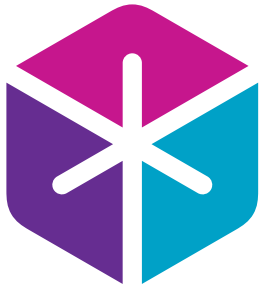
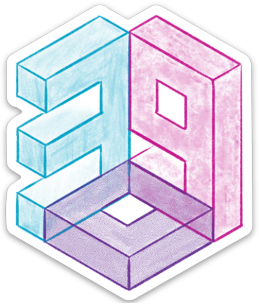




https://introml.mit.edu/

6.390 Intro to Machine Learning

Spring 25 Final Review

Shen Shen

May 13, 2025

5pm, Room 34-101

Outline

- Semester Overview
- Weekly Review
- Exam Prep Resources, live Q&A
- Future courses

- Intro to ML
- Linear Regression
- Gradient Descent
- Linear Classification
- Features, Neural Networks I
- Neural Networks II
- Convolutional Neural Networks
- Representation Learning
- Transformers
- Markov Decision Processes
- Reinforcement Learning
- Non-parametric Models

Many other ways to dissect

Model class

- linear models
- linear model on non-linear features
- fully connected feed-forward
- convolutional
- transformers
- Q-table
- tree, k-nearest neighbor, k-means

Learning process:

- training/validation/testing
- overfitting/underfitting
- regularization
- hyper parameters

Modeling choices:

- Supervised:
  - regression
  - classification
- Unsupervised/self-supervised
- Reinforcement/sequential

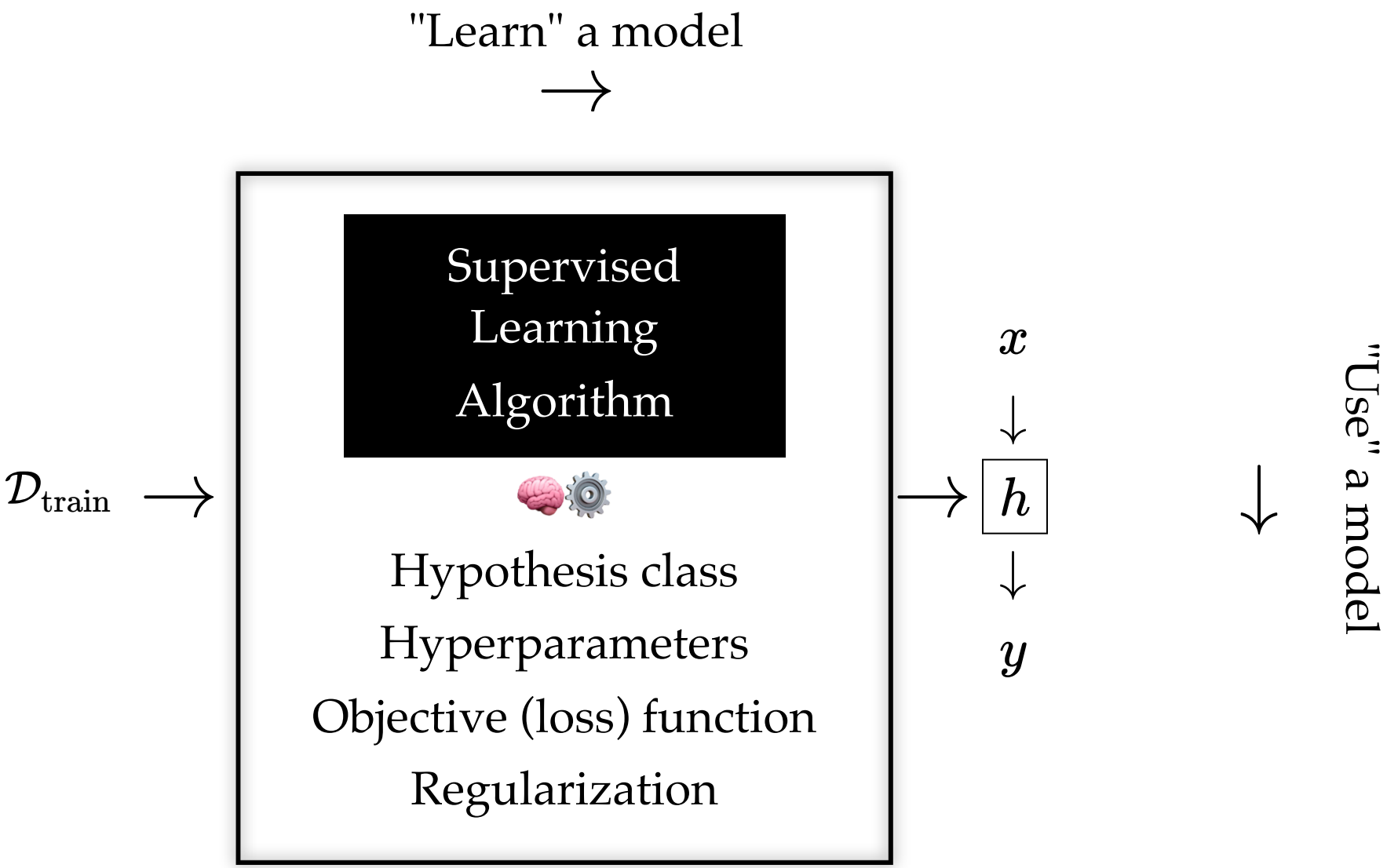
Optimization:

- analytical solutions
- gradient descent
- back propagation
- value iteration, Q-learning
- non-parametric methods

[The lists are non-exclusive, incomplete]

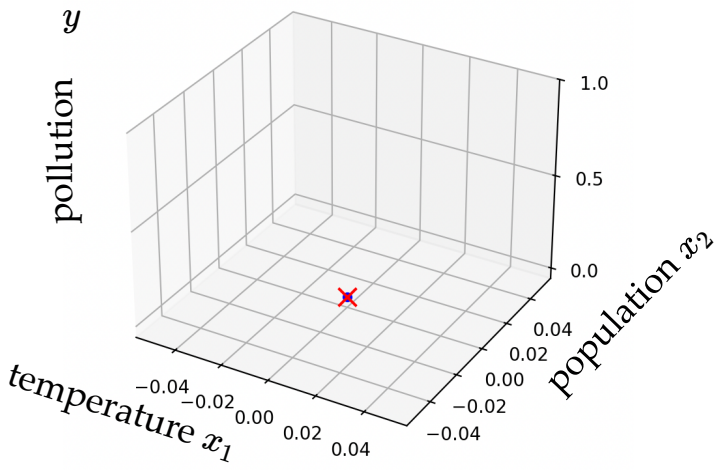
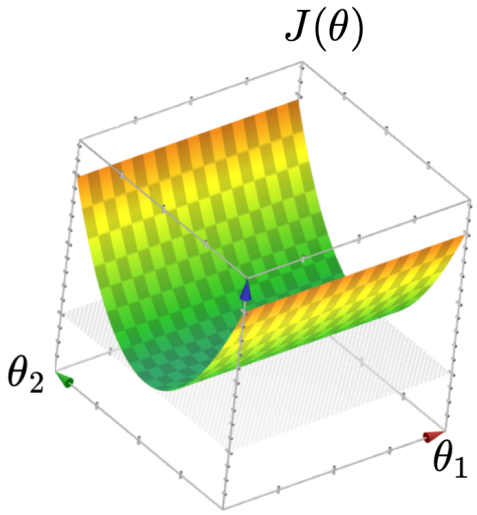
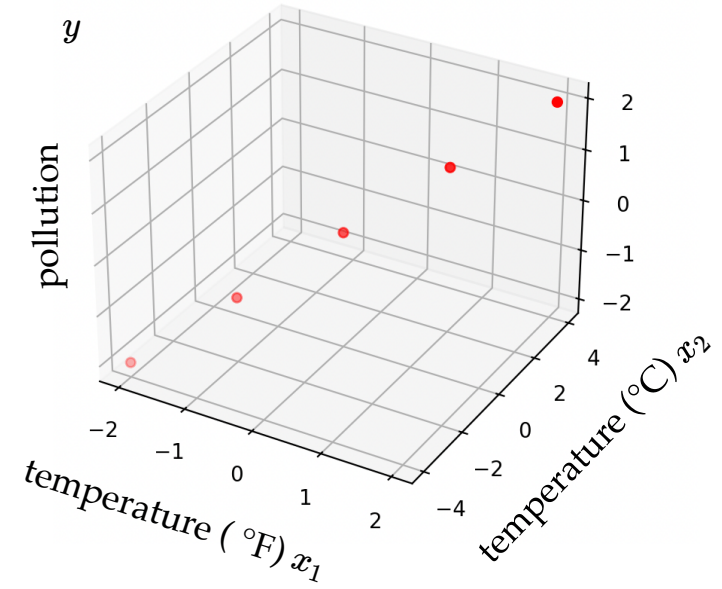
Week 1 - IntroML

- Terminologies
  - Training, validation, testing
  - Identifying overfitting and underfitting
- Concrete processes
  - Learning algorithm
  - Validation and Cross-validation
  - Concept of hyperparameter


"Learn" a model
Supervised Learning Algorithm
$\mathcal{D}_{\text{train}}$
Hypothesis class
Hyperparameters
Objective (loss) function
Regularization
x
h
y
"Use" a model

Week 2 - Regression

- Problem Setup
- Analytical solution formula $\theta^* = \left(X^\top X\right)^{-1} X^\top Y$ (and what's X)
- When $X^\top X$ not invertible (optimal solution**s** still exist; just not via the "formula")
  - Practically (two scenarios)
  - Visually (obj fun no longer of "bowl" shape, instead has "half-pipe" shape)
  - Mathematically (loss of solution uniqueness)
- Regularization
  - Motivation, how to, when to

Case	Example	Objective Function Looks Like	Optimal Parameters
2a. less data than features			infinitely many optimal parameters (that define optimal hyperplanes)
2b. linearly dependent features			

Week 3 - Gradient Descent

- The gradient vector (both analytically and conceptually)
- The gradient-descent algorithm and the key update formula
- (Convex + small-enough step-size + gradient descent + global min exists + run long enough) guarantee convergence to a global min
  - What happens when any of these conditions is violated
- How does the stochastic variant differ (Set up, run-time behavior, and conclusion)

For $f : \mathbb{R}^m \to \mathbb{R}$, its ***gradient*** $\nabla f : \mathbb{R}^m \to \mathbb{R}^m$ is defined at the point $p = (x_1, \ldots, x_m)$ as:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_m}(p) \end{bmatrix}$$

1. The gradient generalizes the concept of a derivative to multiple dimensions.
2. By construction, the gradient's dimensionality always matches the function input.

Sometimes the gradient is undefined or ill-behaved, but today it is well-behaved unless stated otherwise.

3. The gradient can be symbolic or numerical.

example: $f(x, y, z) = x^2 + y^3 + z$

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_m}(p) \end{bmatrix}$$

its *symbolic* gradient:

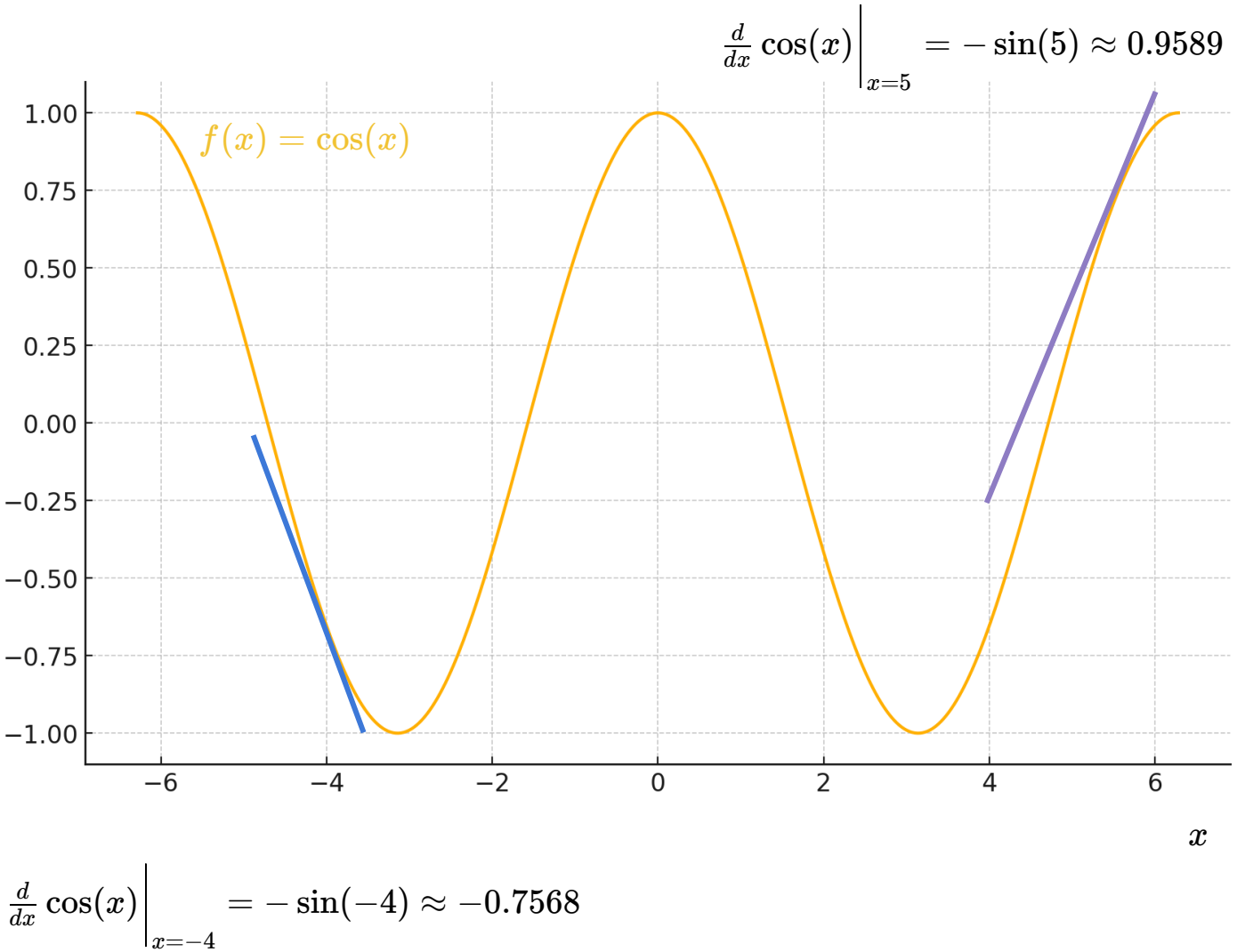
$$\nabla f(x, y, z) = \begin{bmatrix} 2x \\ 3y^2 \\ 1 \end{bmatrix}$$

evaluating the symbolic gradient at a point gives a *numerical* gradient:

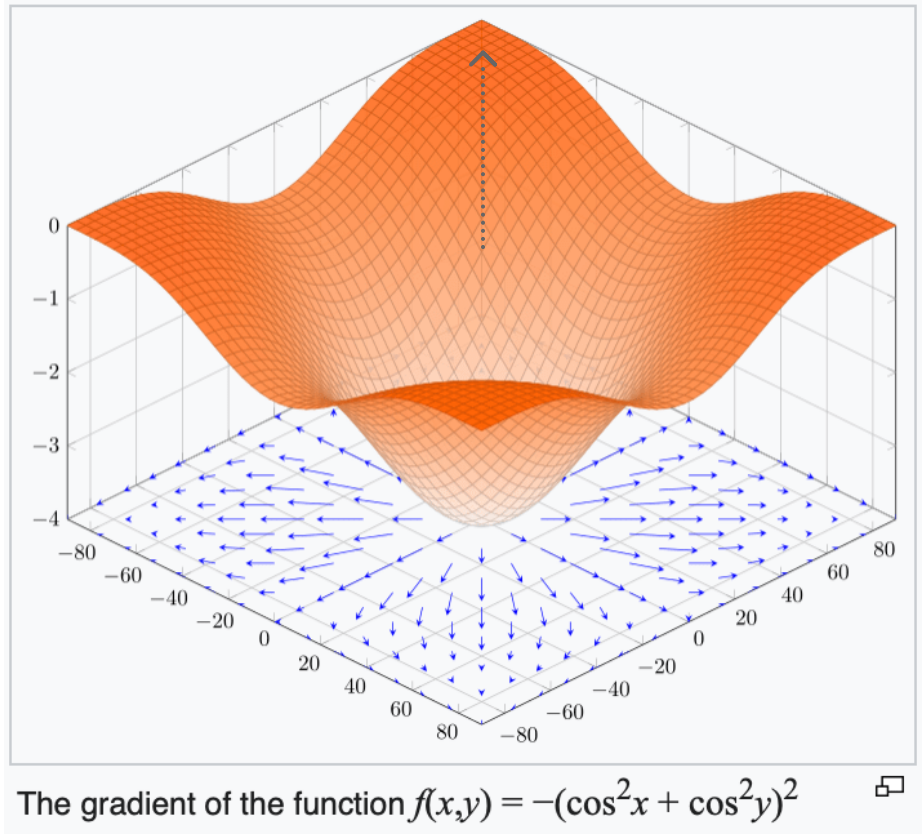
$$\nabla f(3, 2, 1) = \nabla f(x, y, z)\Big|_{(x,y,z)=(3,2,1)} = \begin{bmatrix} 6 \\ 12 \\ 1 \end{bmatrix}$$

just like a derivative can be a function or a number.

4. The gradient points in the direction of the (steepest) *increase* in the function value.



$\frac{d}{dx}\cos(x)\Big|_{x=-4} = -\sin(-4) \approx -0.7568$



The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$

initial guess — learning rate,

hyperparameters — of parameters — aka, step size — precision

```
Gradient-Descent( Θ_init, η, f, ∇_Θ f, ε )
  Initialize  Θ^(0) = Θ_init
  Initialize t = 0
  repeat
    t = t + 1
    Θ^(t) = Θ^(t−1) − η∇_Θ f(Θ^(t−1))
  until  |f(Θ^(t)) − f(Θ^(t−1))| < ε
  Return  Θ^(t)
```

1 Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

2 Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

3 Initialize t = 0

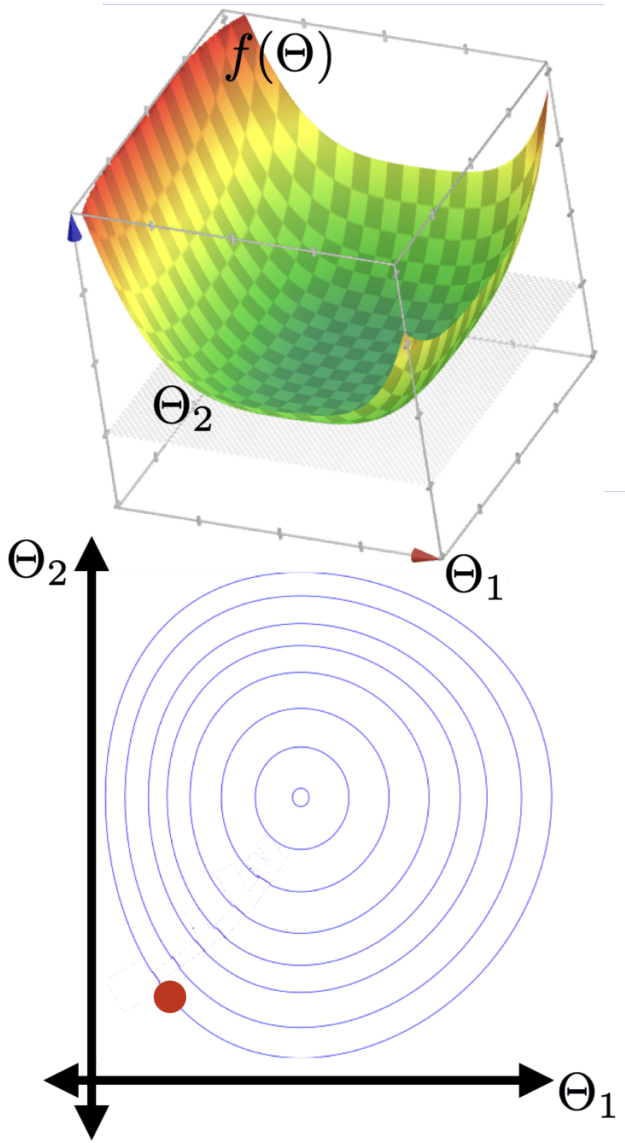
4 **repeat**

5 t = t + 1

6 $\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

7 **until** $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

8 **Return** $\Theta^{(t)}$



1 Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

2 Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

3 Initialize t = 0

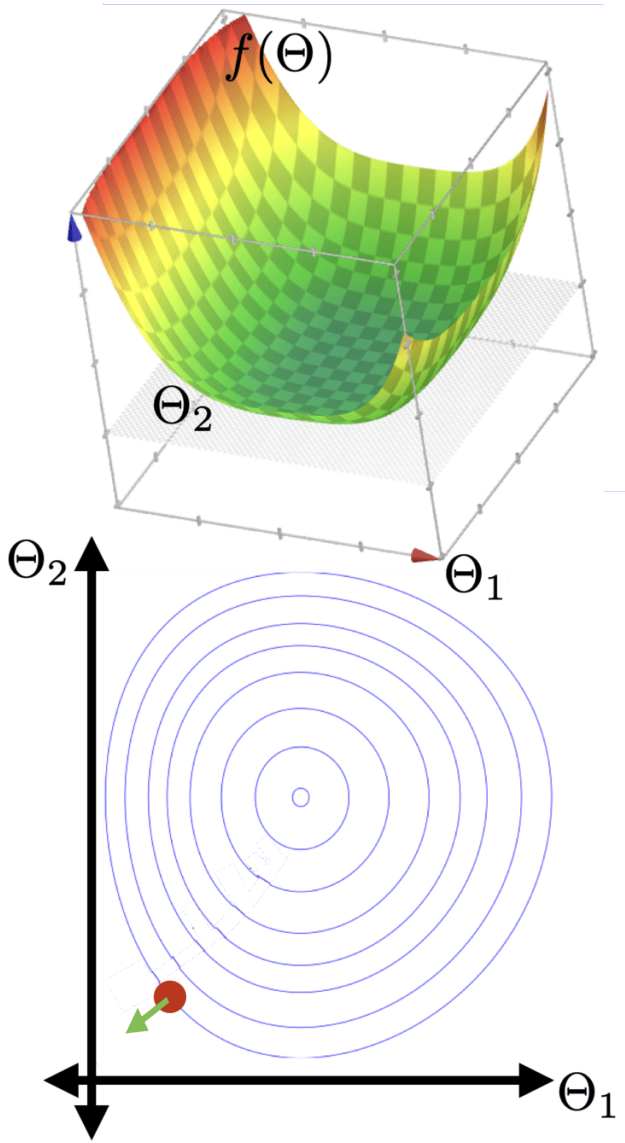
4 **repeat**

5 t = t + 1

6 $\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

7 **until** $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

8 **Return** $\Theta^{(t)}$



1 Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

2 Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

3 Initialize t = 0

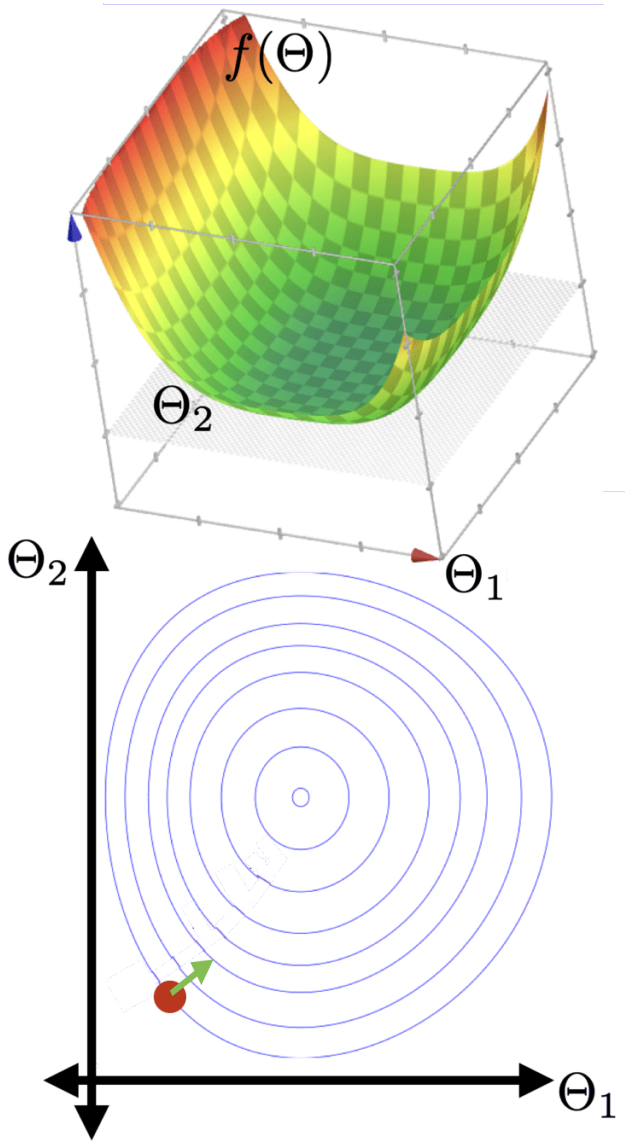
4 **repeat**

5 t = t + 1

6 $\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

7 **until** $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

8 **Return** $\Theta^{(t)}$



1 Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

2 Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

3 Initialize t = 0

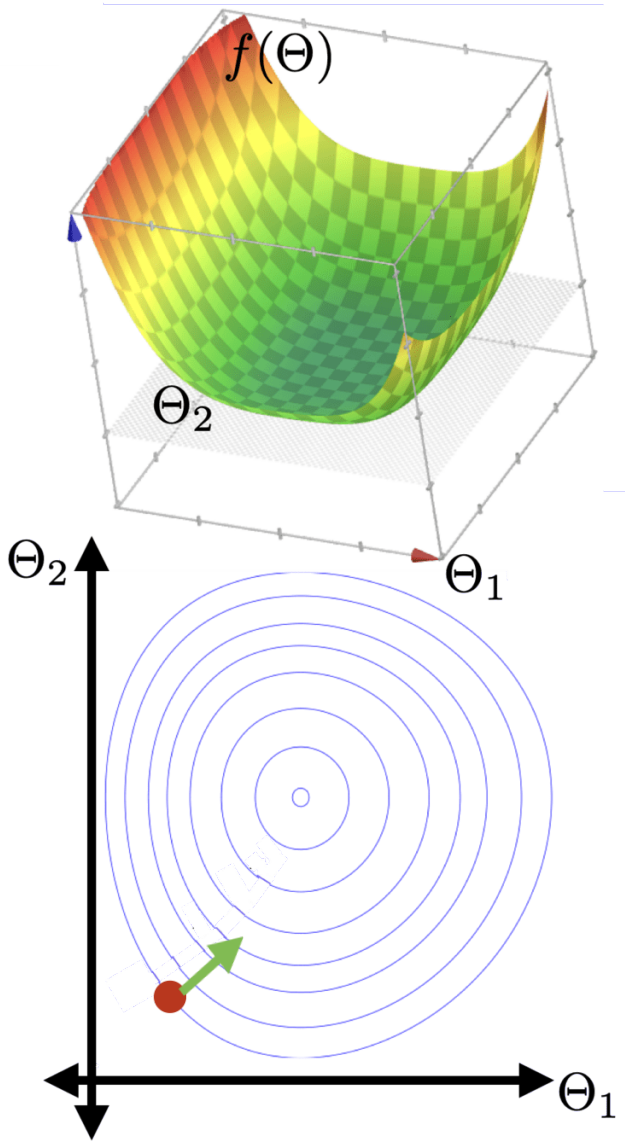
4 **repeat**

5 t = t + 1

6 $\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

7 **until** $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

8 **Return** $\Theta^{(t)}$



1 Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

2 Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

3 Initialize t = 0

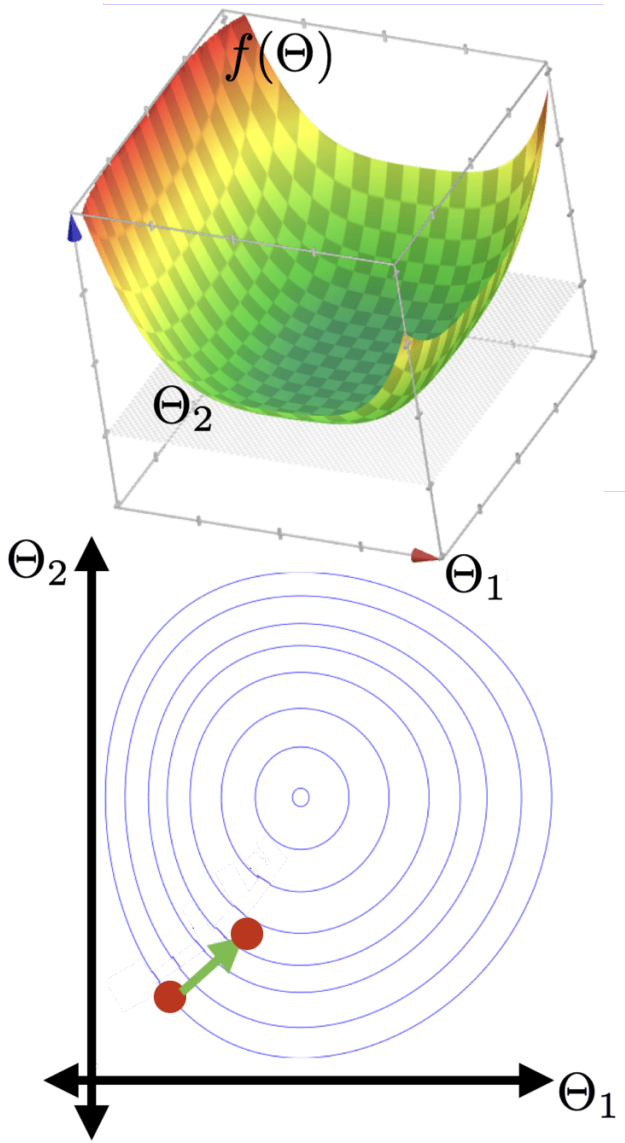
4 **repeat**

5 t = t + 1

6 $\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

7 **until** $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

8 **Return** $\Theta^{(t)}$



1 Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

2 Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

3 Initialize t = 0

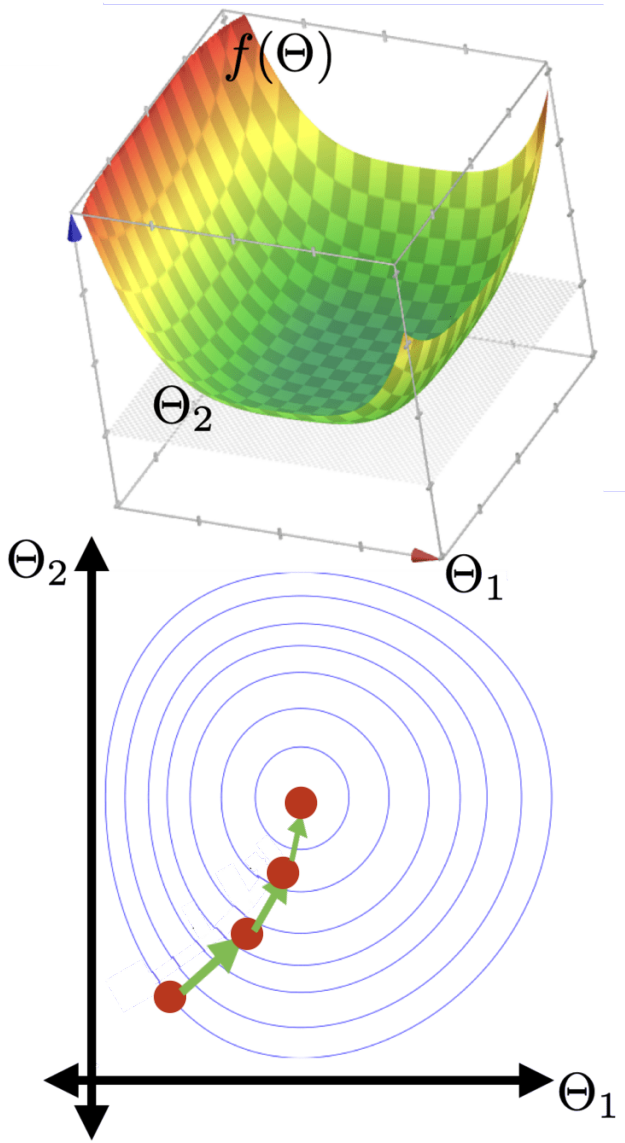
4 **repeat**

5 t = t + 1

6 $\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

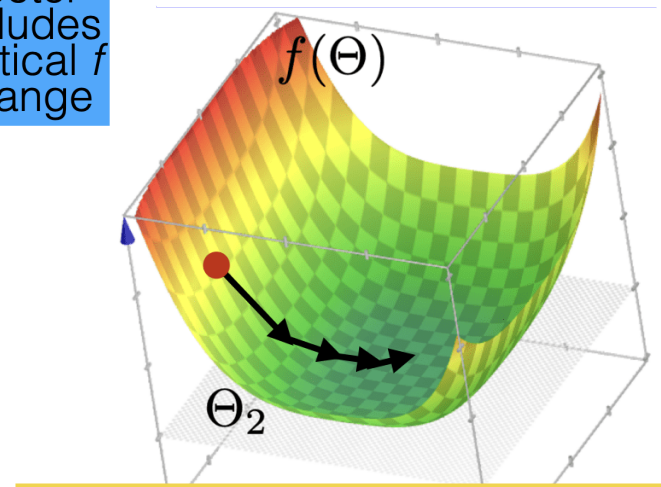
7 **until** $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

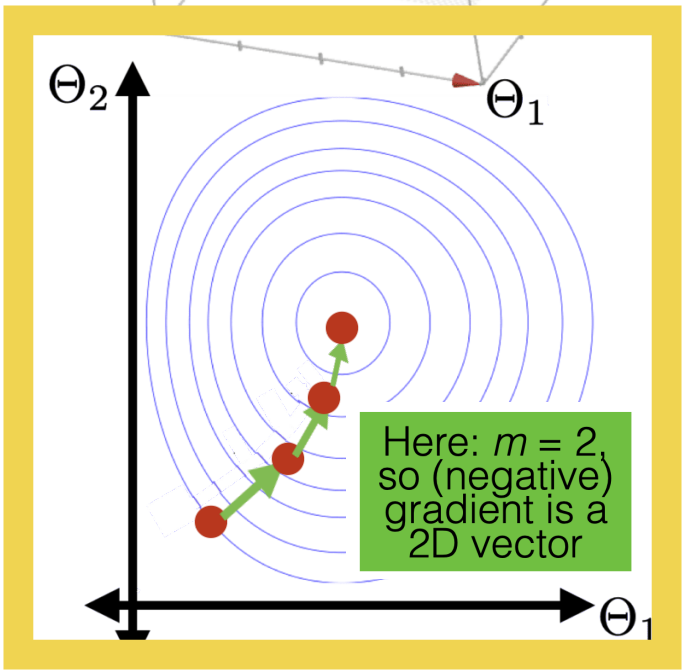
8 **Return** $\Theta^{(t)}$

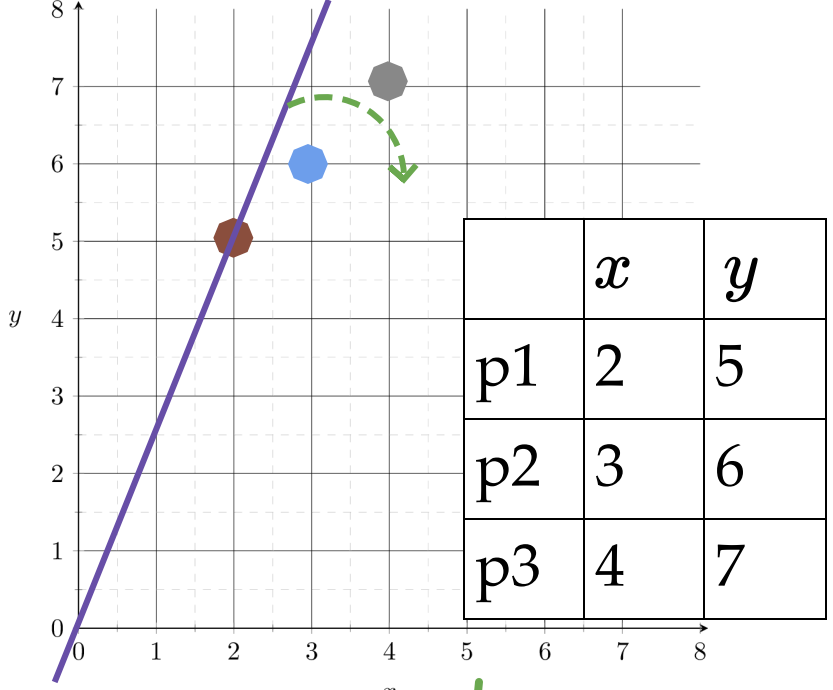


```
Gradient-Descent( Θ_init, η, f, ∇_Θ f, ε )
  Initialize  Θ^(0) = Θ_init
  Initialize t = 0
  repeat
    t = t + 1
    Θ^(t) = Θ^(t−1) − η ∇_Θ f(Θ^(t−1))
  until  |f(Θ^(t)) − f(Θ^(t−1))| < ε
  Return  Θ^(t)
```

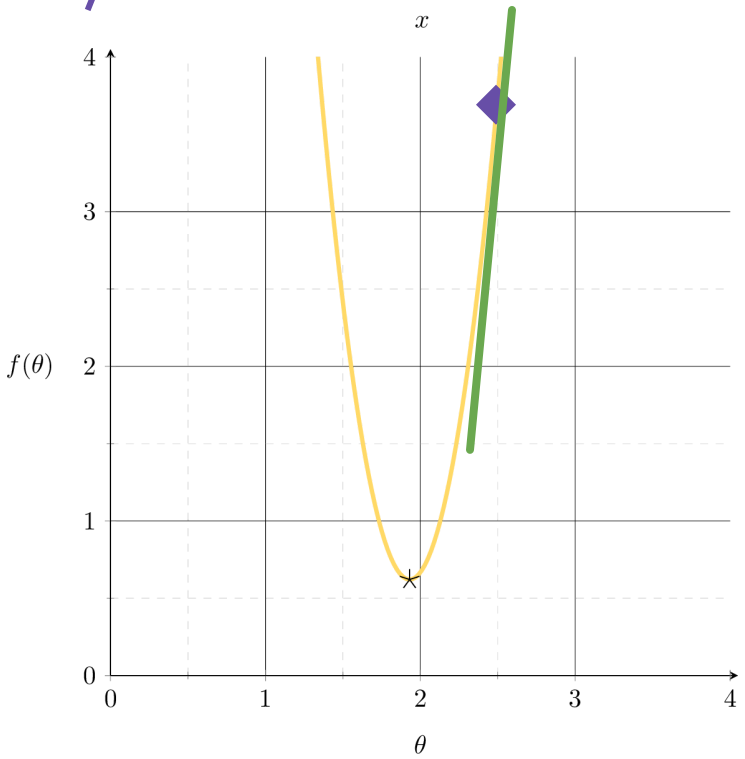
3D vector includes vertical *f* change







	x	y
p1	2	5
p2	3	6
p3	4	7



- the MSE of a linear hypothesis:

$$f(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

- and its gradient w.r.t. θ:

$$\nabla_\theta f = \frac{2}{3}\left[2(2\theta - 5) + 3(3\theta - 6) + 4(4\theta - 7)\right]$$

Gradient of an ML objective

Using our example data set,

- the MSE of a linear hypothesis:

$$f(\theta) = \tfrac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

- and its gradient w.r.t. θ:

$$\nabla_\theta f = \tfrac{2}{3}[2(2\theta - 5) + 3(3\theta - 6) + 4(4\theta - 7)]$$

Using any dataset,

- the MSE of a linear hypothesis:

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^\top x^{(i)} - y^{(i)}\right)^2$$

- and its gradient w.r.t. θ:

$$\nabla f(\theta) = \frac{2}{n}\sum_{i=1}^{n}\left(\theta^\top x^{(i)} - y^{(i)}\right)x^{(i)}$$

Gradient of an ML objective

- the MSE of a linear hypothesis:

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta^\top x^{(i)} - y^{(i)} \right)^2$$

- and its gradient w.r.t. θ:

$$\nabla f(\theta) = \frac{2}{n} \sum_{i=1}^{n} \left(\theta^\top x^{(i)} - y^{(i)} \right) x^{(i)}$$

In general,

- An ML objective function is a finite sum

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

- and its gradient w.r.t. θ:

$$\nabla f(\theta) = \nabla \left(\frac{1}{n} \sum_{i=1}^{n} f_i(\theta) \right) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta)$$

👆

👋 (gradient of the sum) = (sum of the gradient)

Gradient of an ML objective

In general,

- An ML objective function is a finite sum

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

- and its gradient w.r.t. θ:

$$\nabla f(\theta) = \nabla \left(\frac{1}{n} \sum_{i=1}^{n} f_i(\theta) \right) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta)$$

need to add n of them

each of these $\nabla f_i(\theta) \in \mathbb{R}^d$

Costly!

Let's do stochastic gradient descent (on the board).

Stochastic gradient descent

Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

Initialize t = 0

repeat

t = t + 1

$\Theta^{(t)} = \Theta^{(t-1)} - \eta \nabla_\Theta f(\Theta^{(t-1)})$

until $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

Return $\Theta^{(t)}$

Stochastic

Gradient-Descent($\Theta_{\text{init}}, \eta, f, \nabla_\Theta f, \epsilon$)

Initialize $\Theta^{(0)} = \Theta_{\text{init}}$

Initialize t = 0

repeat

t = t + 1

randomly select i from {1,…,n}

$\Theta^{(t)} = \Theta^{(t-1)} - \eta(t) \nabla_\Theta f_i(\Theta^{(t-1)})$

until $\left| f(\Theta^{(t)}) - f(\Theta^{(t-1)}) \right| < \epsilon$

Return $\Theta^{(t)}$

$$\nabla f(\Theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\Theta) \approx \nabla f_i(\Theta)$$

for a randomly picked data point i

Week 4 - Classification

- (Binary) linear classifier (sign based)
- (Binary) Logistic classifiers (sigmoid, NLL loss)
- Linear separator (the equation form <=> the visual form with normal vector)
- Linear separability (its interplay with features)
- How to handle multiple classes
  - Softmax generalization (Softmax, cross-entropy)
  - Multiple sigmoids

https://shenshen.mit.edu/demos/separator.html

	linear binary classifier	linear *logistic* binary classifier
features	$x \in \mathbb{R}^d$	
parameters	$\theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}$	
linear combination	$\theta^T x + \theta_0 = z$	
predict	$\begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} 1 & \text{if } \sigma(z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

	linear regressor	linear binary classifier	linear *logistic* binary classifier
features	$x \in \mathbb{R}^d$		
parameters	$\theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}$		
linear combo	$\theta^T x + \theta_0 = z$		
predict	z	$\begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} 1 & \text{if } g = \sigma(z) > 0.5 \\ 0 & \text{otherwise} \end{cases}$
loss	$(g - y)^2$	$\begin{cases} 0 & \text{if } g = a \\ 1 & \text{otherwise} \end{cases}$	$-\left[y \log g + (1 - y) \log (1 - g)\right]$
optimize via	closed-form or gradient descent	NP-hard to learn	• gradient descent only • need regularization to not overfit

	linear logistic *binary* classifier	one-out-of-K classifier
training data	$x \in \mathbb{R}^d$, $y \in \{0, 1\}$	$x \in \mathbb{R}^d$, y : K-dimensional one-hot
parameters	$\theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}$	$\theta \in \mathbb{R}^{d \times K}$, $\theta_0 \in \mathbb{R}^K$
linear combo	$\theta^T x + \theta_0 = z \in \mathbb{R}$	$\theta^T x + \theta_0 \ = z \in \mathbb{R}^K$
predict	$\sigma(z) = \dfrac{\exp(z)}{\exp(0) + \exp(z)}$ positive if $\sigma(z) > 0.5$	$\text{softmax}(z) = \begin{bmatrix} \exp(z_1) / \sum_i \exp(z_i) \\ \vdots \\ \exp(z_K) / \sum_i \exp(z_i) \end{bmatrix}$ category corresponding to the largest entry in softmax(z)

Negative log-likelihood $K-$ classes loss (aka, cross-entropy)

g : softmax output

g_k : probability or confidence in class k

$$\mathcal{L}_{\text{nllm}}(g, y) = -\sum_{k=1}^{K} y_k \cdot \log\left(g_k\right)$$

y :one-hot encoding label

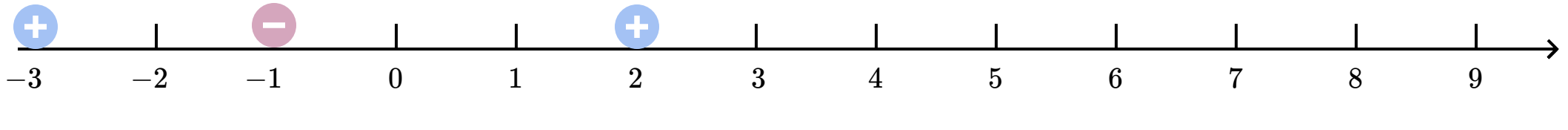
y_k : either 0 or 1

- Generalizes negative log likelihood loss $\mathcal{L}_{\text{nll}}(g, y) = -\left[y \log g + (1-y) \log\left(1-g\right)\right]$
- Appears as summing K terms, but
- for a given data point, only the term corresponding to its true class label matters.

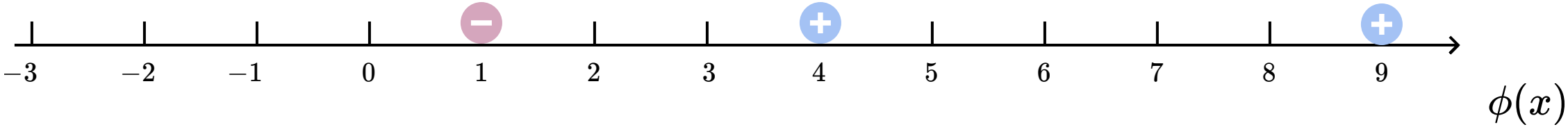
Week 5 - Features, Neural Networks I

- Feature transformations
  - Apply a fixed feature transformation
  - Hand-design feature transformation (e.g. towards getting linear separability)
  - Interplay between the number of features, the quality of features, and the quality of learning algorithms
- Forward-pass (for evaluation)

Not linearly separable in x space

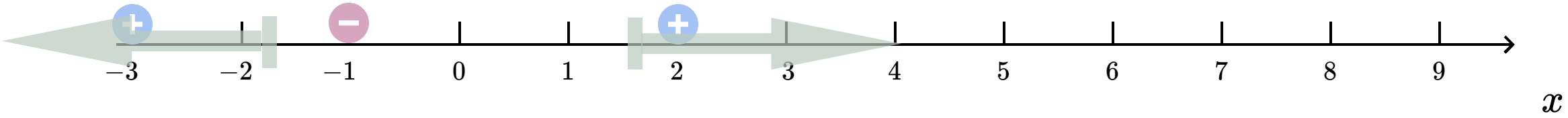


$\Downarrow$ transform via $\phi(x) = x^2$

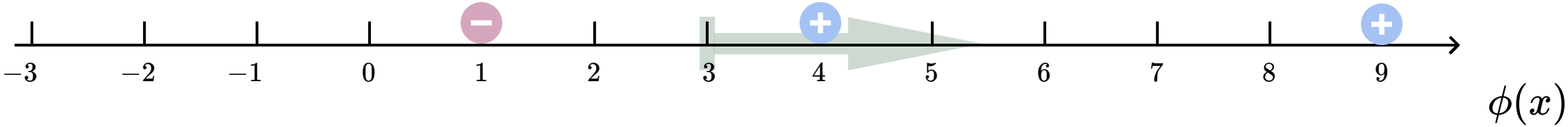


Linearly separable in $\phi(x) = x^2$ space

Non-linearly separated in x space, e.g. predict positive if $x^2 \geq 3$



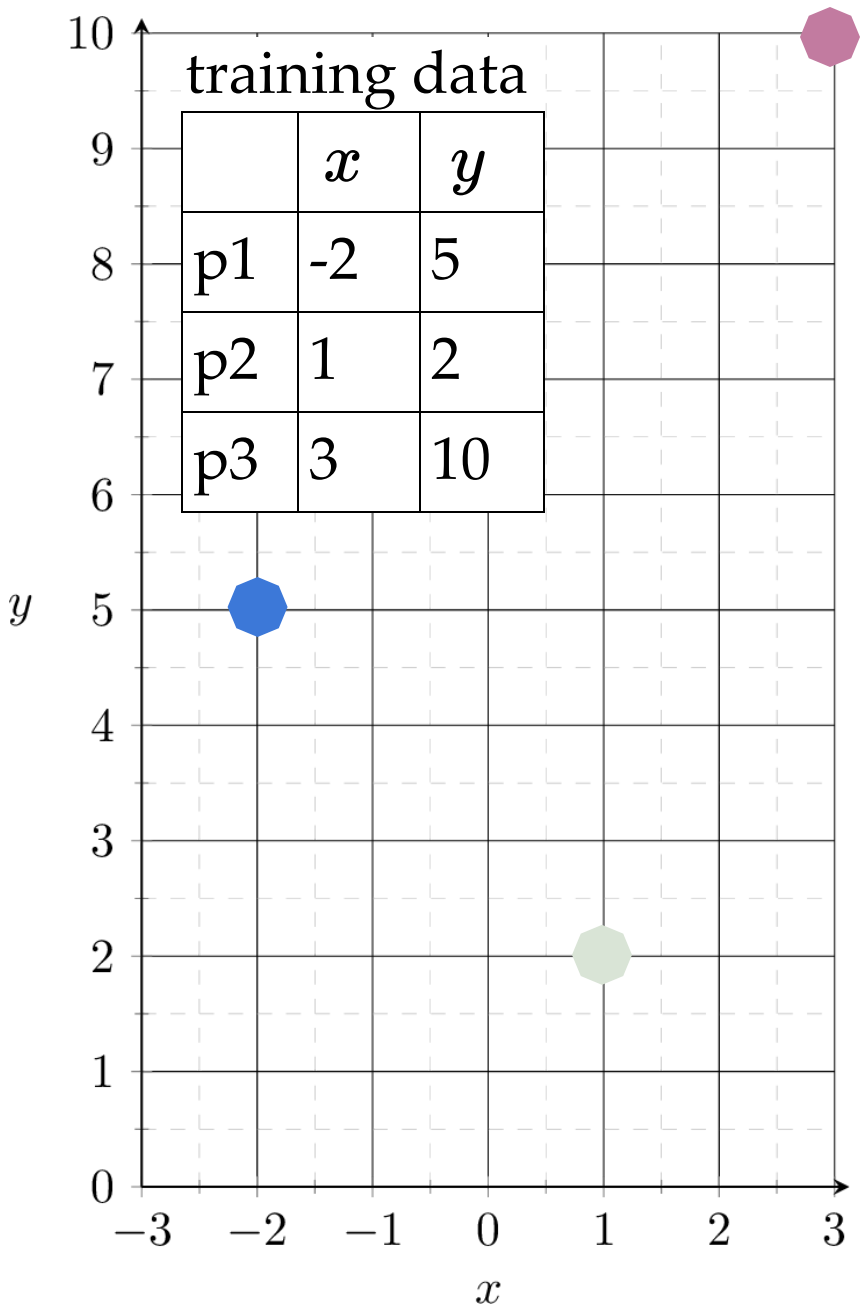
$\Downarrow$ transform via $\phi(x) = x^2$



Linearly separable in $\phi(x) = x^2$ space, e.g. predict positive if $\phi \geq 3$

training data

	x	y
p1	-2	5
p2	1	2
p3	3	10



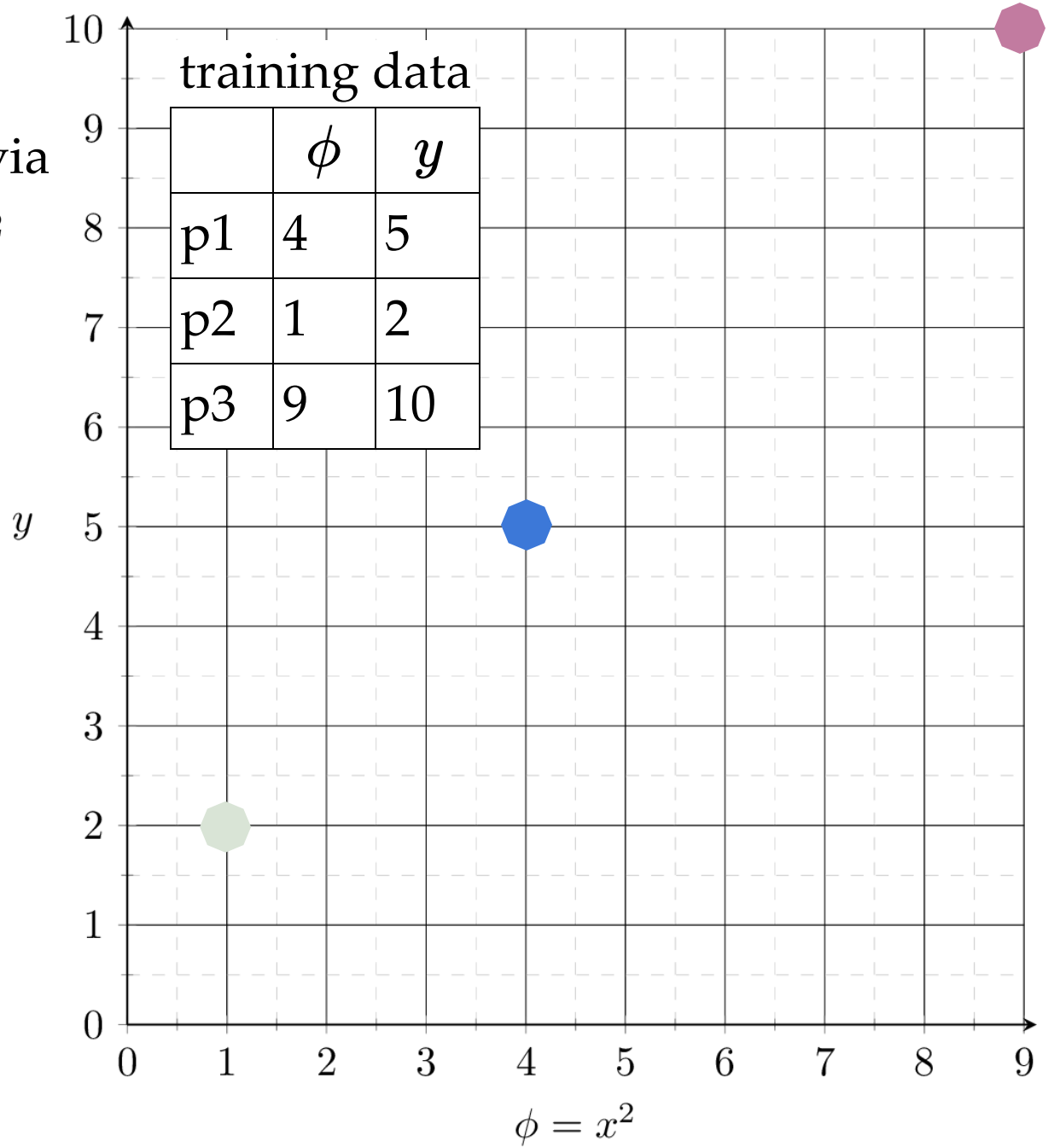
transform via

$$\phi(x) = x^2$$

$$\Rightarrow$$

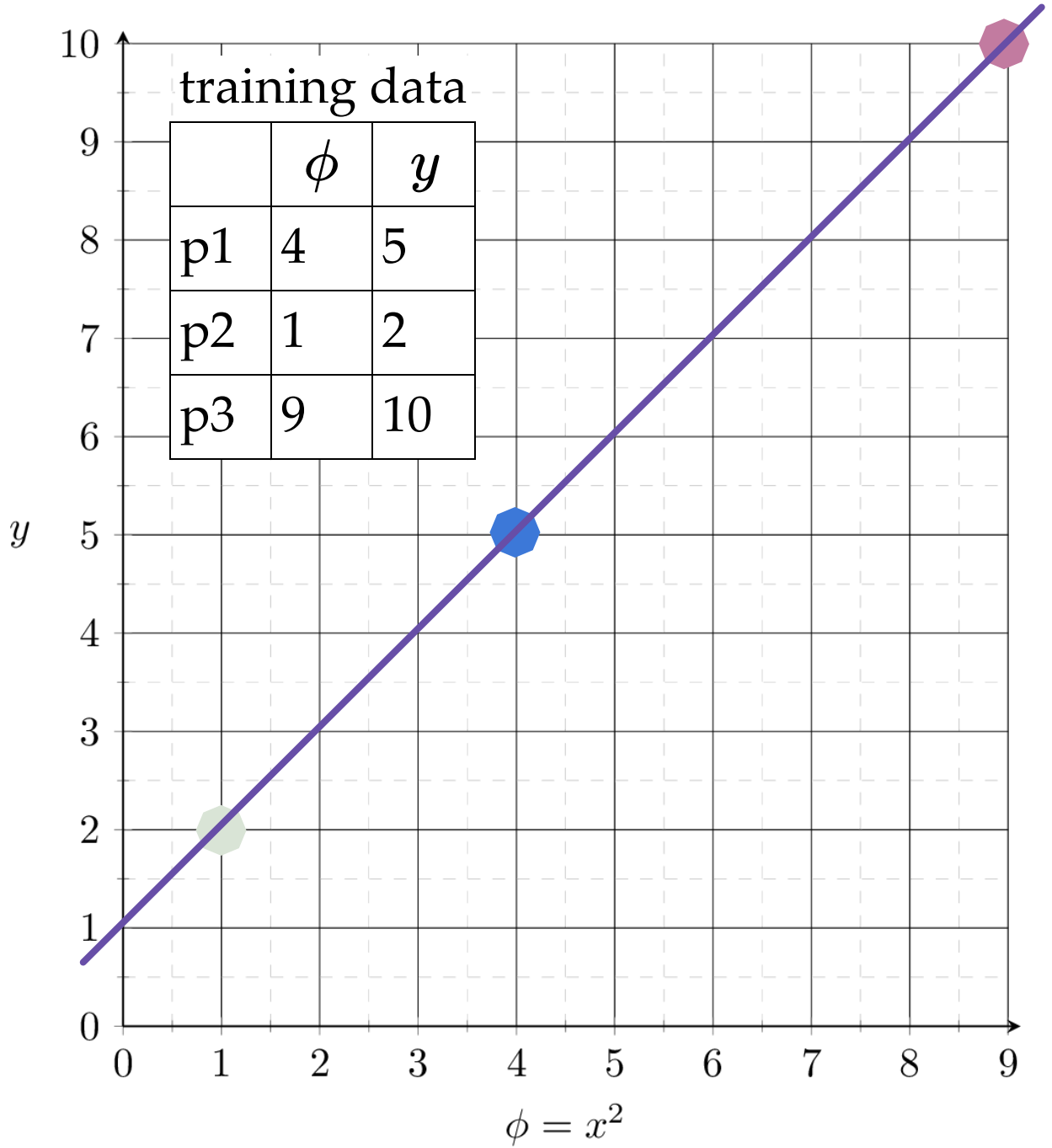
training data

	ϕ	y
p1	4	5
p2	1	2
p3	9	10



training data

	ϕ	y
p1	4	5
p2	1	2
p3	9	10



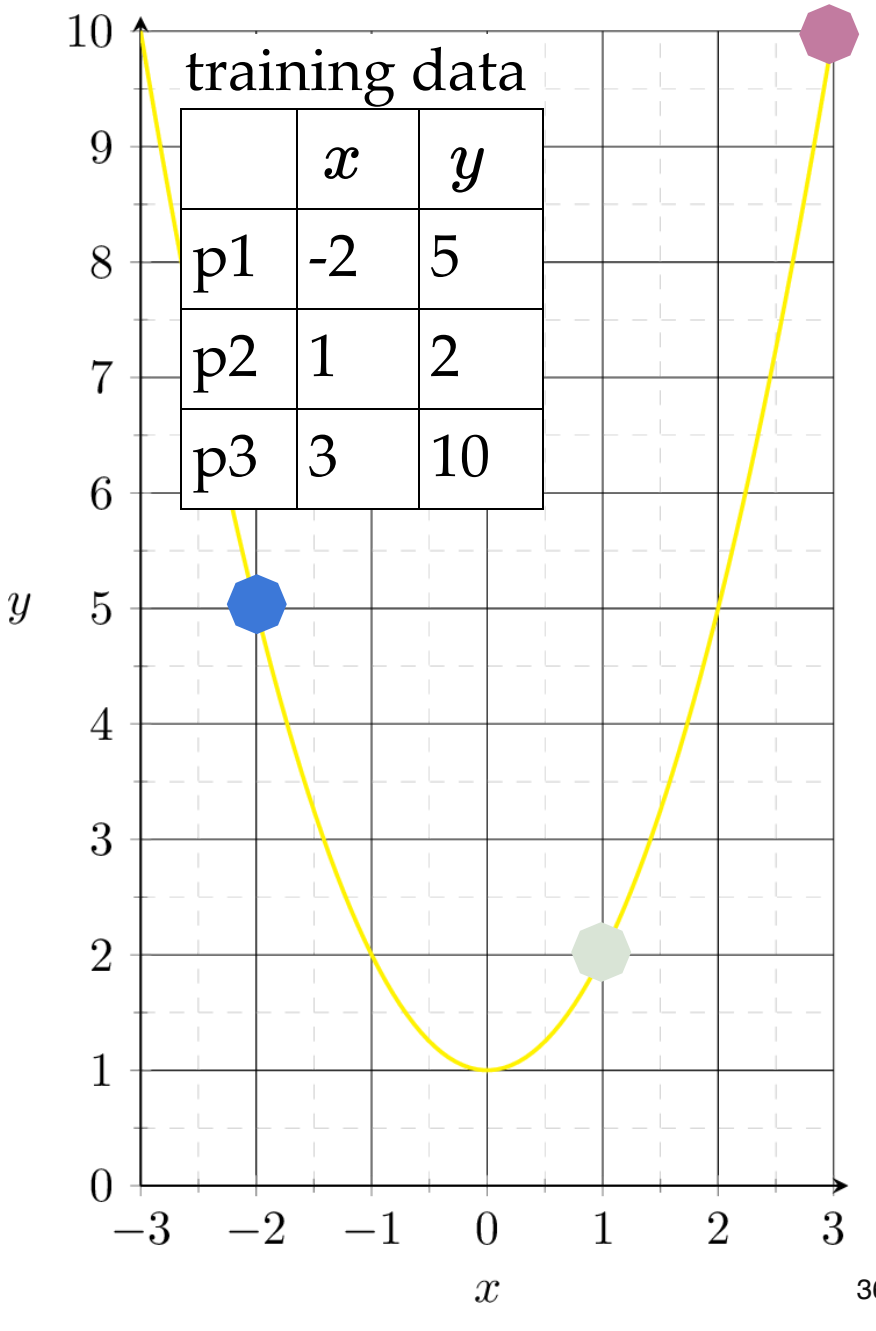
$$y = \phi + 1$$

$$= x^2 + 1$$

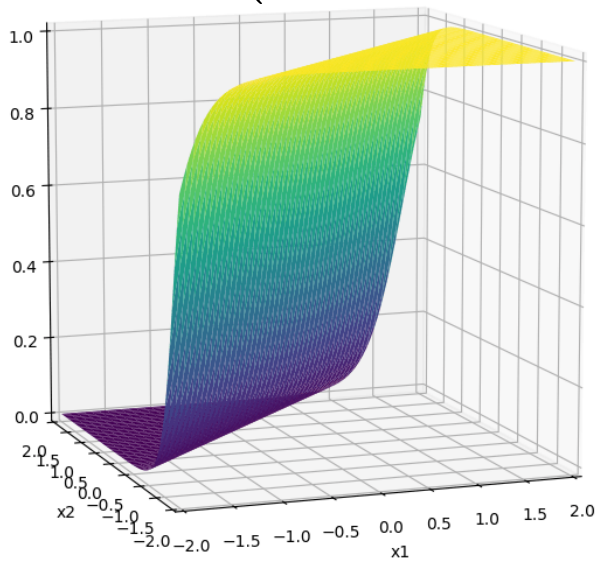
$$\Rightarrow$$

training data

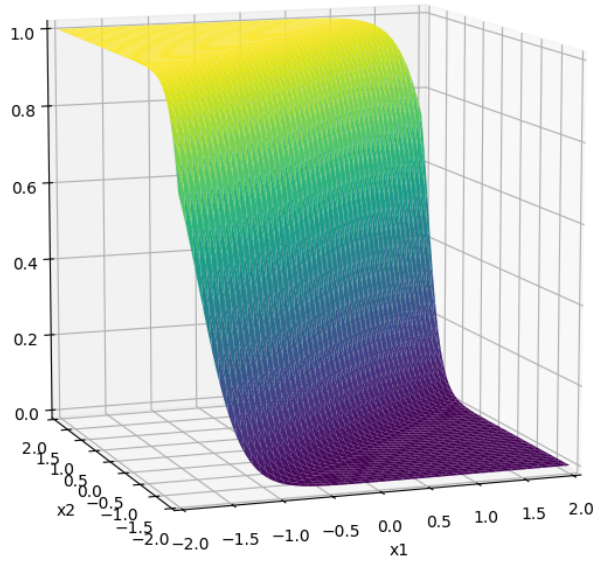
	x	y
p1	-2	5
p2	1	2
p3	3	10



$$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$$

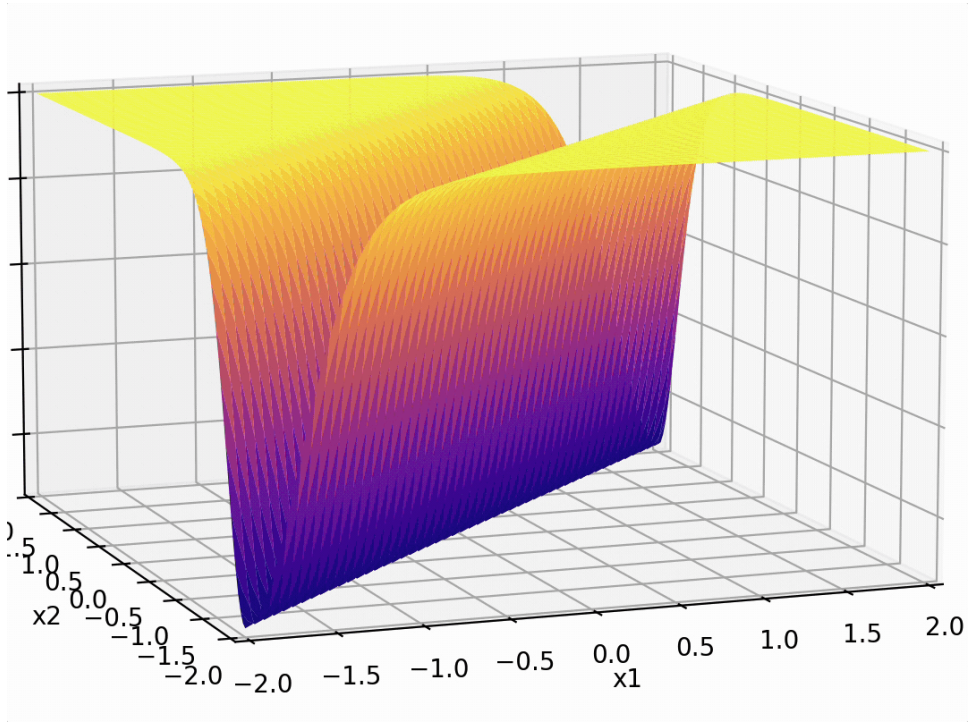


$$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$$



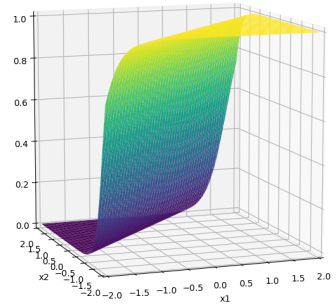
recall this example

$$-3(\sigma_1 + \sigma_2)$$



Recall

$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$



$f(\cdot)$ identity function

x_1 x_2 1

5 -5 1

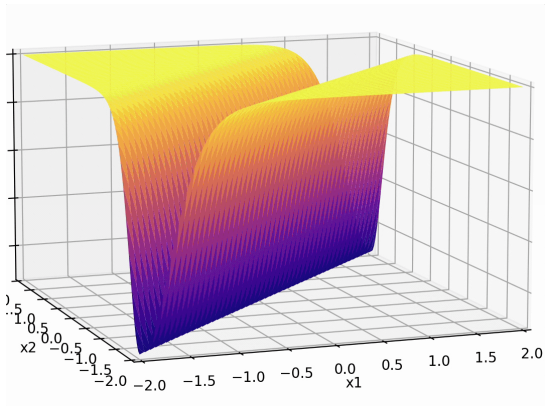
1 5 -5

W^1

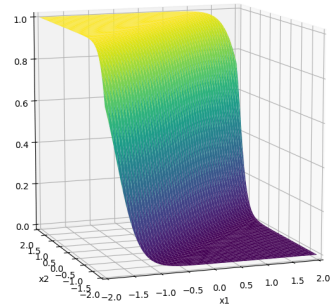
$f = \sigma(\cdot)$

-3 -3

W^2



$-3(\sigma_1 + \sigma_2)$



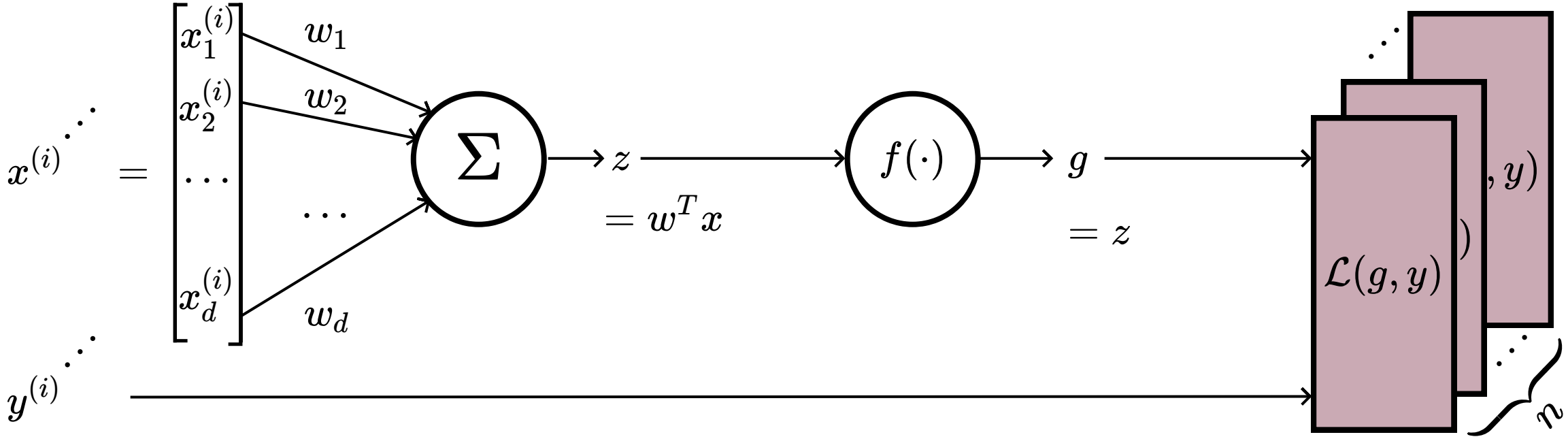
$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$

https://shenshen.mit.edu/demos/2layers.html

Week 6 - Neural Networks

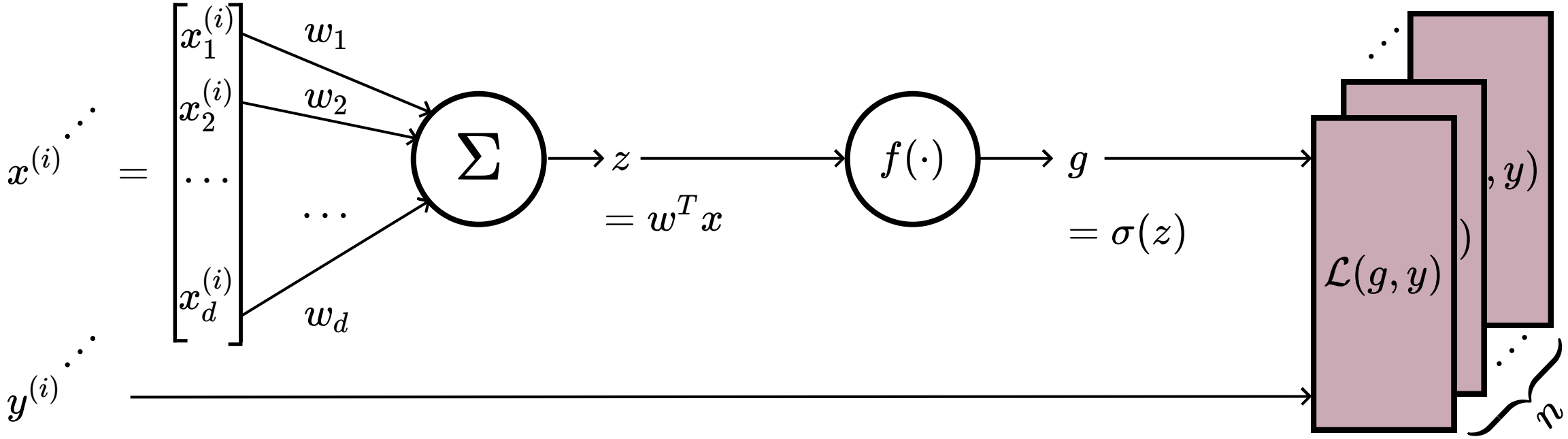
- Backward-pass (via backpropogation, for optimization)
- Source of expressiveness
- Output layer design
  - dimension, activation, loss
- Hand-designing weights
  - to match some given function form
  - achieve some goal (e.g. separate a given data set)

e.g. forward-pass of a linear regressor



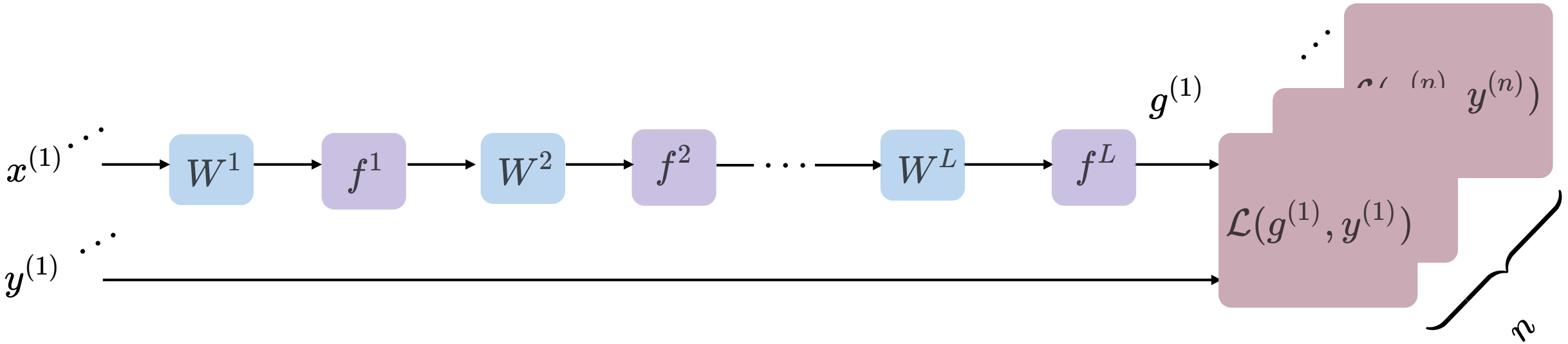
- Activation f is chosen as the identity function
- Evaluate the loss $\mathcal{L} = (g^{(i)} - y^{(i)})^2$
- Repeat for each data point, average the sum of n individual losses

e.g. forward-pass of a linear logistic classifier



- Activation f is chosen as the sigmoid function
- Evaluate the loss $\mathcal{L} = -[y^{(i)} \log g^{(i)} + \left(1 - y^{(i)}\right) \log \left(1 - g^{(i)}\right)]$
- Repeat for each data point, average the sum of n individual losses

Forward pass: evaluate, *given* the current parameters



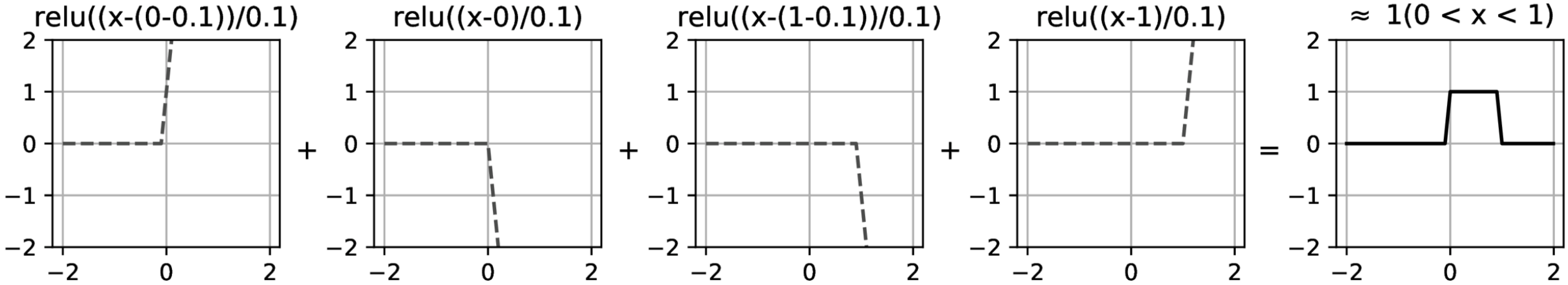
- the model outputs $g^{(i)} = f^L\left(\ldots f^2\left(\ f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2\right); \ldots \mathbf{W}^L\right)$
- the loss incurred on the current data $\mathcal{L}(g^{(i)}, y^{(i)})$
- the training error $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(g^{(i)}, y^{(i)})$

linear combination

(nonlinear) activation

loss function

`Recall:` compositions of ReLU(s) can be quite expressive



in fact, asymptotically, can approximate any function!

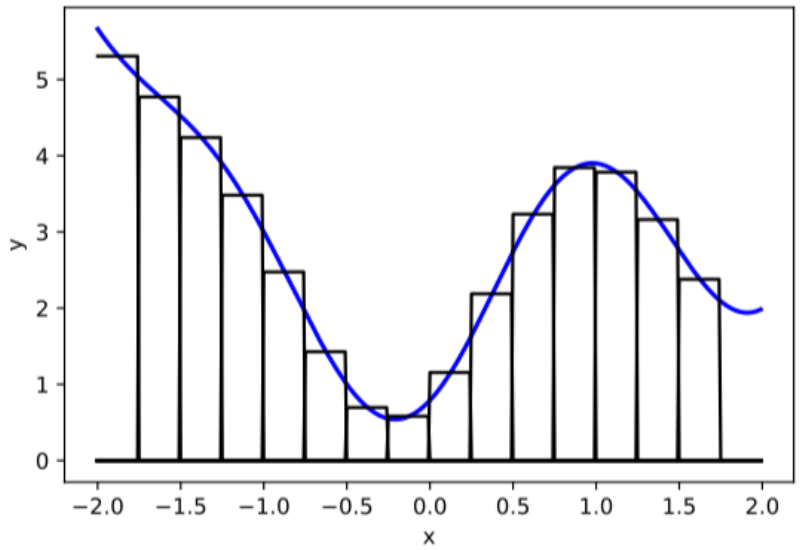


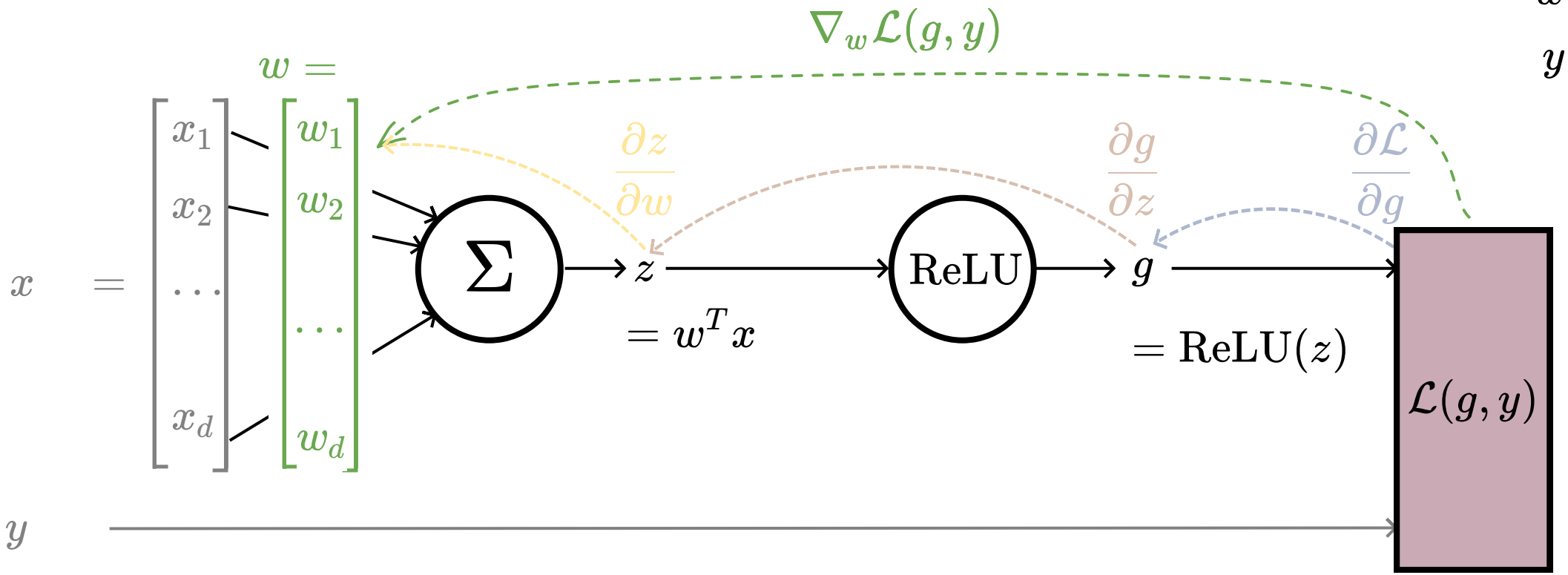
image credit: Phillip Isola

now, slightly more interesting activation:

$x \in \mathbb{R}^d$

$w \in \mathbb{R}^d$

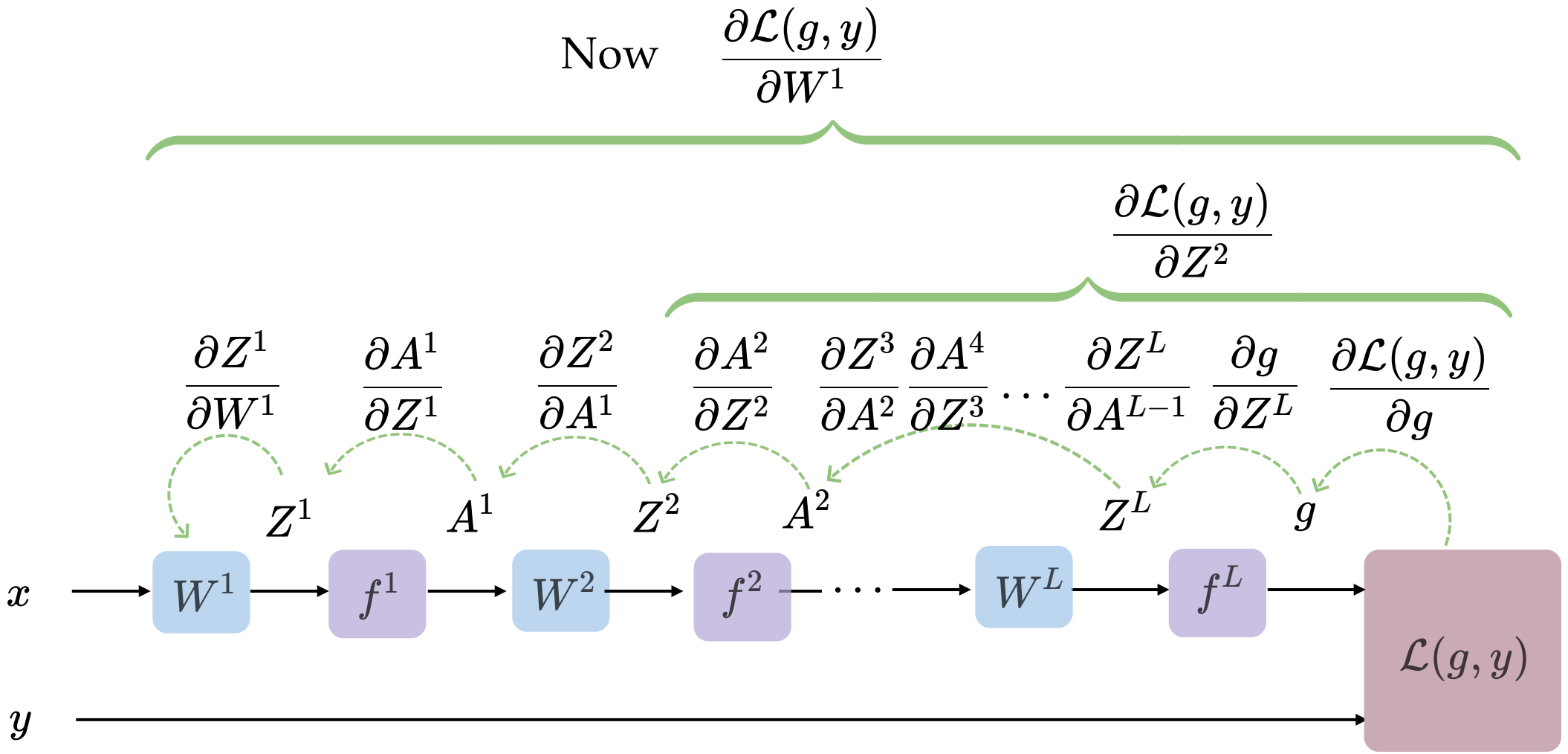
$y \in \mathbb{R}$



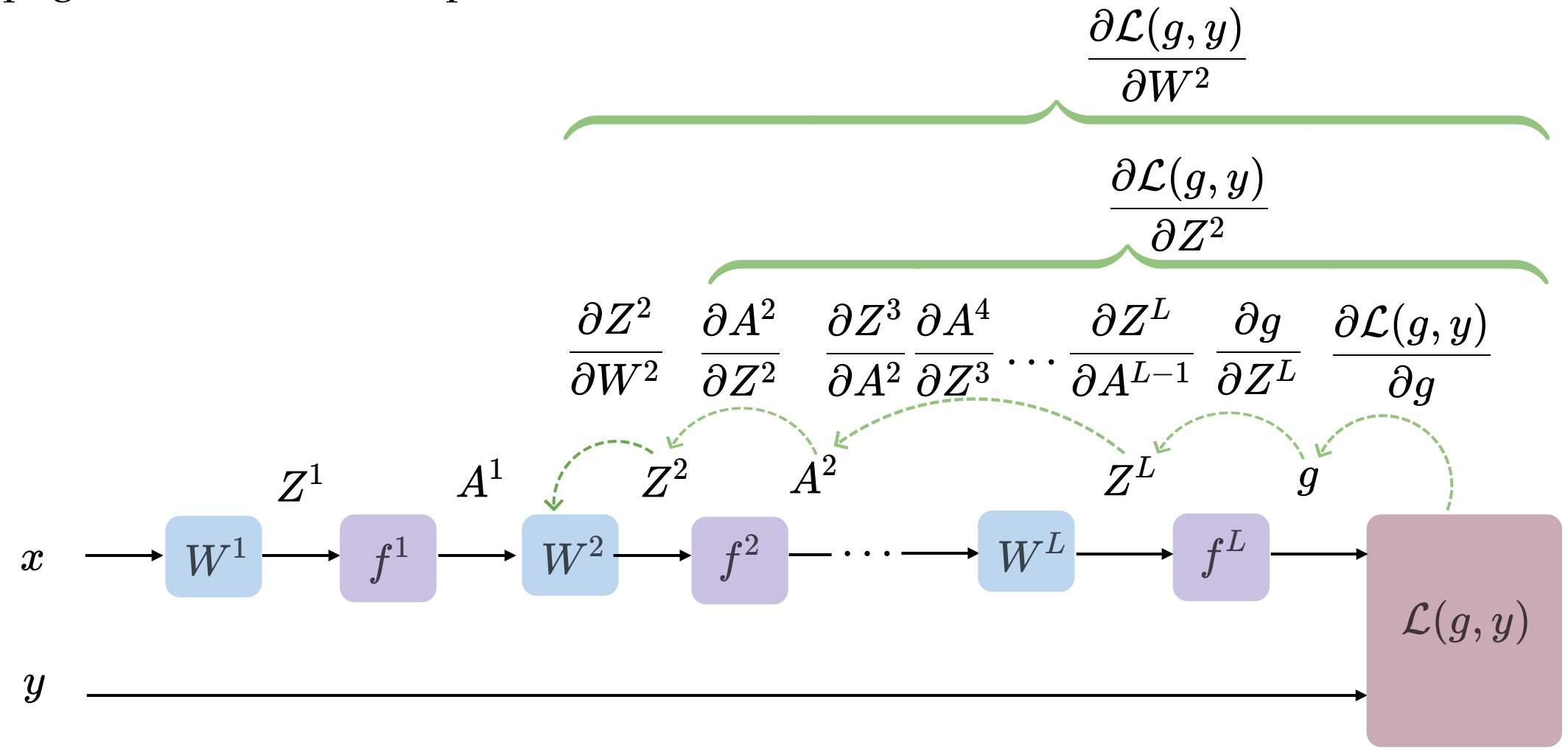
$$\nabla_w \mathcal{L}(g, y) = \frac{\partial \mathcal{L}(g, y)}{\partial w} = \frac{\partial [(g - y)^2]}{\partial w} = x \cdot \frac{\partial [(\text{ReLU}(z))]}{\partial z} \cdot 2(g - y)$$

example on black-board

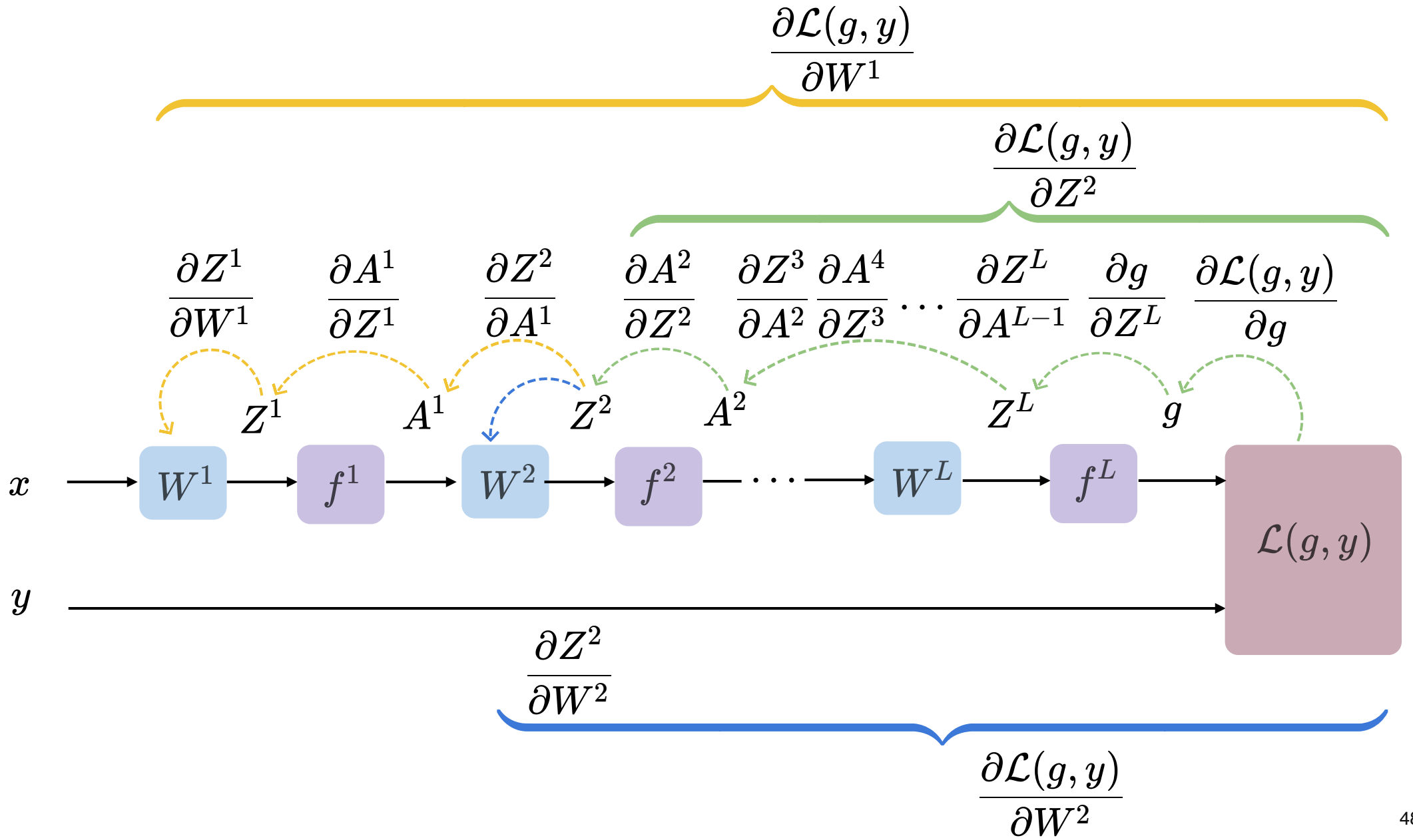
how to find $\frac{\partial \mathcal{L}(g,y)}{\partial W^1}$



back propagation: reuse of computation



back propagation: reuse of computation

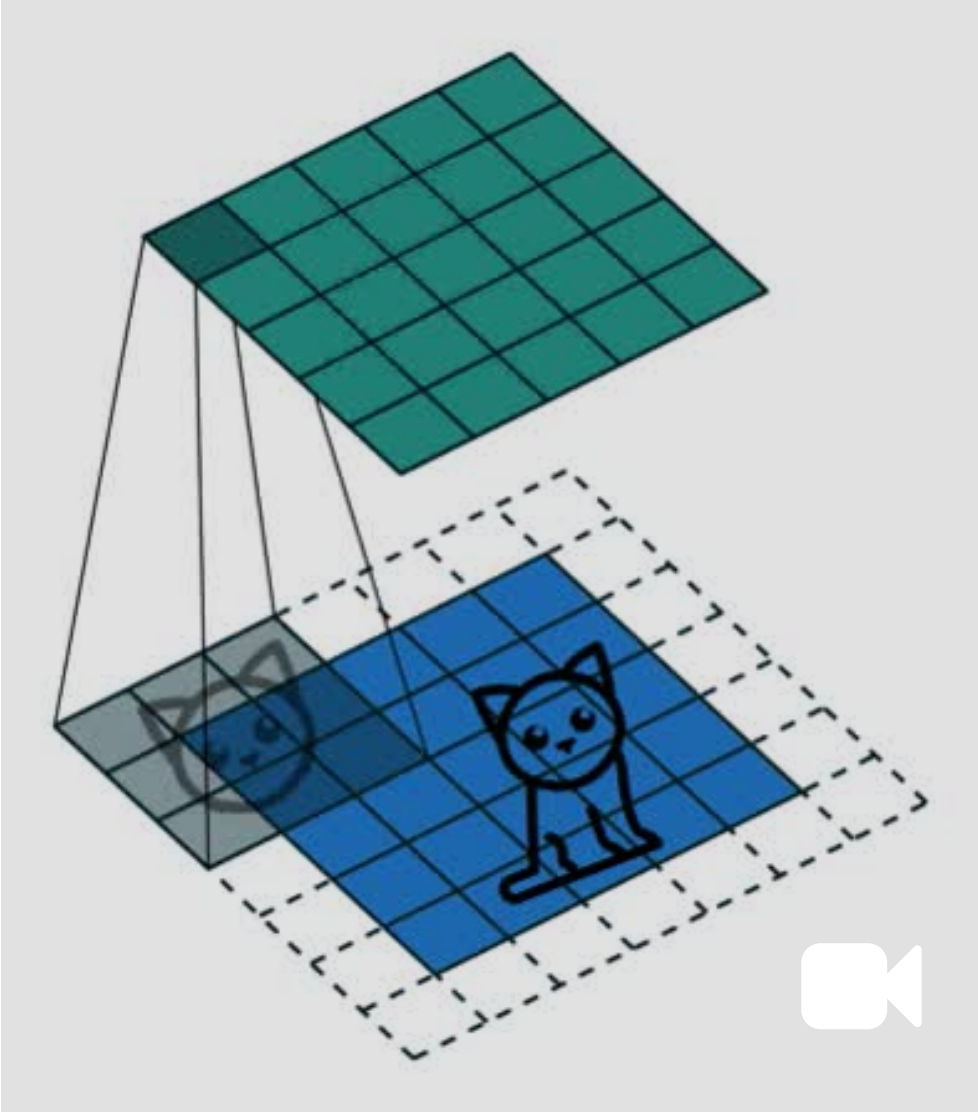


Week 7 - CNN

- Forward pass: convolution operation; max-pooling and the typical "pyramid" stack.
- Backward pass: back-propagation to learn filter weights/bias.
- The convolution/max-pooling operation
  - various hyper-parameters (filter size, padding size, stride) in spatial dimension;
  - the 3rd channel/depth dimension
  - reason about in/out shapes
- Conceptually: weight sharing, "pattern matching" template, independent and parallel processing.

convolution interpretation:

- Looking locally
- Parameter sharing
- Template matching
- Translational equivariance

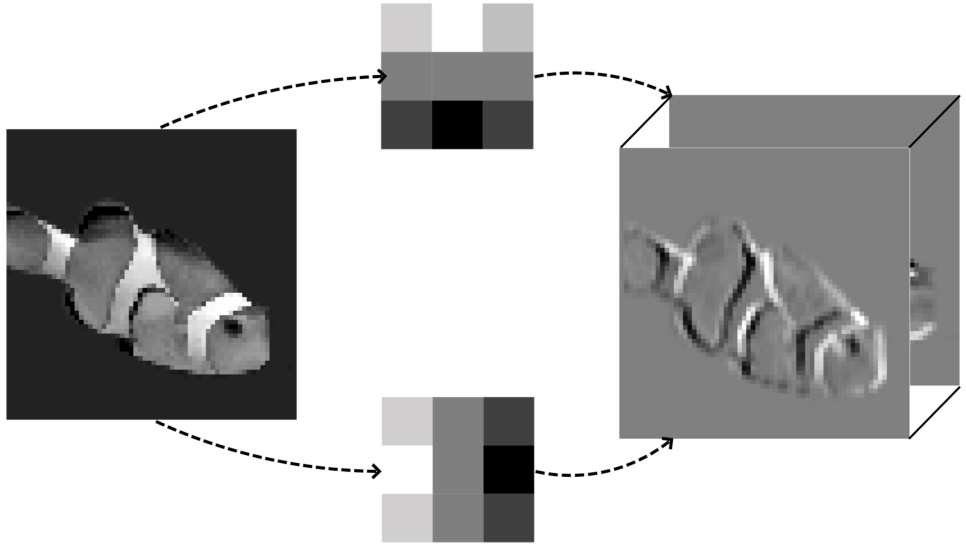


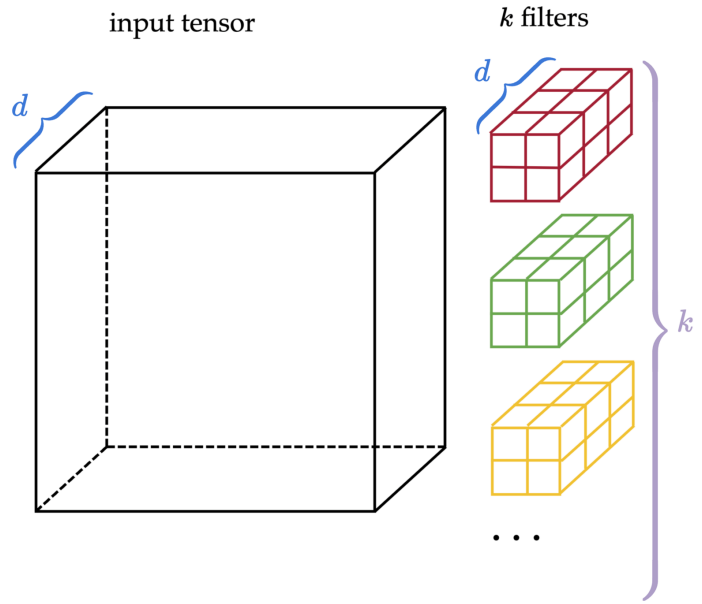
[video credit Lena Voita]

We'd encounter 3d tensor due to:

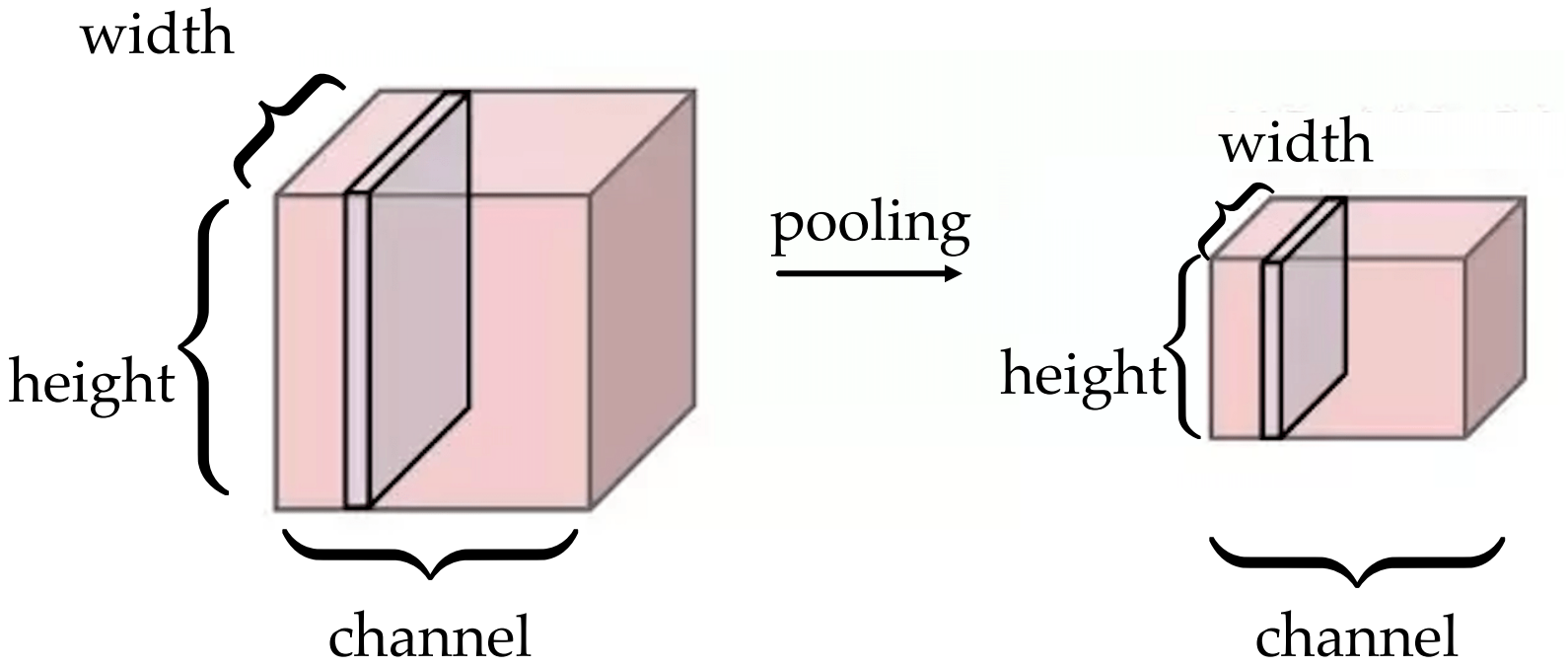
1. color input

2. the use of multiple filters -- in doing 2-dimensional convolution





Pooling across *spatial* locations achieves invariance w.r.t. small translations:



so the *channel* dimension remains *unchanged* after pooling.

Week 8 - Representation Learning

- Unsupervised learning setup
- Auto-encoder: compression and reconstruction
- The concept of embedding space:
  - Similarity comparison
  - vector arithmetic

$$g = \text{softmax}(z_2)$$

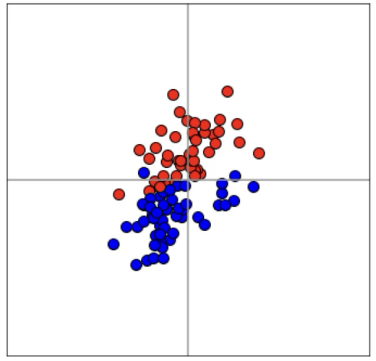
$$z_2 = \text{linear}\,(a_1)$$

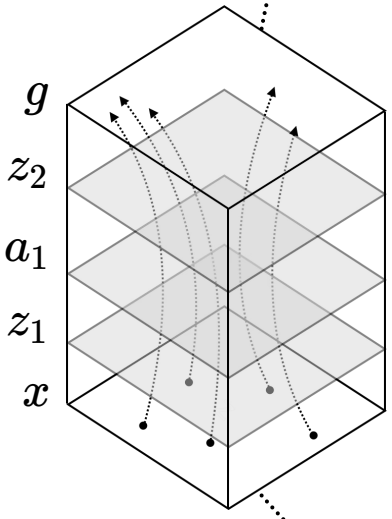
$$a_1 = \text{ReLU}(z_1)$$

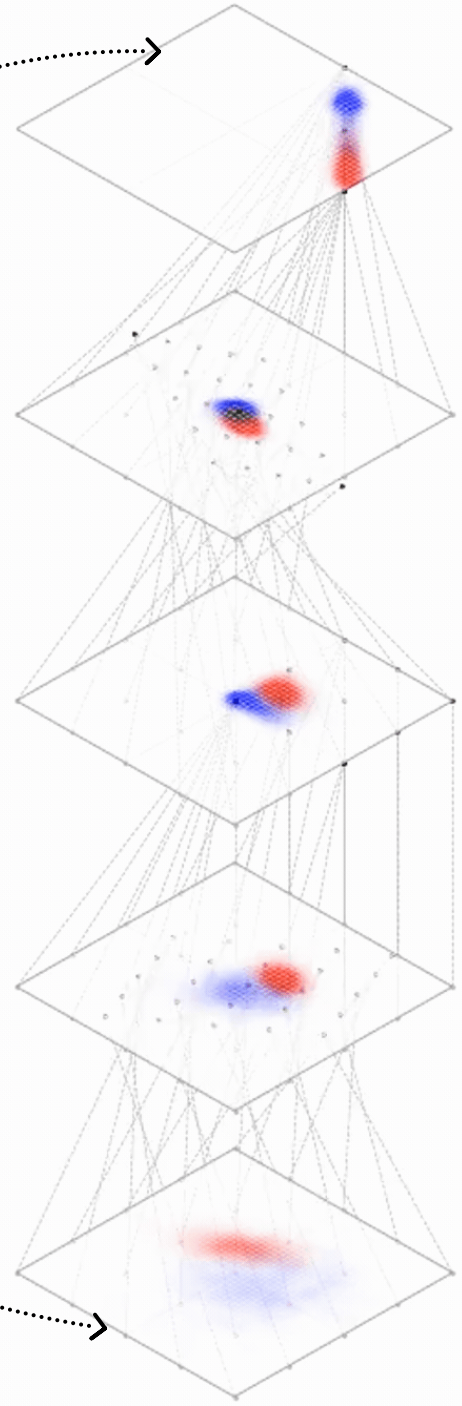
$$z_1 = \text{linear}\,(x)$$

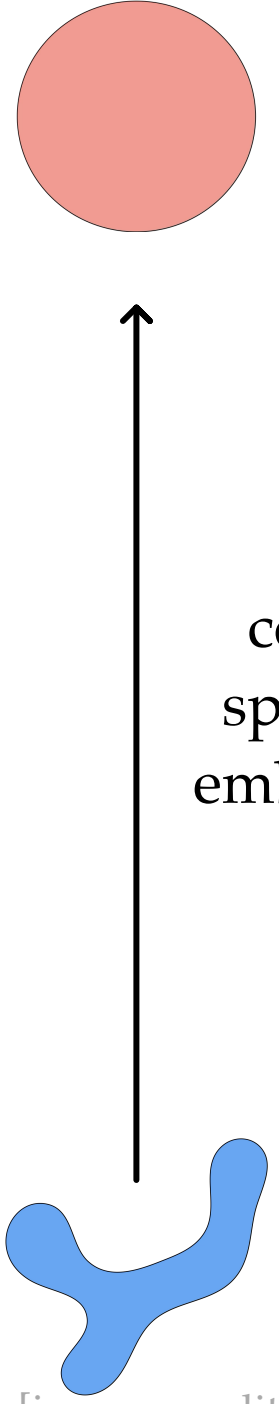
$$x \in \mathbb{R}^2$$

Training data









maps from complex data space to simple embedding space

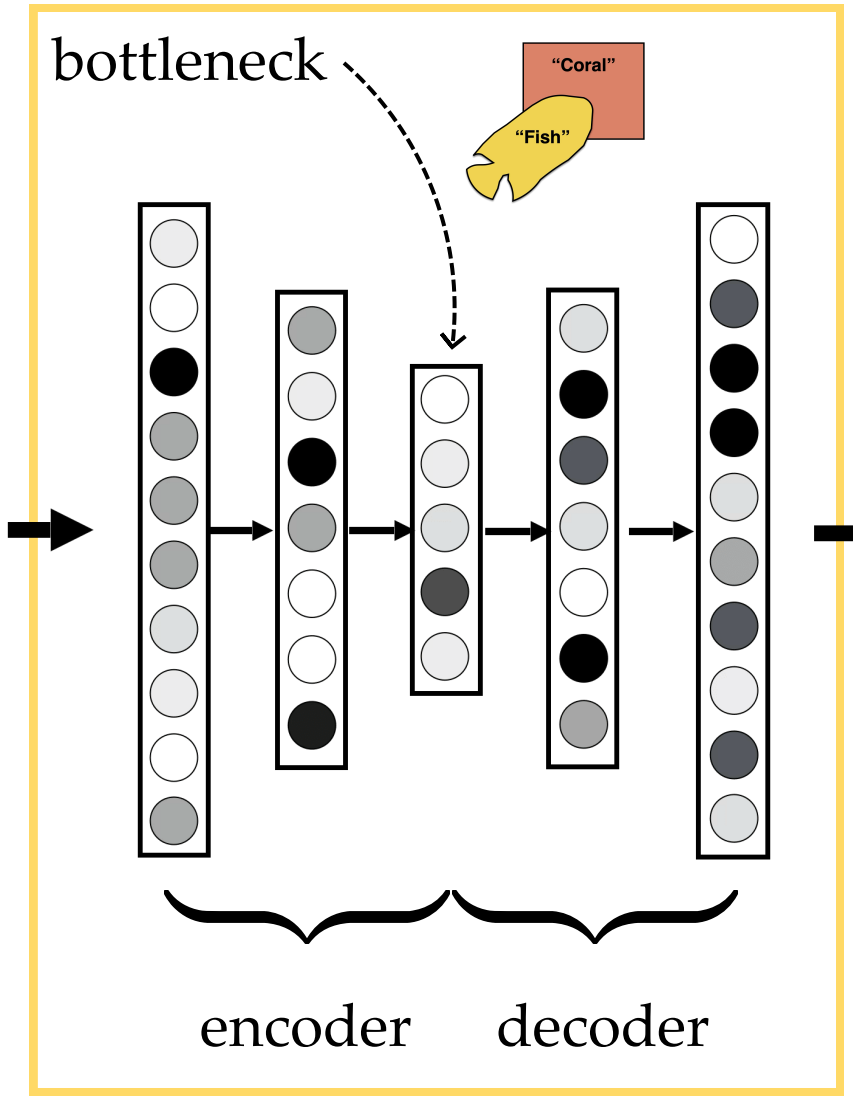
[images credit: visionbook.mit.edu]

Auto-encoder

$$\min_W ||x - \tilde{x}||^2$$

x

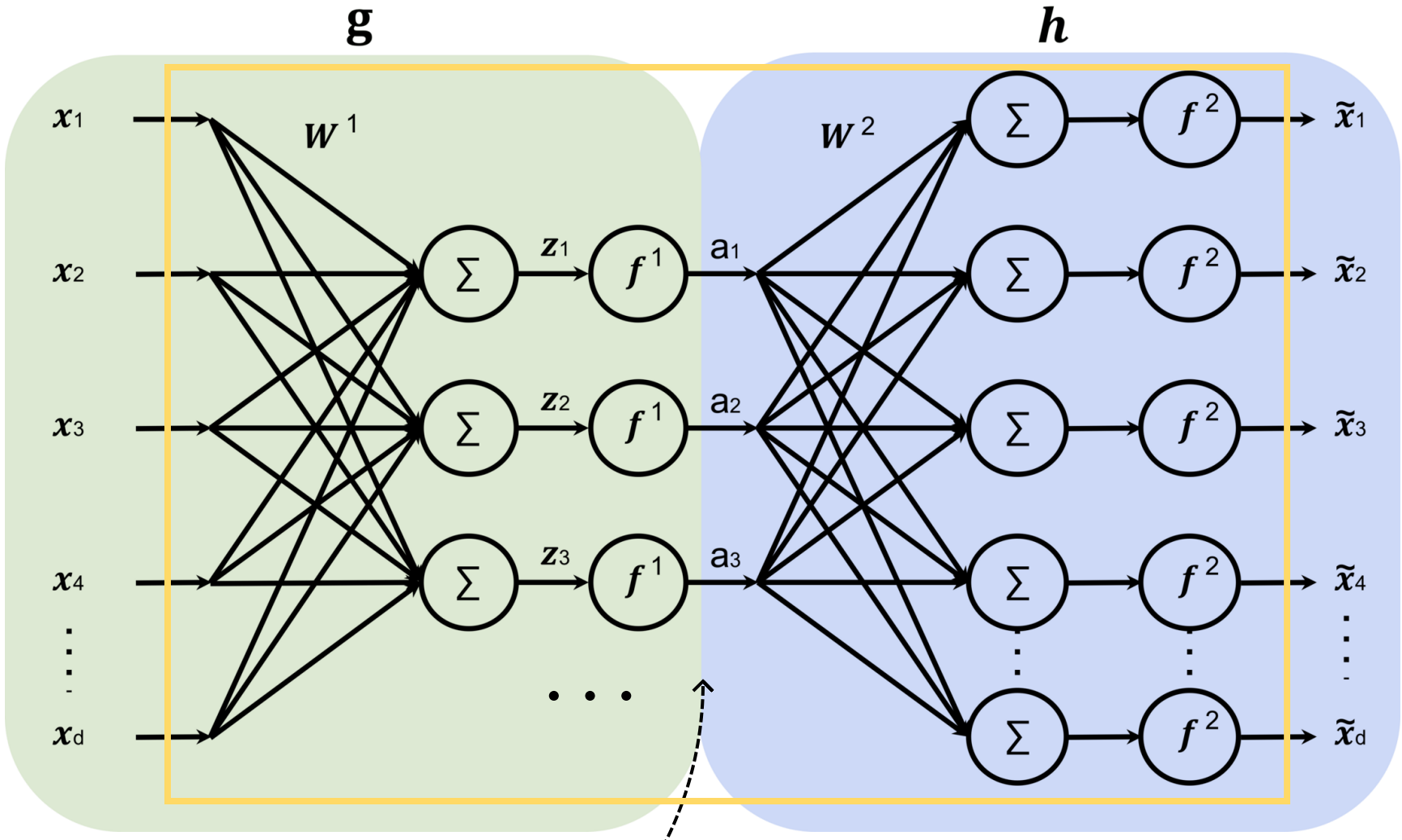
Image



$\tilde{x} = \text{NN}(x; W)$

Reconstructed image

input
$x \in \mathbb{R}^d$



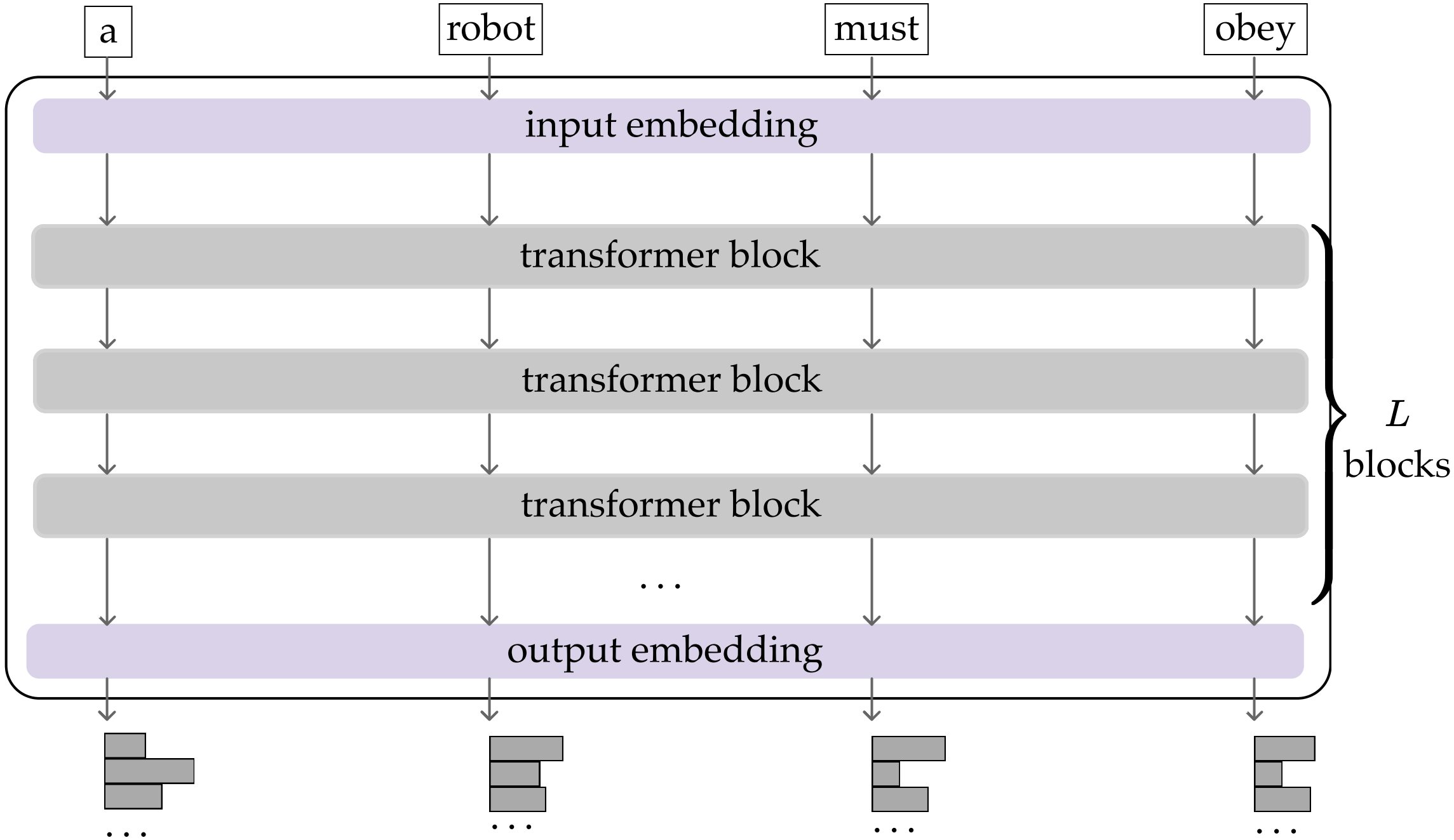
output
$\tilde{x} \in \mathbb{R}^d$

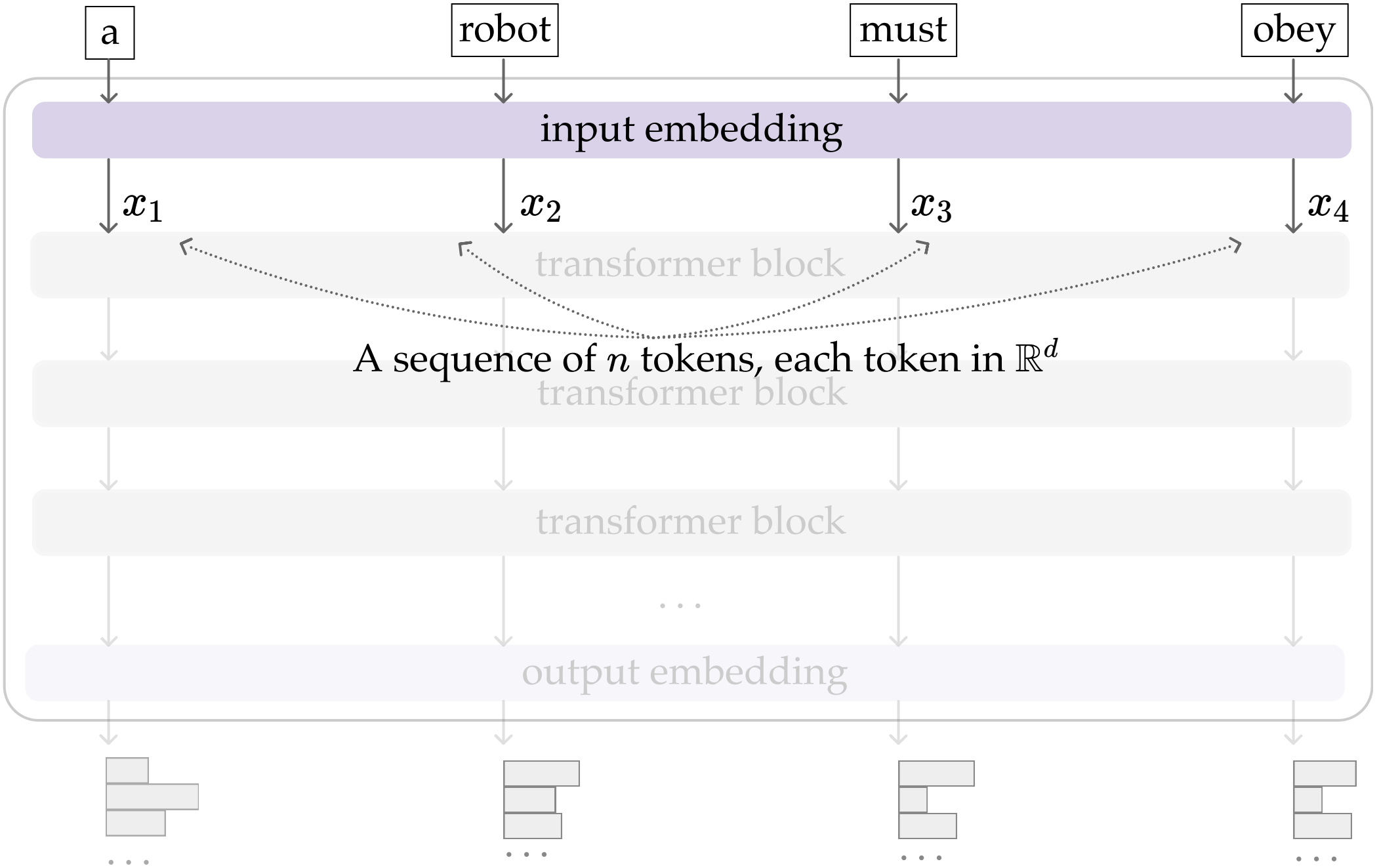
bottleneck

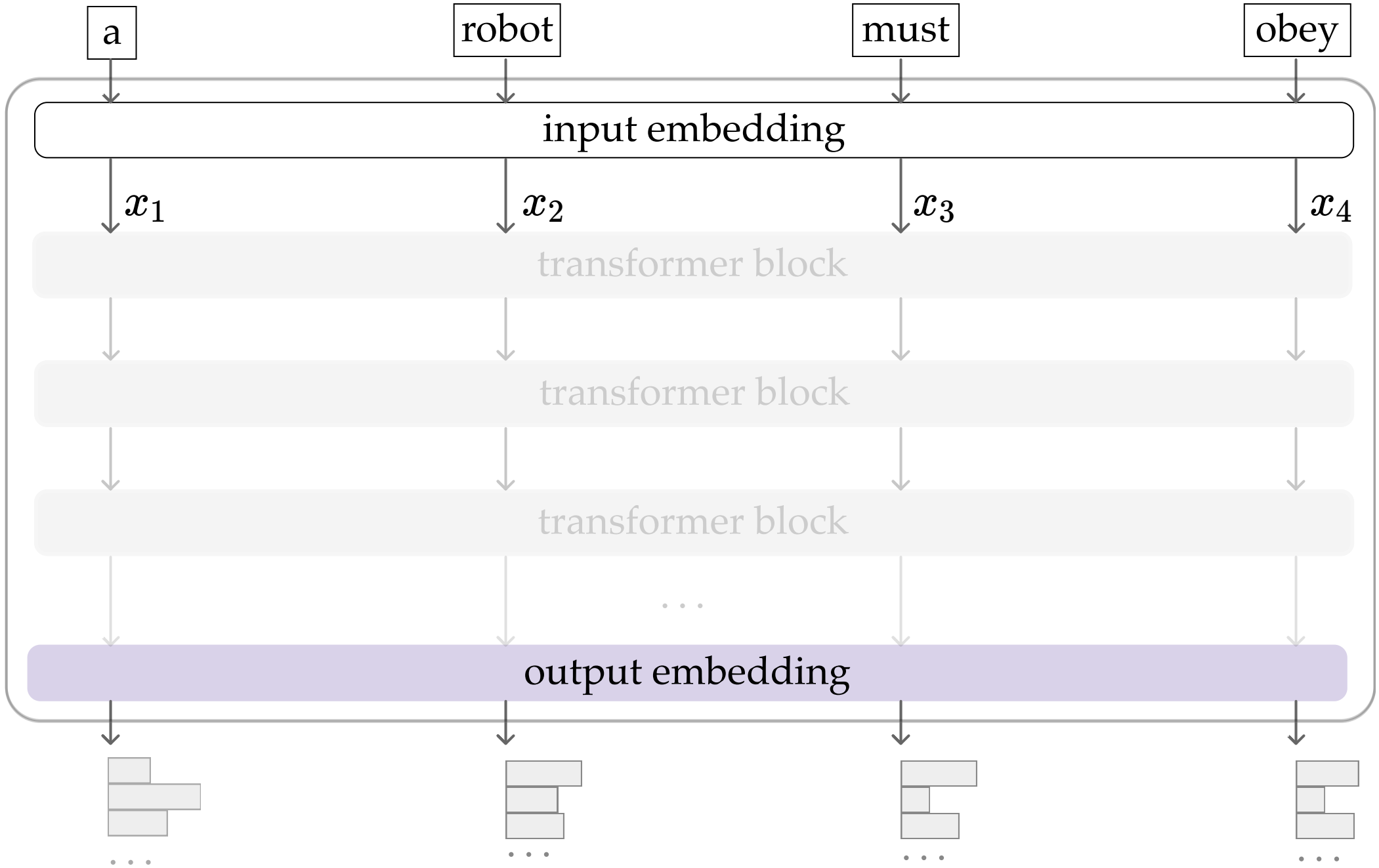
typically, has lower dimension than d

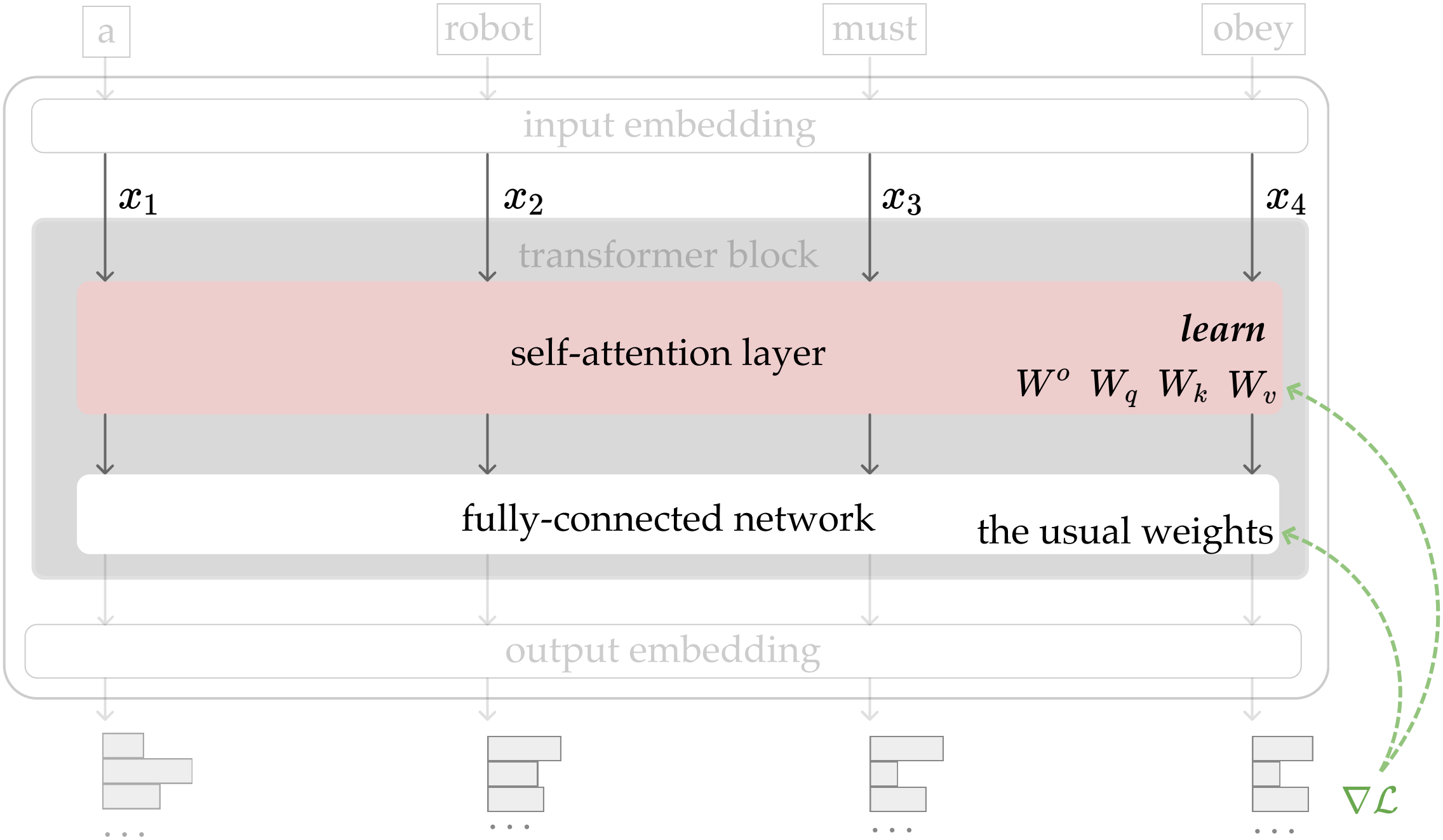
Week 9 - Transformers

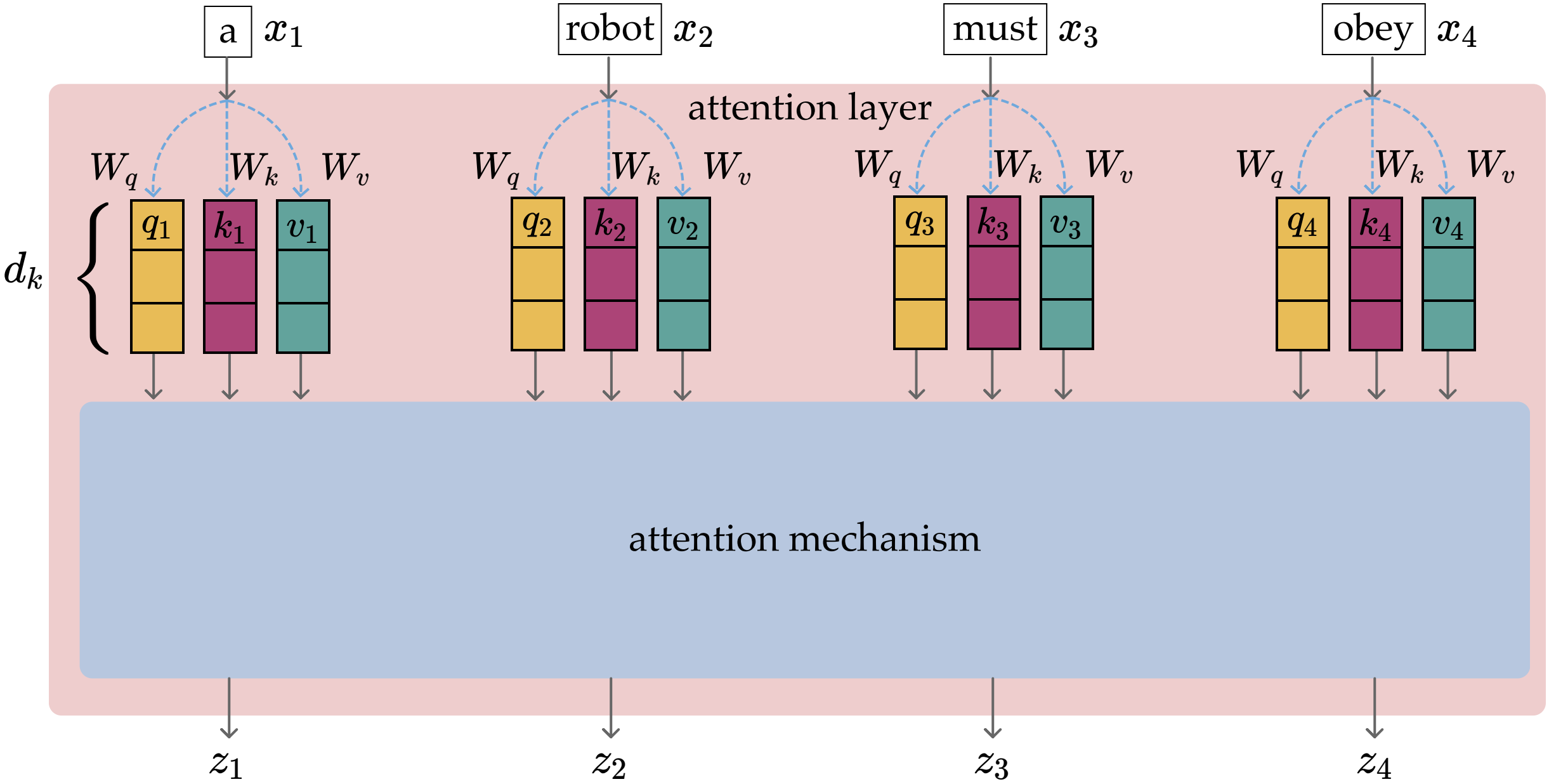
- A sequence of tokenized inputs: n tokens, each token x is d dimensional
- the attention mechanism (one head)
  - learn weights W_q, W_k, W_v to turn raw x inputs into (query, key, value)
  - the mechanics, softmax(raw attention score), shapes
  - masking: why and how
- parallel-processing machines
  - each head is processed in parallel
  - inside a head, each token is processed in parallel

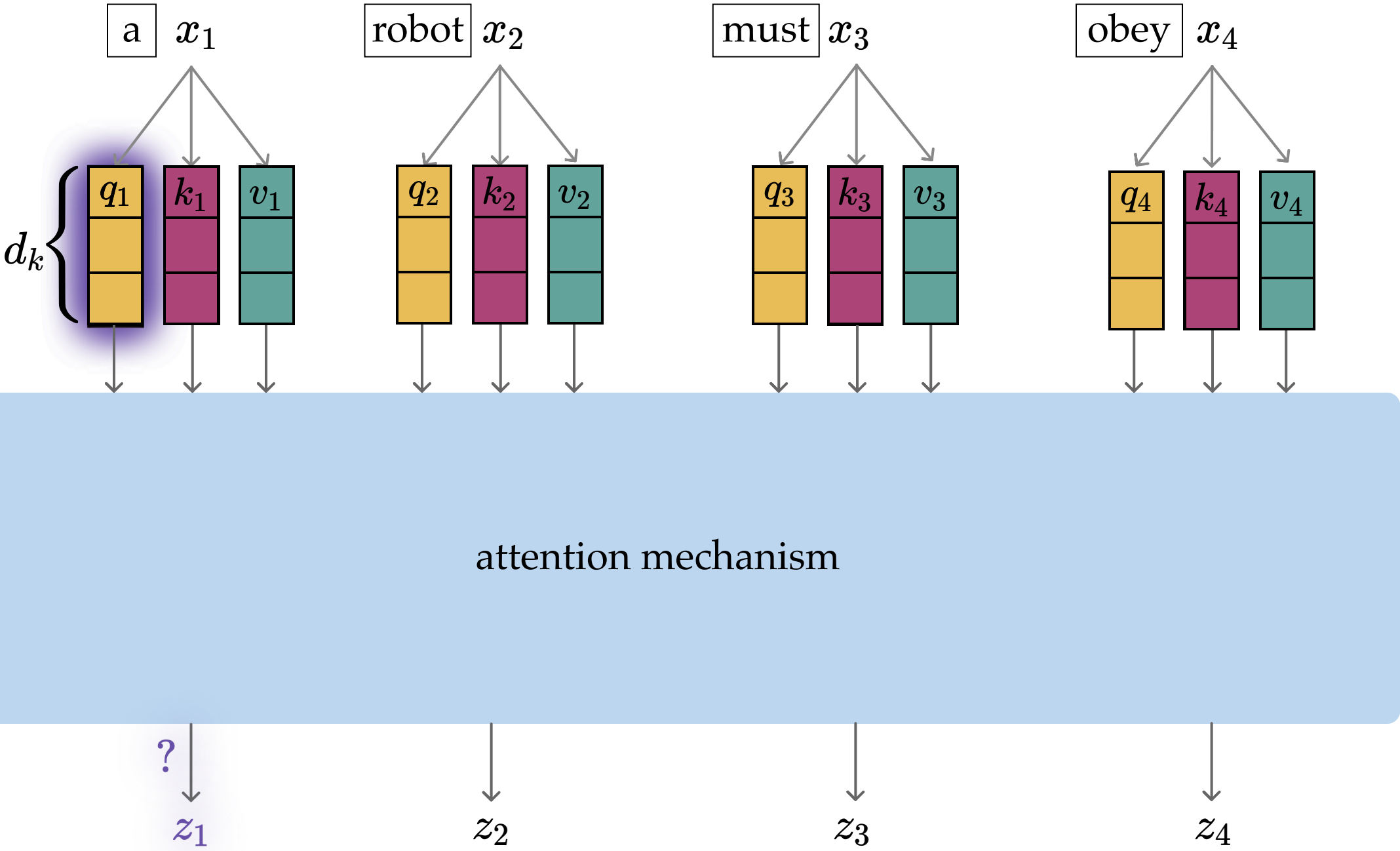

a
robot
must
obey
input embedding
transformer block
transformer block
transformer block
. . .
output embedding
L
blocks
. . .
. . .
. . .
. . .

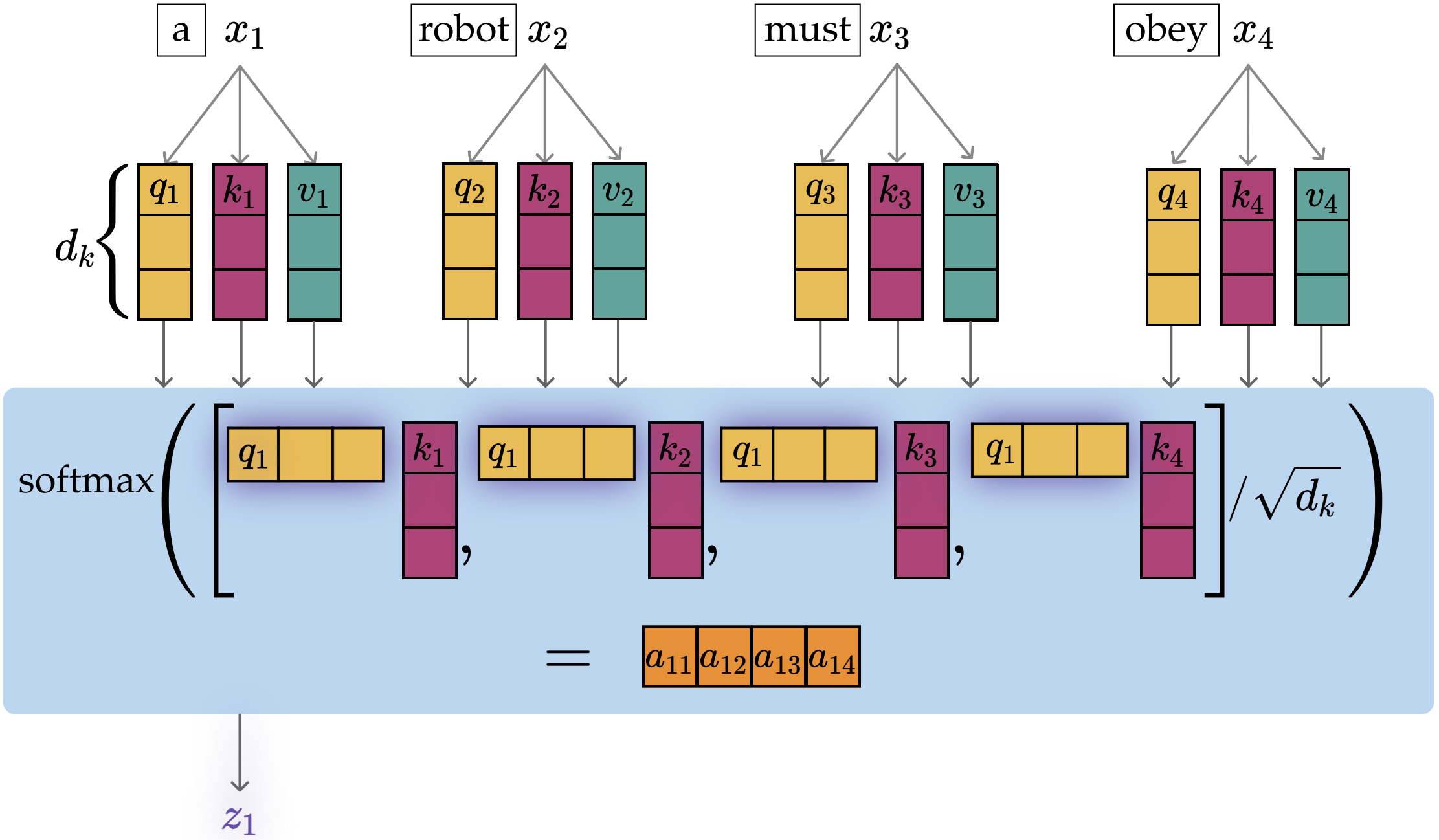

a
robot
must
obey
input embedding
x_1
x_2
x_3
x_4
transformer block
A sequence of n tokens, each token in $\mathbb{R}^d$
transformer block
transformer block
. . .
output embedding

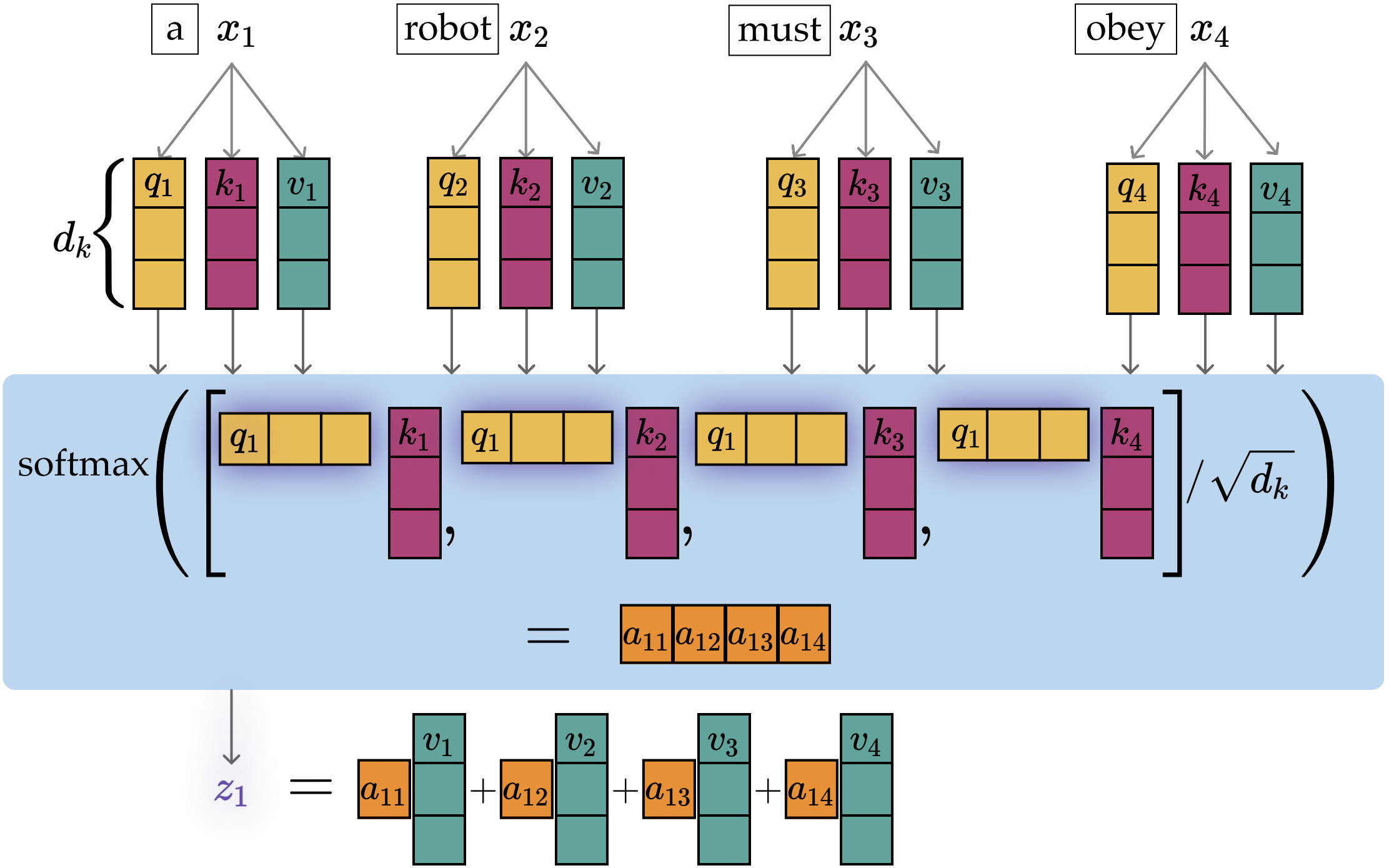

a
robot
must
obey
input embedding
x_1
x_2
x_3
x_4
transformer block
transformer block
transformer block
. . .
output embedding
. . .
. . .
. . .
. . .


a
robot
must
obey
input embedding
x_1
x_2
x_3
x_4
transformer block
self-attention layer
learn
W^o W_q W_k W_v
fully-connected network
the usual weights
output embedding
$\nabla\mathcal{L}$


a x_1
robot x_2
must x_3
obey x_4
attention layer
W_q W_k W_v
d_k
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
attention mechanism
z_1 z_2 z_3 z_4

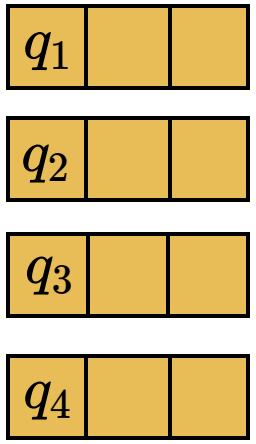

a
x_1
robot
x_2
must
x_3
obey
x_4
d_k
q_1
k_1
v_1
q_2
k_2
v_2
q_3
k_3
v_3
q_4
k_4
v_4
attention mechanism
?
z_1
z_2
z_3
z_4

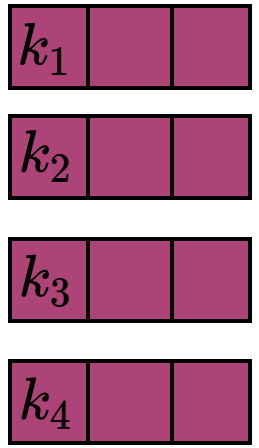

a x_1
robot x_2
must x_3
obey x_4
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
d_k
softmax
$/\sqrt{d_k}$
$=$
a_{11} a_{12} a_{13} a_{14}
z_1


a x_1
robot x_2
must x_3
obey x_4
d_k
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
softmax
$/\sqrt{d_k}$
$=$
a_{11} a_{12} a_{13} a_{14}
z_1 $=$
a_{11} v_1 $+$ a_{12} v_2 $+$ a_{13} v_3 $+$ a_{14} v_4

$\mathbb{R}^{n \times d_k}$

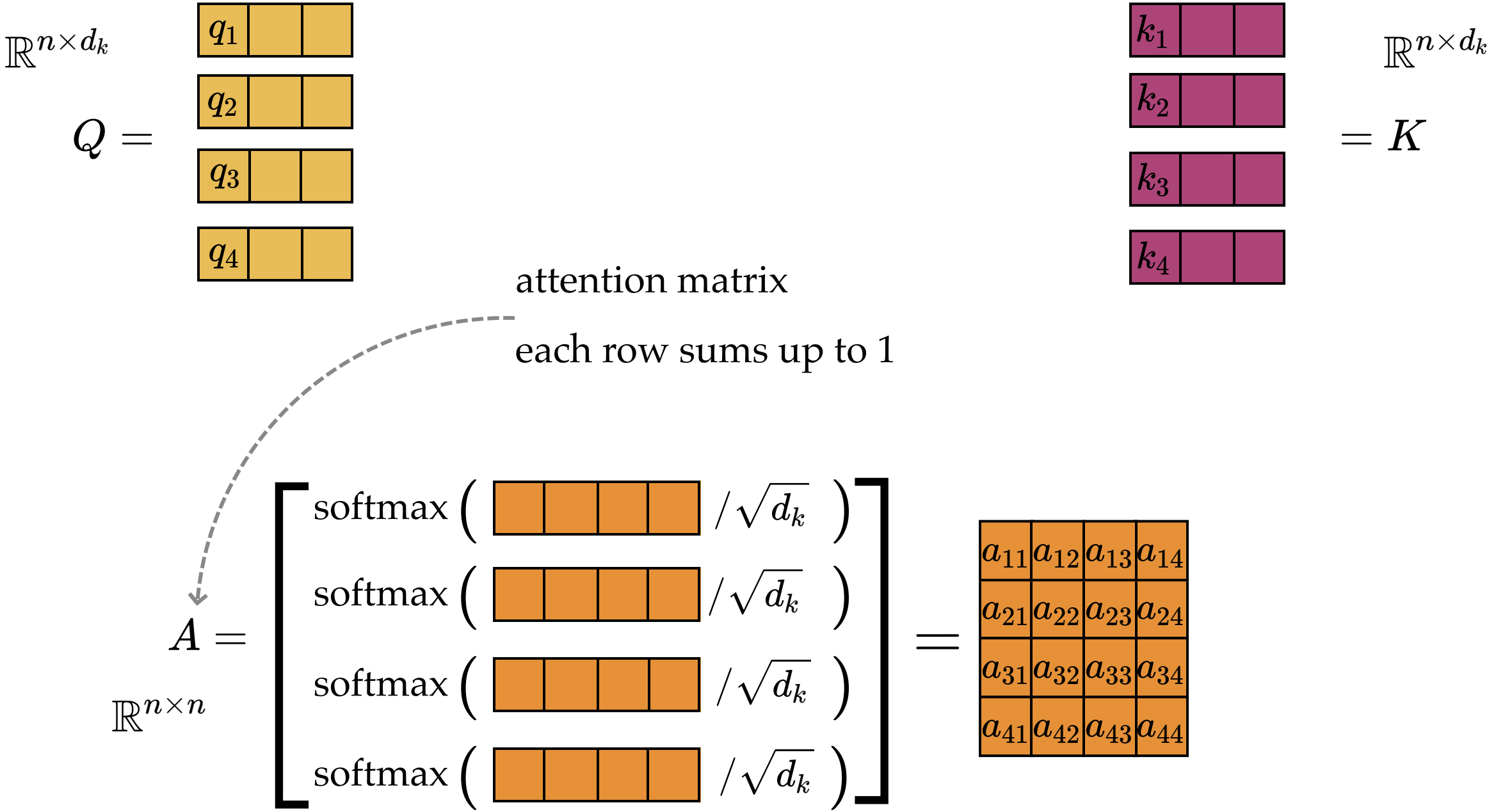
$Q =$

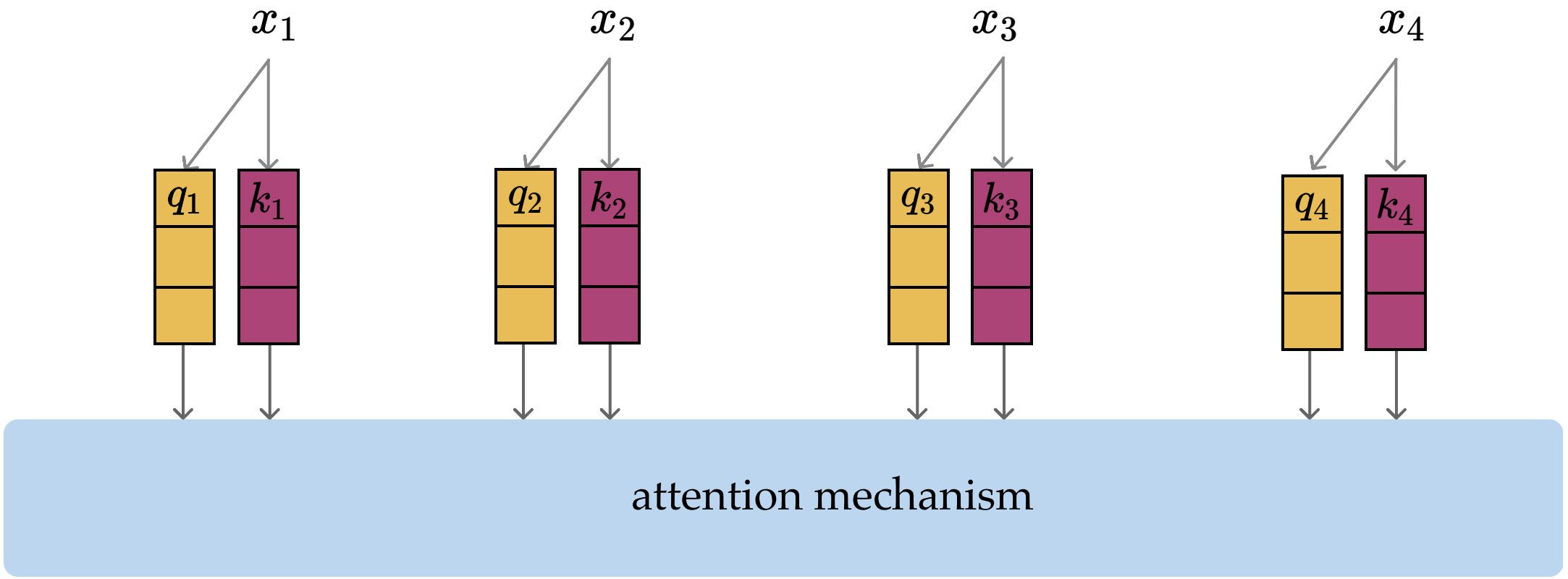




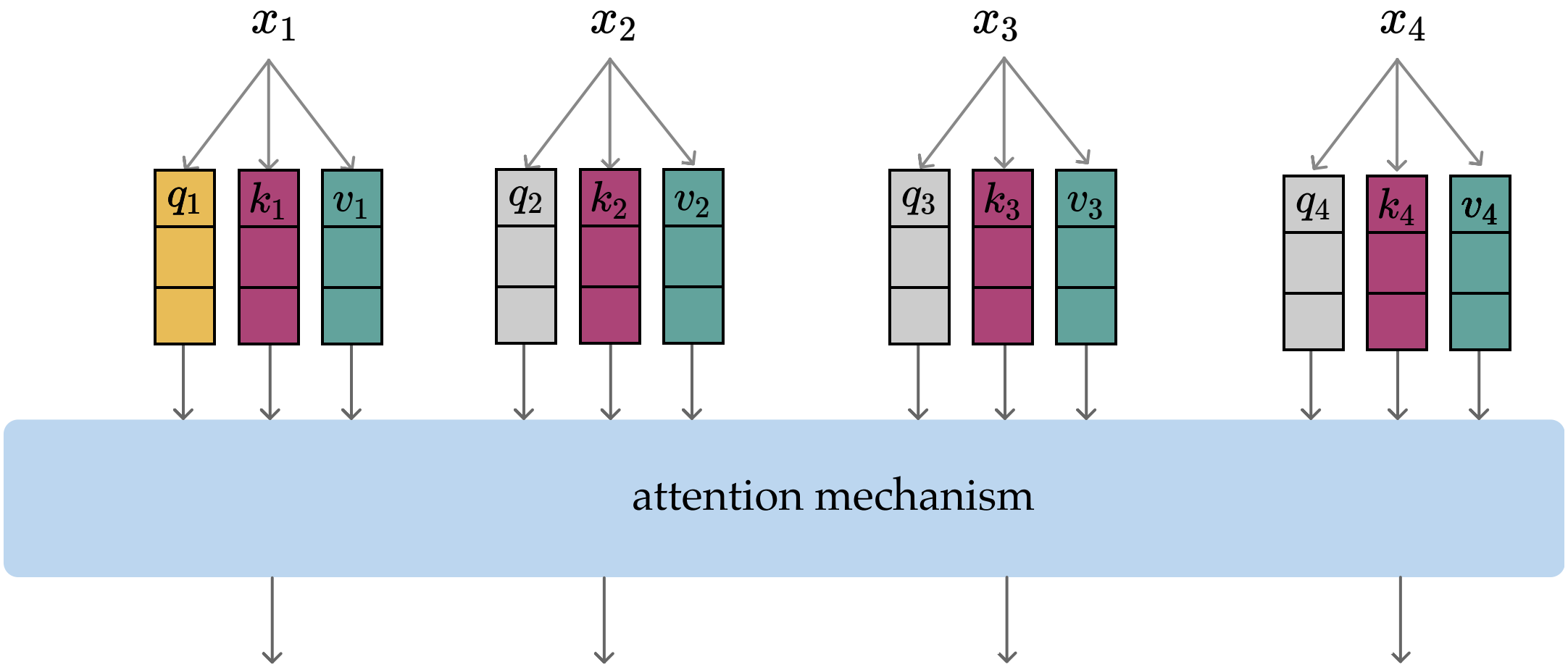
$\mathbb{R}^{n \times d_k}$

$= K$

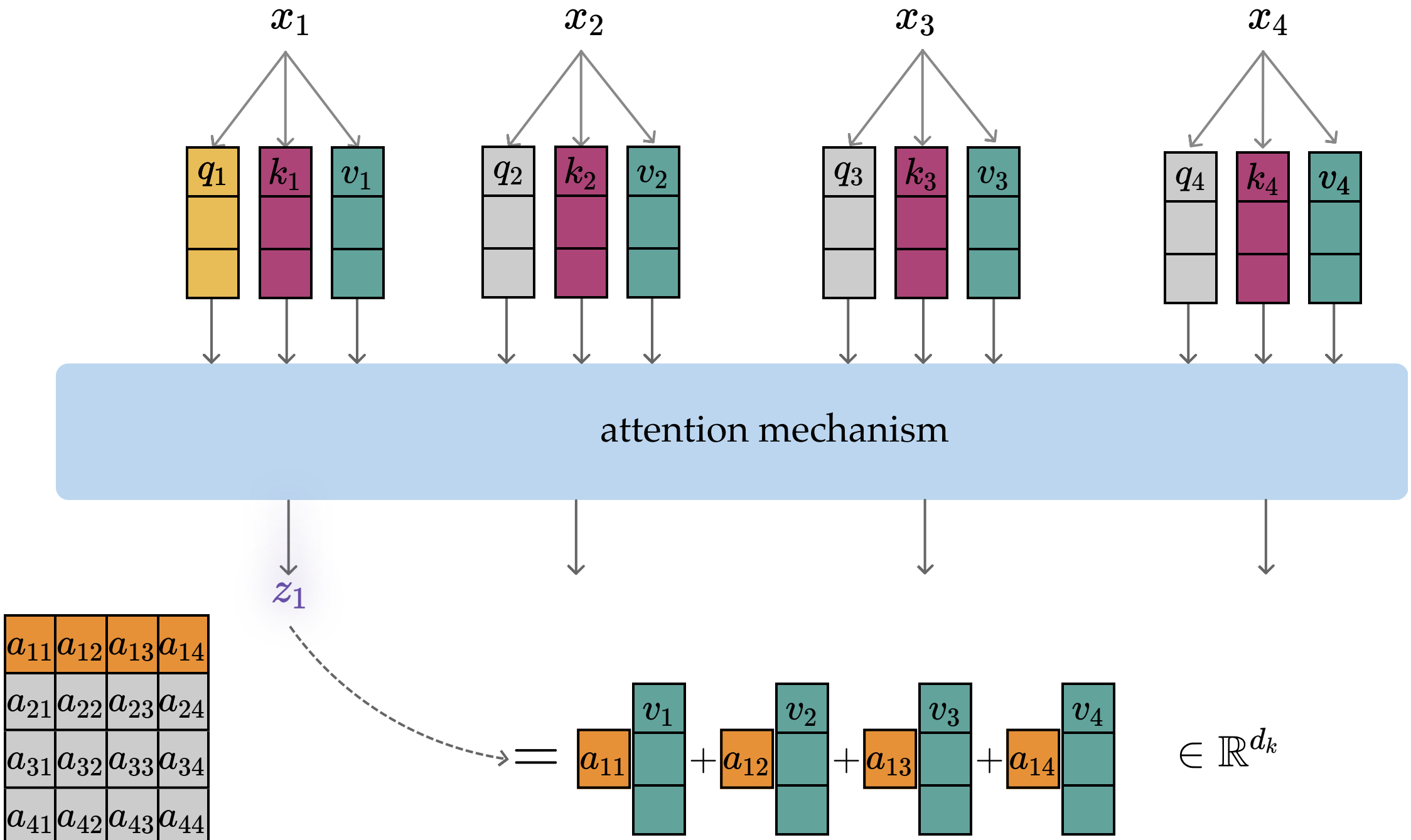

$\mathbb{R}^{n \times d_k}$
$Q =$
q_1
q_2
q_3
q_4
k_1
k_2
k_3
k_4
$\mathbb{R}^{n \times d_k}$
$= K$
attention matrix
each row sums up to 1
$A =$
$\mathbb{R}^{n \times n}$
softmax $/\sqrt{d_k}$
$=$
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{31} a_{32} a_{33} a_{34}
a_{41} a_{42} a_{43} a_{44}



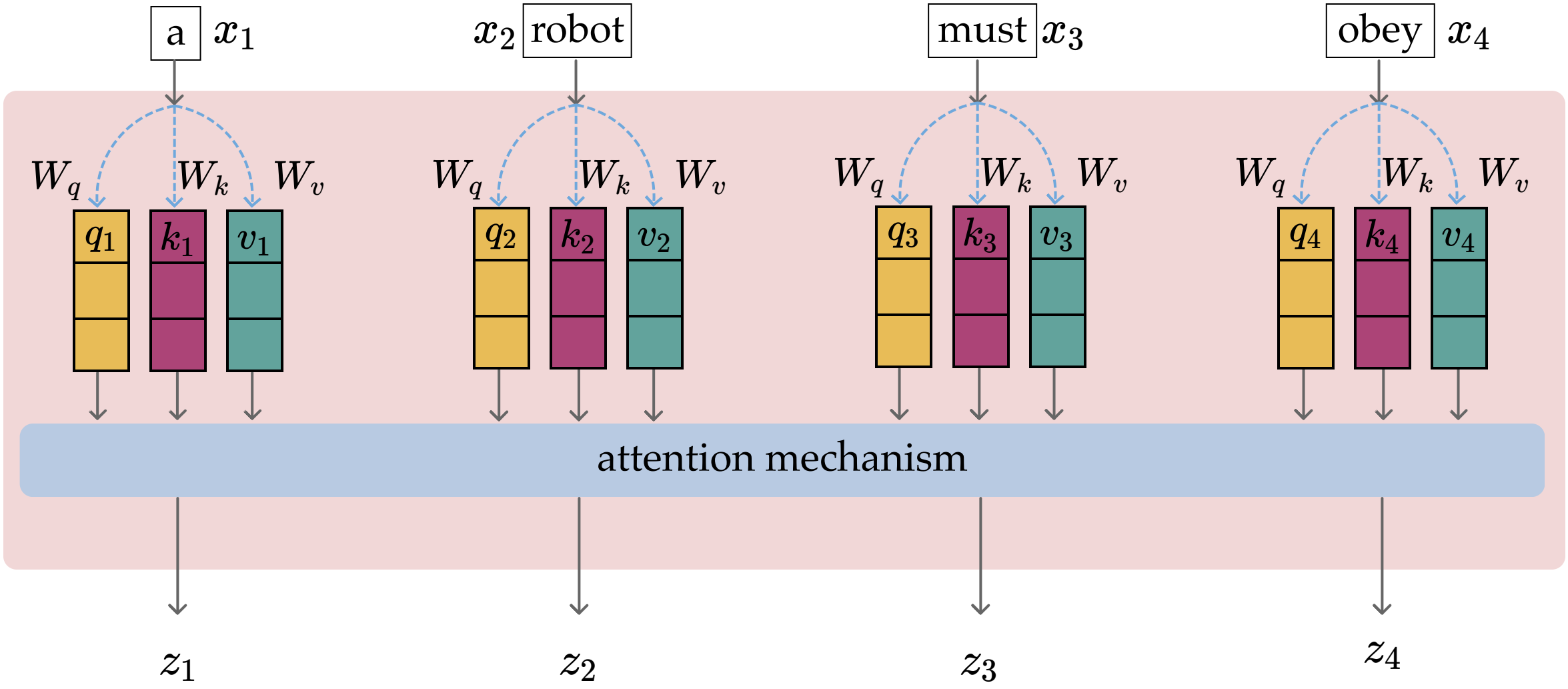
a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

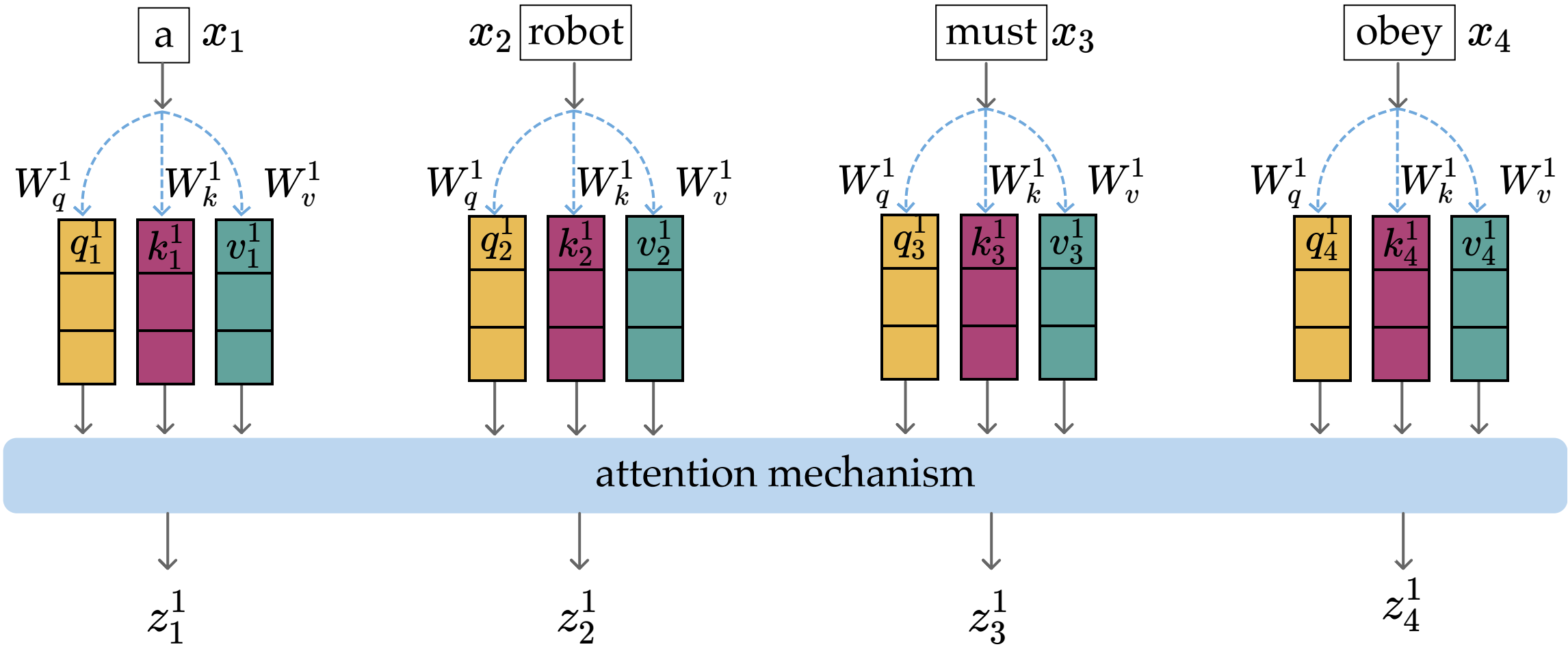


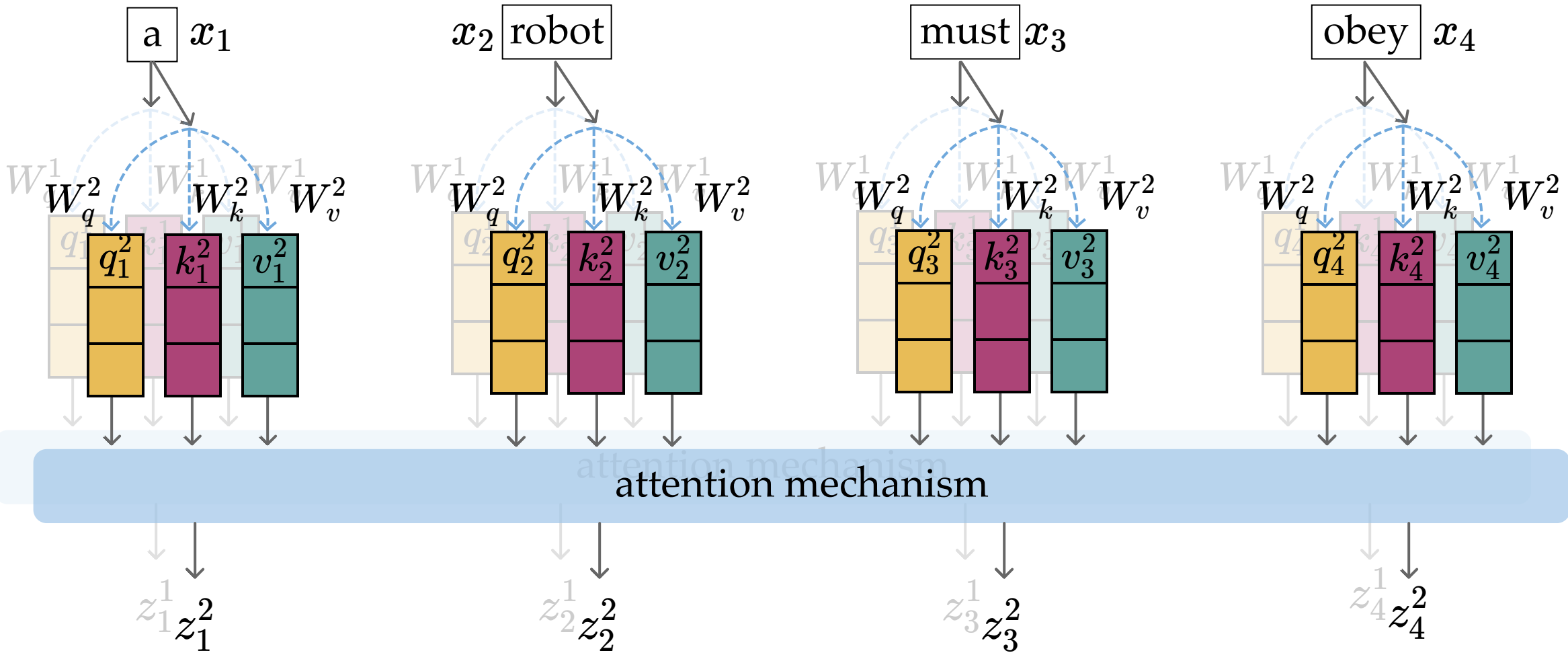
a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

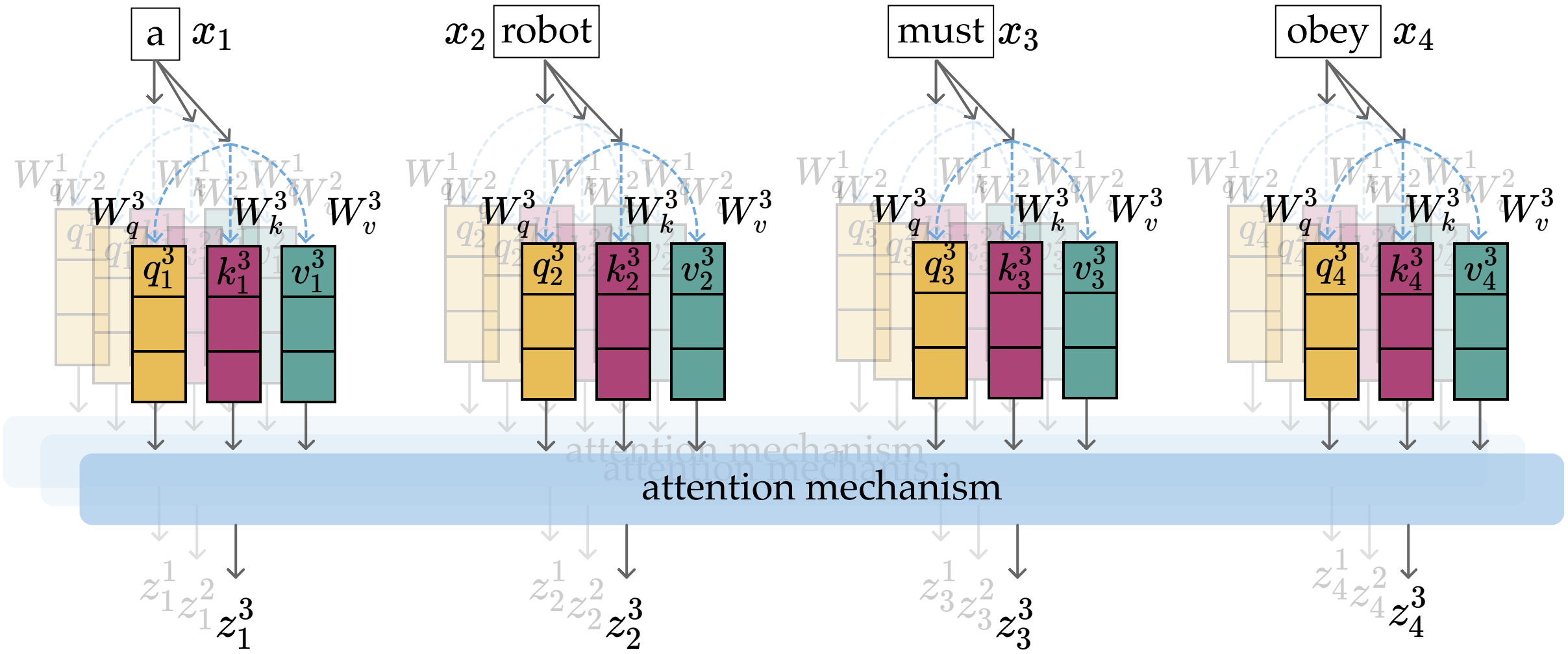

x_1
x_2
x_3
x_4
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
attention mechanism
z_1
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{31} a_{32} a_{33} a_{34}
a_{41} a_{42} a_{43} a_{44}
$= a_{11} v_1 + a_{12} v_2 + a_{13} v_3 + a_{14} v_4$
$\in \mathbb{R}^{d_k}$

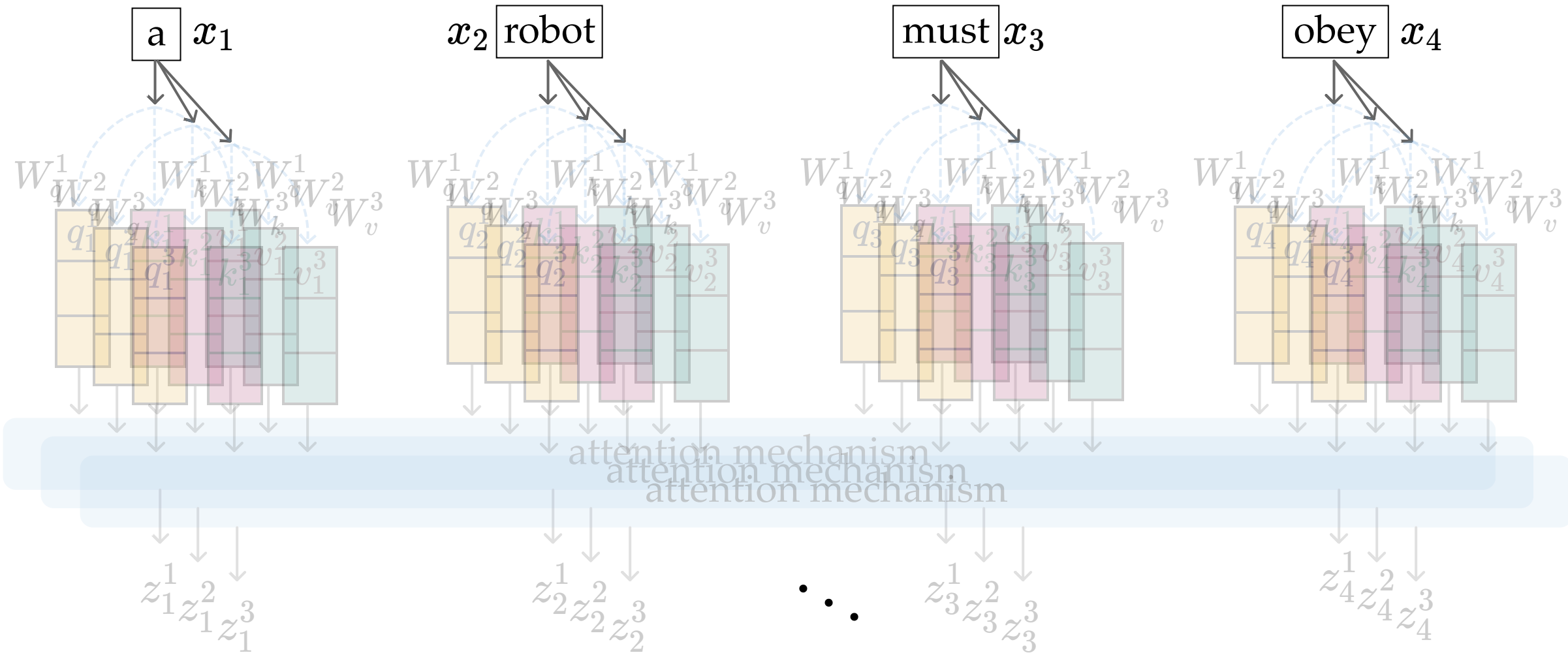
one attention head

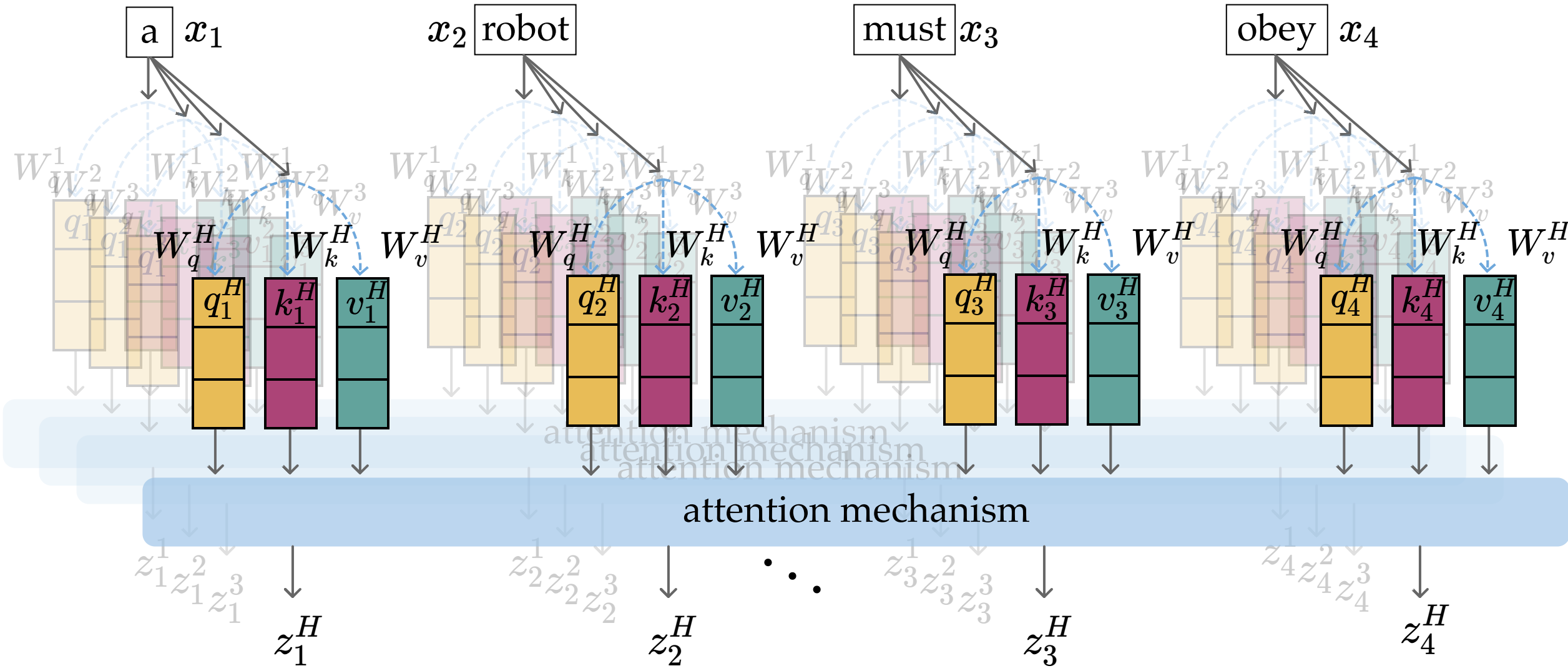


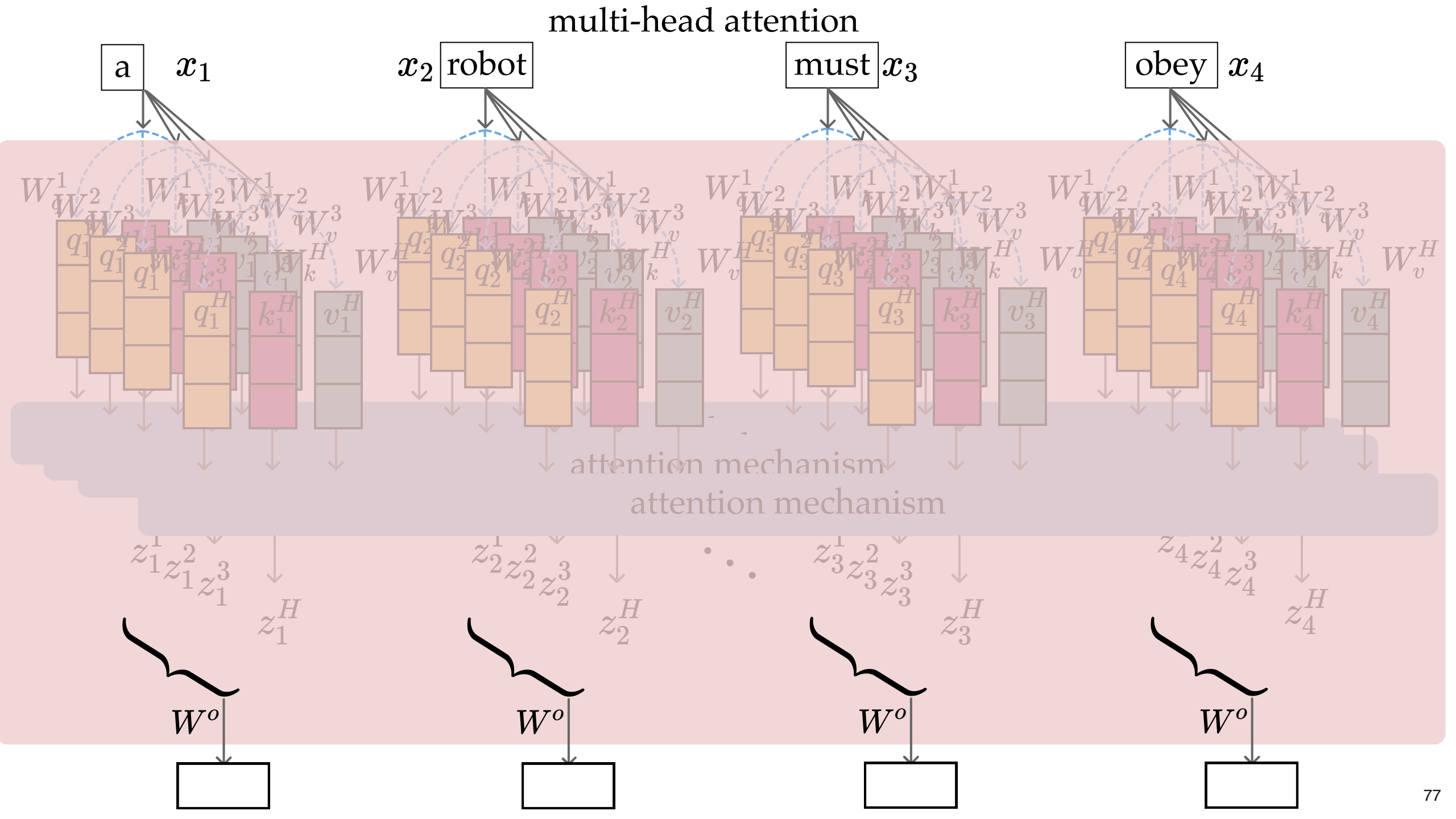

a
x_1
x_2
robot
must
x_3
obey
x_4
W_q^1
W_k^1
W_v^1
q_1^1
k_1^1
v_1^1
q_2^1
k_2^1
v_2^1
q_3^1
k_3^1
v_3^1
q_4^1
k_4^1
v_4^1
attention mechanism
z_1^1
z_2^1
z_3^1
z_4^1


a
x_1
x_2
robot
must
x_3
obey
x_4
W_q^2
W_k^2
W_v^2
q_1^2
k_1^2
v_1^2
q_2^2
k_2^2
v_2^2
q_3^2
k_3^2
v_3^2
q_4^2
k_4^2
v_4^2
attention mechanism
z_1^2
z_2^2
z_3^2
z_4^2


a
x_1
x_2
robot
must
x_3
obey
x_4
W_q^3
W_k^3
W_v^3
q_1^3
k_1^3
v_1^3
q_2^3
k_2^3
v_2^3
q_3^3
k_3^3
v_3^3
q_4^3
k_4^3
v_4^3
attention mechanism
z_1^3
z_2^3
z_3^3
z_4^3


a
x_1
x_2
robot
must
x_3
obey
x_4
attention mechanism
attention mechanism
attention mechanism


a
x_1
x_2
robot
must
x_3
obey
x_4
W_q^H
W_k^H
W_v^H
q_1^H
k_1^H
v_1^H
q_2^H
k_2^H
v_2^H
q_3^H
k_3^H
v_3^H
q_4^H
k_4^H
v_4^H
attention mechanism
z_1^H
z_2^H
z_3^H
z_4^H


multi-head attention
a
x_1
x_2
robot
must
x_3
obey
x_4
attention mechanism
attention mechanism
W^o
W^o
W^o
W^o

Shape Example

n	num tokens	2
d	token dim	4
d_k	(qkv) dim	3
H	num heads	5

W_q^h	query proj	$d \times d_k$	4×3
W_k^h	key proj	$d \times d_k$	4×3
W_v^h	value proj	$d \times d_k$	4×3
W^o	output proj	$d \times Hd_k$	4×15
input	-	$n \times d$	2×4
Q^h	query	$n \times d_k$	2×3
K^h	key	$n \times d_k$	2×3
V^h	value	$n \times d_k$	2×3
A^h	attn matrix	$n \times n$	2×2
Z^h	head out.	$n \times d_k$	2×3
output	-	$n \times d$	2×4

learned

Week 10 - MDPs

- Definition (the five tuple)
  - π, V and Q : definition and interpretation
- Policy evaluation: given $\pi(s)$, calculate $\mathrm{V}^{\pi}(s)$
  - via summation, or via Bellman recursion or equation
- Policy optimization: finding optimal policy $\pi^{*}(s)$
  - toy setup: solve via heuristics; more generally: Q value-iteration
- Interpretation of optimal policy
  - how R, γ, h can change optimal policy

Markov Decision Processes - Definition and terminologies

- $\mathcal{S}$: state space, contains all possible states s.
- $\mathcal{A}$: action space, contains all possible actions a.
- $\mathrm{T}(s, a, s')$: the probability of transition from state s to s' when action a is taken.
- $\mathrm{R}(s, a)$: reward, takes in a (state, action) pair and returns a reward.
- $\gamma \in [0, 1]$: discount factor, a scalar.

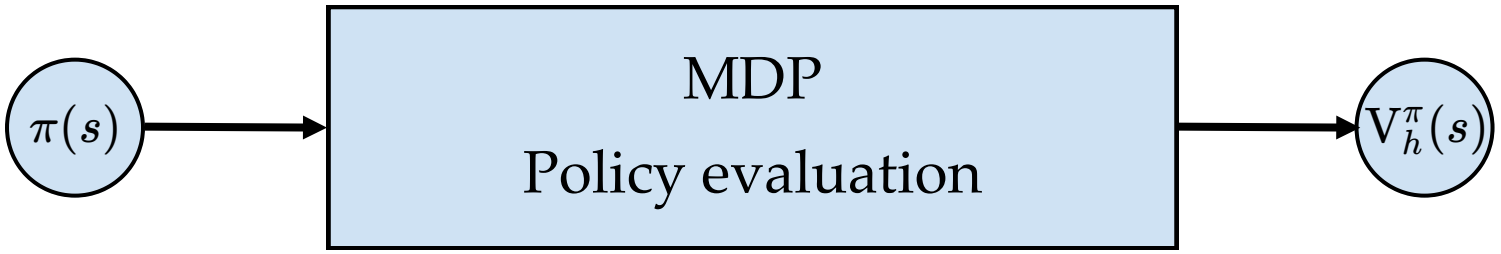
- $\pi(s)$: policy, takes in a state and returns an action.

  The goal of an MDP is to find a "good" policy.

In 6.390,

- $\mathcal{S}$ and $\mathcal{A}$ are small discrete sets, unless otherwise specified.
- s' and a' are short-hand for the next-timestep
- $\mathrm{R}(s, a)$ is deterministic and bounded.
- $\pi(s)$ is deterministic.

Quick summary



1. By summing h terms:

Recall: For a *given* policy $\pi(s)$, the (state) **value functions**

$$\mathrm{V}_h^\pi(s) := \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t \mathrm{R}\left(s_t, \pi\left(s_t\right)\right) \mid s_0 = s, \pi\right], \forall s, h$$

2. By leveraging structure:

finite-horizon Bellman recursions

$$\mathrm{V}_h^\pi(s) = \mathrm{R}(s, \pi(s)) + \gamma \sum_{s'} \mathrm{T}\left(s, \pi(s), s'\right) \mathrm{V}_{h-1}^\pi\left(s'\right), \forall s$$

infinite-horizon Bellman equations

$$\mathrm{V}_\infty^\pi(s) = \mathrm{R}(s, \pi(s)) + \gamma \sum_{s'} \mathrm{T}\left(s, \pi(s), s'\right) \mathrm{V}_\infty^\pi\left(s'\right), \forall s$$

Optimal state-action value functions $\mathrm{Q}_h^*(s, a)$

$\mathrm{Q}_h^*(s, a)$: the expected sum of discounted rewards for

- starting in state s,
- take action a, for one step
- act **optimally** there afterwards for the remaining $(h-1)$ steps

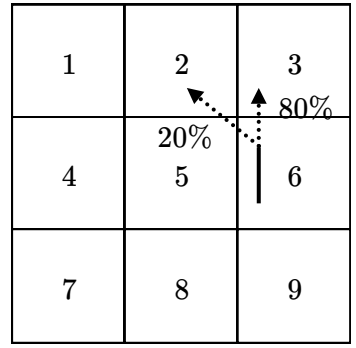
$$\mathrm{Q}_h^*(s, a) = \mathrm{R}(s, a) + \gamma \sum_{s'} \mathrm{T}\left(s, a, s'\right) \max_{a'} \mathrm{Q}_{h-1}^*\left(s', a'\right), \forall s, a, h$$

$Q_h^*(s, a)$: the value for

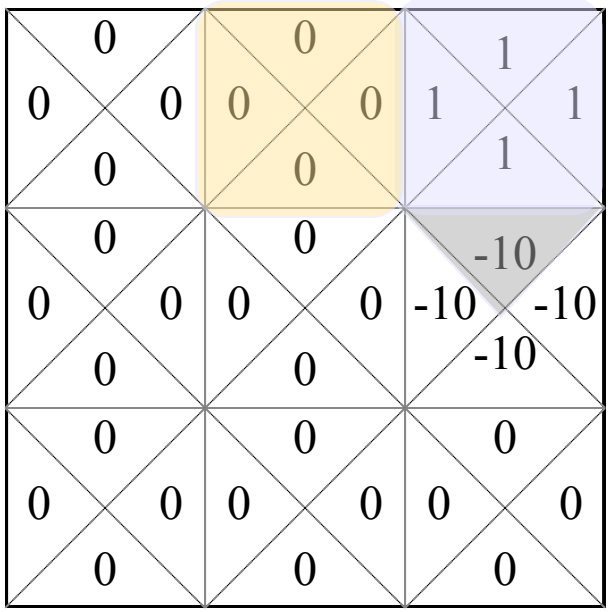
- starting in state s,
- take action a, for one step
- act **optimally** there afterwards for the remaining $(h-1)$ steps

Recall: $\gamma = 0.9$

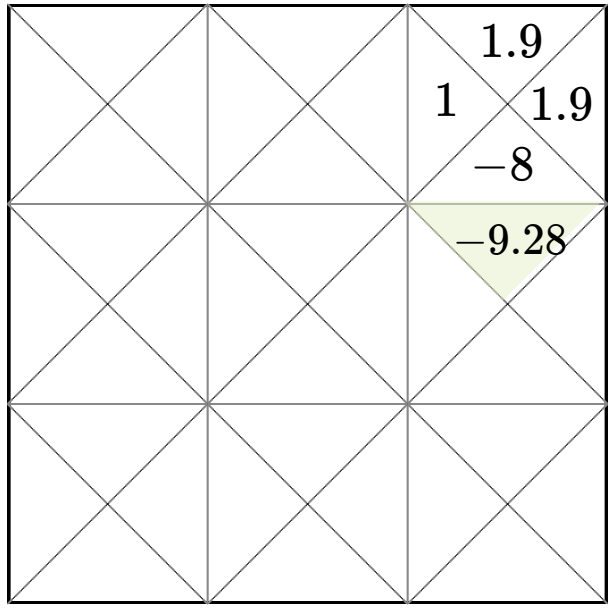
States and one special transition:



$Q_1^*(s, a) = R(s, a)$



$Q_2^*(s, a)$



- receive $R(6, \uparrow)$
- act **optimally** for one more timestep, at the next state s'
  - 20% chance, $s' = 2$, act optimally, receive $\max_{a'} Q_1^*(2, a')$
  - 80% chance, $s' = 3$, act optimally, receive $\max_{a'} Q_1^*(3, a')$

Let's consider $Q_2^*(6, \uparrow)$

$$= R(6, \uparrow) + \gamma[.2 \max_{a'} Q_1^*(2, a') + .8 \max_{a'} Q_1^*(3, a')]$$

$$= -10 + .9[.2 \times 0 + .8 \times 1] = -9.28$$

Given the recursion $\mathrm{Q}_h^*(s,a) = \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s,a,s') \max_{a'} \mathrm{Q}_{h-1}^*(s',a')$

we can have an infinite horizon equation

$$\mathrm{Q}_\infty^*(s,a) = \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s,a,s') \max_{a'} \mathrm{Q}_\infty^*(s',a')$$

`Value Iteration`

1. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2. $\quad \mathrm{Q}_{\text{old}}(\mathrm{s},\mathrm{a}) = 0$
3. **while** True:
4. $\quad$ **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
5. $\quad\quad \mathrm{Q}_{\text{new}}(s,a) \leftarrow \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s,a,s') \max_{a'} \mathrm{Q}_{\text{old}}(s',a')$
6. $\quad$ **if** $\max_{s,a} |Q_{\text{old}}(s,a) - Q_{\text{new}}(s,a)| < \epsilon$:
7. $\quad\quad$ **return** Q_{new}
8. $\quad \mathrm{Q}_{\text{old}} \leftarrow \mathrm{Q}_{\text{new}}$

if run this block h times and break, then the returns are exactly Q_h^*

$\mathrm{Q}_\infty^*(s,a)$

Week 11 - Reinforcement Learning

- How RL setup differs from MDP
- Q-learning algorithm
  - Forward thinking: given experiences, work out Q-values.
  - Backward thinking: given realized Q-values, work out experiences.
  - Two new hyper-parameters (compared with MDP value iteration):
    - $\epsilon-$greedy action selection
    - α the learning rate
- The idea of fitting parameterized Q-functions via regression, can handle larger or continuous state/action space

~~Markov Decision Processes~~ - Definition and terminologies

Reinforcement Learning

- $\mathcal{S}$: state space, contains all possible states s.
- $\mathcal{A}$: action space, contains all possible actions a.
- ~~T (s, a, s') : the probability of transition from state s to s' when action a is taken~~
- ~~R(s, a) : reward, takes in a (state, action) pair and returns a reward~~.
- $\gamma \in [0, 1]$: discount factor, a scalar.

- $\pi(s)$: policy, takes in a state and returns an action.

  The goal of an ~~MDP~~ problem is to find a "good" policy.

  RL

Value Iteration($\mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}, \gamma, \epsilon$)

1. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2. $\quad \mathrm{Q}_{\text{old}}(\mathrm{s}, \mathrm{a}) = 0$
3. **while** True:
4. $\quad$ **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
5. $\quad\quad \mathrm{Q}_{\text{new}}(s, a) \leftarrow \mathrm{R}(s, a) + \gamma \sum_{s'} \mathrm{T}(s, a, s') \max_{a'} \mathrm{Q}_{\text{old}}(s', a')$
6. $\quad$ **if** $\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon$:
7. $\quad\quad$ **return** Q_{new}
8. $\quad \mathrm{Q}_{\text{old}} \leftarrow \mathrm{Q}_{\text{new}}$

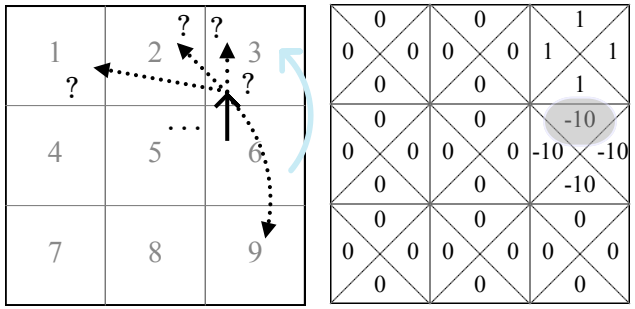
"calculating"

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$ max-iter)

1. $i = 0$
2. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
3. $\quad \mathrm{Q}_{\text{old}}(s, a) = 0$
4. $s \leftarrow s_0$
5. **while** $i <$ max-iter :
6. $\quad a \leftarrow \text{select\_action}(s, \mathrm{Q}_{\text{old}}(s, a))$
7. $\quad r, s' \leftarrow \text{execute}(a)$
8. $\quad \mathrm{Q}_{\text{new}}(s, a) \leftarrow (1 - \alpha)\mathrm{Q}_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} \mathrm{Q}_{\text{old}}(s', a'))$
9. $\quad s \leftarrow s'$
10. $\quad i \leftarrow (i + 1)$
11. $\quad \mathrm{Q}_{\text{old}} \leftarrow \mathrm{Q}_{\text{new}}$
12. **return** Q_{new}

"learning" (estimating)

$\gamma = 0.9$ rewards now known

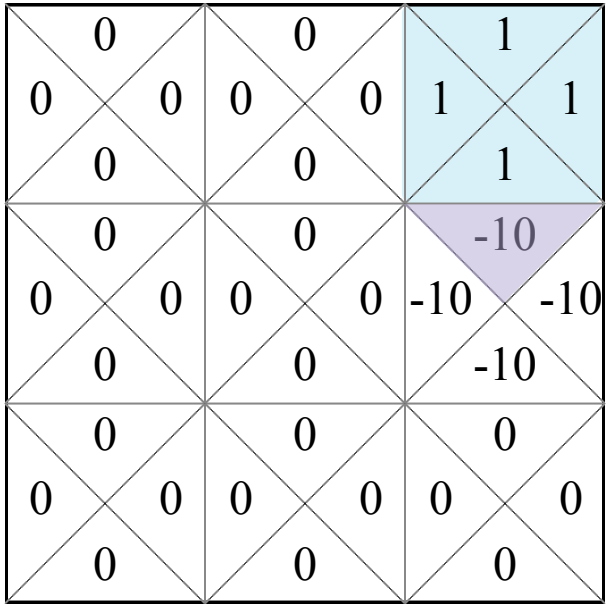


Q-learning update

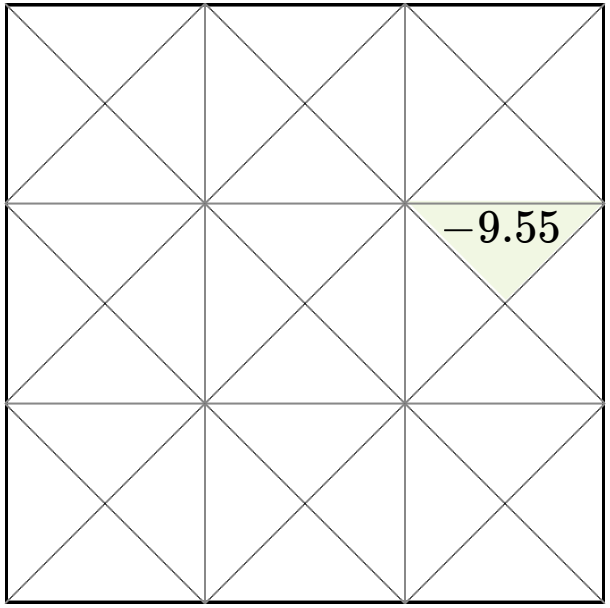
e.g. pick $\alpha = 0.5$

$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



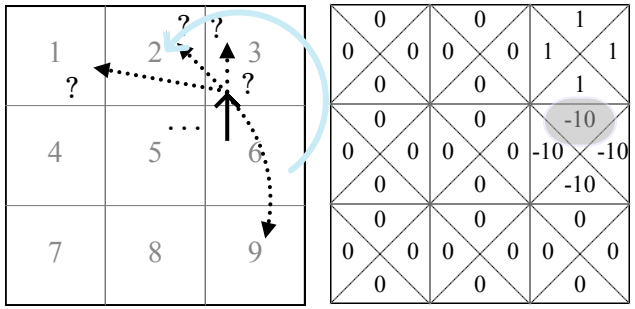
To update the estimate of Q(6, ↑):

- execute (6, ↑)
- observe reward $r = -10$, next state $s' = 3$
- update Q(6, ↑) as:

(1-0.5) * -10+ 0.5(−10+ 0.9 $\max_{a'} Q_{old}(3,a')$)

= -5 + 0.5(-10 + 0.9)= - 9.55

$\gamma = 0.9$ rewards now known

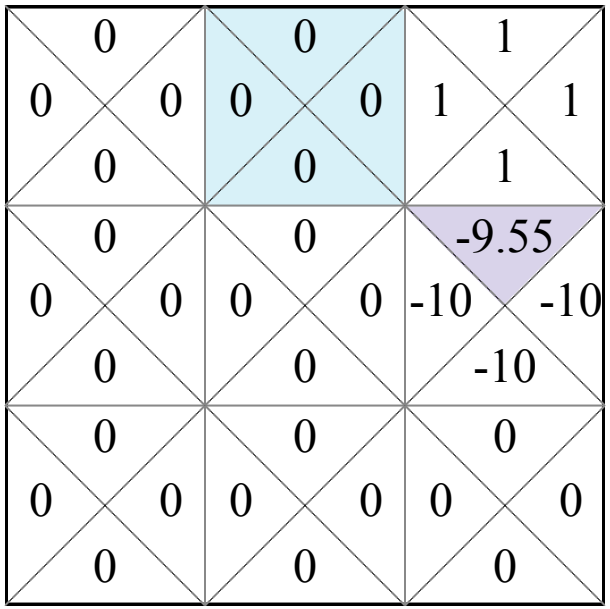


Q-learning update

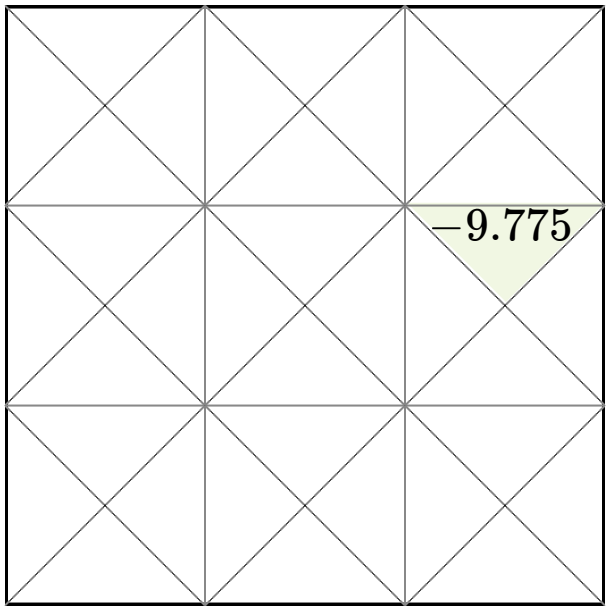
e.g. pick $\alpha = 0.5$

$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



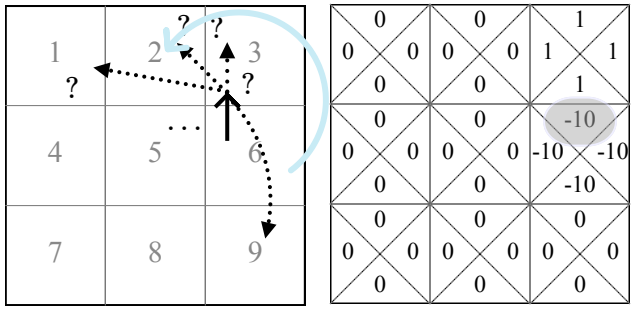
To update the estimate of Q(6, ↑):

- execute (6, ↑)
- observe reward $r = -10$, next state $s' = 2$
- update Q(6, ↑) as:

(1-0.5) * -9.55 + 0.5(−10+ 0.9 $\max_{a'} Q_{old}(2,a')$)

= -4.775 + 0.5(-10 + 0)= - 9.775

$\gamma = 0.9$ rewards now known

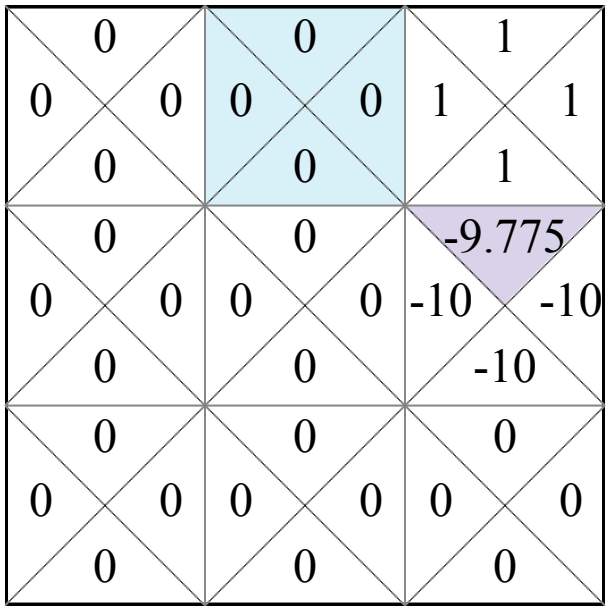


Q-learning update

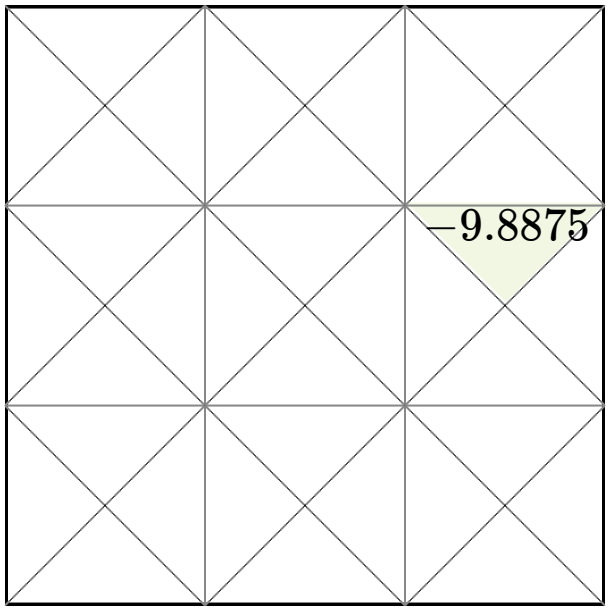
e.g. pick $\alpha = 0.5$

$$Q_{new}(s,a) \leftarrow (1-\alpha)Q_{old}(s,a) + \alpha\left(r + \gamma \max_{a'} Q_{old}(s',a')\right)$$

$Q_{old}(s,a)$



$Q_{new}(s,a)$



To update the estimate of Q(6, ↑):

- execute (6, ↑)
- observe reward $r = -10$, next state $s' = 2$
- update Q(6, ↑) as:

(1-0.5) * -9.775+ 0.5(−10+ 0.9 $\max_{a'} Q_{old}(2,a')$)

= -4.8875 + 0.5(-10 + 0)= - 9.8875

Q-Learning ($\mathcal{S}, \mathcal{A}, \gamma, \alpha, s_0$ max-iter)

1. $i = 0$
2. **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
3. $\quad \mathrm{Q}_{\text{old}}(s, a) = 0$
4. $s \leftarrow s_0$
5. **while** $i <$ max-iter :
6. $\quad a \leftarrow \text{select\_action}(s, \mathrm{Q}_{\text{old}}(s, a))$
7. $\quad r, s' \leftarrow \text{execute}(a)$
8. $\quad \mathrm{Q}_{\text{new}}(s, a) \leftarrow (1 - \alpha)\mathrm{Q}_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} \mathrm{Q}_{\text{old}}(s', a'))$
9. $\quad s \leftarrow s'$
10. $\quad i \leftarrow (i + 1)$
11. $\quad \mathrm{Q}_{\text{old}} \leftarrow \mathrm{Q}_{\text{new}}$
12. **return** Q_{new}

"learning"

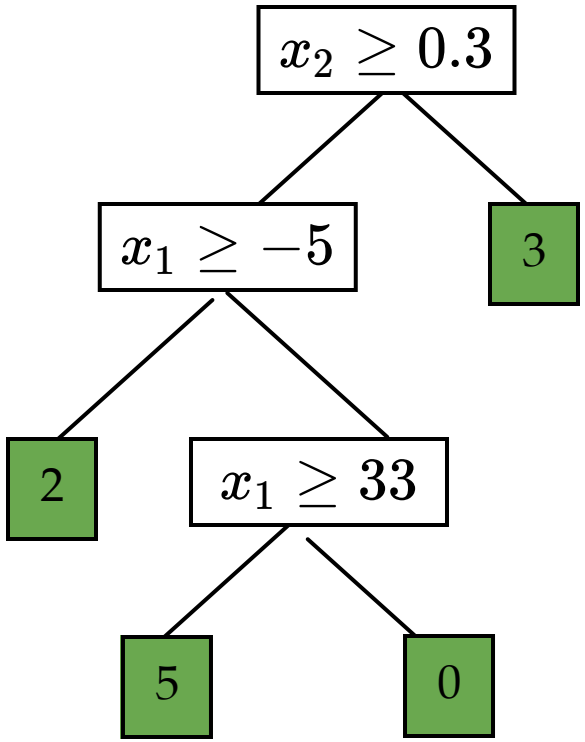
- ϵ-greedy action selection strategy:
  - with probability ϵ, choose an action $a \in \mathcal{A}$ uniformly at random
  - with probability $1 - \epsilon$, choose $\arg\max_{\mathrm{a}} \mathrm{Q}_{\text{old}}(s, \mathrm{a})$

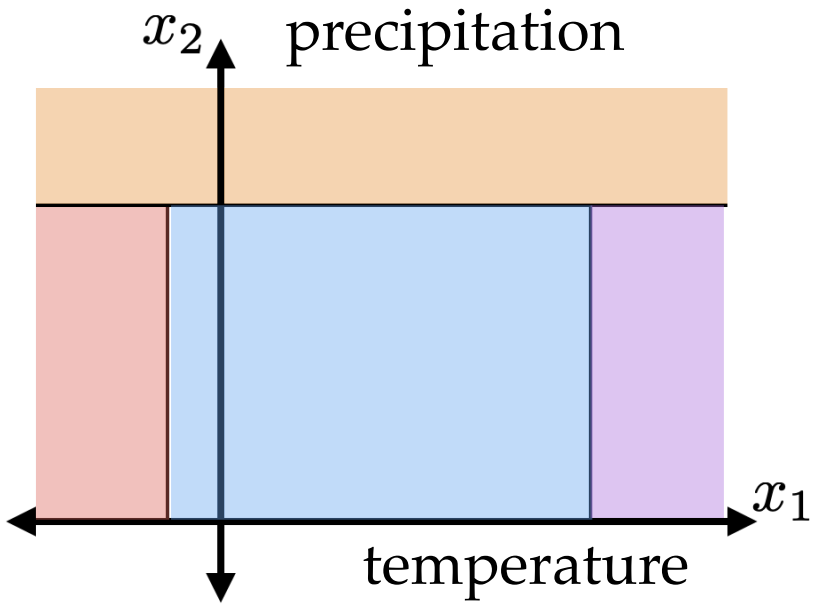
the current estimate of Q values

- ϵ controls the trade-off between *exploration vs. exploitation.*

Week 12 - Non-parametric methods

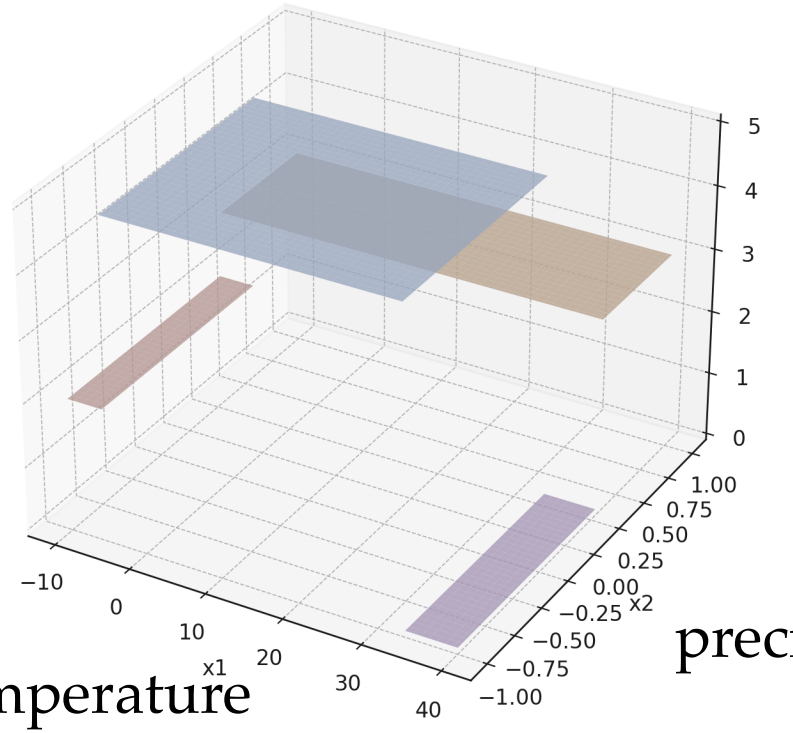
- Decision trees:
  - Split dimension, split value, tree structure (root/decision node and leaf), recursive algorithm, bagging, ensemble, largest leaf size k
- $k-$nearest neighbors:
  - memorizes data, scaling matters, k matters, inefficient in test/prediction time
- k-means clustering: (cluster assignment + cluster center updates), local convergence, initialization matters
- how choice of hyper-parameter k matters in each case





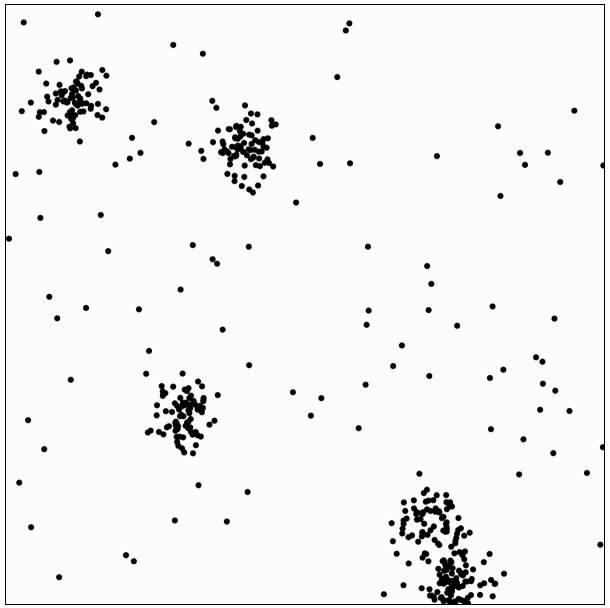
The same prediction applies to an axis-aligned ‘box’ or ‘volume’ in the feature space

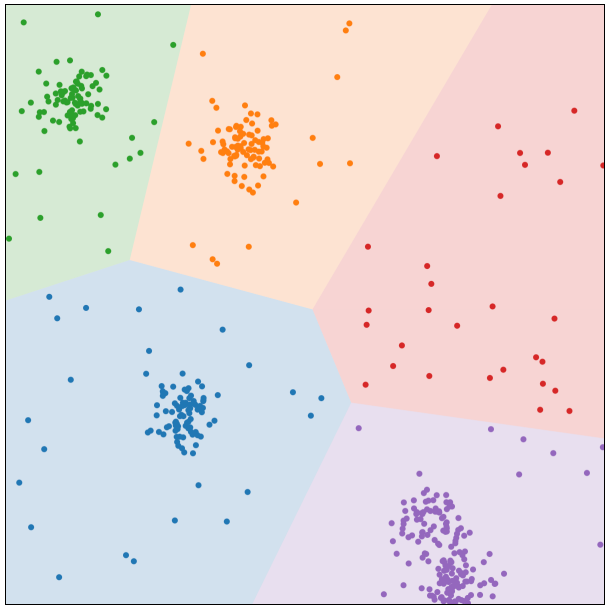
label: km run



precipitation

temperature



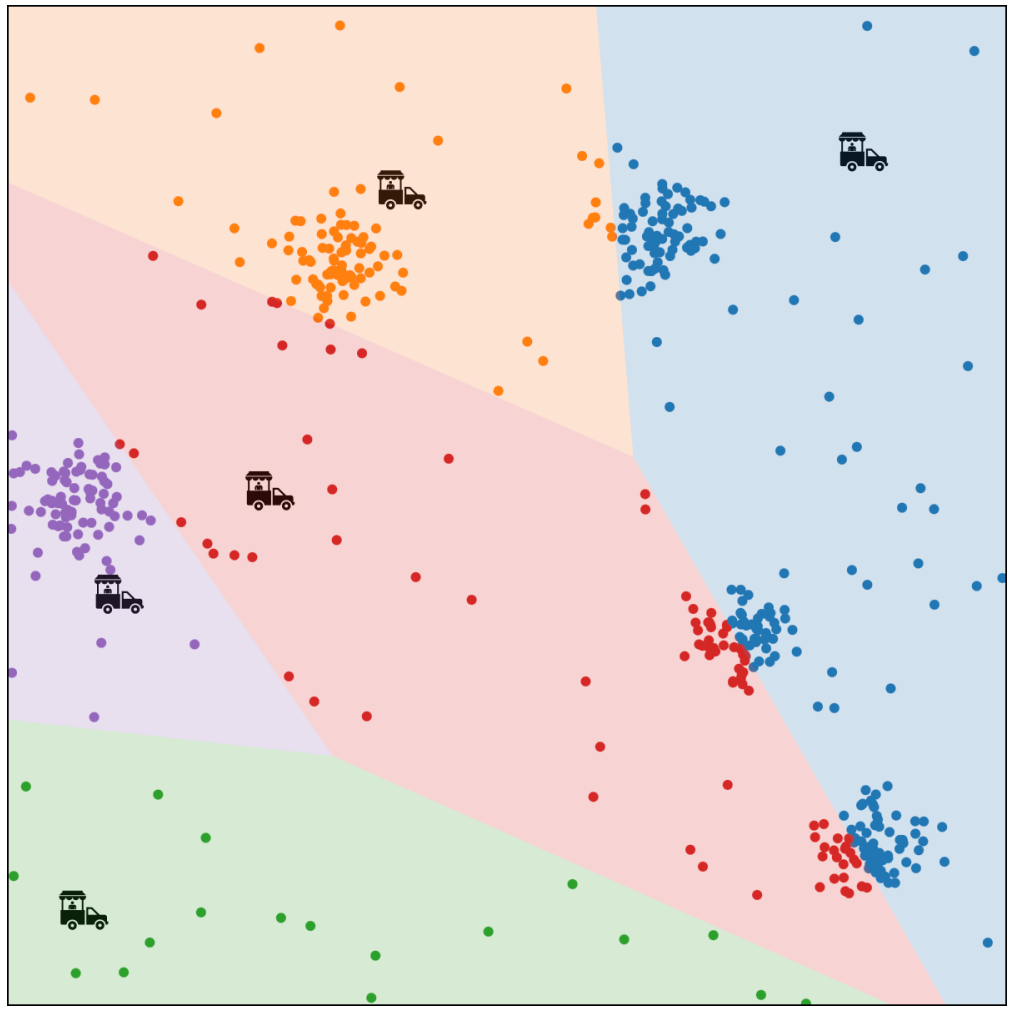


$$\{x^{(1)}\}$$

$$\{x^{(2)}\}$$

$$\{x^{(3)}\}$$

$$\ldots$$



K-MEANS$(k, \tau, \{x^{(i)}\}_{i=1}^{n})$

1 $\mu, y =$ random initialization

2 **for** $t = 1$ to τ

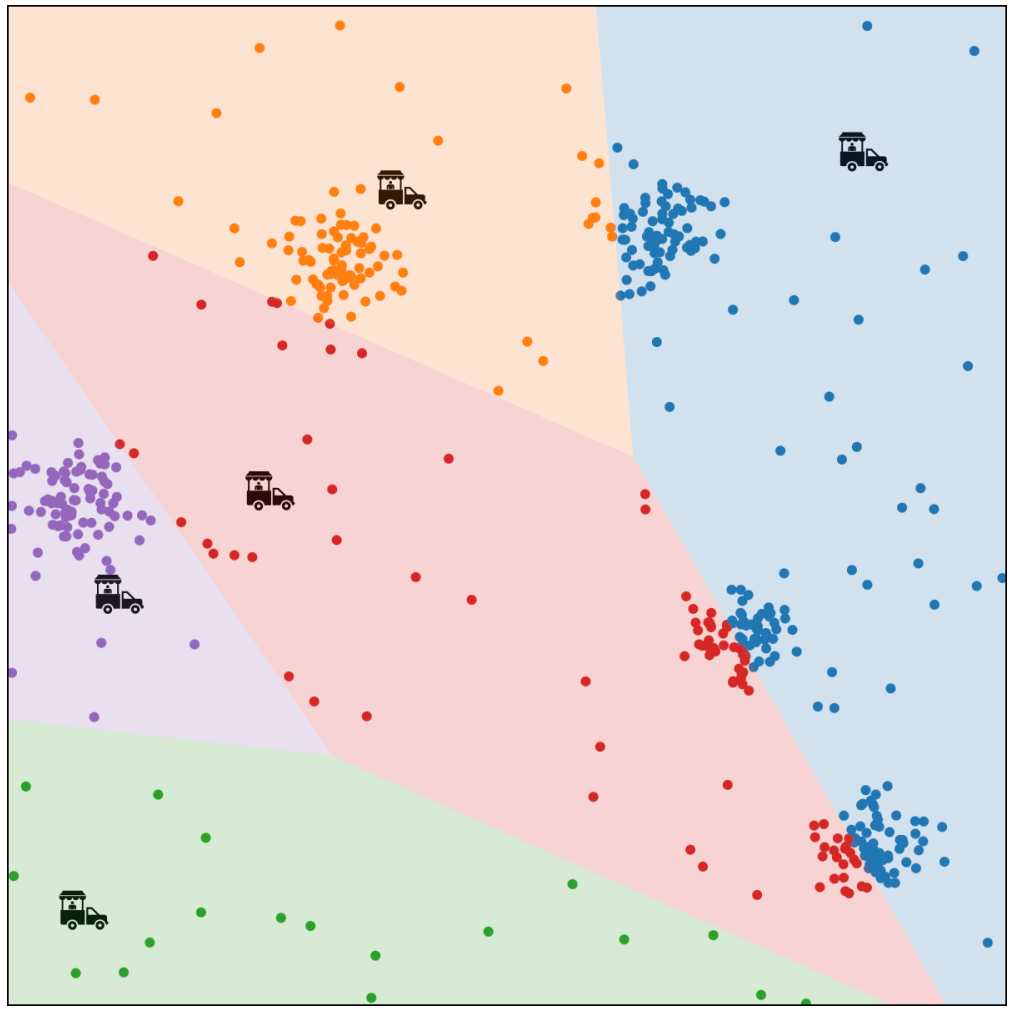
3 $\quad y_{\text{old}} = y$

4 $\quad$ **for** $i = 1$ to n

5 $\quad\quad y^{(i)} = \arg\min_j \left\|x^{(i)} - \mu^{(j)}\right\|^2$

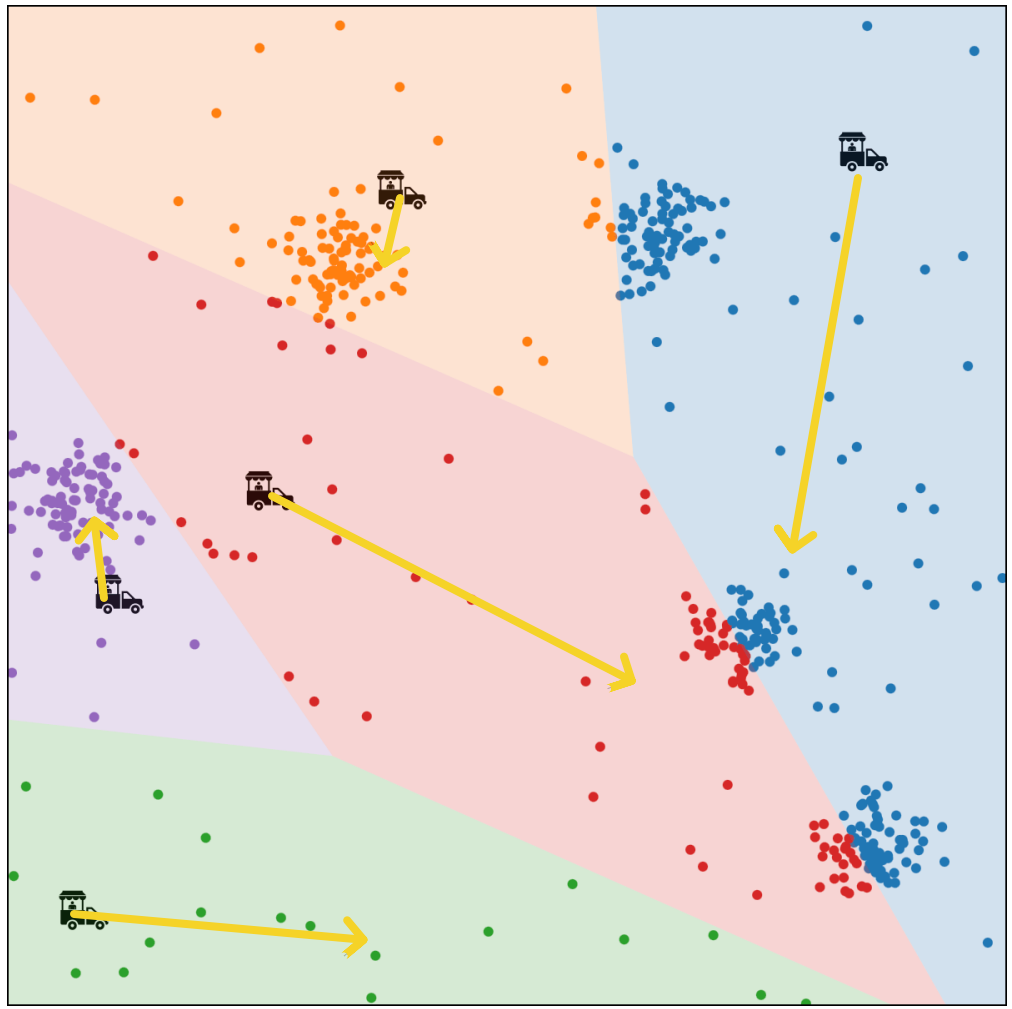
$\quad\quad \ldots$

each person i gets assigned to food truck j, color-coded.



K-MEANS$(k, \tau, \{x^{(i)}\}_{i=1}^{n})$

```
1  μ, y = random initialization
2  for t = 1 to τ
3          y_old = y
4          for i = 1 to n
5                  y^(i) = arg min_j ||x^(i) − μ^(j)||^2
6          for j = 1 to k
7                  μ^(j) = (1/N_j) Σ_{i=1}^{n} 1(y^(i) = j) x^(i)
8          if y == y_old
9                  break
10 return μ, y
```



K-MEANS$(k, \tau, \{x^{(i)}\}_{i=1}^{n})$

1 $\mu, y =$ random initialization

2 **for** $t = 1$ to τ

3 $\quad y_{\text{old}} = y$

4 $\quad$ **for** $i = 1$ to n

5 $\quad\quad y^{(i)} = \arg\min_j \left\| x^{(i)} - \mu^{(j)} \right\|^2$

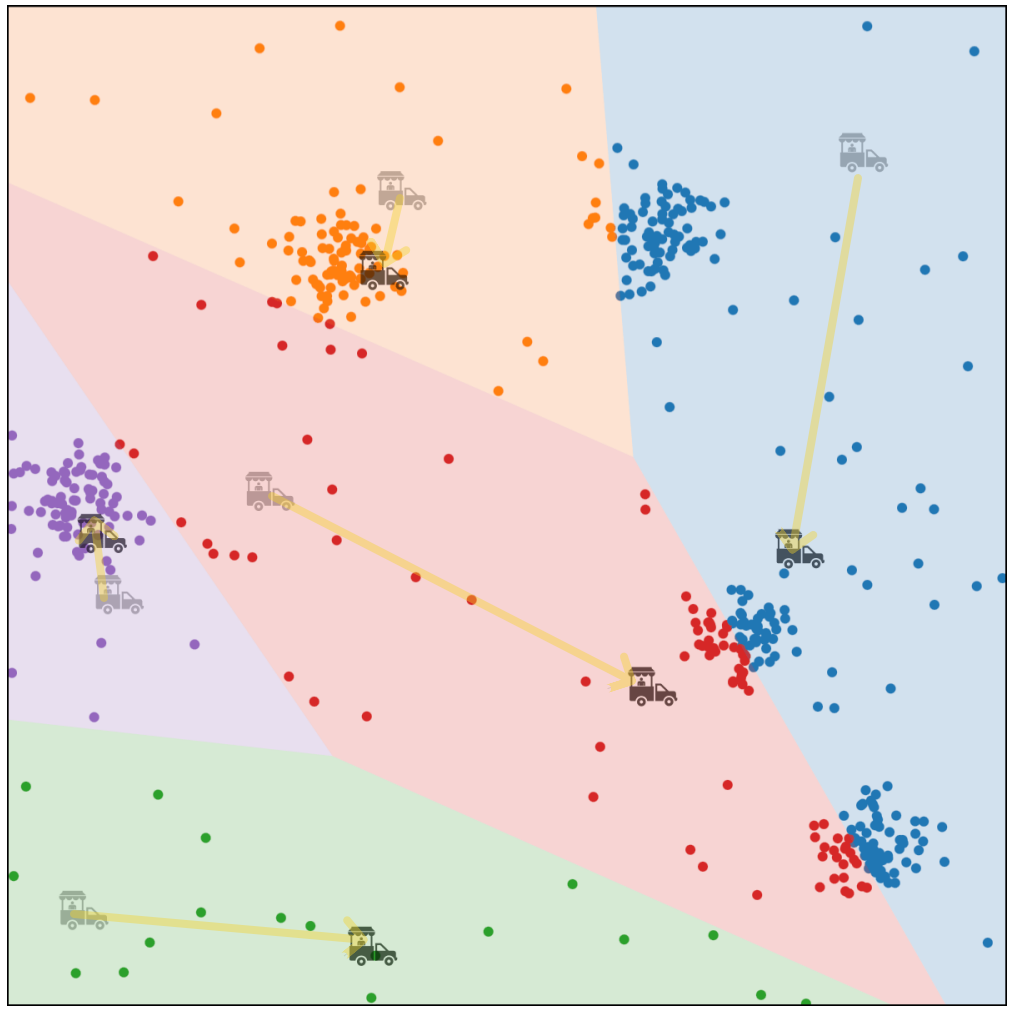
6 $\quad$ **for** $j = 1$ to k

7 $\quad\quad \mu^{(j)} = \frac{1}{N_j} \sum_{i=1}^{n} \mathbf{1}\left(y^{(i)} = \mathrm{j}\right) x^{(i)}$

...

$N_j = \sum_{i=1}^{n} \mathbf{1}\left\{y^{(i)} = j\right\}$

food truck j gets moved to the "central" location of all ppl assigned to it



K-MEANS$(k, \tau, \{x^{(i)}\}_{i=1}^{n})$

$\mu, y =$ random initialization

for $t = 1$ to τ

  $y_{\text{old}} = y$

  for $i = 1$ to n

    $y^{(i)} = \arg\min_j \left\| x^{(i)} - \mu^{(j)} \right\|^2$

  for $j = 1$ to k

    $\mu^{(j)} = \frac{1}{N_j} \sum_{i=1}^{n} \mathbf{1}\left(y^{(i)} = \mathrm{j}\right) x^{(i)}$

  if $y == y_{\text{old}}$

    break

return μ, y

Resources

- All course materials Week 1 - Week 12
- Past-exam Sampler

```python
import random
terms= ["spring2024", "fall2023", "spring2023", "fall2022", "spring2022",
        "fall2021", "fall2019", "fall2018", "fall2018"]

qunums = range(1,10)
base_URL = "https://introml.mit.edu/_static/fall24/final/review/final-"

term = random.choice(terms)
num = random.choice(qunums)
print("term:", term)
print("question number:", num)
print(f"Link: {base_URL+term}.pdf")
```

- Video walk-through of recent final exams
- Piazza and OHs

Course Evaluations

https://registrar.mit.edu/subjectevaluation

https://eduapps.mit.edu/subjeval/studenthome.htm

We'd love to hear your thoughts on the course: this provides valuable feedback for us and other students, for future semesters! Thank you!🙏

https://www.youtube.com/embed/kRPAGejCAeY?enablejsapi=1

https://shenshen.mit.edu/tree

(The demo won't embed in PDF. But the direct link below works.)

https://shenshen.mit.edu/tree

Polya's Problem Solving Techniques

In 1945 George Polya published the book *How To Solve It* which quickly became his most prized publication. It sold over one million copies and has been translated into 17 languages. In this book he identifies four basic principles of problem solving.

Polya's First Principle: Understand the problem

This seems so obvious that it is often not even mentioned, yet studens are often stymied in their efforts to solve problems simply because they don't understand it fully, or even in part. Polya taught teachers to ask students questions such as:

- Do you understand all the words used in stating the problem?
- What are you asked to find or show?
- Can you restate the problem in your own words?
- Can you think of a picture or diagram that might help you understand the problem?
- Is there enough information to enable you to find a solution?

Polya's Second Principle: Devise a plan

Polya mentions that there are many reasonable ways to solve problems. The skill at choosing an appropriate strategy is best learned by solving many problems. You will find choosing a strategy increasingly easy. A partial list of strategies is included:

- Guess and check
- Make an orderly list
- Eliminate possibilities
- Use symmetry
- Consider special cases
- Use direct reasoning
- Solve an equation
- Look for a pattern
- Draw a picture
- Solve a simpler problem
- Use a model
- Work backwards
- Use a formula
- Be ingenious

Polya's Third Principle: Carry out the plan

This step is usually easier than devising the plan. In general, all you need is care and patience, given that you have the necessary skills. Persist with the plan that you have chosen. If it continues not to work discard it and choose another. Don't be misled, this is how mathematics is done, even by professionals.

Polya's Fourth Principle: Look back

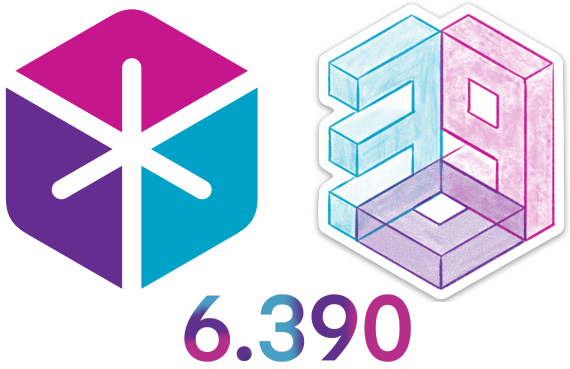
Polya mentions that much can be gained by taking the time to reflect and look back at what you have done, what worked, and what didn't. Doing this will enable you to predict what strategy to use to solve future problems.

More detailed CliffsNotes

Exam-taking tips

- Arrive 5min early to get settled in.
- Bring a pencil (and **eraser**), a watch, and some water.
- Look over whole exam and strategize for the order you do problems.

Good luck!



https://introml.mit.edu/

Thanks for the semester!