# Image Classification

# Image Classification

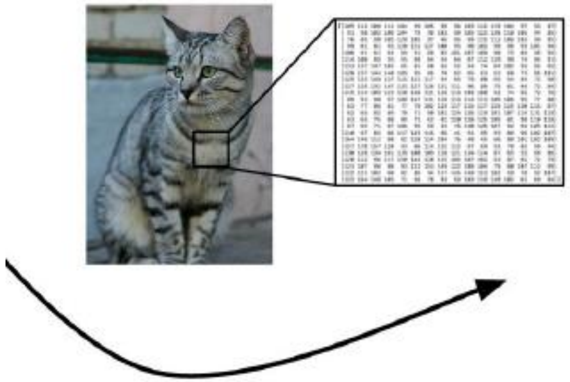


(assume given a set of labels)
{dog, cat, truck, plane, ...}

cat
dog
bird
deer
truck

# Challenges of recognition

Viewpoint



Illumination



Deformation



Occlusion

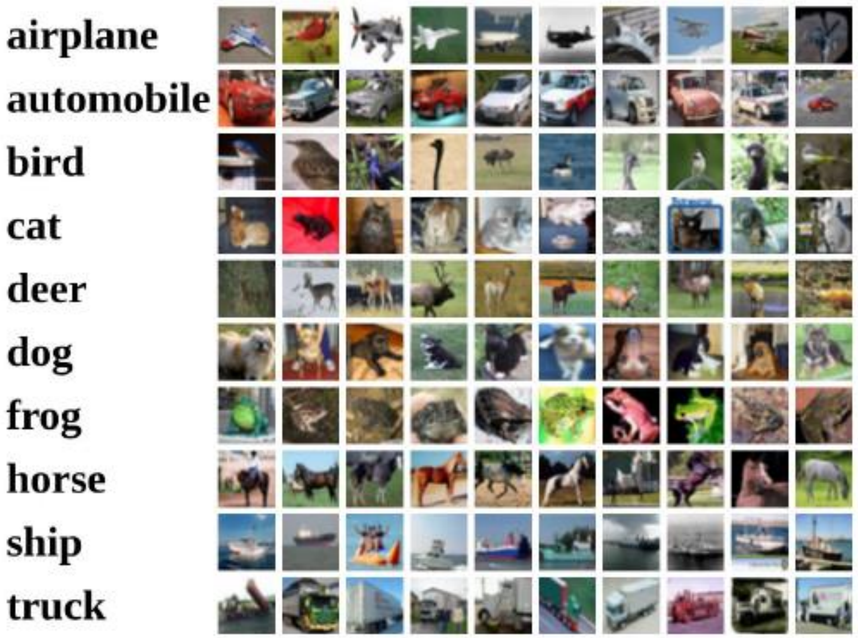


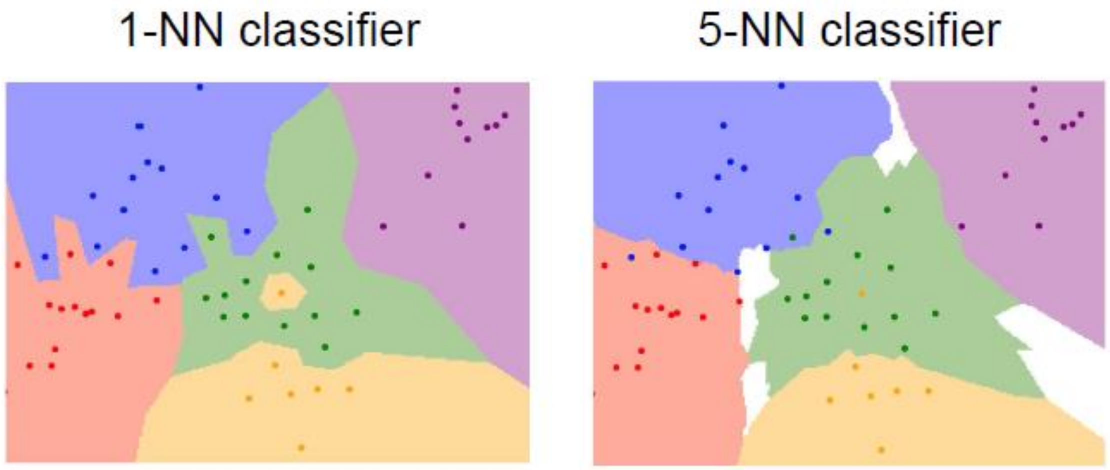
Clutter

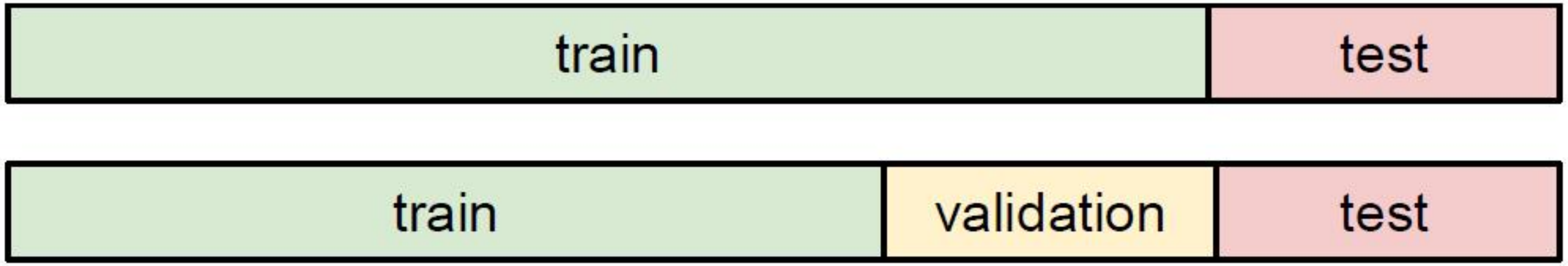


Intraclass Variation

# Data-driven approach, KNN



# Linear Classifier

Neural Network

Linear classifiers

# Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(x, W)$

**10** numbers giving
class scores

W

parameters
or weights

# Parametric Approach

- In image classification, the **parametric approach** refers to the use of a predefined model structure with a fixed set of parameters.

- By summarizing the information into a set of parameters, the parametric approach allows the model to be more efficient and scalable.

- Once the parameters are learned from the training data, the model can quickly make predictions for new, unseen images by applying the learned parameters to the input data.

- This is in contrast to the kNN approach, where the entire dataset needs to be stored and searched for each prediction, making it computationally expensive and less efficient for large datasets.

- We will start out with arguably the simplest possible function, a linear mapping

# Parametric Approach: Linear Classifier

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

Image

Array of **32x32x3** numbers
(3072 numbers total)

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

**10** numbers giving
class scores

W

parameters
or weights

# Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{10x1} = \underbrace{\boxed{W}}_{10x3072}\overbrace{\boxed{x}}^{3072x1}$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ f(**x**,**W**) → **10** numbers giving class scores

↑

W
parameters
or weights

# Parametric Approach: Linear Classifier

3072x1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x} + \boxed{b}$$

10x1 10x3072 10x1

Image

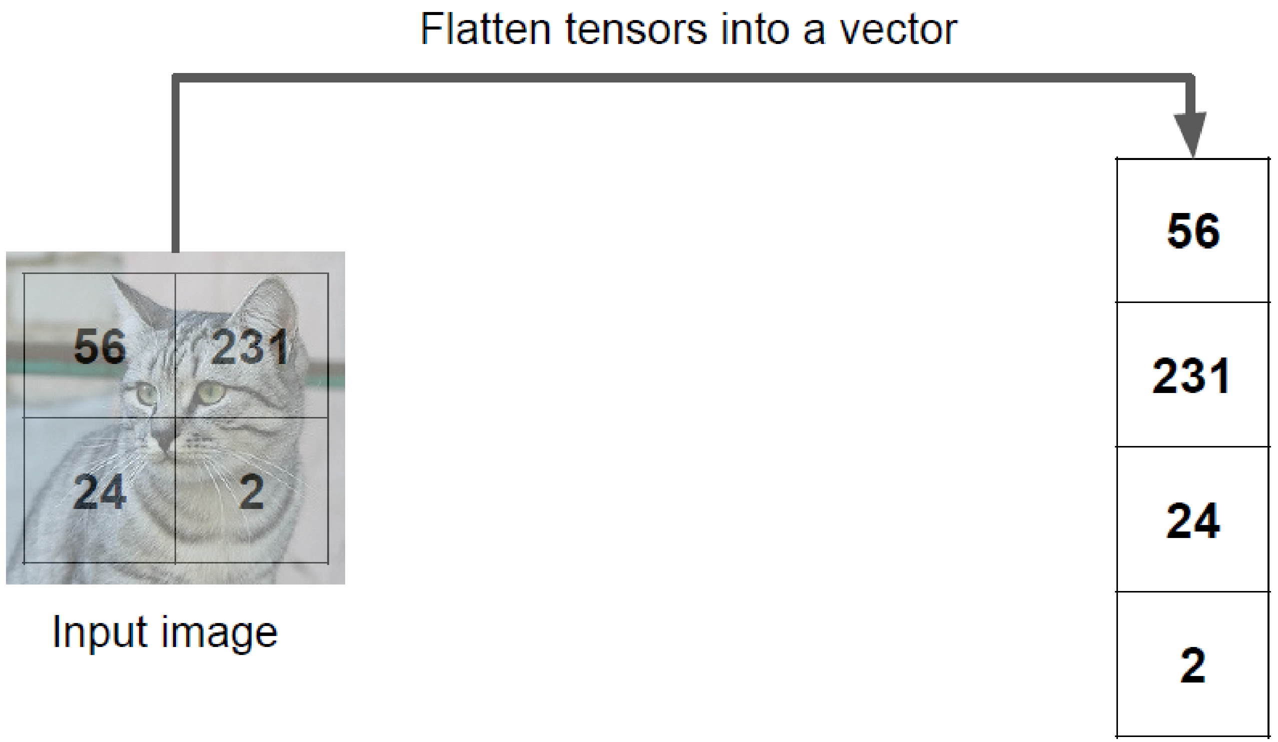
Array of **32x32x3** numbers
(3072 numbers total)

$\longrightarrow$ f(**x**,**W**) $\longrightarrow$ **10** numbers giving class scores

W

parameters
or weights

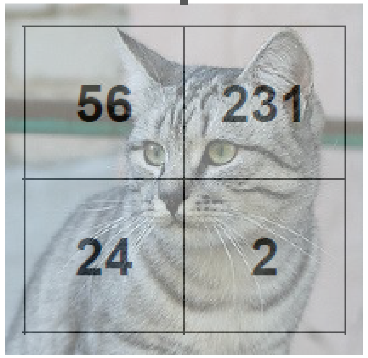
# Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

## Algebraic Viewpoint

Flatten tensors into a vector



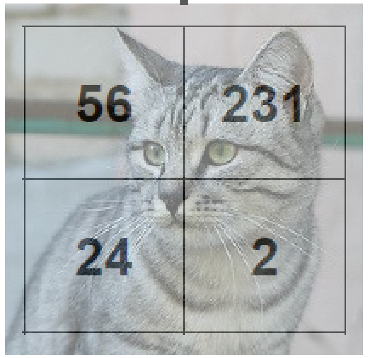

Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} =$$

# Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

## Algebraic Viewpoint

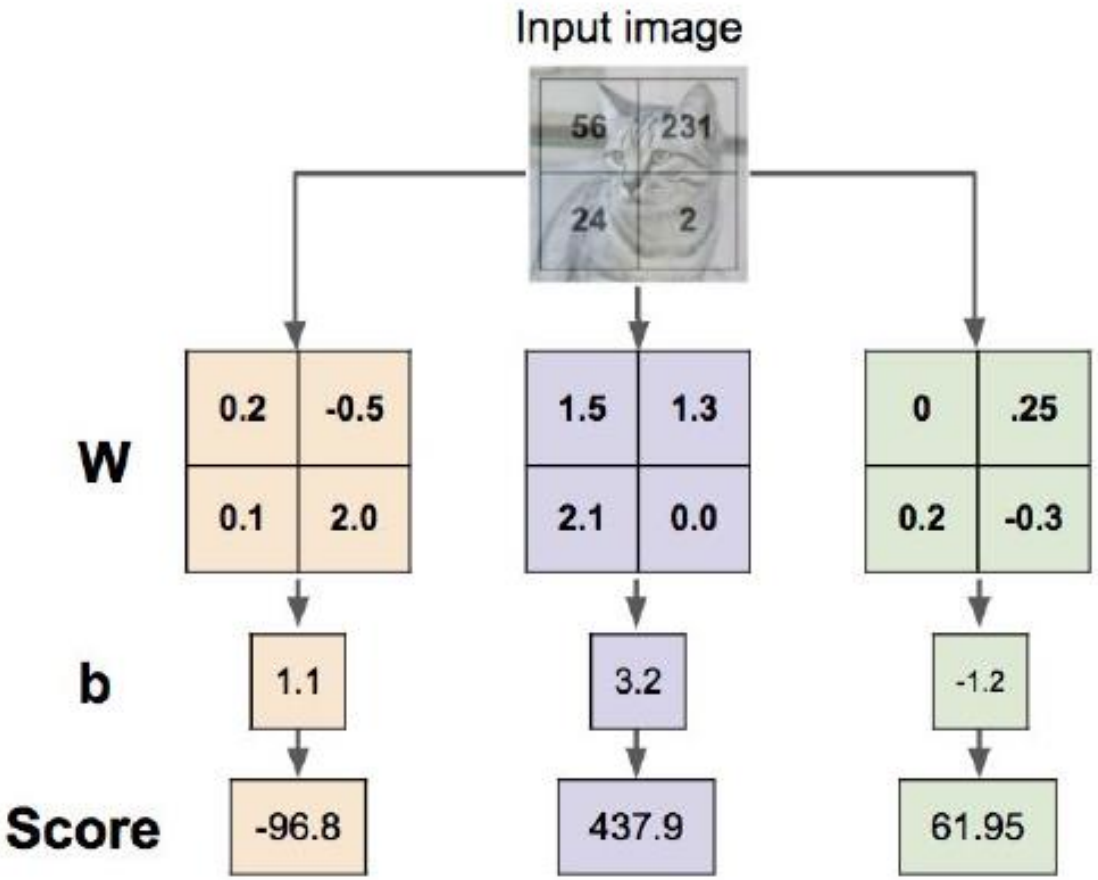
Flatten tensors into a vector




Input image

$$
\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W}
\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}
+
\underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b}
=
\begin{array}{cl} \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} & \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array} \end{array}
$$

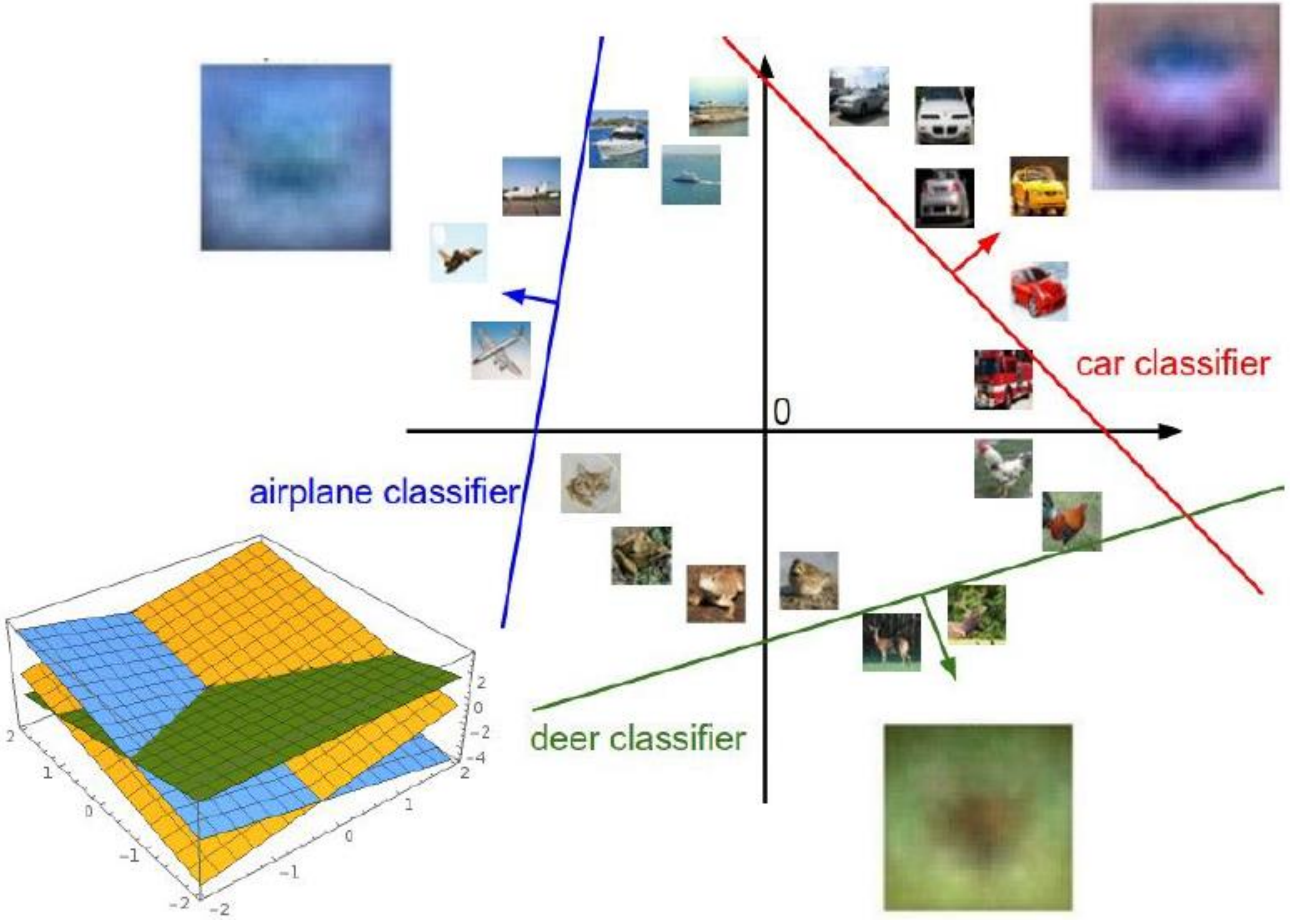
# Interpreting a Linear Classifier: Visual Viewpoint



# Interpreting a Linear Classifier: Visual Viewpoint


Example trained weights of linear classifier on CIFAR-10

# Interpreting a Linear Classifier: Geometric Viewpoint




$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

# Linear classifier

- Now we need …
  - A **score function** that maps the raw data to class scores
  - Define a **loss function** that quantifies the agreement between the predicted scores and the ground truth labels.
  - Come up with a way of efficiently finding the parameters that minimize the loss function. (Optimization)

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where $x_i$ is image and $y_i$ is (integer) label

Loss over the dataset is a average of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |

## Multiclass SVM loss:

Given an example $(x_i, y_i)$
where $x_i$ is the image and
where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |

**Multiclass SVM loss:**

Given an example $(x_i, y_i)$
where $x_i$ is the image and
where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Losses: | 2.9 | | |

## Multiclass SVM loss:

Given an example $(x_i, y_i)$
where $x_i$ is the image and
where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Losses: | 2.9 | 0 | |

## Multiclass SVM loss:

Given an example $(x_i, y_i)$
where $x_i$ is the image and
where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Losses: | 2.9 | 0 | 12.9 |

## Multiclass SVM loss:

Given an example $(x_i, y_i)$
where $x_i$ is the image and
where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Losses: | 2.9 | 0 | 12.9 |

## Multiclass SVM loss:

Given an example $(x_i, y_i)$
where $x_i$ is the image and
where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Loss over full dataset is average:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 12.9)/3
= **5.27**

# Softmax classifier (Multinomial Logistic Regression)

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ Softmax Function

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

| | | | |
|---|---|---|---|
| cat | **3.2** | | **24.5** |
| car | 5.1 | $\xrightarrow{\text{exp}}$ | 164.0 |
| frog | -1.7 | | 0.18 |

unnormalized probabilities

# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

| | | | | | |
|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** |
| car | 5.1 | $\xrightarrow{\text{exp}}$ | 164.0 | $\xrightarrow{\text{normalize}}$ | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |

unnormalized probabilities

probabilities

# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

| | Unnormalized log-probabilities / logits | | unnormalized probabilities | | probabilities |
|---|---|---|---|---|---|
| cat | **3.2** | exp → | **24.5** | normalize → | **0.13** |
| car | 5.1 | | 164.0 | | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |

# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

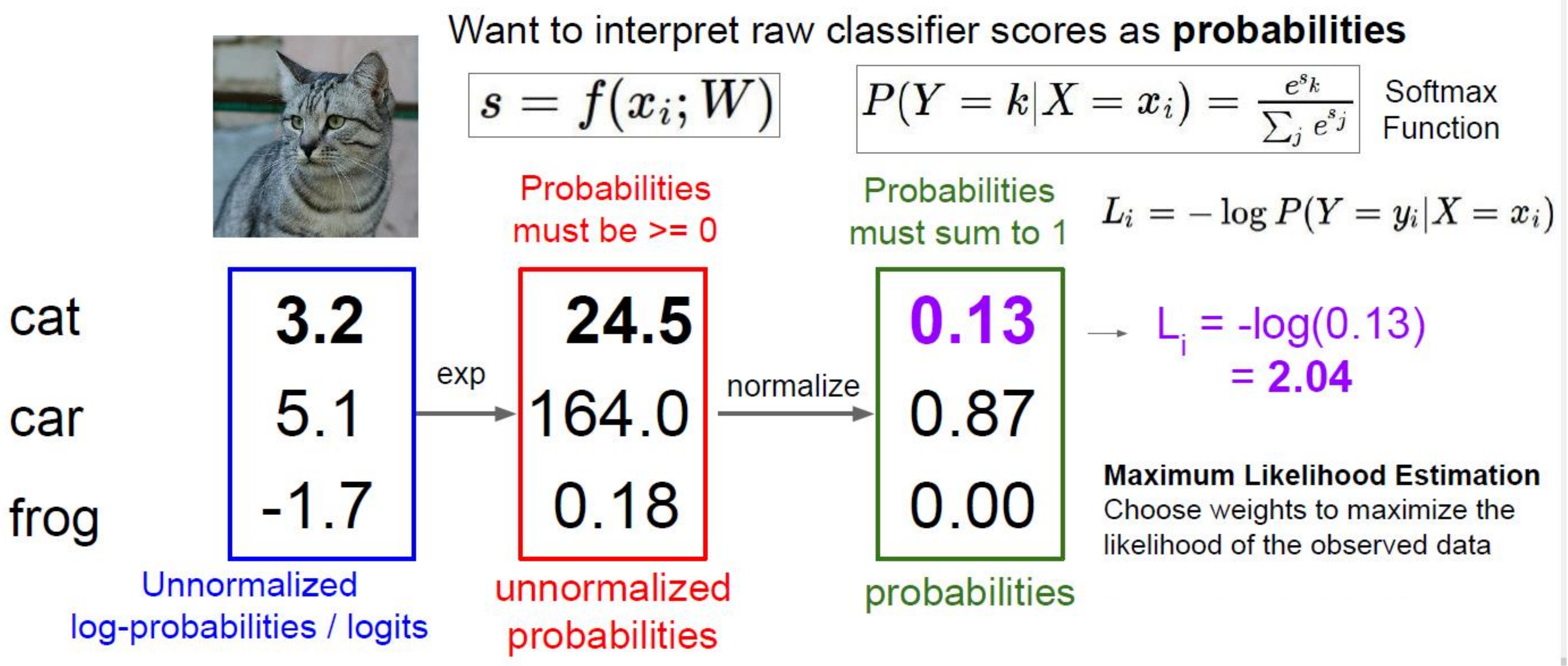
Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$

| | Unnormalized log-probabilities / logits | | unnormalized probabilities | | probabilities |
|---|---|---|---|---|---|
| cat | **3.2** | exp → | **24.5** | normalize → | **0.13** |
| car | 5.1 | | 164.0 | | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |

→ $L_i$ = -log(0.13)
= **2.04**

# Softmax classifier (Multinomial Logistic Regression)



# Softmax classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Maximize probability of correct class

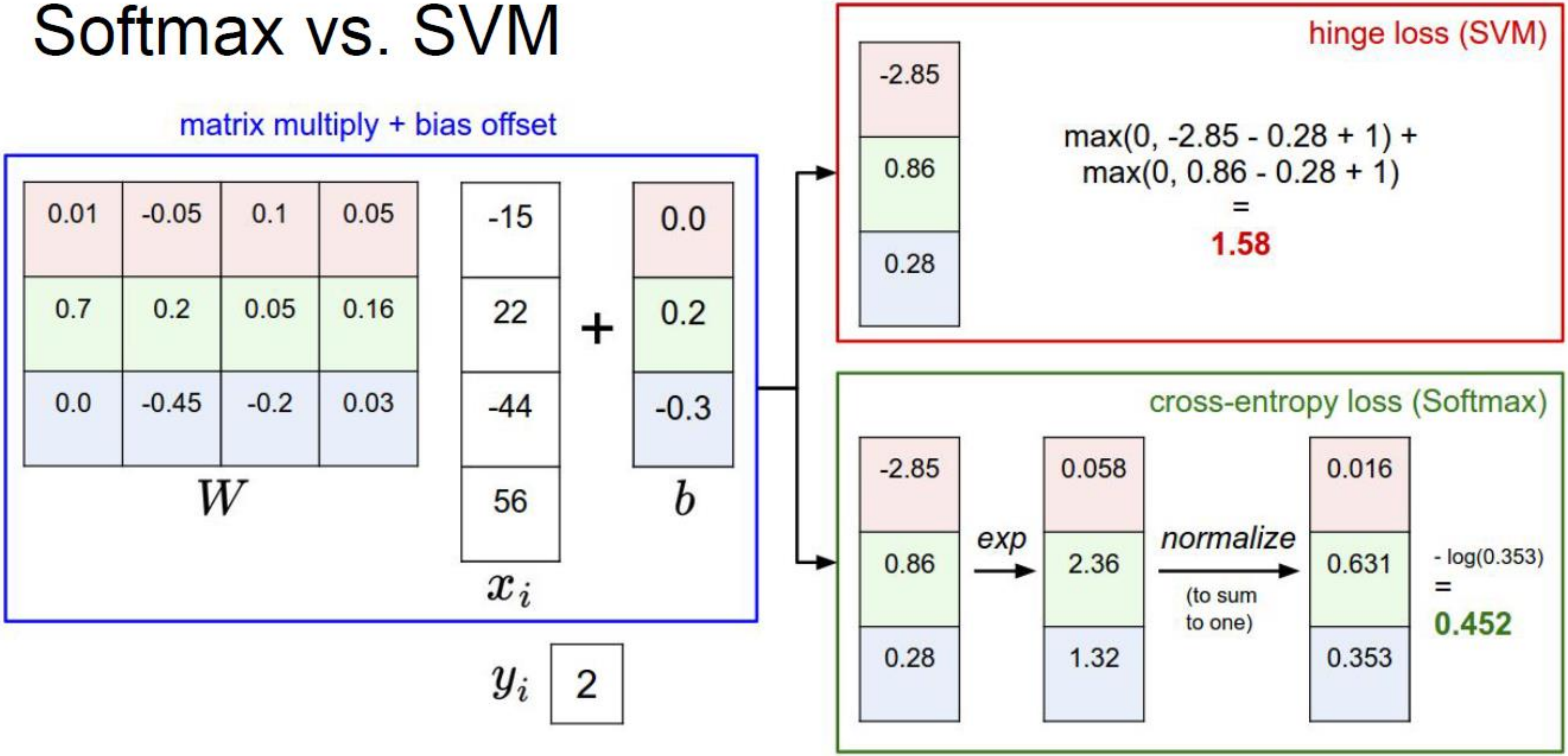
$$L_i = -\log P(Y = y_i | X = x_i)$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Putting it all together:

$$L_i = -\log\left(\frac{e^{s y_i}}{\sum_j e^{s_j}}\right)$$

# Softmax vs. SVM



# Softmax classifier (Multinomial Logistic Regression)

- In practice, SVM and Softmax are usually comparable. The performance difference between the SVM and Softmax are usually very small, and different people will have different opinions on which classifier works better.

- Compared to the Softmax classifier, the SVM is a more local objective

- the Softmax classifier is never fully happy with the scores it produces: the correct class could always have a higher probability and the incorrect classes always a lower probability and the loss would always get better.

- However, the SVM is happy once the margins are satisfied, and it does not micromanage the exact scores beyond this constraint.

# So Far …

- We defined a **score function** from image pixels to class scores (in this section, a linear function that depends on weights W and biases b).

- Unlike kNN classifier, the advantage of this **parametric approach** is that once we learn the parameters we can discard the training data. Additionally, the prediction for a new test image is fast since it requires a single matrix multiplication with W, not an exhaustive comparison to every single training example.

- We defined a **loss function** (we introduced two commonly used losses for linear classifiers: the SVM and the Softmax) that measures how compatible a given set of parameters is with respect to the ground truth labels in the training dataset. We also saw that the loss function was defined in such way that making good predictions on the training data is equivalent to having a small loss.

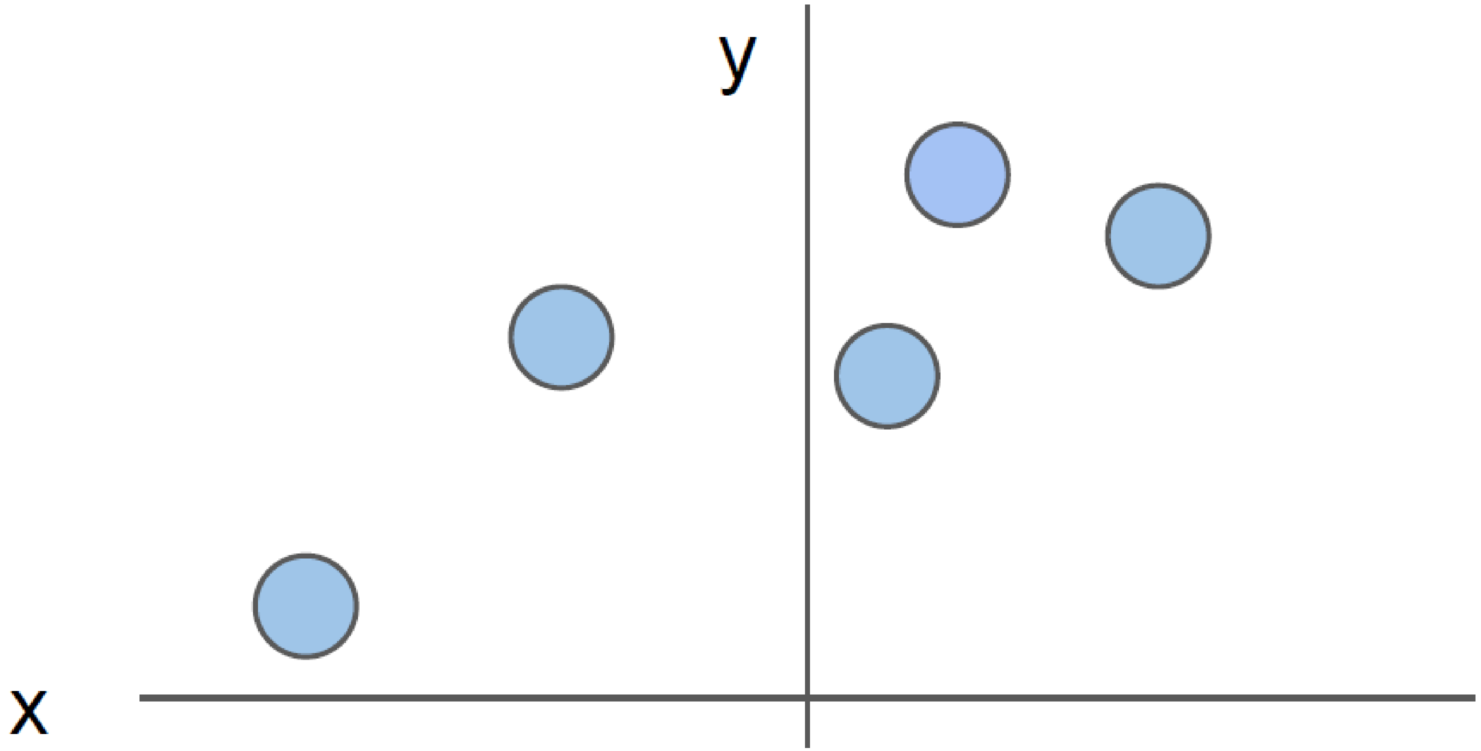
# Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

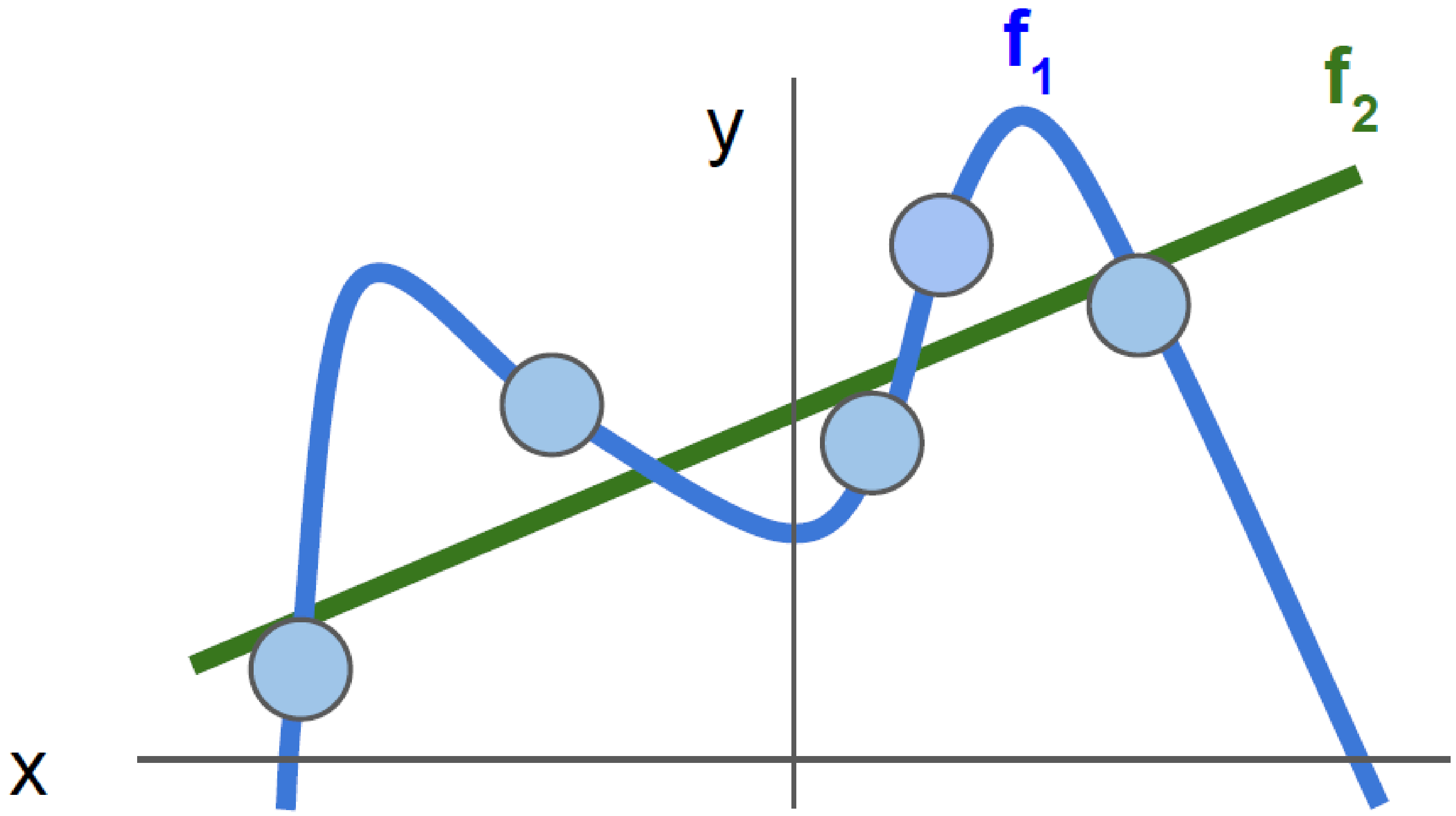
**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing *too* well on training data

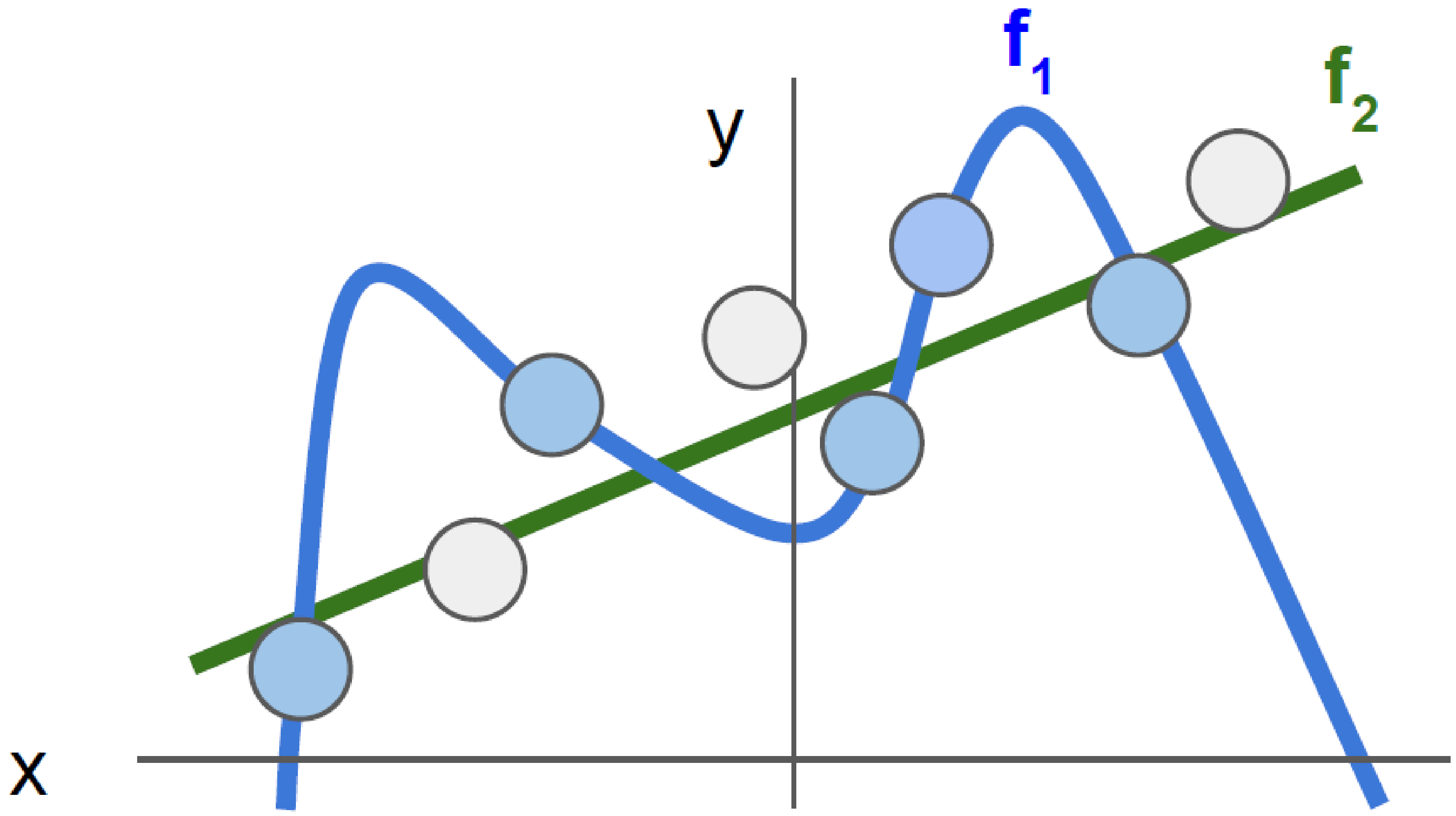
# Regularization intuition: toy example training data



# Regularization intuition: toy example training data



# Regularization intuition: toy example training data




Regularization pushes against fitting the data *too* well so we don't fit noise in the data

# Regularization

$\lambda$ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing *too* well on training data

**Simple examples**

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

**More complex**:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

# Regularization

$$x = [1, 1, 1, 1]$$
$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Which of w1 or w2 will
the L2 regularizer prefer?

# Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = \boxed{[0.25, 0.25, 0.25, 0.25]}$$

$$w_1^T x = w_2^T x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Which of w1 or w2 will the L2 regularizer prefer?

L2 regularization likes to "spread out" the weights

# Recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

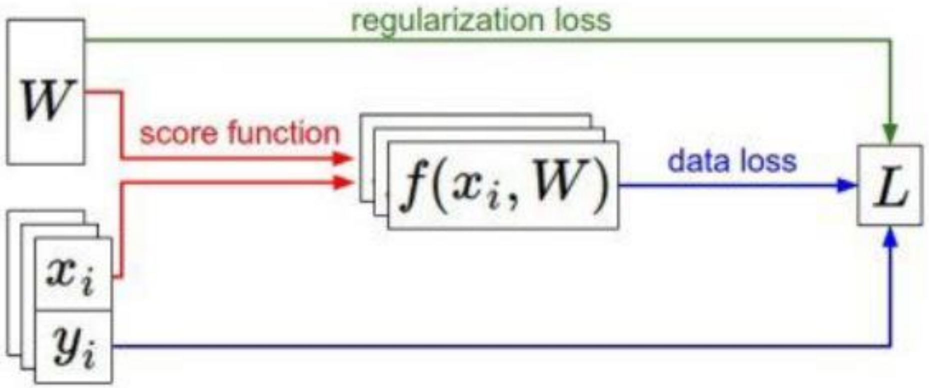
Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss



The softmax function is applied to the logits to:

a) Normalize the scores into a probability distribution

b) Increase the magnitude of the scores

c) Decrease the magnitude of the scores

d) Convert the scores into binary values