Lecture 4

Tian Han

Outline

- Multi-class classification
- K-nearest neighbor

Multi-Class Classification

Multi-Class Classification

Tasks

Methods

Algorithms

Multi-Class Classification

Example 1: face recognition.

- #classes = #people

Multi-Class Classification

Example 2: hand-written digit recognition.

- #classes = 10

Can we use linear regression?

Looks like "3" ➔ $f = \mathbf{w}^T\mathbf{x}$ is close to 3

$\mathbf{x}$

Looks like "8" ➔ $f = \mathbf{w}^T\mathbf{x}$ is close to 8

Can we use linear regression?

Looks like "3" $\rightarrow$ $f = \mathbf{w}^T\mathbf{x}$ is close to 3

$f = \mathbf{w}^T\mathbf{x}$ is 4.835

Looks like "8" $\rightarrow$ $f = \mathbf{w}^T\mathbf{x}$ is close to 8

$\mathbf{x}$

Can we use linear regression?

$\mathbf{x}$

Looks like "3" ➔ $f = \mathbf{w}^T\mathbf{x}$ is close to 3

$f = \mathbf{w}^T\mathbf{x}$ is 4.835

Looks like "8" ➔ $f = \mathbf{w}^T\mathbf{x}$ is close to 8

rounding

Linear regression believes $\mathbf{x}$ is "5"

The Right Approach

vectorization

$\mathbf{x}$

$$\phi = \mathbf{W}\mathbf{x} = \begin{array}{cl} \begin{bmatrix} -5 \\ -8 \\ 0.4 \\ 15 \\ -2 \\ 0.9 \\ 2.3 \\ -4.2 \\ 13.1 \\ 2.3 \end{bmatrix} & \begin{array}{l} \\ \\ \\ \text{Digit “3”} \\ \\ \\ \\ \\ \text{Digit “8”} \\ \\ \end{array} \end{array}$$

Preliminaries

One-Hot Encoding

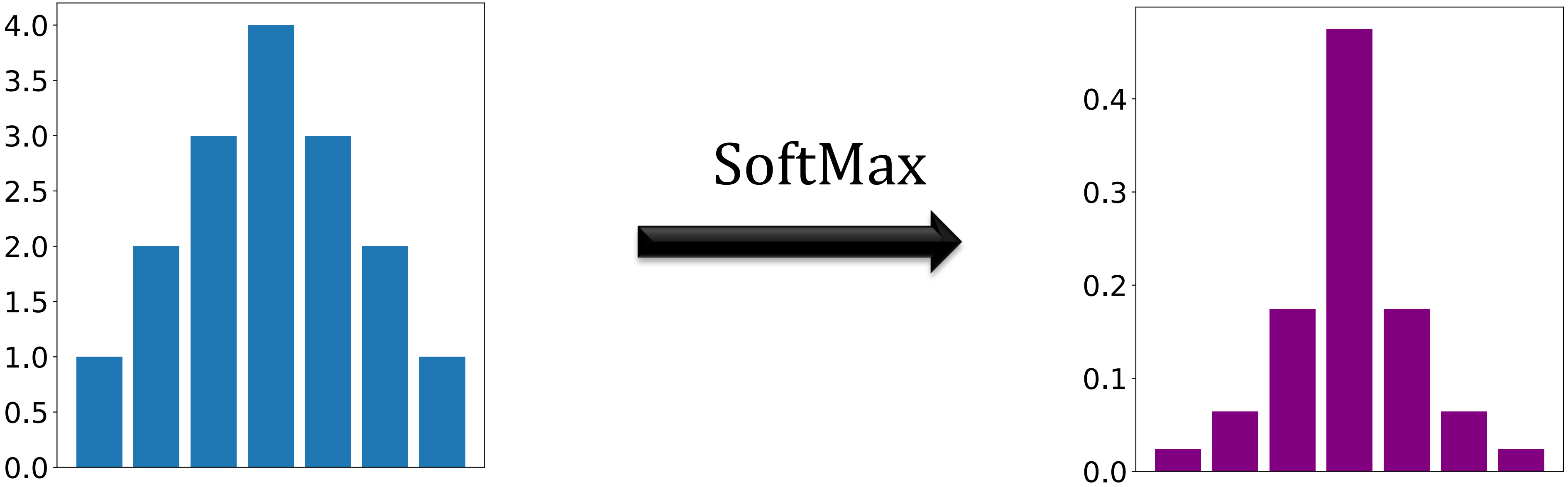
- #Class = 10 (e.g., in digit recognition).
- One-hot encode of $y = 3$:

$$\mathbf{y} \; = \; [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]^T \; \in \; \{1,0\}^{10}$$

Softmax Function

- $\boldsymbol{\phi} \in \mathbb{R}^K$
- $\mathbf{p} = \mathrm{SoftMax}(\boldsymbol{\phi}) \in \mathbb{R}^K$; its entries are

$$p_k = \frac{\exp(\phi_k)}{\sum_{j=1}^{K} \exp(\phi_j)}, \text{ for } k = 1, \cdots, K.$$



Cross-Entropy

- The vectors $\mathbf{y}$ and $\mathbf{p}$ are K-dim vectors with nonnegative entries.

$$y_1 + \cdots + y_K = 1 \quad \text{and} \quad p_1 + \cdots + p_K = 1.$$

- Cross-entropy between $\mathbf{y}$ and $\mathbf{p}$:

$$H(\mathbf{y}, \mathbf{p}) = -\sum_{l=1}^{K} y_l \log p_l \, .$$

Cross-Entropy

- The vectors $\mathbf{y}$ and $\mathbf{p}$ are K-dim vectors with nonnegative entries.

$$y_1 + \cdots + y_K = 1 \quad \text{and} \quad p_1 + \cdots + p_K = 1.$$

- Cross-entropy between $\mathbf{y}$ and $\mathbf{p}$:

$$H(\mathbf{y}, \mathbf{p}) = -\sum_{l=1}^{K} y_l \log p_l \,.$$

- Cross-entropy measures the dissimilarity between $\mathbf{y}$ and $\mathbf{p}$.
- When used as loss function, $H(\mathbf{y}, \mathbf{p})$ can be replaced by

$$||\mathbf{y} - \mathbf{p}||_2^2 \quad \text{or} \quad ||\mathbf{y} - \mathbf{p}||_1.$$

Softmax Classifier: Model Formulation

Tasks

Methods

Algorithms

Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

n: #samples d: #features K: #classes

Remark: If the given labels are scalars $y_1, \cdots, y_n \in \{0, 1, \cdots, K-1\}$, turn them to K-dim vectors $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \{0,1\}^K$ using one-hot encoding.

Example: One-hot encode of $y_i = 3$ (where K=10):

$$\mathbf{y}_i = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]^T \in \{1,0\}^{10}$$

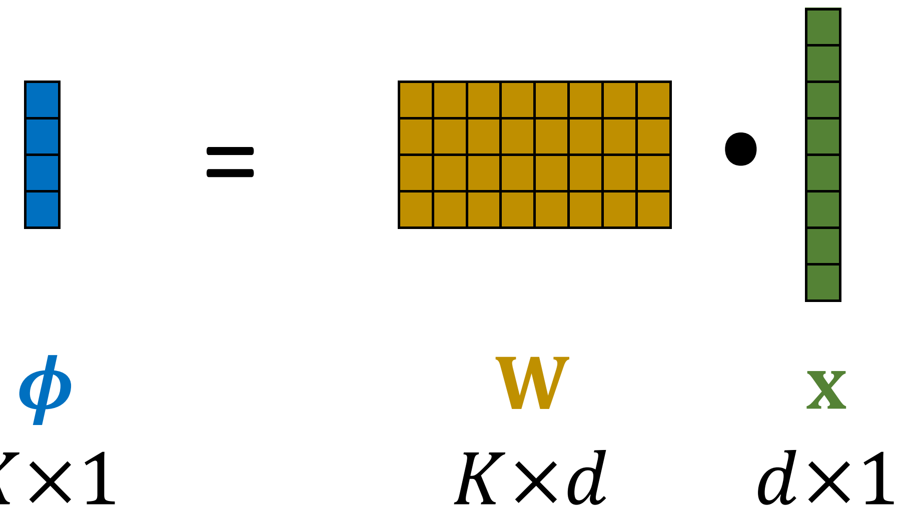
Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

n: #samples d: #features K: #classes

- $\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$

Here, $\mathbf{x}$ is one of $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$



Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

n: #samples $\quad$ d: #features $\quad$ K: #classes

- $\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$
- ϕ_k (the k-th entry of $\boldsymbol{\phi}$) indicates how likely $\mathbf{x}$ is in the k-th class.

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} = \begin{bmatrix} -5 \\ -8 \\ 0.4 \\ 15 \\ -2 \\ 0.9 \\ 2.3 \\ -4.2 \\ 13.1 \\ 2.3 \end{bmatrix} \begin{matrix} \\ \\ \\ \rightarrow \text{Digit “3”} \\ \\ \\ \\ \\ \rightarrow \text{Digit “8”} \\ \\ \end{matrix}$$

Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

n: #samples d: #features K: #classes

- $\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$
- ϕ_k (the k-th entry of $\boldsymbol{\phi}$) indicates how likely $\mathbf{x}$ is in the k-th class.
- Softmax function: $p_k = \frac{\exp(\phi_k)}{\sum_{j=1}^{K} \exp(\phi_j)}$.

- $p_1 + \cdots + p_K = 1$.
- Thus $\mathbf{p} = [p_1, \cdots, p_K] \in \mathbb{R}^K$ is a distribution.

Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

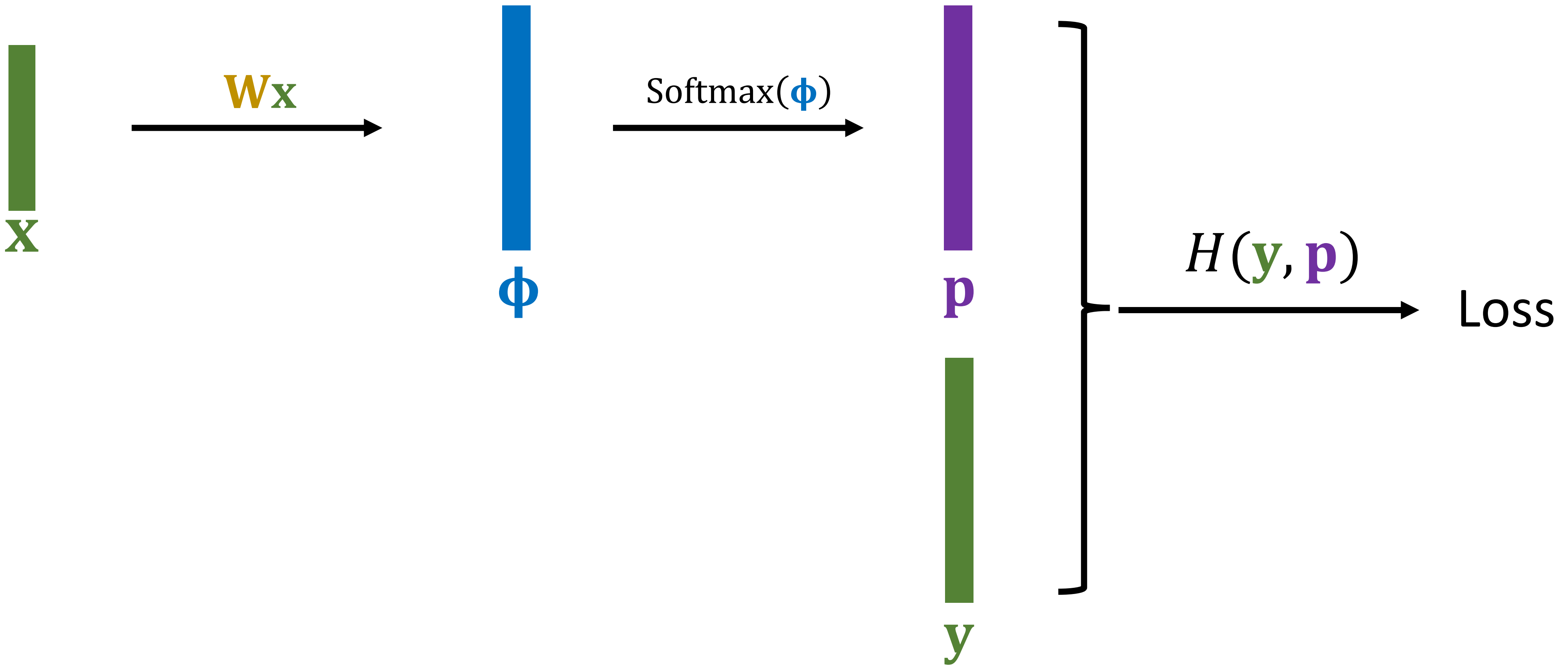
n: #samples d: #features K: #classes

- $\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$
- ϕ_k (the k-th entry of $\boldsymbol{\phi}$) indicates how likely $\mathbf{x}$ is in the k-th class.
- Softmax function: $p_k = \frac{\exp(\phi_k)}{\sum_{j=1}^{K} \exp(\phi_j)}$.
- Cross-entropy loss: $H(\mathbf{y}, \mathbf{p}) = -\sum_{k=1}^{K} y_k \log p_k$.

Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

n: #samples d: #features K: #classes



Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

- Softmax classifier: $\min_{\mathbf{W}} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)$

$$\boldsymbol{\phi}_i = \mathbf{W}\mathbf{x}_i \in \mathbb{R}^K, \quad \mathbf{p}_i = \mathrm{SoftMax}(\boldsymbol{\phi}_i), \quad \text{and} \quad H(\mathbf{y}_i, \mathbf{p}_i) = -\sum_{k=1}^{K} y_{i,k} \log(p_{i,k}).$$

Softmax function

Cross-entropy loss

Softmax Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

- Softmax classifier: $\boxed{\min_{\mathbf{W}} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)}$

$$\boldsymbol{\phi}_i = \mathbf{W}\mathbf{x}_i \in \mathbb{R}^K, \quad \mathbf{p}_i = \mathrm{SoftMax}(\boldsymbol{\phi}_i), \quad \text{and} \quad H(\mathbf{y}_i, \mathbf{p}_i) = -\sum_{k=1}^{K} y_{i,k} \log\left(p_{i,k}\right).$$

- The role of minimizing $H(\mathbf{y}_i, \mathbf{p}_i)$ is making $\mathbf{p}_i$ similar to $\mathbf{y}_i$.
- $H(\mathbf{y}_i, \mathbf{p}_i)$ can be replaced by $||\mathbf{y}_i - \mathbf{p}_i||_2^2$.
- In practice, cross-entropy works better.

Softmax Classifier: Algorithm

Tasks

Methods

Algorithms

Softmax Classifier

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K, \qquad p_q = \frac{e^{\phi_q}}{\sum_{j=1}^{K} e^{\phi_j}}, \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{q=1}^{K} y_q \log(p_q).$$

Softmax Classifier

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K, \qquad \boxed{p_q = \frac{e^{\phi_q}}{\sum_{j=1}^{K} e^{\phi_j}}}, \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{q=1}^{K} y_q \log\left(\boxed{p_q}\right).$$

- $H(\mathbf{y}, \mathbf{p}) = -\sum_{q=1}^{K} y_q \phi_q + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right) \cdot \sum_{q=1}^{K} y_q.$

Softmax Classifier

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K, \qquad p_q = \frac{e^{\phi_q}}{\sum_{j=1}^{K} e^{\phi_j}}, \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{q=1}^{K} y_q \log(p_q).$$

- $H(\mathbf{y}, \mathbf{p}) = -\sum_{q=1}^{K} y_q \phi_q + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right) \cdot \underbrace{\boxed{\sum_{q=1}^{K} y_q}}_{=1}.$

Softmax Classifier

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$

- $H(\mathbf{y}, \mathbf{p}) = -\sum_{q=1}^{K} y_q \phi_q + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right) \cdot \boxed{\sum_{q=1}^{K} y_q}.$

$= 1$

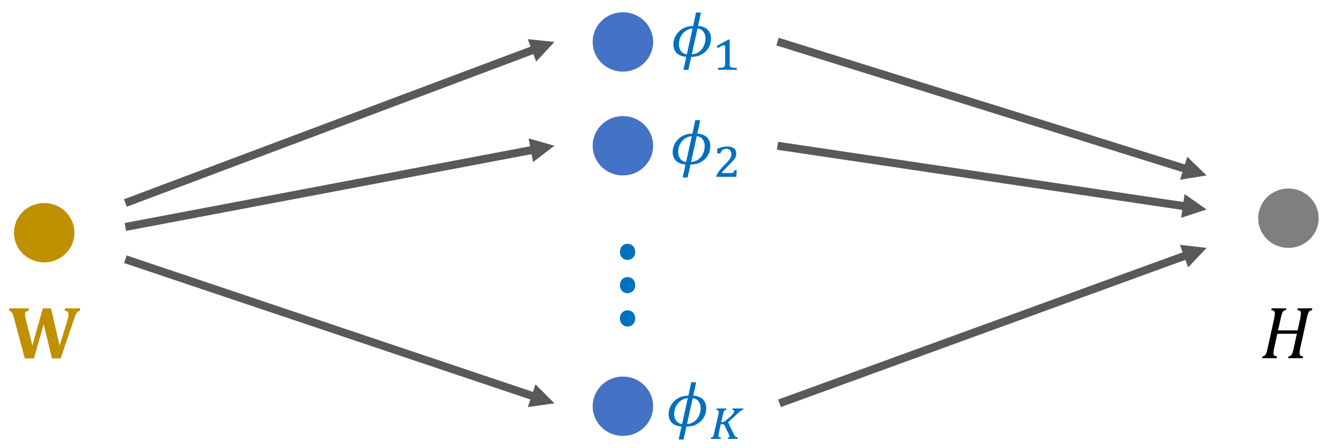
Softmax Classifier

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$

$$\begin{tikzcd} \mathbf{W} \arrow[r] & \boldsymbol{\phi} = \mathbf{W}\mathbf{x} \arrow[r] & H \end{tikzcd}$$

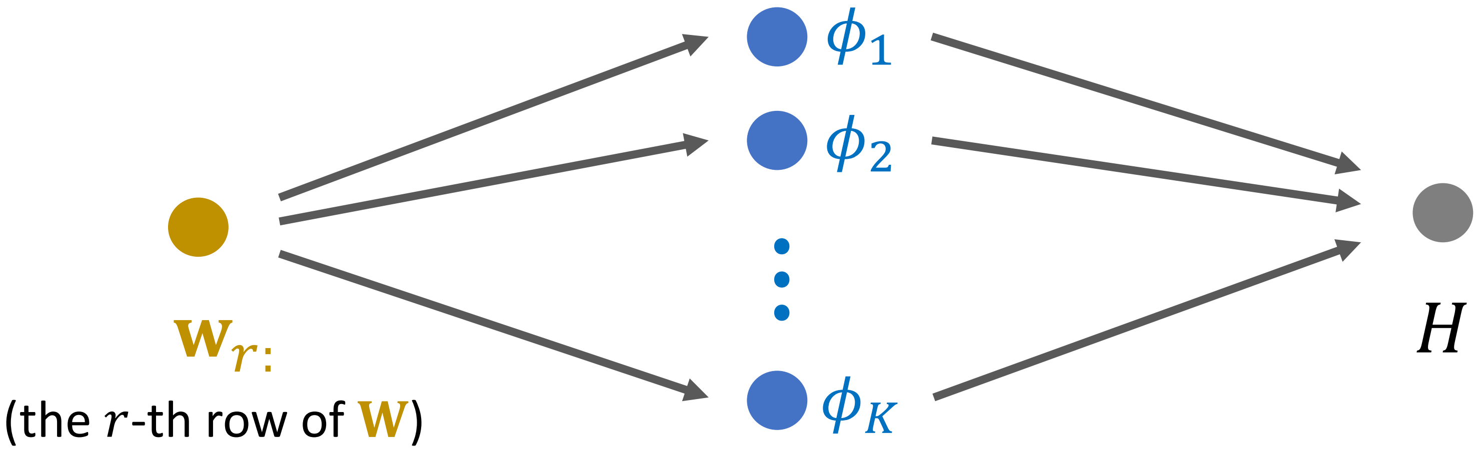
Softmax Classifier

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$



Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$

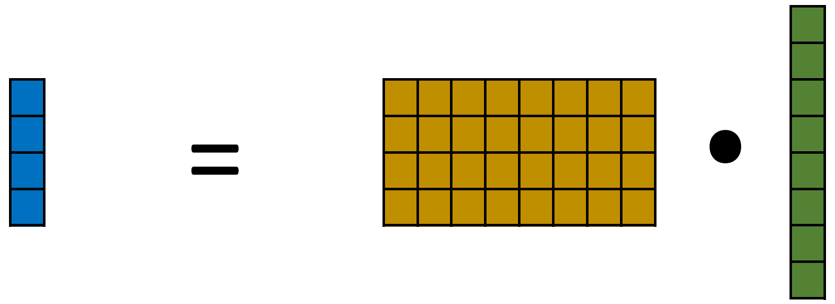


Chain rule: $\frac{\partial H}{\partial \mathbf{w}_{r:}} = \sum_{q=1}^{K} \frac{\partial \phi_q}{\partial \mathbf{w}_{r:}} \cdot \frac{\partial H}{\partial \phi_q}$

Gradient

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$$

- $\phi_q = \mathbf{x}^T \mathbf{w}_{q:}$.



$\boldsymbol{\phi}$ $K \times 1$ $\mathbf{W}$ $K \times d$ $\mathbf{x}$ $d \times 1$

Gradient

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$$

- $\phi_q = \mathbf{x}^T \mathbf{w}_{q:}$.
- $\dfrac{\partial H}{\partial \mathbf{w}_{1:}} = \sum_{q=1}^{K} \dfrac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \dfrac{\partial H}{\partial \phi_q}$

Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right)$.

- $\phi_q = \mathbf{x}^T \mathbf{w}_{q:}$.
- $\frac{\partial H}{\partial \mathbf{w}_{1:}} = \sum_{q=1}^{K} \frac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_q} = \frac{\partial \phi_1}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_1} + \sum_{q \neq 1} \frac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_q}$.

Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$

- $\phi_q = \mathbf{x}^T \mathbf{w}_{q:}$.
- $\frac{\partial H}{\partial \mathbf{w}_{1:}} = \sum_{q=1}^{K} \frac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_q} = \boxed{\frac{\partial \phi_1}{\partial \mathbf{w}_{1:}}} \cdot \frac{\partial H}{\partial \phi_1} + \sum_{q \neq 1} \frac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_q}.$

$$\boxed{= \frac{\partial\, \mathbf{x}^T \mathbf{w}_{1:}}{\partial\, \mathbf{w}_{1:}} = \mathbf{x}}$$

Gradient

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$$

- $\phi_q = \mathbf{x}^T \mathbf{w}_{q:}$.
- $\frac{\partial H}{\partial \mathbf{w}_{1:}} = \sum_{q=1}^{K} \frac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_q} = \boxed{\frac{\partial \phi_1}{\partial \mathbf{w}_{1:}}} \cdot \frac{\partial H}{\partial \phi_1} + \sum_{q \neq 1} \boxed{\frac{\partial \phi_q}{\partial \mathbf{w}_{1:}}} \cdot \frac{\partial H}{\partial \phi_q}.$

$$\boxed{= \frac{\partial \, \mathbf{x}^T \mathbf{w}_{1:}}{\partial \, \mathbf{w}_{1:}} = \mathbf{x}} \qquad \boxed{= \frac{\partial \, \mathbf{x}^T \mathbf{w}_{q:}}{\partial \, \mathbf{w}_{1:}} = \mathbf{0}}$$

Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$

- $\phi_q = \mathbf{x}^T \mathbf{w}_{q:}$.
- $\frac{\partial H}{\partial \mathbf{w}_{1:}} = \sum_{q=1}^{K} \frac{\partial \phi_q}{\partial \mathbf{w}_{1:}} \cdot \frac{\partial H}{\partial \phi_q} = \boxed{\frac{\partial \phi_1}{\partial \mathbf{w}_{1:}}} \cdot \frac{\partial H}{\partial \phi_1} + \sum_{q \neq 1} \boxed{\frac{\partial \phi_q}{\partial \mathbf{w}_{1:}}} \cdot \frac{\partial H}{\partial \phi_q}.$

$$\boxed{= \frac{\partial\, \mathbf{x}^T \mathbf{w}_{1:}}{\partial \mathbf{w}_{1:}} = \mathbf{x}} \qquad \boxed{= \frac{\partial\, \mathbf{x}^T \mathbf{w}_{q:}}{\partial \mathbf{w}_{1:}} = \mathbf{0}}$$

- $\frac{\partial H}{\partial \mathbf{w}_{1:}} = \mathbf{x} \cdot \frac{\partial H}{\partial \phi_1}.$

Gradient

$$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K \qquad \text{and} \qquad H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$$

- $\frac{\partial H}{\partial \mathbf{w}_{q:}} = \mathbf{x} \cdot \frac{\partial H}{\partial \phi_q}.$

Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right).$

- $\dfrac{\partial\, H}{\partial\, \mathbf{w}_{q:}} = \mathbf{x} \cdot \dfrac{\partial\, H}{\partial\, \phi_q}.$
- $\dfrac{\partial\, H}{\partial\, \phi_q} = -y_q + \dfrac{e^{\phi_q}}{\sum_{j=1}^{K} e^{\phi_j}} = -y_q + p_q.$

Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right)$.

- $\frac{\partial H}{\partial \mathbf{w}_{q:}} = \mathbf{x} \cdot \frac{\partial H}{\partial \phi_q}$.
- $\frac{\partial H}{\partial \phi_q} = -y_q + \frac{e^{\phi_q}}{\sum_{j=1}^{K} e^{\phi_j}} = -y_q + p_q$.
- $\rightarrow$ $\frac{\partial H}{\partial \mathbf{w}_{q:}} = \mathbf{x} \cdot \left(-y_q + p_q\right)$

Gradient

$\boldsymbol{\phi} = \mathbf{W}\mathbf{x} \in \mathbb{R}^K$ and $H(\mathbf{y}, \mathbf{p}) = -\sum_{j=1}^{K} y_j \phi_j + \log\left(\sum_{j=1}^{K} e^{\phi_j}\right)$.

- $\frac{\partial H}{\partial \mathbf{w}_{q:}} = \left(p_q - y_q\right) \cdot \mathbf{x}$.
- $\frac{\partial H}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial H}{\partial \mathbf{w}_{1:}^T} \\ \frac{\partial H}{\partial \mathbf{w}_{2:}^T} \\ \vdots \\ \frac{\partial H}{\partial \mathbf{w}_{K:}^T} \end{bmatrix} = \begin{bmatrix} (p_1 - y_1) \cdot \mathbf{x}^T \\ (p_2 - y_2) \cdot \mathbf{x}^T \\ \vdots \\ (p_K - y_K) \cdot \mathbf{x}^T \end{bmatrix} = (\mathbf{p} - \mathbf{y}) \cdot \mathbf{x}^T$.

Stochastic Gradient Descent

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

- Model: $\min_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)$

  $\boldsymbol{\phi}_i = \mathbf{W}\mathbf{x}_i \in \mathbb{R}^K$, $\quad \mathbf{p}_i = \mathrm{SoftMax}(\boldsymbol{\phi}_i)$, and $\quad H(\mathbf{y}_i, \mathbf{p}_i) = -\sum_{k=1}^{K} y_{i,k} \log(p_{i,k})$.

- A (stochastic) gradient: $\quad \mathbf{G}_i = \frac{\partial H(\mathbf{y}_i, \mathbf{p}_i)}{\partial \mathbf{W}} = (\mathbf{p}_i - \mathbf{y}_i) \cdot \mathbf{x}_i^T \in \mathbb{R}^{K \times d}$.

Stochastic Gradient Descent

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$.

- Model: $\min_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)$

  $\boldsymbol{\phi}_i = \mathbf{W}\mathbf{x}_i \in \mathbb{R}^K$, $\quad \mathbf{p}_i = \mathrm{SoftMax}(\boldsymbol{\phi}_i)$, and $\quad H(\mathbf{y}_i, \mathbf{p}_i) = -\sum_{k=1}^{K} y_{i,k} \log\left(p_{i,k}\right)$.

- A (stochastic) gradient: $\mathbf{G}_i = \frac{\partial H(\mathbf{y}_i, \mathbf{p}_i)}{\partial \mathbf{W}} = (\mathbf{p}_i - \mathbf{y}_i) \cdot \mathbf{x}_i^T \in \mathbb{R}^{K \times d}$.

SGD Algorithm:.

1. Randomly sample i from $\{1, 2, \cdots, n\}$.
2. Compute $\mathbf{G}_i$ using $(\mathbf{x}_i, y_i)$.
3. $\mathbf{W} \leftarrow \mathbf{W} - \alpha\, \mathbf{G}_i$.

Softmax Classifier: Train and Test

- Train (given feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$)
  - Compute $\mathbf{W}^\star = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)$ by SGD or other algorithms.
- Test (for a sample $\mathbf{x}' \in \mathbb{R}^d$)
  - $\boldsymbol{\phi}' = \mathbf{W}^\star \mathbf{x}' \in \mathbb{R}^K$.
  - Return the index of the largest entry of $\boldsymbol{\phi}'$.

Softmax Classifier: Train and Test

- Train (given feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$)
  - Compute $\mathbf{W}^\star = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)$ by SGD or other algorithms.
- Test (for a sample $\mathbf{x}' \in \mathbb{R}^d$)
  - $\boldsymbol{\phi}' = \mathbf{W}^\star \mathbf{x}' \in \mathbb{R}^K$.
  - Return the index of the largest entry of $\boldsymbol{\phi}'$.

What is the predicted class?

$$\boldsymbol{\phi}' = \begin{bmatrix} 0.9 \\ -5 \\ 10 \\ -20 \\ 0 \end{bmatrix}$$

Softmax Classifier: Train and Test

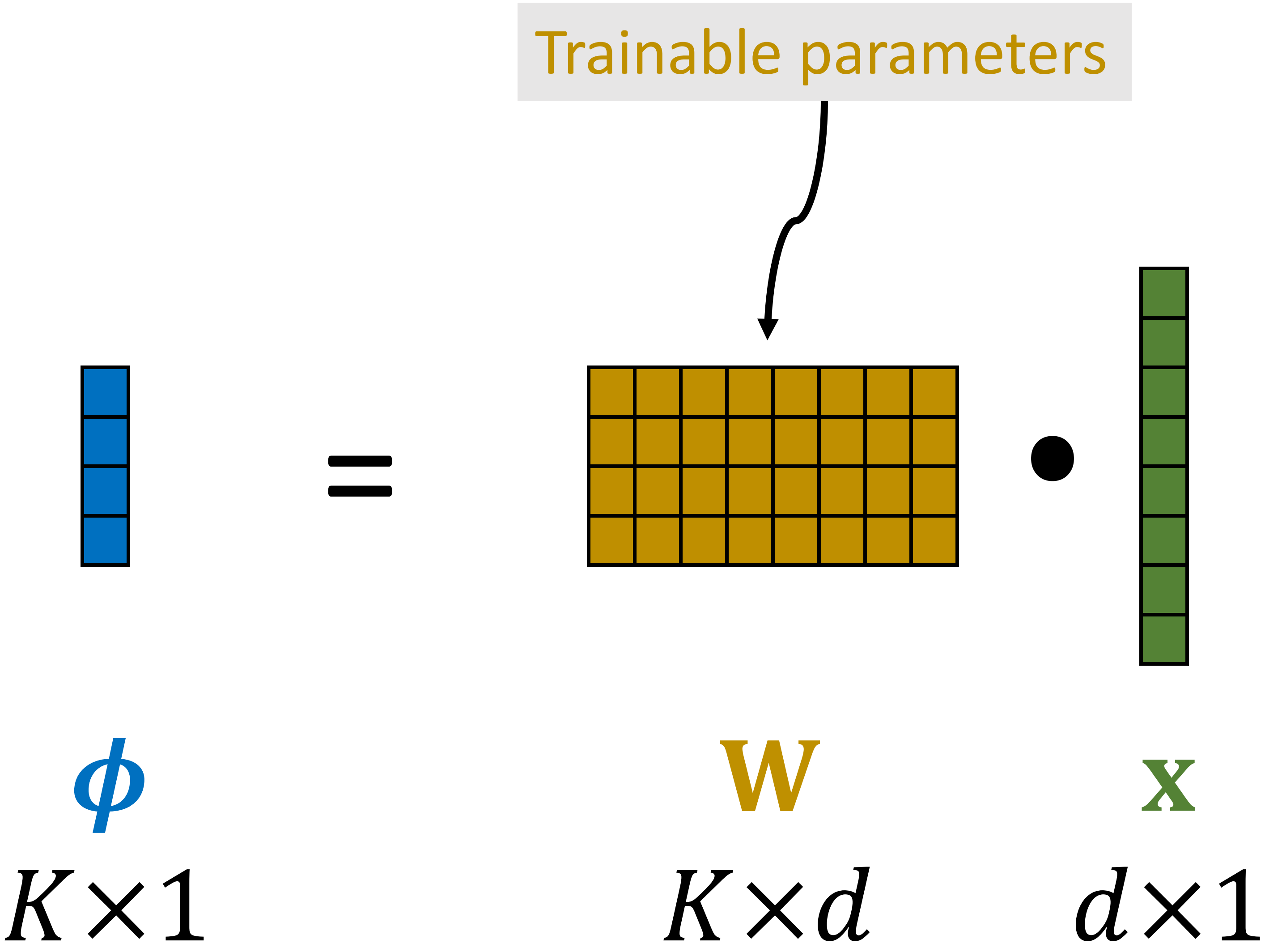
- Train (given feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $\mathbf{y}_1, \cdots, \mathbf{y}_n \in \mathbb{R}^K$)
  - Compute $\mathbf{W}^\star = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{n} H(\mathbf{y}_i, \mathbf{p}_i)$ by SGD or other algorithms.
- Test (for a sample $\mathbf{x}' \in \mathbb{R}^d$)
  - $\boldsymbol{\phi}' = \mathbf{W}^\star \mathbf{x}' \in \mathbb{R}^K$.
  - Return the index of the largest entry of $\boldsymbol{\phi}'$.

What is the predicted class?

$$\boldsymbol{\phi}' = \begin{bmatrix} 0.9 \\ -5 \\ \boxed{10} \\ -20 \\ 0 \end{bmatrix}$$

Limitations of Softmax Classifier

#Parameter v.s. #Classes



#Parameter v.s. #Classes

- Suppose $\#\text{features} = 1K$.
- Suppose $\#\text{classes} = 10$ (e.g., digit recognition).
  - $1K \times 10 = 10K$ parameters.
- Suppose $\#\text{classes} = 1K$ (e.g., ImageNet image recognition).
  - $1K \times 1K = 1M$ parameters.
- Suppose $\#\text{classes} = 1M$ (e.g., face recognition).
  - $1K \times 1M = 1G$ parameters ➔ Heavy computation and memory costs.

#Parameter v.s. #Classes

- Suppose $\#\text{features} = 1K$.
- Suppose $\#\text{classes} = 10$ (e.g., digit recognition).
  - $1K \times 10 = 10K$ parameters.
- Suppose $\#\text{classes} = 1K$ (e.g., ImageNet image recognition).
  - $1K \times 1K = 1M$ parameters.
- Suppose $\#\text{classes} = 1M$ (e.g., face recognition).
  - $1K \times 1M = 1G$ parameters ➔ Heavy computation and memory costs.
- What if $\#\text{classes} = 1G$? (E.g., face recognition for all the Chinese citizens.)

Nearest Neighbor Methods

K-Nearest Neighbor (KNN)

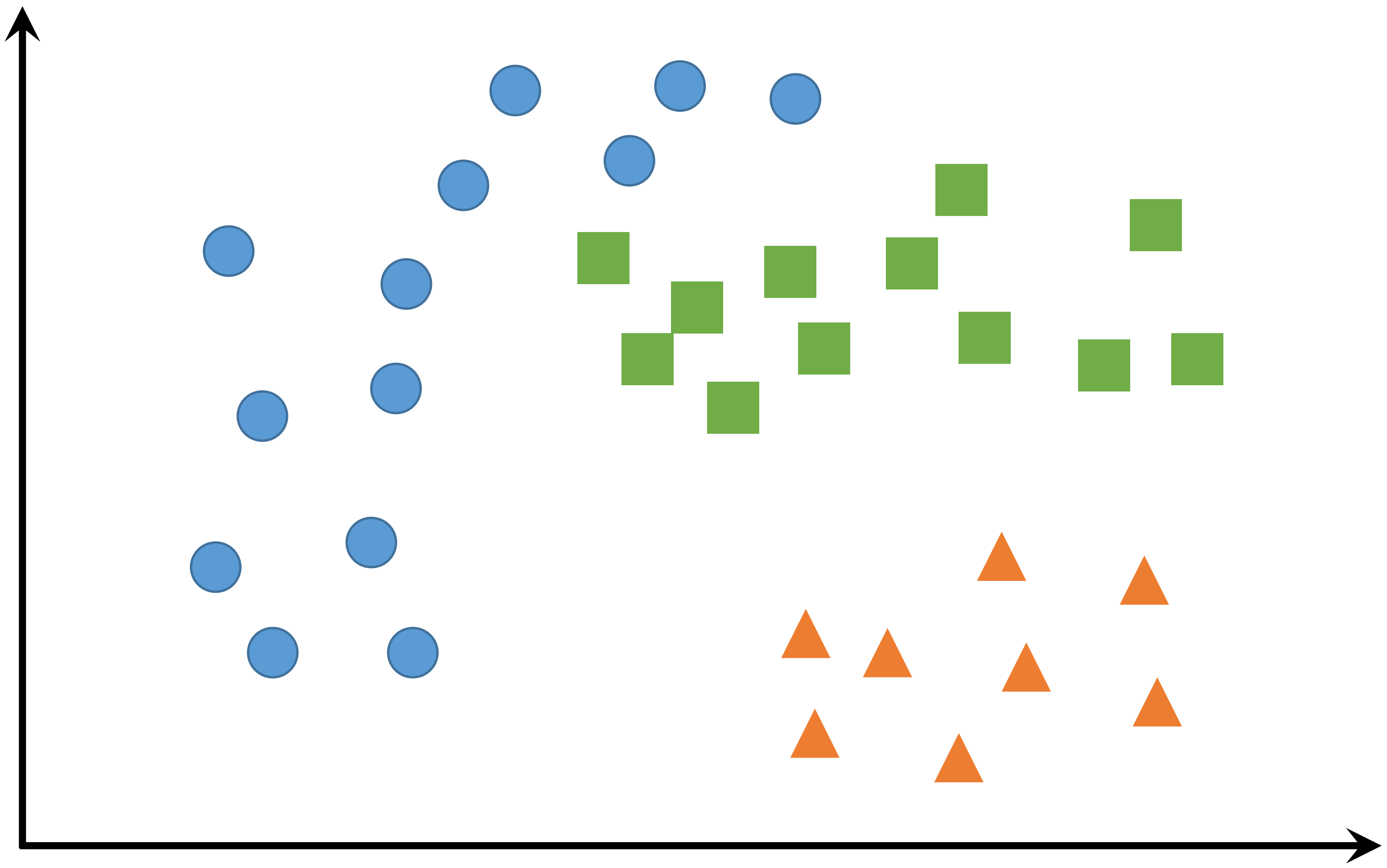
Tasks

Methods

Algorithms

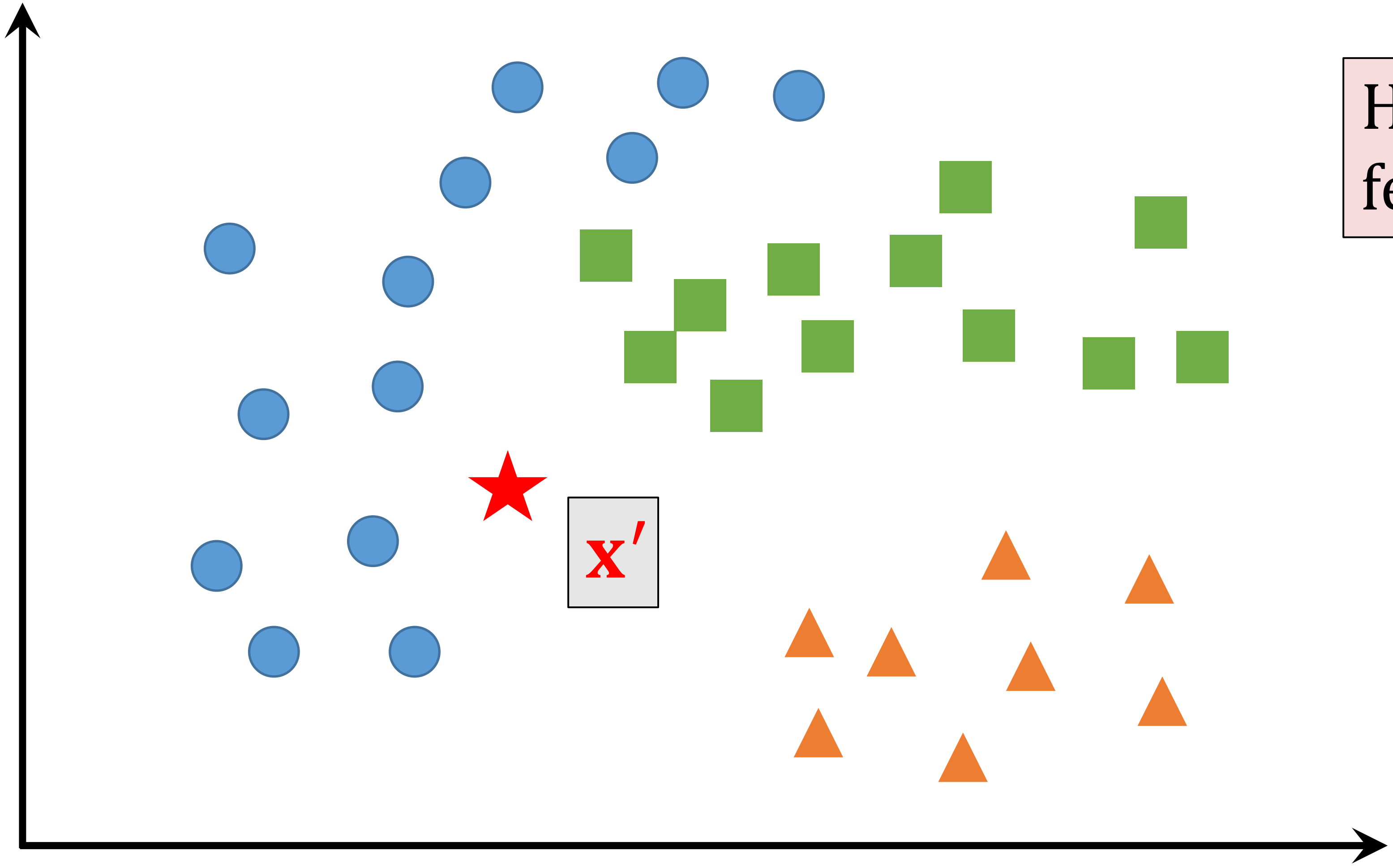
Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.



Nearest Neighbor Classifier

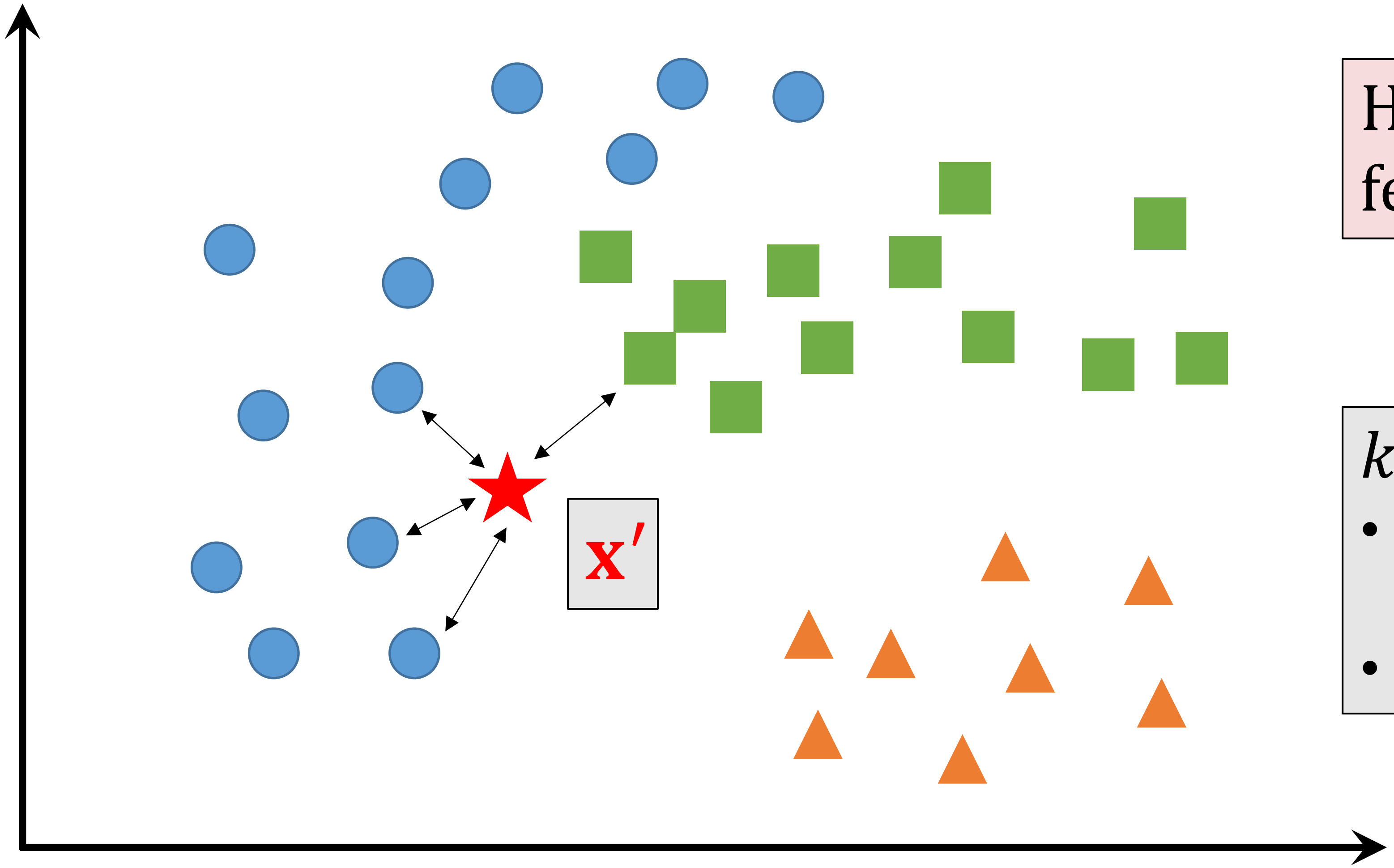
Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.



How to classify an test feature vector $\mathbf{x}'$?

Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.



How to classify an test feature vector $\mathbf{x}'$?

k-Nearest Neighbor (KNN):

- Find the k nearest neighbors (NN) of $\mathbf{x}'$.
- Let the k NNs vote.

Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.

k-**Nearest Neighbor (KNN) classifier**:

- Find the k nearest neighbors of $\mathbf{x}'$.
- Let the NNs vote.

Question: How to set k?

- Treat k as hyper-parameter.
- Tune k using cross-validation.

Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.

k-**Nearest Neighbor (KNN) classifier**:

- Find the k nearest neighbors of $\mathbf{x}'$.
- Let the NNs vote.

Question: How to measure similarity?

- Cosine similarity: $\mathrm{sim}(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\|_2 \|\mathbf{x}'\|_2}$.
- Gaussian kernel: $\mathrm{sim}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{x}'\|_2^2\right)$.
- Laplacian kernel: $\mathrm{sim}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{\sigma} \|\mathbf{x} - \mathbf{x}'\|_1\right)$.

Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.

k-**Nearest Neighbor (KNN) classifier**:
- Find the k nearest neighbors of $\mathbf{x}'$.
- Let the NNs vote.

Question: How to find the k nearest neighbors?

- Naïve algorithm
  - compute all the similarities $\mathrm{sim}(\mathbf{x}_1, \mathbf{x}'), \cdots, \mathrm{sim}(\mathbf{x}_n, \mathbf{x}')$
  - Sort the scores and find the top k.
  - $O(nd)$ time complexity (n: #samples, d: # features).
- Efficient algorithms (to be discussed later).

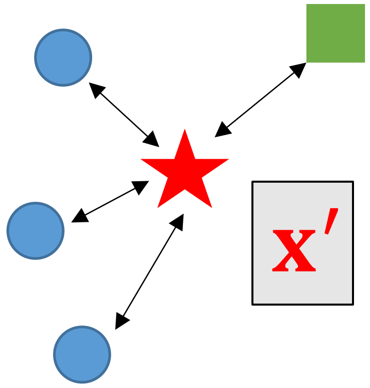
Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.

k-**Nearest Neighbor (KNN) classifier**:
- Find the k nearest neighbors of $\mathbf{x}'$.
- Let the NNs vote.

Question: How to vote?

- Option 1: Every neighbor has the same weight.



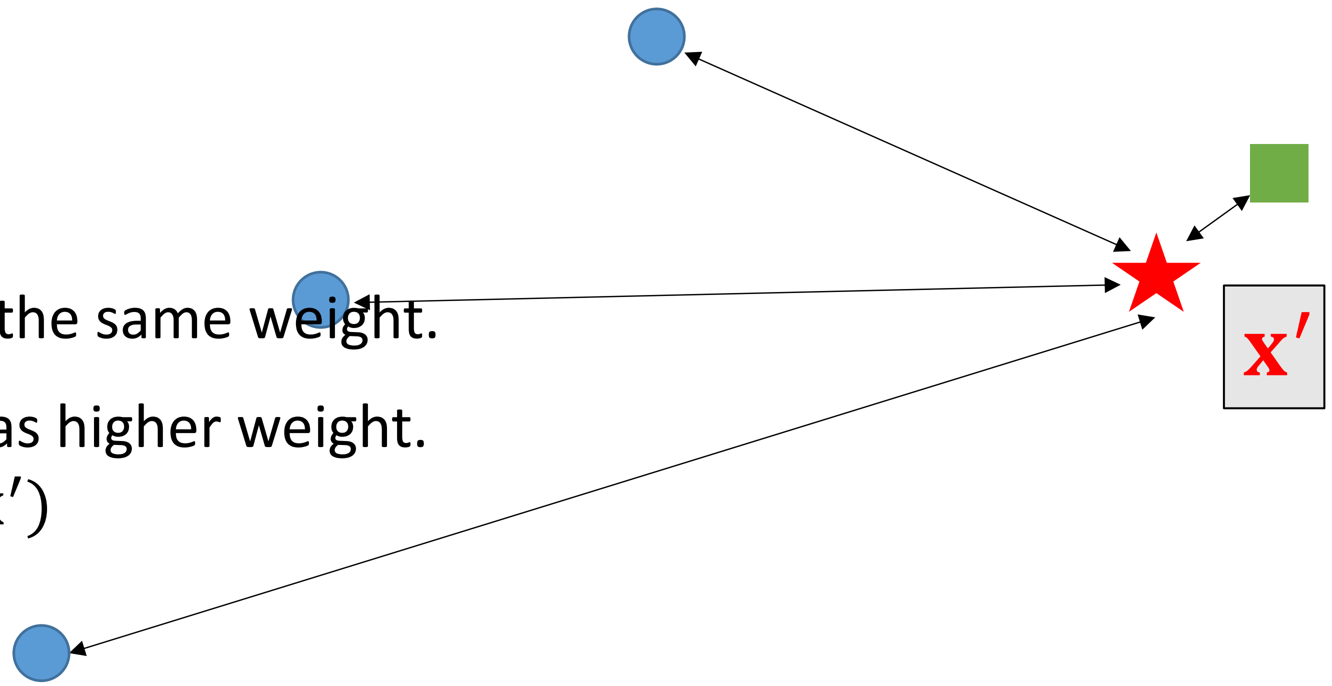
Nearest Neighbor Classifier

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.

k-**Nearest Neighbor (KNN) classifier**:

- Find the k nearest neighbors of $\mathbf{x}'$.
- Let the NNs vote.

Question: How to vote?



- Option 1: Every neighbor has the same weight.
- Option 2: Nearer neighbor has higher weight.
  - E.g., $\text{weight}_i = \text{sim}(\mathbf{x}_i, \mathbf{x}')$

Nearest Neighbor Classifier

Tasks

Methods

Algorithms

KNN: Naïve Algorithm

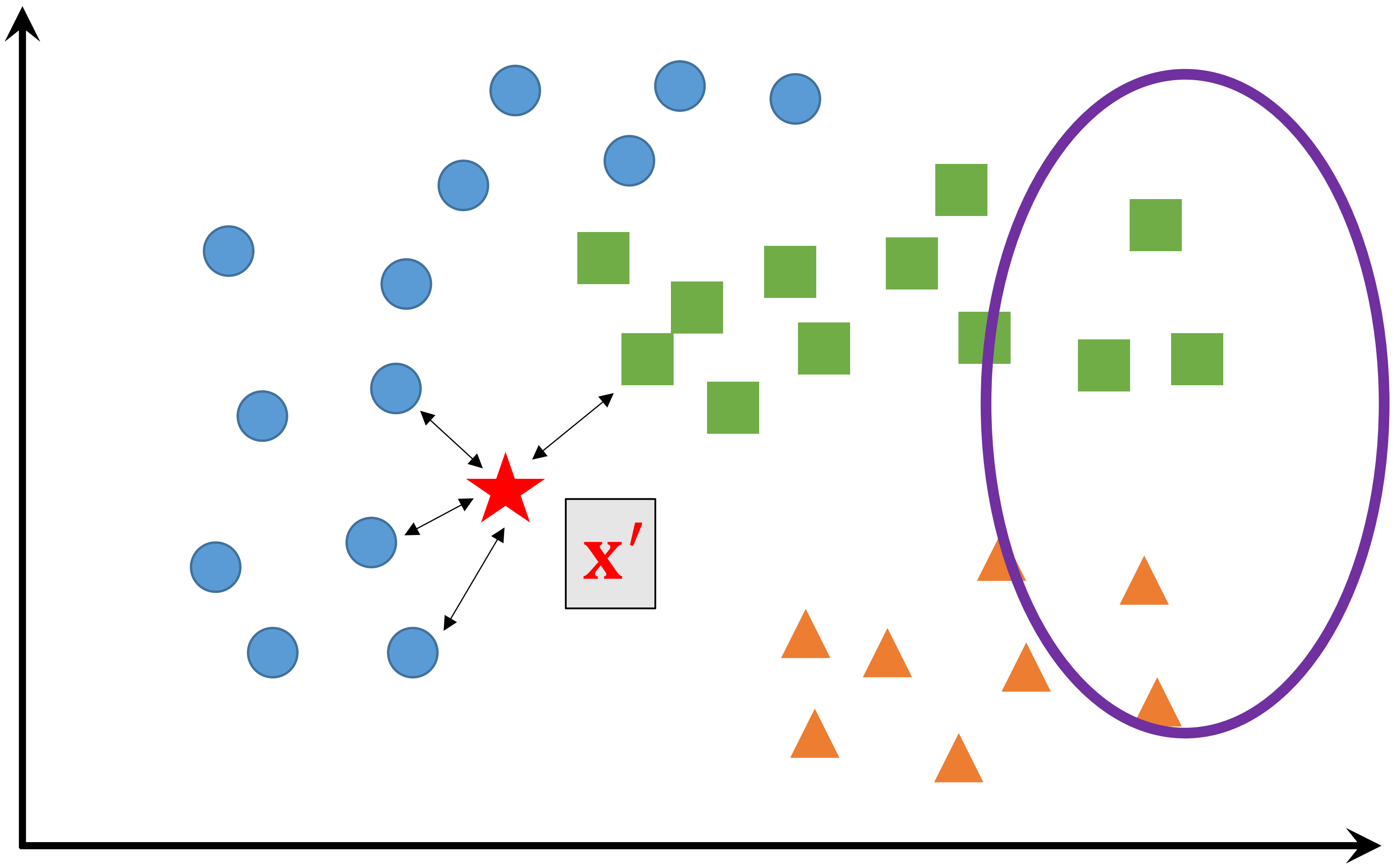
Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.

Algorithm: find the k nearest neighbors to $\mathbf{x}'$.

- Naïve algorithm
  - compute all the similarities $\mathrm{sim}(\mathbf{x}_1, \mathbf{x}'), \cdots, \mathrm{sim}(\mathbf{x}_n, \mathbf{x}')$ and find the top k.
- Training: no training at all.
- Test: for each query, $O(nd)$ time complexity

KNN: Efficient Algorithm

Input: feature vectors $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ and labels $y_1, \cdots, y_n \in \mathbb{N}$.



Does the computer have to look at this region?

Question: find your nearest post office (given longitude & latitude).

Question: find your nearest post office (given longitude & latitude).

Data: $n = 30{,}000$ post offices' latitude and longitude:

- Post office 1: $(\mathrm{lat}_1,\ \mathrm{lon}_1)$
- Post office 2: $(\mathrm{lat}_2,\ \mathrm{lon}_2)$
- Post office 3: $(\mathrm{lat}_3,\ \mathrm{lon}_3)$
- Post office 4: $(\mathrm{lat}_4,\ \mathrm{lon}_4)$

$\vdots$

- Post office n: $(\mathrm{lat}_n,\ \mathrm{lon}_n)$

Query: your own latitude and longitude:

- $(40.74627,\ \ -74.02431)$

Question: find your nearest post office (given longitude & latitude).

Data: $n = 30{,}000$ post offices' latitude and longitude:

- Post office 1: $(\mathrm{lat}_1,\ \mathrm{lon}_1)$
- Post office 2: $(\mathrm{lat}_2,\ \mathrm{lon}_2)$
- Post office 3: $(\mathrm{lat}_3,\ \mathrm{lon}_3)$
- Post office 4: $(\mathrm{lat}_4,\ \mathrm{lon}_4)$

$\vdots$

- Post office n: $(\mathrm{lat}_n,\ \mathrm{lon}_n)$

Query: your own latitude and longitude:

- $(40.74627, \quad -74.02431)$

Question: Which is your nearest post office?

Vector Quantization for KNN

Training:

1. Vector quantization (build landmarks)

Vector Quantization for KNN

Training:

1. Vector quantization (build landmarks)
2. Assign each post office to its nearest landmarks.

Vector Quantization for KNN

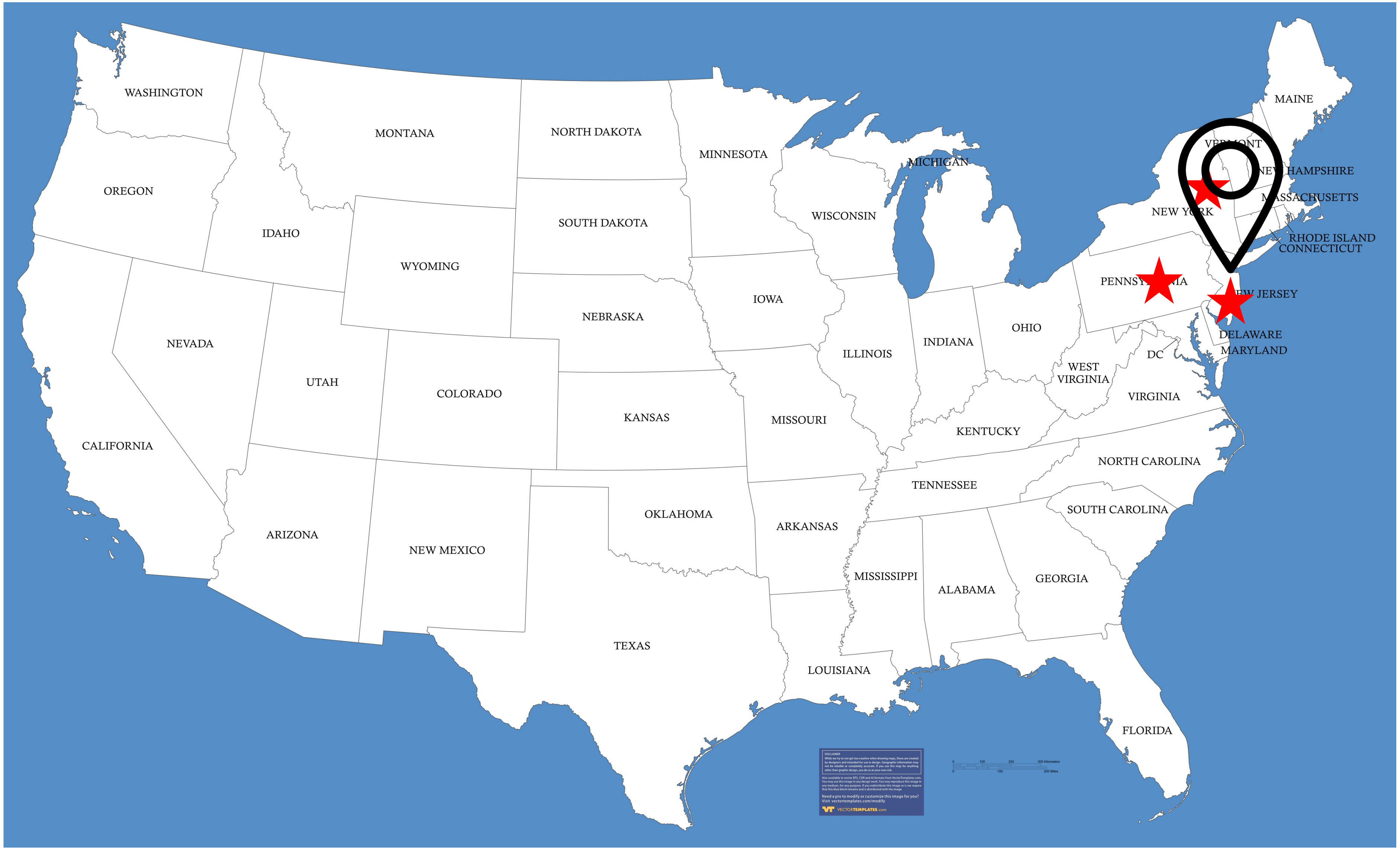
Training:

1. Vector quantization (build landmarks)
2. Assign each post office to its nearest landmarks.

Test

1. Compare your location with all the landmarks and find the nearest landmarks.

Vector Quantization for KNN



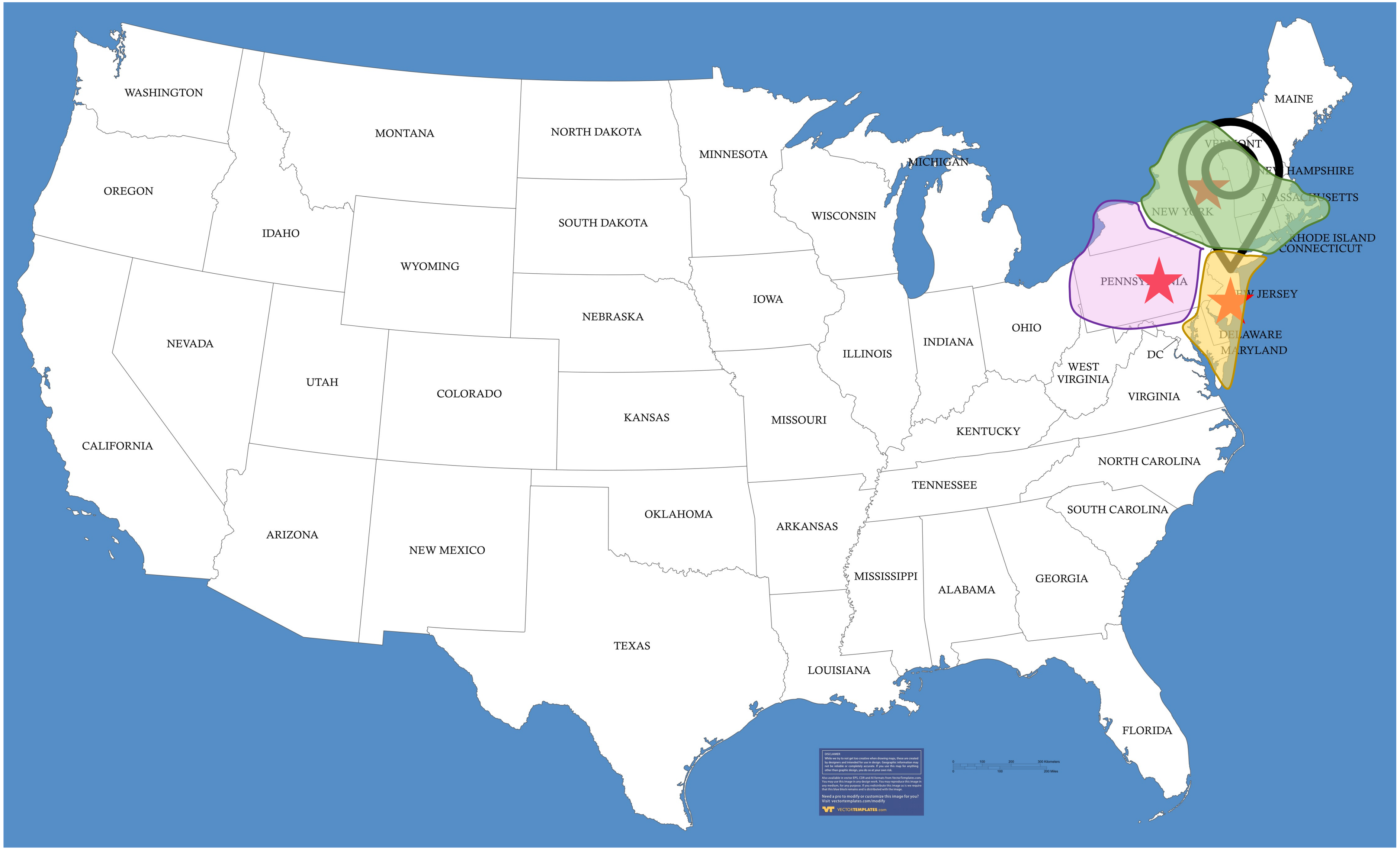
Training:

1. Vector quantization (build landmarks)
2. Assign each post office to its nearest landmarks.

Test

1. Compare your location with all the landmarks and find the nearest landmarks.

Vector Quantization for KNN



Training:

1. Vector quantization (build landmarks)
2. Assign each post office to its nearest landmarks.

Test

1. Compare your location with all the landmarks and find the nearest landmarks.
2. Compare with the postal offices assigned to the landmarks.

KNN: Efficient Algorithms

- Fast algorithms
  - Vector Quantization
  - KD-tree
  - Locality sensitive hashing

- More resources:
  - KNN Search (Wikipedia)

Summary

- KNN method for multi-class classification.
- KNN’s advantage over Softmax classifier:
  - When #class is huge, Softmax classifier is expensive.
  - E.g., in the face recognition problem, #class can be millions.

Summary

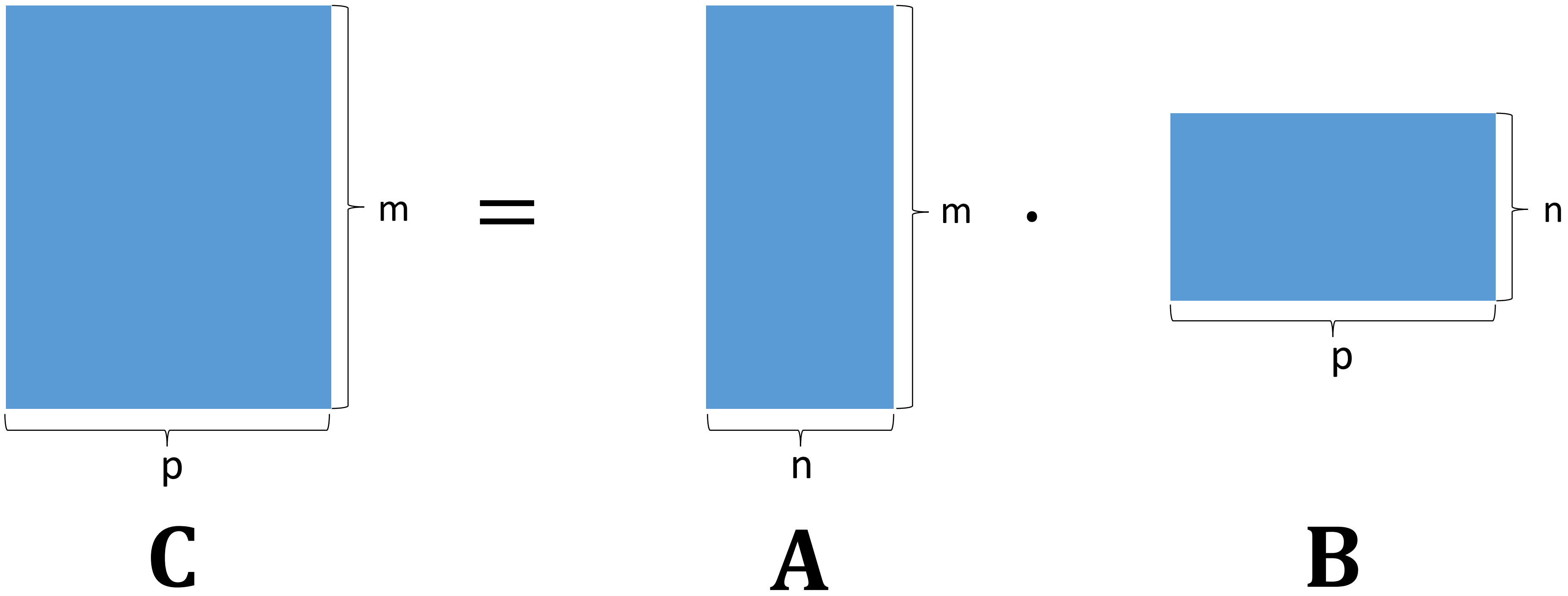
- Training: partition the feature space to regions.

- Prediction (for a test feature vector $\mathbf{x}'$):
  1. Find the nearest regions.
  2. Retrieve all the training feature vectors in the regions.
  3. Compare $\mathbf{x}'$ with the retrieved feature vectors (using similarity score) and return the k nearest.
  4. Weighted/unweighted votes by the k nearest neighbors.

A few words about matrix multiplication

Matrix Multiplication

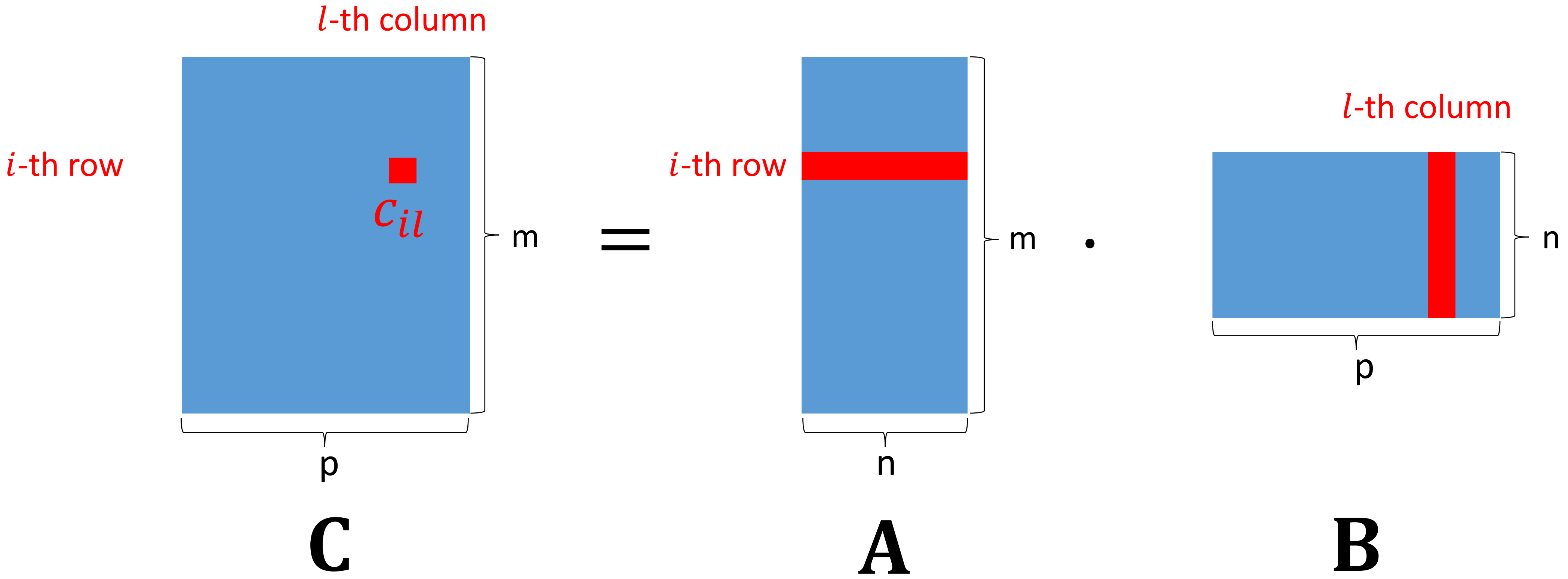
Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.



Matrix Multiplication

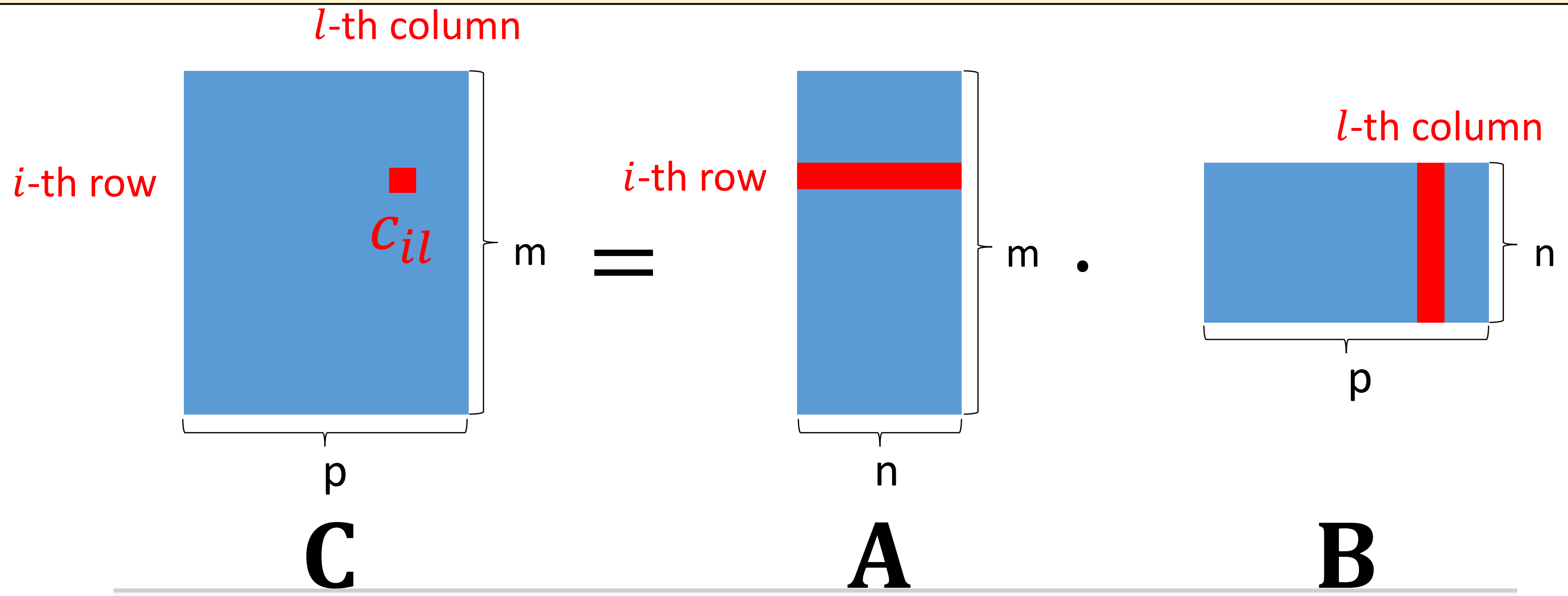
Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

- Suppose you **do not have any** vector or matrix multiplication library.



Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.




```
C = numpy.zeros((m, p))
for i in range(m):
    for j in range(p):
        for l in range(n):
            C[i, j] += A[i, l] * B[l, j]
```

Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

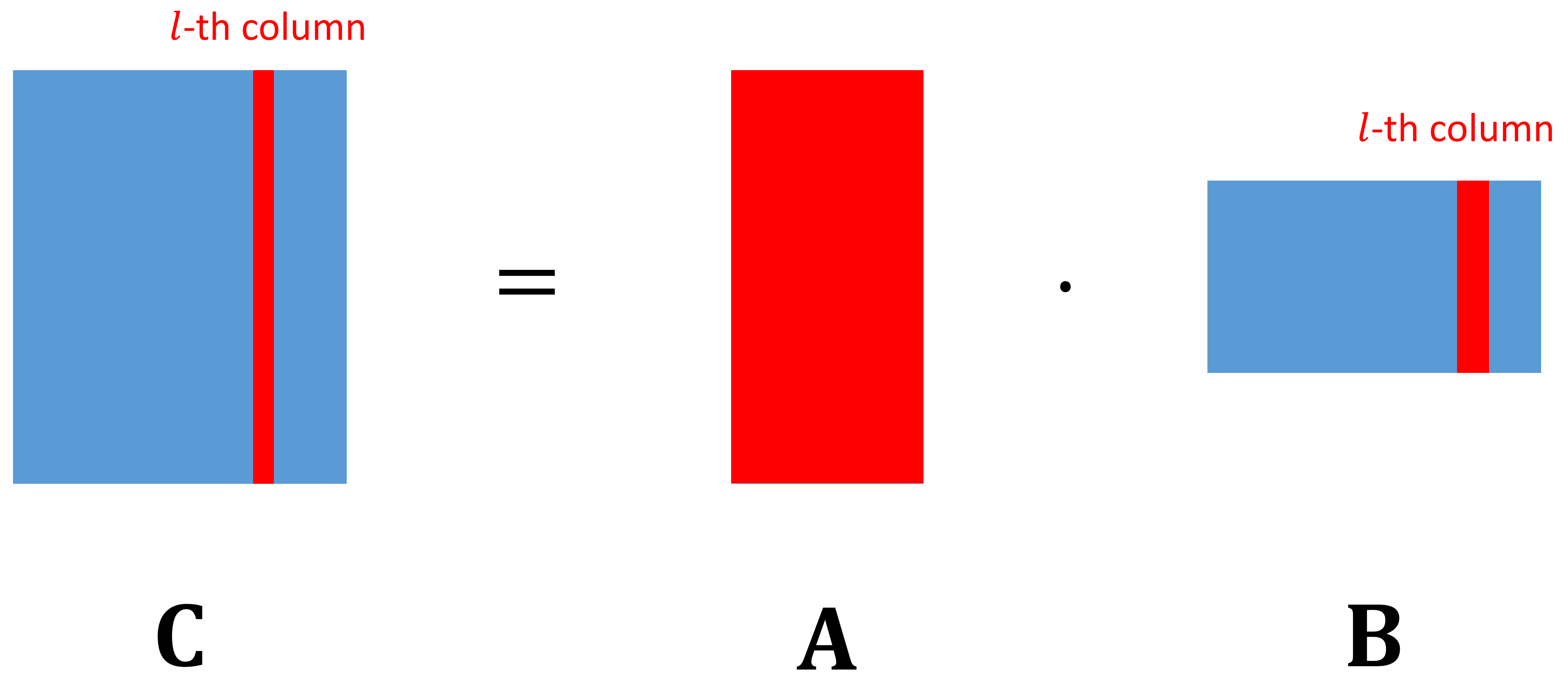
- Suppose you have only **vector-vector multiplication** libraries.

```
C = numpy.zeros((m, p))
for i in range(m):
    for l in range(p):
        C[i, l] = numpy.dot(A[i, :], B[:, l])
```

Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

- Suppose you have **matrix-vector multiplication** libraries.



Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

- Suppose you have **matrix-vector multiplication** libraries.

```
C = numpy.zeros((m, p))
for l in range(p):
    C[:, l] = numpy.dot(A, B[:, l])
```

Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

- Suppose you have **matrix-matrix multiplication** libraries.

```
C = numpy.dot(A, B)
```

Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

- Which is the most efficient?
  - 3-level loop of **scalar multiplication**.
  - 2-level loop of **vector-vector multiplication**.
  - 1-level loop of **matrix-vector multiplication**.
  - Directly use **matrix-matrix multiplication** library.

Matrix Multiplication

Task: Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, compute $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$.

- Which is the most efficient?
  - 3-level loop of **scalar multiplication**.
  - 2-level loop of **vector-vector multiplication**.
  - 1-level loop of **matrix-vector multiplication**.
  - Directly use **matrix-matrix multiplication** library.

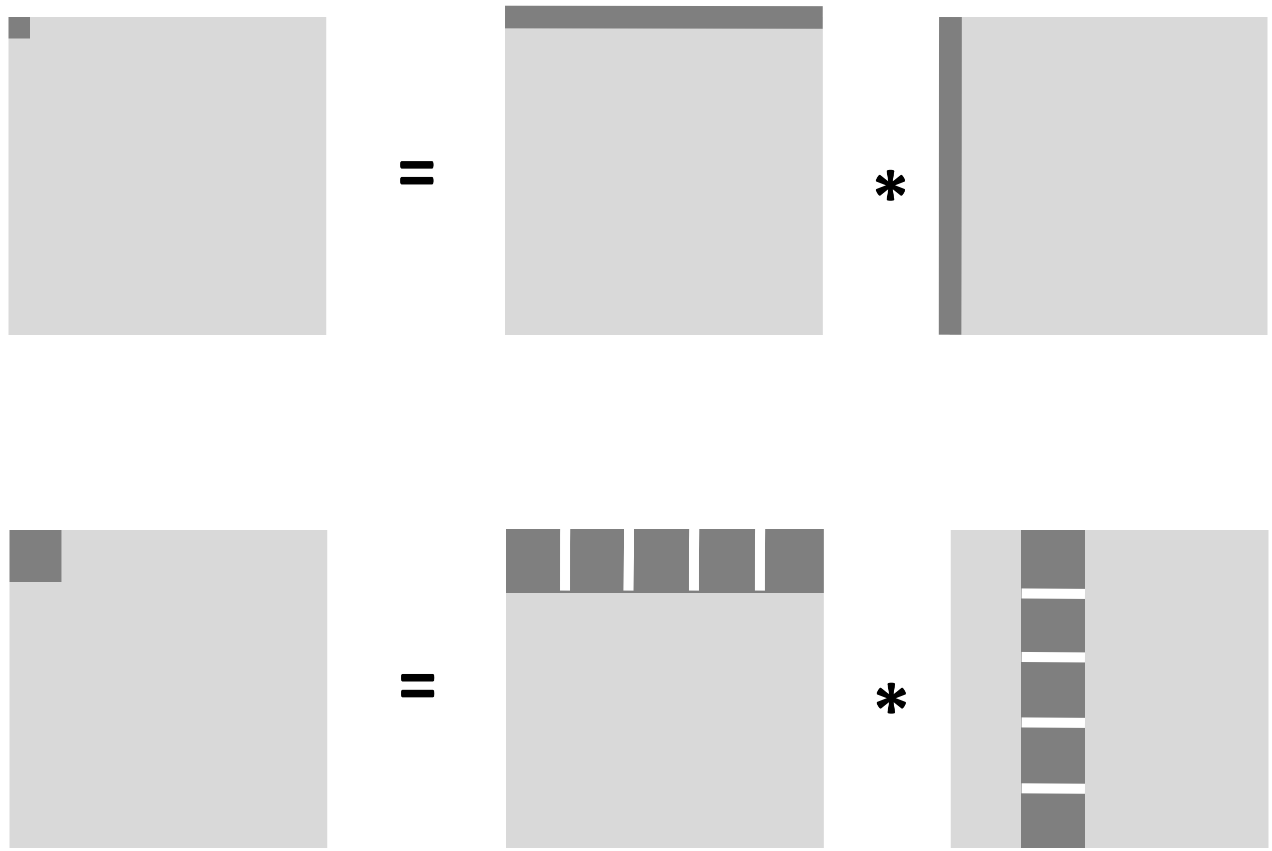
- Is your answer the same if the programming language is C or Fortran?

Basic Linear Algebra Subprograms (BLAS)

- **BLAS**: a library of standard building blocks for performing basic vector and matrix operations
- **Level 1 BLAS** perform scalar, vector, and vector-vector operations.
  - E.g., $\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$, $a \leftarrow \mathbf{x}^T\mathbf{y}$, and $b \leftarrow ||\mathbf{x}||_2$.
- **Level 2 BLAS** perform matrix-vector operations.
  - E.g, $\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$ and $\mathbf{A} \leftarrow \alpha\mathbf{x}\mathbf{y}^T + \mathbf{A}$.
- **Level 3 BLAS** perform matrix-matrix operations.
  - E.g, $\mathbf{A} \leftarrow \mathbf{A}^T$, $\mathbf{C} \leftarrow \mathbf{A}\mathbf{A}^T$, and $\mathbf{C} \leftarrow \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$.

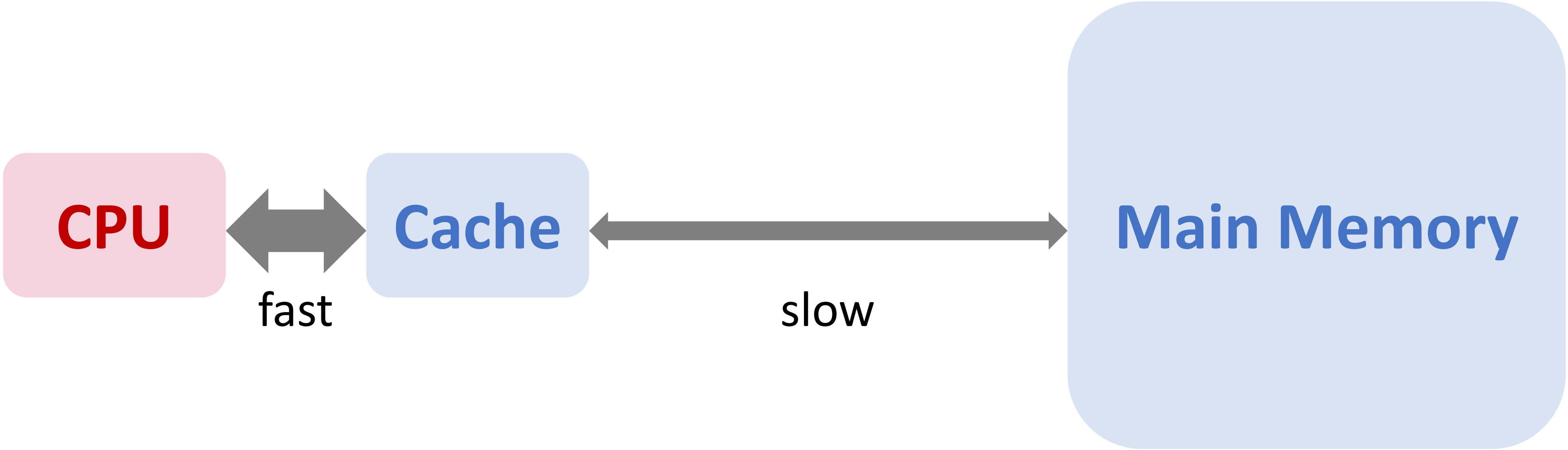
Why are BLAS Fast?

- Efficient algorithms, e.g., blocking, to reduce time complexity.



Why are BLAS Fast?

- Efficient algorithms, e.g., blocking, to reduce time complexity.
- Cache optimization by, e.g., spatial locality.



Why are BLAS Fast?

- Efficient algorithms, e.g., blocking, to reduce time complexity.
- Cache optimization by, e.g., spatial locality.
- Optimized for CPUs/GPUs, e.g.,
  - Intel MKL,
  - NVIDIA cuBLAS

Linear Algebra Package (LAPACK)

- LAPACK provides routines for numerical linear algebra, e.g.,
  - solving least squares $\min_{\mathbf{w}} \left|\left|\mathbf{Xw} - \mathbf{y}\right|\right|_2^2$,
  - eigenvalue problems $\mathbf{A} = \mathbf{U\Lambda U}^T$,
  - SVD $\mathbf{X} = \mathbf{U\Lambda V}^T$,
  - etc.

Linear Algebra Package (LAPACK)

- LAPACK provides routines for numerical linear algebra, e.g.,
  - solving least squares $\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2$,
  - eigenvalue problems $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$,
  - SVD $\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$,
  - etc.
- LAPACK uses Level 3 BLAS as much as possible.

Linear Algebra Package (LAPACK)

- LAPACK provides routines for numerical linear algebra, e.g.,
  - solving least squares $\min_{\mathbf{w}} \left|\left|\mathbf{X}\mathbf{w} - \mathbf{y}\right|\right|_2^2$,
  - eigenvalue problems $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$,
  - SVD $\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$,
  - etc.
- LAPACK uses Level 3 BLAS as much as possible.
- Numpy uses BLAS and LAPACK for matrix computation.
- Deep learning platforms (e.g., Pytorch, Tensorflow) are built upon BLAS.

Thank you