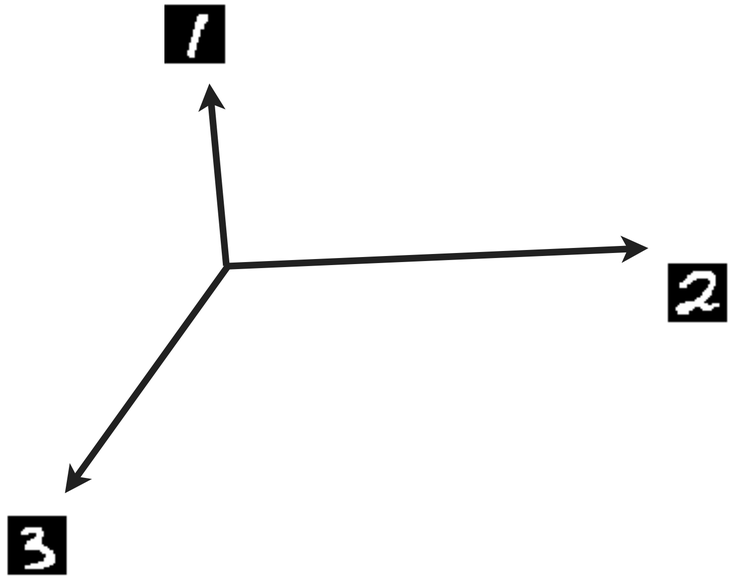
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$

$\mathbf{W}_{m \times d}$

(best agreement w/ prototype)

np.argmax($\mathbf{W} \cdot \mathbf{x}$)





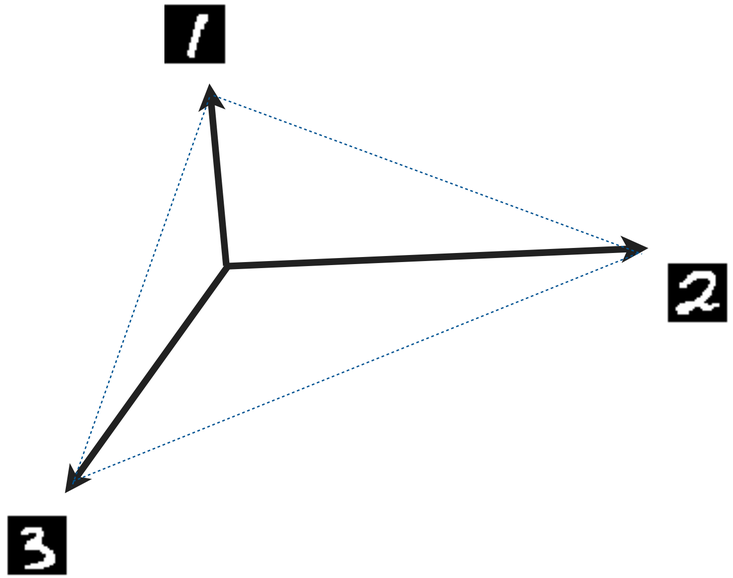
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$

$\mathbf{W}_{m \times d}$

(best agreement w/ prototype)

np.argmax($\mathbf{W} \cdot \mathbf{x}$)





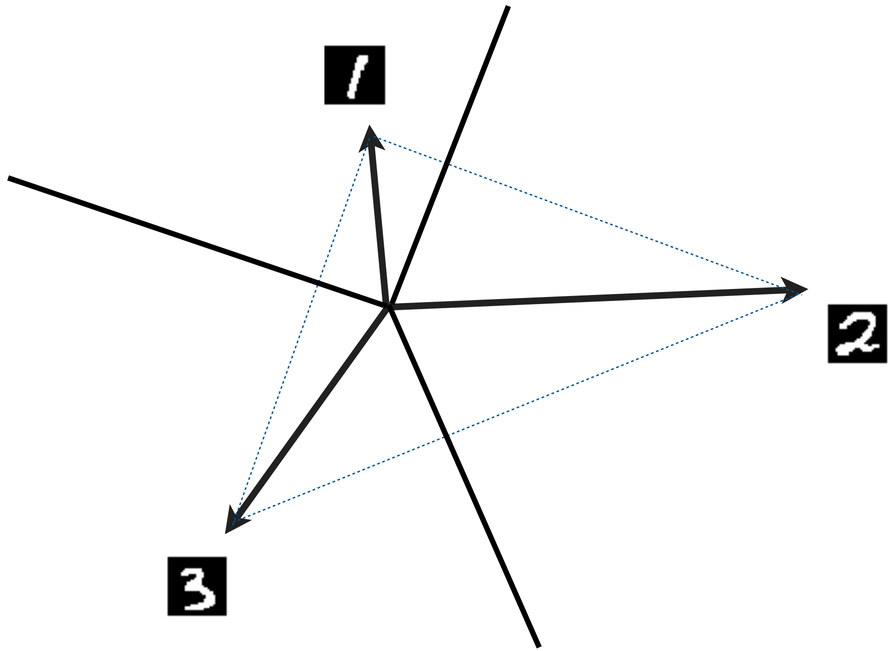
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg \max_{k=1 \ldots m} \mathbf{w}_k \cdot \mathbf{x}$



(best agreement w/ prototype)

np.argmax($\mathbf{W} \cdot \mathbf{x}$)





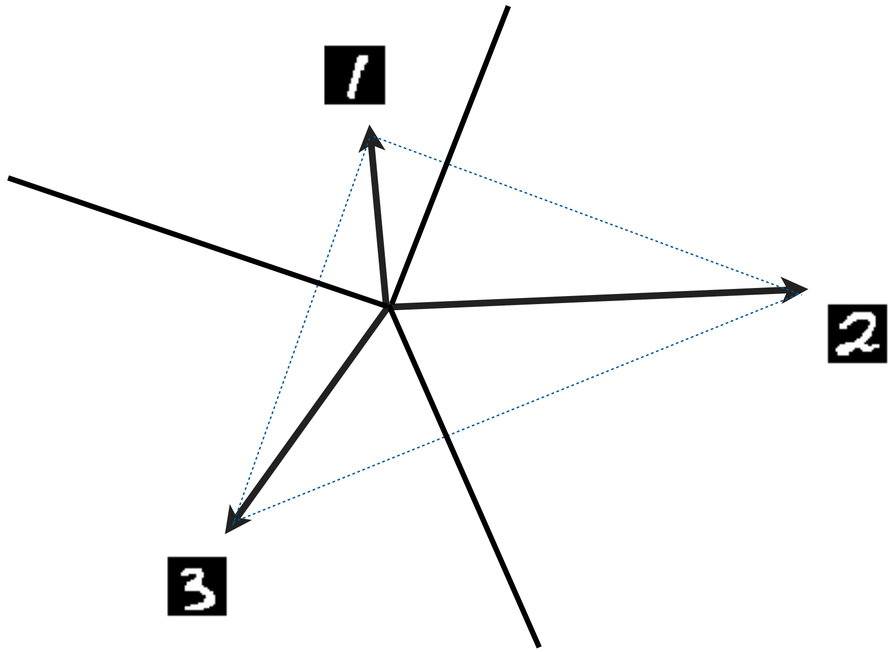
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$

$\mathbf{W}_{m \times d}$

(best agreement w/ prototype)

np.argmax($\mathbf{W} \cdot \mathbf{x}$)



Q1: what about 2-class?



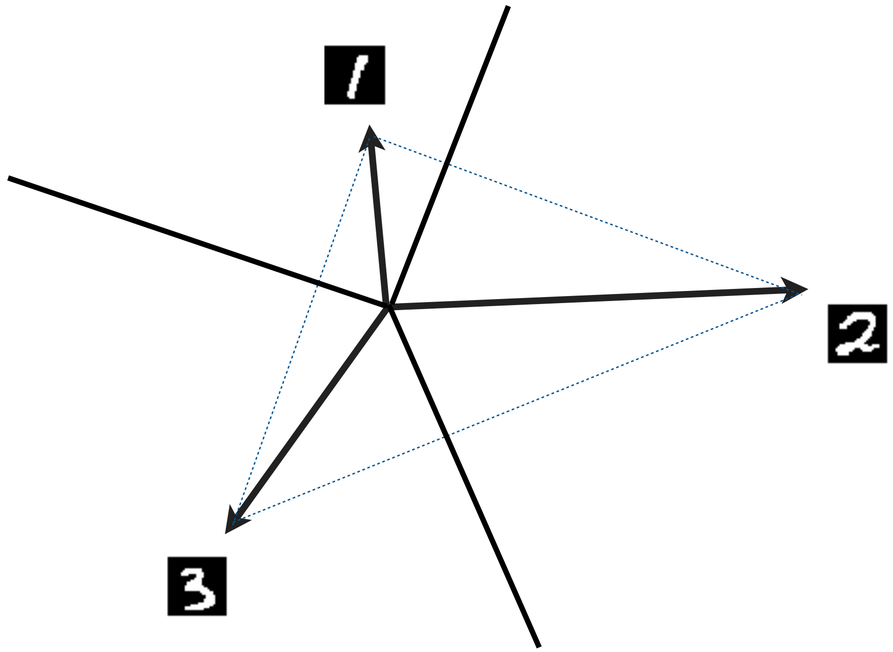
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$

$\mathbf{W}_{m \times d}$

(best agreement w/ prototype)

np.argmax($\mathbf{W} \cdot \mathbf{x}$)



Q1: what about 2-class?

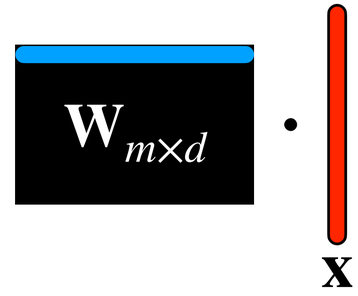
Q2: do we still need augmented space?



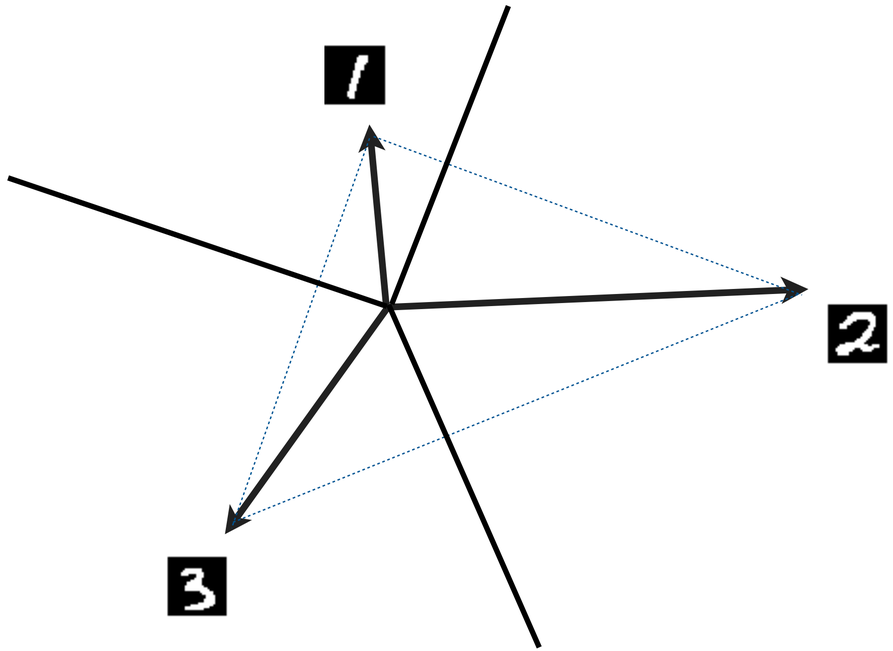
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$

(best agreement w/ prototype)



np.argmax($\mathbf{W} \cdot \mathbf{x}$)



Q1: what about 2-class?

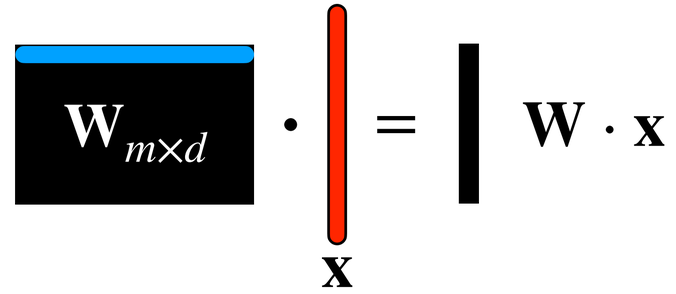
Q2: do we still need augmented space?



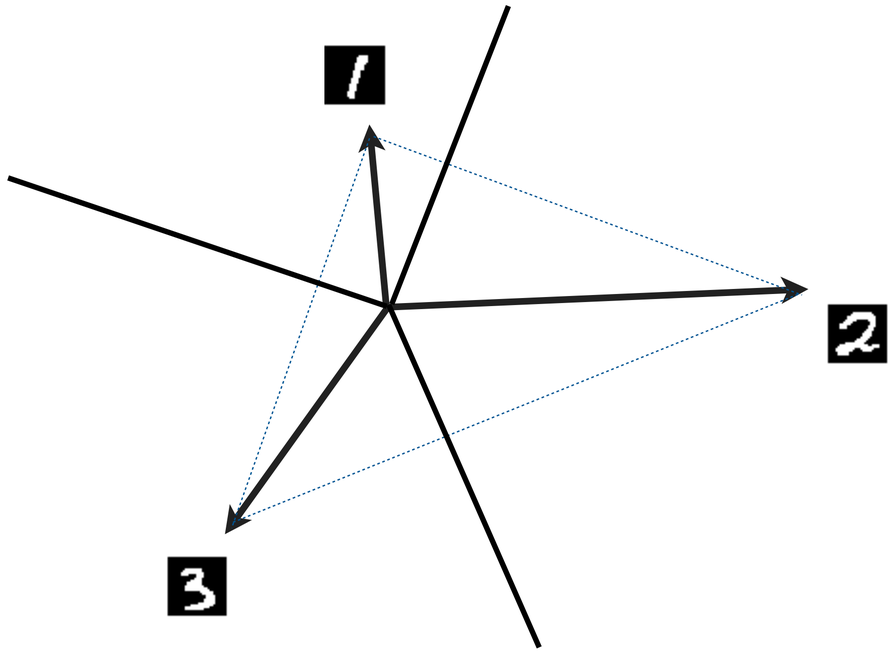
Multiclass Linear Classification

- one weight vector ("prototype") per class (organized as an $m \times d$ matrix): $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_m)$
- multiclass decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$

(best agreement w/ prototype)



np.argmax($\mathbf{W} \cdot \mathbf{x}$)



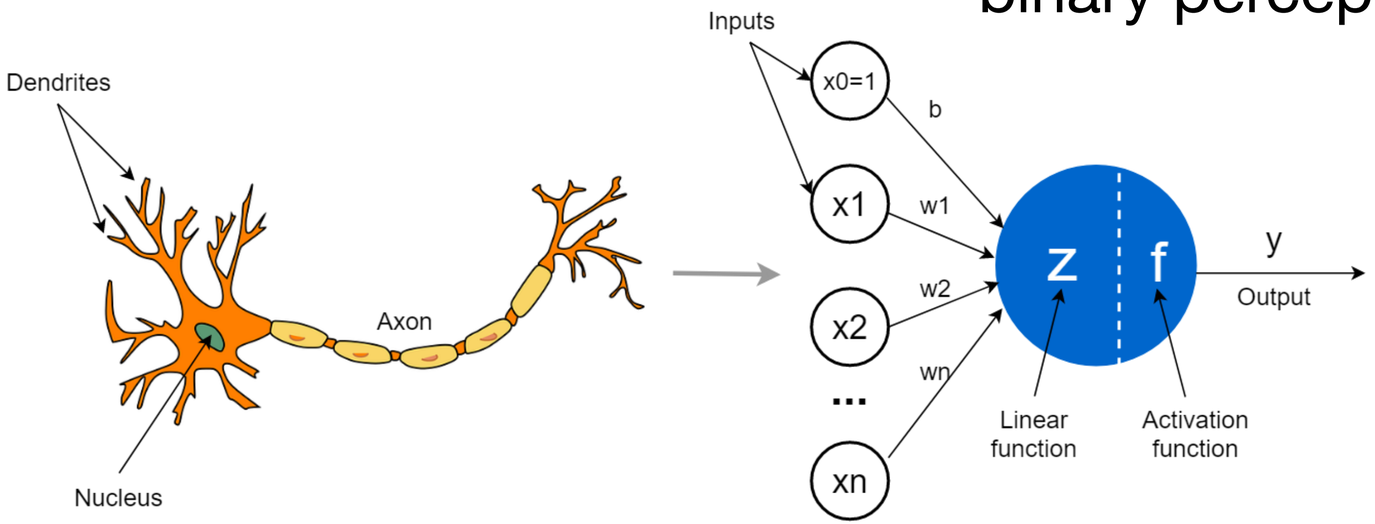
Q1: what about 2-class?

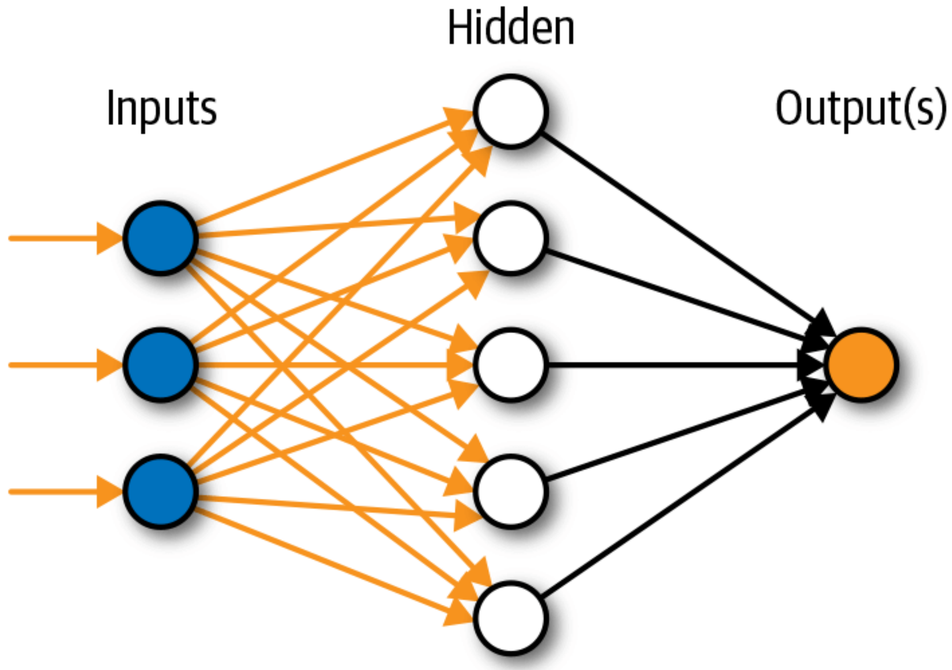
Q2: do we still need augmented space?



Perceptron and Neural Network

binary perceptron (dot product)

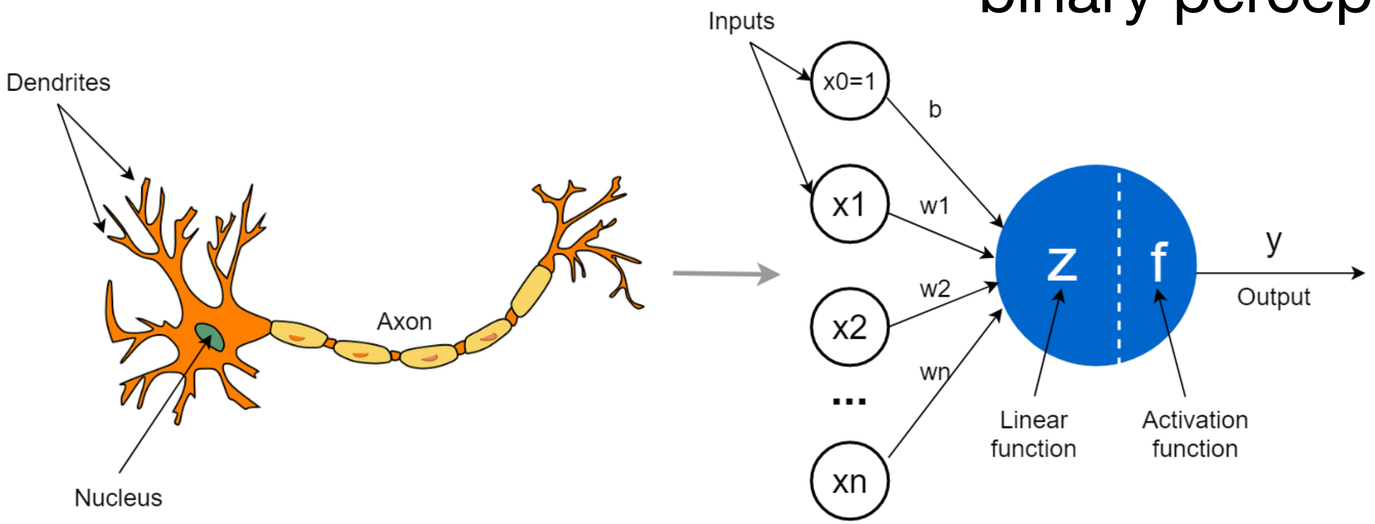


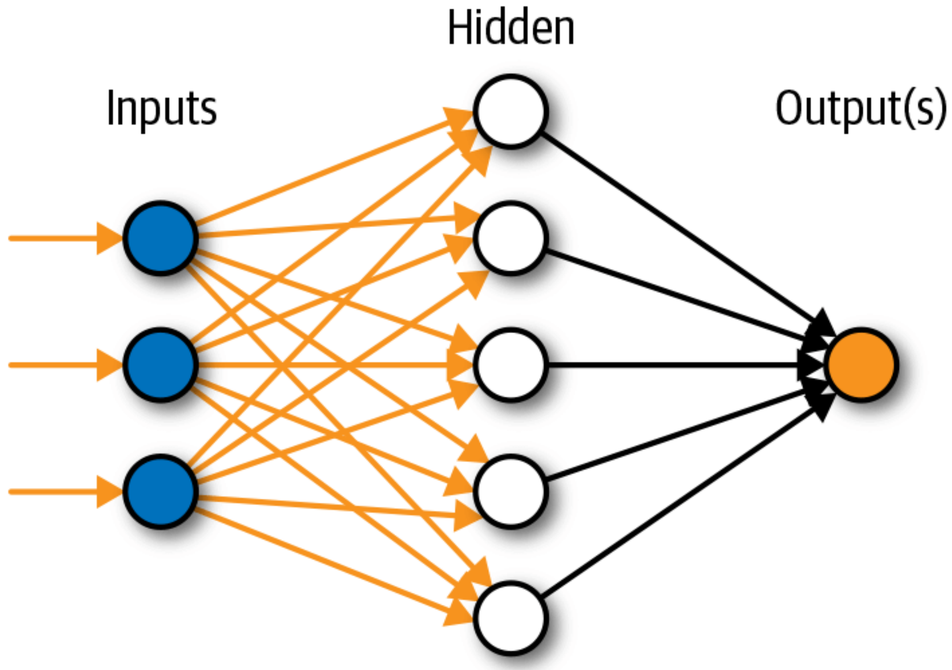


multiclass perceptron (matrix-vector product)

Perceptron and Neural Network

binary perceptron (dot product)

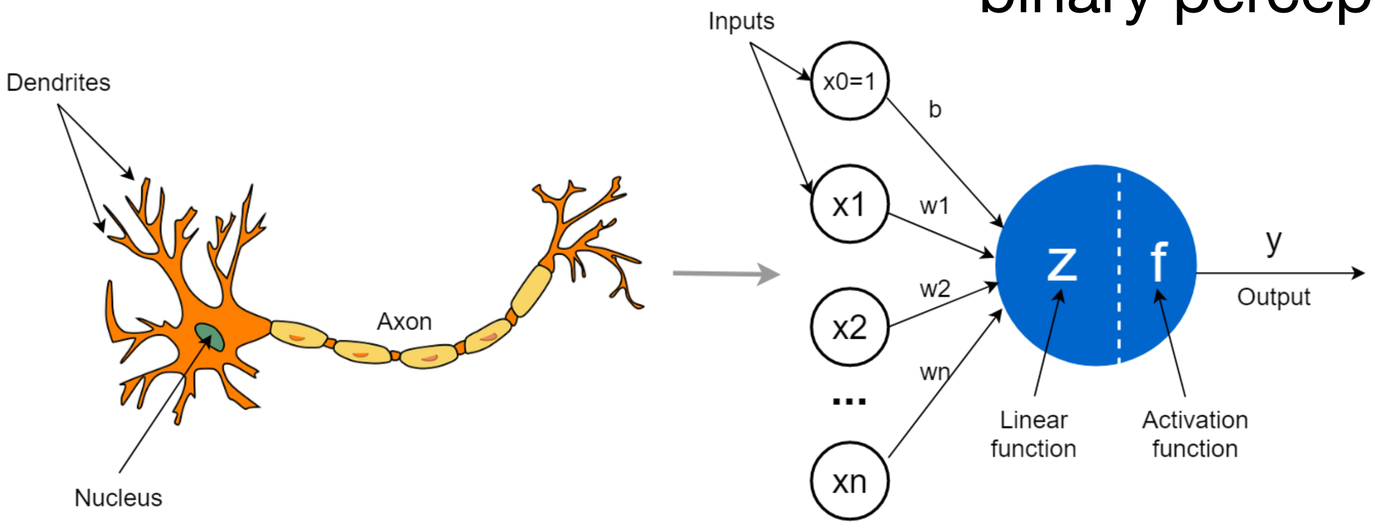


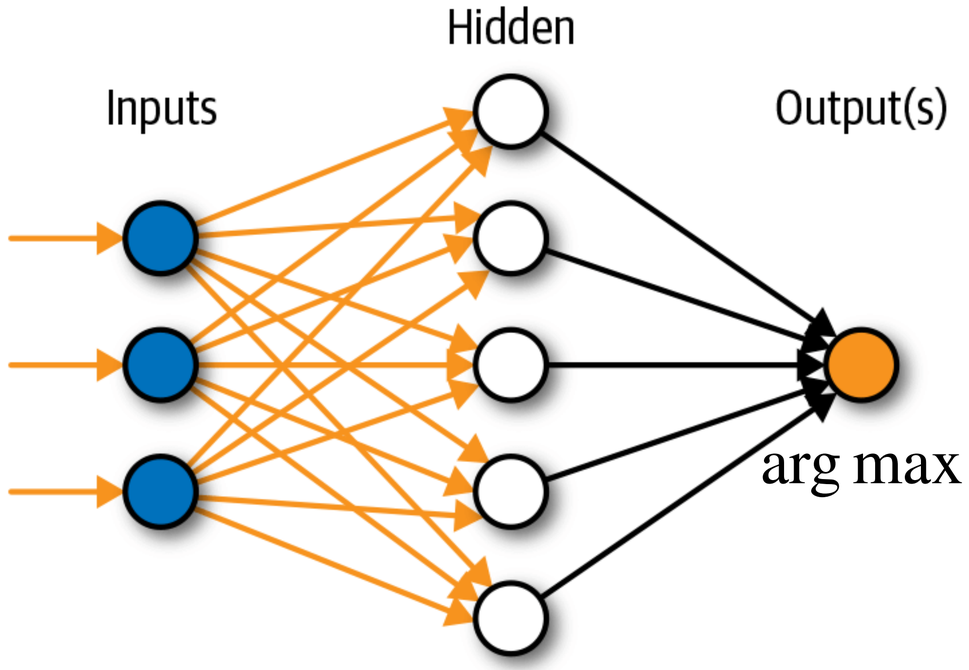


multiclass perceptron (matrix-vector product)

Perceptron and Neural Network

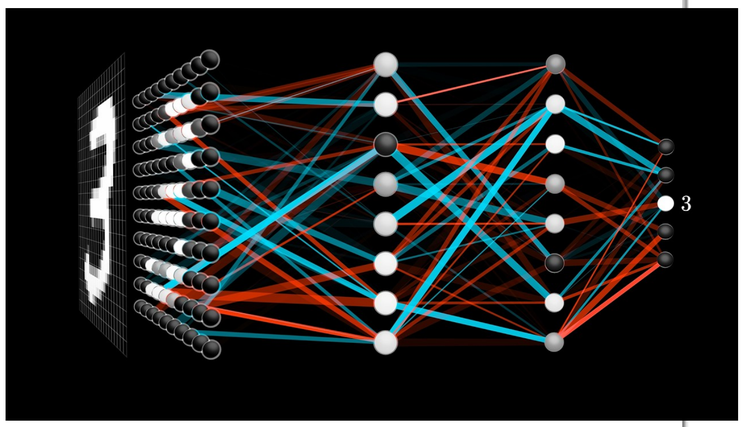
binary perceptron (dot product)

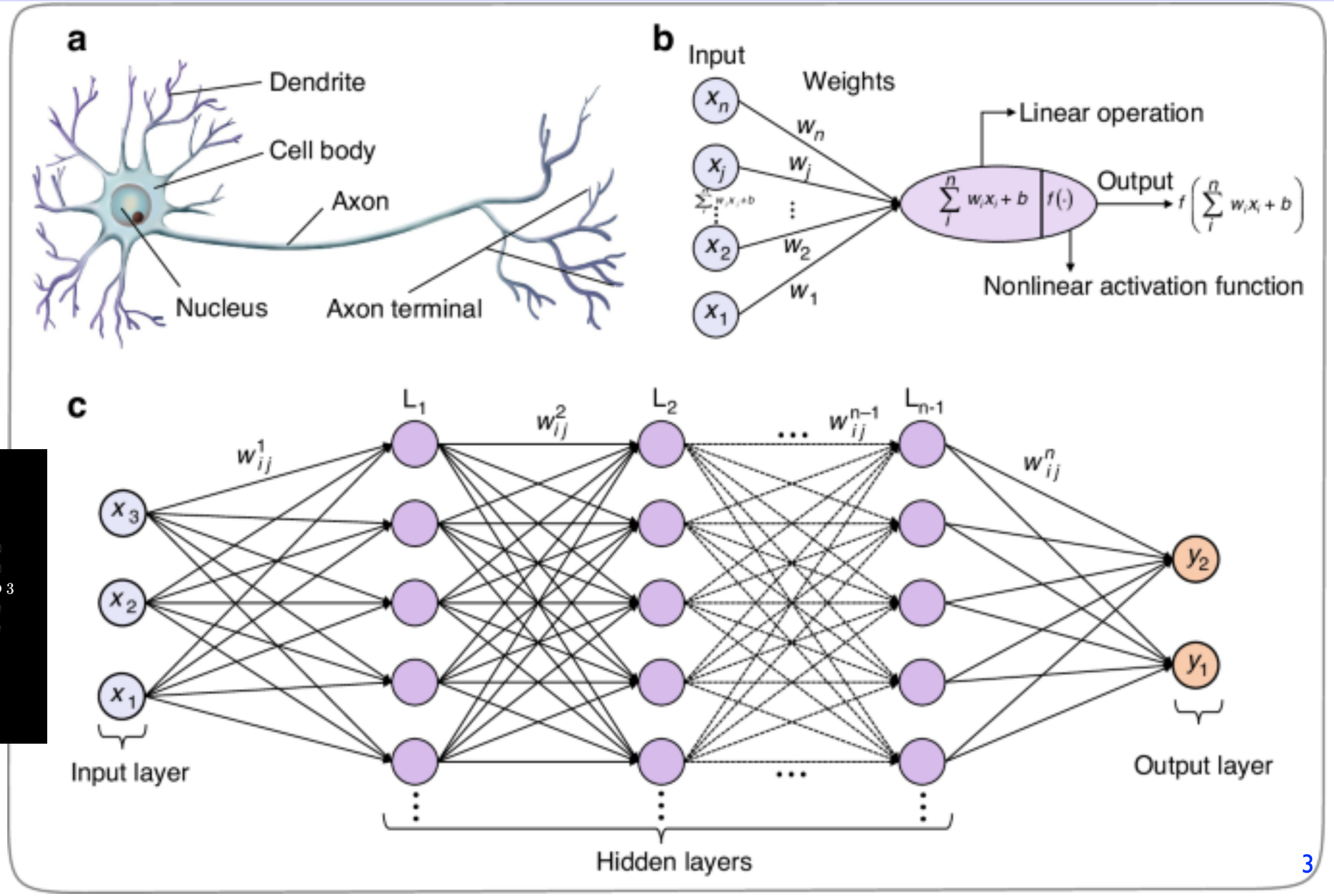




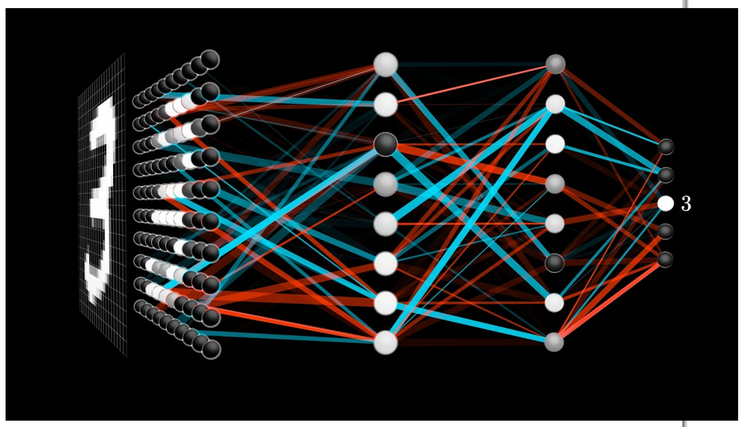
multiclass perceptron (matrix-vector product)

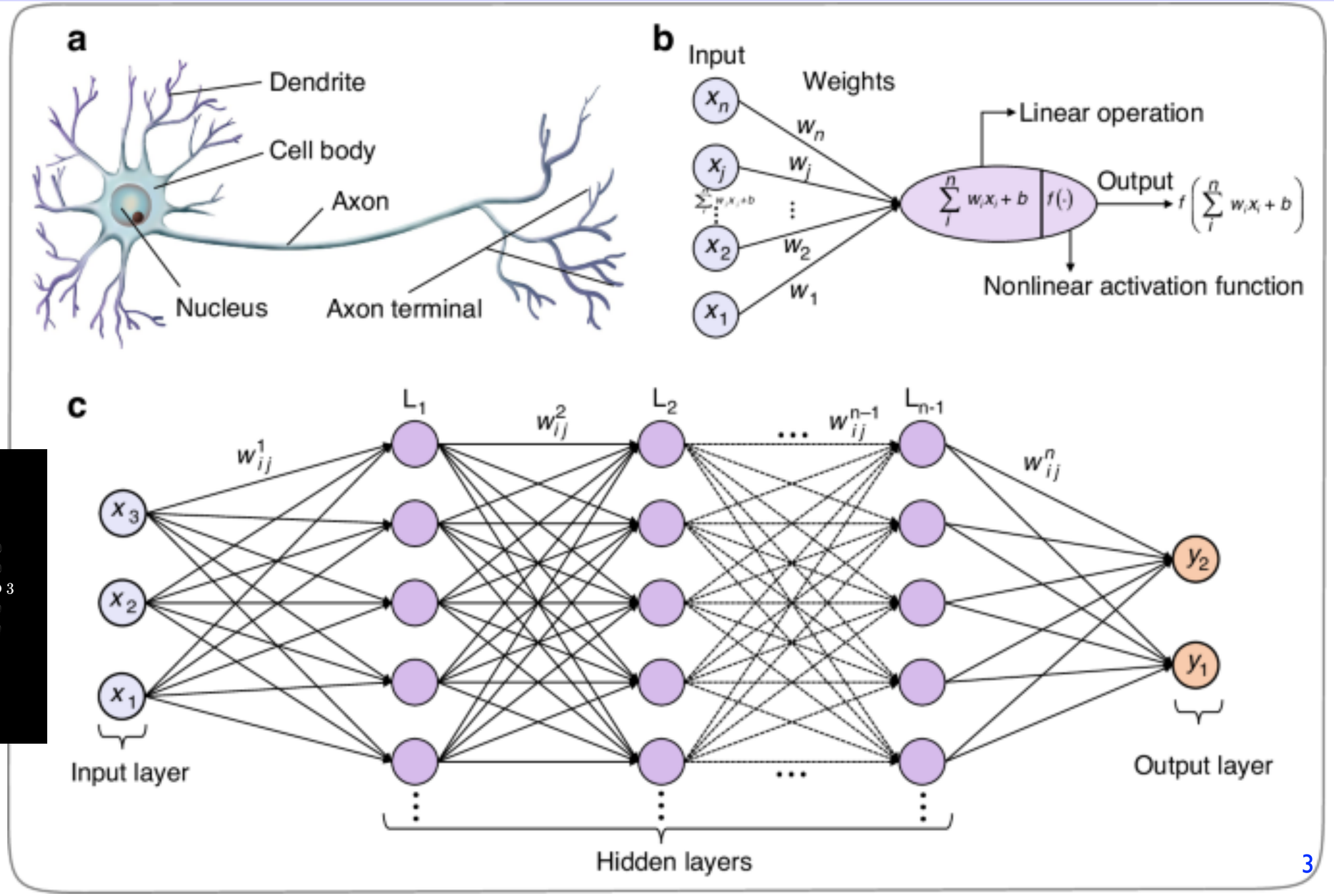
Multilayer Perceptron (Preview of DL)





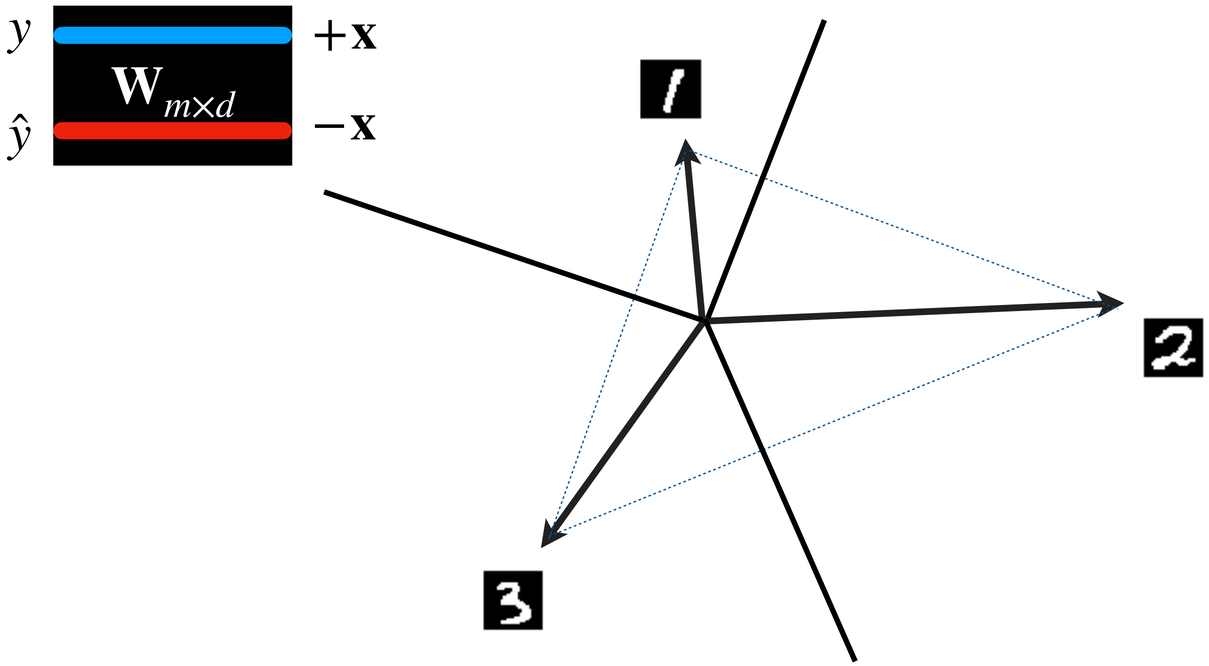
Multilayer Perceptron (Preview of DL)





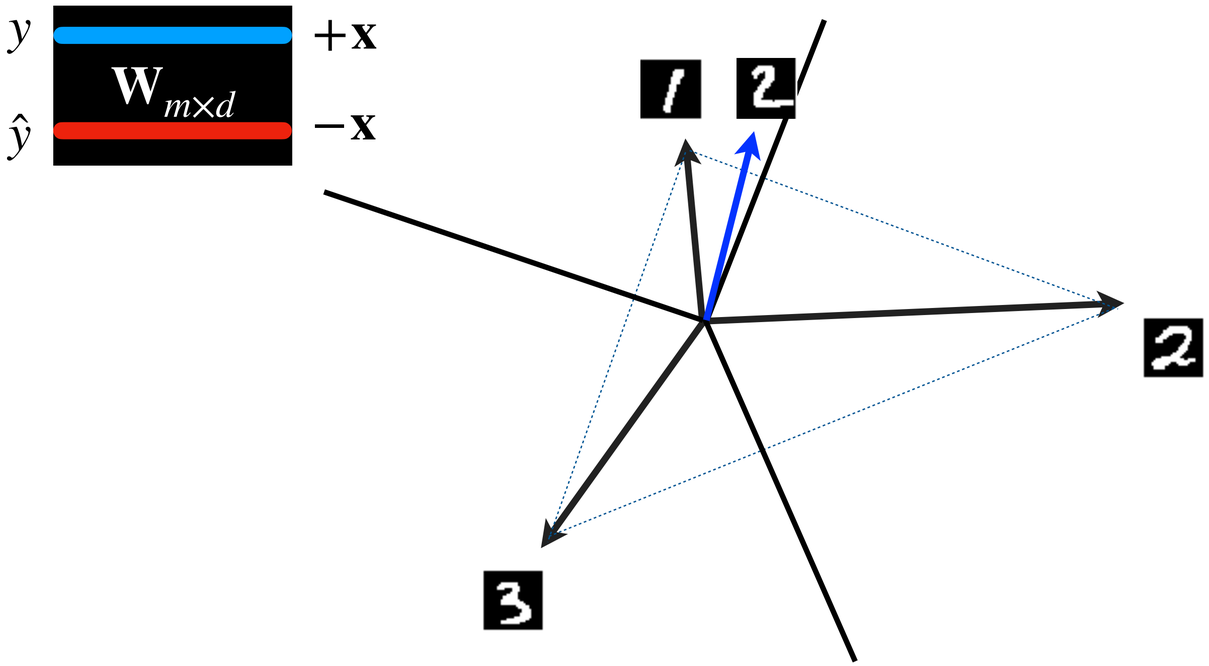
Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



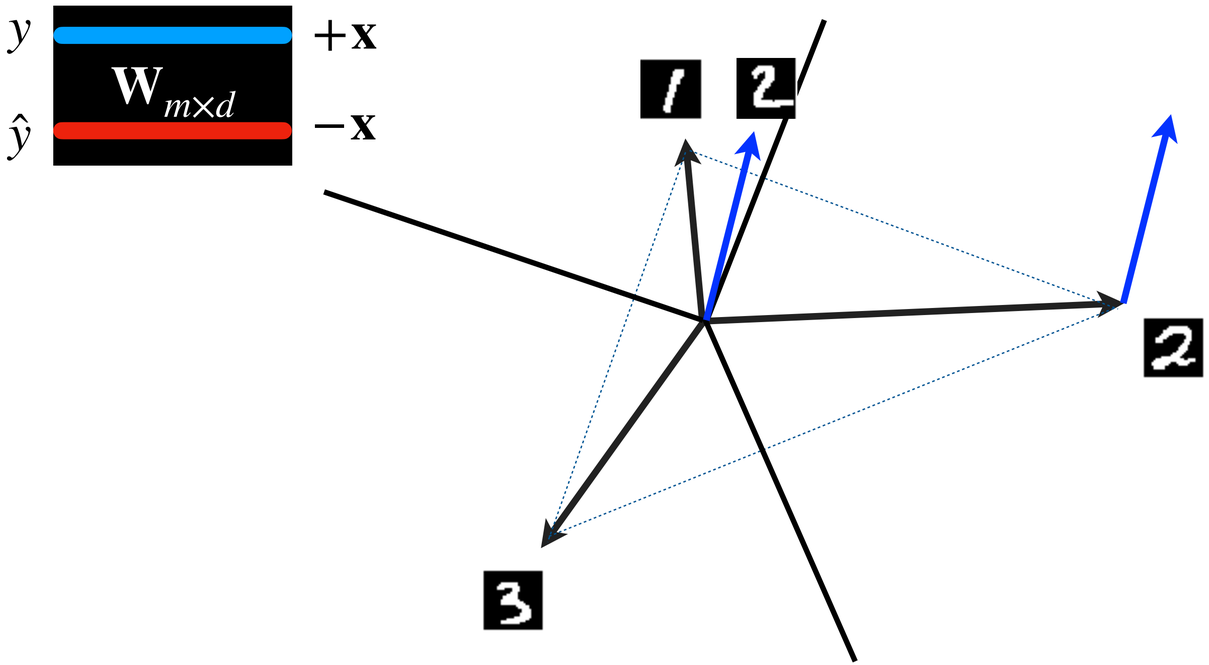
Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



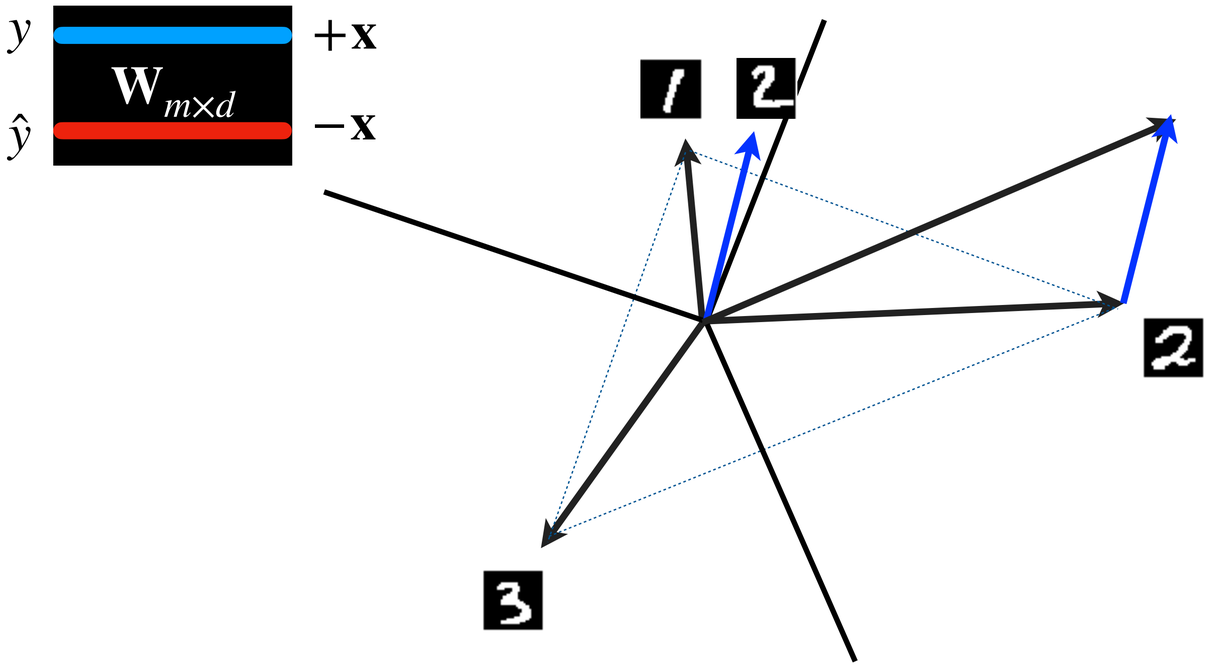
Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



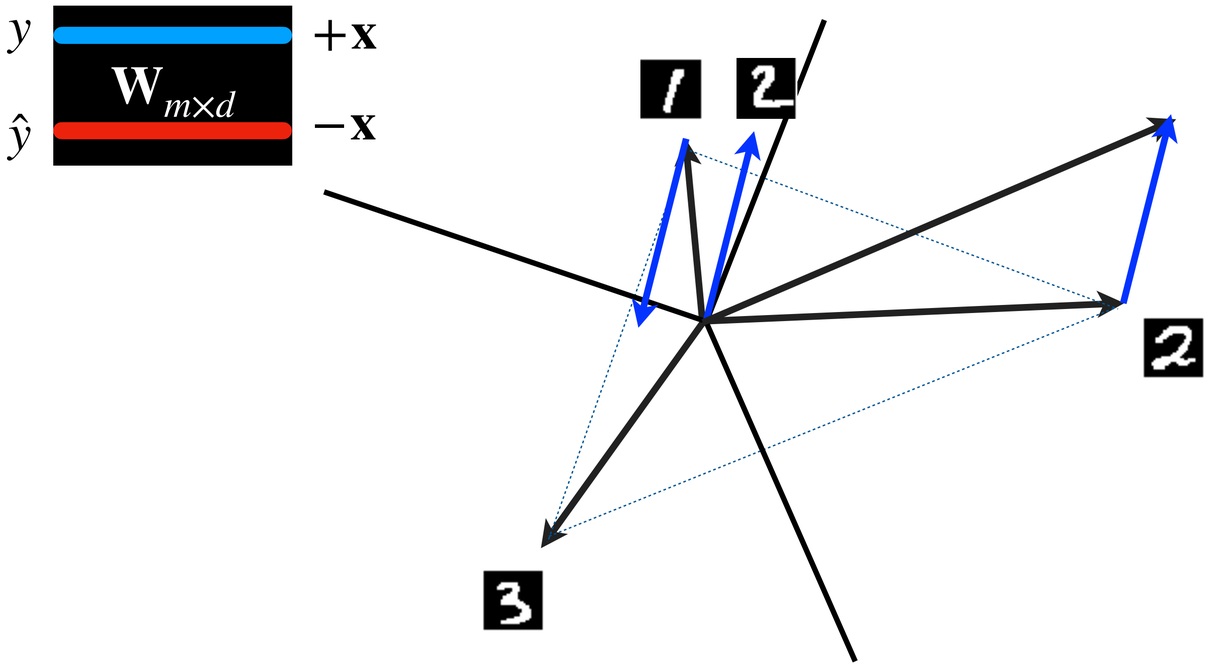
Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



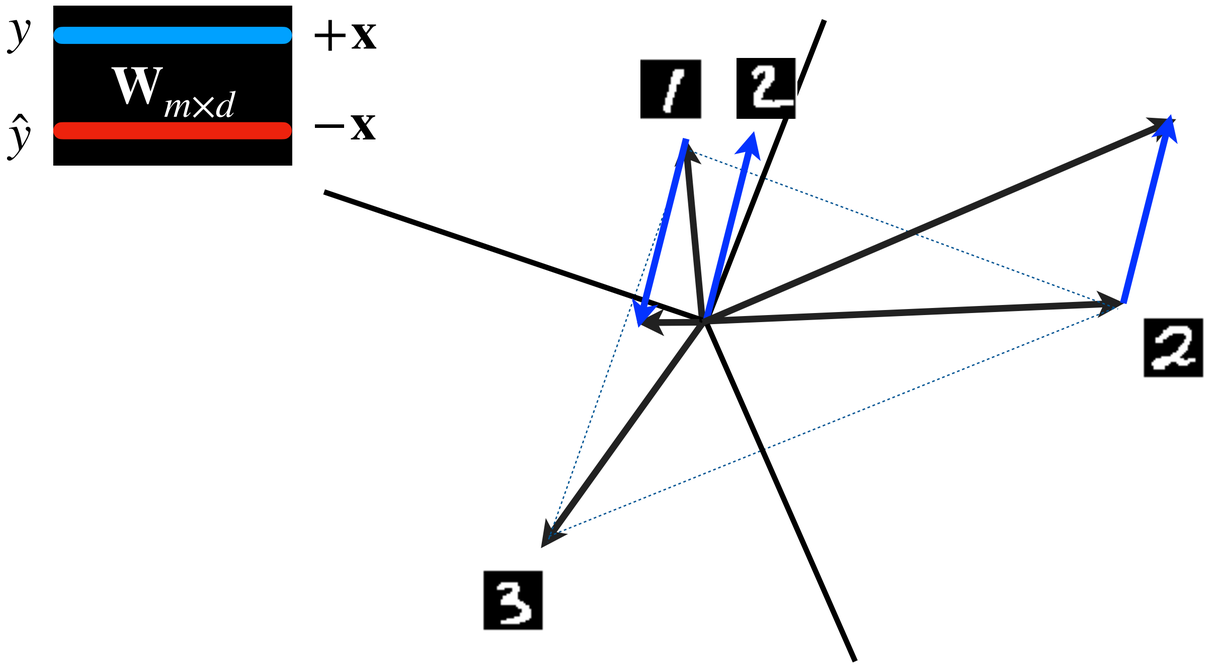
Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



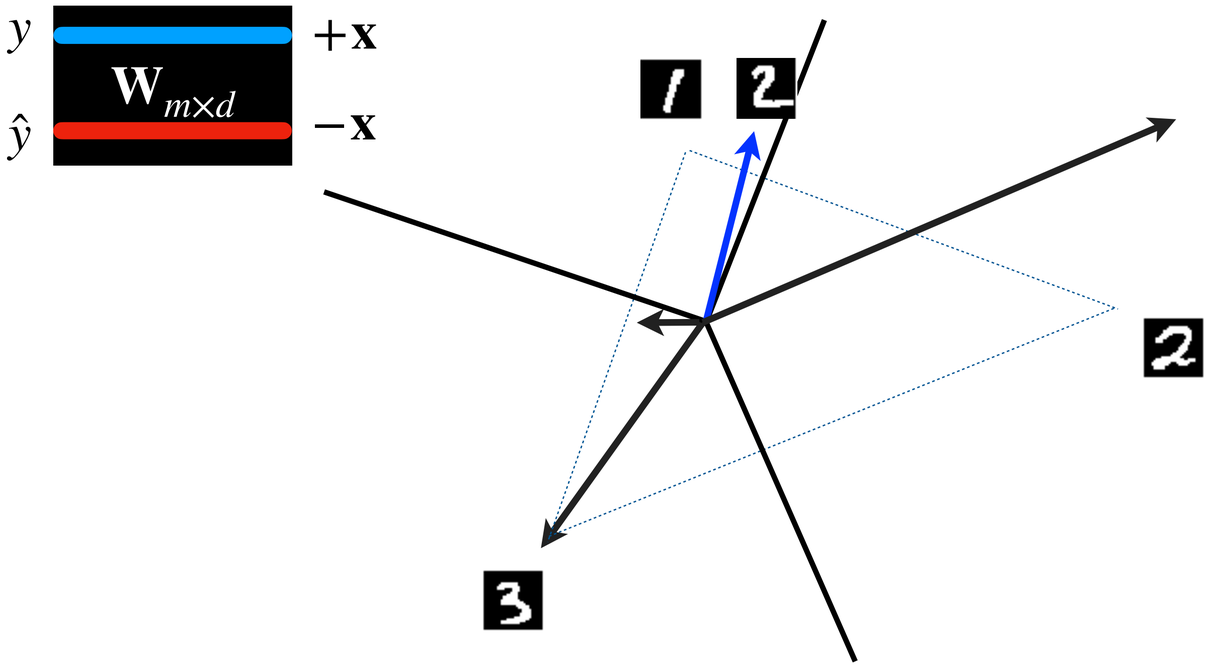
Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



Multiclass Perceptron

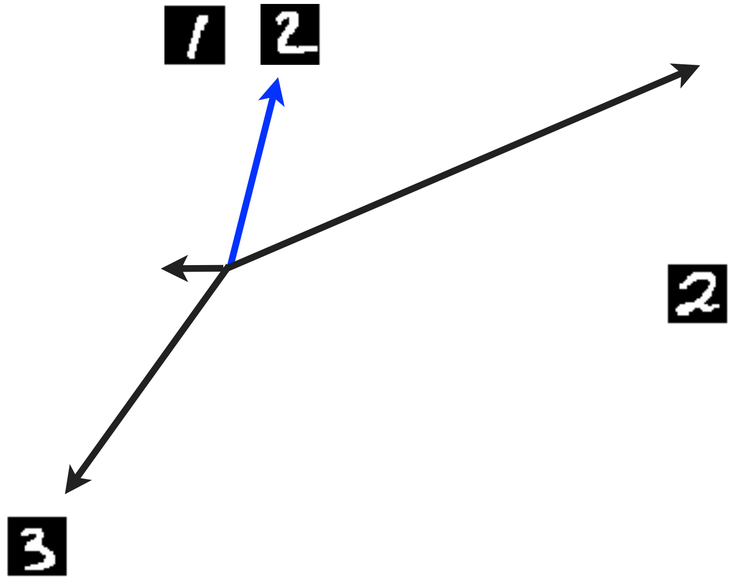
- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$

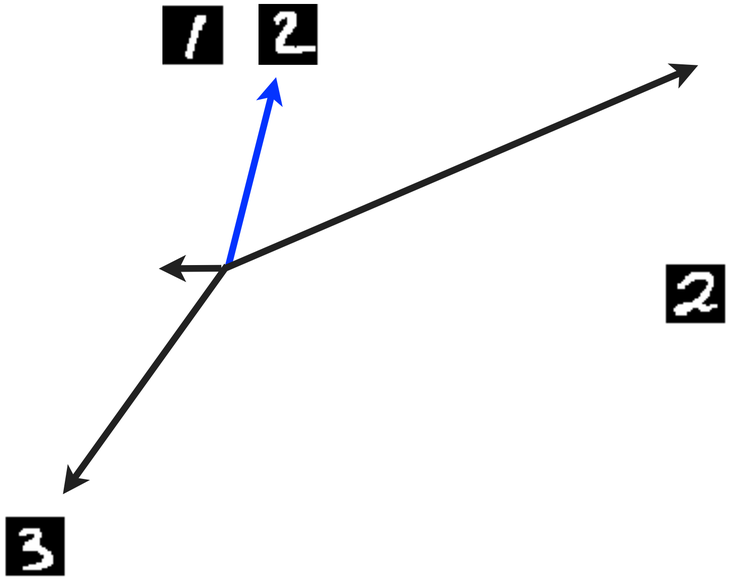


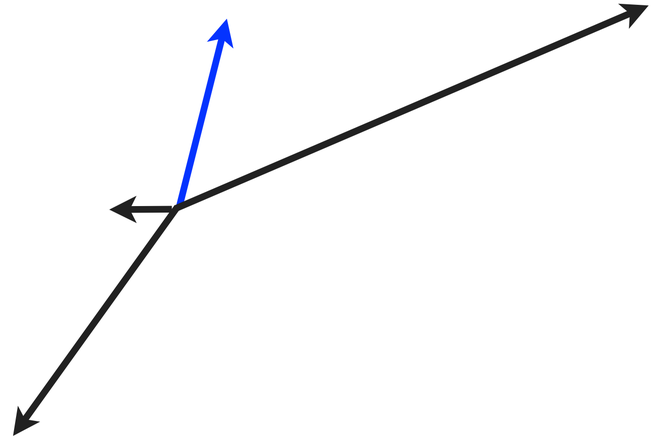


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



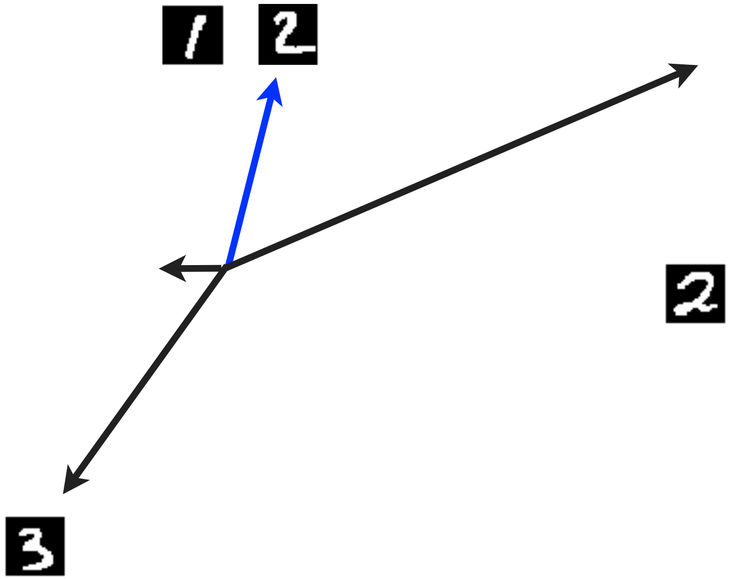


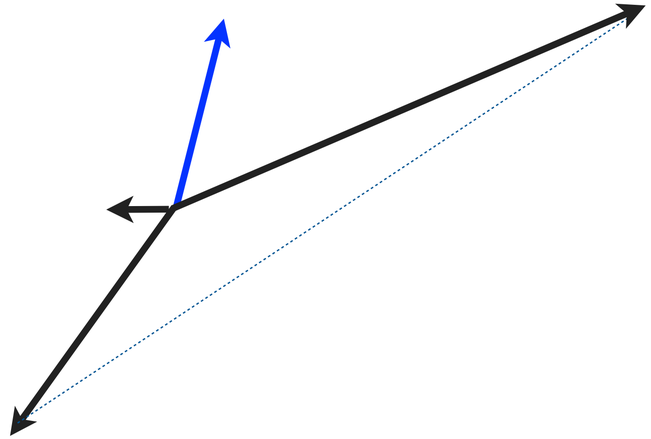


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



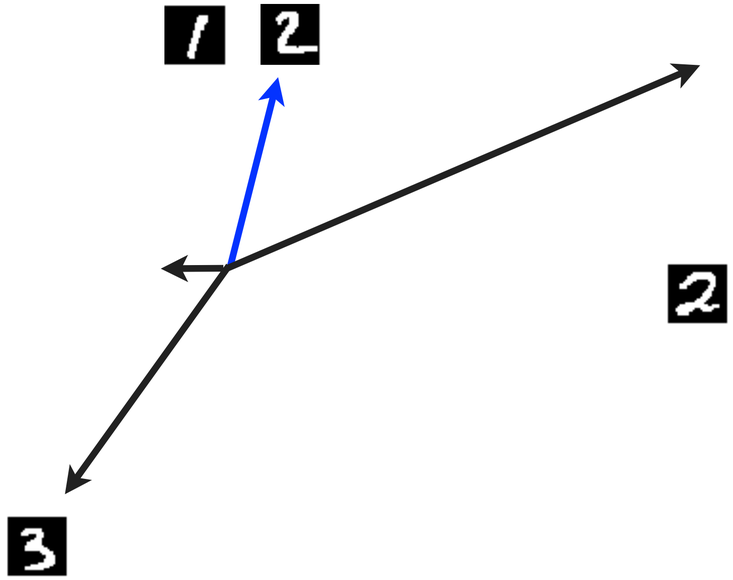


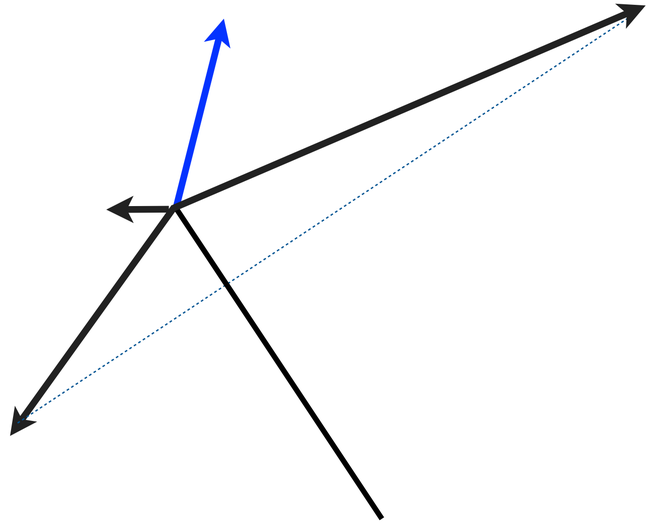


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



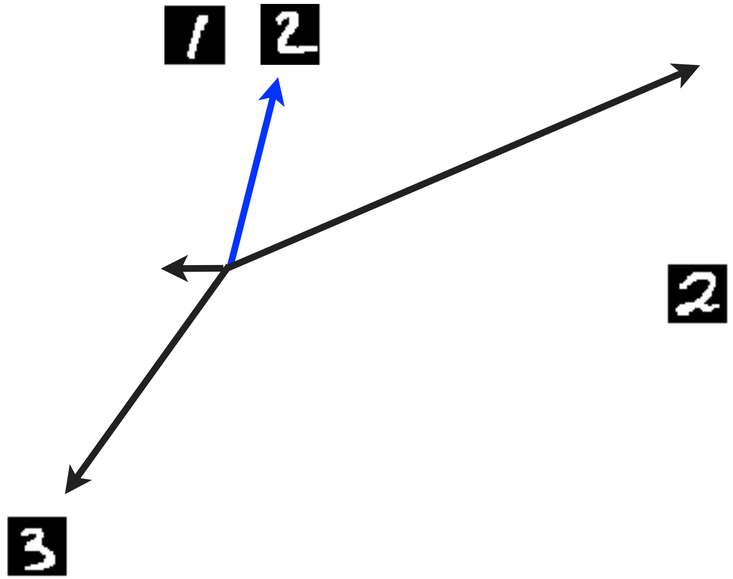


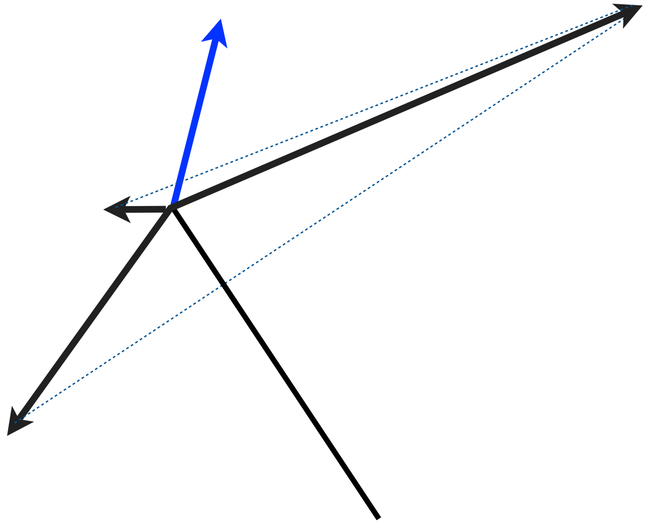


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



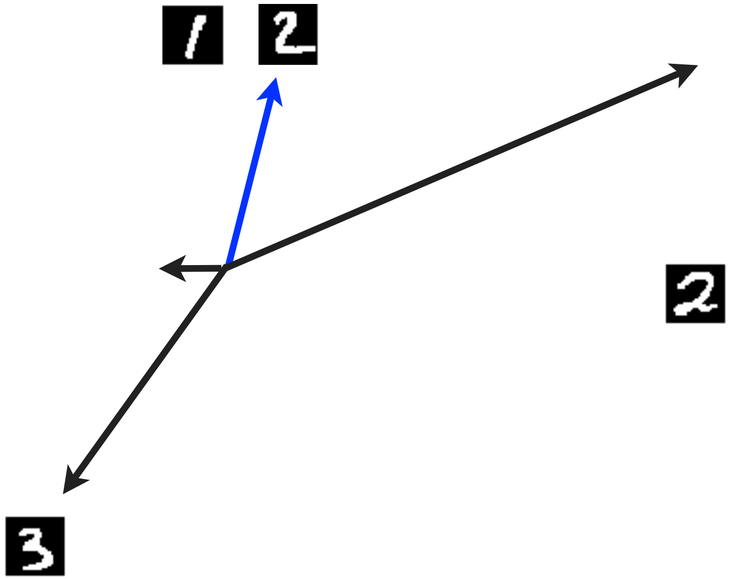


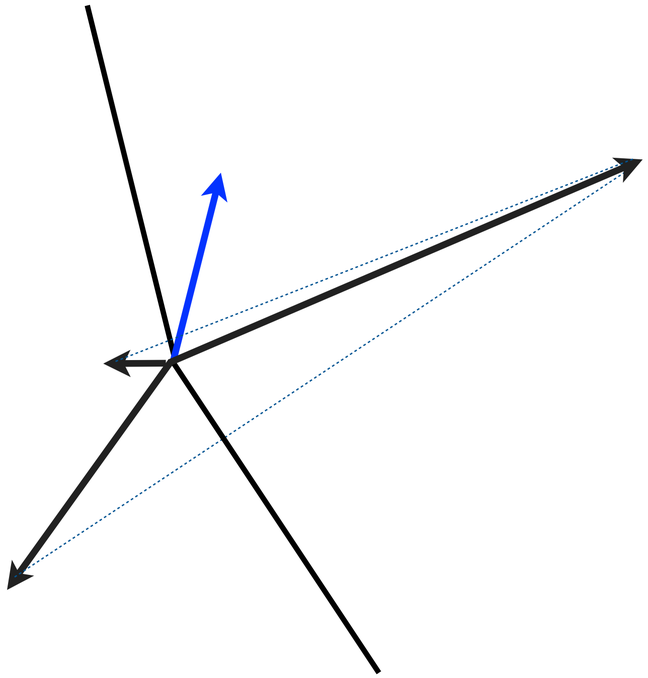


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



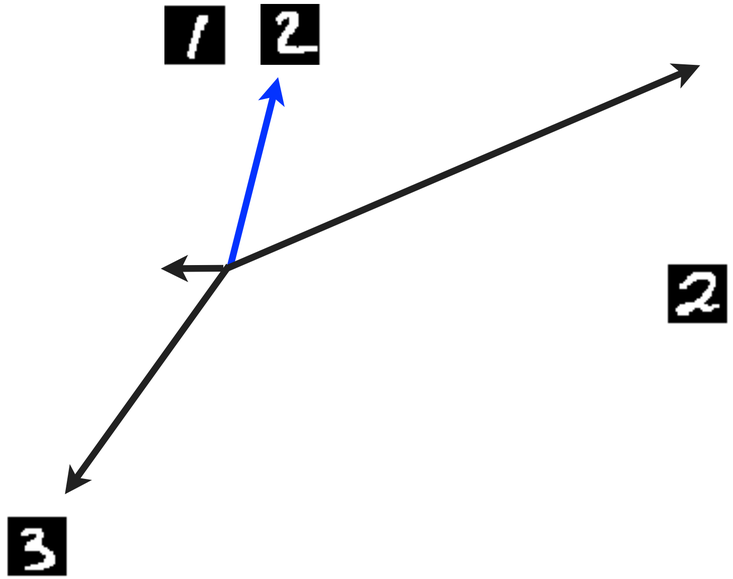


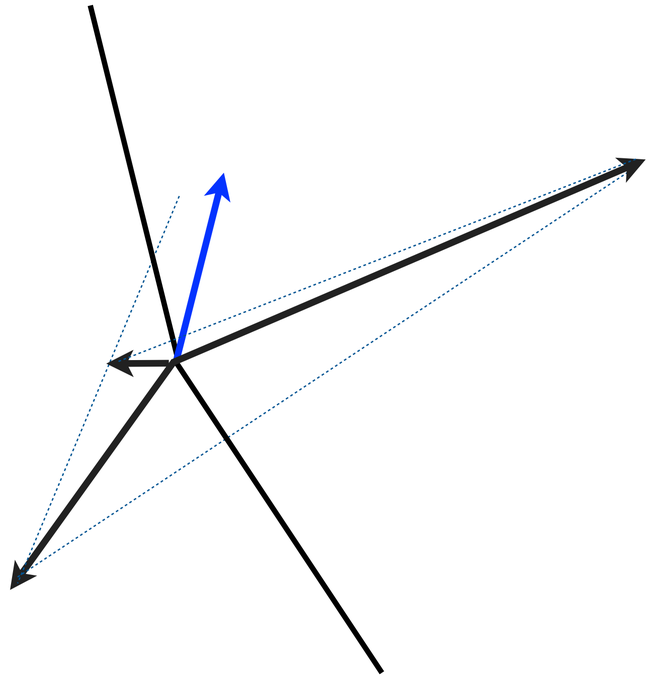


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



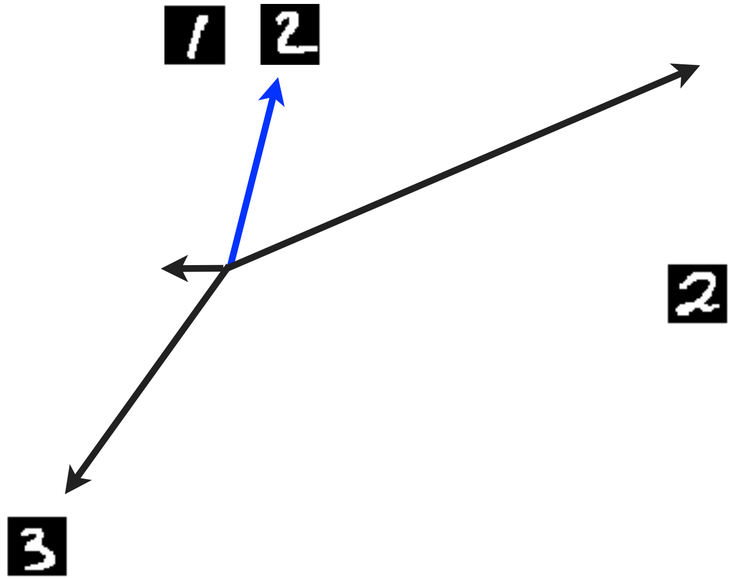


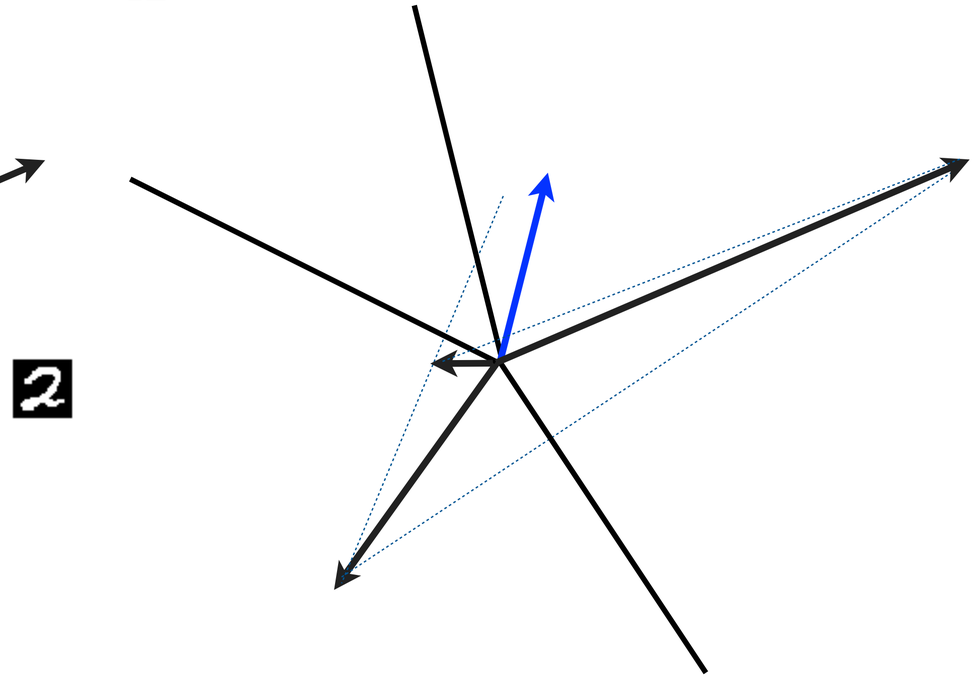


Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$



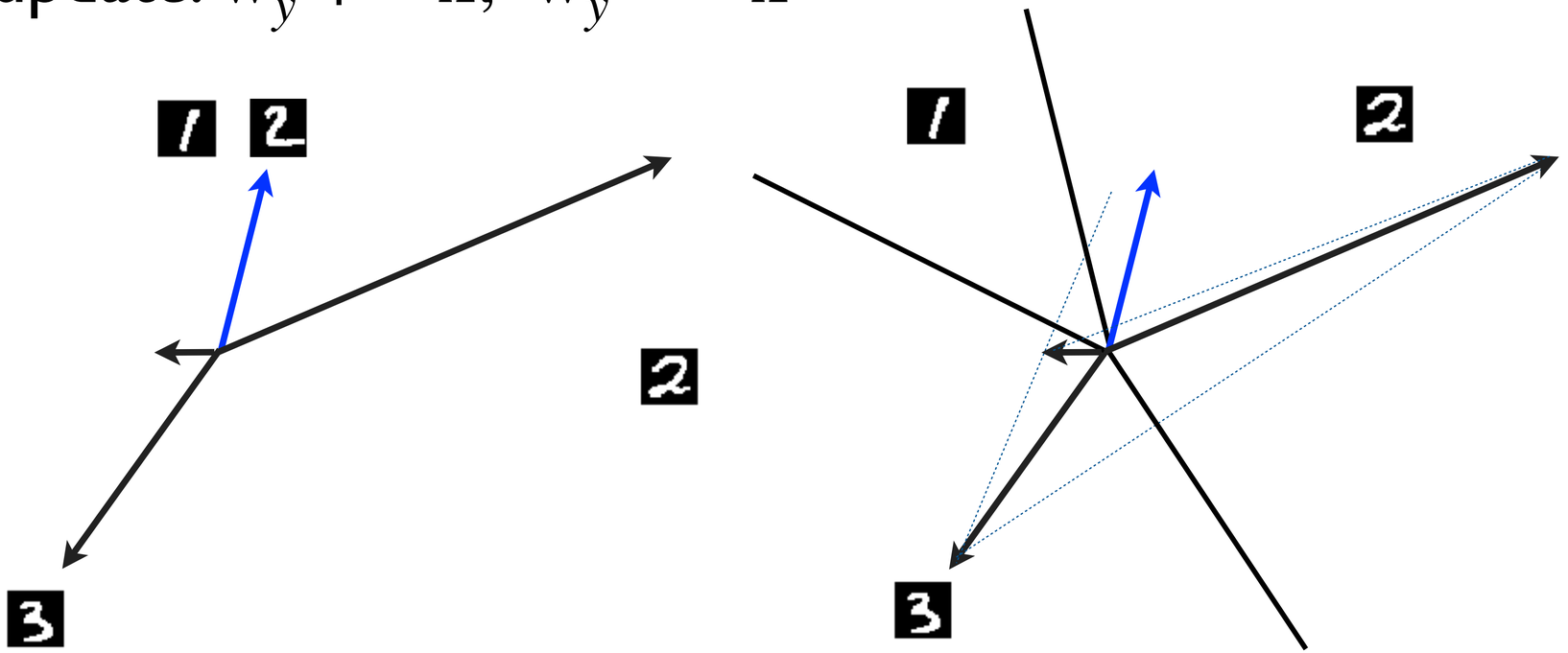




Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x}$; $\mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$





Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x};\ \mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$

with explicit bias:

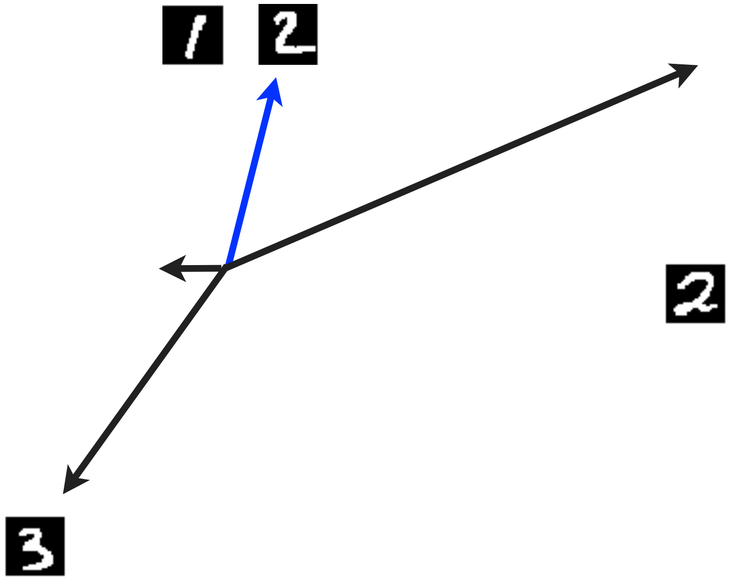
$$\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x} + b_k$$

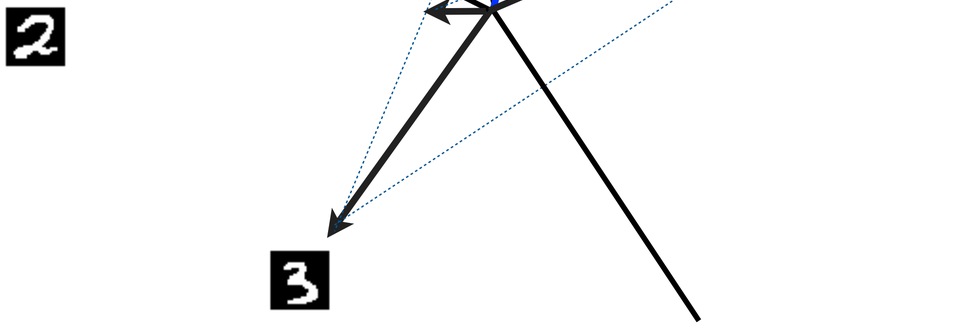
update rule:

$$\mathbf{w}_y \mathrel{+}= \mathbf{x};\ \mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x};$$

$$b_y \mathrel{+}= 1;\ b_{\hat{y}} \mathrel{-}= 1$$







Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x};\ \mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$

with explicit bias:

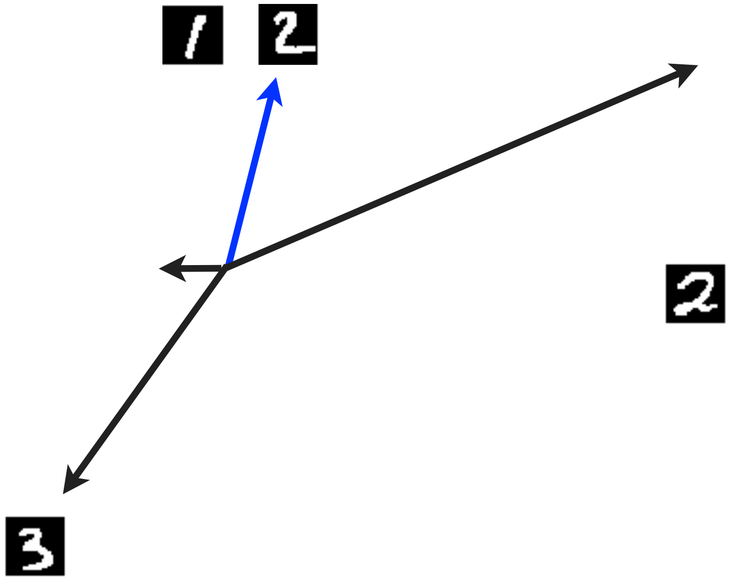
$$\hat{y} = \arg\max_{k=1\ldots m} \mathbf{w}_k \cdot \mathbf{x} + b_k$$

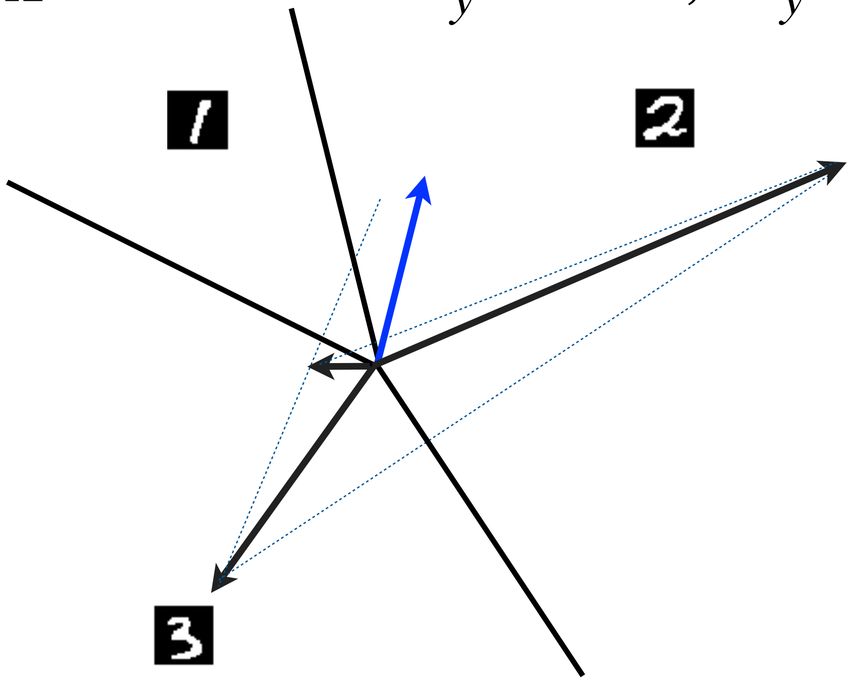
update rule:

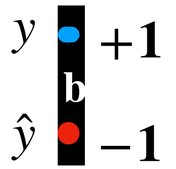
$$\mathbf{w}_y \mathrel{+}= \mathbf{x};\ \mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x};$$

$$b_y \mathrel{+}= 1;\ b_{\hat{y}} \mathrel{-}= 1$$









Multiclass Perceptron

- decision rule: $\hat{y} = \arg\max_{k=1...m} \mathbf{w}_k \cdot \mathbf{x}$
- on an error, penalize the weight for the wrong class, and reward the weight for the true class
- i.e., if $\hat{y} \neq y$, update: $\mathbf{w}_y \mathrel{+}= \mathbf{x};\ \mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x}$

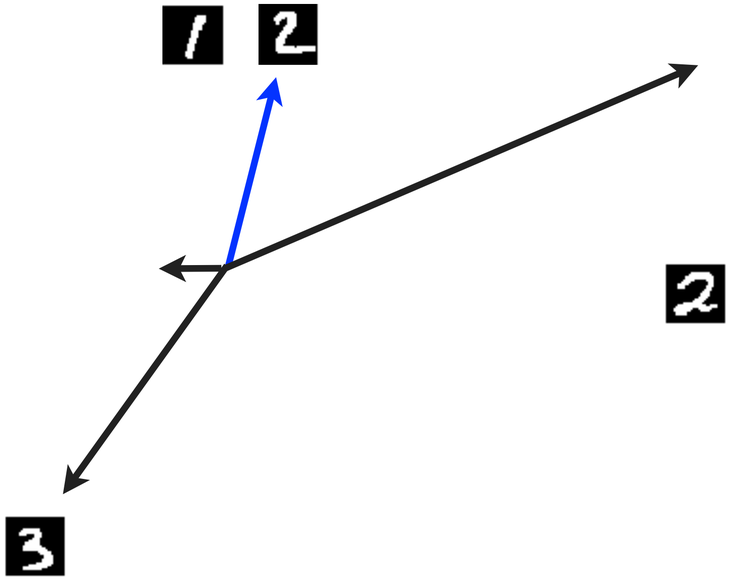
with explicit bias:

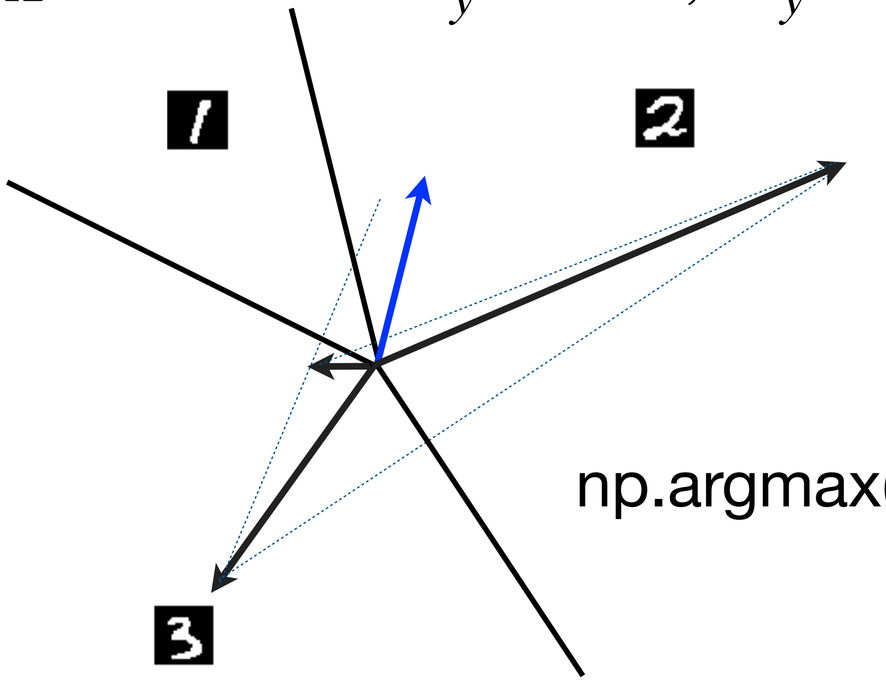
$$\hat{y} = \arg\max_{k=1...m} \mathbf{w}_k \cdot \mathbf{x} + b_k$$

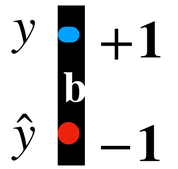
update rule:

$$\mathbf{w}_y \mathrel{+}= \mathbf{x};\ \mathbf{w}_{\hat{y}} \mathrel{-}= \mathbf{x};$$
$$b_y \mathrel{+}= 1;\ b_{\hat{y}} \mathrel{-}= 1$$









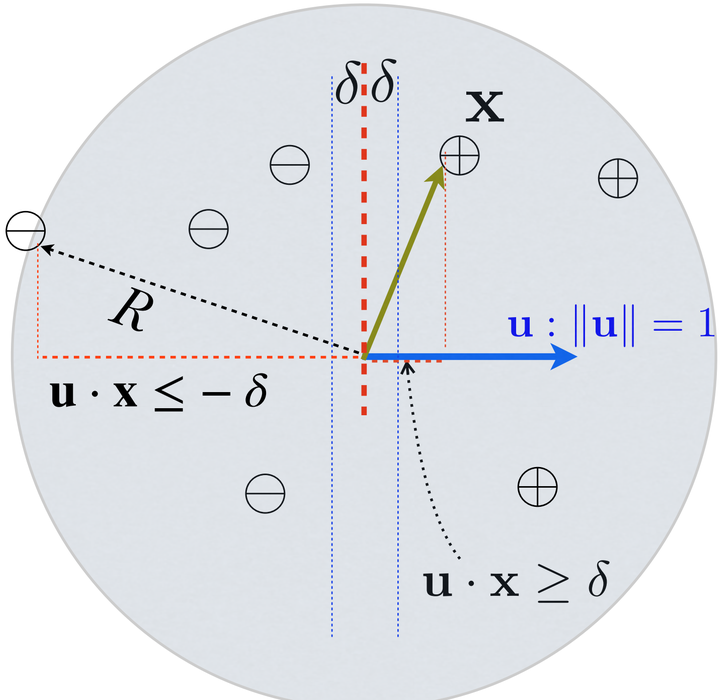
np.argmax($\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)

Review: Linear Separation; Convergence Theorem

- dataset *D* is said to be "linearly separable" if there exists some unit oracle vector $\mathbf{u}$: $||\mathbf{u}|| = 1$ which correctly classifies every example $(\mathbf{x}, y)$ with a margin at least δ:

$$y(\mathbf{u} \cdot \mathbf{x}) \geq \delta \text{ for all } (\mathbf{x}, y) \in D$$

- then the perceptron must converge to a linear separator after at most R^2/δ^2 mistakes (updates) where $R = \max_{(\mathbf{x},y)\in D} \|\mathbf{x}\|$
- convergence rate R^2/δ^2
  - dimensionality independent
  - dataset size independent
  - order independent (but order matters in output)
  - scales with 'difficulty' of problem

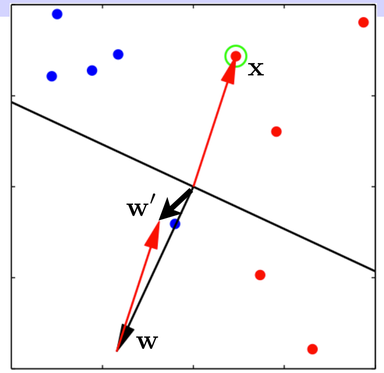


Convergence of Multiclass Perceptron

- dataset $D = \{(\mathbf{x}, y)\}_1^n$ is said to be linearly (mutliclass) separable
  - if there exists a row-normalized weight matrix $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_m)$ (i.e., $\forall i, |\mathbf{u}_i| = 1$)
  - such that for each example $(\mathbf{x}, y) \in D$,
  - for any wrong label $z \neq y$, we have $\mathbf{u}_y \cdot \mathbf{x} - \mathbf{u}_z \cdot x \geq \delta$ (correct - wrong is big enough)
- if a dataset is separable with margin δ, then multiclass perceptron is bound to converge after at most R^2/δ^2 updates where radius $R = \max_{\mathbf{x}} |\mathbf{x}|$
- proof is a simple (but ingenious) extension of the Novikoff (1962) proof for the binary perceptron; this is from the voted/avg perceptron paper (1999)

From Binary MIRA to Multiclass MIRA

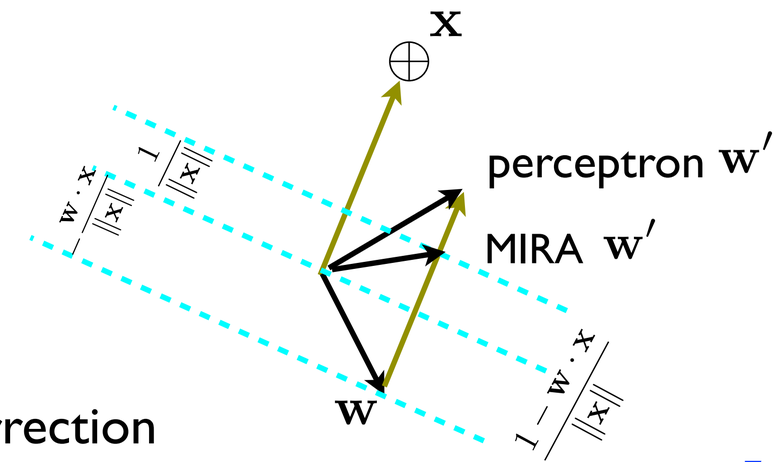
- perceptron often
  - either makes too big updates (over-correction)
  - or too small updates (under-correction)



under-correction

- can we make "just enough" update to correct the mistake w/ some margin?

1: Input: training set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
2: Output: weight vector $\mathbf{w}$
3: $\mathbf{w} \leftarrow \mathbf{0}$
4: **repeat**
5: **for each** example $(\mathbf{x}, y)$ **in** D **do**
6: **if** $y(\mathbf{w} \cdot \mathbf{x}) \leq 1 - \epsilon$ **then**
7: $\mathbf{w} \leftarrow \mathbf{w} + \frac{y - \mathbf{w} \cdot \mathbf{x}}{\|\mathbf{x}\|^2}\mathbf{x}$
8: **until** converged

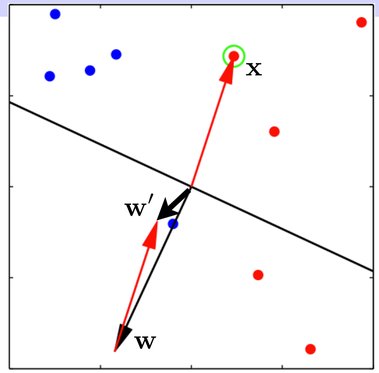


over-correction

Multiclass MIRA

- perceptron often
  - either makes too big updates (over-correction)
  - or too small updates (under-correction)

under-correction



- can we make "just enough" update to correct the mistake w/ some margin?

1: Input: training set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
2: Output: weight vector $\mathbf{w}$
3: $\mathbf{w} \leftarrow \mathbf{0}$
4: **repeat**
5: **for each** example $(\mathbf{x}, y)$ **in** D **do**
6: **if** $y(\mathbf{w} \cdot \mathbf{x}) \leq 1 - \epsilon$ **then**
7: $\mathbf{w} \leftarrow \mathbf{w} + \frac{y - \mathbf{w} \cdot \mathbf{x}}{\|\mathbf{x}\|^2}\mathbf{x}$
8: **until** converged

worst offending class: $\hat{y} = \arg\max_{z \neq y} \mathbf{w}_z \cdot \mathbf{x}$

margin = $\mathbf{w}_y \cdot \mathbf{x} - \mathbf{w}_{\hat{y}} \cdot \mathbf{x}$

if margin $\leq$ aggr: # aggr: 0…1

update ratio $\tau = \dfrac{1 - \text{margin}}{2\,|\mathbf{x}|^2}$

$\mathbf{w}_y \mathrel{+}= \tau\mathbf{x}$

$\mathbf{w}_{\hat{y}} \mathrel{-}= \tau\mathbf{x}$

goal: after update, margin=1

Derivation (here $L(y, \hat{y}) = 1$)

The margin constraint requires:

$$(\mathbf{w}'_y - \mathbf{w}'_{\hat{y}}) \cdot \mathbf{x} \geq L(y, \hat{y})$$

Substitute $\mathbf{w}'_y = \mathbf{w}_y + \tau\mathbf{x}$ and $\mathbf{w}'_{\hat{y}} = \mathbf{w}_{\hat{y}} - \tau\mathbf{x}$:

$$((\mathbf{w}_y + \tau\mathbf{x}) - (\mathbf{w}_{\hat{y}} - \tau\mathbf{x})) \cdot \mathbf{x} = L(y, \hat{y})$$

$$(\mathbf{w}_y - \mathbf{w}_{\hat{y}} + 2\tau\mathbf{x}) \cdot \mathbf{x} = L(y, \hat{y})$$

$$(\mathbf{w}_y - \mathbf{w}_{\hat{y}}) \cdot \mathbf{x} + \mathbf{2}\tau\|\mathbf{x}\|^{\mathbf{2}} = L(y, \hat{y})$$

Isolate $2\tau\|\mathbf{x}\|^2$:

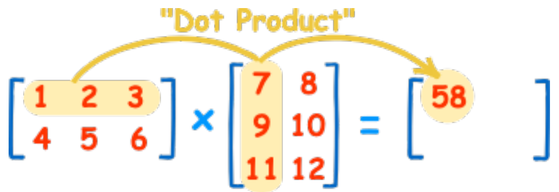
$$2\tau\|\mathbf{x}\|^2 = L(y, \hat{y}) - (\mathbf{w}_y \cdot \mathbf{x} - \mathbf{w}_{\hat{y}} \cdot \mathbf{x})$$

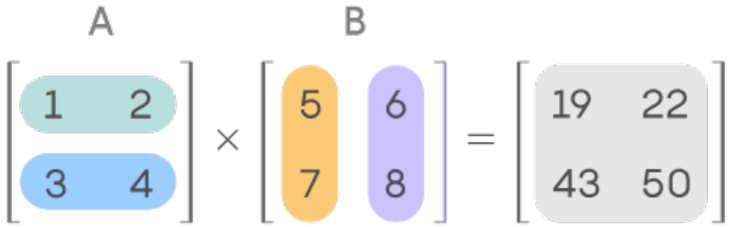
Divide both sides by $2\|\mathbf{x}\|^2$:

$$\tau = \frac{L(y, \hat{y}) - (\mathbf{w}_y \cdot \mathbf{x} - \mathbf{w}_{\hat{y}} \cdot \mathbf{x})}{\mathbf{2}\|\mathbf{x}\|^{\mathbf{2}}}$$

Test Time of Multiclass Linear Classification

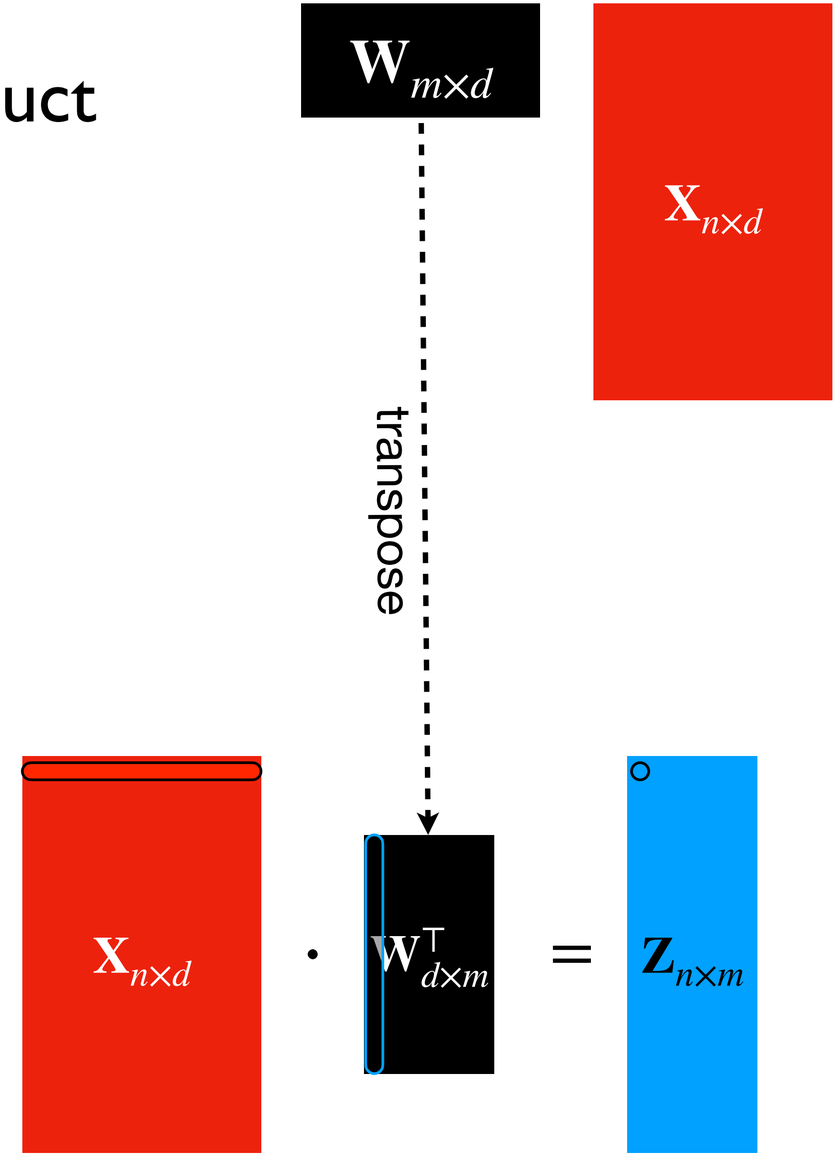
- matrix-matrix multiplication is generalized dot-product
- transpose weight matrix: $\mathbf{W}_{d\times m}^{\top}$
- $\mathbf{X}_{n\times d} \cdot \mathbf{W}_{d\times m}^{\top} = \mathbf{Z}_{n\times m}$
- np.argmax($\mathbf{Z}_{n\times m}$, axis=1) gets the predictions





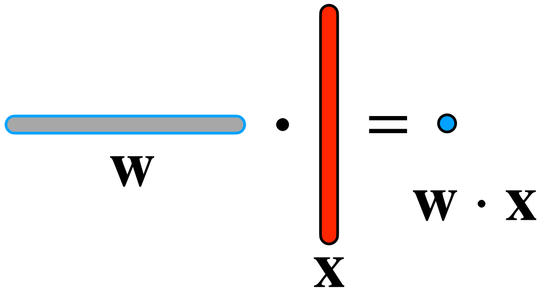
$$\begin{matrix} & \vec{b_1} & \vec{b_2} \\ & \downarrow & \downarrow \\ \vec{a_1}\rightarrow & 1 & 7 \\ \vec{a_2}\rightarrow & 2 & 4 \end{matrix} \cdot \begin{bmatrix} 3 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} \vec{a_1}\cdot\vec{b_1} & \vec{a_1}\cdot\vec{b_2} \\ \vec{a_2}\cdot\vec{b_1} & \vec{a_2}\cdot\vec{b_2} \end{bmatrix}$$

$A \qquad B \qquad C$

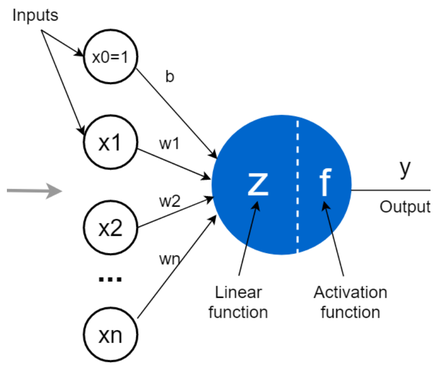


Summary of Binary & Multiclass Perceptrons

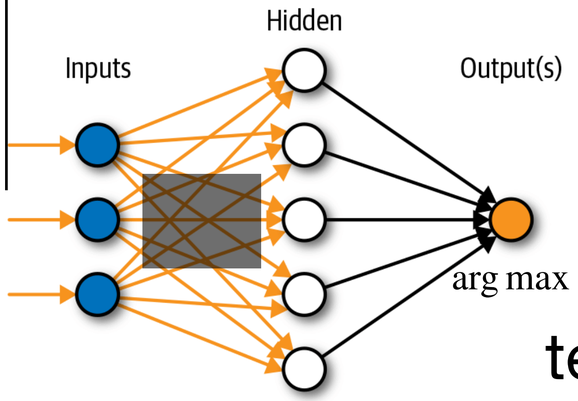
training (per example):
vector-vector dot-product



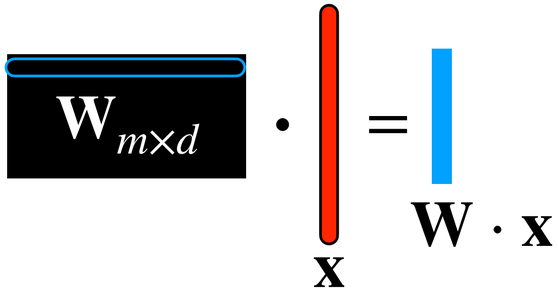
binary



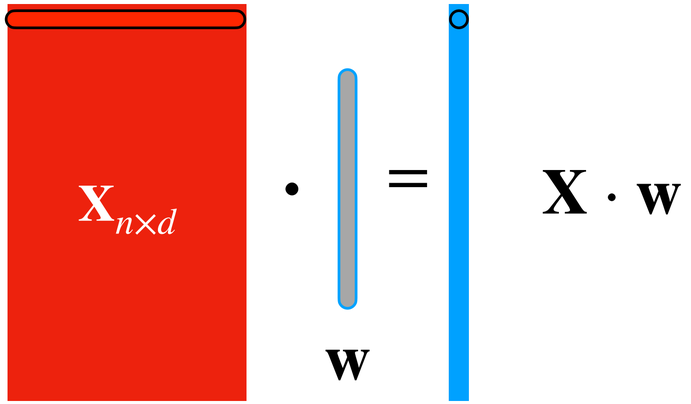
multiclass



training (per example):
matrix-vector product



testing (all examples):
matrix-vector product



in numpy, a vector can be interpreted as both a row vector and a column vector depending on context

testing (all examples):
matrix-matrix product

