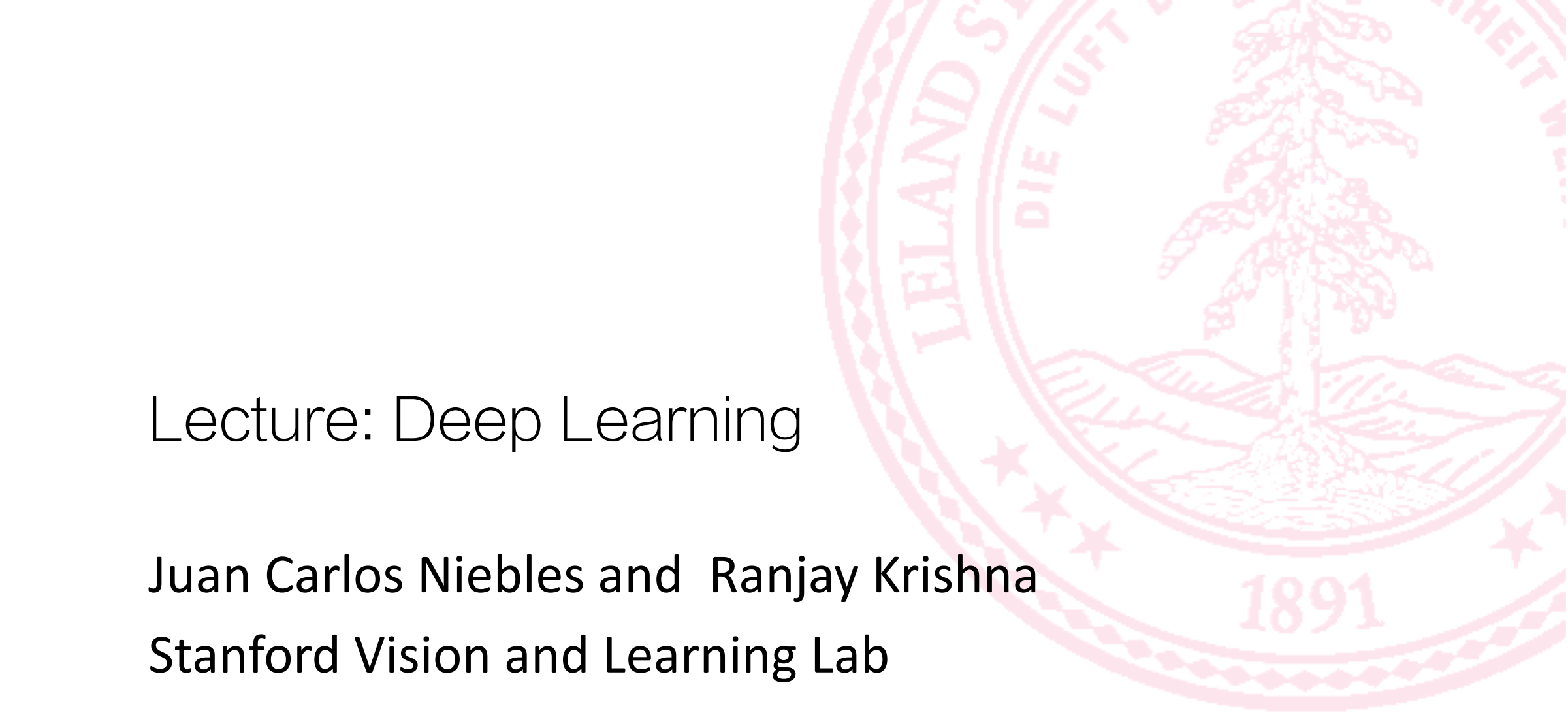


Lecture: Deep Learning

Juan Carlos Niebles and Ranjay Krishna

Stanford Vision and Learning Lab

Announcements

- Exam next week
  - Hewlett Teaching Center, Room 200
  - December 10, 3:30 to 6:30pm
  - Practice exam + solutions released on piazza
- HW8
  - due tomorrow
- Extra credit
  - Writing lecture notes (especially for this week's lectures)
  - Explanations on how to write and submit notes is available on the webpage

CS 131 Roadmap

Pixels	Segments	Images	Videos	Web
Convolutions Edges Descriptors	Resizing Segmentation Clustering	Recognition Detection Ontology	Motion Tracking	Neural networks Convolutional neural networks

Today's agenda

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

Today's agenda

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- 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)

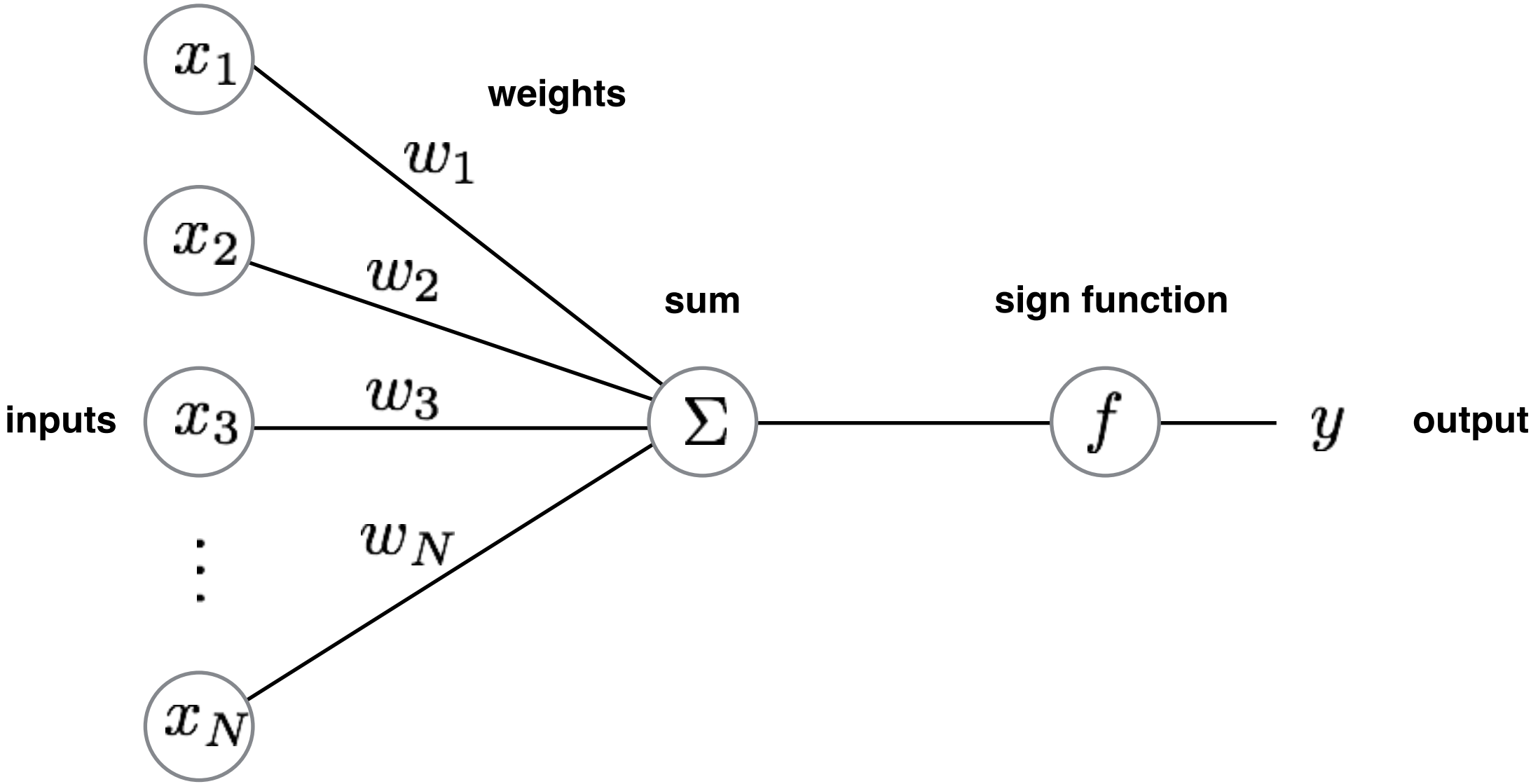
1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

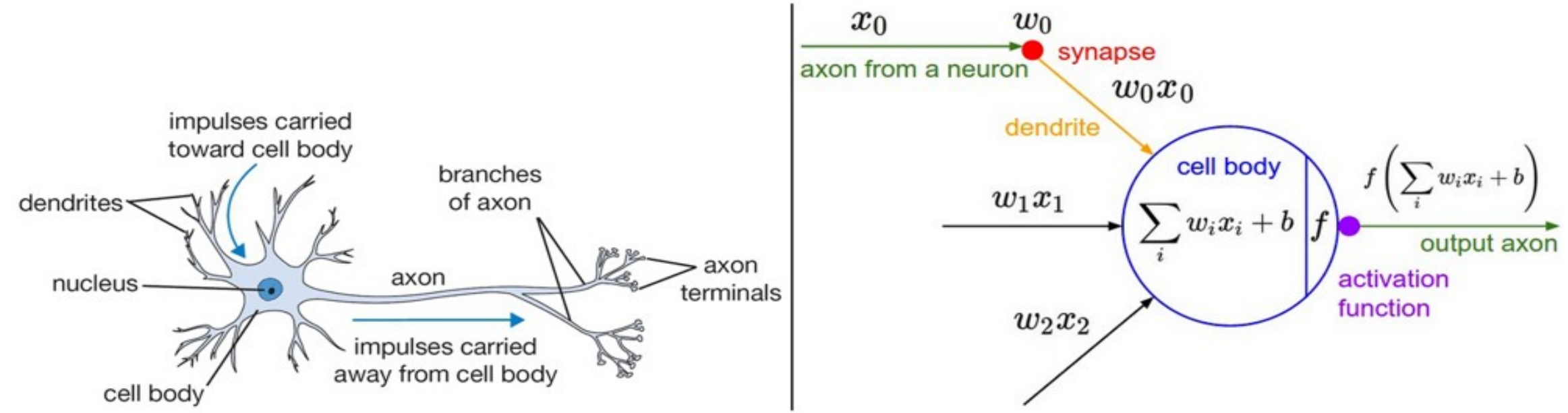
2010s Age of the Deep Network

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

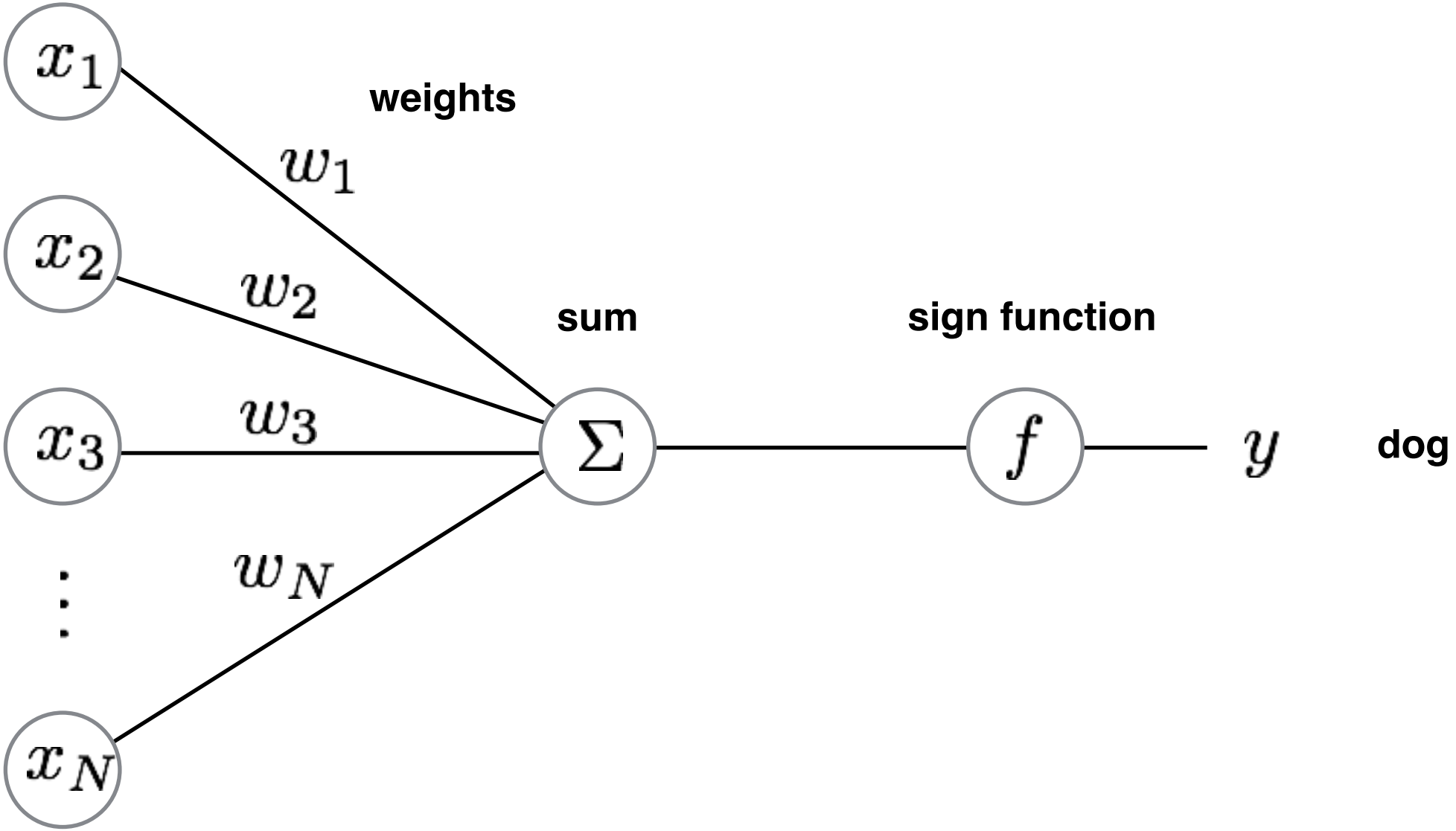


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

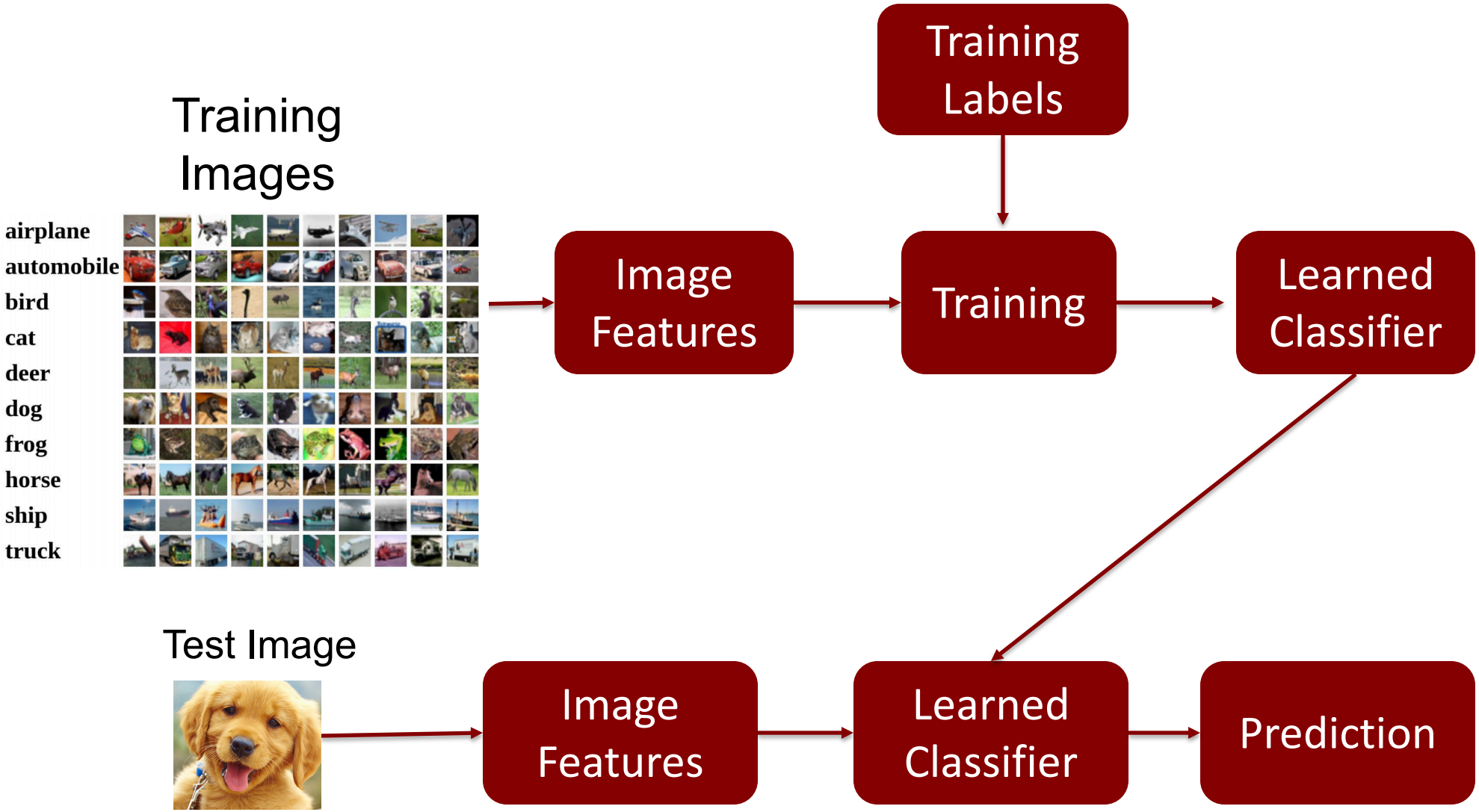
Neural nets/perceptrons are loosely inspired by biology.

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

Perceptron: for image classification



Recall: image classification pipeline



Recall: we can featurize images into a vector

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

Image
Vector

Recall: we can featurize images into a vector

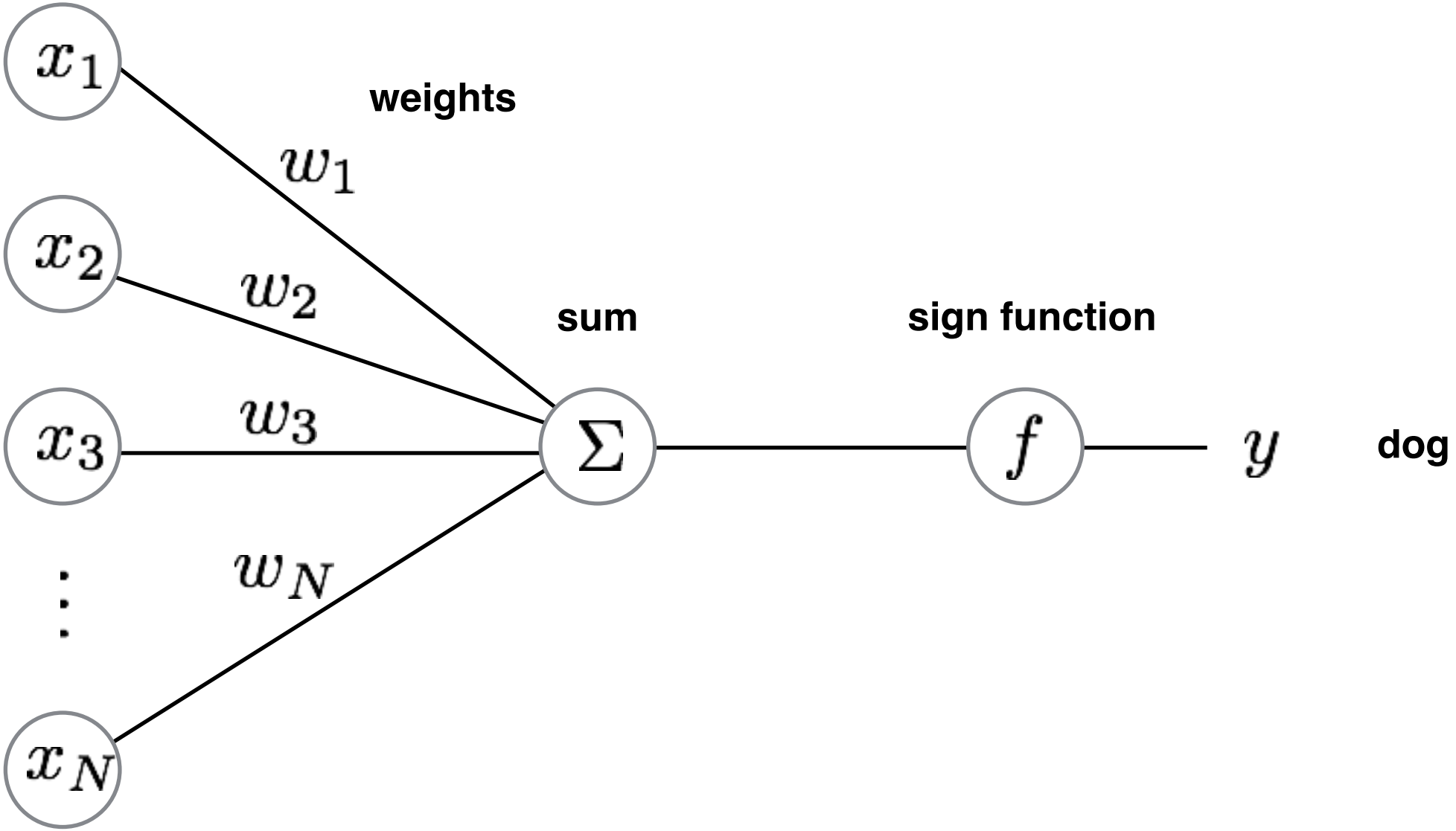
Raw pixels
Raw pixels + (x,y)
PCA
LDA
BoW
BoW + spatial pyramids

Image Vector

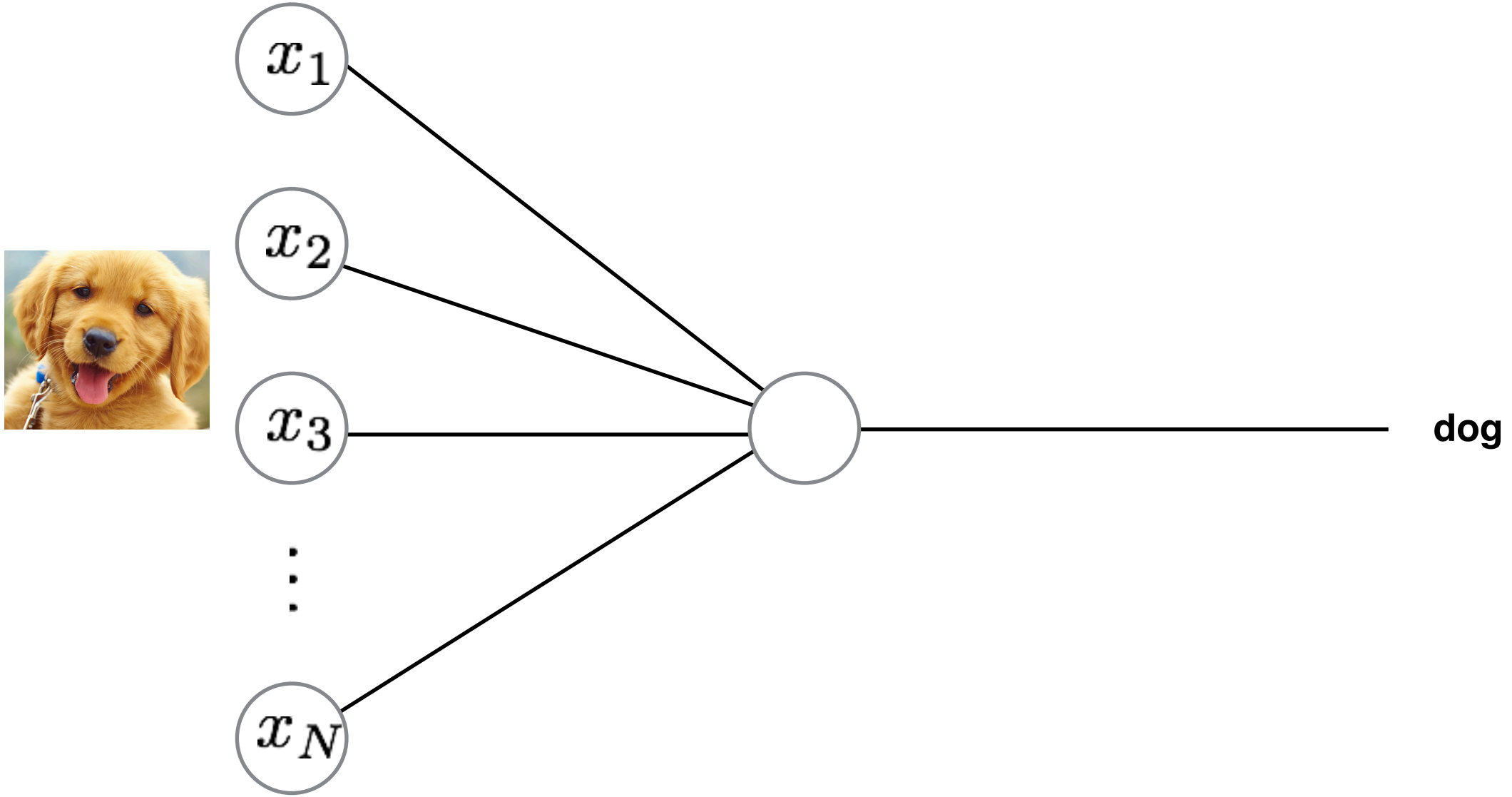
x_1
x_2
x_3
...
...
...
...
...
...
...
...
...
...

x_n

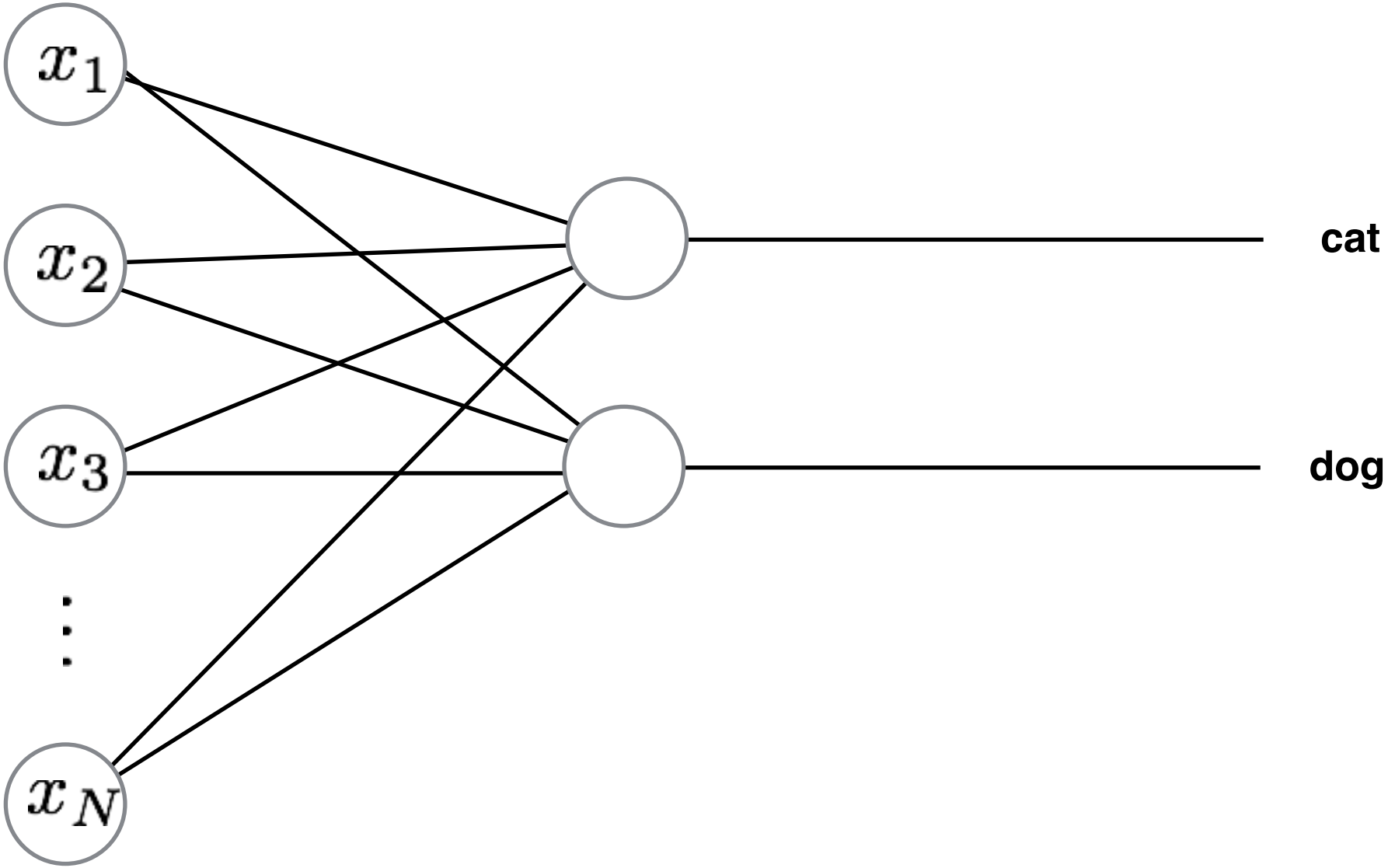
Perceptron: for image classification



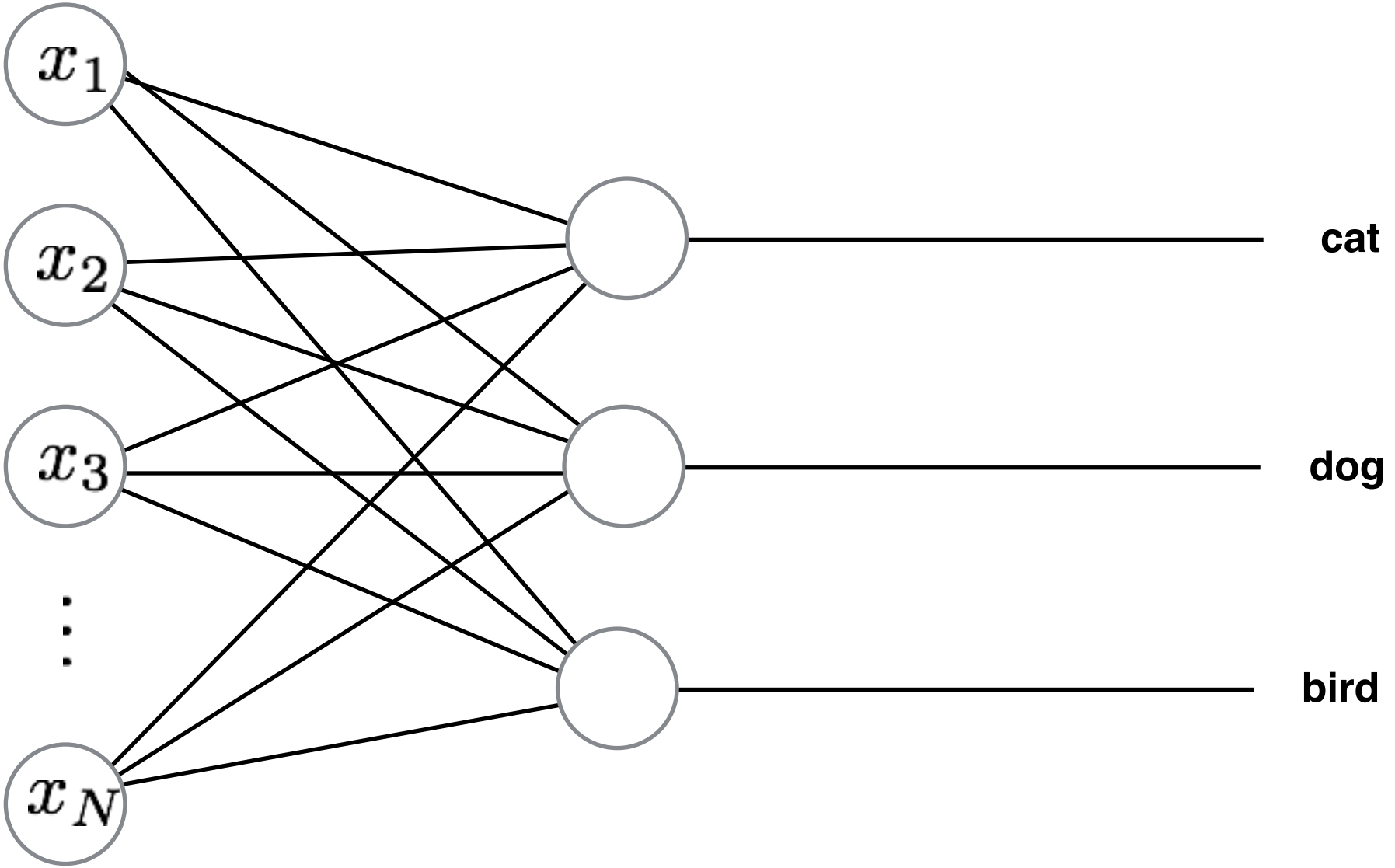
Perceptron: simplified view with one perceptron



Perceptron: simplified view with two perceptron



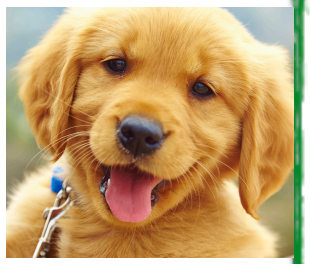
Linear classifier as a set of perceptrons

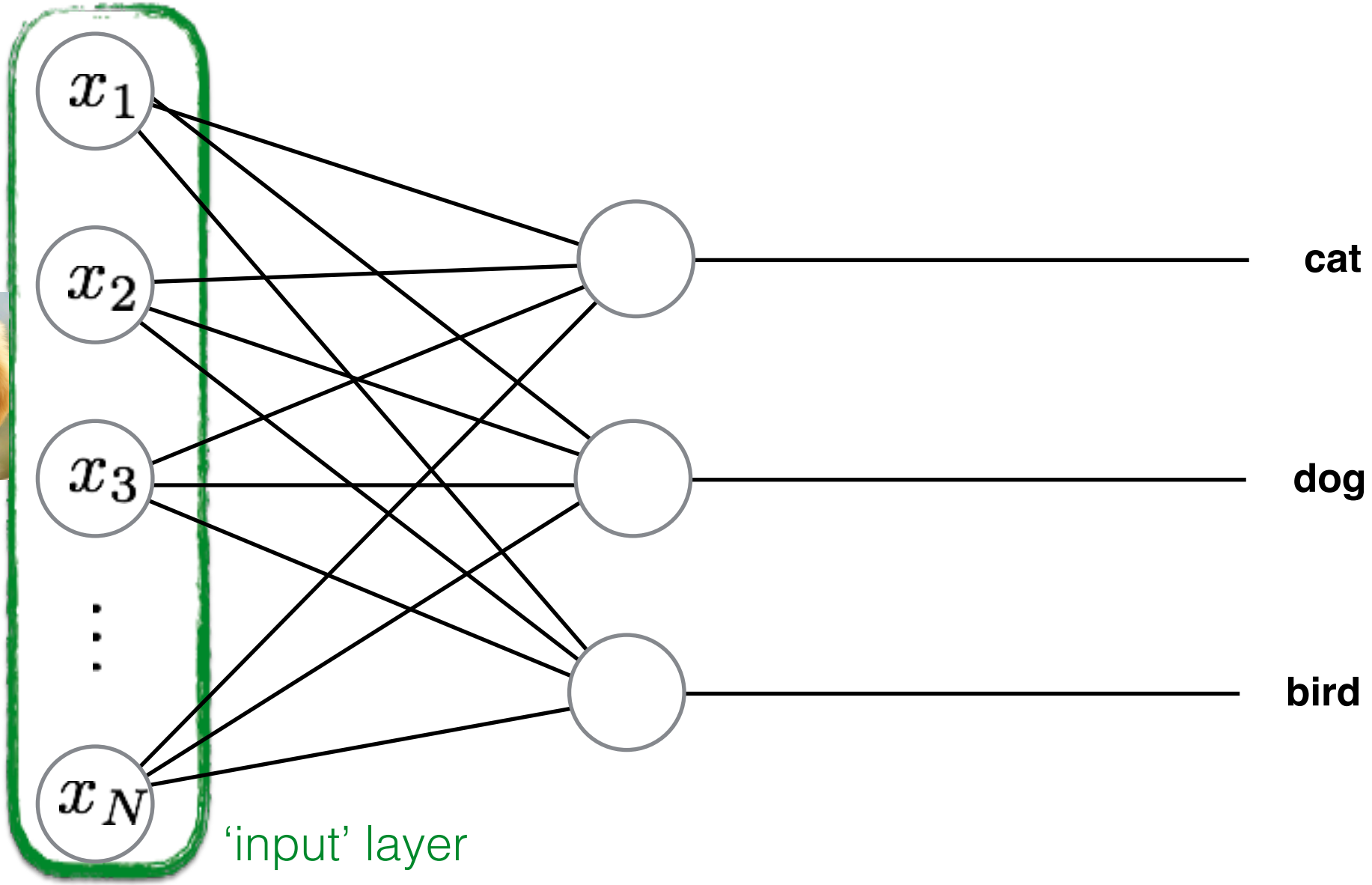


Today's agenda

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

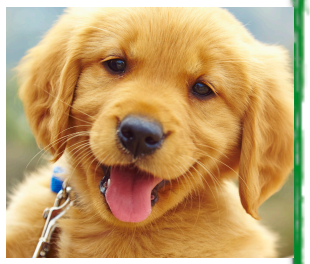
Linear classifier: input layer

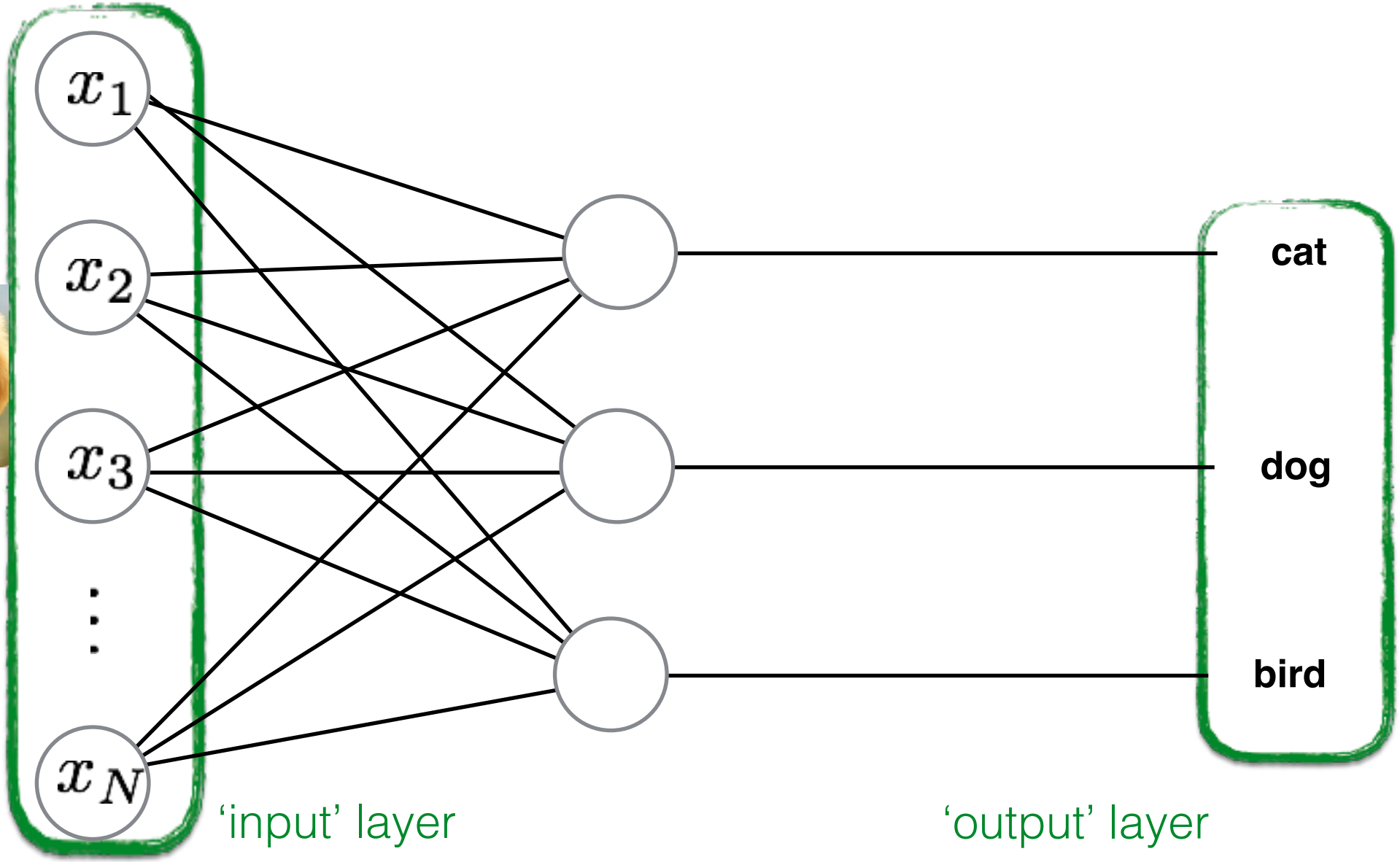




'input' layer

Linear classifier: output layer





Linear classifier: mathematical formulation

x_1

x_2

x_3

$\vdots$

x_N

$f(x, W)$

10 numbers giving class scores

W

Weights or Parameters

Linear classifier: mathematical formulation

(32x32x3)
3072 dimensional vector

x_1

x_2

x_3

$\vdots$

x_N

$f(x, W) = Wx$

$x = 3072 \times 1$

$W = ?$

$f(x, W)$

10 numbers giving class scores

W

Weights or Parameters

Linear classifier: mathematical formulation

(32x32x3)
3072 dimensional vector

x_1

x_2

x_3

$\vdots$

x_N

$f(x, W) = Wx$

$x = 3072 \times 1$

$W = 10 \times 3072$

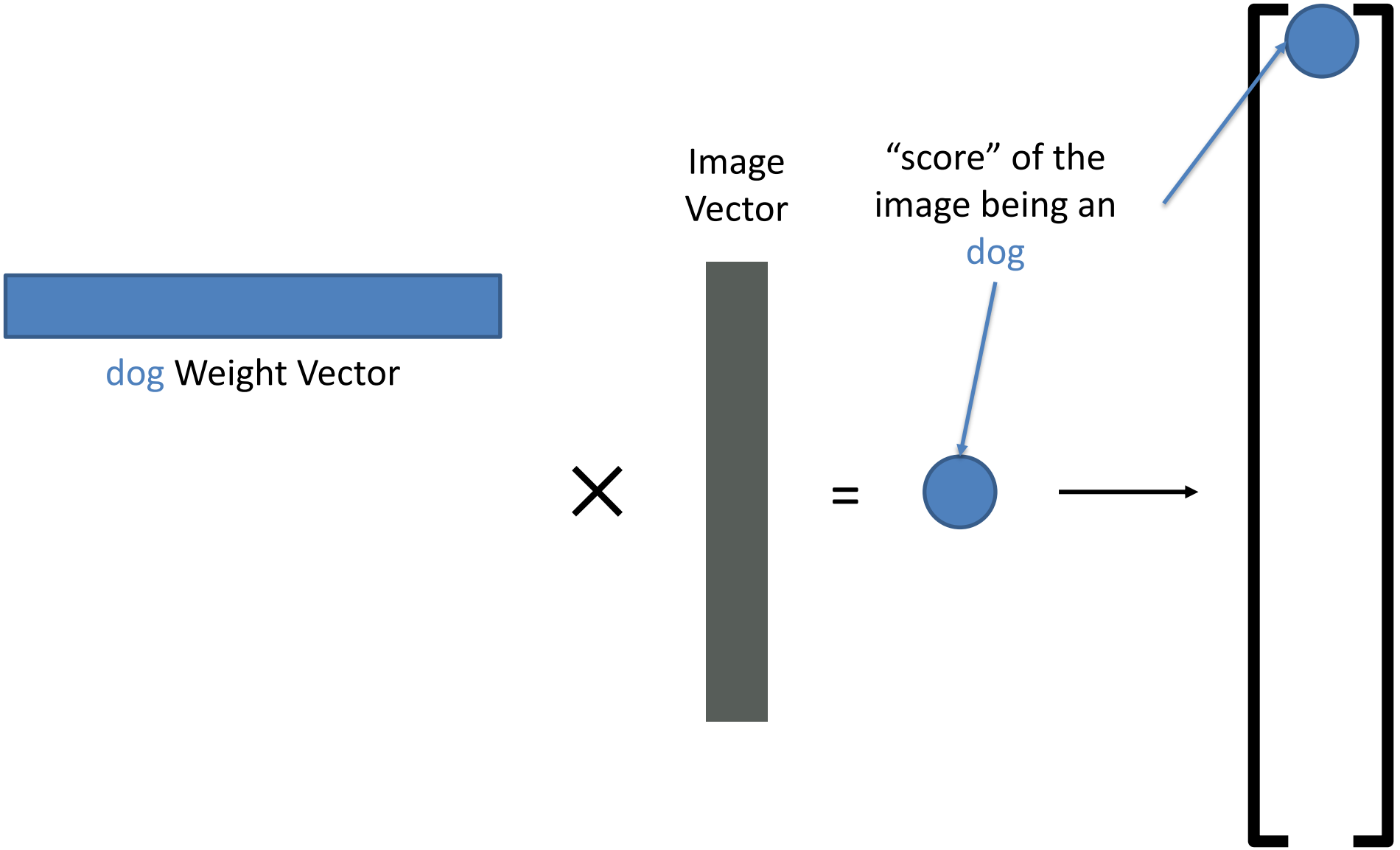
$f(x, W)$

10 numbers giving class scores

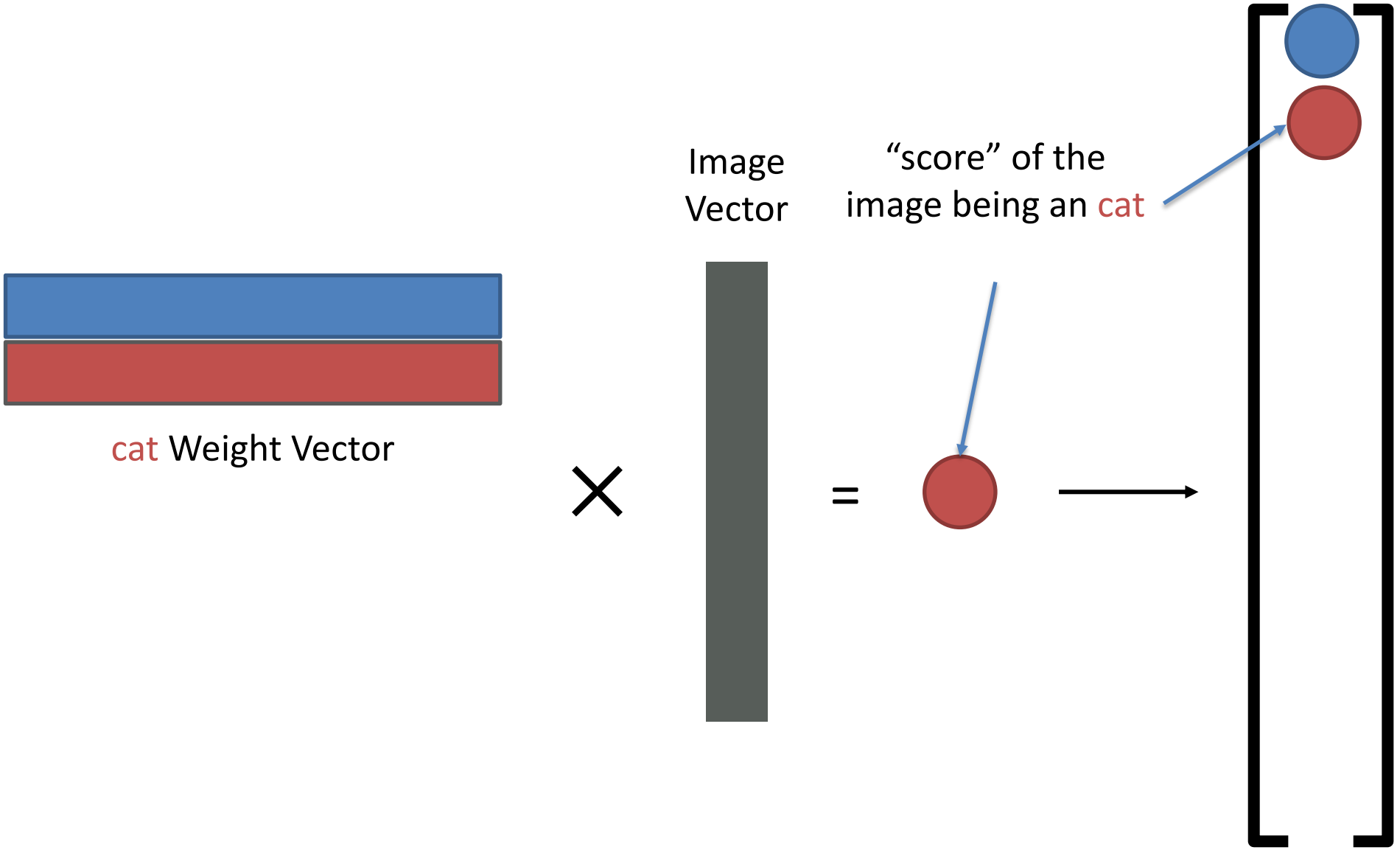
W

Weights or Parameters

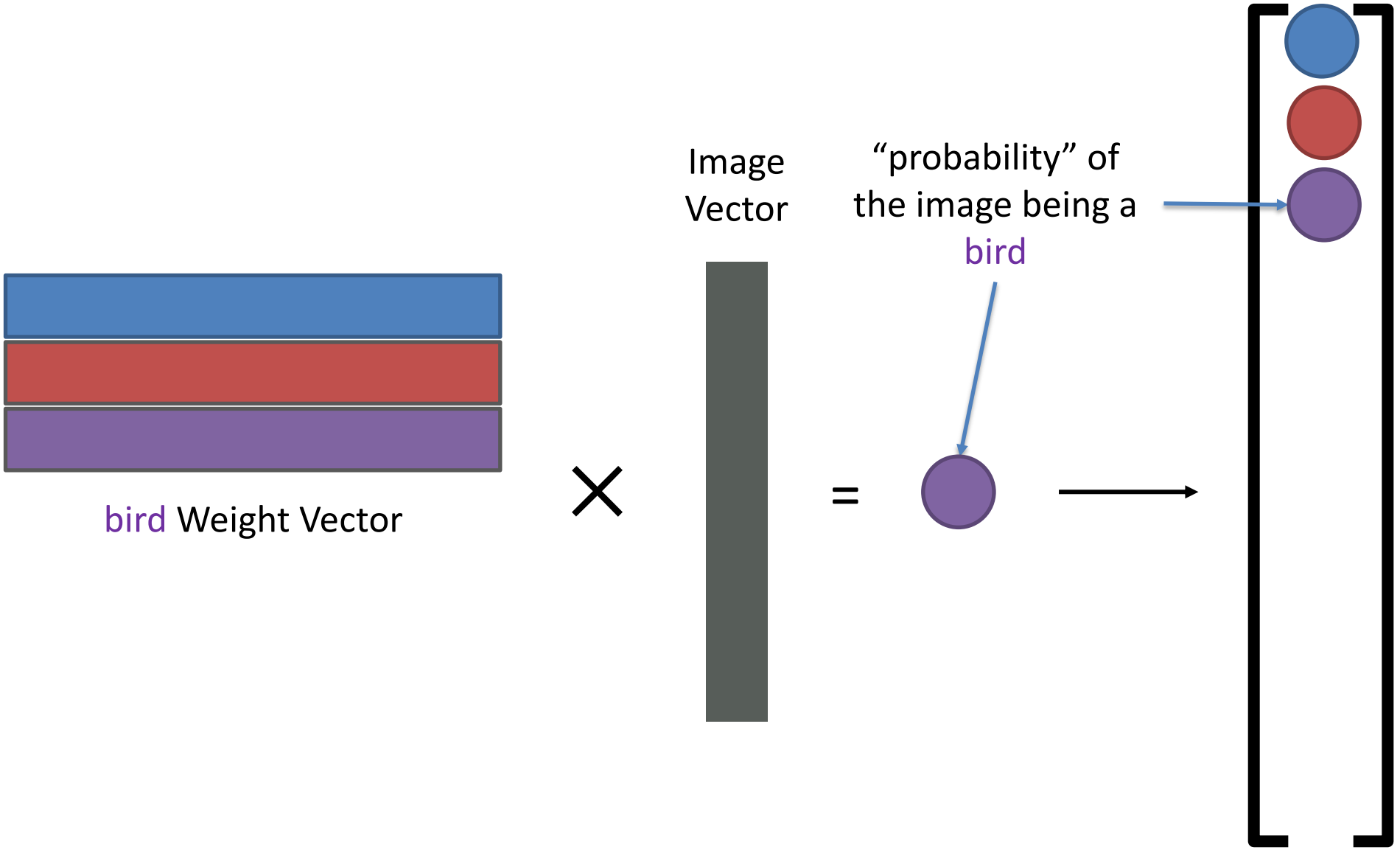
Linear classifier: function visualized



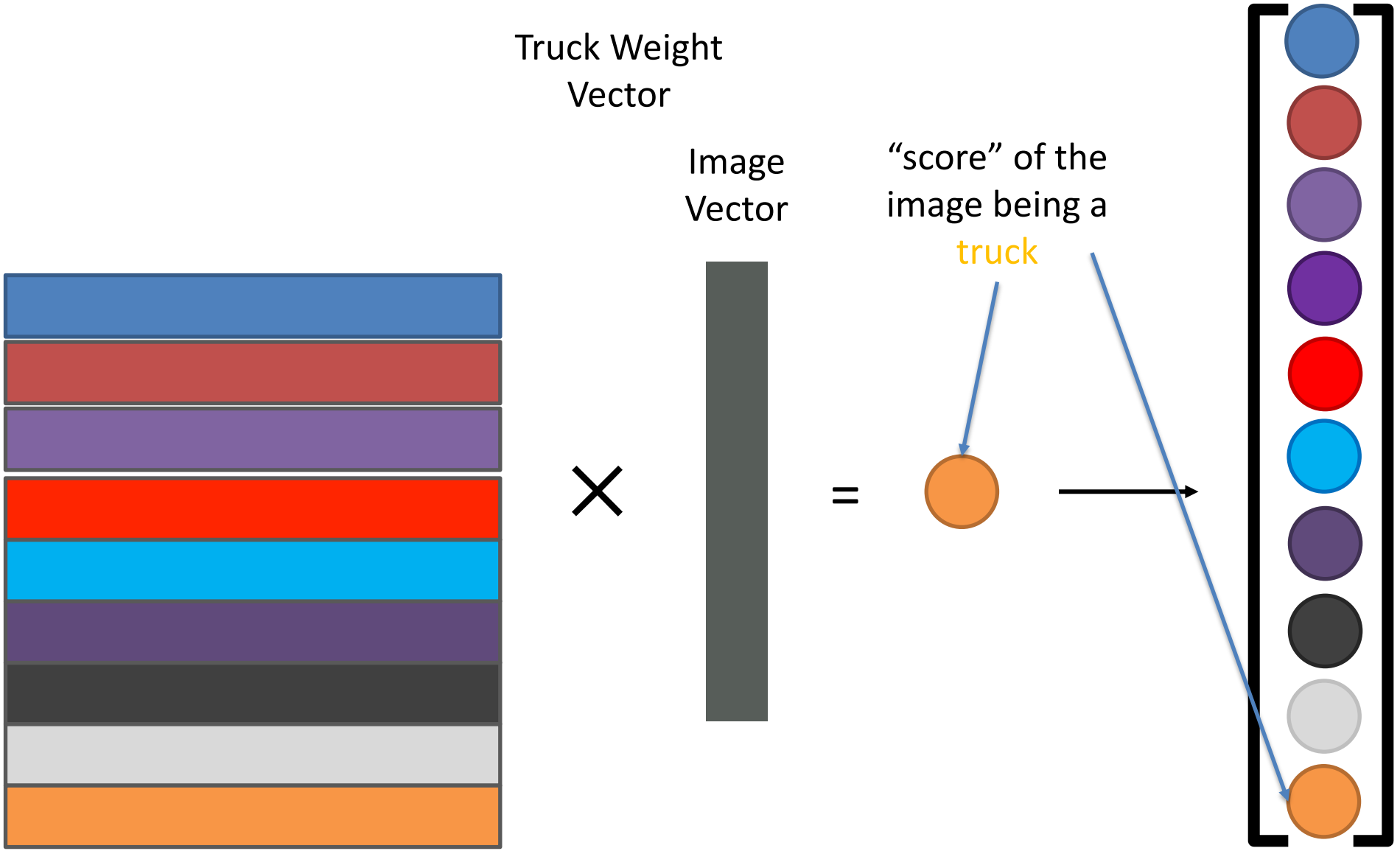
Linear classifier: function visualized



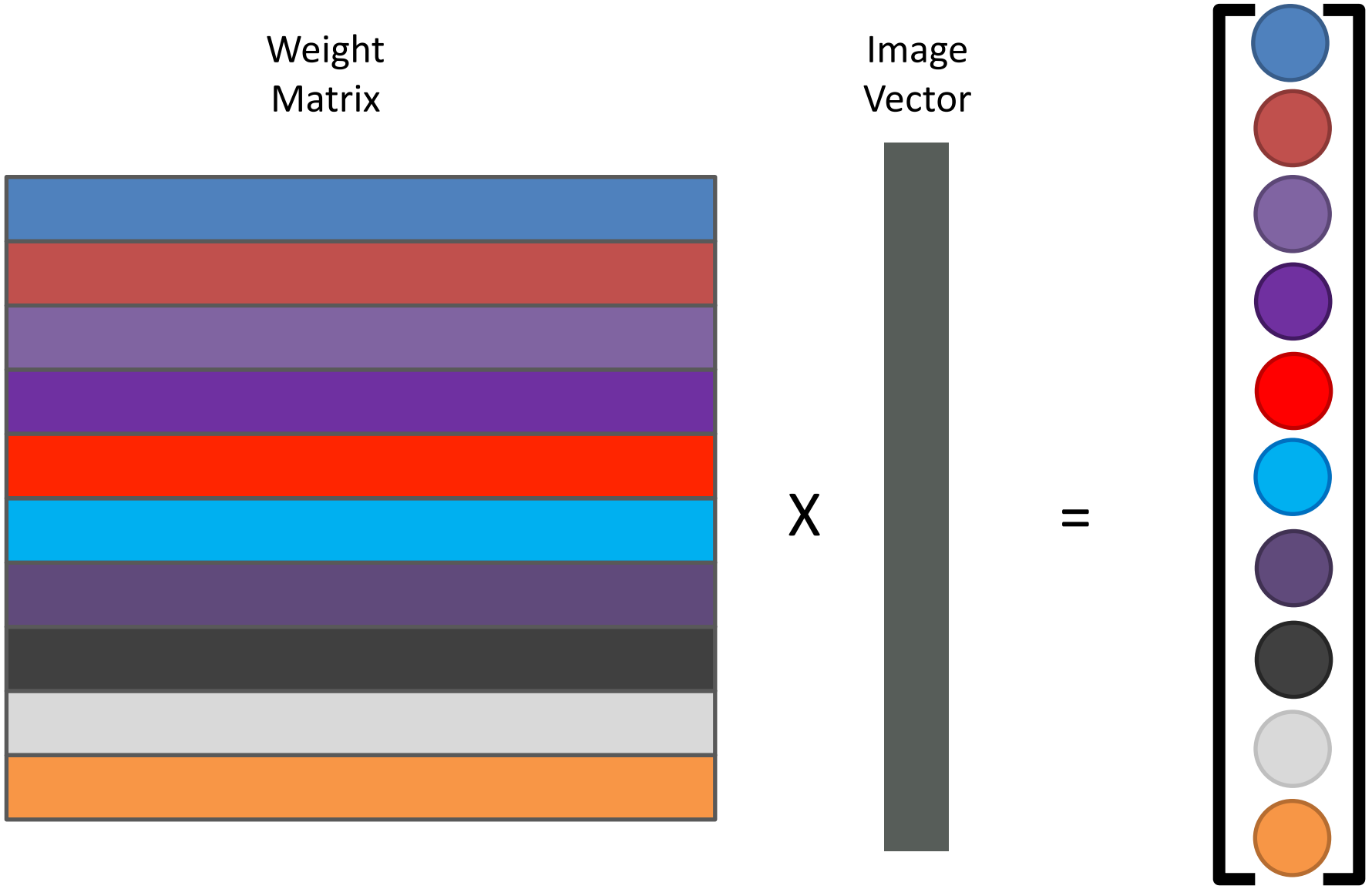
Linear classifier: function visualized



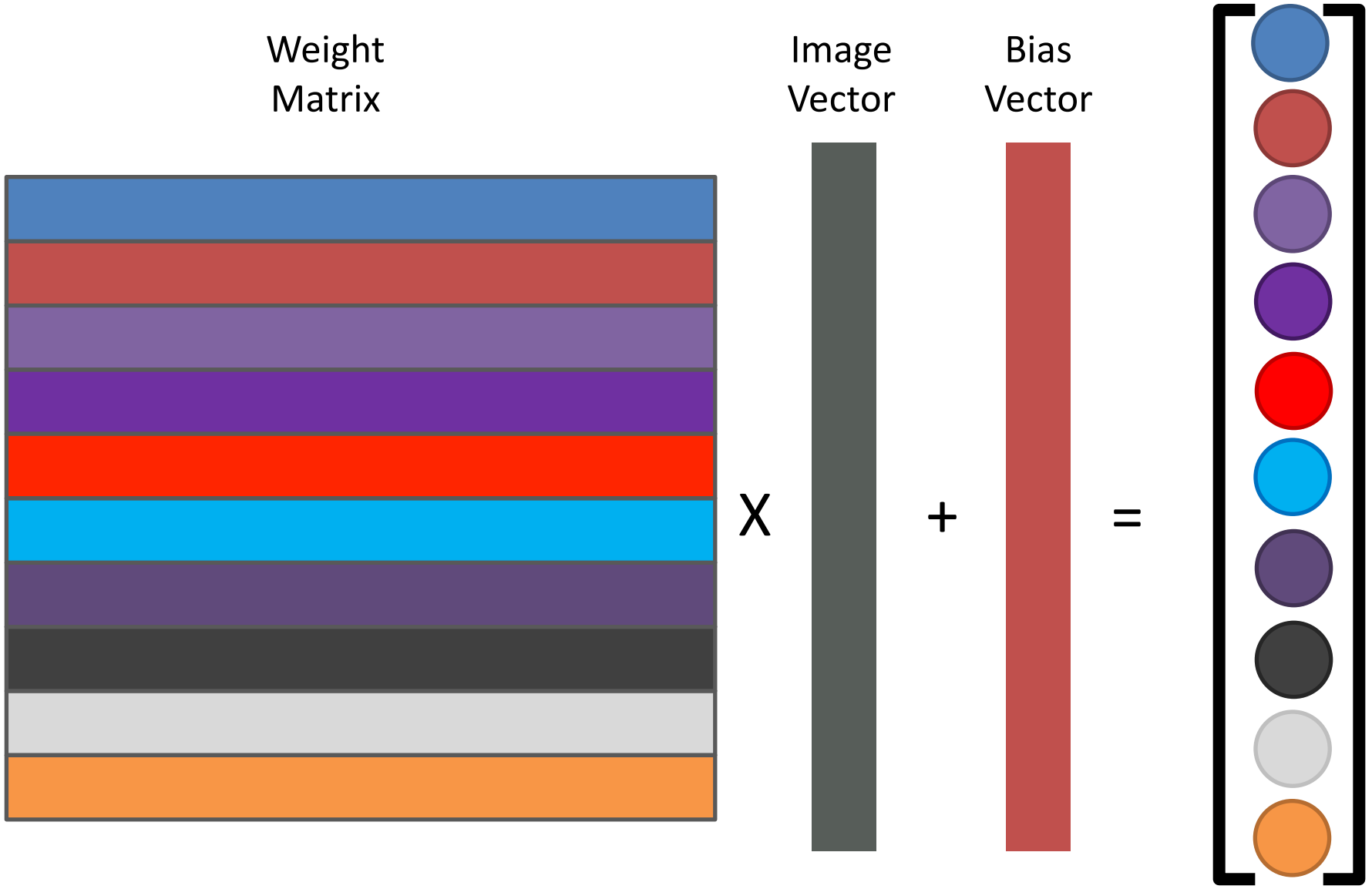
Linear classifier: function visualized



Linear classifier: function visualized

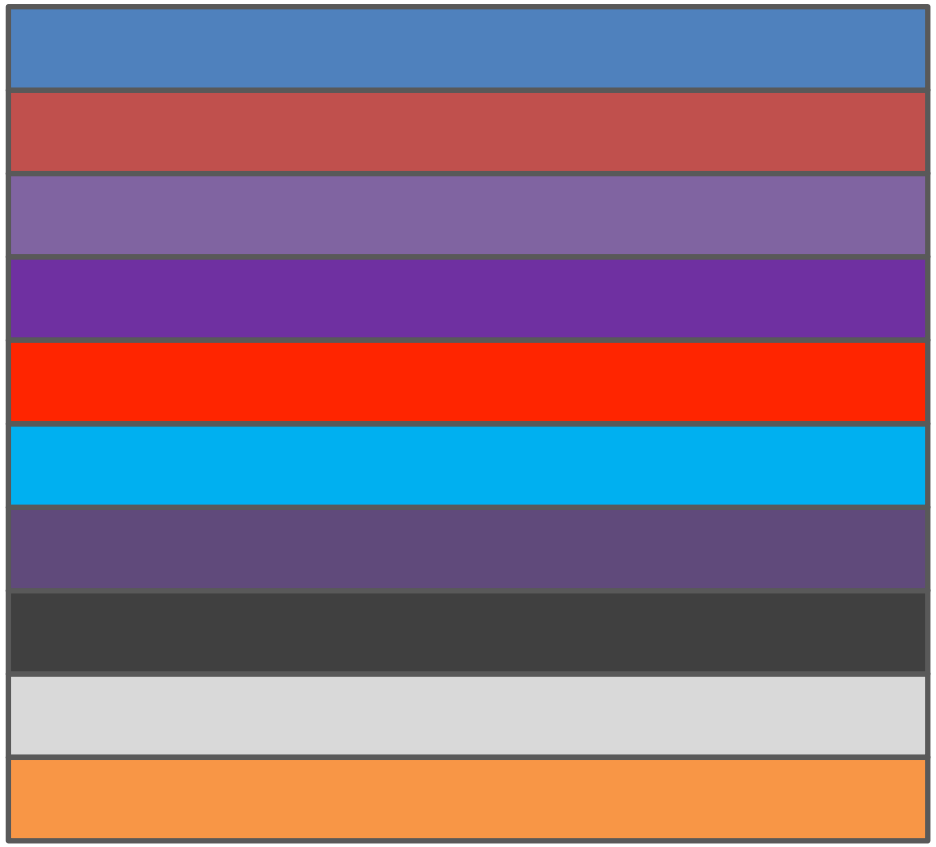


Linear classifier: bias vector



Linear classifier: size

10x3072
Weight
Matrix



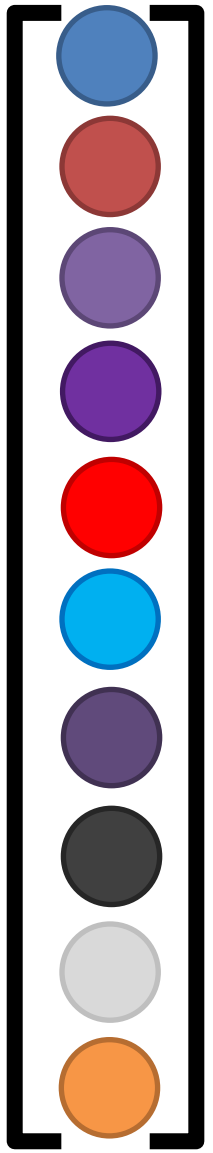
X

3072x1
Image
Vector

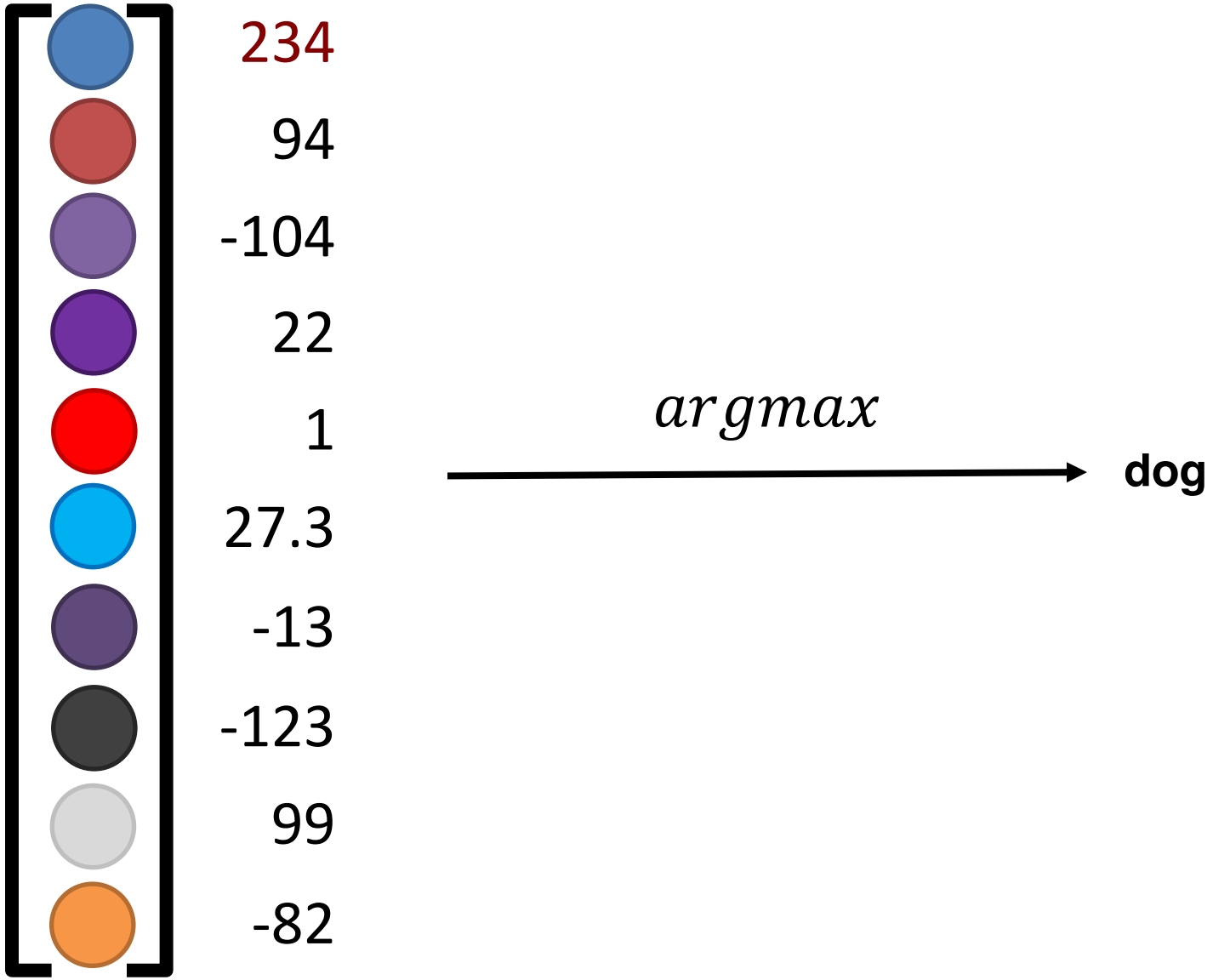
+

10x1
Bias
Vector

=

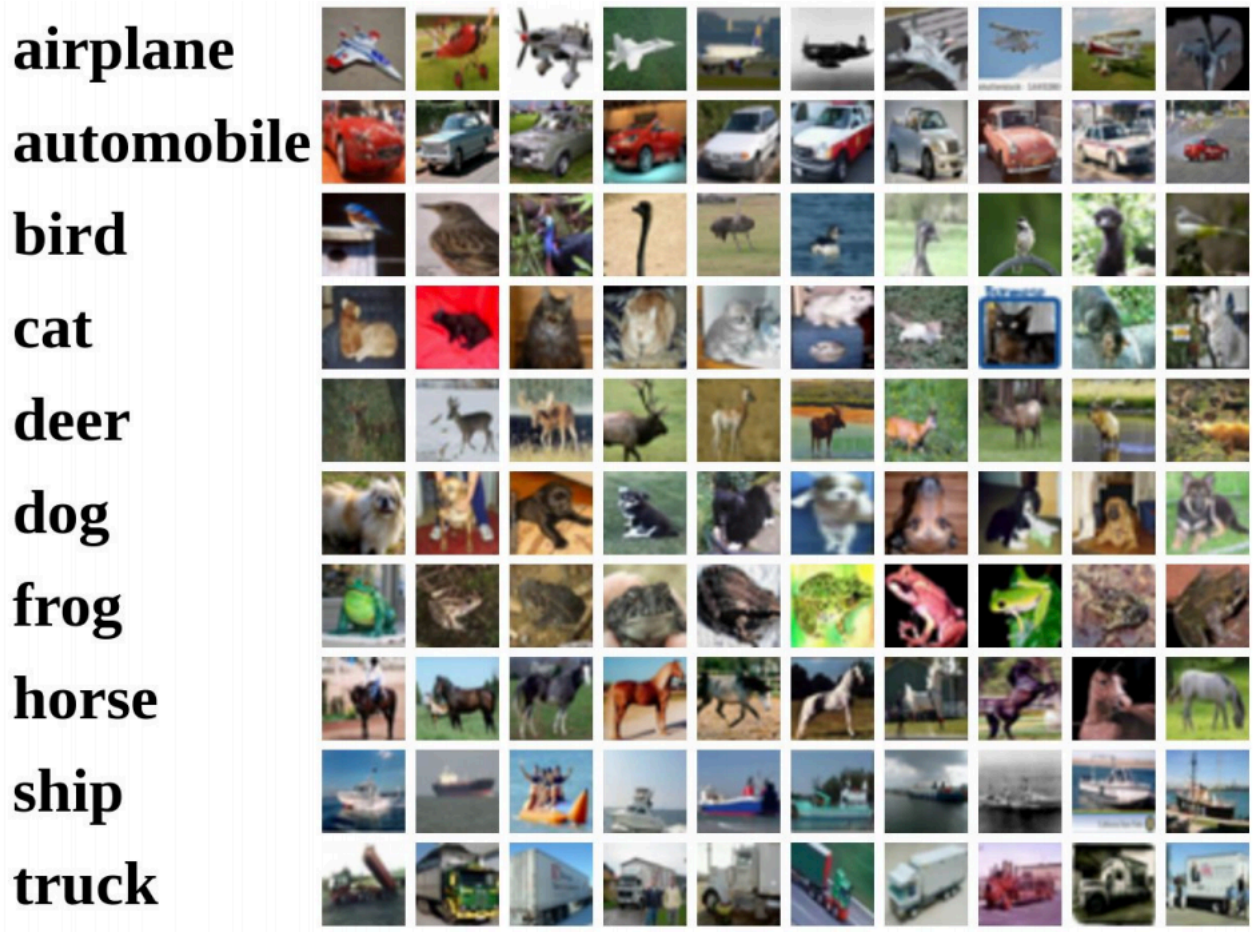


Linear classifier: Making a classification



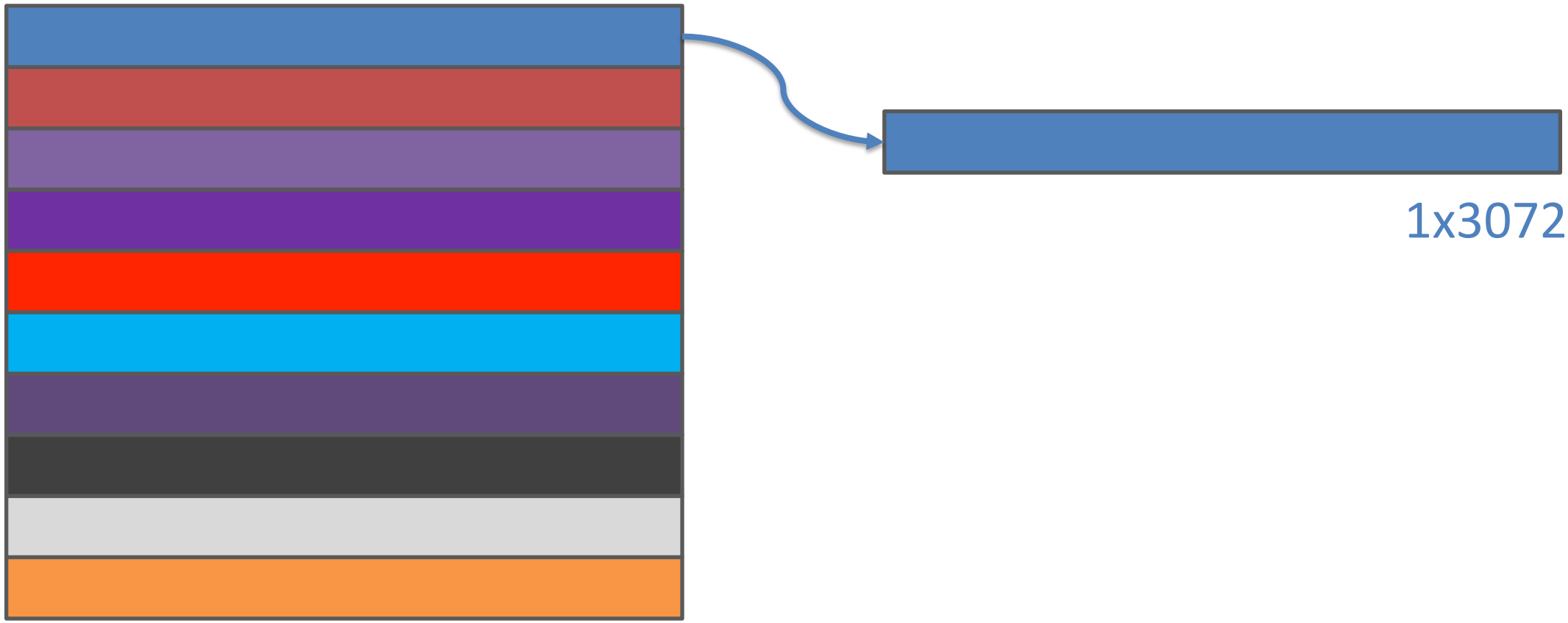
Interpreting the weights

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



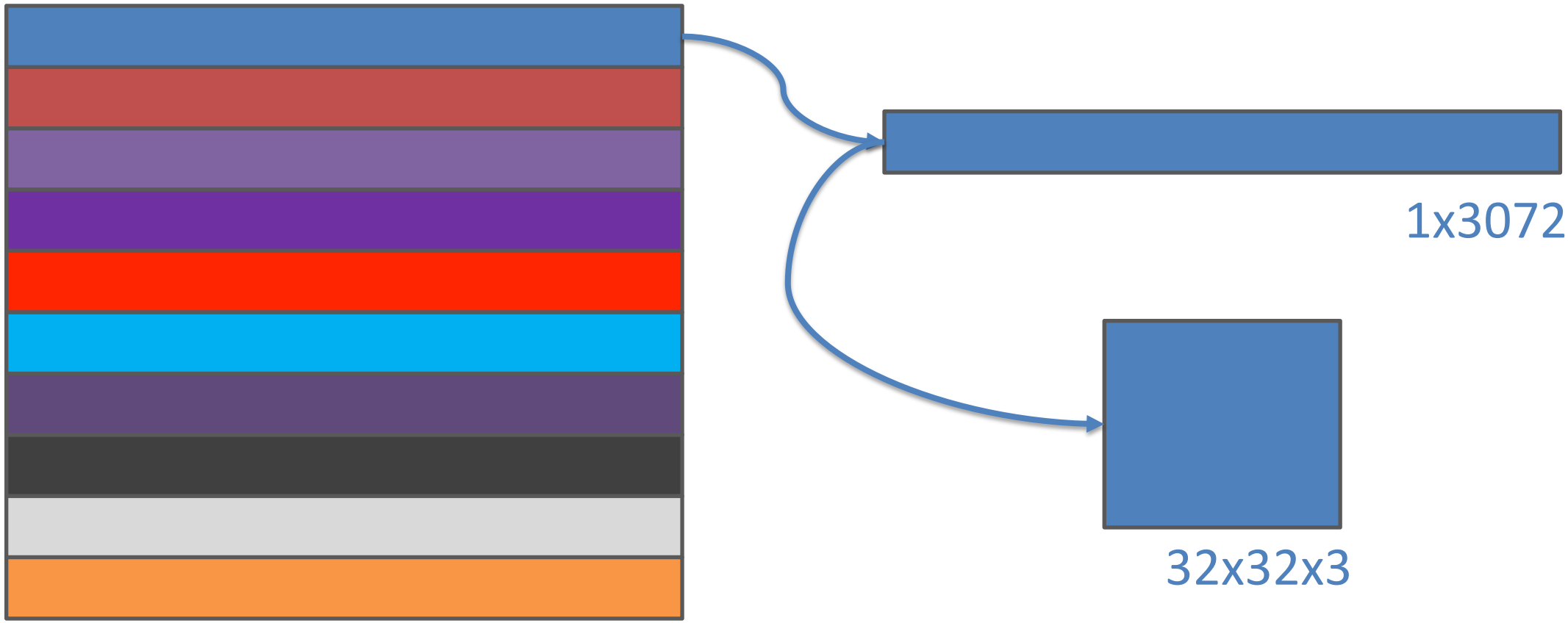
Interpreting the weights as templates

Let us look at each row of the weight matrix



Interpreting the weights as templates

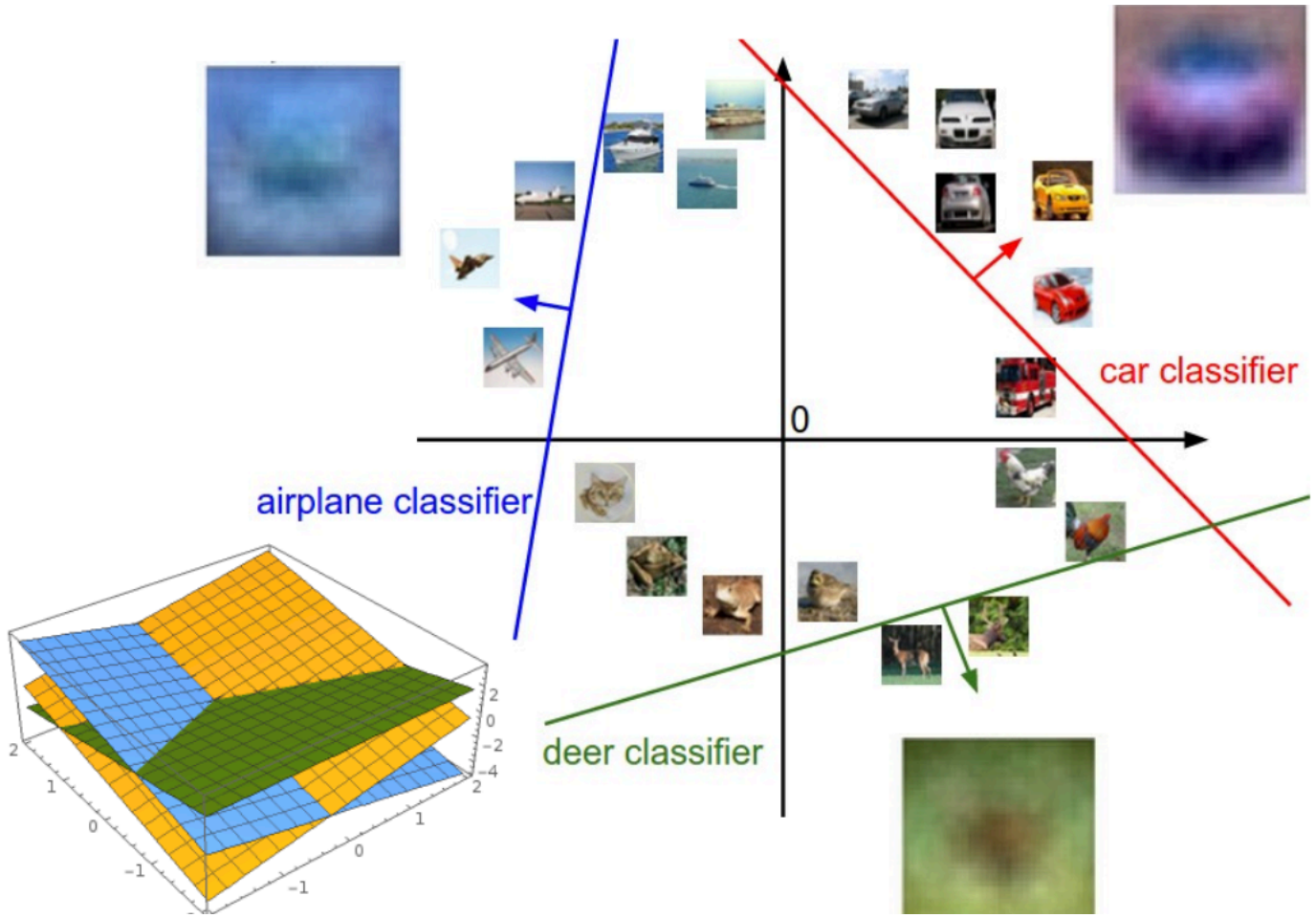
We can reshape the vector back in to the shape of an image



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

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

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

All we have to do is define a loss function!

Given training data:

$$y = wx$$

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think is a good approximation weight parameter for this data point?

Given training data:

$$y = wx$$

x	y	x	y
10	10.1	2	20.1
2	1.9	4	13.1
3.5	3.4	5	2.4
1	1.1	0.6	0.1

What about this one?

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 $\mathbf{x_i}$ is image and $\mathbf{y_i}$ is (integer) label
(0 for dog, 1 for cat, etc.)

A loss function $\mathbf{L_i}(y_i, \hat{y}_i)$ tells us how good our current classifier
- When the classifier predicts correctly ($y_i = \hat{y}$), the loss should be low
- When the classifier makes mistakes ($y_i \neq \hat{y}$), , 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 $\mathbf{x_i}$ is image and $\mathbf{y_i}$ is (integer) label
(0 for dog, 1 for cat, etc.)

Loss over the entire dataset is an average of loss over examples:

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

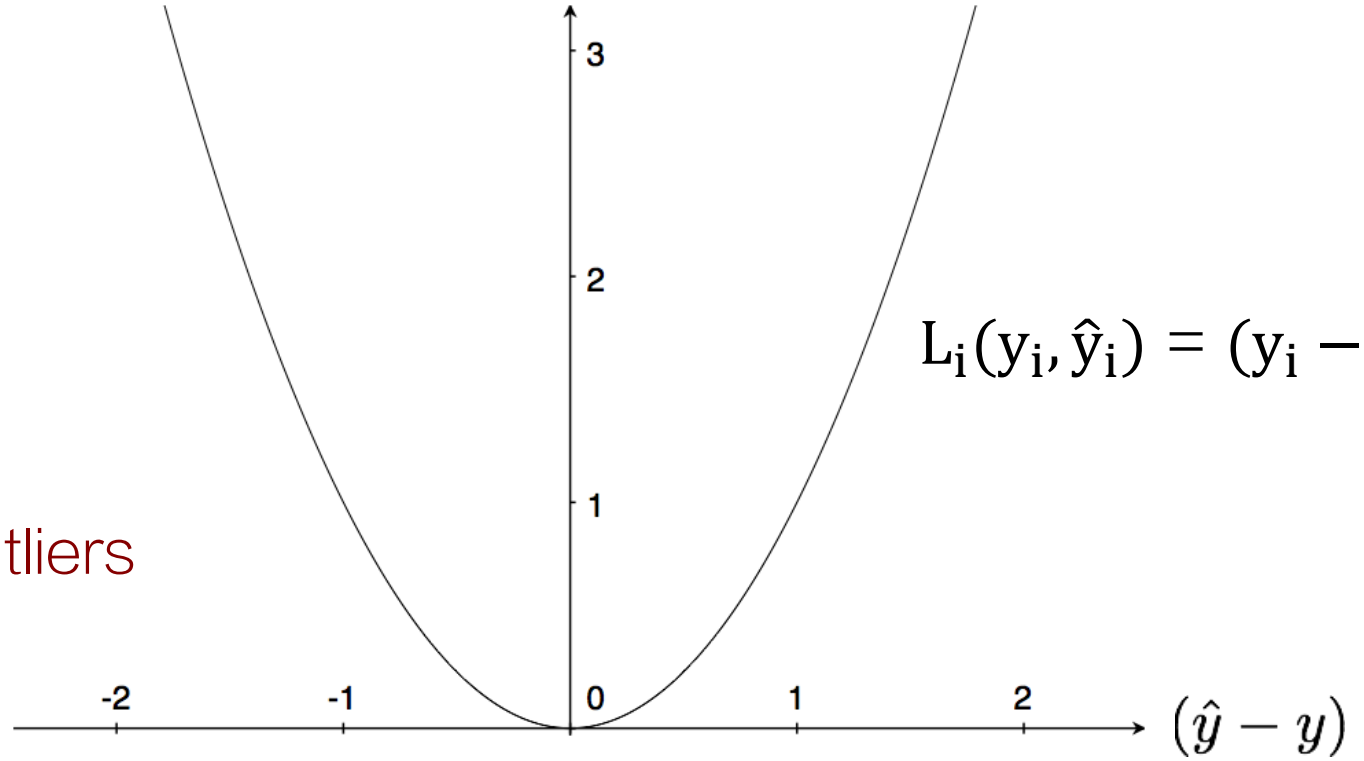
How do we choose L_i ?

YOU get to chose the loss function!

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

Squared Error (L2)

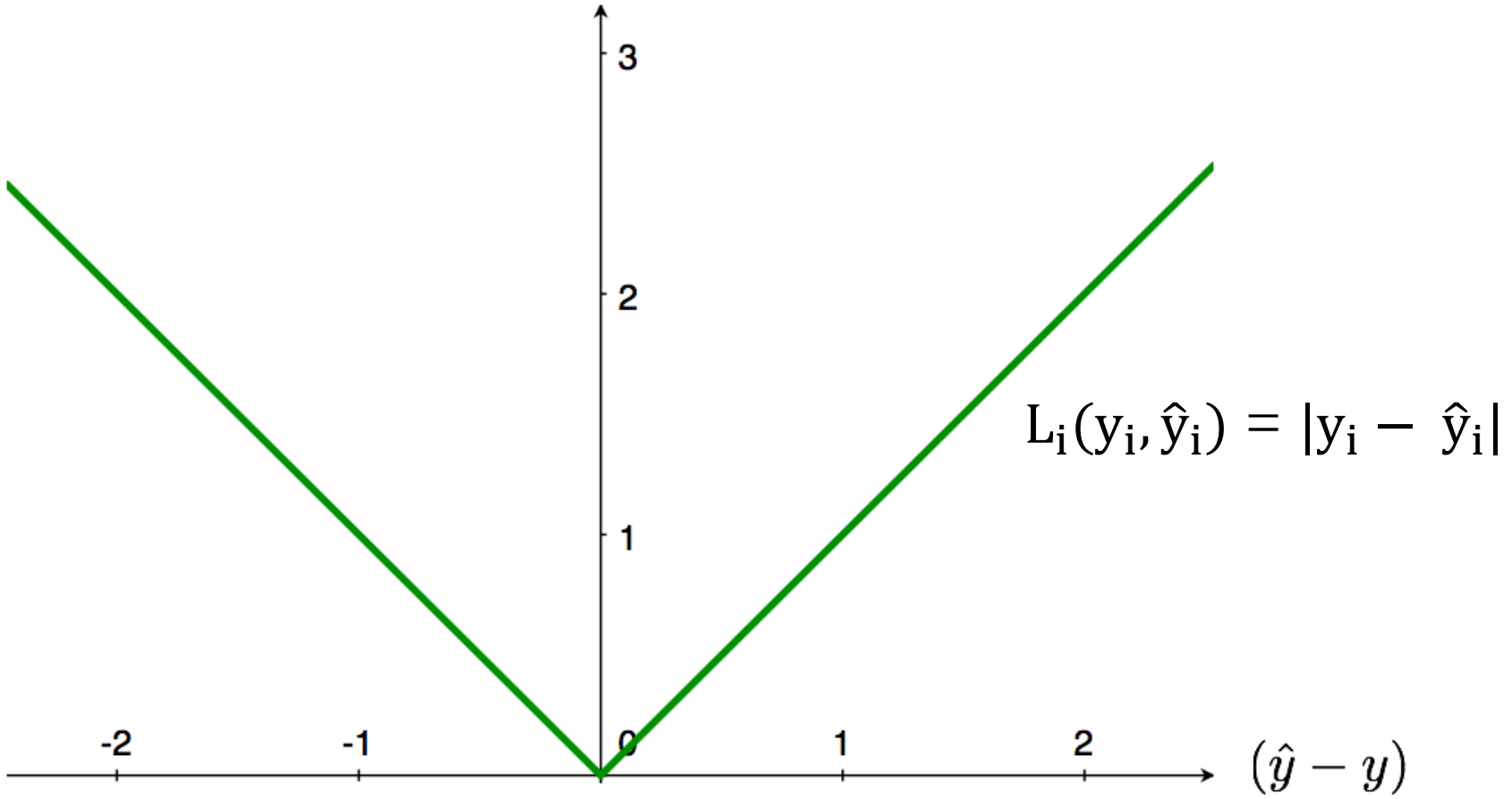
(a popular loss function) ((why?))



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

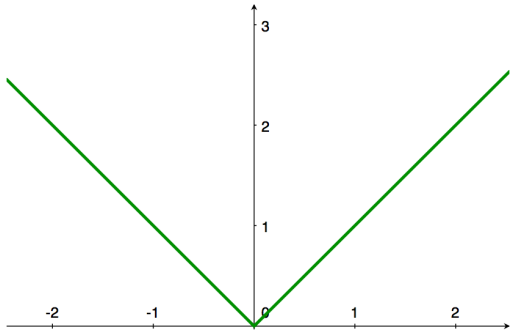
Not robust to outliers

L1 loss



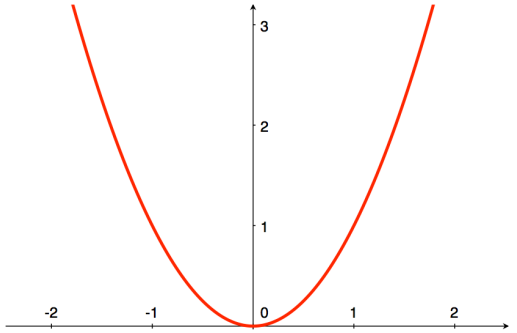
L1 Loss

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



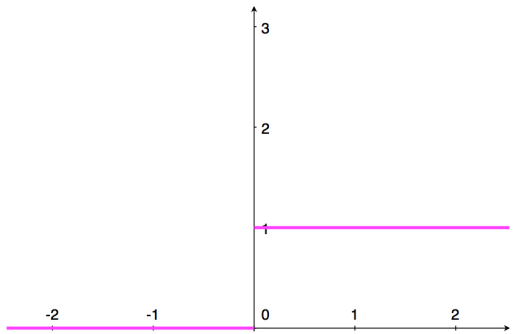
L2 Loss

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



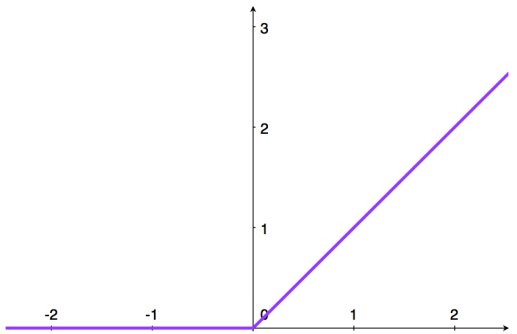
Zero-One Loss

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



Hinge Loss

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



Softmax Classifier (Multinomial Logistic Regression)

- It allows us to 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 the correct class.

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

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

dog
cat
bird

correct outputs

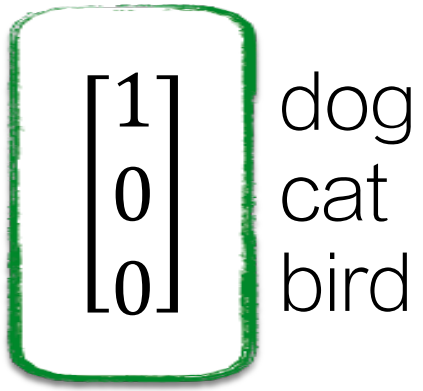
Softmax Classifier (Multinomial Logistic Regression)

- KL divergence.

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

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

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



correct outputs

Softmax Classifier (Multinomial Logistic Regression)

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

Remember our linear classifier:

$$\hat{y} = wx$$

there are no limits on the output space. Meaning that the model can generate outputs > 1 or < 0.

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

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 outputs into probability ranges [0,1].

Solution: 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

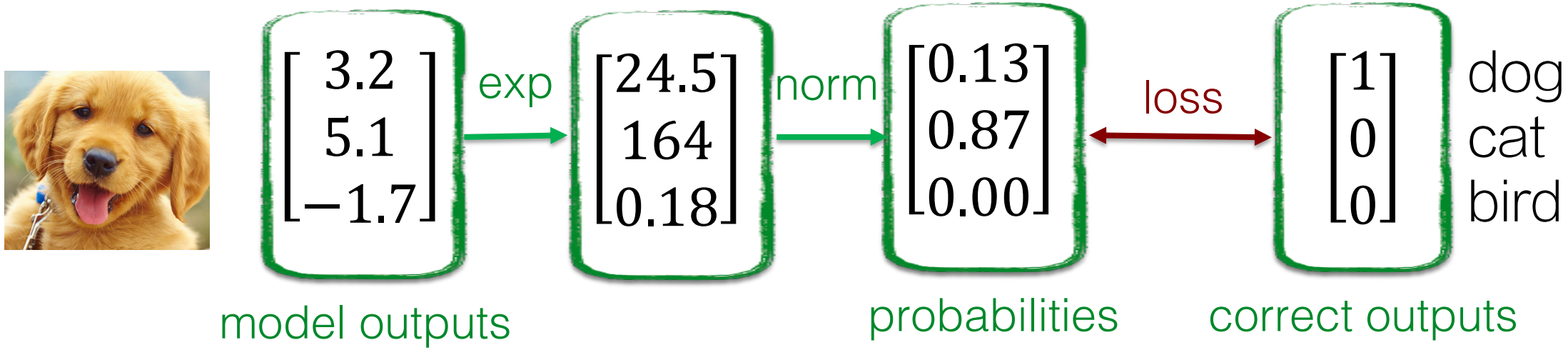
correct outputs

Softmax Classifier (Multinomial Logistic Regression)

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

We need a mechanism to convert or normalize the outputs into probability ranges [0,1].

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

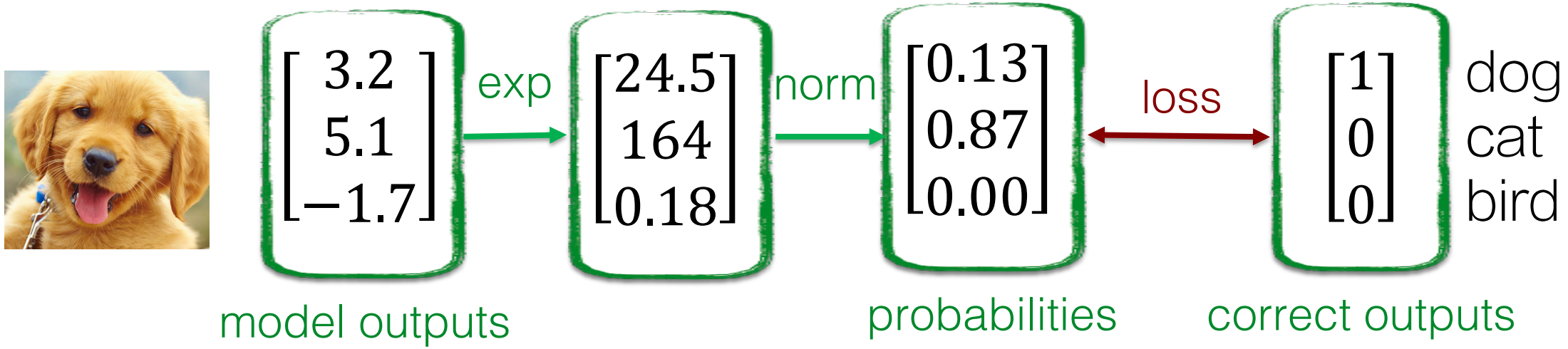


Softmax Classifier (Multinomial Logistic Regression)

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

In this case, what is the loss:

$$L_i = ??$$

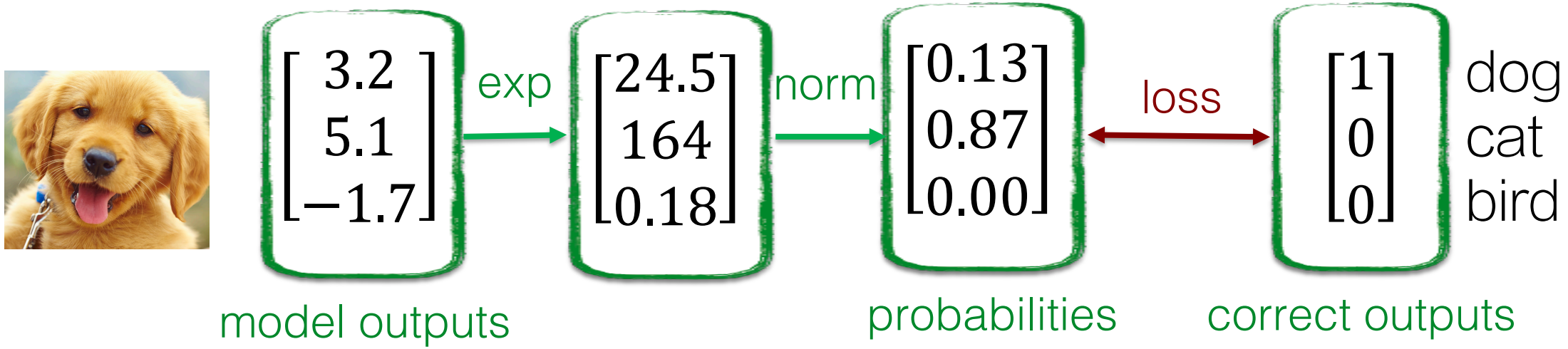


Softmax Classifier (Multinomial Logistic Regression)

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

In this case, what is the loss:

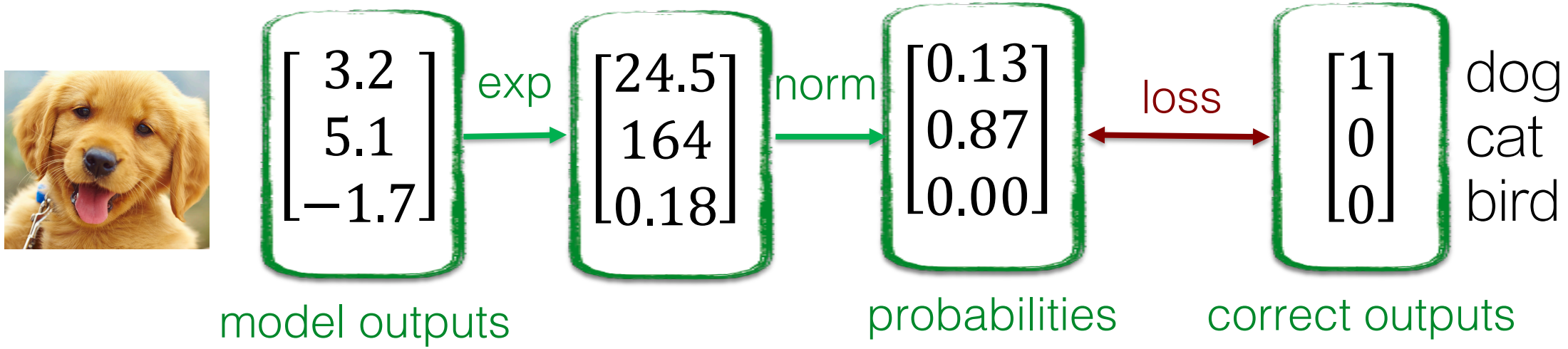
$$L_i = -\log(0.13) = 2.04$$



Softmax Classifier (Multinomial Logistic Regression)

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

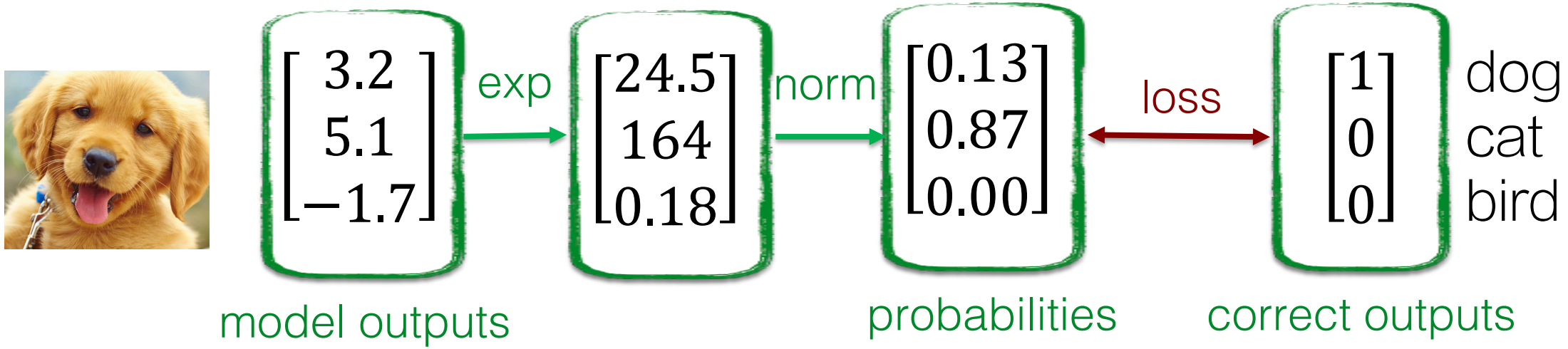
What is the minimum and maximum values that the loss can be?



Softmax Classifier (Multinomial Logistic Regression)

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

At initialization, all the weights will be random. In this case, we can assume that the outputs will have the same probability. In this case, what will the initial loss be?

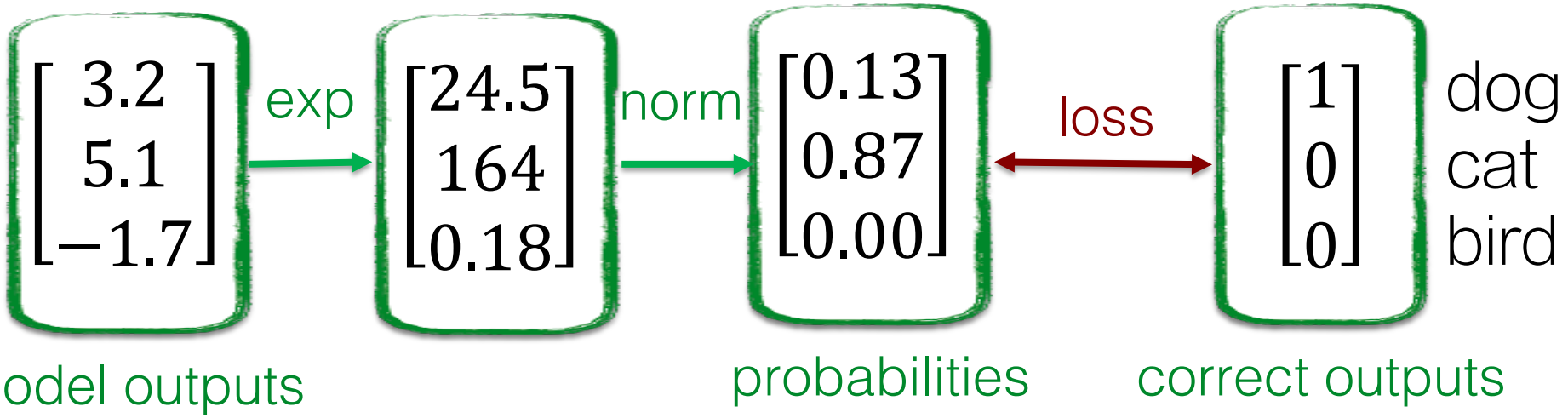


Softmax Classifier (Multinomial Logistic Regression)

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

At initialization, all the weights will be random. In this case, we can assume that the outputs will have the same probability. In this case, what will the initial loss be?

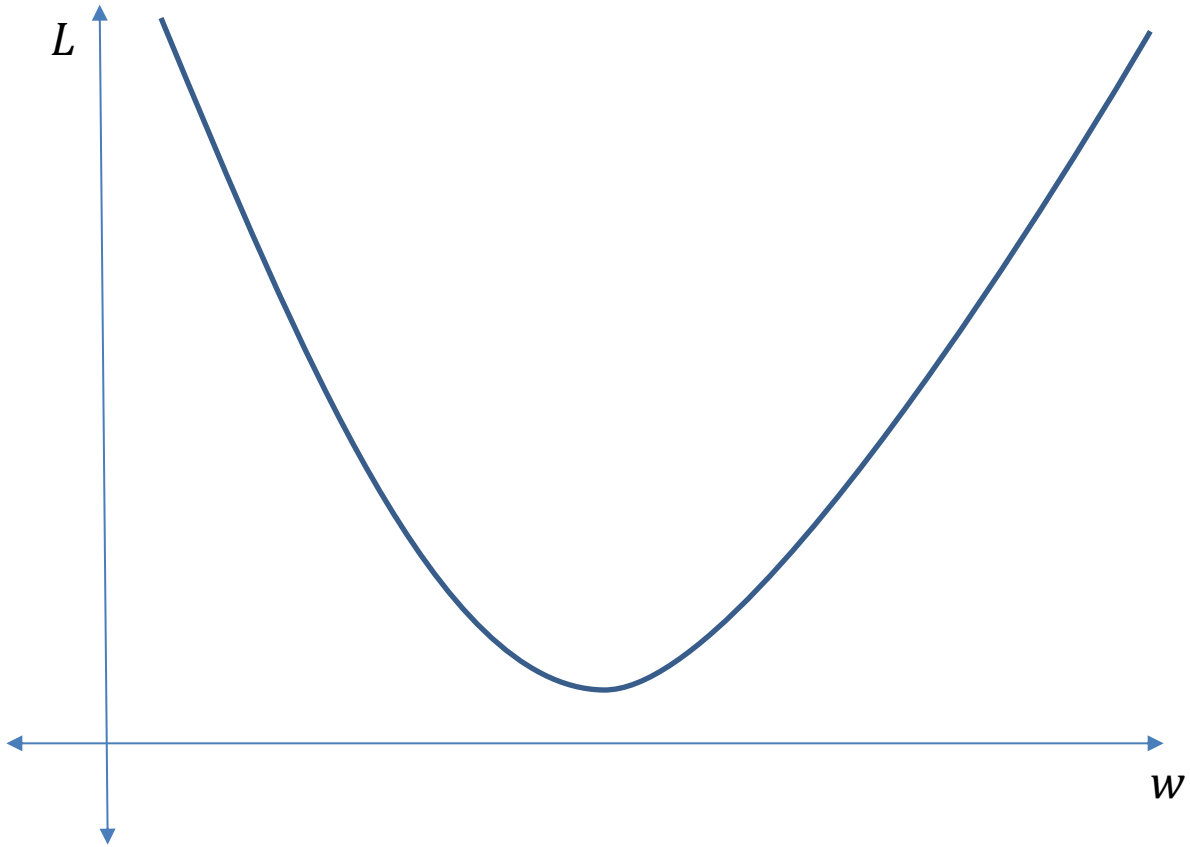
$$L_i = -\log\left(\frac{1}{C}\right) = \log(C) = \log(10) = 2.03$$



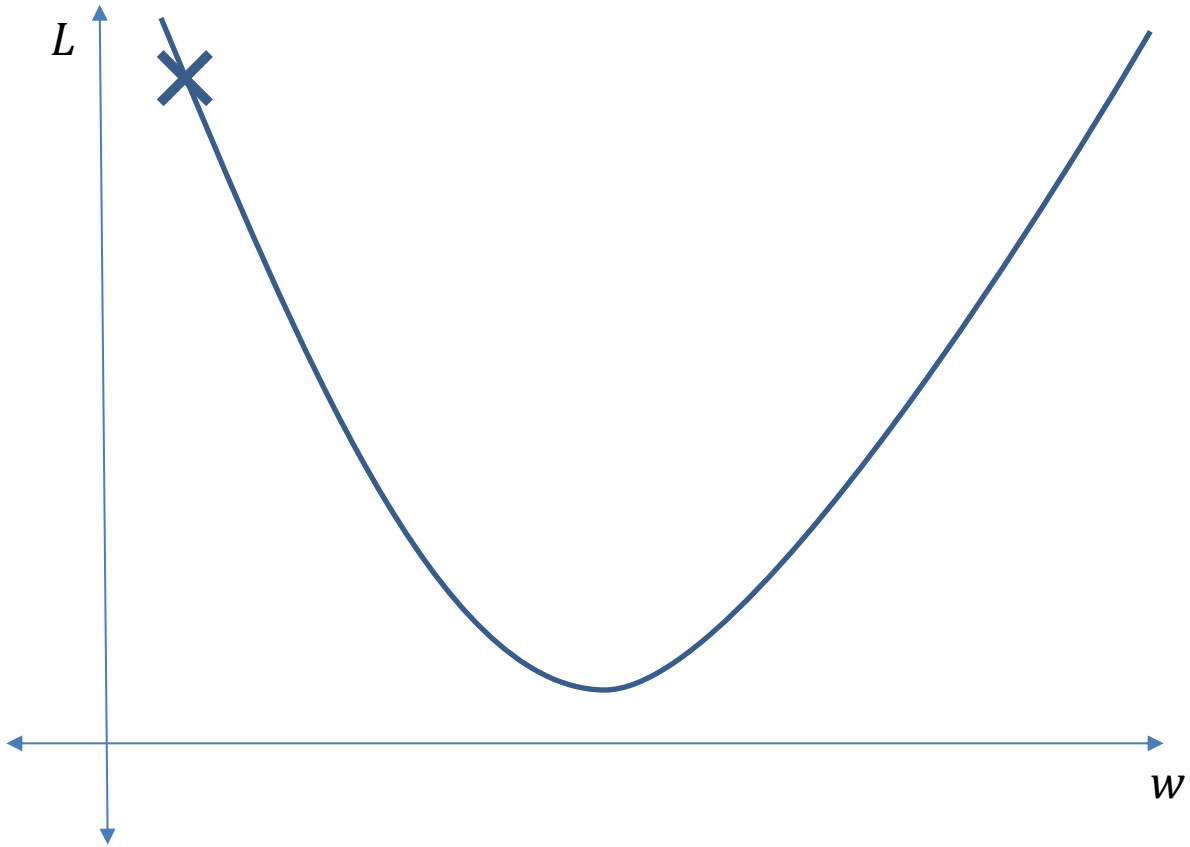
Today's agenda

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

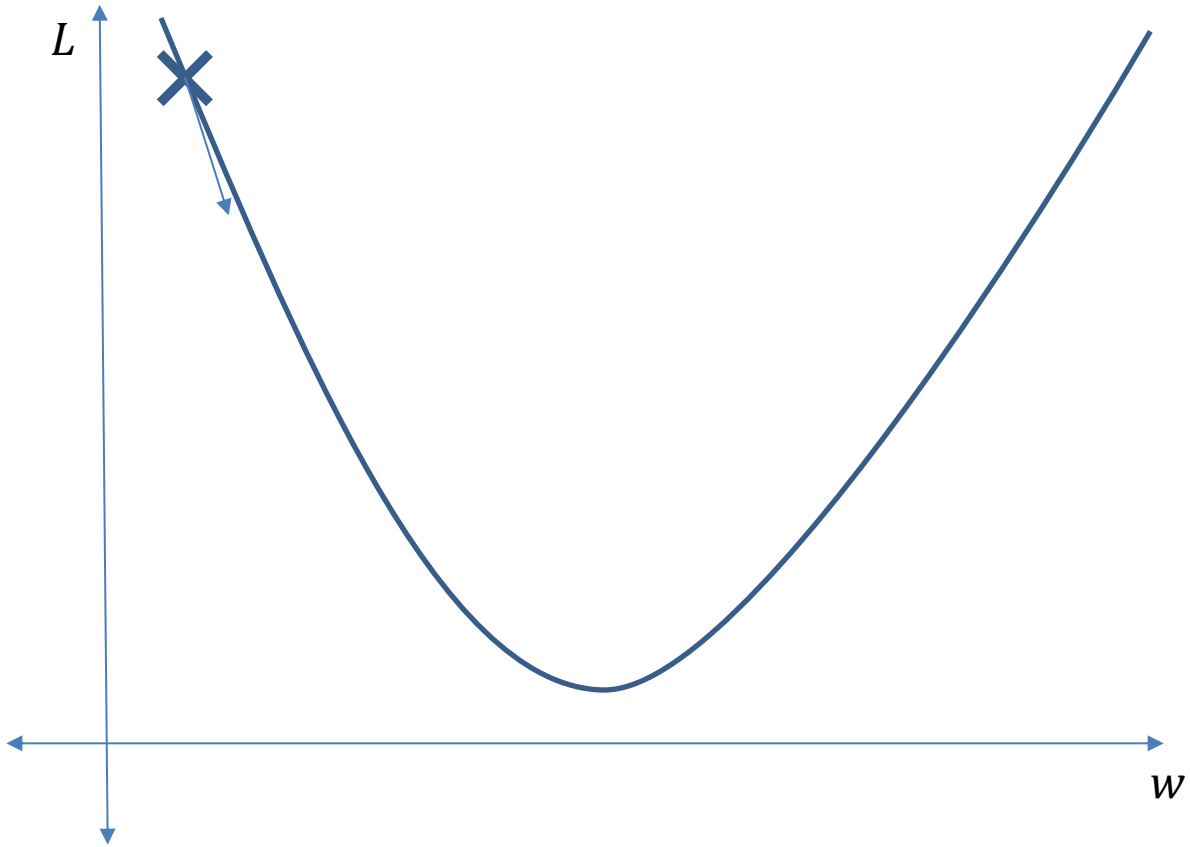
Gradient descent visualized: Minimizing loss



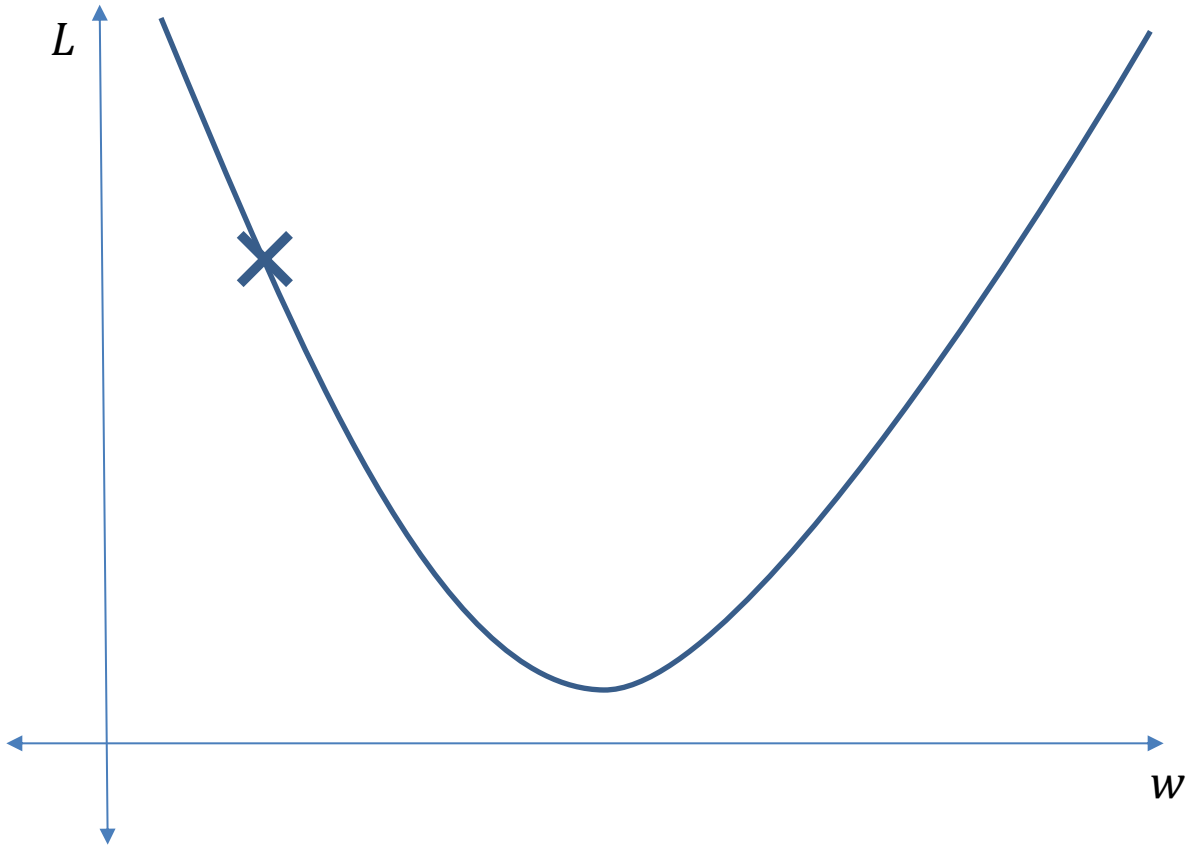
Gradient descent visualized: Minimizing loss



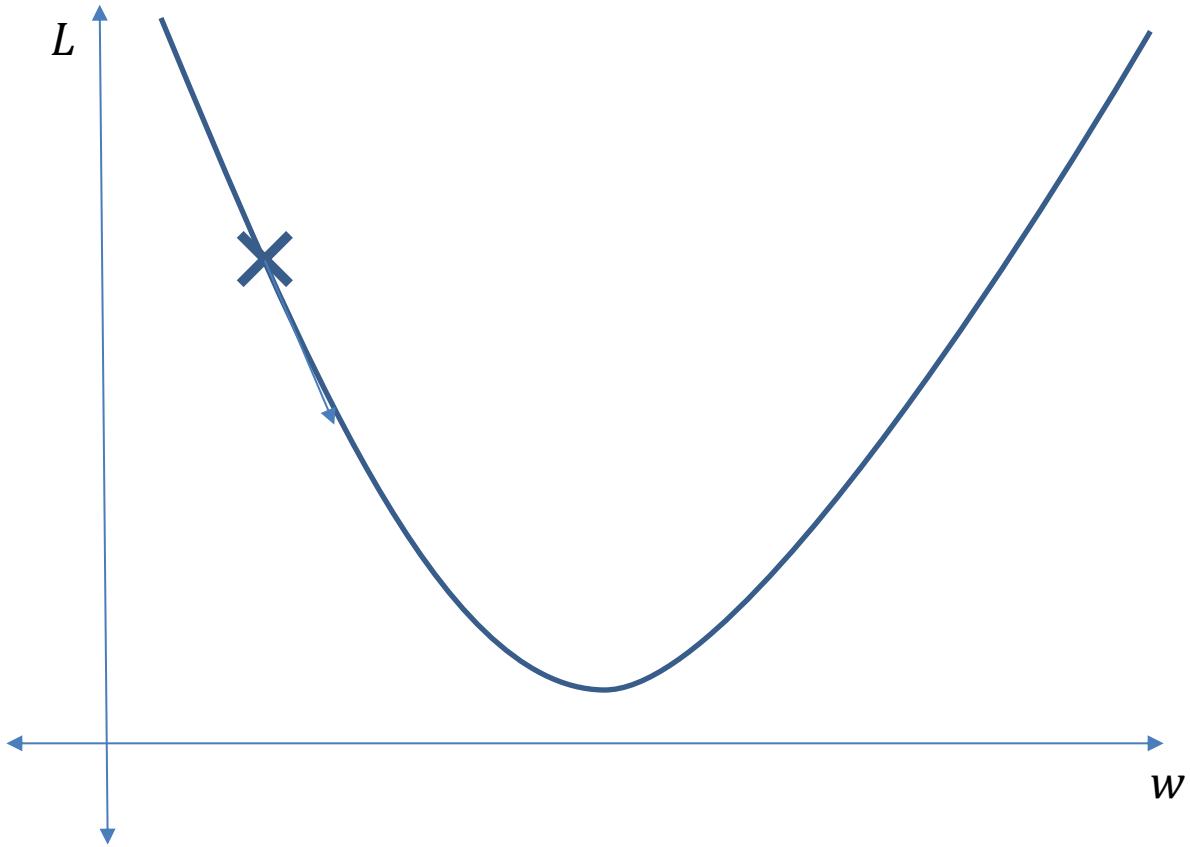
Gradient descent visualized: Minimizing loss



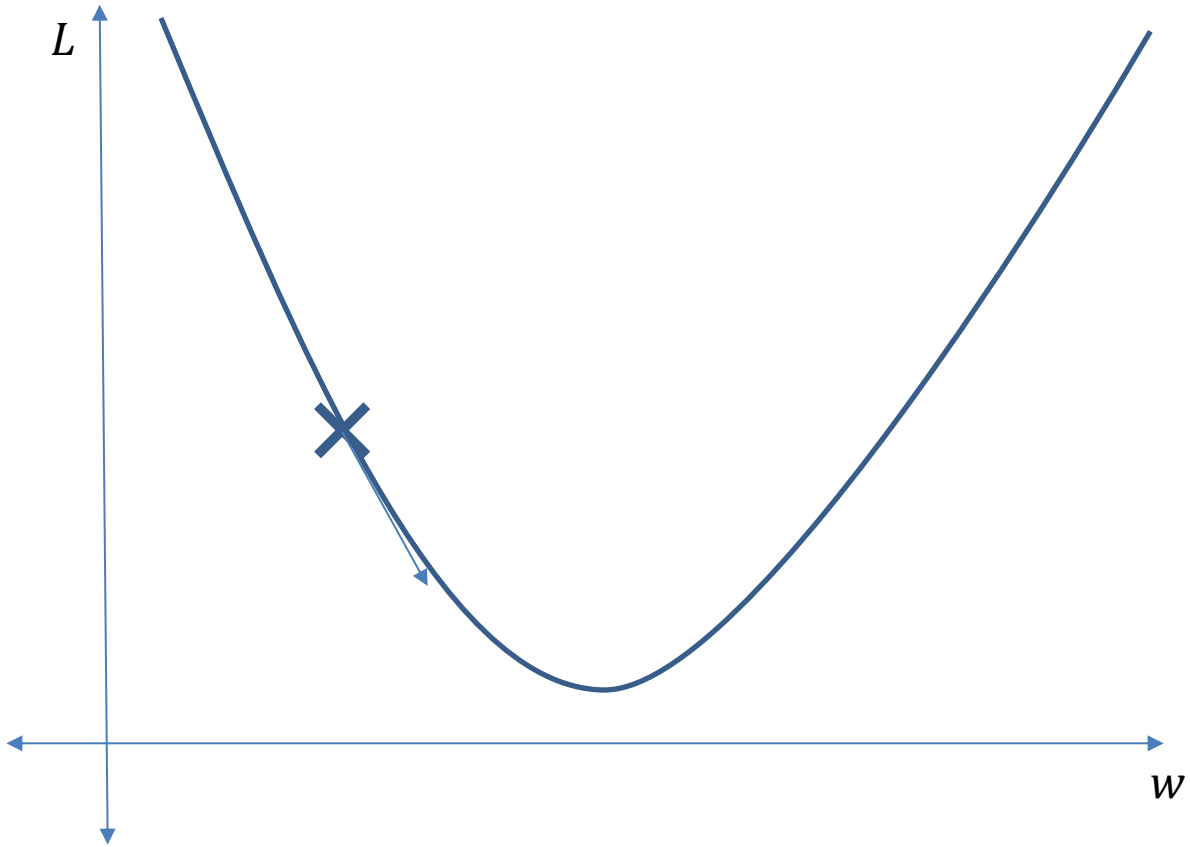
Gradient descent visualized: Minimizing loss



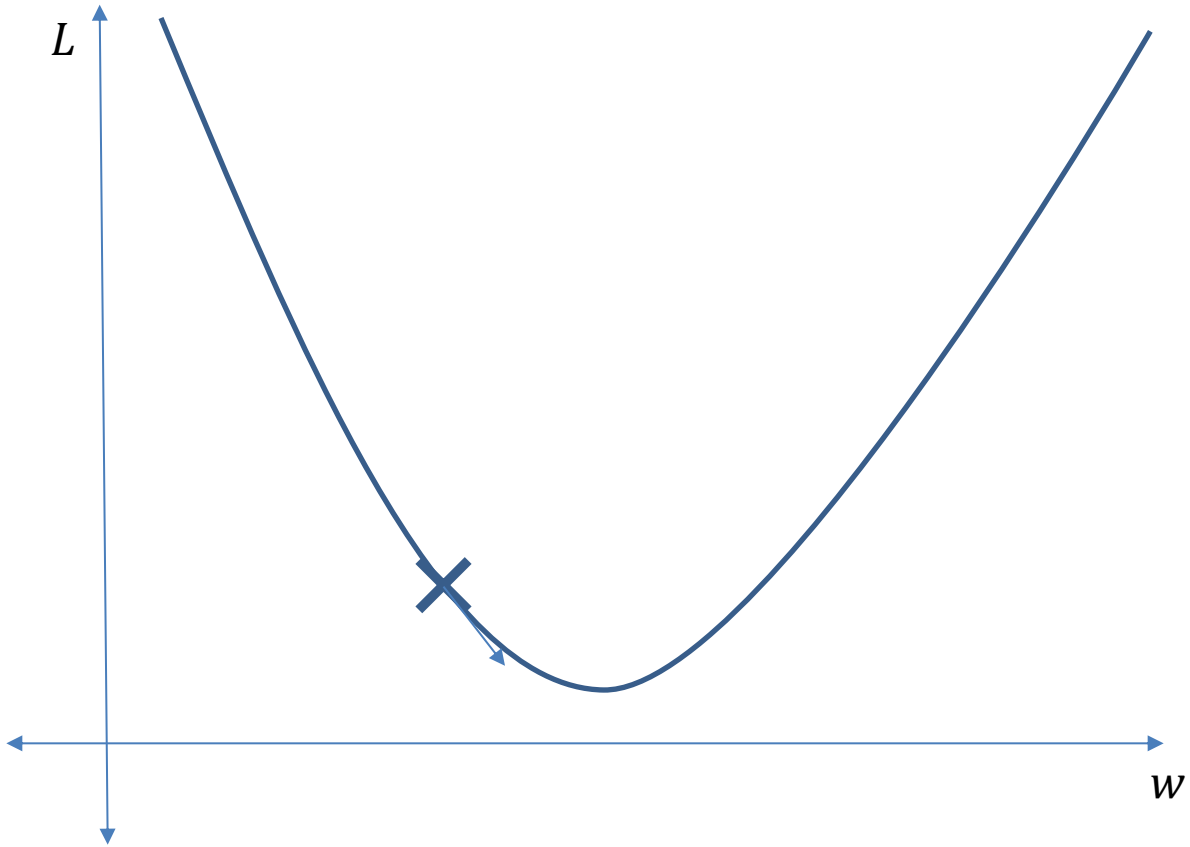
Gradient descent visualized: Minimizing loss



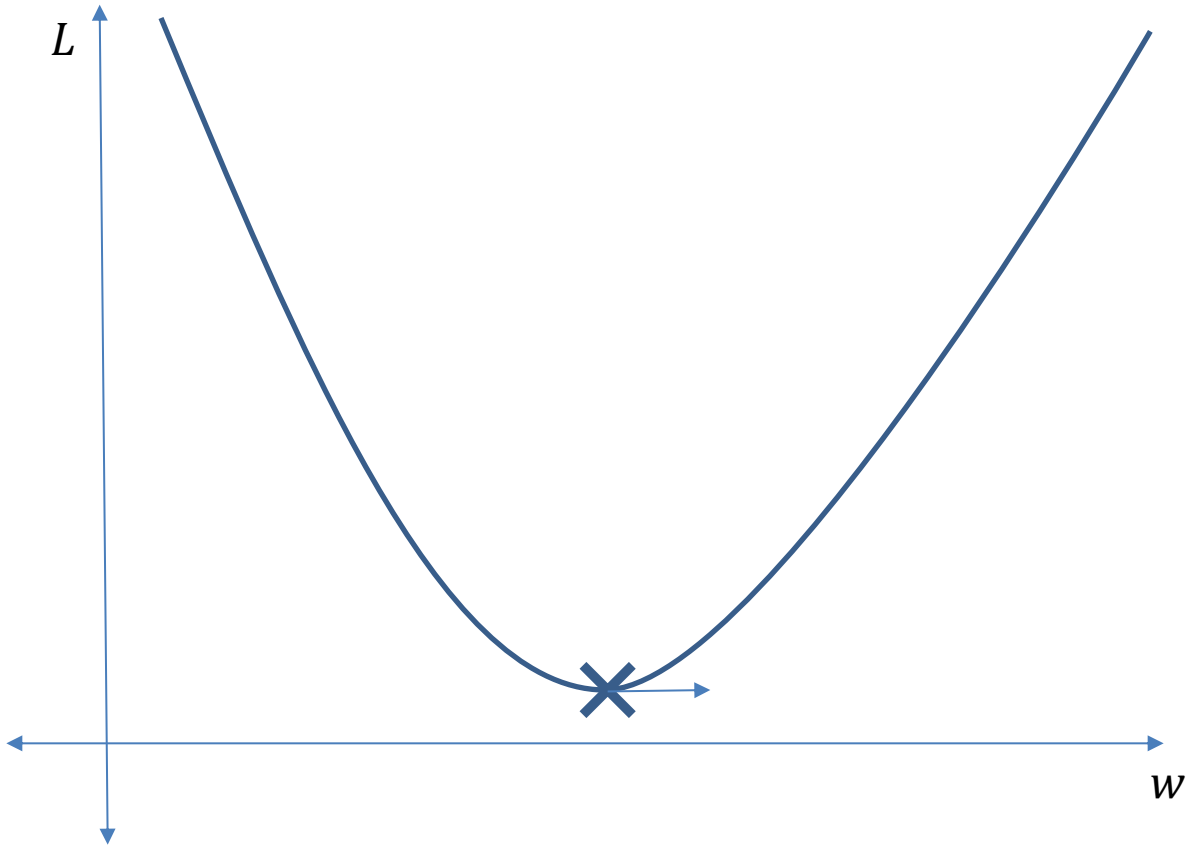
Gradient descent visualized: Minimizing loss



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:

$\hat{y}_i = f(x_i, W)$

$L \mathrel{+}= L_i(y_i, \hat{y}_i)$

$\frac{dL}{dW} = ???$

$W := W - \alpha \frac{dL}{dW}$

Partial derivative of loss to update weights

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

Let's assume that the correct label is class $\mathbf{k}$, implying $\mathbf{y} = \mathbf{k}$

$$\text{Loss} = \mathrm{L}(\hat{\mathrm{y}}, \mathrm{y}) = -\log \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

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

Calculating the loss $\frac{dL}{dW}$ is hard mathematically. But we can use the chain rule to make it simpler:

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

Partial derivative of loss to update weights

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

Let's assume that the correct label is class $\mathbf{k}$, implying $\mathbf{y} = \mathbf{k}$

$$\mathbf{Loss} = -\hat{\mathbf{y}}_{\mathbf{k}} + \log \sum_{\mathbf{j}} \mathbf{e}^{\hat{\mathbf{y}}_{\mathbf{j}}}$$

Now, we want to update the weights W by calculating the direction in which to change the weights to reduce the loss:

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

We know that : $\frac{\mathbf{d\hat{y}}}{\mathbf{dW}} = \mathbf{x}$ but what about $\frac{\mathbf{dL}}{\mathbf{d\hat{y}}}$?

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:

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

Partial derivative of loss to update weights

Putting it all together:

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

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

Gradient Descent Pseudocode

for _ in {0,…,num_epochs}:

$L = 0$

for x_i, y_i in data:

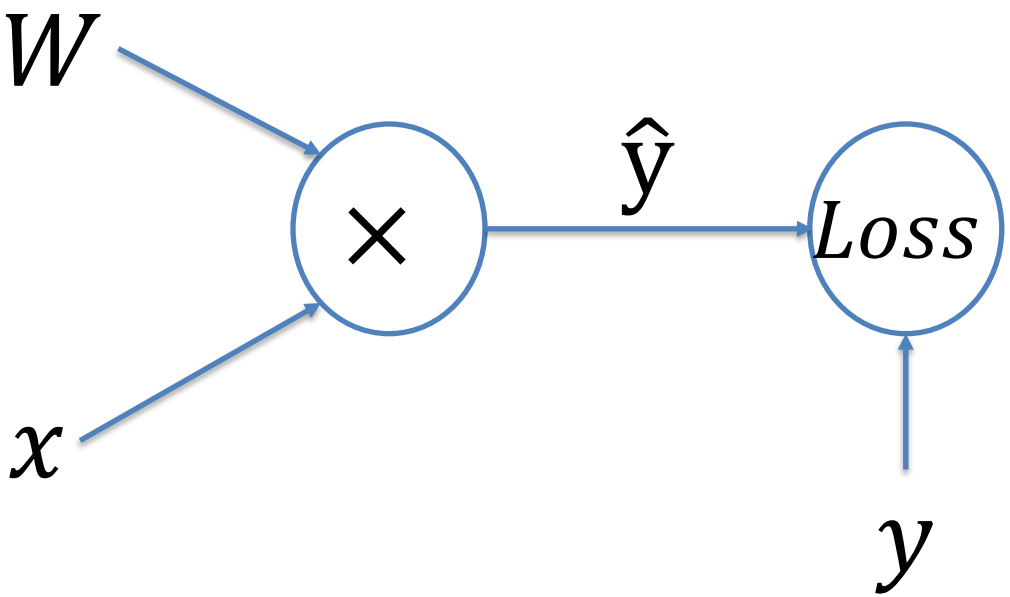
$\hat{y}_i = f(x_i, W)$

$L \mathrel{+}= L_i(y_i, \hat{y}_i)$

$\frac{dL}{dW} =$ *We know how to calculate this now!*

$W := W - \alpha \frac{dL}{dW}$

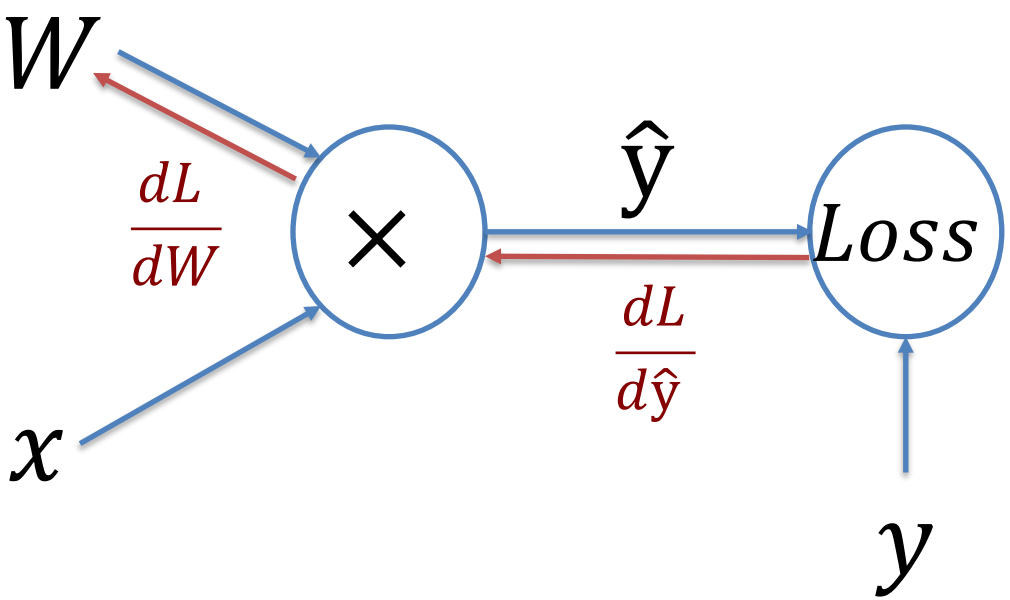
Backprop – another way of computing gradients



$$\hat{y} = Wx$$

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{dW}$$

Backprop – another way of computing gradients



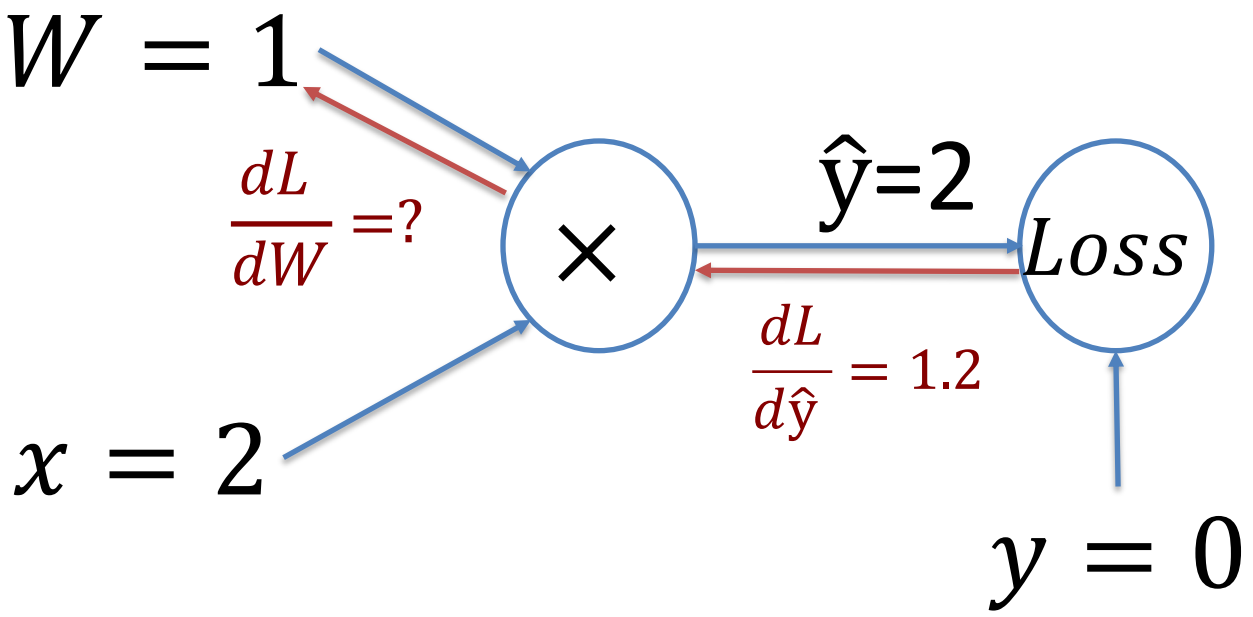
$$\hat{y} = Wx$$
$$L = Loss(\hat{y}, y)$$

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

Key Insight:

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

Backprop example in 1D:



We know that:

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

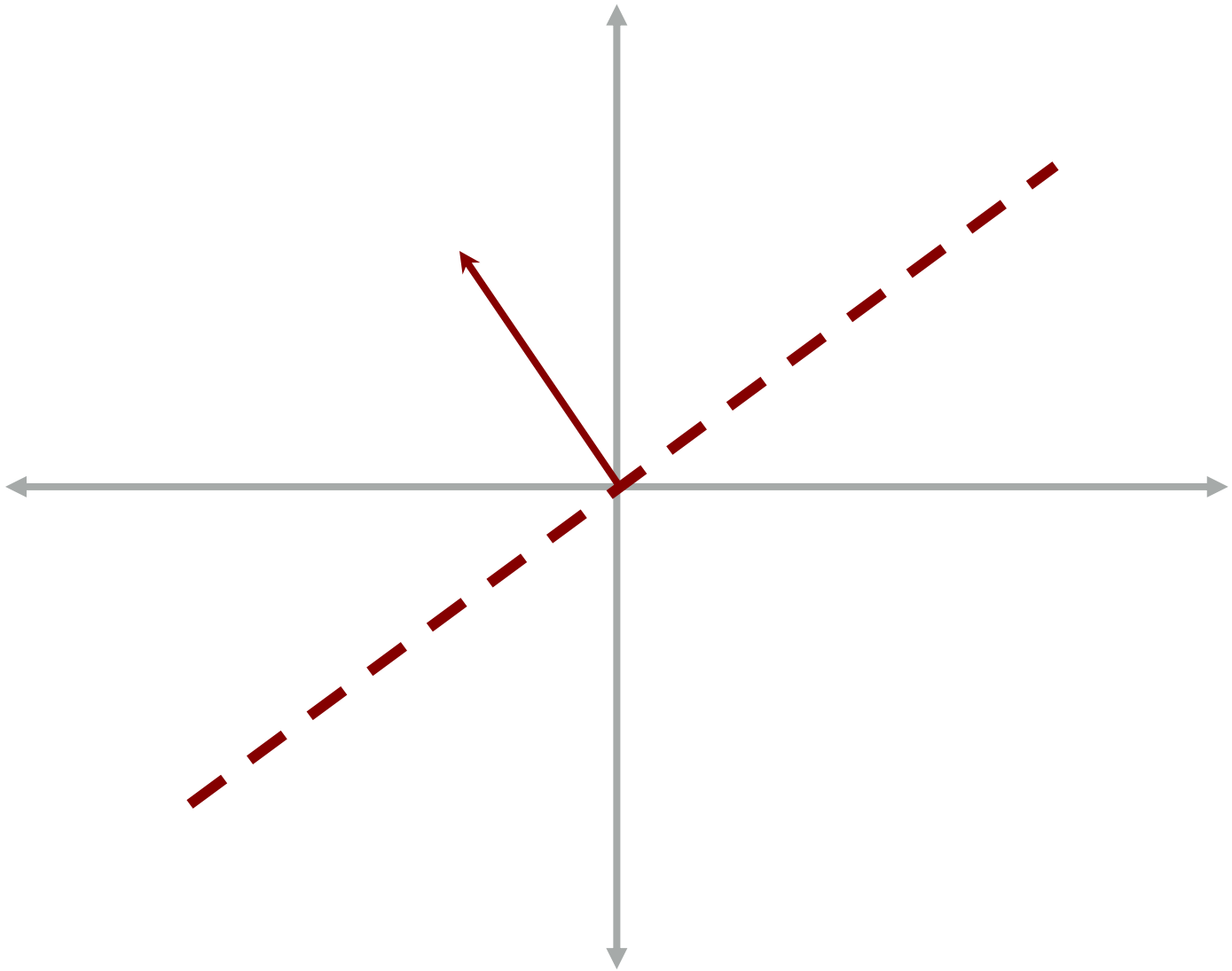
$$= \frac{dL}{d\hat{y}}x$$

$$= 1.2x$$

$$= 1.2 \times 2$$

$$= 2.4$$

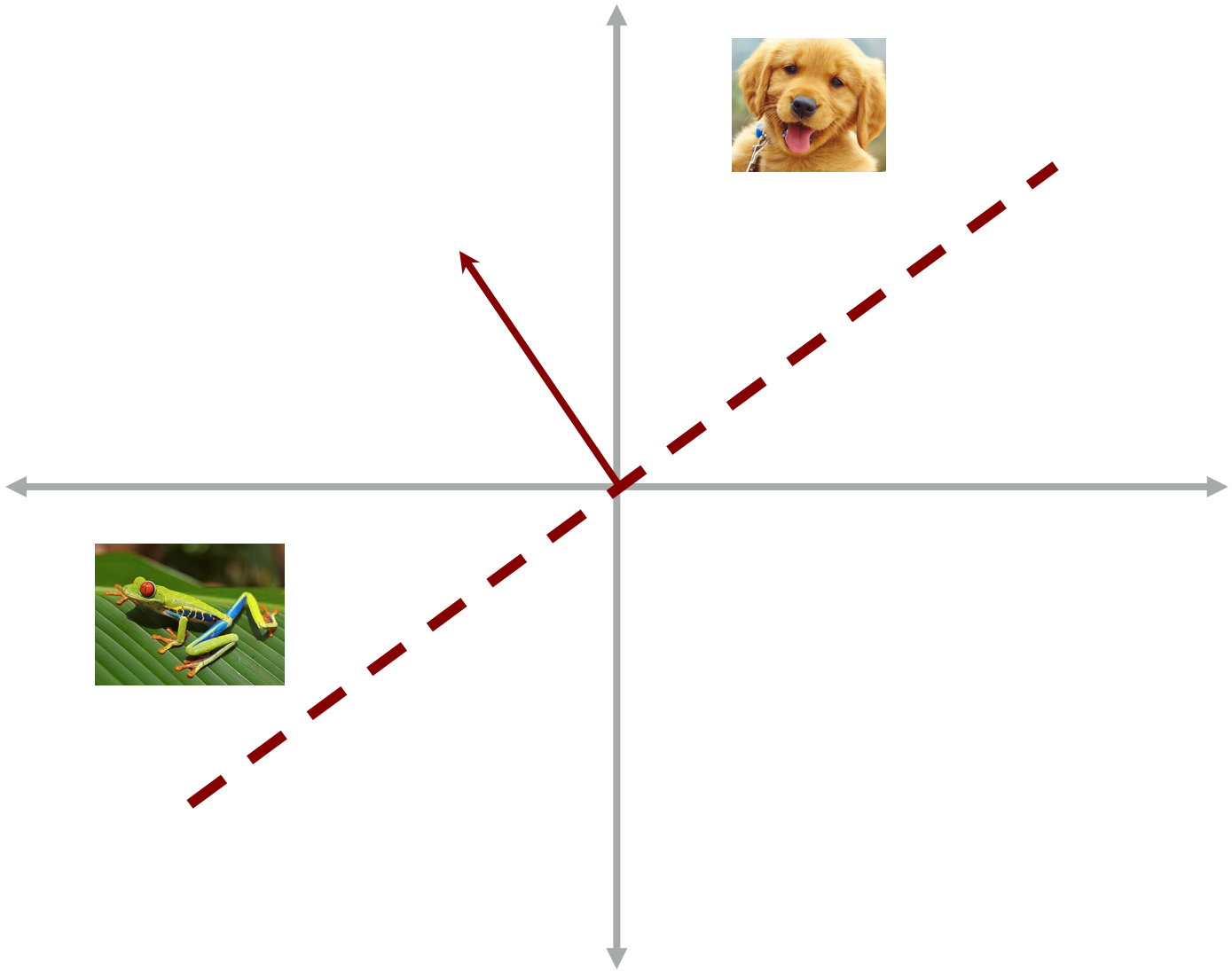
Interpreting the weights geometrically



- 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

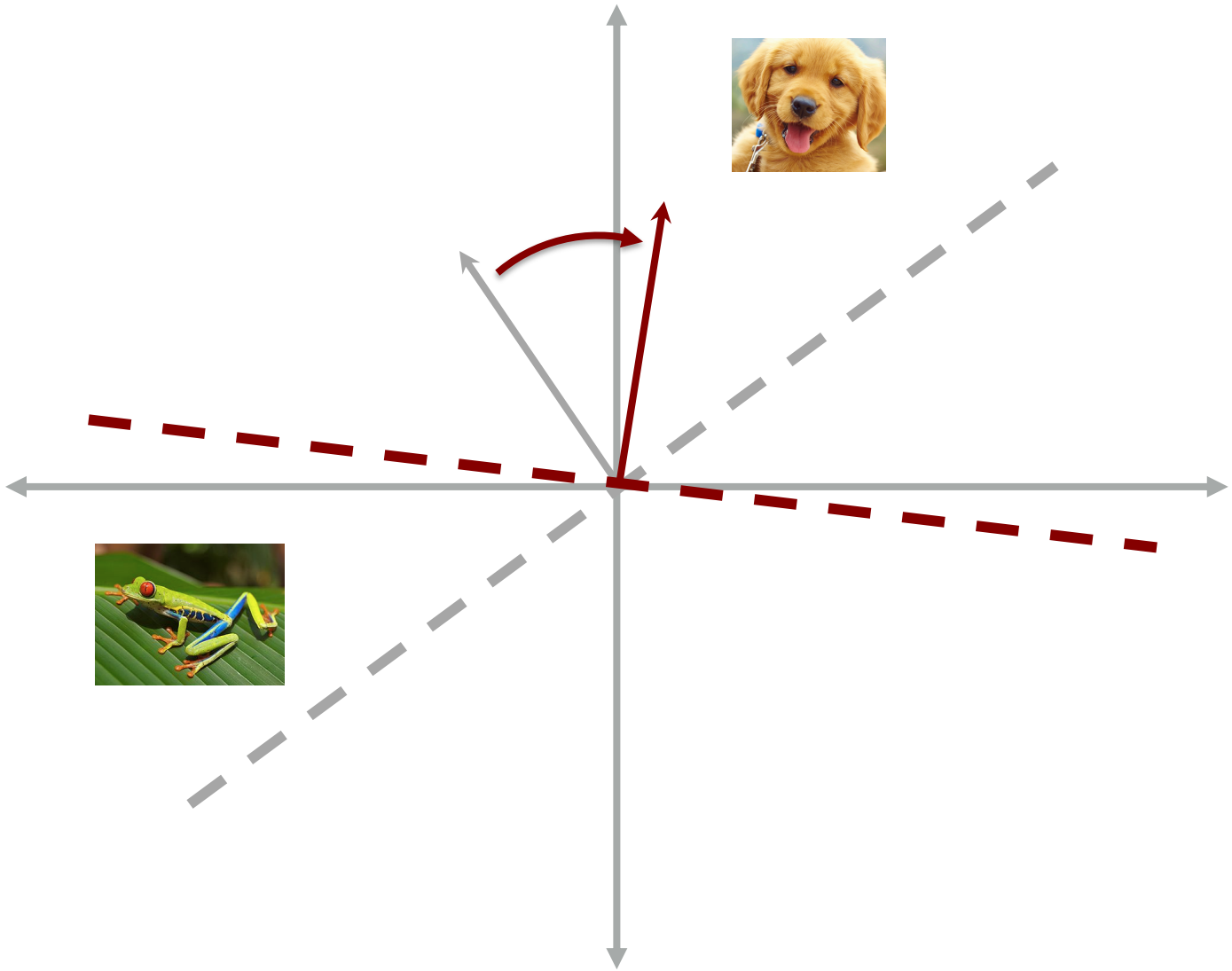
- What is the bias vector here?

Interpreting the weights geometrically



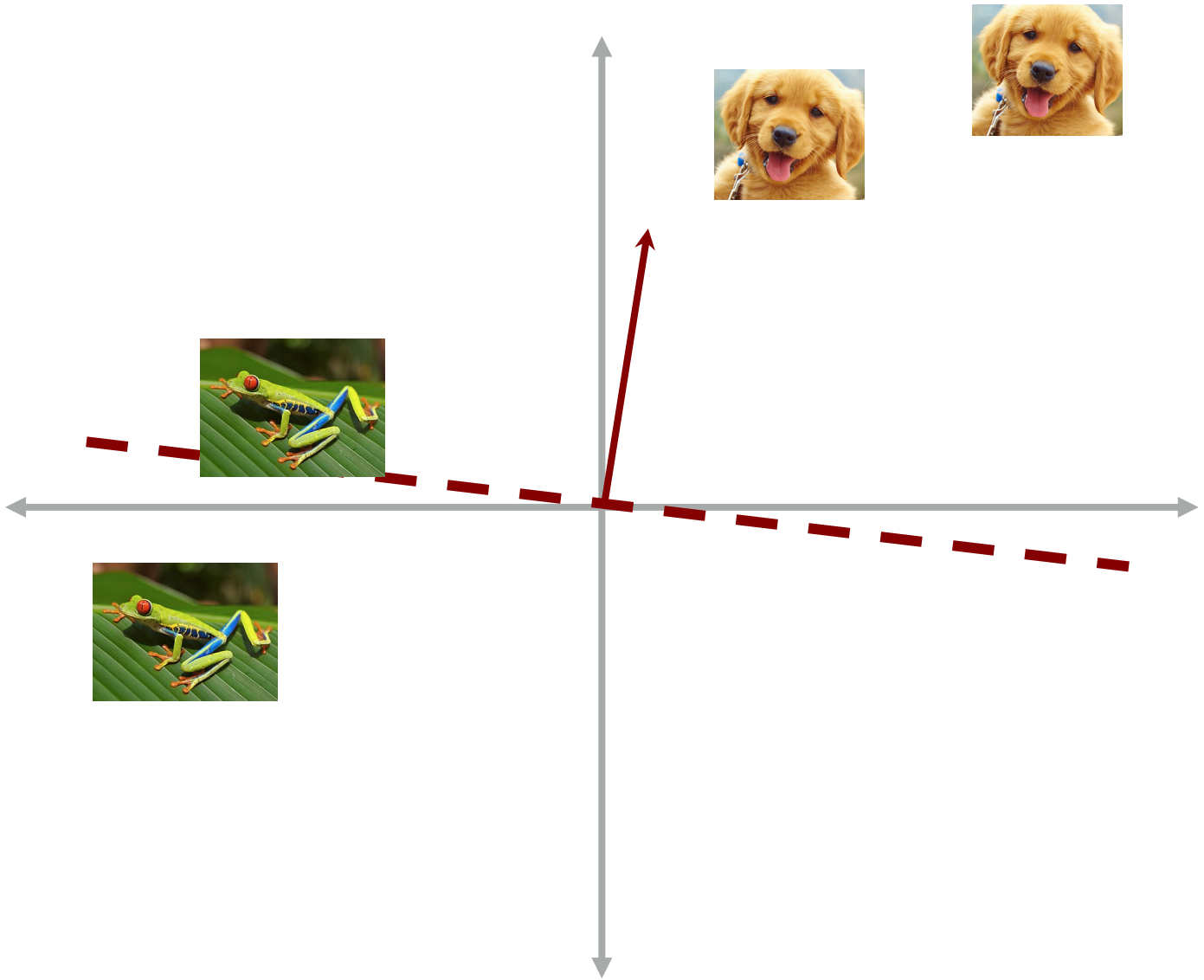
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two data points

Interpreting the weights geometrically



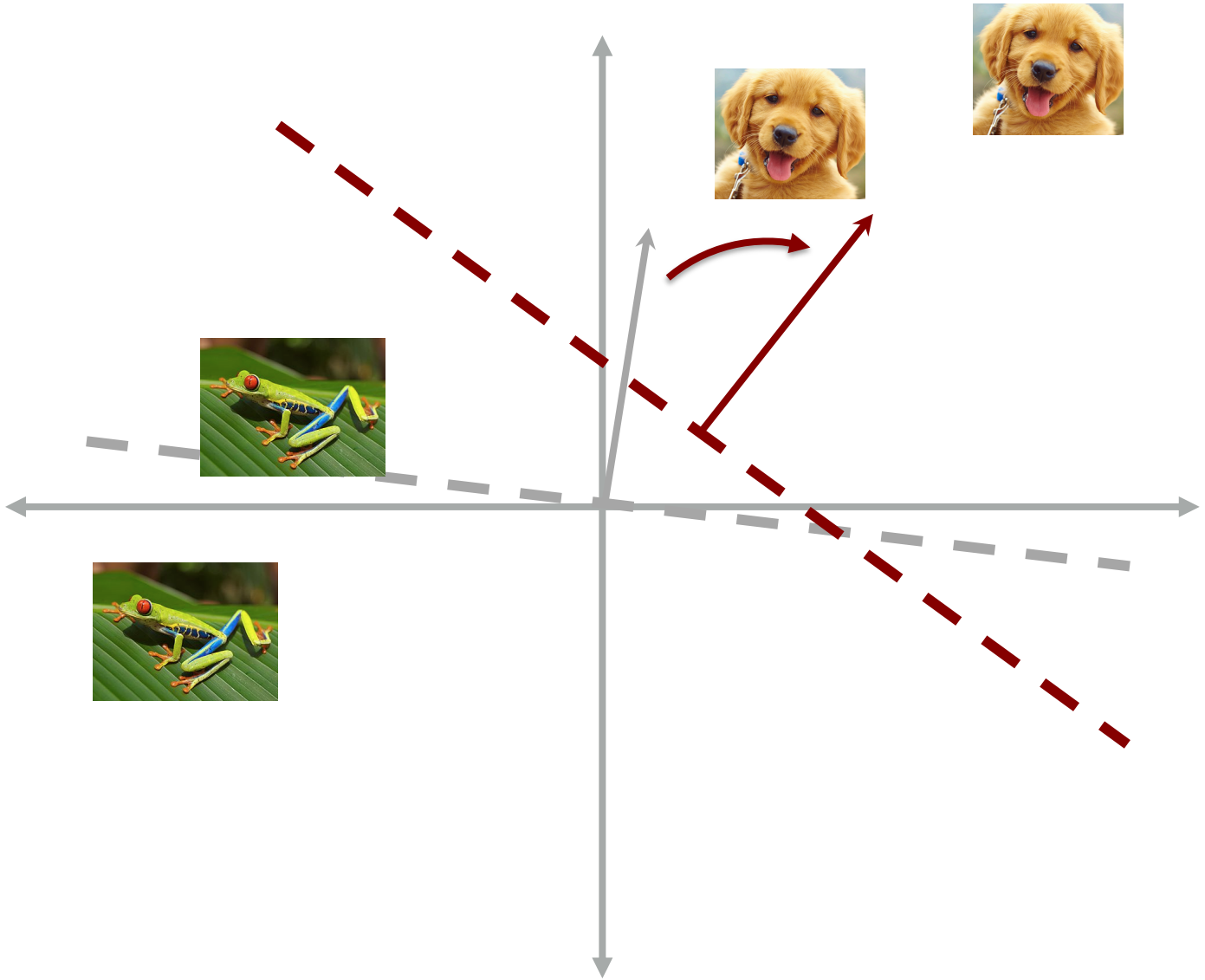
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two data points
- Update the weights

Interpreting the weights geometrically



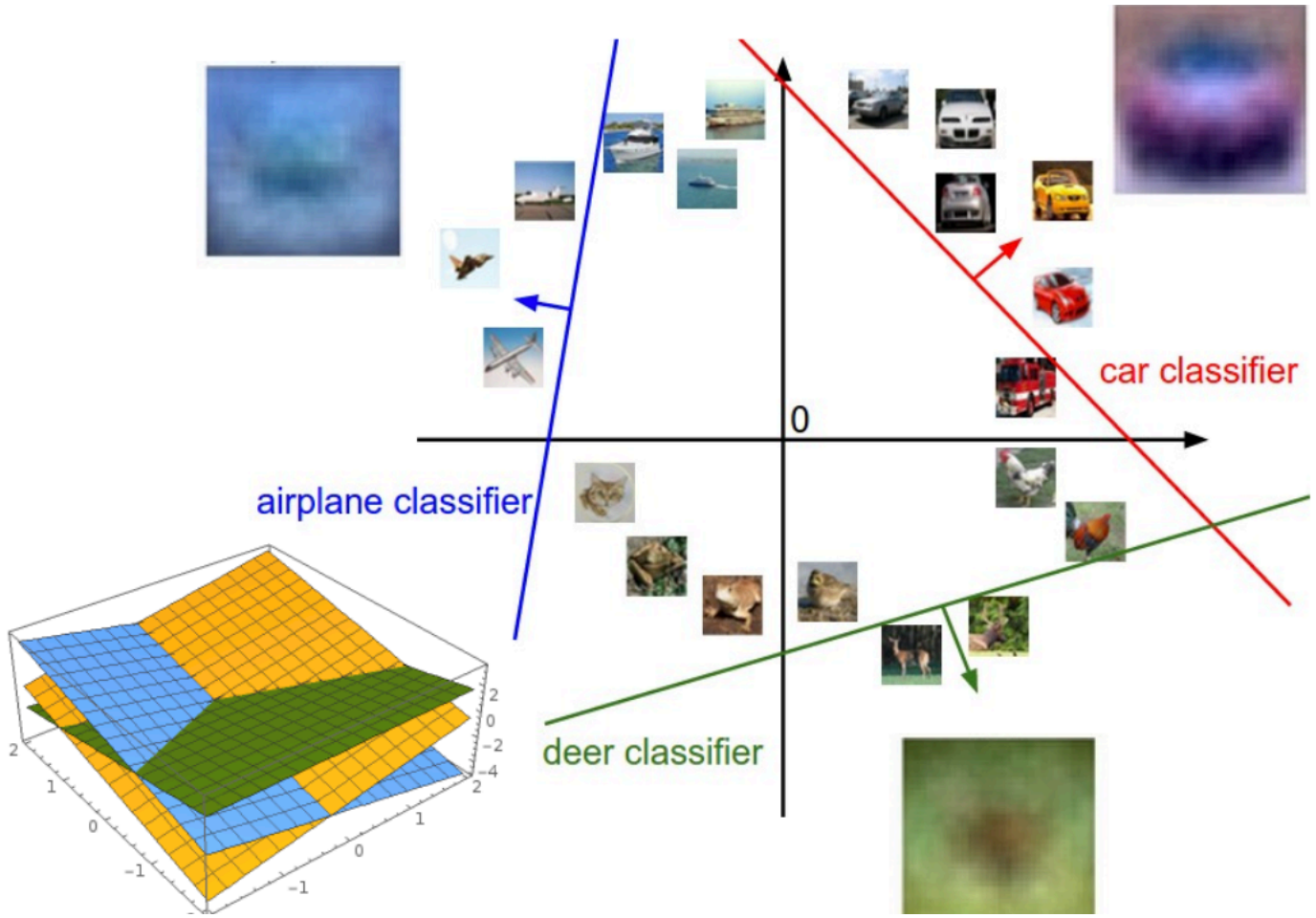
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two more data points
- Update the weights

Interpreting the weights geometrically



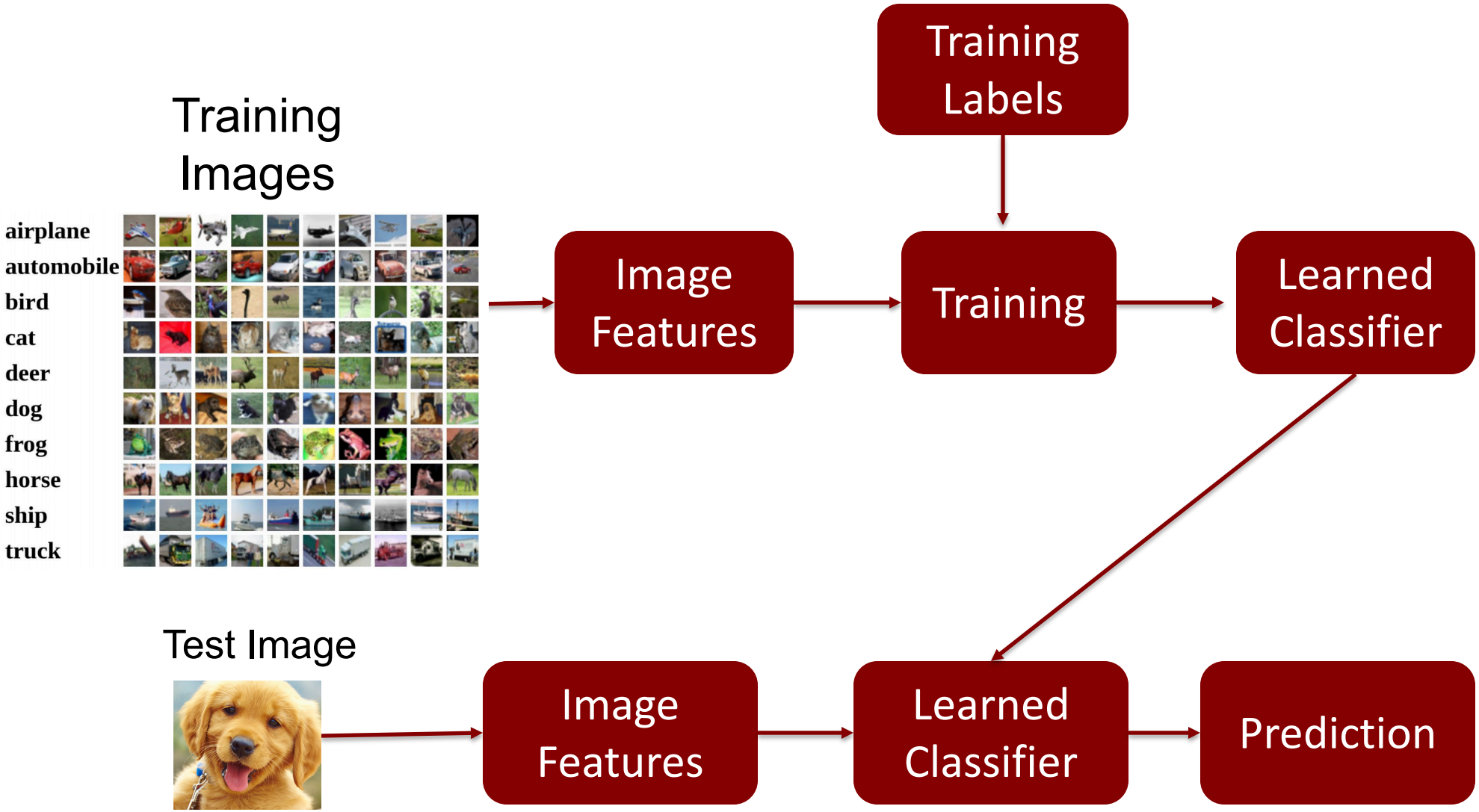
- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two more data points
- Update the weights

Interpreting the weights geometrically



Plot created using [Wolfram Cloud]

Recall: image classification pipeline

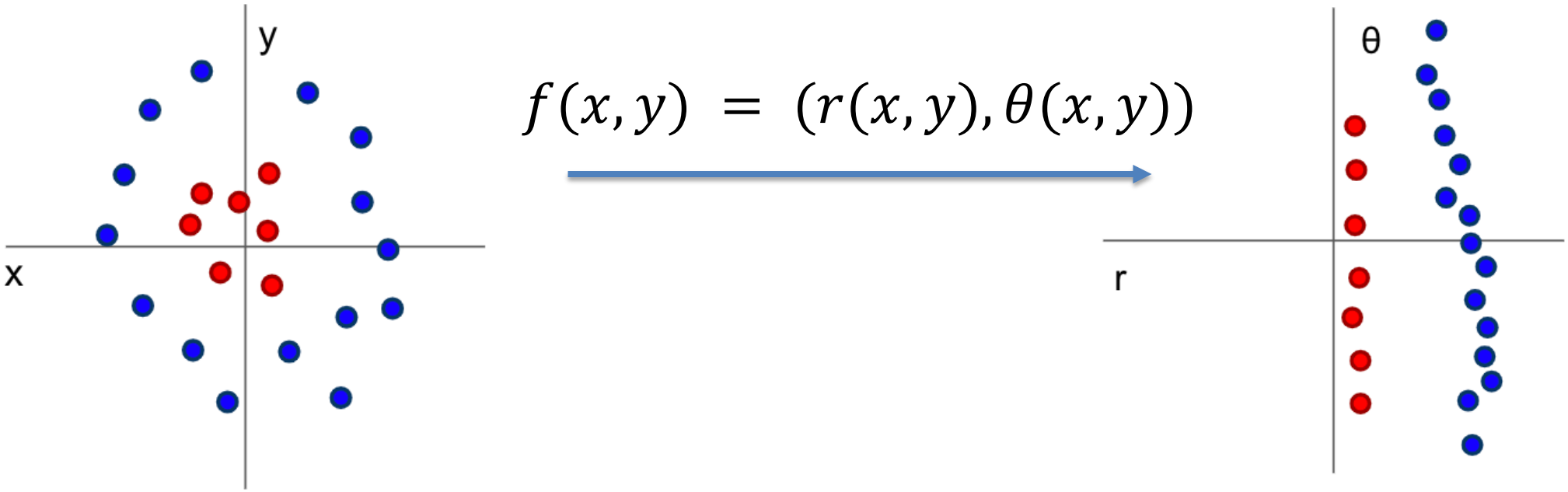


Recall: we can featurize images into a vector

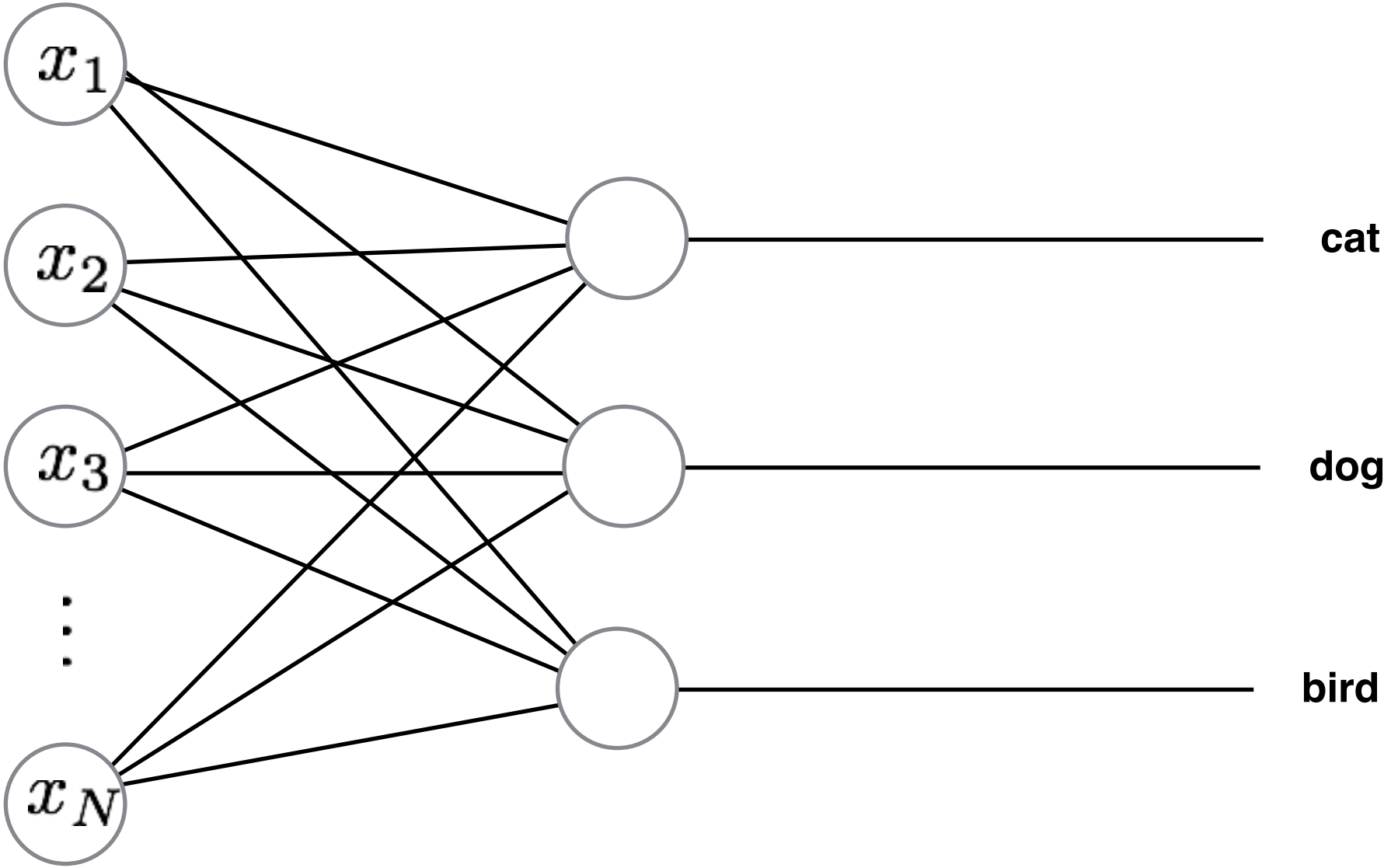
Raw pixels
Raw pixels + (x,y)
PCA
LDA
BoW
BoW + spatial pyramids

Image
Vector

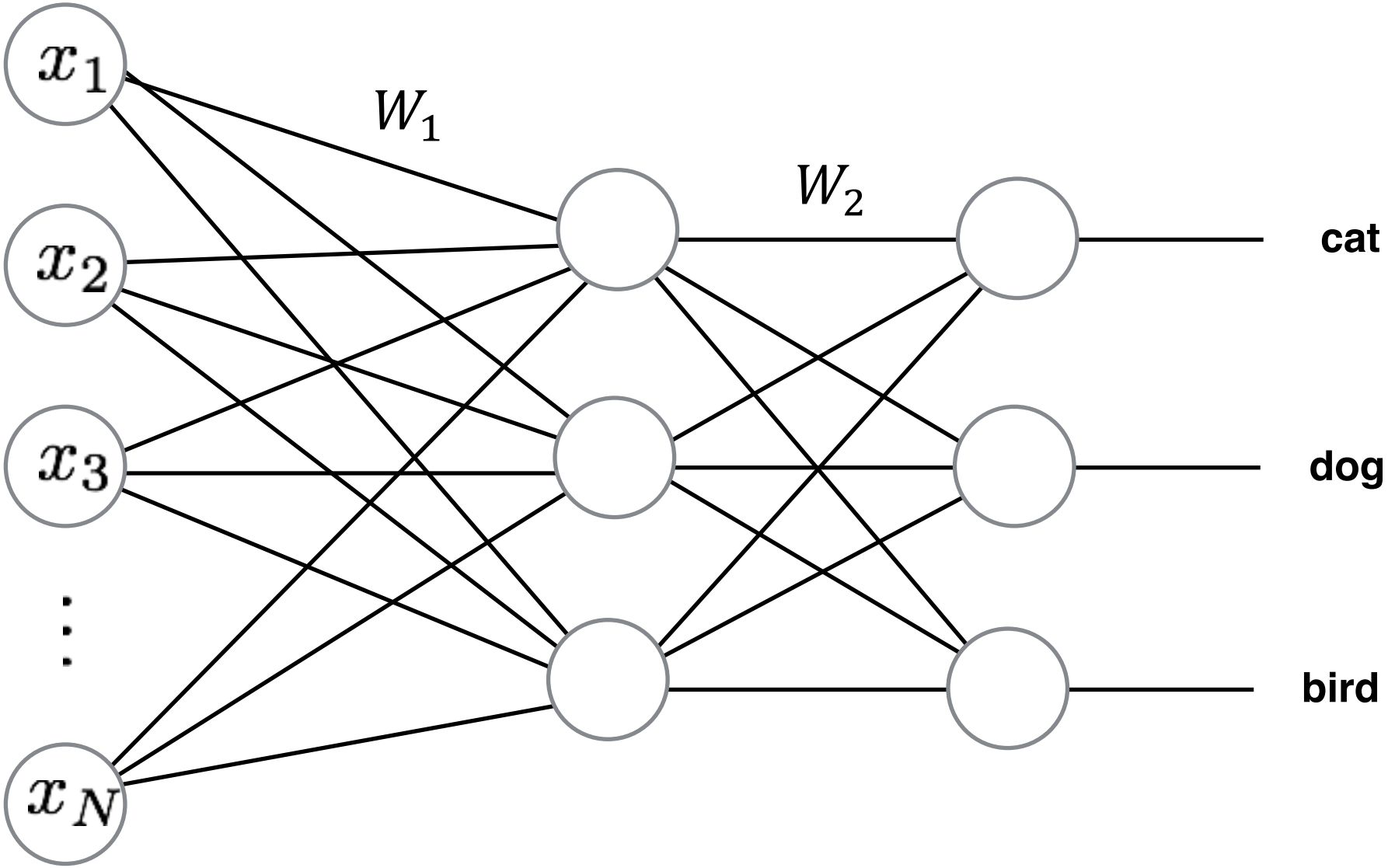
Features sometimes might not be linearly separable



Remember our linear classifier



Let's change the features by adding another layer



2-layer network: mathematical formula

- linear classifier:

$$y = Wx$$

- 2-layer network:

$$y = W_2 \max(0, W_1 x)$$

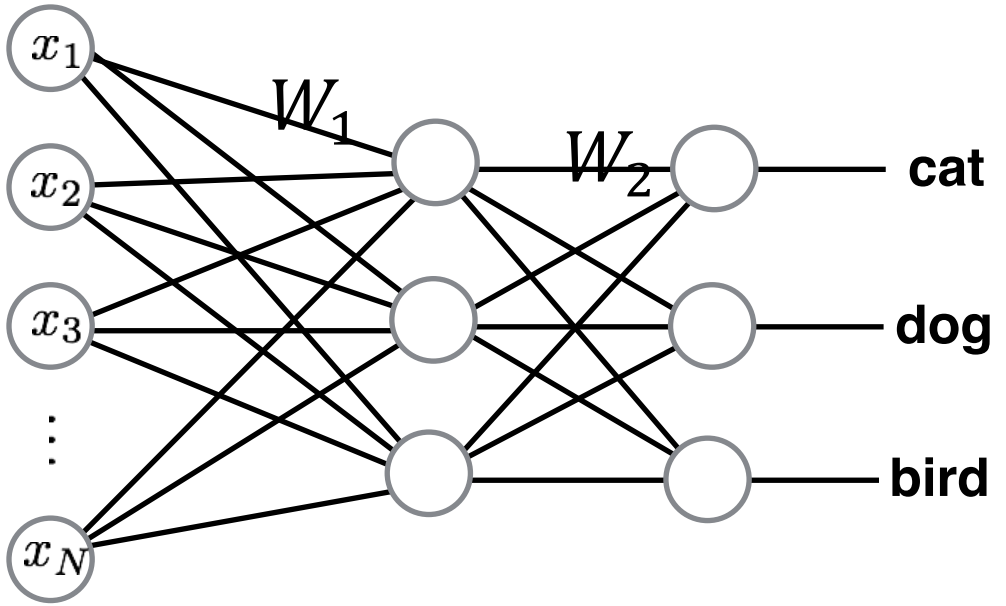
- 3-layer network:

$$y = W_3 \max(0, W_2 \max(0, W_1 x))$$

Choosing the number of layers is a new hyperparameter!

Today's agenda

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



2-layer network: mathematical formula

- linear classifier:

$$y = Wx$$

- 2-layer network:

$$y = W_2 \max(0, W_1 x)$$

We know the size of $x = 1\times3072$ and $\mathbf{y} = 10\times1$
So what are W_1 and W_2

We know that they must be: $W_1 = h\times3072$ and $W_2 = 10\times h$
h is a new hyperparameter!

2-layer network: mathematical formula

- linear classifier:

$$y = Wx$$

- 2-layer network:

$$y = W_2 \max(0, W_1 x)$$

Why is the max(0, _) necessary? Let's see what happens when we remove it:

$$y = W_2 W_1 x = \mathrm{W}x$$

where

$$\mathrm{W} = W_2 W_1$$

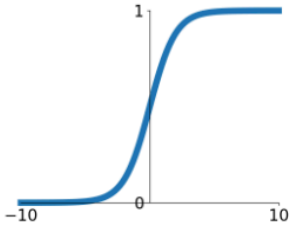
Activation function

The non-linear max function allows models to learn more complex transformations for features.

Choosing the right activation function is another new hyperparameter!

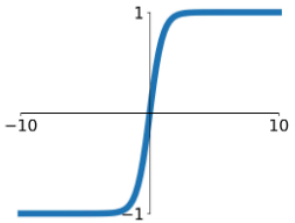
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



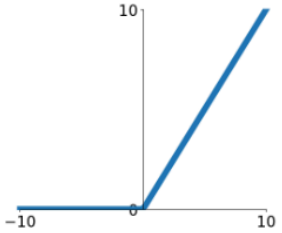
tanh

$$\tanh(x)$$



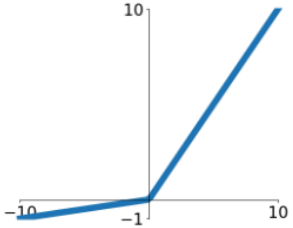
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

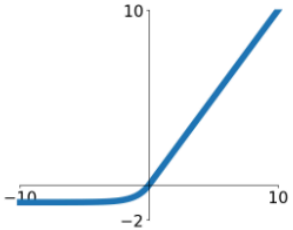


Maxout

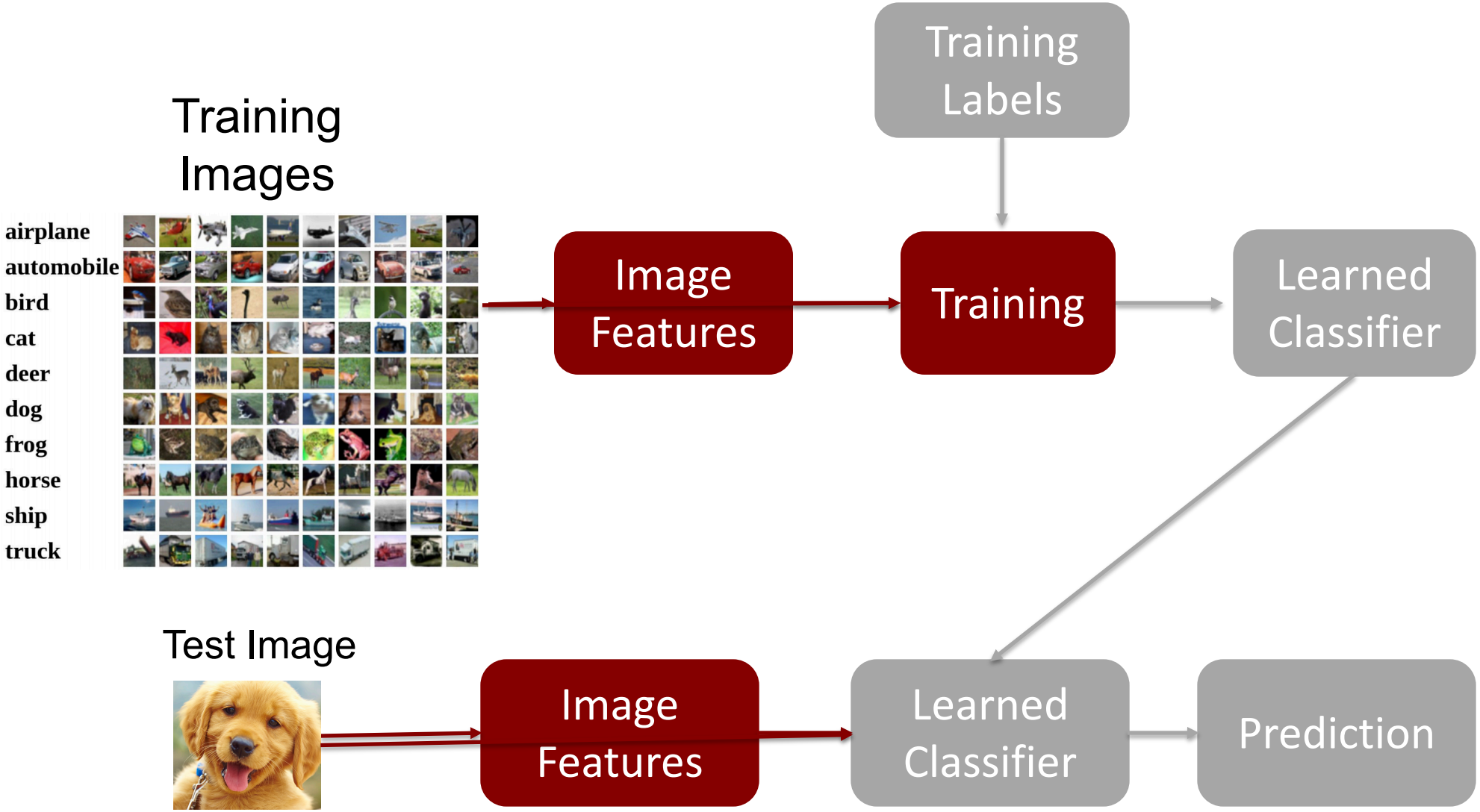
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Recall: image classification pipeline



2-layer neural network performance

- ~40% accuracy on CIFAR-10 test
  - Best class: Truck (~60%)
  - Worst class: Horse (~16%)

- Check out the model at: **https://tinyurl.com/cifar10**

Next Time…

Backpropagation with deep neural networks

Convolutional neural networks

Designing architectures