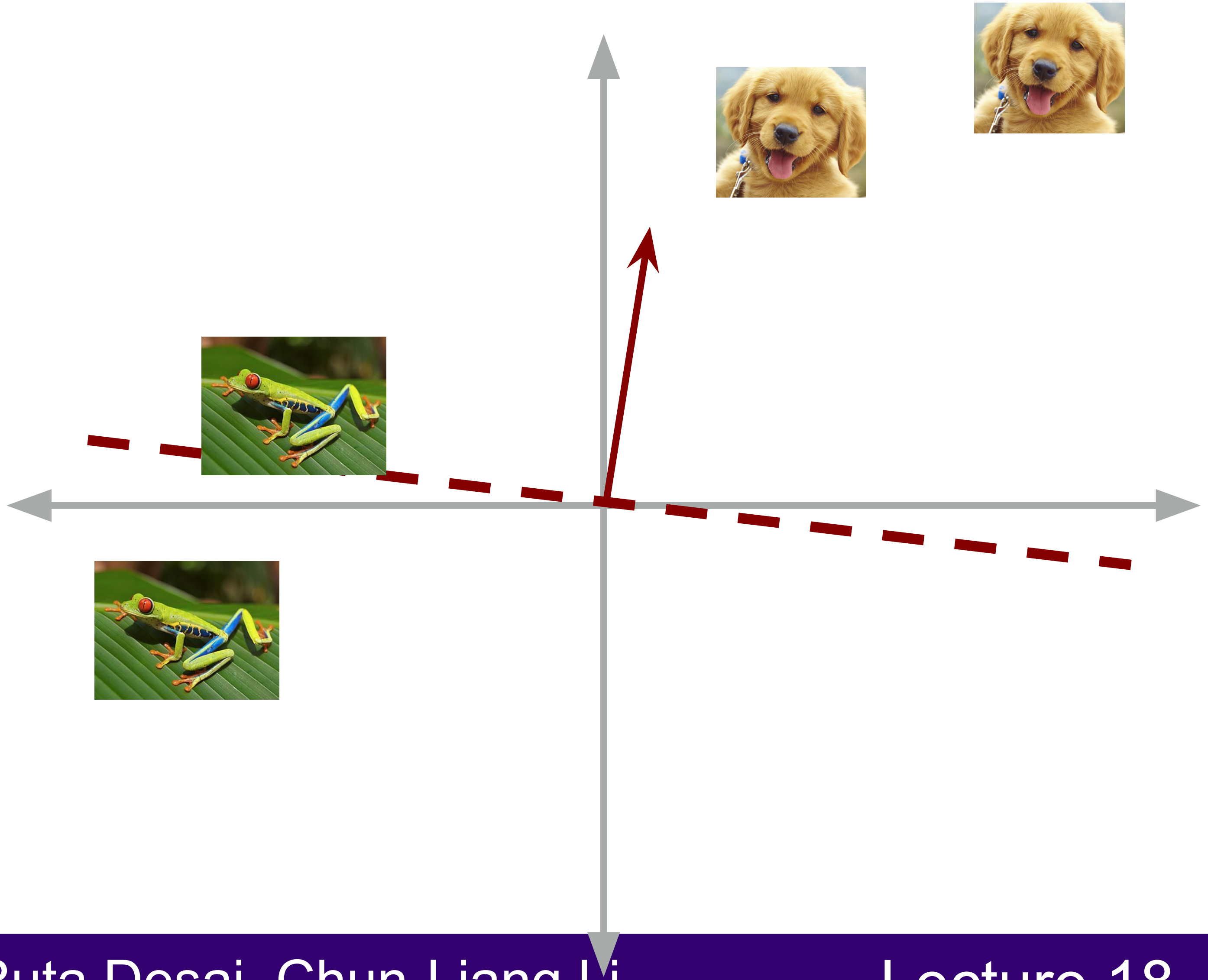
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two data points

Interpreting the weights geometrically



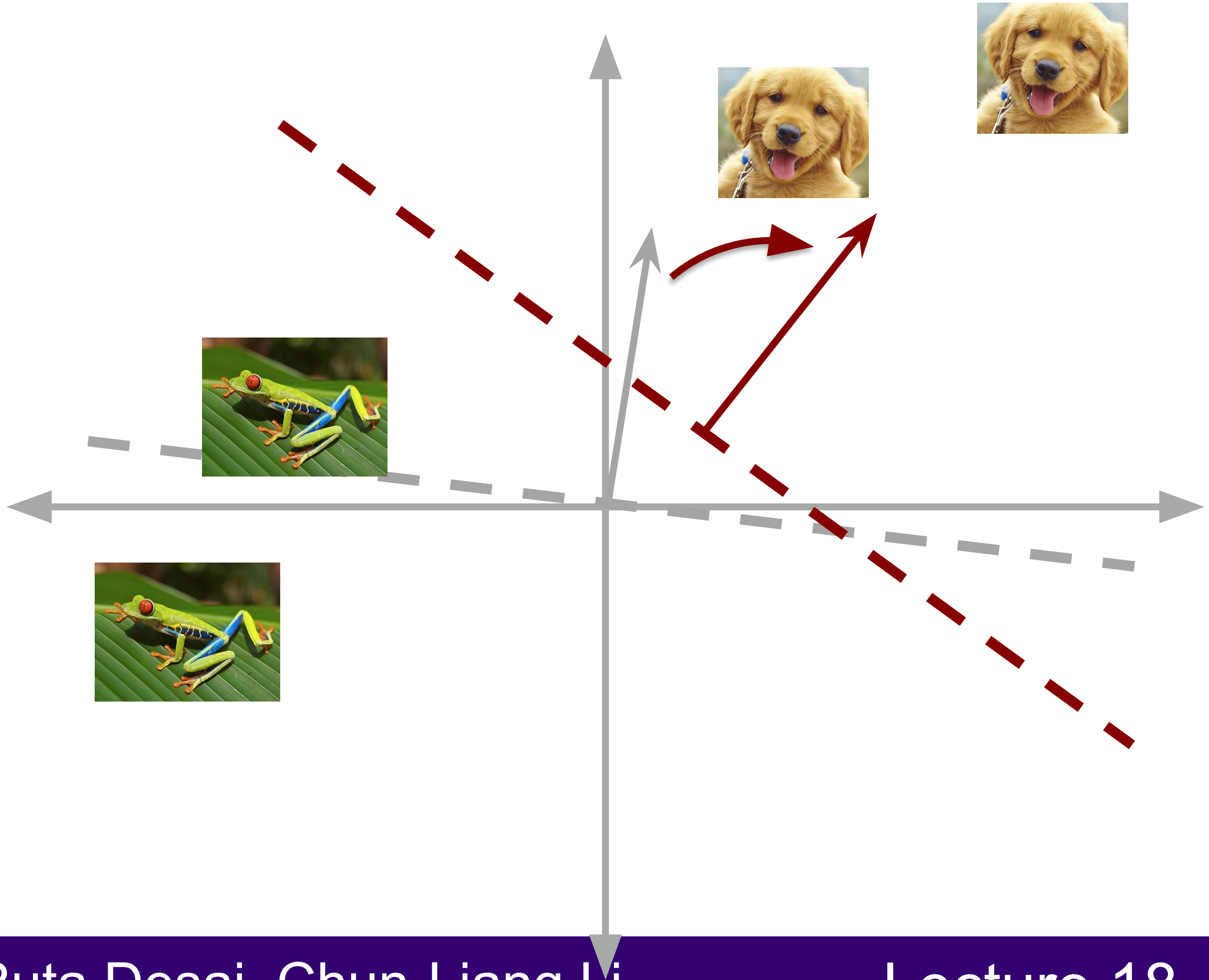
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two data points
- Update the weights

Interpreting the weights geometrically



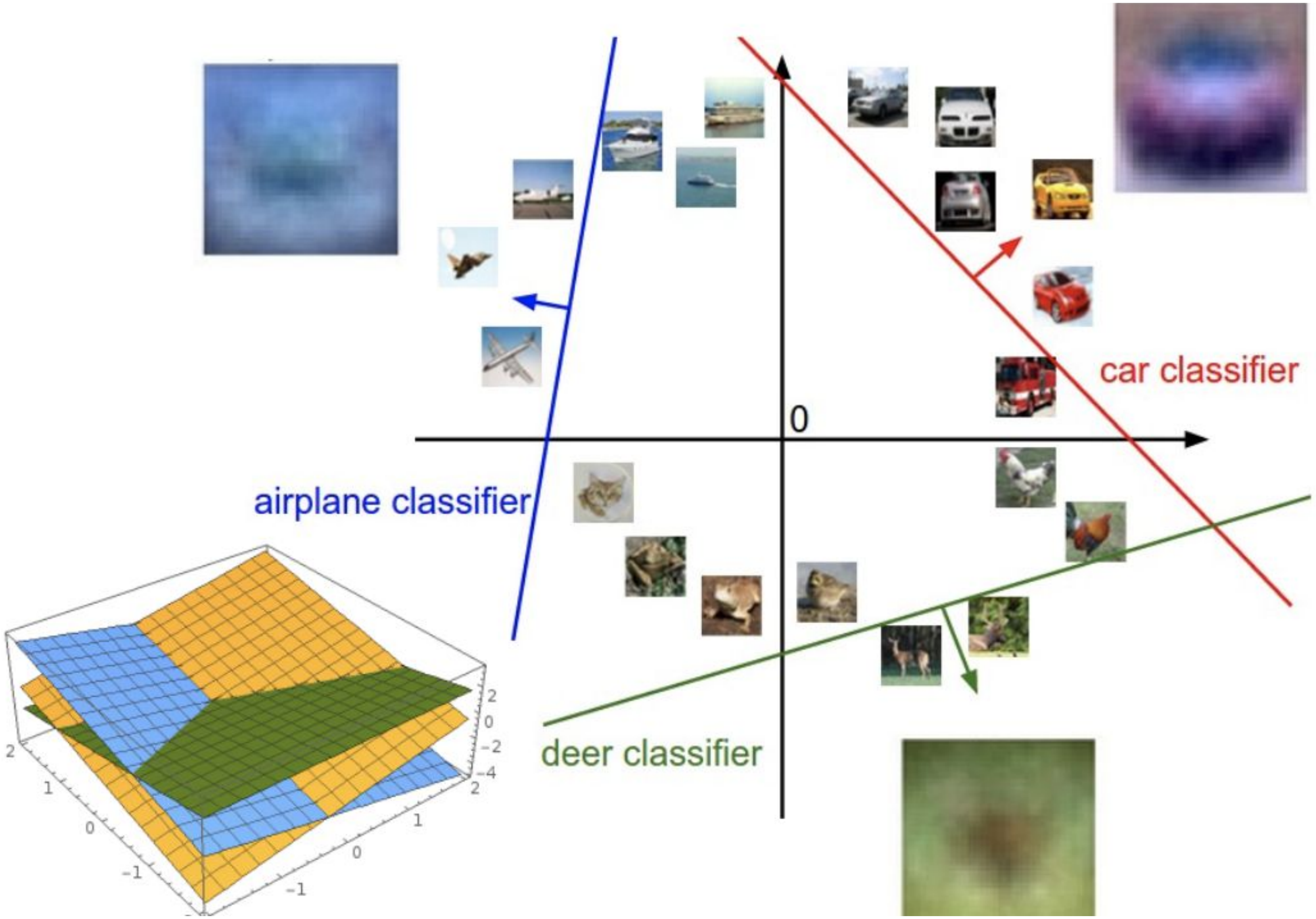
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two more data points
- Update the weights

Interpreting the weights geometrically



- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two more data points
- Update the weights

Interpreting the weights geometrically



Plot Created using [Wolfram Cloud]