Xi'an Jiaotong-Liverpool University

西交利物浦大学

INT305 Machine Learning

Lecture 12

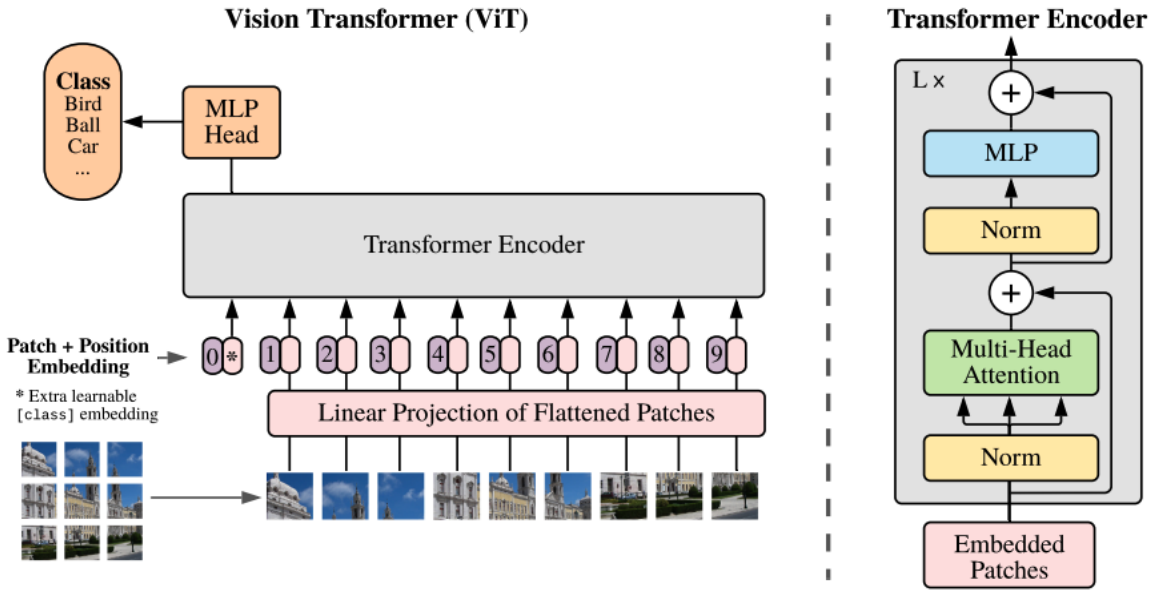
Basic information of MLLMs and Revision

Siyue Yu

Department Intelligence Science

siyue.yu02@xjtlu.edu.cn

Basic knowledge of MLLMs- attention mechanism and FFN



From paper: AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Revision

Supervised Learning, Linear Methods for Regression, Optimization

- First learning algorithm: KNN
- Second learning algorithm of the course: linear regression.
  - Task: predict scalar-valued targets (e.g. stock prices)
  - Architecture: linear function of the inputs
- While KNN was a complete algorithm, linear regression exemplifies a modular approach that will be used through this course:
  - choose a model describing the relationships between variables of interest
  - define a loss function quantifying how bad the fit to the data is
  - choose a regularizer saying how much we prefer different candidate models (or explanations of data)
  - fit a model that minimizes the loss function and satisfies the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- Mixing and matching these modular components give us a lot of new ML methods.

Linear Classifiers, Logistic Regression, Multiclass Classification

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(>2)-valued target

- Examples of binary classification
  - Predict whether a patient has a disease, given the presence or absence of various symptoms
  - Classify e-mails as spam or non-spam
  - Predict whether a financial transaction is fraudulent

Linear Classifiers, Logistic Regression, Multiclass Classification

Binary linear classification

- **Classification**: given a D-dimensional input $\mathbf{x} \in \mathbb{R}^D$ predict a discrete-valued target
- **Binary**: predict a binary target $t \in \{0,1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples. Sorry.
  - $t \in \{0,1\}$ or $t \in \{-1,+1\}$ is for computational convenience.
- **Linear**: model prediction y is a linear function of $\mathbf{x}$, followed by a threshold r:

$$z = \mathbf{w}^\intercal \mathbf{x} + b$$

$$y = \begin{cases} 1 & if\ z \geq r \\ 0 & if\ z < r \end{cases}$$

Summary | Binary Linear Classifiers

- **Summary:** targets $t \in \{0,1\}$, inputs $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$, and model is defined by weights $\mathbf{w}$ and

$$z = \mathbf{w}^\intercal \mathbf{x}$$

$$y = \begin{cases} 1 & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases}$$

- How can we find good values for $\mathbf{w}$?
- If training set is linearly separable, we could solve for $\mathbf{w}$ using linear programming
  - We could also apply an iterative procedure known as the *perceptron algorithm* (but this is primarily of historical interest).
- If it‘s not linearly separable, the problem is harder
  - Data is almost never linearly separable in real life.

Logistic Regression

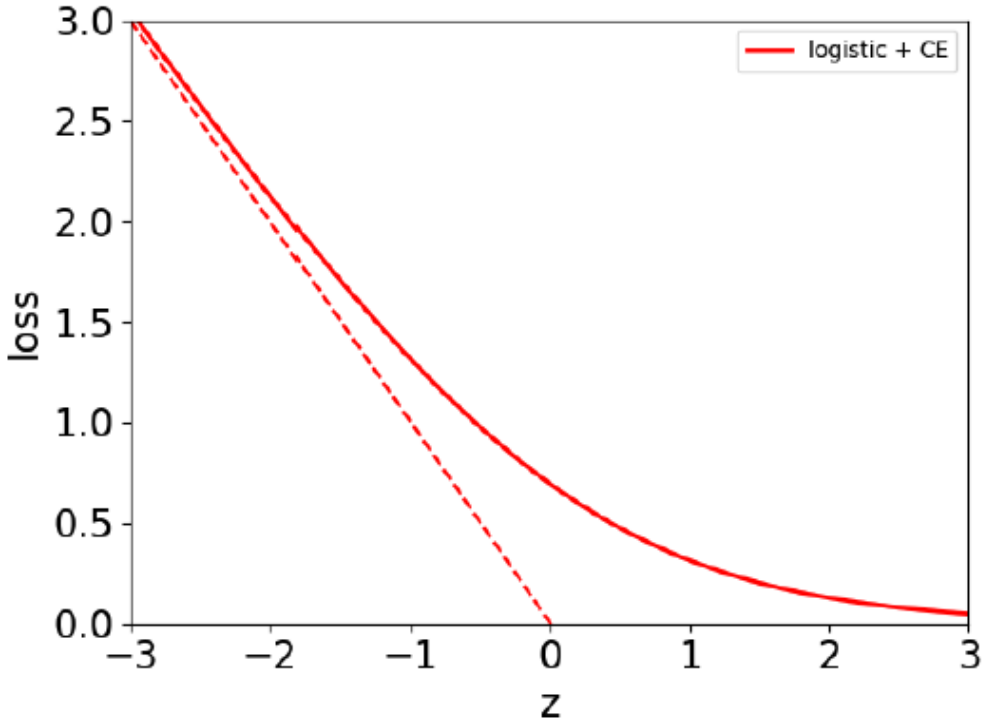
Logistic regression:

$$z = \mathbf{w}^\top \mathbf{x}$$

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

$$\mathcal{L}_{\mathrm{CE}}(y, t) = -t \log y - (1 - t) \log(1 - y)$$

Plot is for target $t = 1$.



Gradient Descent for Logistic Regression

- How do we minimize the cost $\mathcal{J}$ for logistic regression? No direct solution.
  - Taking derivatives of $\mathcal{J}$ w.r.t. $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- However, the logistic loss is a convex function in $\mathbf{w}$, so let's consider the gradient descent method from the last lecture.
  - Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.
  - A standard initialization is $\mathbf{w} = 0$. (why?)

Gradient of Logistic Loss

Back to logistic regression:

$$\mathcal{L}_{\mathrm{CE}}(y,t) = -t\log y - (1-t)\log(1-y)$$

$$y = 1/(1+e^{-z}) \text{ and } z = \mathbf{w}^{\mathsf{T}}\mathbf{x}$$

Therefore

$$\frac{\partial \mathcal{L}_{\mathbf{CE}}}{\partial w_j} = \frac{\partial \mathcal{L}_{\boldsymbol{CE}}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1-t}{1-y}\right) \cdot y(1-y) \cdot x_j$$

$$= (y-t)x_j$$

(verify this)

Gradient descent (coordinatewise) update to find the weights of logistic regression:

$$w_j \leftarrow w_j - \alpha \frac{\partial \boldsymbol{\mathcal{J}}}{\partial w_j}$$

$$= w_j - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})x_j^{(i)}$$

Gradient Descent for Logistic Regression

Comparison of gradient descent updates:

- Linear regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- Logistic regression:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- Not a coincidence! These are both examples of generalized linear models. But we won't go in further detail.
- Notice $\frac{1}{N}$ in front of sums due to averaged losses. This is why you need smaller learning rate when cost is summed losses ($\alpha' = \alpha/N$).

Multiclass Classification

- Targets form a discrete set $\{1, \cdots, K\}$.

- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \cdots, 0, 1, 0 \cdots, 0)}_{\text{entry } k \text{ is } 1} \in \mathbb{R}^K$$

Multiclass Linear Classification

- We can start with a linear function of the inputs.
- Now there are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a K-dimensional vector $\mathbf{b}$ of biases.
- A linear function of the inputs:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \quad \text{for} \quad k = 1,2,\cdots,K$$

- We can eliminate the bias $\mathbf{b}$ by taking $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$. So, vectorized:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad \text{or with dummy } x_0 = 1 \quad \mathbf{z} = \mathbf{W}\mathbf{x}$$

Multiclass Linear Classification

- How can we turn this linear prediction into a one-hot prediction?
- We can interpret the magnitude of z_k as a measure of how much the model prefers k as its prediction.
- If we do this, we should set

$$y_i = \begin{cases} 1 & i = \arg\max_k z_k \\ 0 & \text{otherwise} \end{cases}$$

Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e., $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.
- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \cdots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

  - Outputs can be interpreted as probabilities (positive and sum to 1)
  - If z_k is much larger than the others, then $\text{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.

Softmax Regression

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{y}, \mathbf{t}) = -\sum_{k=1}^{K} t_k \log y_k$$
$$= -\mathbf{t}^{\mathsf{T}}(\log \mathbf{y}),$$

  where the log is applied elementwise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Softmax Regression

- Softmax regression (with dummy $x_0 = 1$):

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$
$$\boldsymbol{\mathcal{L}}_{\text{CE}} = -\mathbf{t}^{\intercal}(\log \mathbf{y})$$

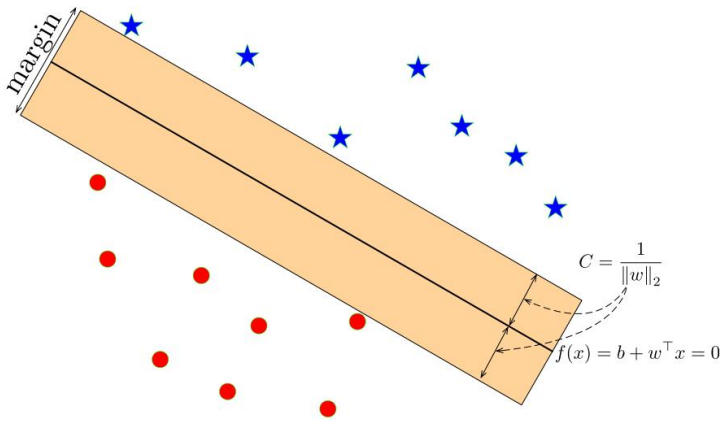
- Gradient descent updates can be derived for each row of $\mathbf{W}$:

$$\frac{\partial \boldsymbol{\mathcal{L}}_{\text{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \boldsymbol{\mathcal{L}}_{\text{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(y_k^{(i)} - t_k^{(i)} \right) \mathbf{x}^{(i)}$$

- Similar to linear/logistic reg (no coincidence) (verify the update)

SVM



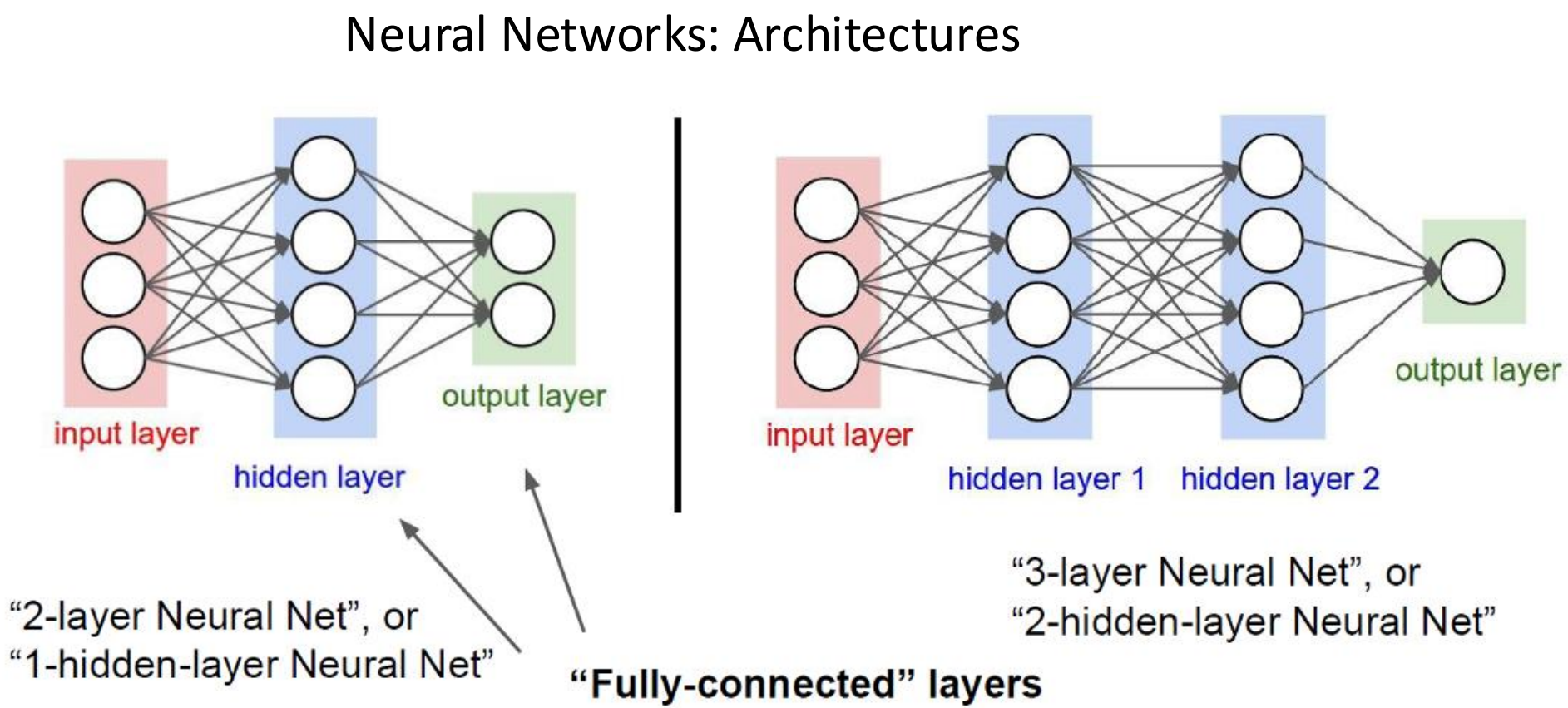
A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is particularly effective in high-dimensional spaces and is well-suited for scenarios where there is a clear margin of separation between classes.

The basic idea behind SVM is to find the hyperplane that best divides a dataset into two classes.

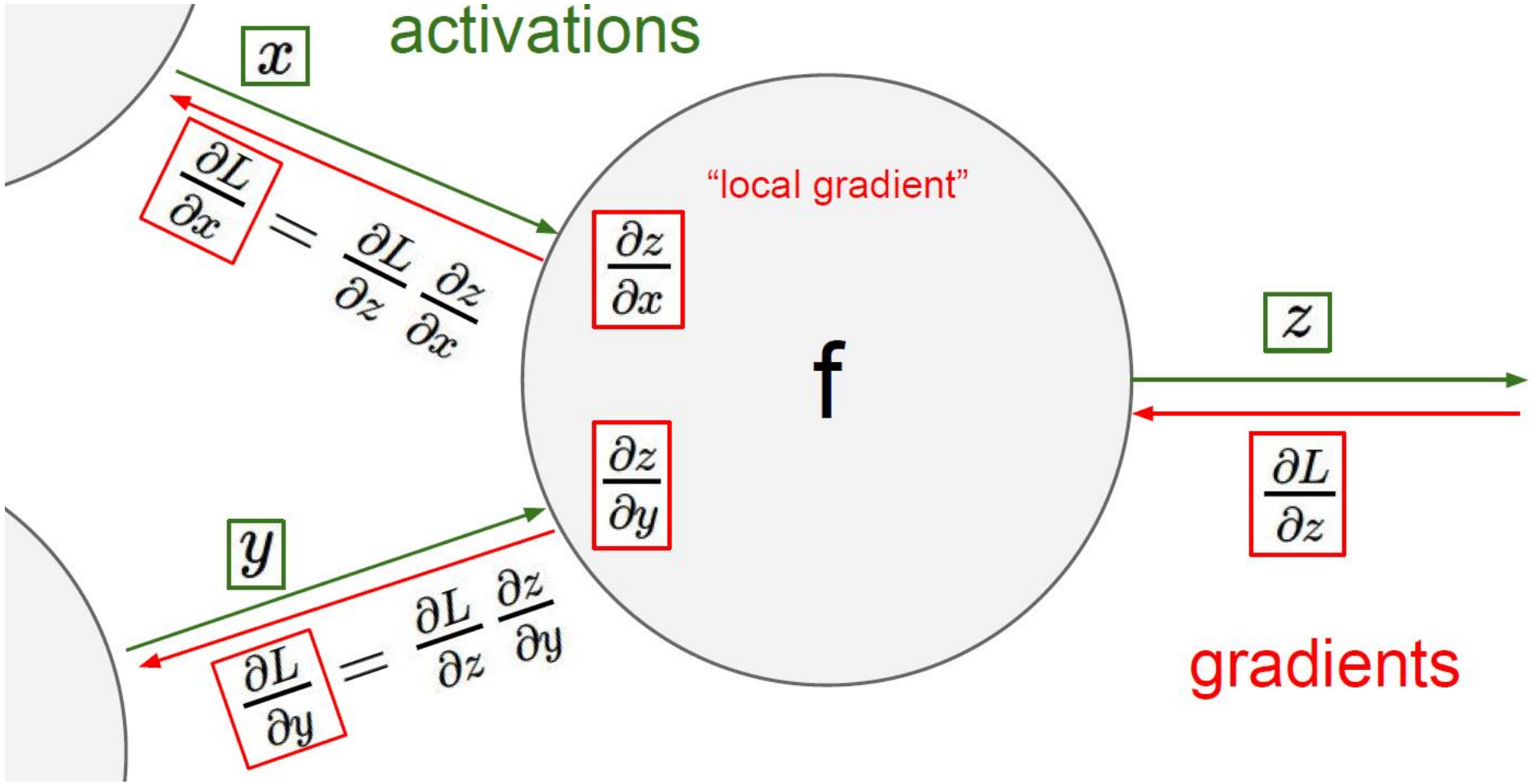
The term "support vector" refers to the data points that lie closest to the decision boundary, and these points are crucial in determining the optimal hyperplane.

The goal is to maximize the margin, which is the distance between the hyperplane and the nearest data points of each class.

Neural network



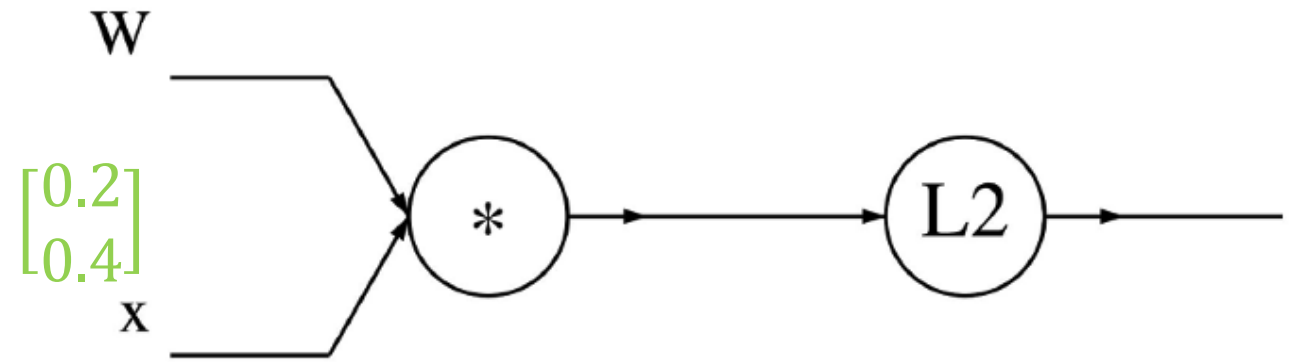
Chain rule


activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

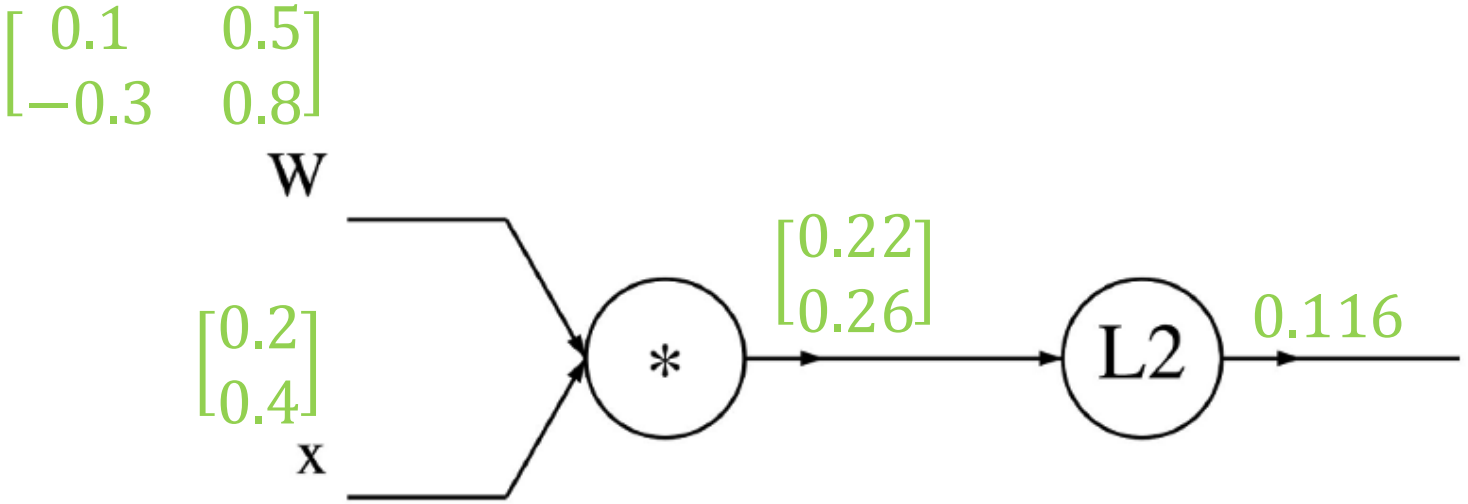


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

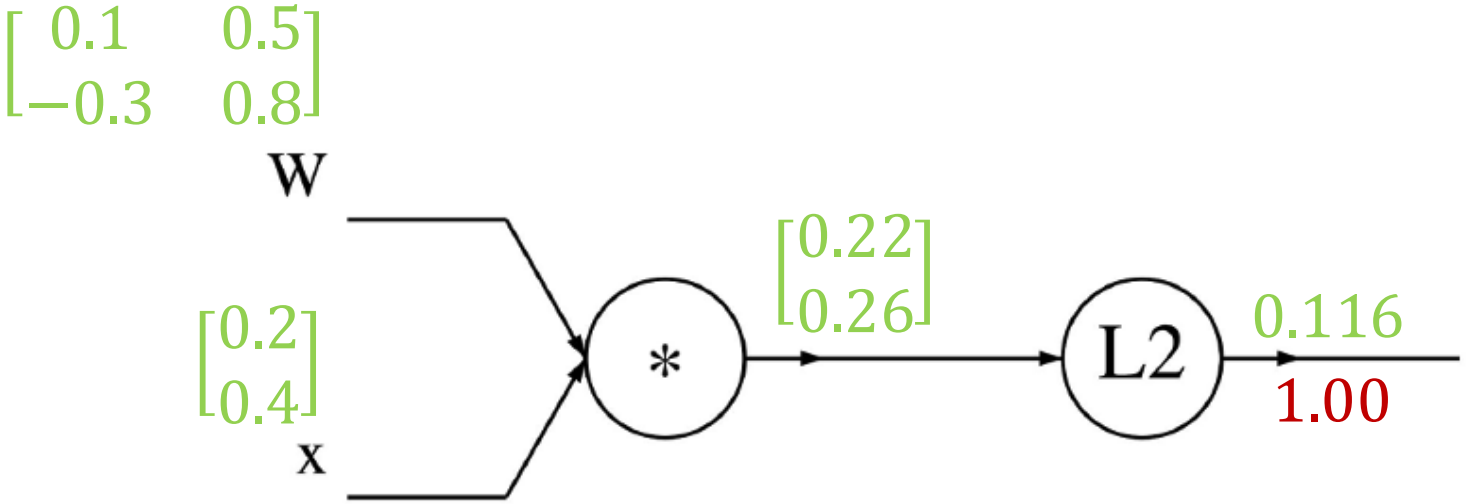


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

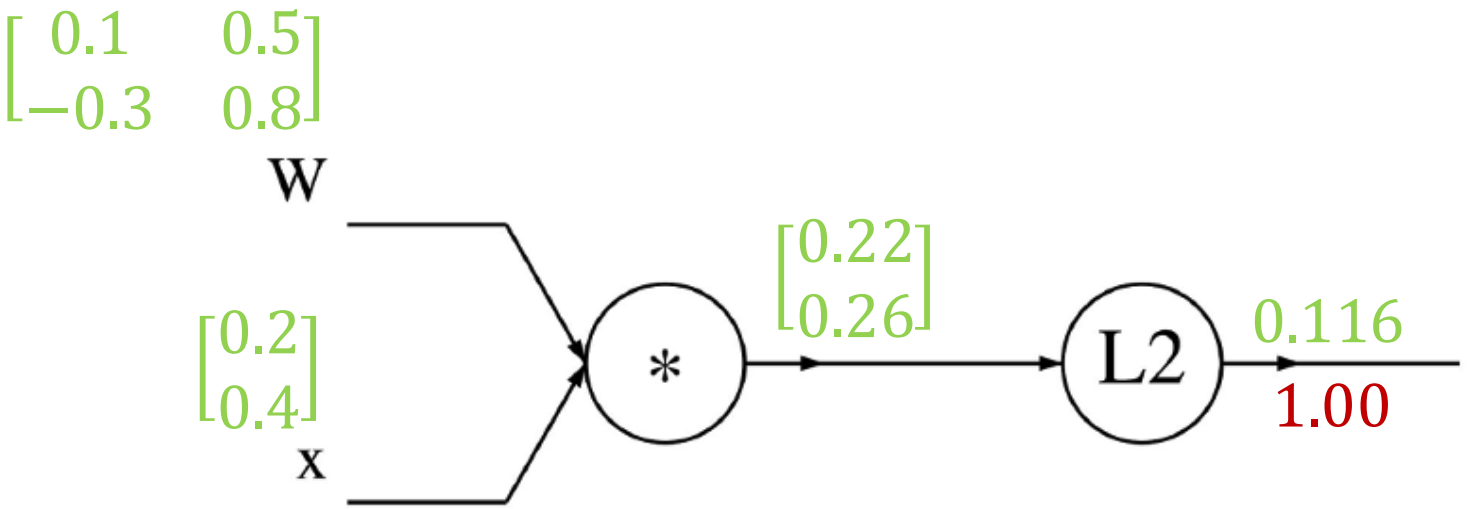


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

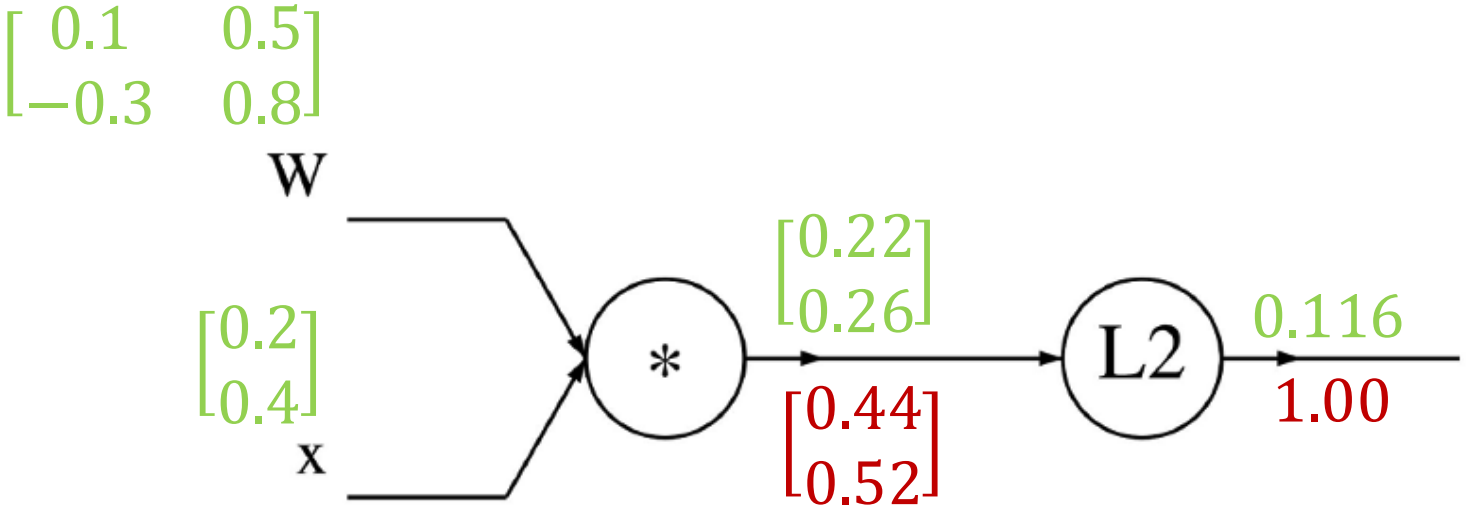
$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2\mathrm{q}}$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

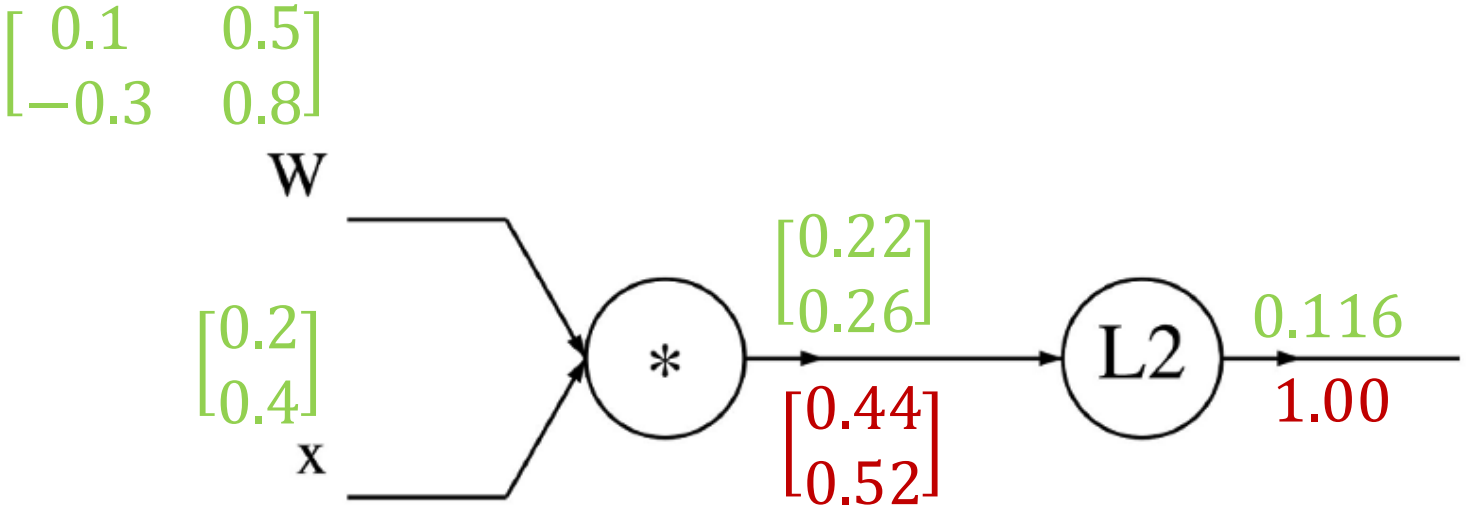
$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2\mathrm{q}}$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



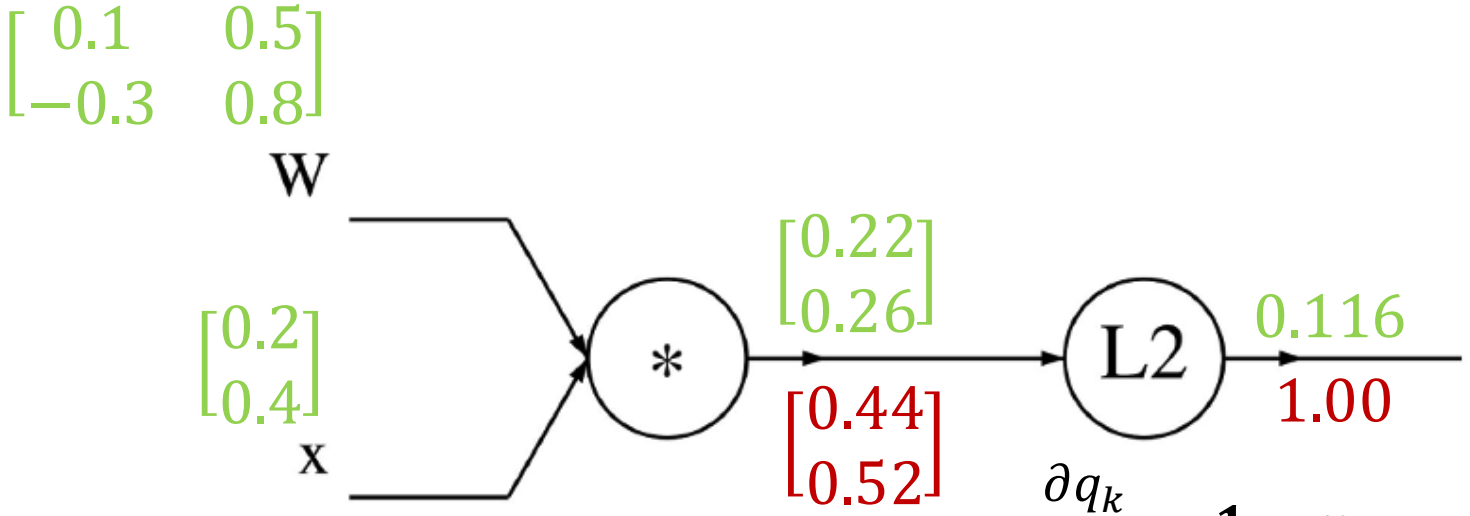
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

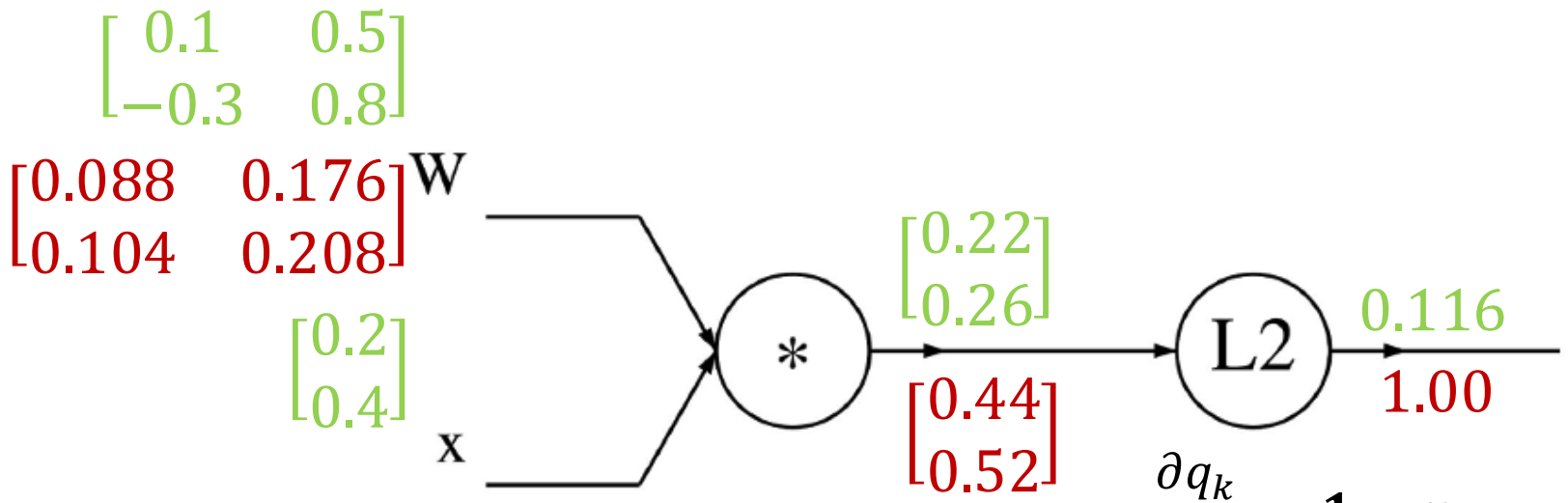
$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) = 2q_i x_j$$

Gradients for vector

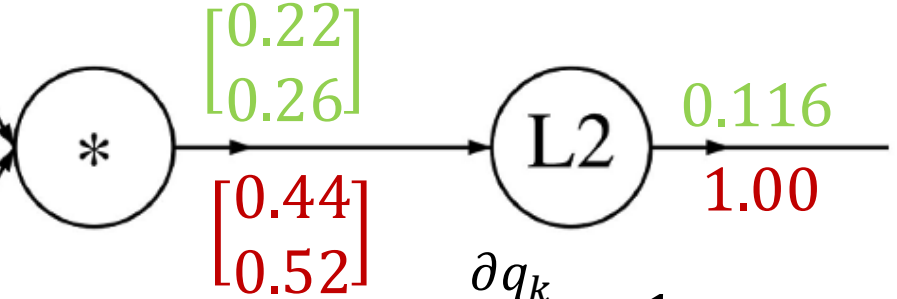
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix} \text{W}$$

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

x



$$\begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

0.116

1.00

$$\nabla_W f = 2q \cdot x^\intercal$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Gradients for vector

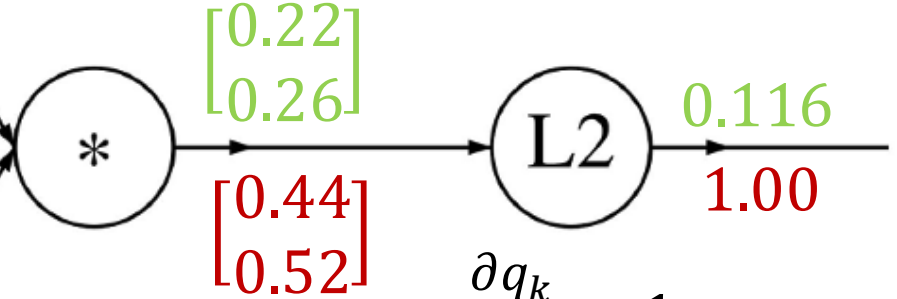
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix} \text{W}$$

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

x



$$\begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

0.116

1.00

$$\nabla_W f = 2q \cdot x^\top$$

Always check: The gradient with respect to a variable should have the same shape as the variable

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

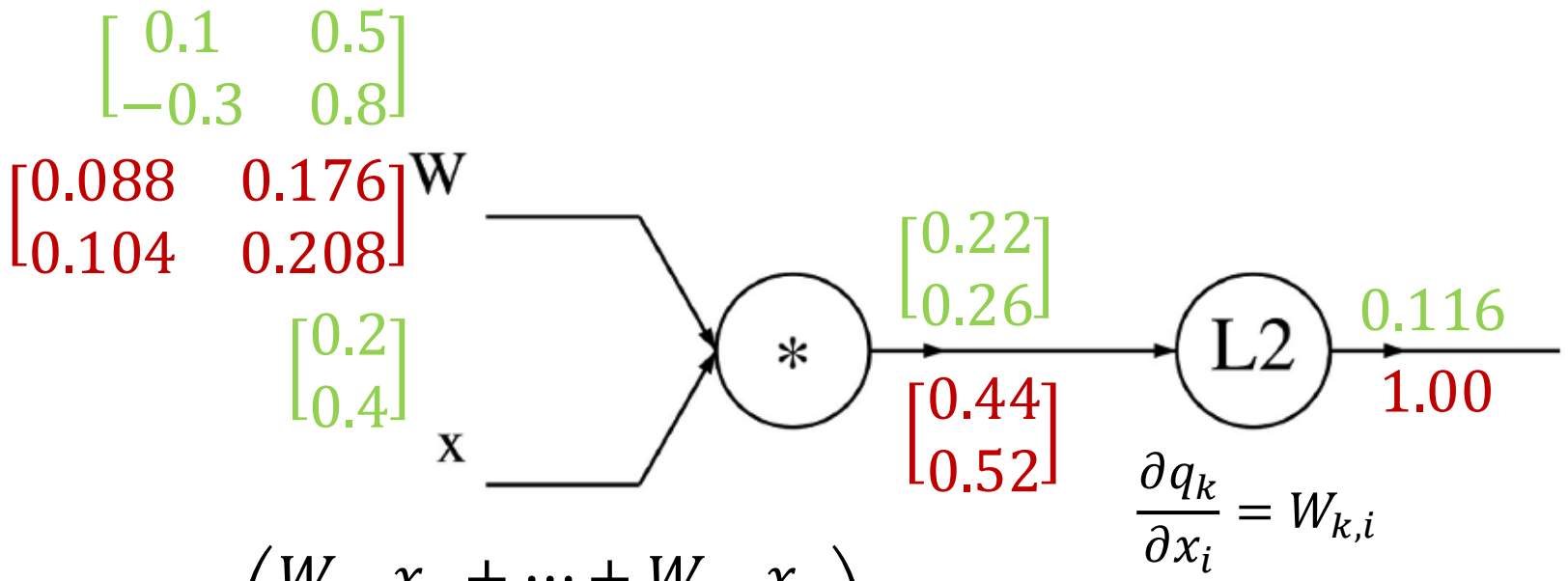
$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

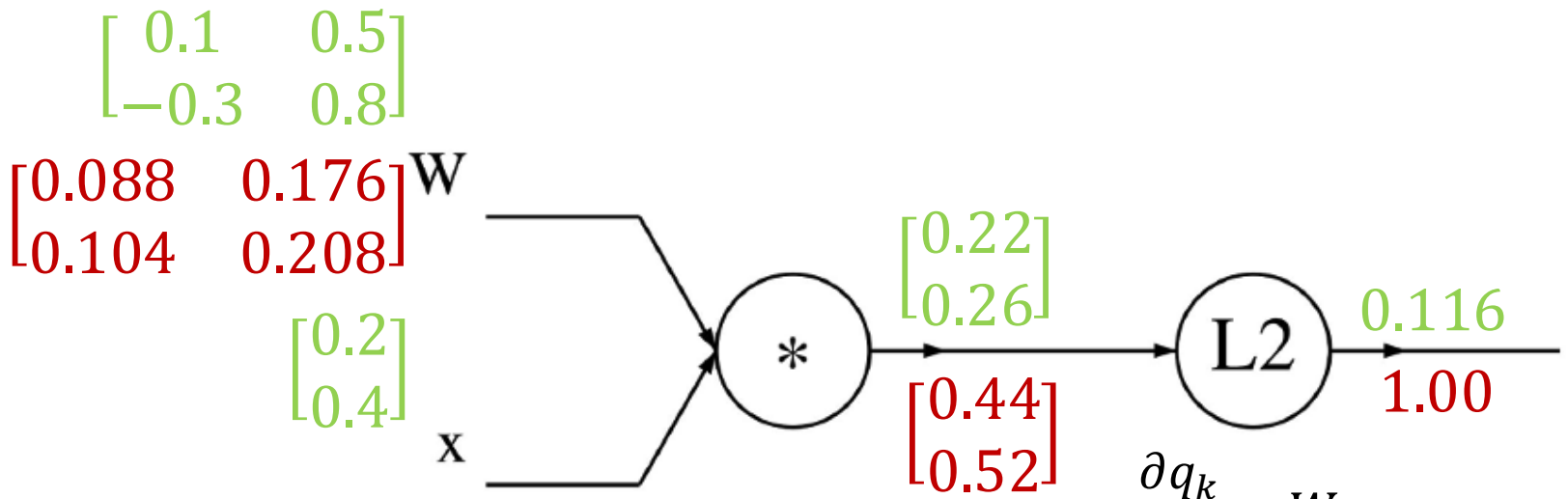


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

Gradients for vector

A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k}\frac{\partial q_k}{\partial x_i} = \sum_k 2q_k W_{k,i}$$

Gradients for vector

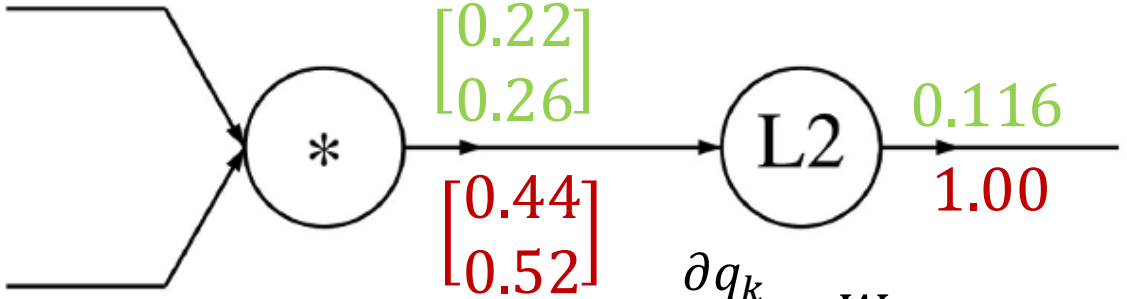
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

$$\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix} \text{W}$$

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$\begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$



$$\begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix}$$

0.116

1.00

$$\nabla_x f = 2W^\intercal \cdot q$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

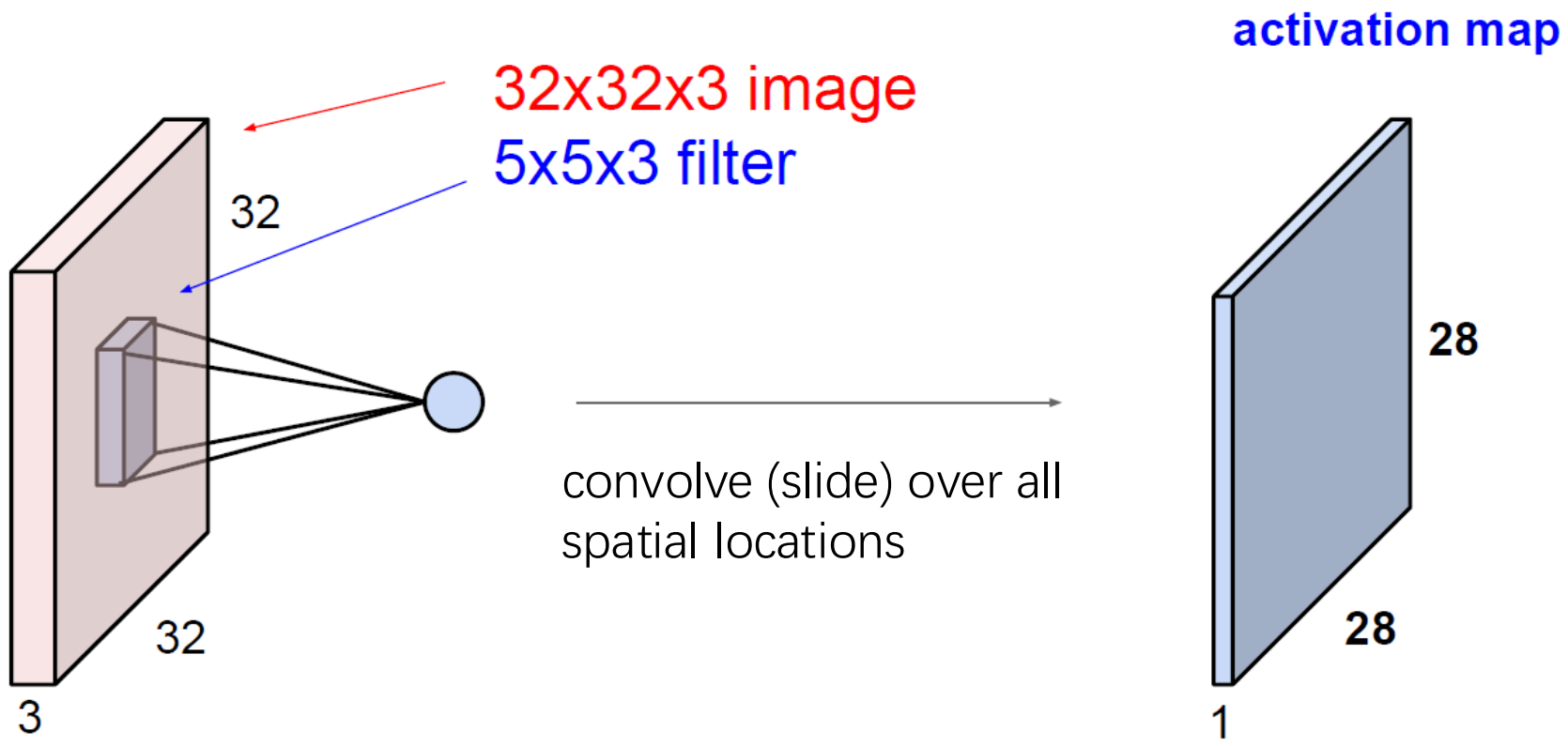
$$f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

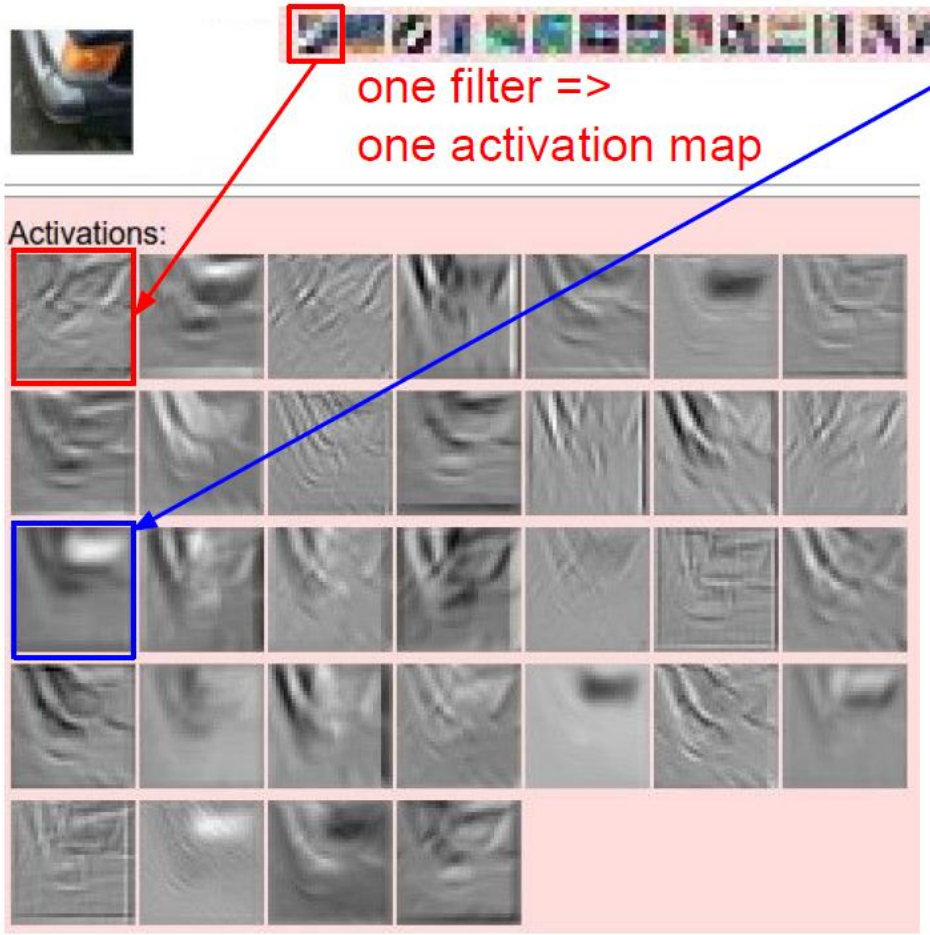
$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

CNN

A closer look at spatial dimensions:



CNN



example 5x5 filters
(32 total)

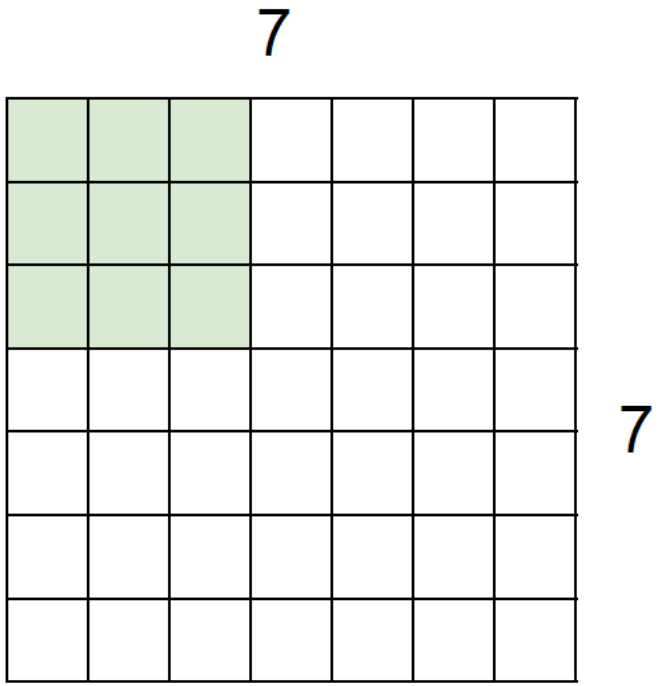
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

CNN

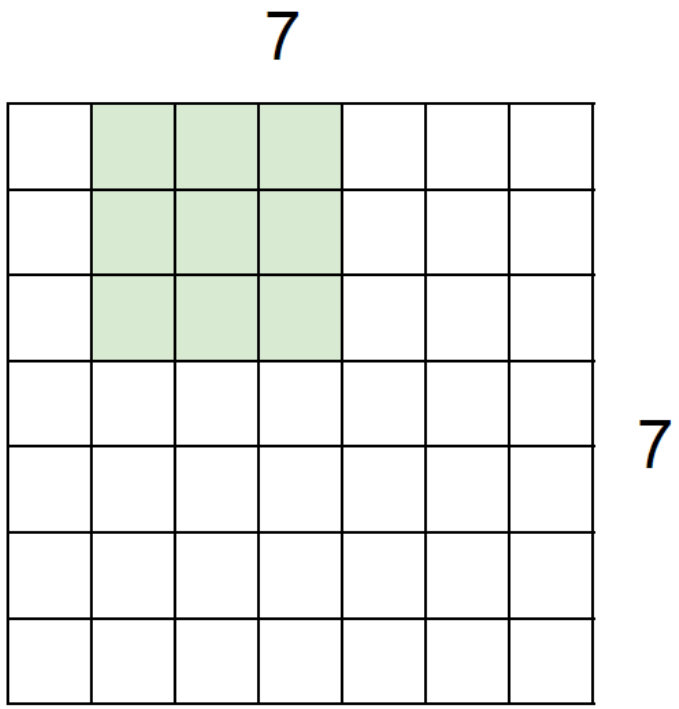
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

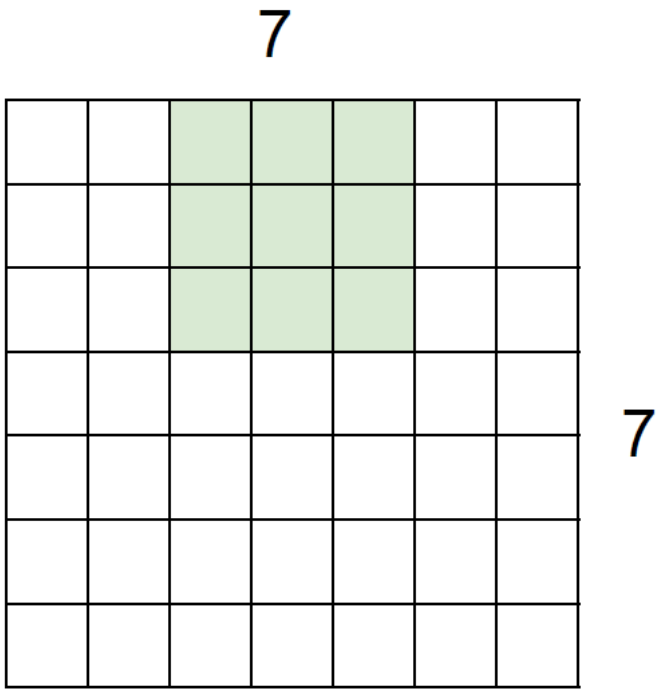
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

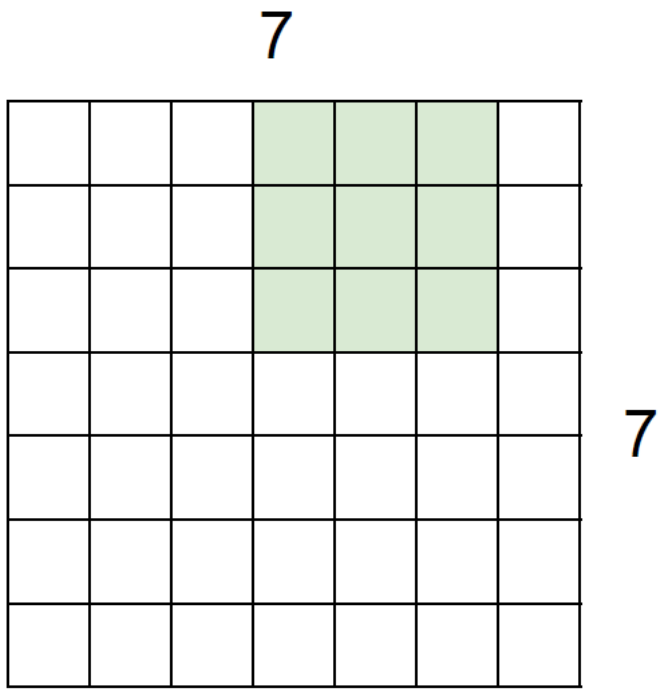
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

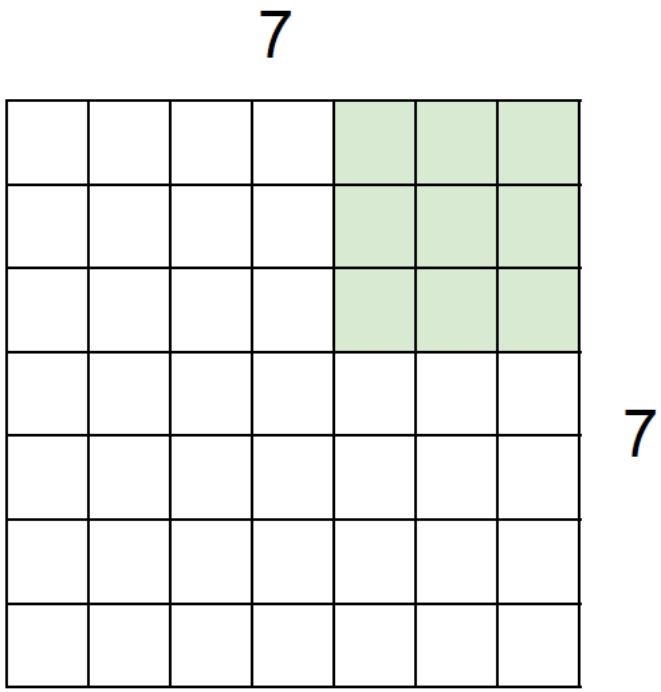
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

A closer look at spatial dimensions:

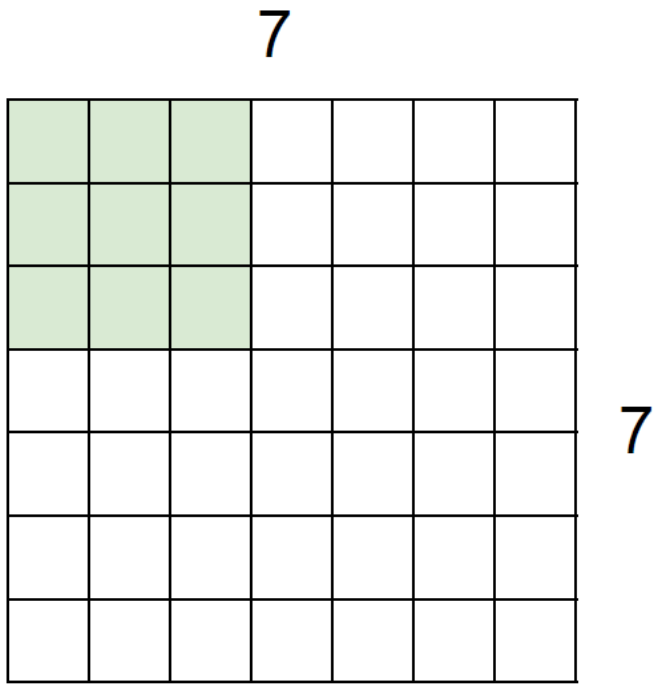


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

CNN

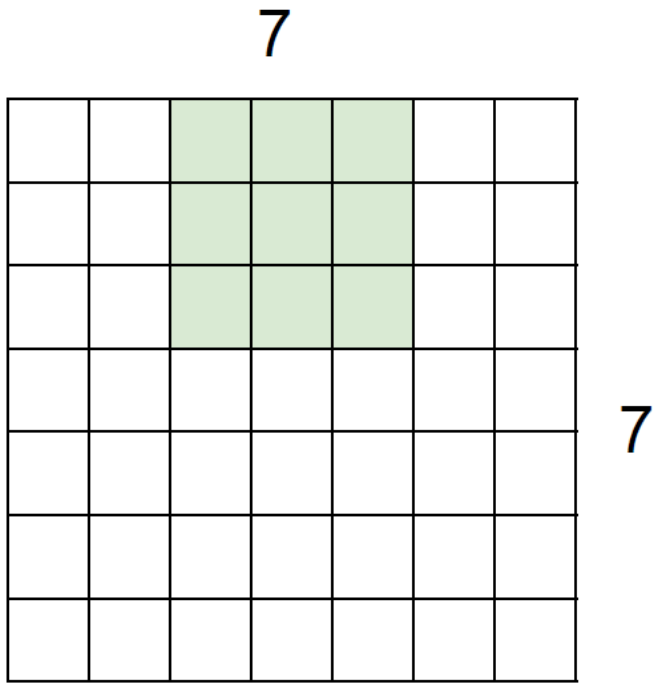
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

CNN

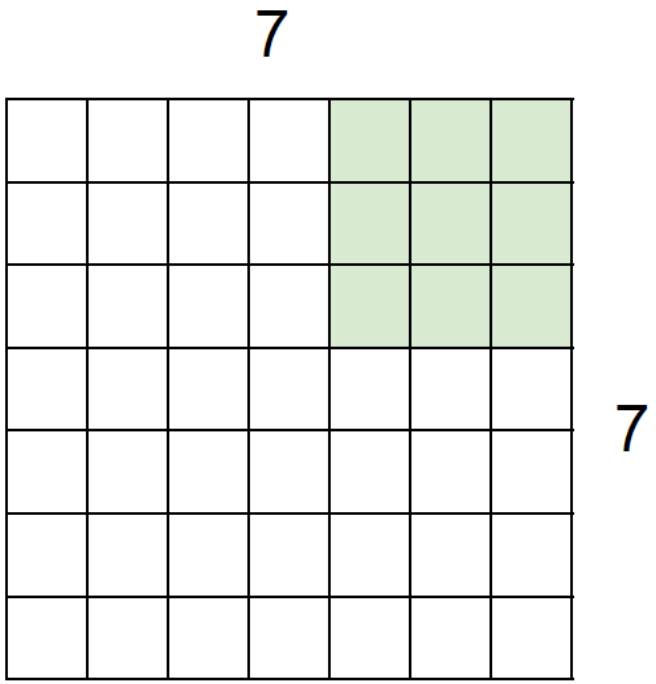
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

CNN

A closer look at spatial dimensions:

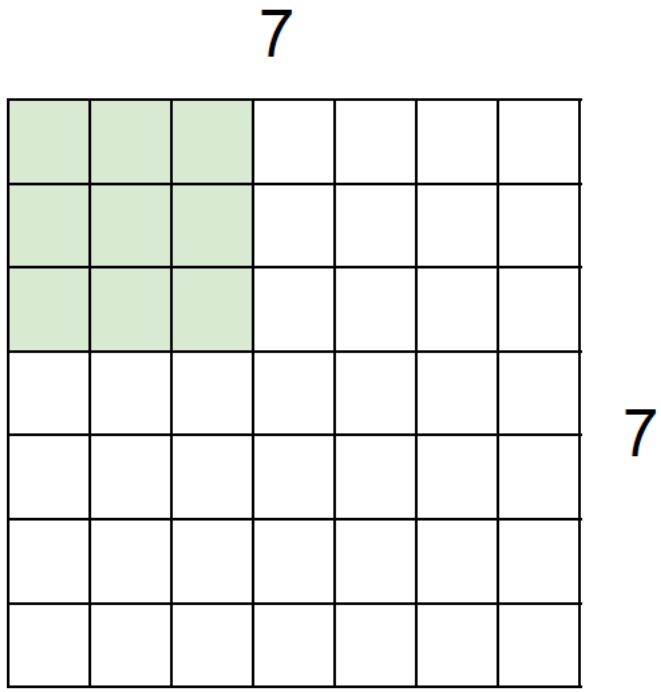


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

=> 3x3 output!

CNN

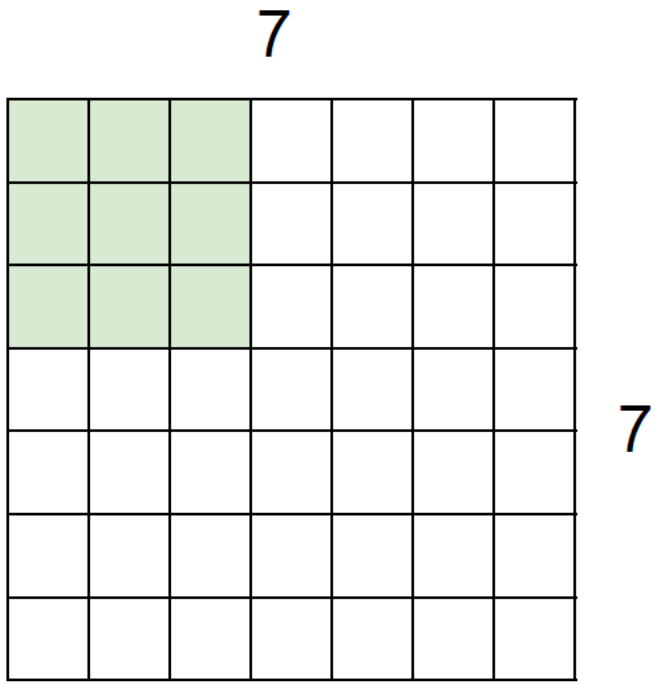
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

CNN

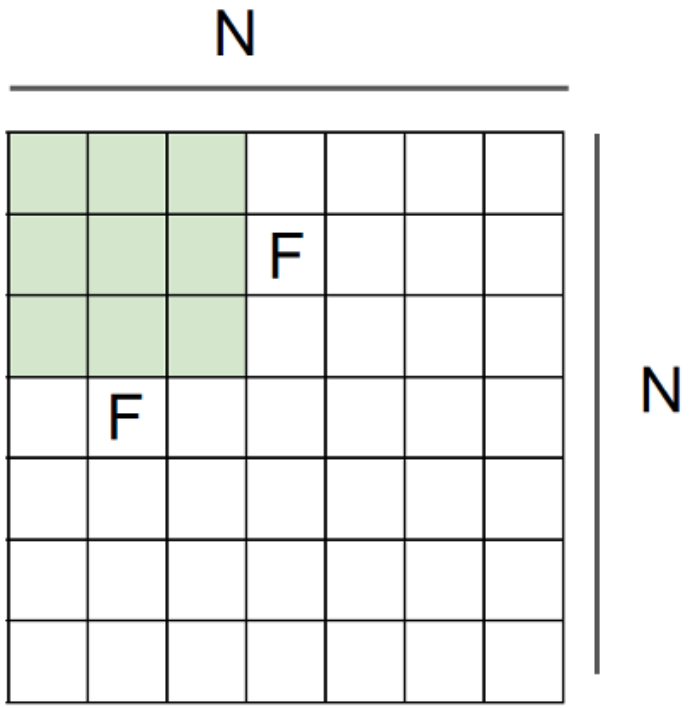
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn‘t fit!
Cannot apply 3x3 filter on 7x7 input with stride 3.

CNN

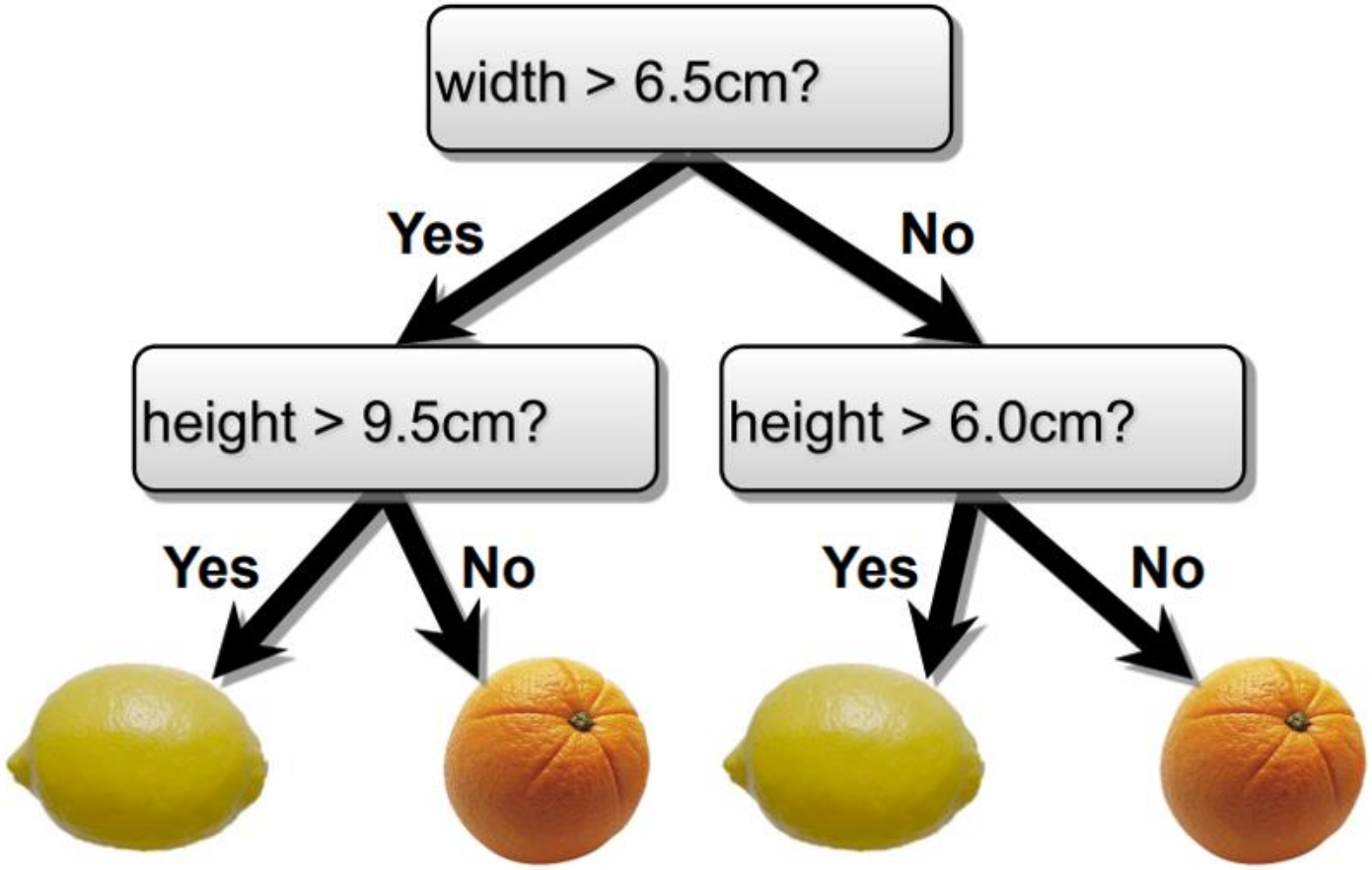


Output size:
(N - F)/stride +1

e.g., N=7, F=3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

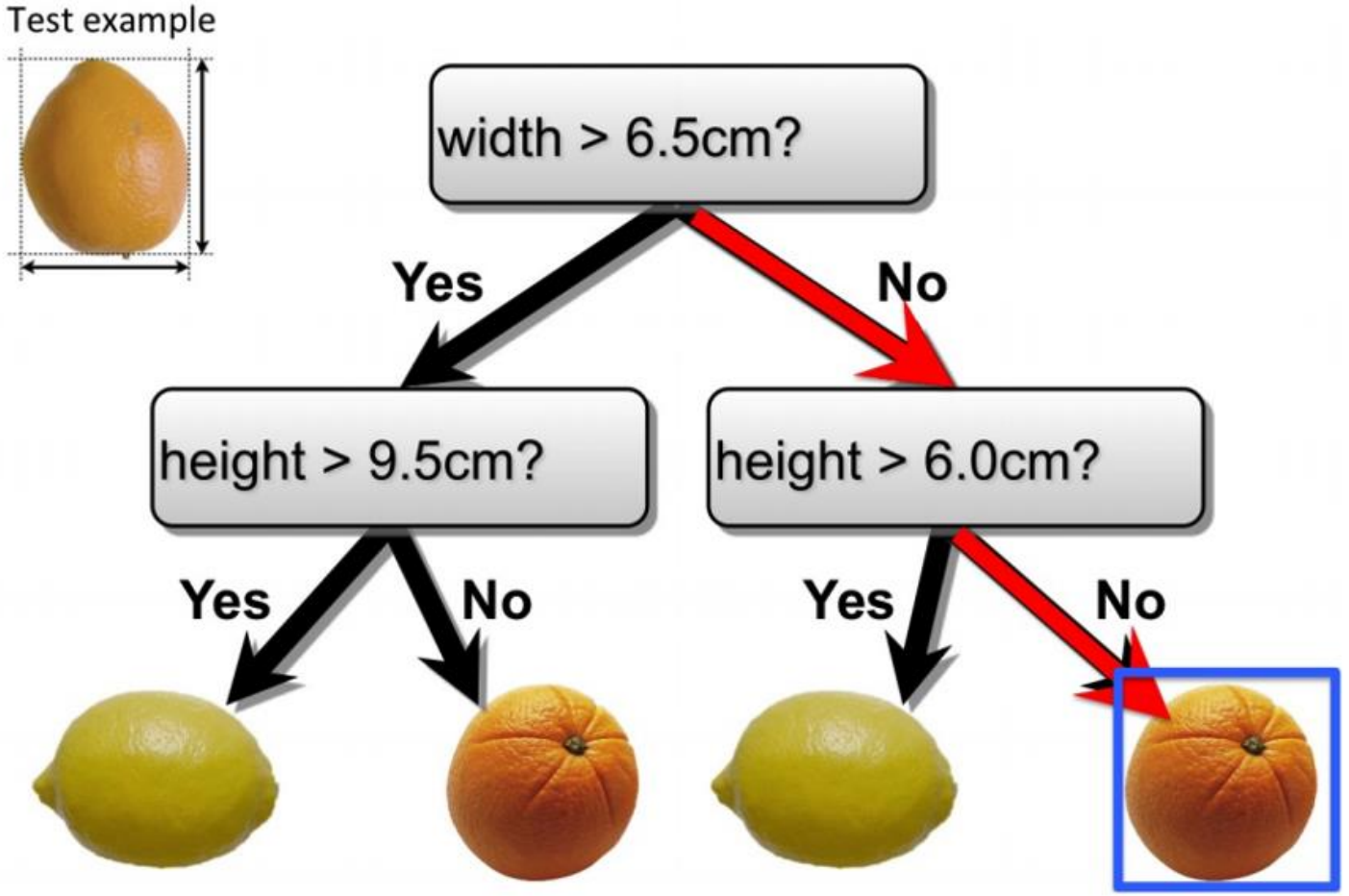
Decision Trees

- Make predictions by splitting on features according to a tree structure.



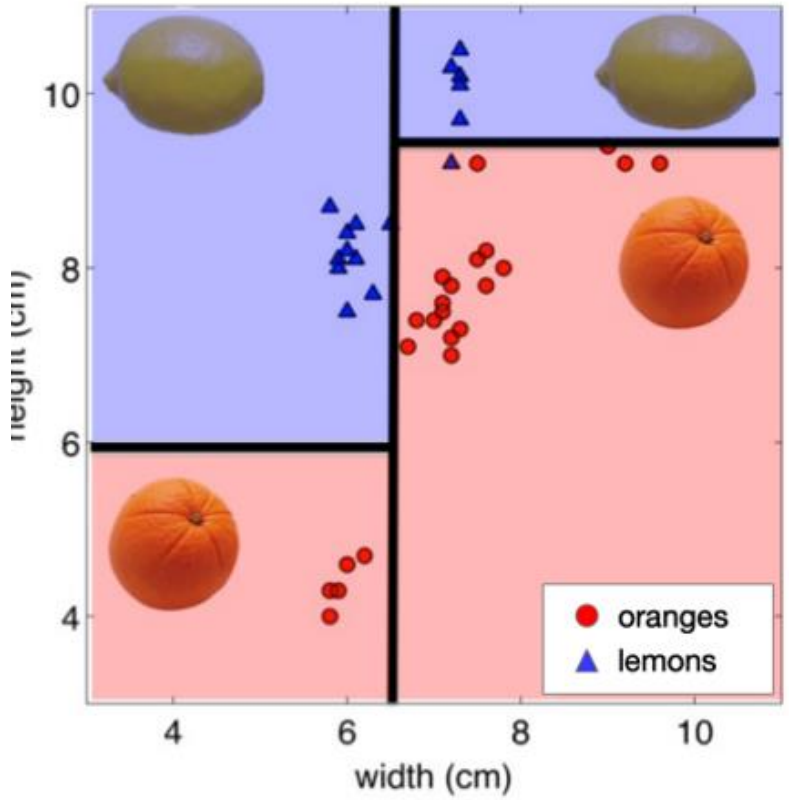
Decision Trees

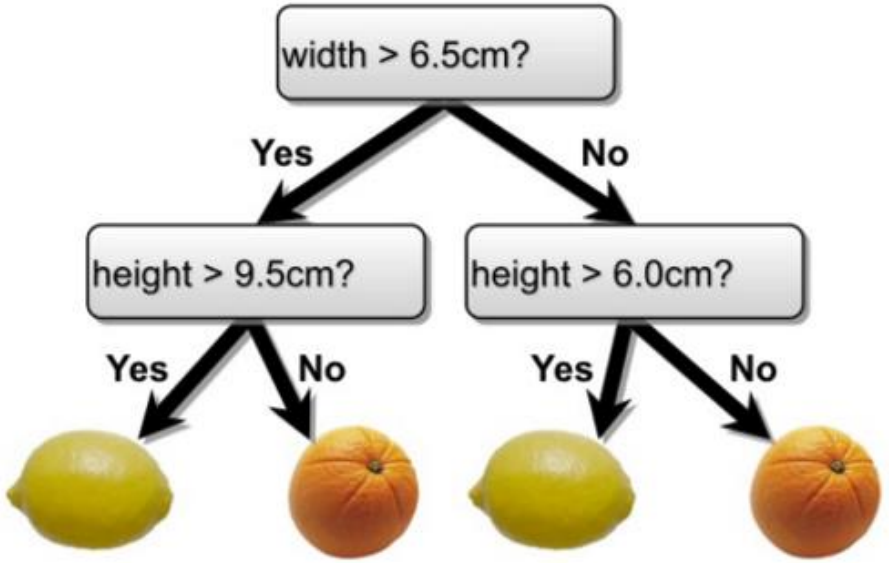
- Make predictions by splitting on features according to a tree structure.



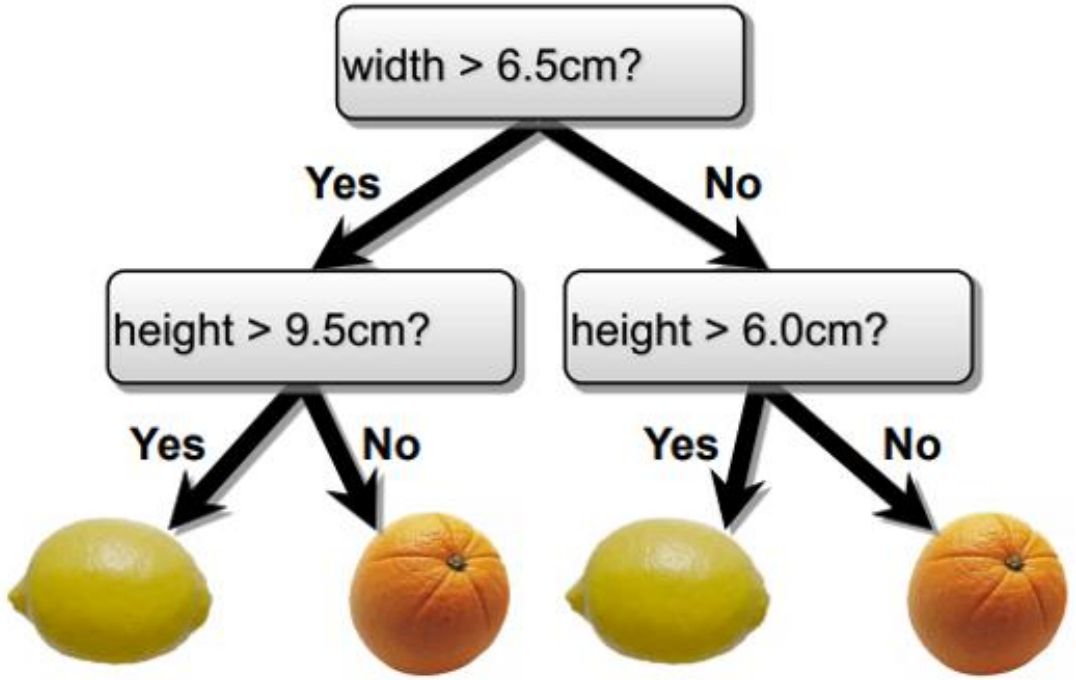
Decision Trees— Continuous Features

- Split *continuous features* by checking whether that feature is greater than or less than some threshold.
- Decision boundary is made up of axis-aligned planes.





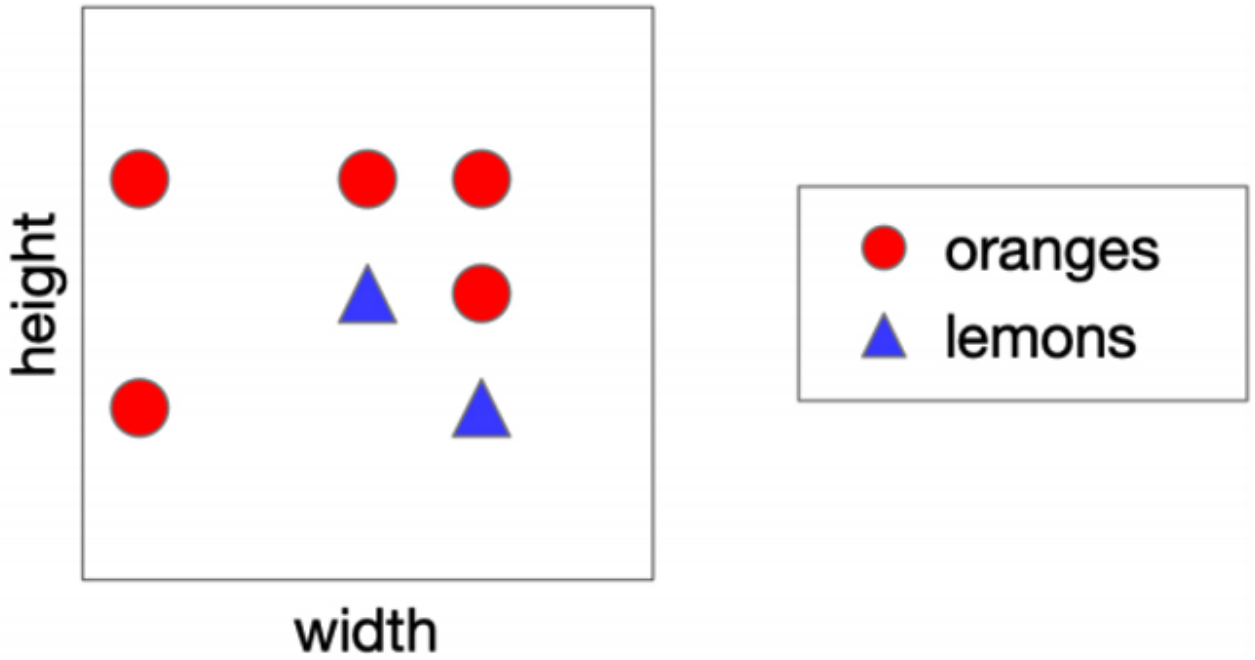
Decision Trees



- Internal nodes test a feature
- Branching is determined by the feature value
- Leaf nodes are outputs (predictions)

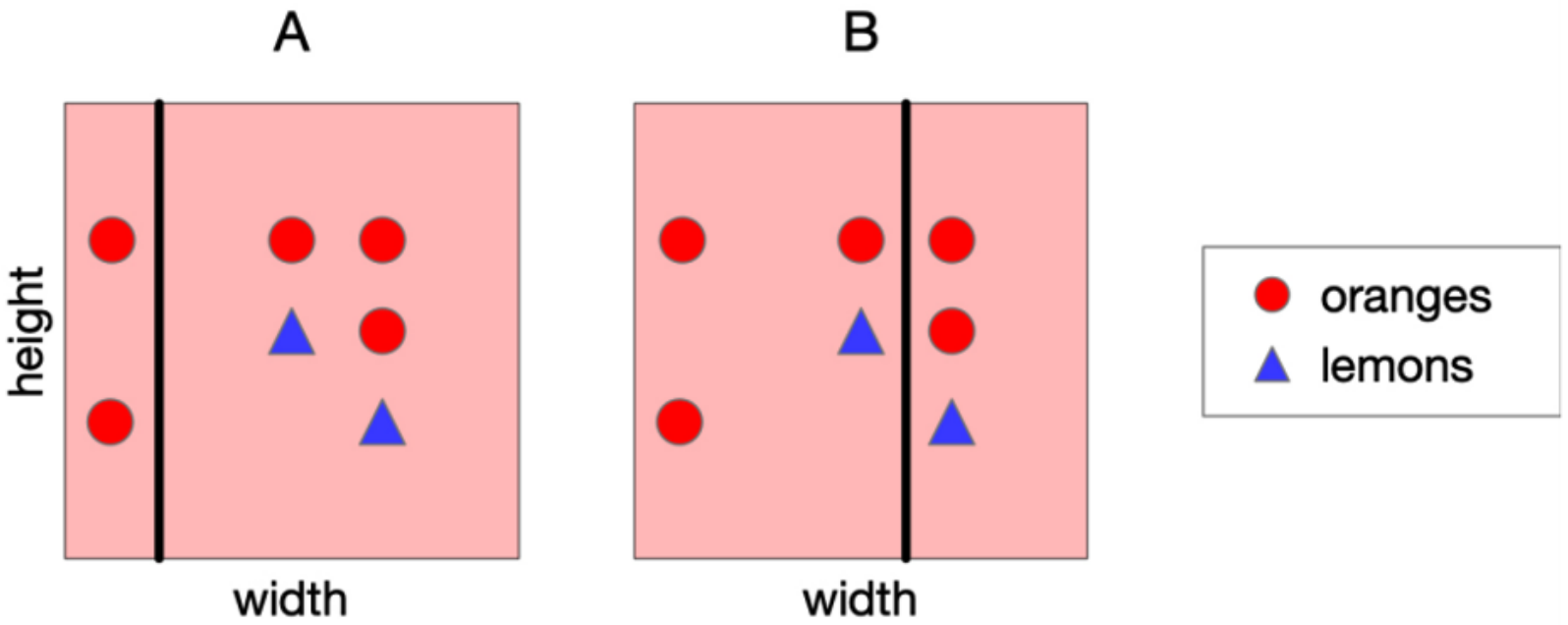
Decision Tree-Examples

- Consider the following data. Let's split on width.



Choosing a Good Split

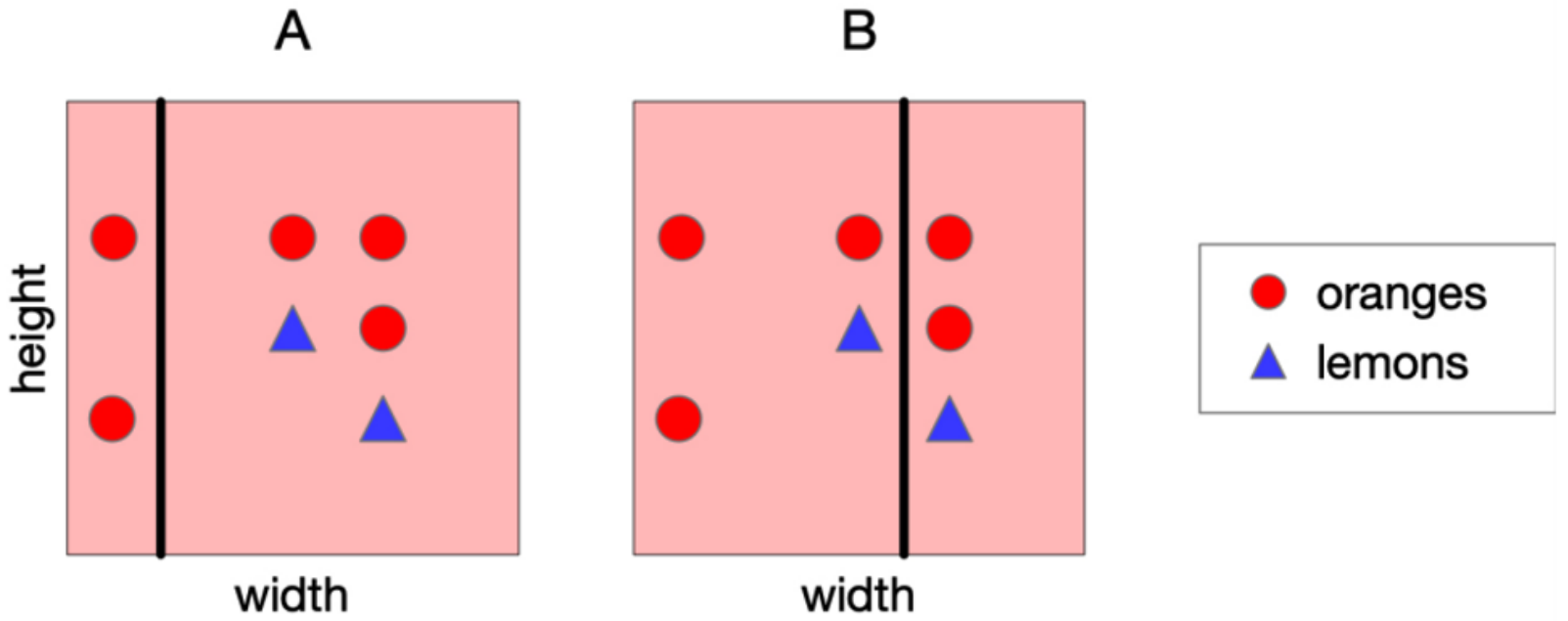
- Recall: classify by majority.



- A and B have the same misclassification rate, so which is the best split? Vote!

Choosing a Good Split

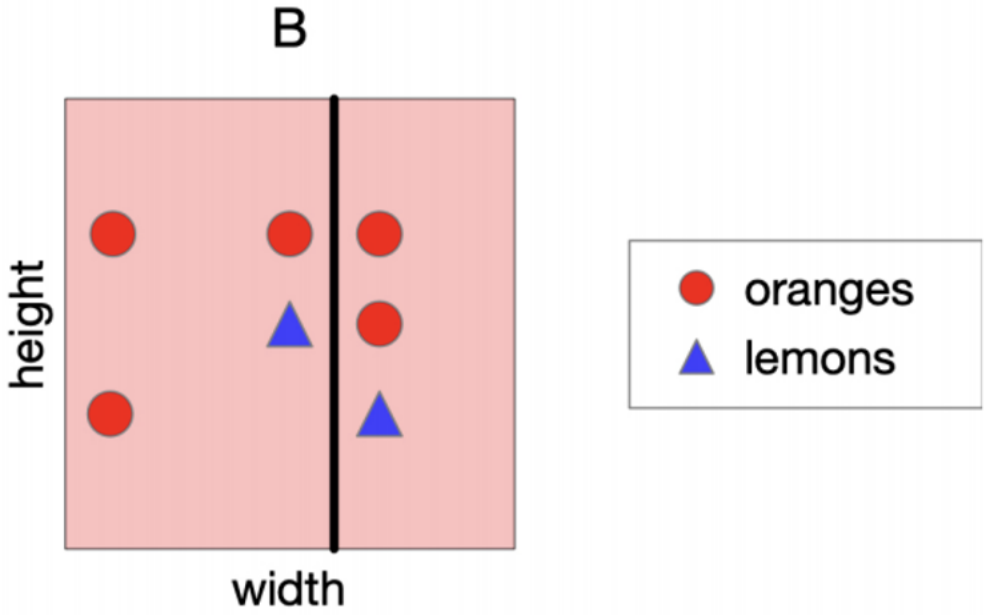
- A feels like a better split, because the left-hand region is very certain about whether the fruit is an orange.



- Can we quantify this?

Revisiting Our Original Example

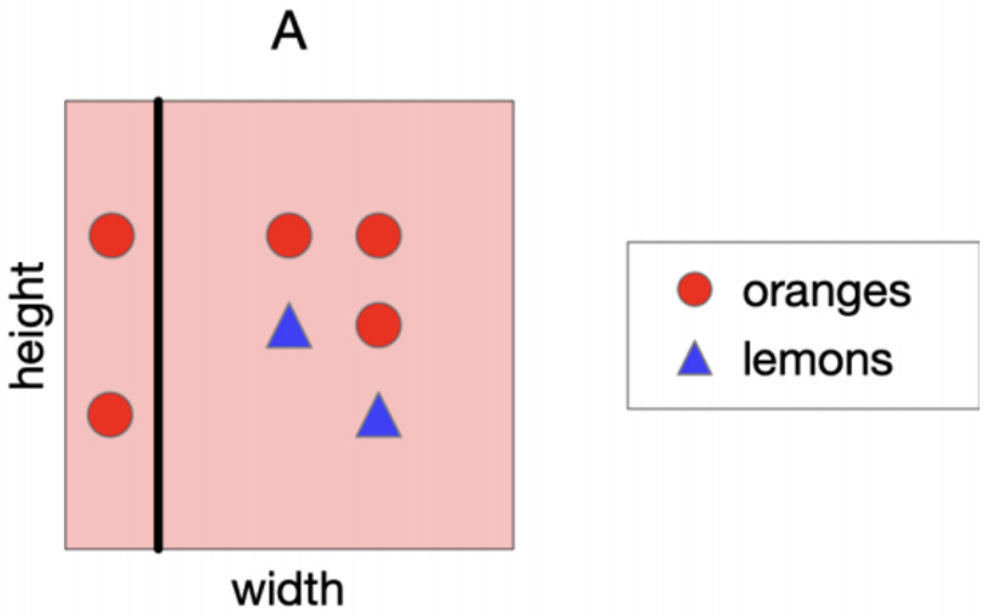
- What is the information gain of split B? Not terribly informative…



- Root entropy of class outcome: $H(Y) = -\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}) \approx 0.86$
- Leaf conditional entropy of class outcome: $H(Y|left) \approx 0.81, H(Y|right) \approx 0.92$
- $IG(split) \approx 0.86 - (\frac{4}{7} \cdot 0.81 + \frac{3}{7} \cdot 0.92) \approx 0.006$

Revisiting Our Original Example

- What is the information gain of split A? Very informative!

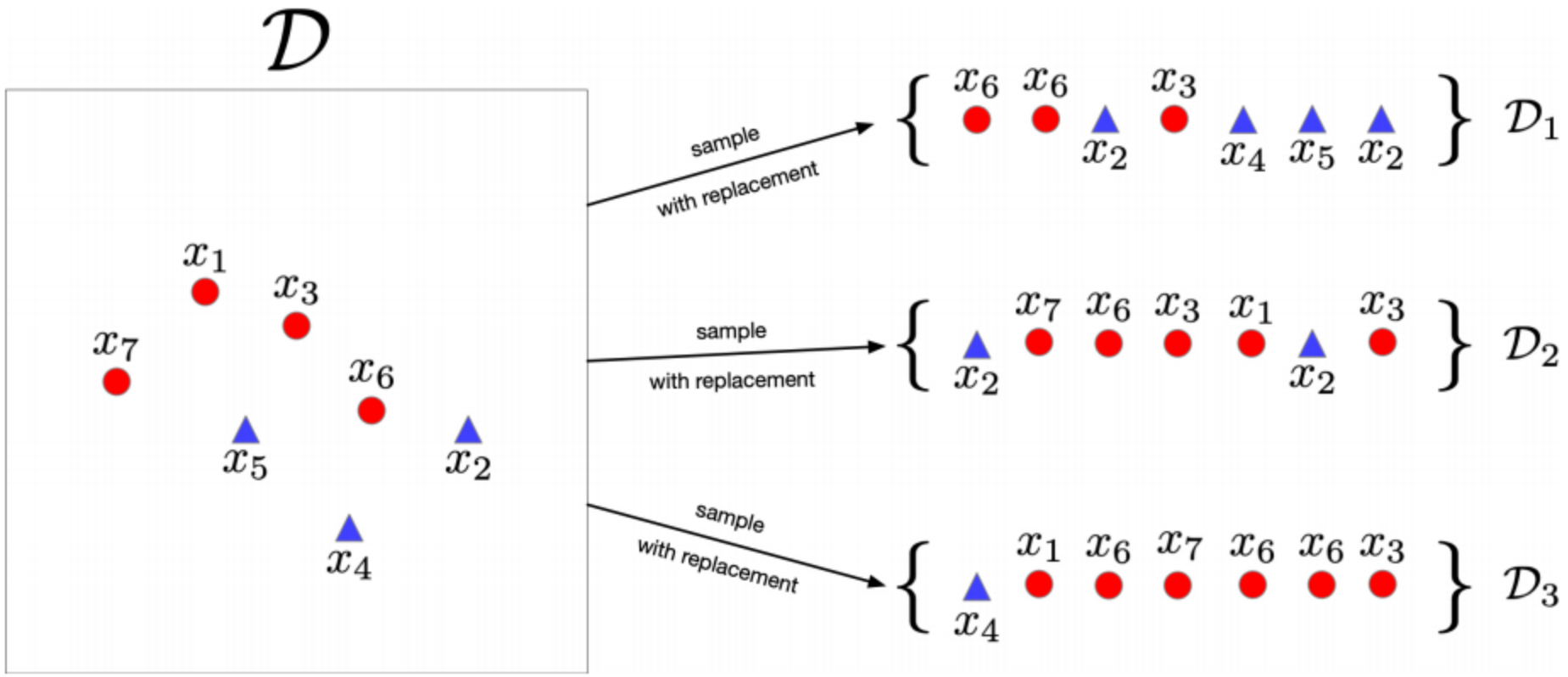


- Root entropy of class outcome: $H(Y) = -\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}) \approx 0.86$
- Leaf conditional entropy of class outcome: $H(Y|left) \approx 0, H(Y|right) \approx 0.97$
- $IG(split) \approx 0.86 - (\frac{2}{7} \cdot 0 + \frac{5}{7} \cdot 0.97) \approx 0.17!$

Ensemble methods - Bagging: The Idea

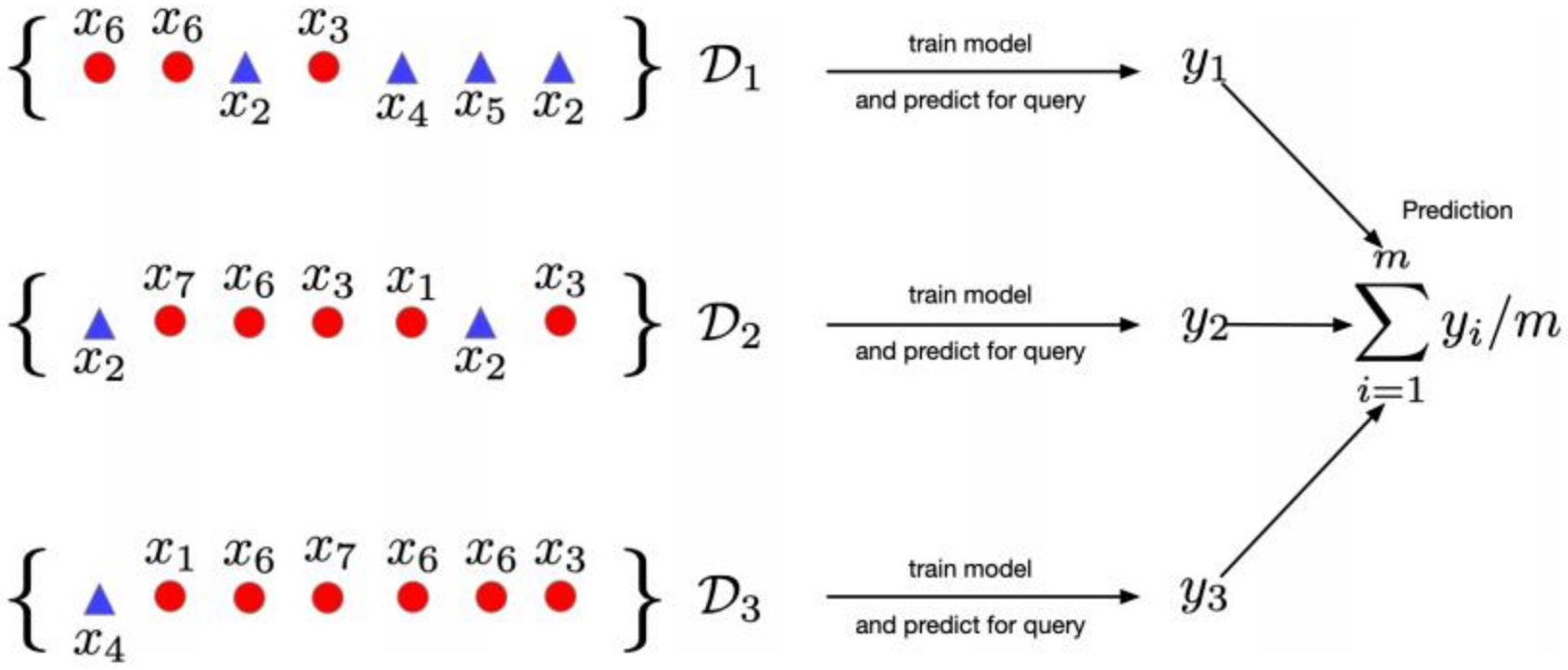
- In practice, the sampling distribution p_{sample} is often finite or expensive to sample from.
- So training separate models on independently sampled datasets is very wasteful of data!
  - ➢ Why not train a single model on the union of all sampled datasets?
- Solution: given training set $\mathcal{D}$, use the empirical distribution $p_{\mathcal{D}}$ as a proxy for p_{sample}. This is called bootstrap aggregation, or bagging.
  - ➢ Take a single dataset $\mathcal{D}$ with n examples.
  - ➢ Generate m new datasets ("resamples" or "bootstrap samples"), each by sampling n training examples from $\mathcal{D}$, with replacement.
  - ➢ Average the predictions of models trained on each of these datasets.
- The bootstrap is one of the most important ideas in all of statistics!
  - ➢ Intuition: As $|\mathcal{D}| \to \infty$, we have $p_{\mathcal{D}} \to p_{\text{sample}}$.

Bagging



In this example $n = 7, m = 3$

Bagging



Predicting on a query point x

Ensemble methods - Ada Boost Algorithm

- Input: Data $\mathcal{D}_N$, weak classifier WeakLearn (a classification procedure that returns a classifier h, e.g. best decision stump, from a set of classifiers $\mathcal{H}$, e.g., all possible decision stumps), number of iterations T
- Output: Classifier $H(x)$
- Initialize sample weights: $w^{(n)} = \frac{1}{N}$ for $n = 1, \cdots, N$
- For $t = 1, \cdots, T$
  - Fit a classifier to weighted data ($h_t \leftarrow \text{WeakLearn}(\mathcal{D}_N, \mathbf{w})$), e.g.,

$$h_t \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h(\mathbf{x}^{(n)}) \neq t^{(n)}\}$$

  - Compute weighted error $\text{err}_t = \frac{\sum_{n=1}^{N} w^{(n)} \mathbb{I}\{h_t(x^{(n)}) \neq t^{(n)}\}}{\sum_{n=1}^{N} w^{(n)}}$
  - Compute classifier coefficient $\alpha_t = \frac{1}{2} \log \frac{1-\text{err}_t}{\text{err}_t}$ $(\in (0, \infty))$
  - Update data weights

$$w^{(n)} \leftarrow w^{(n)} \exp\left(-\alpha_t t^{(n)} h_t(\mathbf{x}^{(n)})\right) \left[\equiv w^{(n)} \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)\right]$$

- Return $H(\mathrm{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

Bagging & Boosting

- Ensembles combine classifiers to improve performance
- Boosting
  - Reduces bias
  - Increases variance (large ensemble can cause overfitting)
  - Sequential
  - High dependency between ensemble elements
- Bagging
  - Reduces variance (large ensemble can't cause overfitting)
  - Bias is not changed (much)
  - Parallel
  - Want to minimize correlation between ensemble elements.

Maximum Likelihood Estimation

- Notice, in the coin example we are actually minimizing cross-entropies!

$$\begin{aligned}\hat{\theta}_{\mathrm{ML}} &= \max_{\theta \in [0,1]} \ell(\theta) \\ &= \min_{\theta \in [0,1]} -\ell(\theta) \\ &= \min_{\theta \in [0,1]} \sum_{i=1}^{N} -x_i \log \theta - (1 - x_i) \log(1 - \theta)\end{aligned}$$

- This is an example of maximum likelihood estimation.
  - define a model that assigns a probability (or has a probability density at) to a dataset
  - maximize the likelihood (or minimize the neg. log-likelihood).
- Many examples we've considered fall in this framework! Let's consider classification again.

Naive Bayes

- Naive assumption: Naive Bayes assumes that the word features x_i are conditionally independent given the class c.
  - This means x_i and x_j are independent under the conditional distribution $p(\mathbf{x}|c)$.
  - Note: this doesn't mean they're independent.
  - Mathematically,

$$p(c, x_1, \ldots, x_D) = p(c)p(x_1|c)\cdots p(x_D|c).$$

- Compact representation of the joint distribution
  - Prior probability of class: $p(c = 1) = \pi$ (e.g. spam email)
  - Conditional probability of word feature given class: $p(x_j = 1|c) = \theta_{jc}$ (e.g. word "price" appearing in spam)
  - $2D + 1$ parameters total (before $2^{D+1} - 1$)

Naive Bayes: Learning

- The parameters can be learned efficiently because the log-likelihood decomposes into independent terms for each feature.

$$
\begin{aligned}
\ell(\boldsymbol{\theta}) &= \sum_{i=1}^{N} \log p(c^{(i)}, \mathbf{x}^{(i)}) = \sum_{i=1}^{N} \log \left\{ p(\mathbf{x}^{(i)} | c^{(i)}) p(c^{(i)}) \right\} \\
&= \sum_{i=1}^{N} \log \left\{ p(c^{(i)}) \prod_{j=1}^{D} p(x_j^{(i)} \mid c^{(i)}) \right\} \\
&= \sum_{i=1}^{N} \left[\log p(c^{(i)}) + \sum_{j=1}^{D} \log p(x_j^{(i)} \mid c^{(i)}) \right] \\
&= \underbrace{\sum_{i=1}^{N} \log p(c^{(i)})}_{\substack{\text{Bernoulli log-likelihood} \\ \text{of labels}}} + \sum_{j=1}^{D} \underbrace{\sum_{i=1}^{N} \log p(x_j^{(i)} \mid c^{(i)})}_{\substack{\text{Bernoulli log-likelihood} \\ \text{for feature } x_j}}
\end{aligned}
$$

- Each of these log-likelihood terms depends on different sets of parameters, so they can be optimized independently.

Naive Bayes: Learning

- We can handle these terms separately. For the prior we maximize: $\sum_{i=1}^{N} \log p(c^{(i)})$
- This is a minor variant of our coin flip example. Let $p(c^{(i)} = 1) = \pi$. Note $p(c^{(i)}) = \pi^{c^{(i)}}(1-\pi)^{1-c^{(i)}}$
- Log-likelihood:

$$\sum_{i=1}^{N} \log p(c^{(i)}) = \sum_{i=1}^{N} c^{(i)} \log \pi + \sum_{i=1}^{N} (1 - c^{(i)}) \log(1-\pi)$$

- Obtain MLEs by setting derivatives to zero:

$$\hat{\pi} = \frac{\sum_i \mathbb{I}[c^{(i)} = 1]}{N} = \frac{\#\text{ spams in dataset}}{\text{total }\#\text{ samples}}$$

Naive Bayes: Learning

- Each θ_{jc}'s can be treated separately: maximize $\sum_{i=1}^{N} \log p(x_j^{(i)}|c^{(i)})$
- This is (again) a minor variant of our coin flip example. Let $\theta_{jc} = p(x_j^{(i)} = 1|c)$. Note $p\left(x_j^{(i)}\middle|c\right) = \theta_{jc}^{x_j^{(i)}} (1-\theta_{jc})^{1-x_j^{(i)}}$.
- Log-likelihood:

$$\sum_{i=1}^{N} \log p(x_j^{(i)} \mid c^{(i)}) = \sum_{i=1}^{N} c^{(i)} \left\{ x_j^{(i)} \log \theta_{j1} + (1 - x_j^{(i)}) \log(1 - \theta_{j1}) \right\}$$
$$+ \sum_{i=1}^{N} (1 - c^{(i)}) \left\{ x_j^{(i)} \log \theta_{j0} + (1 - x_j^{(i)}) \log(1 - \theta_{j0}) \right\}$$

- Obtain MLEs by setting derivatives to zero:

$$\hat{\theta}_{jc} = \frac{\sum_i \mathbb{I}[x_j^{(i)} = 1 \ \& \ c^{(i)} = c]}{\sum_i \mathbb{I}[c^{(i)} = c]} \overset{\text{for } c=1}{=} \frac{\#\text{word } j \text{ appears in spams}}{\#\text{ spams in dataset}}$$

Naive Bayes: Inference

- We predict the category by performing inference in the model.
- Apply Bayes' Rule:

$$p(c \mid \mathbf{x}) = \frac{p(c)p(\mathbf{x} \mid c)}{\sum_{c'} p(c')p(\mathbf{x} \mid c')} = \frac{p(c)\prod_{j=1}^{D} p(x_j \mid c)}{\sum_{c'} p(c')\prod_{j=1}^{D} p(x_j \mid c')}$$

- We need not compute the denominator if we're simply trying to determine the most likely c.
- Shorthand notation:

$$p(c \mid \mathbf{x}) \propto p(c)\prod_{j=1}^{D} p(x_j \mid c)$$

- For input $\mathbf{x}$, predict by comparing the values of $p(c)\prod_{j=1}^{D} p(x_j|c)$ for different c (e.g. choose the largest).

Naive Bayes

- Naive Bayes is an amazingly cheap learning algorithm!
- Training time: estimate parameters using maximum likelihood
  - Compute co-occurrence counts of each feature with the labels.
  - Requires only one pass through the data!
- Test time: apply Bayes' Rule
  - Cheap because of the model structure. (For more general models, Bayesian inference can be very expensive and/or complicated.)
- We covered the Bernoulli case for simplicity. But our analysis easily extends to other probability distributions.
- Unfortunately, it's usually less accurate in practice compared to discriminative models due to its "naive" independence assumption.

Gaussian Mixture Model (GMM)

What is $p(x)$?

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(z=k)p(\mathbf{x}|z=k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \mathbf{I})$$

- This distribution is an example of a Gaussian Mixture Model (GMM), and π_k are known as the mixing coefficients

- In general, we would have different covariance for each cluster, i.e., $p(\mathbf{x}|z=k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.For this lecture, we assume $\boldsymbol{\Sigma}_k = \mathbf{I}$ for simplicity.

EM Algorithm for GMM

- Initialize the means $\hat{\boldsymbol{\mu}}_k$ and mixing coefficients $\hat{\pi}_k$
- Iterate until convergence:
  - E-step: Evaluate the responsibilities $r_k^{(n)}$ given current parameters

$$r_k^{(n)} = p(z^{(n)}=k|\mathbf{x}^{(n)}) = \frac{\hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \mathbf{I})}{\sum_{j=1}^{K} \hat{\pi}_j \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_j, \mathbf{I})} = \frac{\hat{\pi}_k \exp\{-\frac{1}{2}\|\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_k\|^2\}}{\sum_{j=1}^{K} \hat{\pi}_j \exp\{-\frac{1}{2}\|\mathbf{x}^{(n)} - \hat{\boldsymbol{\mu}}_j\|^2\}}$$

  - M-step: Re-estimate the parameters given current responsibilities

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_k^{(n)} \mathbf{x}^{(n)}$$

$$\hat{\pi}_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^{N} r_k^{(n)}$$

  - Evaluate log likelihood and check for convergence

$$\log p(\mathcal{D}) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \hat{\pi}_k \mathcal{N}(\mathbf{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \mathbf{I}) \right)$$

Relation to k-Means

- The K-Means Algorithm:

  1. Assignment step: Assign each data point to the closest cluster
  2. Refitting step: Move each cluster center to the average of the data assigned to it

- The EM Algorithm:

  1. E-step: Compute the posterior probability over z given our current model
  2. M-step: Maximize the probability that it would generate the data it is currently responsible for.