MULTICLASS CLASSIFICATION

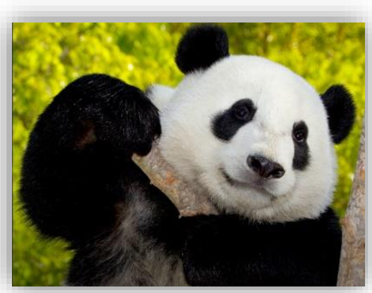
Itay Evron

Outline

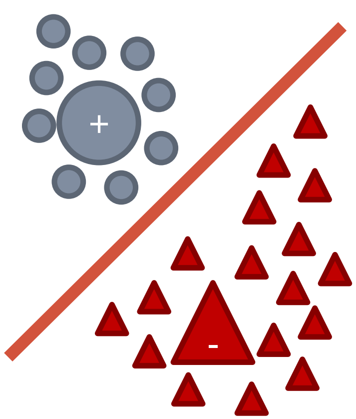
- One vs. All
- Multinomial logistic regression
  - Cross-entropy loss
- Multiclass in deep learning

Binary classification

- Goal: find a binary hypothesis $h: \mathcal{X} \rightarrow \{-1, +1\}$
- Simple, well studied and well understood, elegant.



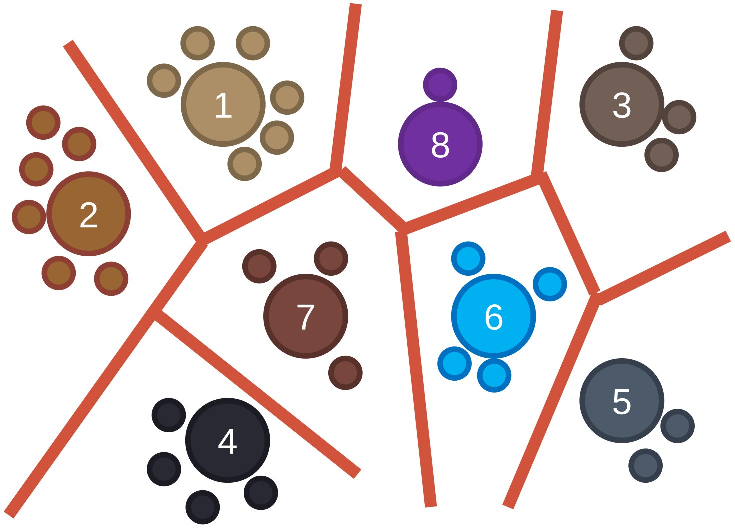
$\rightarrow$ $\boxed{h}$ $\rightarrow$ {bear, not bear}



Multiclass classification

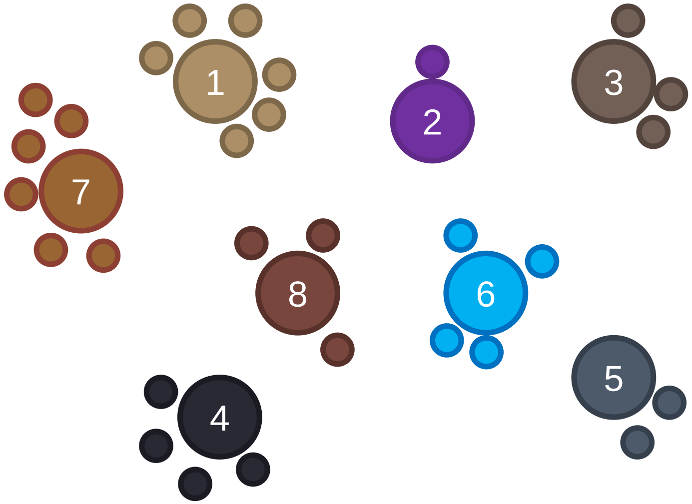
- Goal: classify into many classes $h: \mathcal{X} \to \{1,2,\ldots,K\}$
- Tasks are more specific and more complicated.

$\longrightarrow$ $\boxed{h}$ $\longrightarrow$ {black, brown, polar, panda, …}



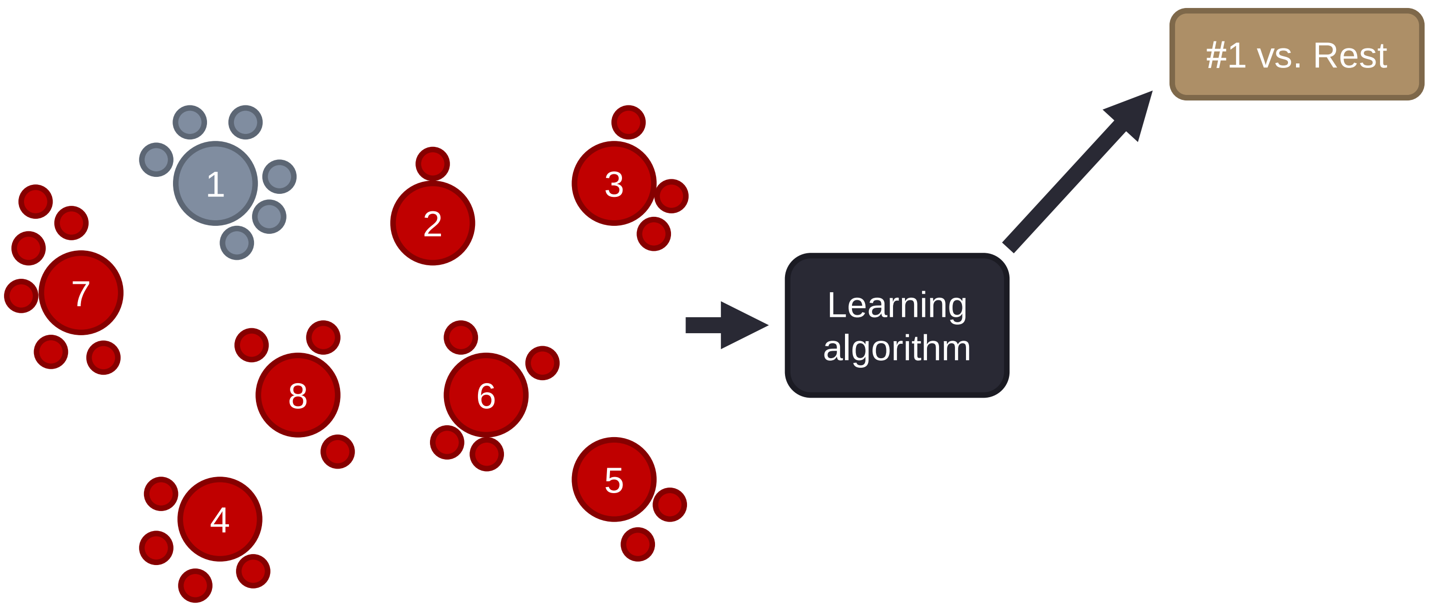
One vs. All (OVA)

- Goal: classify into many classes $h: \mathcal{X} \rightarrow \{1,2, \dots, K\}$
- Solution: reduce to K separate binary tasks.



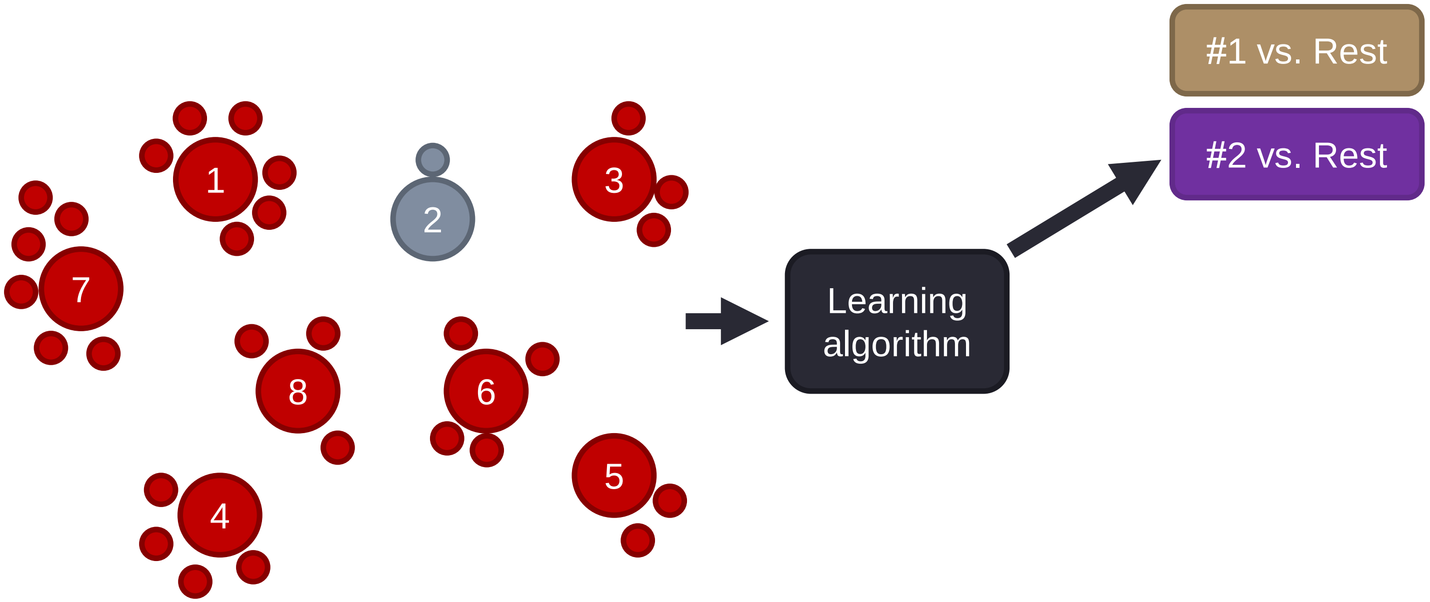
One vs. All: Training

- Goal: classify into many classes $h: \mathcal{X} \rightarrow \{1,2,\ldots,K\}$
- Solution: reduce to K separate binary tasks.



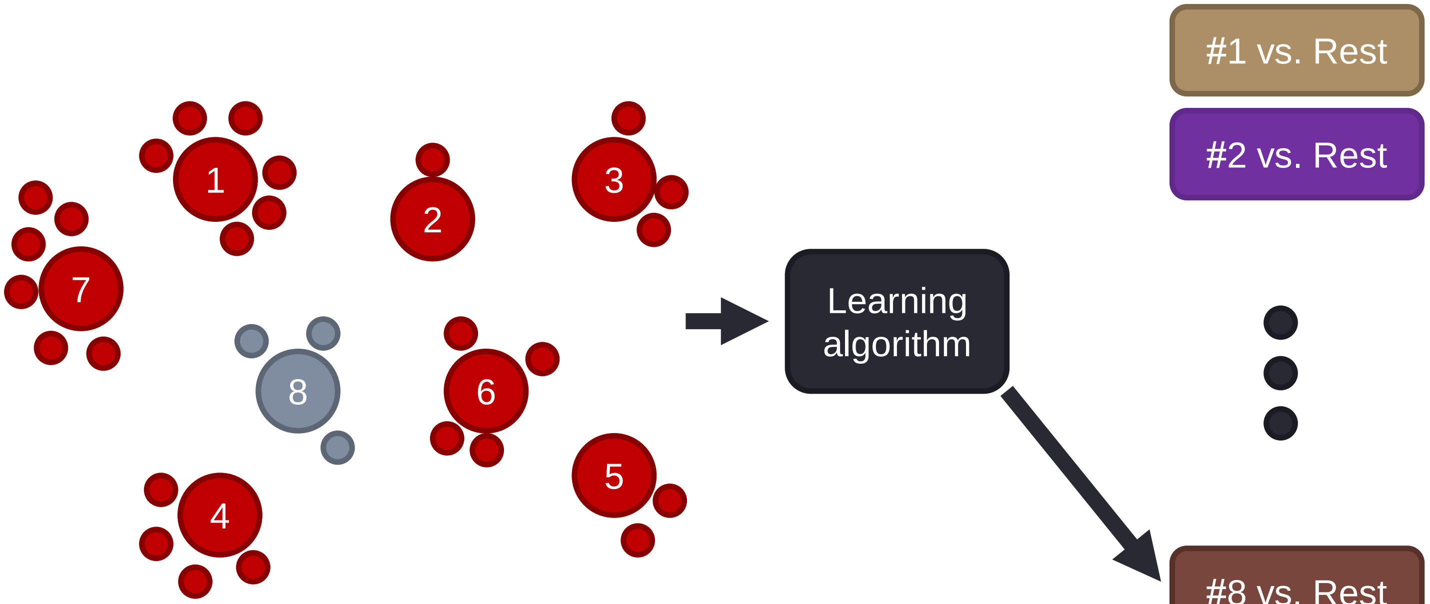
One vs. All: Training

- Goal: classify into many classes $h: \mathcal{X} \rightarrow \{1,2, \ldots, K\}$
- Solution: reduce to K separate binary tasks.



One vs. All: Training

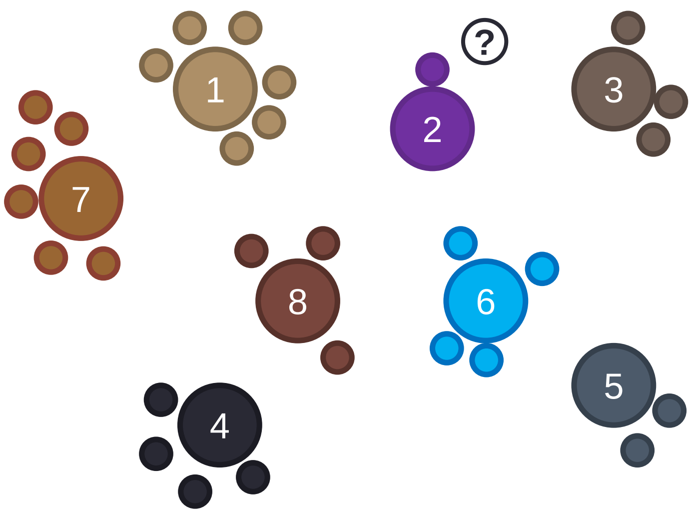
- Goal: classify into many classes $h : \mathcal{X} \to \{1, 2, \dots, K\}$
- Solution: reduce to K separate binary tasks.

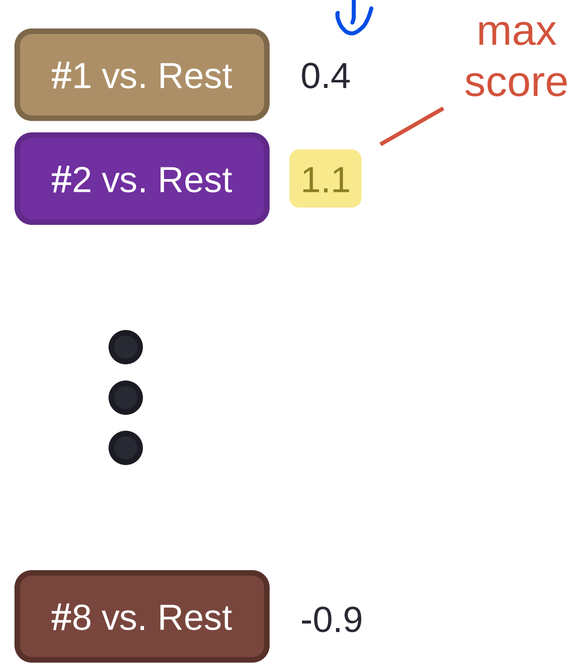


One vs. All: Prediction

- Goal: classify into many classes $h: \mathcal{X} \rightarrow \{1, 2, \ldots, K\}$
- Solution: reduce to K separate binary tasks.

$w_i X$ = margin





MNIST: Demo

- Famous computer vision dataset
- In Tutorial 08, we solved only a binary classification task: 0 or not 0
- Now we know how to solve the multiclass task, using many binary ones!

Load data

```
from keras.datasets import mnist
(train_X, train_y), (test_X, test_y) = mnist.load_data()
train_X = train_X.reshape(-1, 784)     # shape: (60000, 784)
test_X = test_X.reshape(-1, 784)       # shape: (10000, 784)
```

MNIST: Training

Train using many logistic regression binary classifiers

```
from sklearn.linear_model import LogisticRegression

classifiers  = []

# For each digit (class)
for k in range(10):
  # Make binary labels (current class vs. rest)
  train_binary_y = [1 if y == k else -1 for y in train_y]

  # Train a binary logistic regression classifier
  h = LogisticRegression(penalty="none")
  h.fit(train_X, train_binary_y)

  # Save classifier
  classifiers.append(h)
```

- Question: can training be done in parallel?

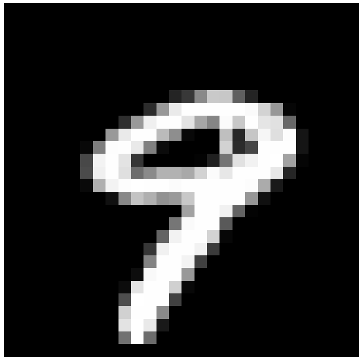
MNIST: Prediction

Function: predict one example

```
def predict(classifiers, x):
  scores = [h.predict_proba([x])[0][1] for h in hypotheses]
  y_pred = np.argmax(scores)

  return y_pred
```

- Example: predict the image below.
  - Which digits does it resemble?

True label: 9



Score

class	score
0	7.7e-09
1	2.2e-16
2	2.8e-10
3	2.1e-07
4	8.4e-03
5	3.5e-07
6	5.1e-08
7	0.16
8	5.5e-03
9	0.91

MNIST: Testing

Function: predict one example

```
def predict(classifiers, x):
  scores = [h.predict_proba([x])[0][1] for h in hypotheses]
  y_pred = np.argmax(scores)

  return y_pred
```

Evaluate test accuracy

```
from sklearn.metrics import accuracy_score

y_predicted = [predict(classifiers, x) for x in test_X]
test_accuracy = accuracy_score(test_y, y_predicted) * 100
print("Test accuracy: {:.2f}%".format(test_accuracy))
```

```
Test accuracy: 91.71%
```

Nice!

MNIST: Training with sklearn

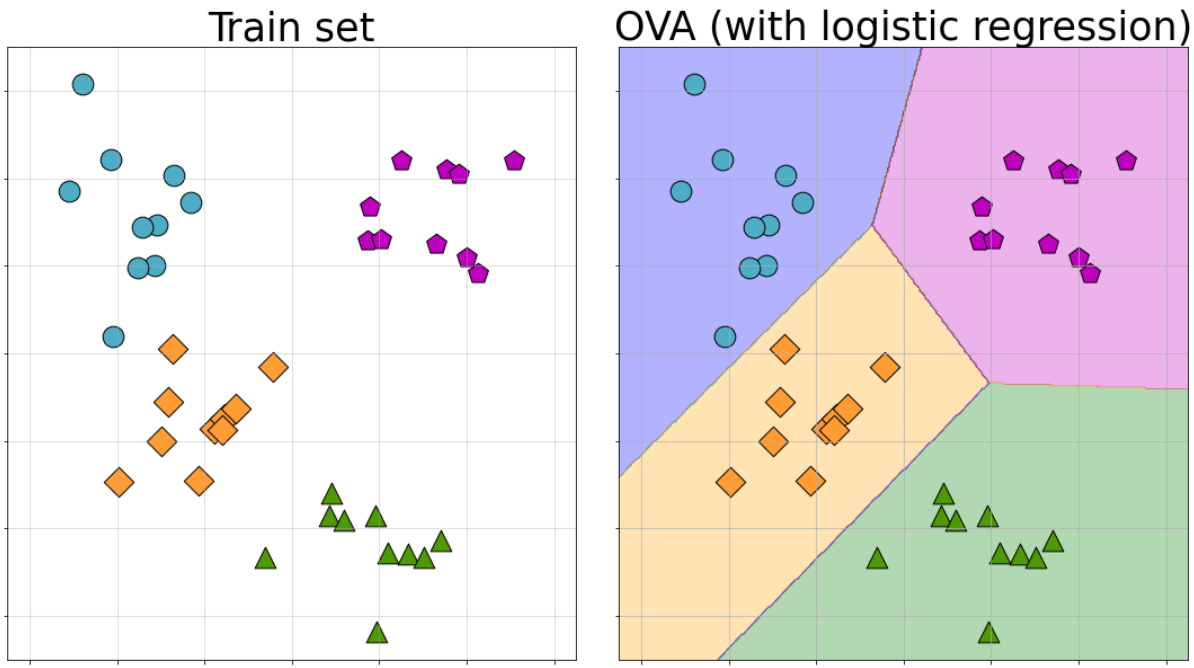
Train using many logistic regression binary classifiers

```
from sklearn.linear_model import LogisticRegression

classifiers  = []

# For each digit (class)
for k in range(10):
  # Make binary labels (current class vs. rest)
  train_binary_y = [1 if y == k else -1 for y in train_y]

  # Train a binary logistic regression classifier
  h = LogisticRegression(penalty="none")
  h.fit(train_X, train_binary_y)
  classifiers.append(h)
```

- sklearn's One vs. Rest implementation:

```
H = LogisticRegression(multi_class="ovr", penalty="none")
h.fit(train_X, train_y)
```

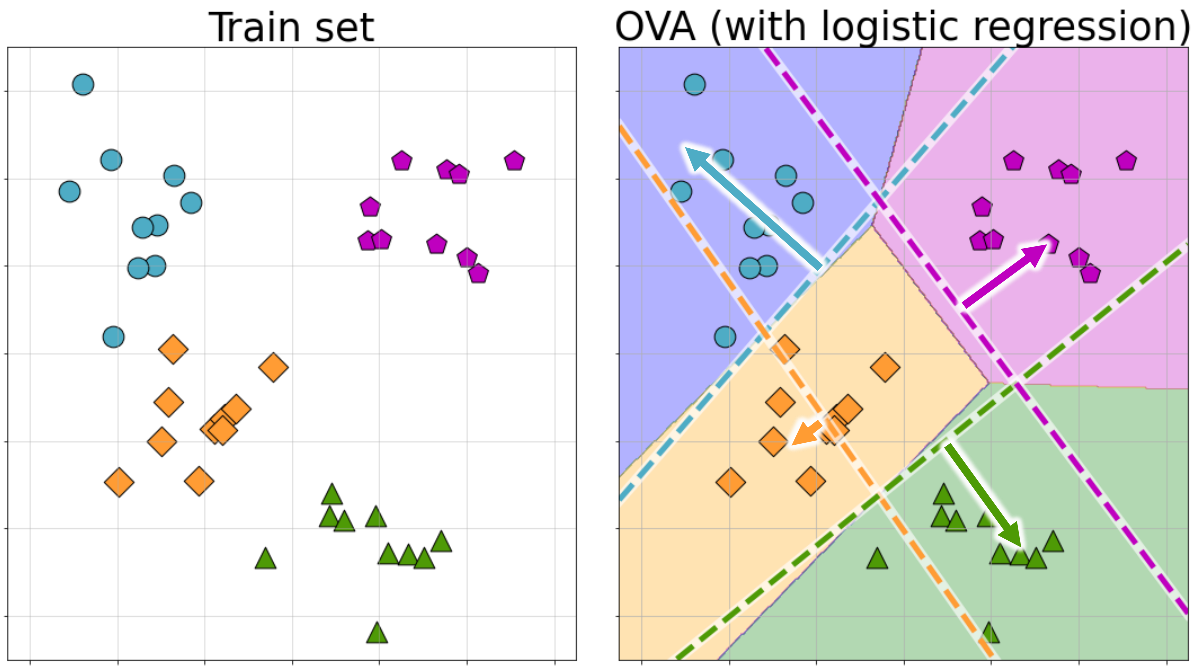
Decision boundaries: Toy dataset



Is each class
linearly separable
from the rest?

Think:
How do linear classifiers
perfectly fit the data?

Decision boundaries: Toy dataset



Is each class
linearly separable
from the rest?

Think:
How do linear classifiers
perfectly fit the data?

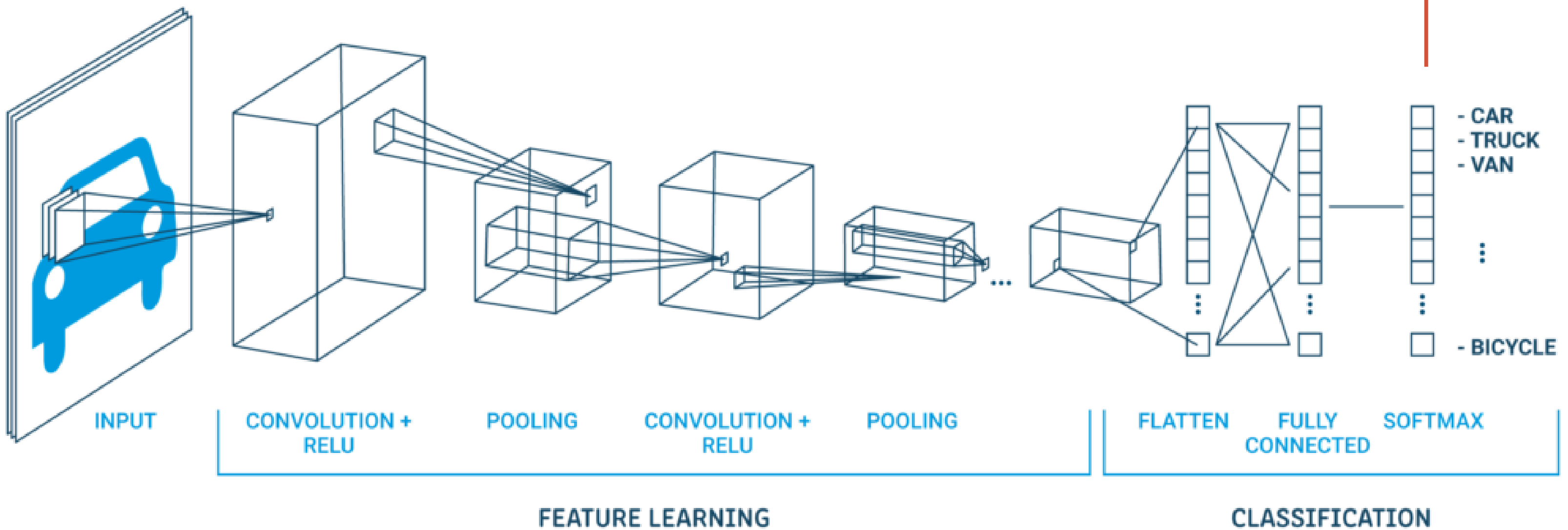
Hint 1:
See the “binary” separators

Hint 2:
The key is in the norms
of the normals
(their effect on inner products)

Multiclass models

- Goal: classify into many classes $h: \mathcal{X} \rightarrow \{1,2,\ldots,K\}$
- Solution: train <u>one</u> multiclass model that predicts a class distribution
- The common practice in Deep learning today:

Predicts
a class distribution



We will understand these layers.
But first, back to simpler models

Cross entropy (discrete distributions)

- Roughly: measures how one distribution differs from another.
- We use it to create a loss over class distributions:

$$\ell^{\mathrm{CE}}\big(\boldsymbol{p}, \widehat{\boldsymbol{p}}(\boldsymbol{x})\big) = H\big(\boldsymbol{p}, \widehat{\boldsymbol{p}}(\boldsymbol{x})\big) = -\sum_{k=1}^{K} p_k \ln \hat{p}_k$$

True distribution

Predicted distribution given x

- Since each example belongs to one class only, the true distribution is "one hot":

$$\ell^{\mathrm{CE}}(y, \widehat{\boldsymbol{p}}) = H(\mathrm{onehot}(\mathrm{y}), \widehat{\boldsymbol{p}}) = -\ln \hat{p}_y$$

Predicted probability for true class

- For example,

$$\ell^{\mathrm{CE}}\left(\boldsymbol{p}=\begin{bmatrix}0\\1\\0\end{bmatrix}, \widehat{\boldsymbol{p}}=\begin{bmatrix}0.1\\0.6\\0.3\end{bmatrix}\right) = -\ln 0.6 \approx 0.51$$

$$\ell^{\mathrm{CE}}\left(\boldsymbol{p}=\begin{bmatrix}0\\1\\0\end{bmatrix}, \widehat{\boldsymbol{p}}=\begin{bmatrix}0.4\\0.3\\0.3\end{bmatrix}\right) = -\ln 0.3 \approx 1.2$$

$\Longrightarrow$ ℓ^{CE} pushes $\widehat{\boldsymbol{p}}$ to $\boldsymbol{p}$

Recap: Binary logistic regression

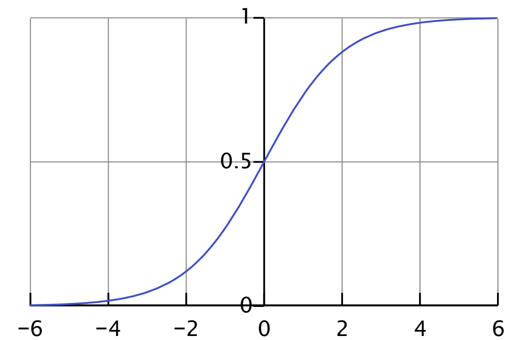
- Models the binomial distribution of y given $\boldsymbol{x}$

$$y = 1 \mid \boldsymbol{x}_i, \boldsymbol{w} \ \sim \mathrm{Binomial}\big(\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)\big)$$

$$\Longrightarrow \hat{p}_{+1} = \Pr[y = 1 | \boldsymbol{x}_i, \boldsymbol{w}] \quad = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)$$

$$\Longrightarrow \hat{p}_{-1} = \Pr[y = -1 | \boldsymbol{x}_i, \boldsymbol{w}] = 1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) = \sigma(-\boldsymbol{w}^\top \boldsymbol{x}_i)$$

Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



- Maximizes the likelihood of the dataset S:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}} \underbrace{\Pr[S; \boldsymbol{w}]}_{\text{Likelihood}} = \cdots = \underset{\boldsymbol{w}}{\operatorname{argmin}} \underbrace{\sum\nolimits_{(\boldsymbol{x}_i, y_i) \in S} - \ln \sigma(y_i \boldsymbol{w}^\top \boldsymbol{x}_i)}_{\text{Negative log-likelihood (NLL)}}$$

Extra: prove this loss is convex

Auxiliary:

$$-\ln \sigma(y_i \boldsymbol{w}^\top \boldsymbol{x}_i) = -\underbrace{\mathbf{1}\{y_i = +1\}}_{p_{+1}} \underbrace{\ln \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)}_{\ln \hat{p}_{+1}} - \underbrace{\mathbf{1}\{y_i = -1\}}_{p_{-1}} \underbrace{\ln \sigma(-\boldsymbol{w}^\top \boldsymbol{x}_i)}_{\ln \hat{p}_{-1}} = -\ln \hat{p}_{y_i}$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_i \ell^{\mathrm{CE}}\left(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i) = \begin{bmatrix} \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) \\ 1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) \end{bmatrix}\right)$$

Predicted binomial distribution given $\boldsymbol{x}_i$

Logistic regression: Decision rule

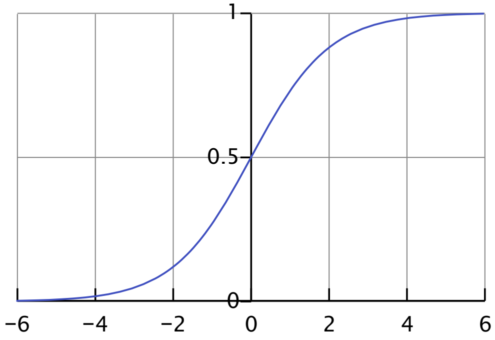
- Models the binomial distribution of y given $\boldsymbol{x}$

$$y = 1 \mid \boldsymbol{x}_i, \boldsymbol{w} \ \sim \mathrm{Binomial}\big(\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)\big)$$

$$\Rightarrow \hat{p}_{+1} = \Pr[y = 1 | \boldsymbol{x}_i, \boldsymbol{w}] \quad = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)$$

$$\Rightarrow \hat{p}_{-1} = \Pr[y = -1 | \boldsymbol{x}_i, \boldsymbol{w}] = 1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) = \sigma(-\boldsymbol{w}^\top \boldsymbol{x}_i)$$

Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



- Predicts the class with the highest probability:

$$h(\boldsymbol{x}_i) = \underset{y \in \{-1,+1\}}{\operatorname{argmax}} \ \sigma(y_i \boldsymbol{w}^\top \boldsymbol{x}_i)$$

- Exercise: how is this a linear classifier if σ is non-linear?

Multinomial logistic regression

- Models the distribution of all classes given $\boldsymbol{x}_i$

$$y_i | \boldsymbol{x}_i, \boldsymbol{w} \sim \text{Multinomial}(\hat{p}_1, \dots, \hat{p}_K)$$

Need a generalization of the sigmoid!

- Trains a linear classifier $\boldsymbol{w}_k \in \mathbb{R}^d$ for each class k
- The multinomial distribution given $\boldsymbol{x}_i$:
  - Score each class using $\boldsymbol{w}_k^\top \boldsymbol{x}_i$
  - Turn scores into a normalized distribution, i.e., softmax: $\hat{p}_k(\boldsymbol{x}_i) = \dfrac{e^{\boldsymbol{w}_k^\top \boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{w}_j^\top \boldsymbol{x}_i}}$
- Missing piece of the puzzle: which loss to use?
  - The cross-entropy loss: $\boldsymbol{\Theta}^* = \underset{\boldsymbol{w}_1,\dots,\boldsymbol{w}_K}{\operatorname{argmin}} \sum_i -\ln \hat{p}_{y_i}(\boldsymbol{x}_i) = \underset{\boldsymbol{w}_1,\dots,\boldsymbol{w}_K}{\operatorname{argmin}} \sum_i -\ln \dfrac{e^{\boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{w}_j^\top \boldsymbol{x}_i}}$

Computing the gradient

- We saw that the multinomial logistic regression formulation is:

$$\mathbf{\Theta}^* = \underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K}{\operatorname{argmin}} \sum_i \ell^{\mathrm{CE}}\big(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\big) = \underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K}{\operatorname{argmin}} \sum_i -\ln \frac{e^{\boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{w}_j^\top \boldsymbol{x}_i}}$$

- Advanced: prove the loss is convex in $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K$

- Exercise: compute the gradient $\nabla_{\boldsymbol{w}_k} \ell^{\mathrm{CE}}\big(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\big)$ w.r.t each vector $\boldsymbol{w}_k$

Computing the gradient

- We saw that the multinomial logistic regression formulation is:

$$\mathbf{\Theta}^* = \underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K}{\operatorname{argmin}} \sum_i \ell^{\mathrm{CE}}\big(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\big) = \underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K}{\operatorname{argmin}} \sum_i -\ln \frac{e^{\boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{w}_j^\top \boldsymbol{x}_i}}$$

- Advanced: prove the loss is convex in $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K$
- Exercise: compute the gradient $\nabla_{\boldsymbol{w}_k} \ell^{\mathrm{CE}}\big(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\big)$ w.r.t each vector $\boldsymbol{w}_k$
- Solution: $\nabla_{\boldsymbol{w}_k} \ell^{\mathrm{CE}}\big(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\big) = \big(-\mathbb{I}[k = y_i] + \underbrace{\hat{p}_k(\boldsymbol{x}_i)}_{= \frac{e^{\boldsymbol{w}_k^\top \boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{w}_j^\top \boldsymbol{x}_i}}}\big)\, \boldsymbol{x}_i$

The normalization "binds" all linear models together

MNIST: Training with sklearn

- One vs. Rest:

```
H = LogisticRegression(multi_class="ovr", penalty="none")
h.fit(train_X, train_y)
```

```
Train: 93.15%, Test: 91.71%
```

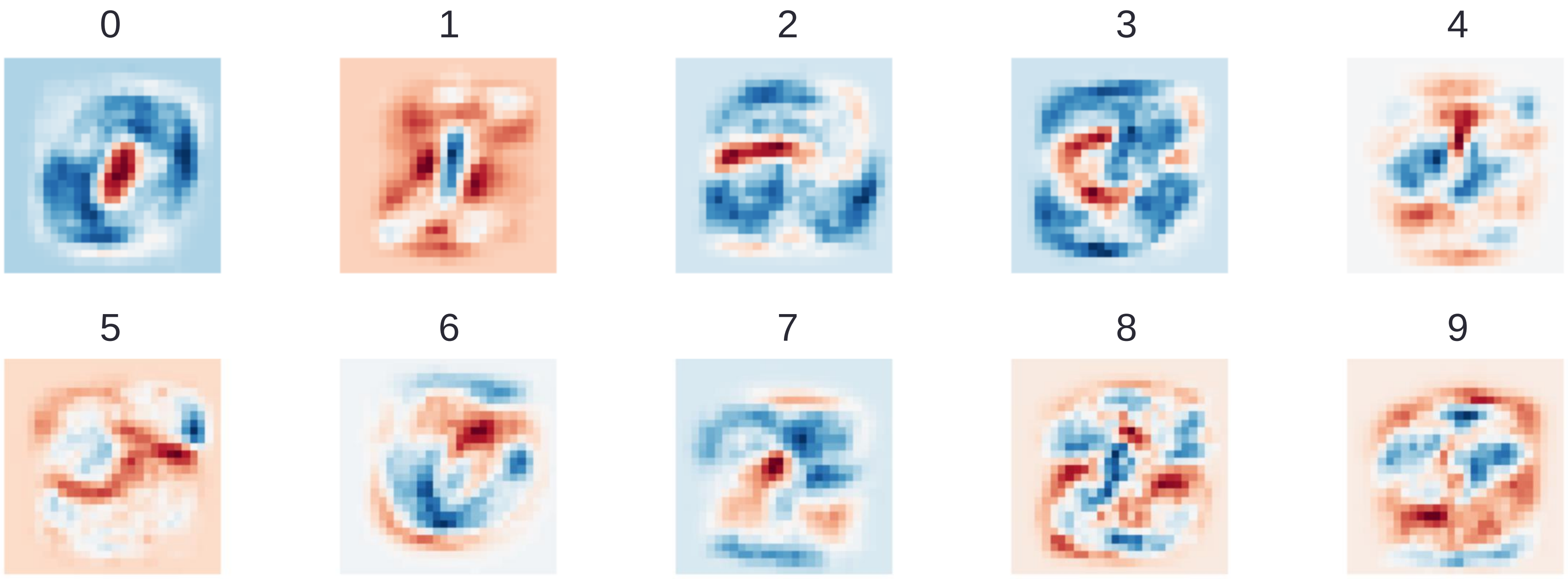
- Multinomial regression:

```
H = LogisticRegression(multi_class="multinomial", penalty="none")
h.fit(train_X, train_y)
```

```
Train: 94.35%, Test: 92.27%
```

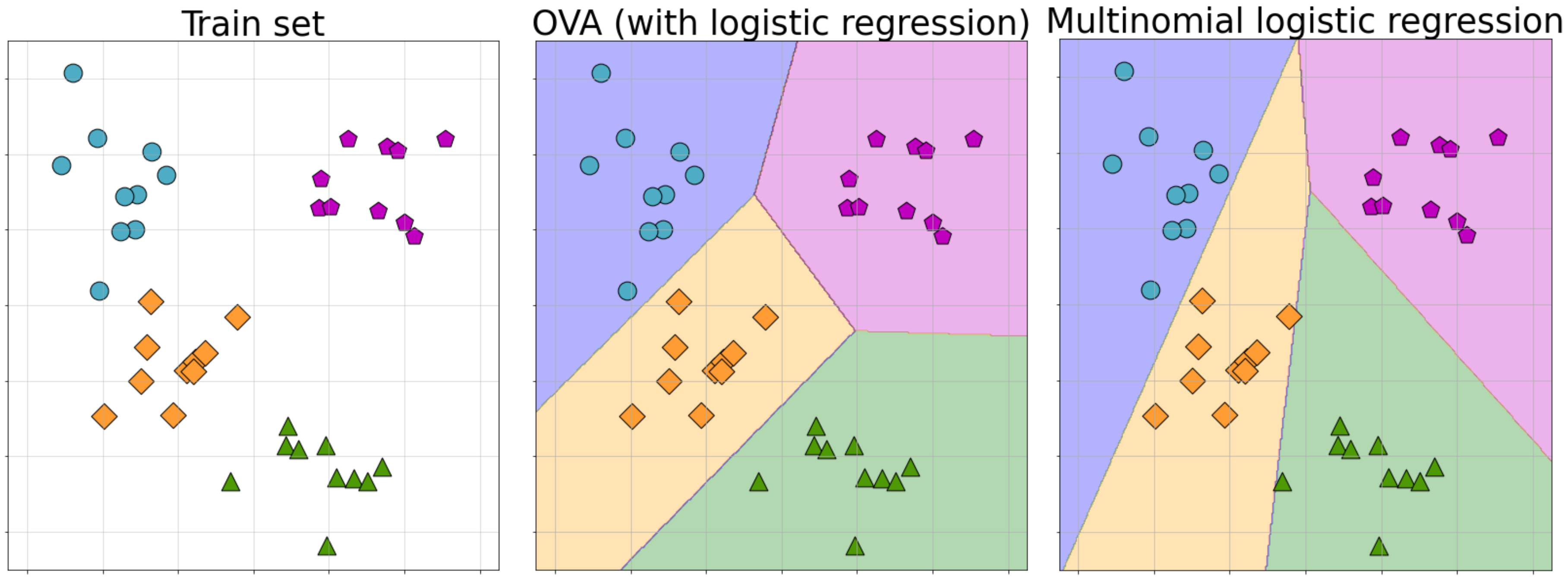
MNIST: Visualization

- In both approaches we saw, each class has its own vector $\boldsymbol{w}_k$

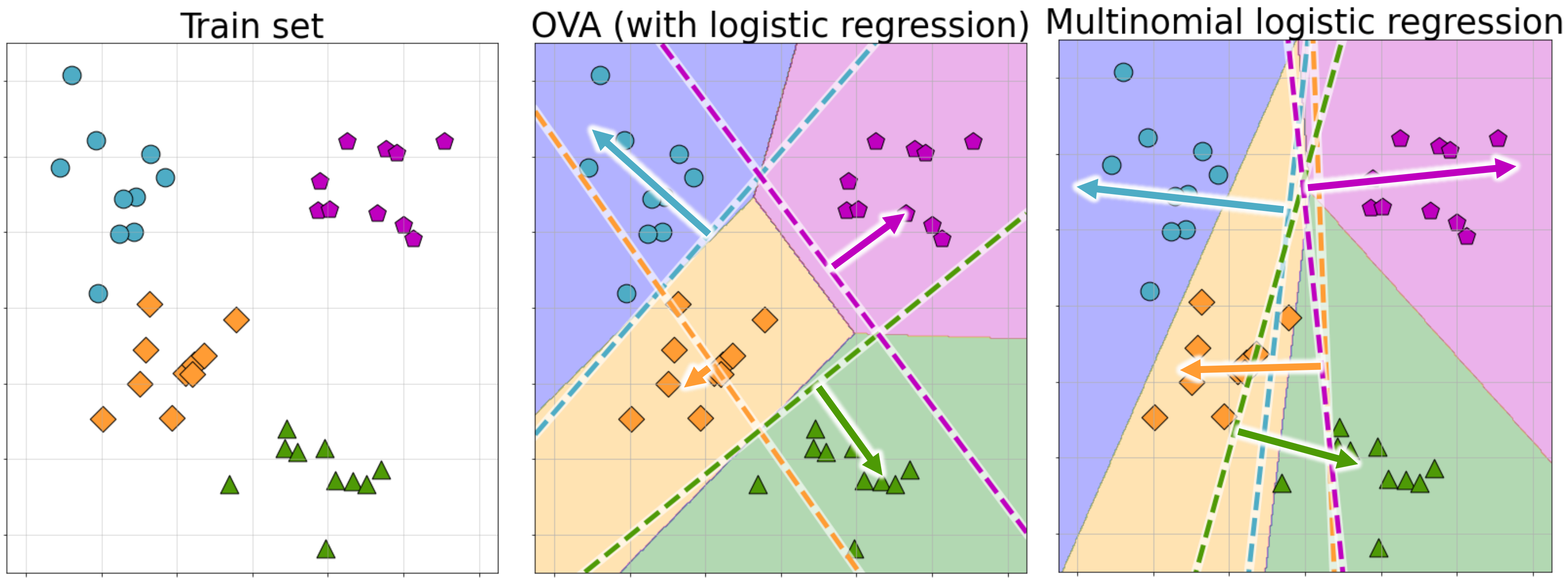


Source: StackExchange

Decision boundaries: Toy dataset



Decision boundaries: Toy dataset



Think about
the inner products here

From linear models to deep models

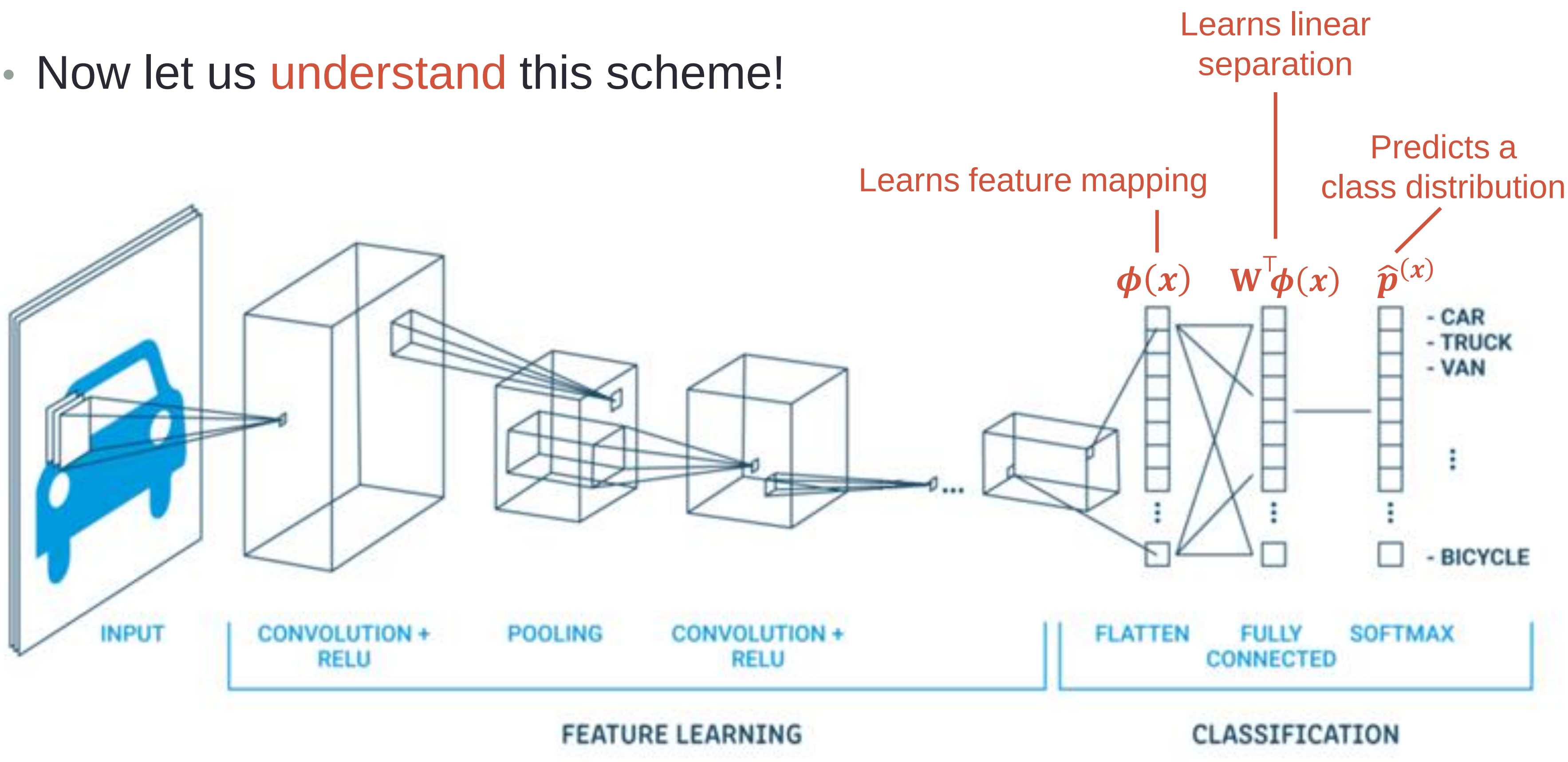
- Models the distribution of all classes given $\boldsymbol{x}_i$

$$y_i | \boldsymbol{x}_i, \boldsymbol{w} \sim \mathrm{Multinomial}(\hat{p}_1, \dots, \hat{p}_K)$$

- Learns a feature mapping $\boldsymbol{\phi}: \mathcal{X} \to \mathbb{R}^p$
- Trains a linear classifier $\boldsymbol{w}_k \in \mathbb{R}^p$ for each class k
- The multinomial distribution given $\boldsymbol{x}_i$:
  - Score each class using $\boldsymbol{w}_k^\top \boldsymbol{\phi}(\boldsymbol{x}_i)$
  - Turn scores into a normalized distribution, i.e., softmax: $\hat{p}_k(\boldsymbol{x}_i) = \dfrac{e^{\boldsymbol{w}_k^\top \boldsymbol{\phi}(\boldsymbol{x}_i)}}{\sum_j e^{\boldsymbol{w}_j^\top \boldsymbol{\phi}(\boldsymbol{x}_i)}}$
- Missing piece of the puzzle: which loss to use?
  - The cross-entropy loss: $\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\mathrm{argmin}} \sum_i \ell^{\mathrm{CE}}\big(y_i, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\big)$

Multiclass in Deep learning

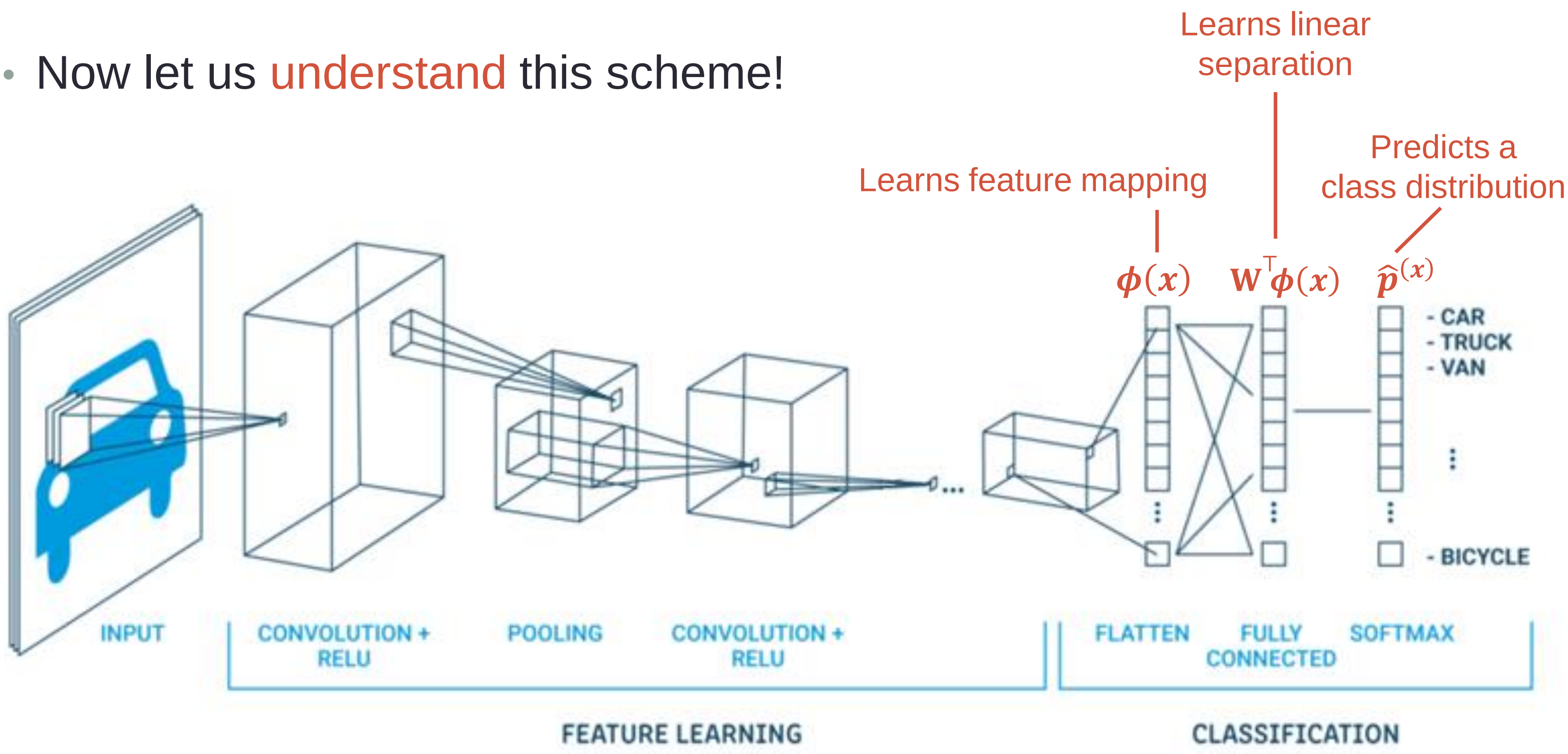
- Now let us understand this scheme!



- Again, we use the cross-entropy loss $\ell^{\mathrm{CE}}\left(y, \hat{\boldsymbol{p}}(\boldsymbol{x}_i)\right)$

Multiclass in Deep learning

- Now let us understand this scheme!



- Important: the feature mapping and linear separation are learned jointly.
- The network learns features that are easy to separate linearly!

Tutorial summary

- One vs. All reduces a multiclass task into separate binary tasks.
- Multinomial regression trains linear separators jointly
  - Create a class distribution using softmax
  - Train using the cross-entropy loss

Course summary

- Supervised binary classification
  - Decision trees, k-NN, SVM
- Aspects of learning
  - Statistical, Model selection, Optimization, Practical aspects
- More supervised learning
  - Regression, Bagging and boosting, Deep learning, Multiclass classification
- Beyond supervised learning
  - Dimensionality reduction, Self-supervised, Semi-supervised

נשמח אם תמלאו משובי הוראה!

Exam

- Moed A: Sunday, 16/07, 09:00
  - Questions: in the Piazza (not by email)
  - We will update you soon regrading office hours, exam structure, etc.
- Moed B: Thursday, 19/10

Good luck!