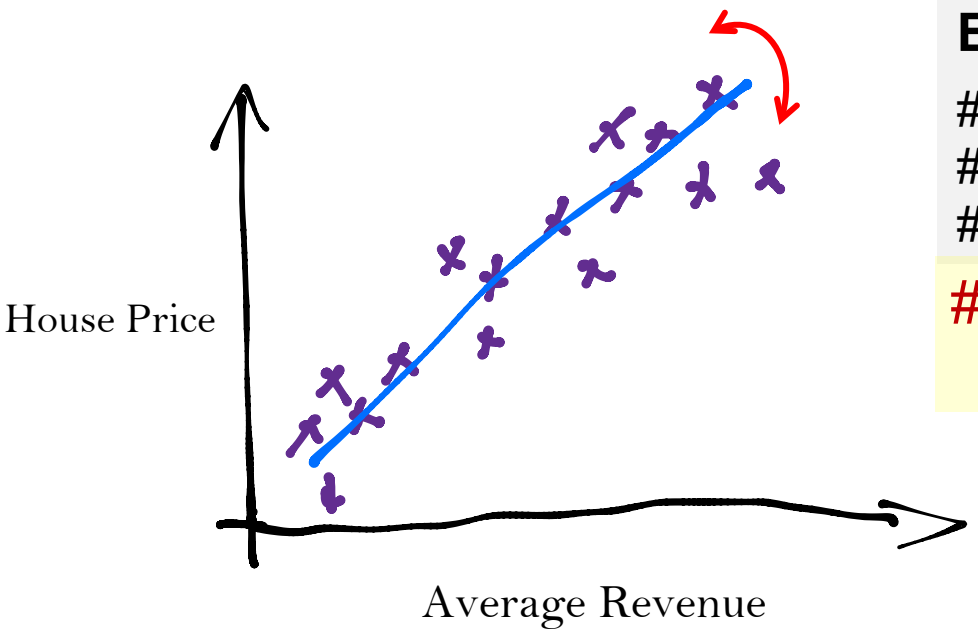
Key Elements of Machine Learning

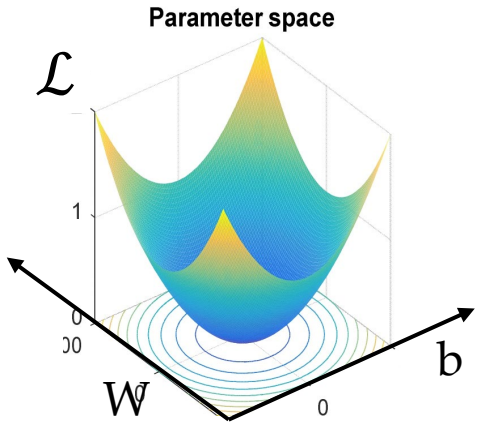


Elements:

#1 Data (Experience)
#2 Model (Hypothesis)
#3 Loss Function (Objective)

#4 Optimization Algorithm
(Improvement)

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta|\mathcal{D})$$



How to find the optimal model?

How to Approach a Machine Learning Problem?

① **consider the nature of available data D**

- how much amount of data can you obtain? how would it cost (in time, computation, human efforts)?

② **select a representation for the input X**

- data preprocessing, feature extraction, etc.

③ **choose a set of possible models H (hypothesis space)**

- set of functions h: X → Y

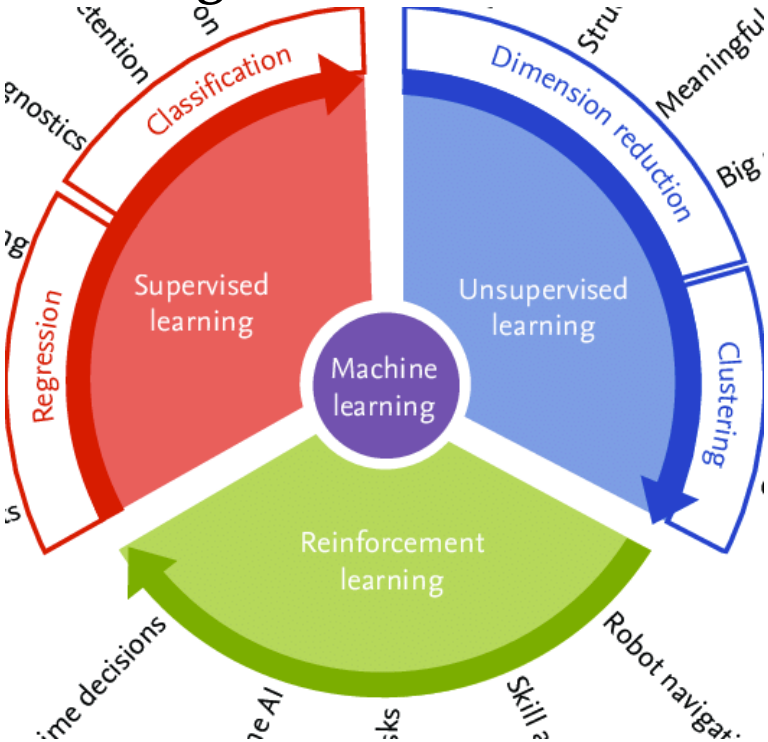
④ **choose the performance measure P (loss function)**

⑤ **choose or design a learning algorithm**

- for using examples (**E**) to converge on a member of **H** that optimizes **P**

Categories of Machine Learning Algorithms

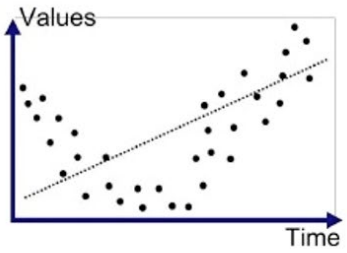
- **Supervised Learning**
- **Unsupervised Learning**
- **Reinforcement Learning**



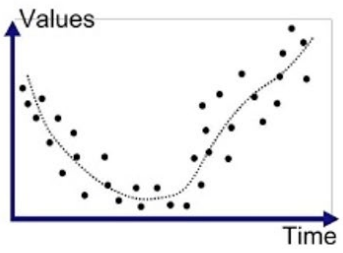
Overfitting

Underfitting – the model isn't complex enough to capture the real knowledge, the assumption may not be true.

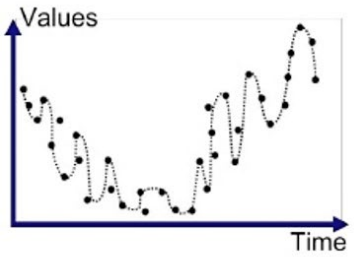
Overfitting – the model is too complex and thus describes the details of data (e.g., random noise) instead of underlying knowledge



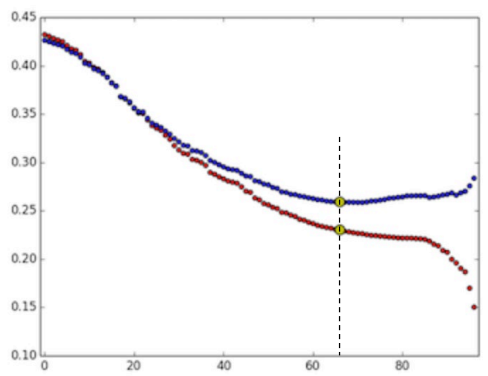
Underfitted



Good Fit/Robust



Overfitted



Example: https://ml.berkeley.edu/blog/posts/crash-course/part-4/

Prevent Overfitting

Selecting appropriate model complexity

In general, design a good loss function to indicate the performance of the predictive model over the whole-data distribution instead of the training data can help achieve compromise between simplicity and complexity of the model structure

Widely used approaches (to prevent overfitting)

- Increase training data
- Regularization (penalizing model complexity)
- Hold-out & cross validation (unseen data to ensure generalization)
- Early stopping
- Prior knowledge (e.g., Bayesian prior)
- …

Prevent Overfitting

Fit the overall distribution instead of the training set.

- To estimate the generalization error, we need data unseen during model training.

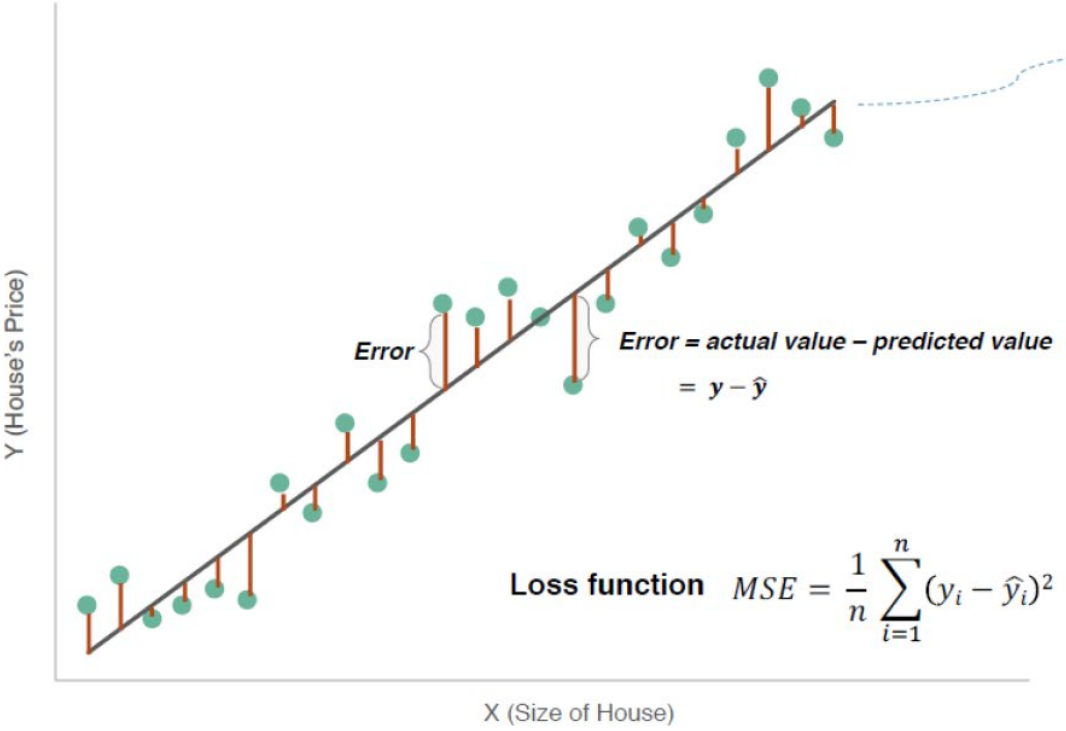
- **Data Splitting (Hold-Out):**
  - Training set (e.g., 50%)
  - **Validation set (e.g., 25%)**
  - Test **set (e.g., 25%)**

平时练习

模拟考试

高考!

Linear Regression



$$y = f_\theta(x) = \theta_0 + \sum_{j=1}^{d} \theta_j x_j = \theta^\top x$$

$$x = (1, x_1, x_2, \ldots, x_d)$$

$$J_\theta = \frac{1}{2N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))^2 \qquad \min_\theta J_\theta$$

Gradient Descent

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} - \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta}$$

Batch Gradient Descent

$$\theta_{\text{new}} = \theta_{\text{old}} + \eta \frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))x_i$$

Stochastic Gradient Descent

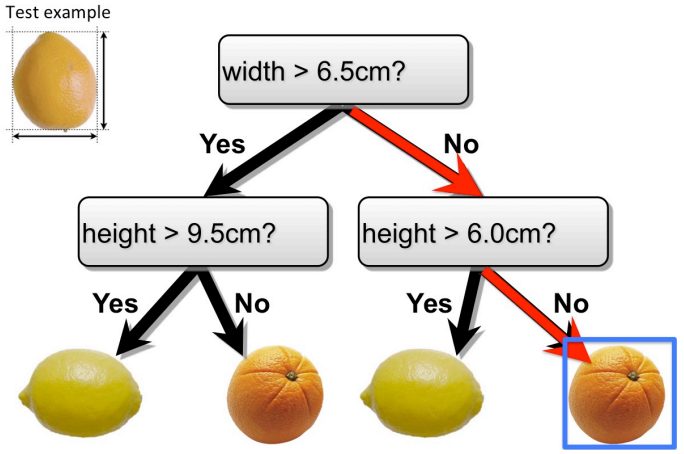
$$\theta_{\text{new}} = \theta_{\text{old}} + \eta(y_i - f_\theta(x_i))x_i$$

Training – (Build a Decision Tree)

top-down divide-and-conquer learning procedure

1.Construct a root node which contains the whole data set.
2.Selecting an attribute that benefit the task most according to some criterion.
3.Split the examples of the current node into subsets based on values of the selected attributes.
4.Create a child node for each subset and passes the examples in the subset to the node.
5.Recursively repeat step 2~4 until some stopping criterion is met.

ID	width	height	Type
1	5.2	8.0	lemons
2	6.7	9.8	lemons
3	7.2	7.5	orang
4	6.1	5.3	orang
5	4.1	6.5	lemons



Bayesian Networks for Classification

Input:

- a data sample $x = (x_1, x_2, \ldots, x_d)$
- a fixed set of classes $C = \{C_1, \ldots, C_j\}$.

Output:

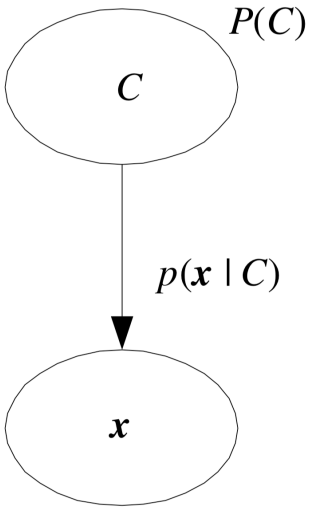
- the most probable class c ∈ C:

$$c_{\text{MAP}} = \arg\max_{c \in C} P(c|x)$$

$$= \arg\max_{c \in C} \frac{P(x|C)P(c)}{P(x)}$$

$$= \arg\max_{c \in C} P(x|c)P(c)$$

$$= \arg\max_{c \in C} p(x_1, x_2, \ldots, x_d|c)\, P(c)$$

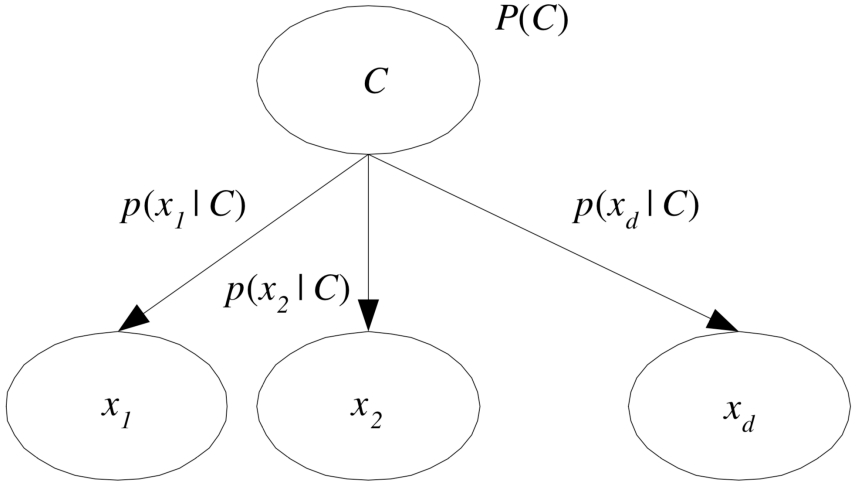


Naïve Bayes Independent Assumption

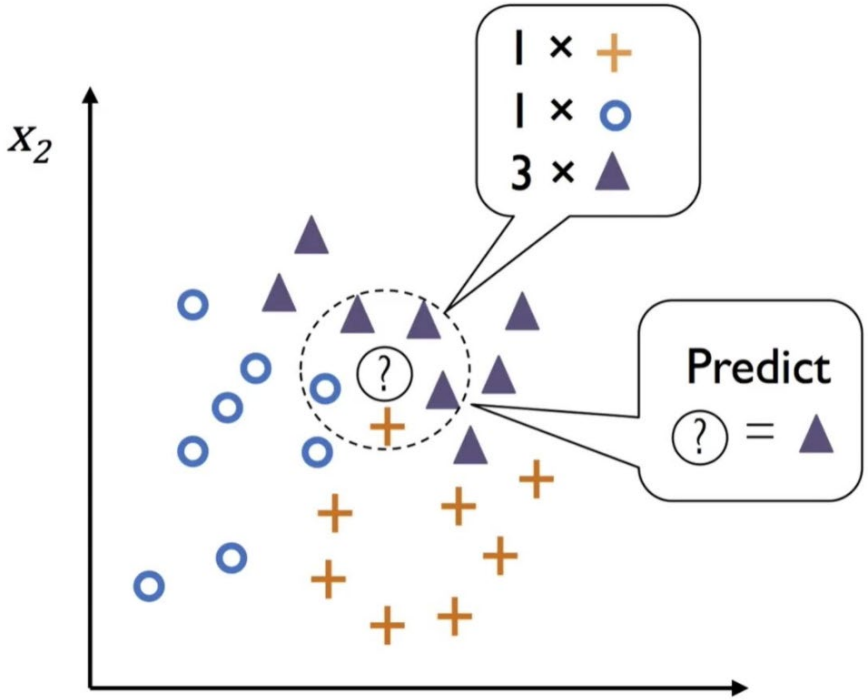
$$P(x_1, x_2, \ldots, x_d \mid c)$$

Conditional Independence: assume the input features x_j are **independent** given the class c

$$P(x_1,\ldots,x_n|c) = P(x_1|c)\bullet P(x_2|c)\bullet P(x_3|c)\bullet \ldots \bullet P(x_n|c)$$



K-Nearest Neighbors

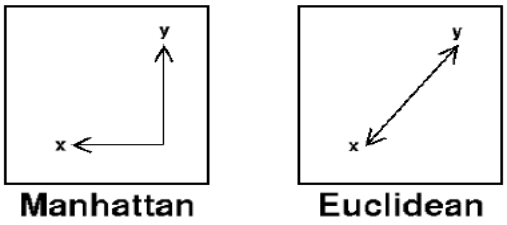


KNN: Pseudocode

- Step 1: Determine parameter *K* = number of nearest neighbors
- Step 2: Calculate the distance between the query-instance and all the training examples.
- Step 3: Sort the distance and determine nearest neighbors based on the *k*-th minimum distance.
- Step 4: Gather the category Y of the nearest neighbors.
- Step 5: Use simple majority of the category of nearest neighbors as the prediction value of the query instance.

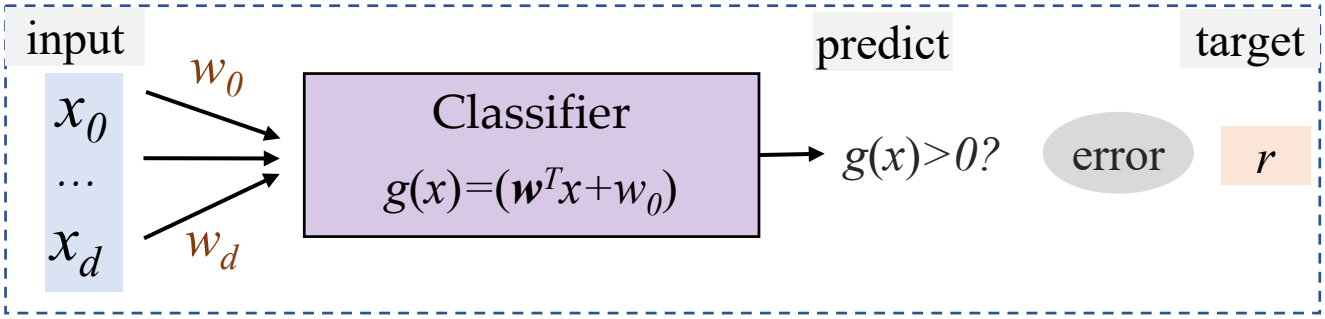
Euclidean Distance $dist = \sqrt{\sum_{k=1}^{p}(a_k - b_k)^2}$

Manhattan Distance $|x_1 - x_2| + |y_1 - y_2|$



Perceptron

A simple, naïve linear classifier.



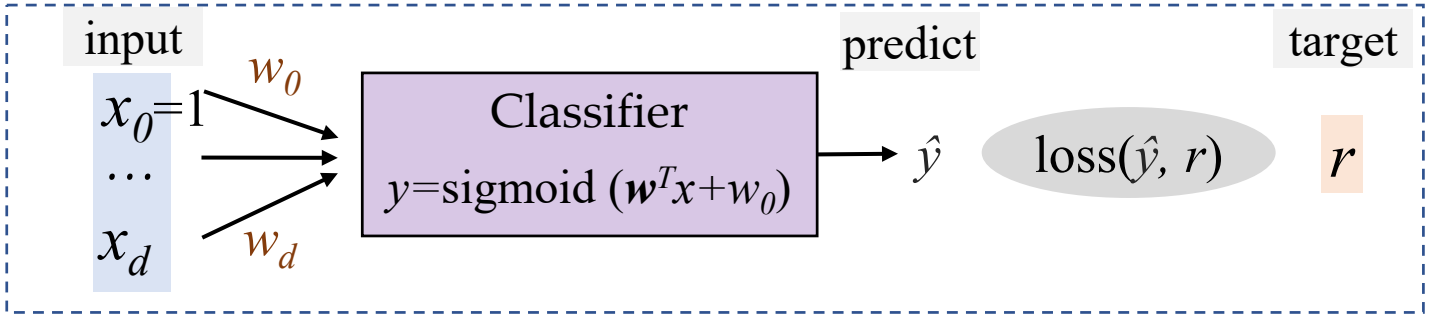
- Train:
  - estimate the parameters $\mathbf{w}$ and w_0 from data
- Test:
  - calculate $g(x) = (\mathbf{w}^T x+w_0)$ and choose C_1 if $g(x) > 0$ or choose C_2 if g(x)<0.

perceptron classifies data based on which **side** of the plane the new point lies on.

Logistic Regression

A classifier that estimates the decision boundary as a logistic function:

$$y = \text{sigmoid}\,(\boldsymbol{w}^{\mathrm{T}}x + w_0) = \frac{1}{1+\exp[-(\boldsymbol{w}^{T}x+w_0)]}$$



- Train:
  - estimate the parameters $\boldsymbol{w}$ and w_0 from data
- Test:
  - calculate $y = \text{sigmoid}\,(\boldsymbol{w}^{\mathrm{T}}x+w_0)$ and choose C_1 if $y > 0.5$ (y can be interpreted as a posterior probability).

Loss Function

$y \equiv \text{p}(C_1|x) = \text{sigmoid}(w^{\text{T}}x+w_0)$ is the estimated posterior probability of $x \in C_1$.

- For a given input x, the model outputs a probability y of $x \in C_1$. Let $r \in \{0, 1\}$ be the label of the real class (r =1: $x \in C_1$, r = 0: $x \in C_2$):
  - if r = 1: we aim to maximize $\log p(C_1|x) = \log y$, cost is $-\log y$
  - if r = 0: we aim to maximize $\log p(C_2|x) = \log (1-y)$, cost is $-\log (1-y)$
- Can write this succinctly as a cross-entropy loss:

$$\ell\,(\boldsymbol{w}, w_0 \,|\, x, r) = -\,r \log y - (1-r) \log (1-y)$$

nonzero only if r=1 nonzero only if r=0

$$\text{CE(p, q)} = \mathbb{E}_{x\sim p(x)}\left\{\log \frac{1}{q(x)}\right\} = -\sum_{x\in X} p(x)\log_2 q(x)$$

- Equivalent to maximize the likelihood:

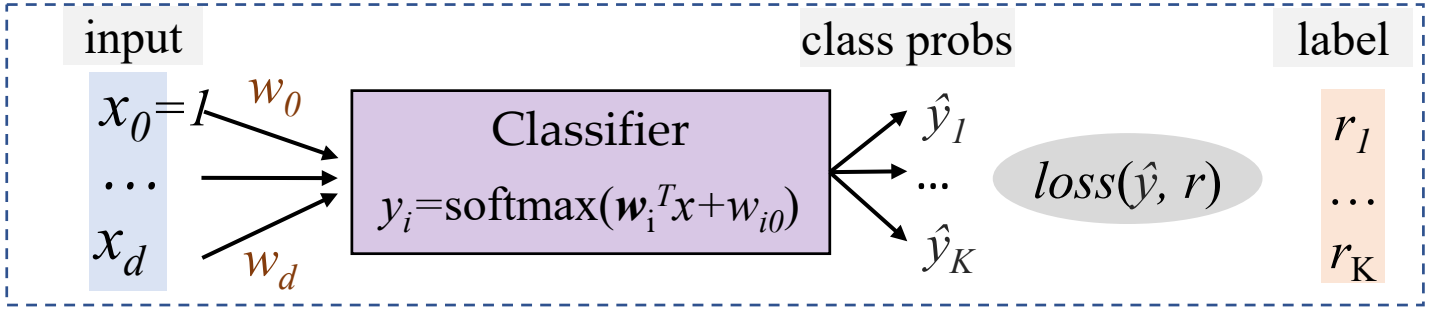
$$r \,|\, x \sim \text{Bernoulli}(y)$$

$$p(r|x) = y^{\text{r}}\,(1\text{-}y)^{(1\text{-}r)} = \begin{cases} y & \text{if } r=1 \\ 1-y & \text{if } r=0 \end{cases}$$

Softmax Regression

A classifier that estimates the decision boundaries as Softmax functions:

$$y_i = \text{softmax}\,(\boldsymbol{w}_i^{\mathrm{T}} x + w_{i0}) = \frac{\exp\,[\boldsymbol{w}_i^T x + w_{i0}]}{\Sigma_{j=1}^{K} \exp\,[\boldsymbol{w}_j^T x + w_{j0}]} \quad (i = 1,..,K)$$



- Train:
  - estimate the parameters $\boldsymbol{w}_i$ and w_{i0} (i=1:K) from data
- Test:
  - calculate $y_i = \text{softmax}\,(\boldsymbol{w}_i^{\mathrm{T}} x + w_0)$ and choose C_i if $y_i = \max\{y_{1:K}\}$ (y can be interpreted as a posterior probability).

Loss Function

$$\boldsymbol{r} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1}$$

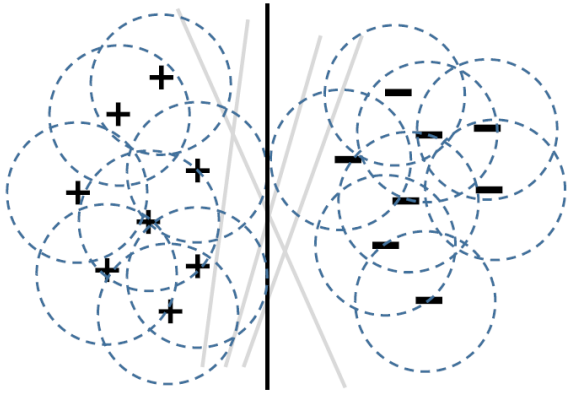
- For a given input $\boldsymbol{x}$, the model outputs a vector of class probabilities $\boldsymbol{y} = (y_1, \ldots y_K)$, and the label of target class is a one-hot vector $\boldsymbol{r} = (r_1, \ldots, r_K)$ (r_i=1:$x \in C_i$, r_i=0: $x \notin C_i$)
  - if $r_1 = 1$: we aim to maximize $\log p(C_1 | x) = \log y_1$, cost is $-\log y_1$
  - if $r_2 = 1$: we aim to maximize $\log p(C_2 | x) = \log y_2$, cost is $-\log y_2$
  - ……
- We can write this succinctly as a cross-entropy loss function:

$$L_{\text{CE}}(\boldsymbol{y}, \boldsymbol{r}) = -\sum_{i=1}^{K} r_i \log y_i = -\boldsymbol{r}^T(\log \boldsymbol{y})$$

where the *log* is applied elementwise.

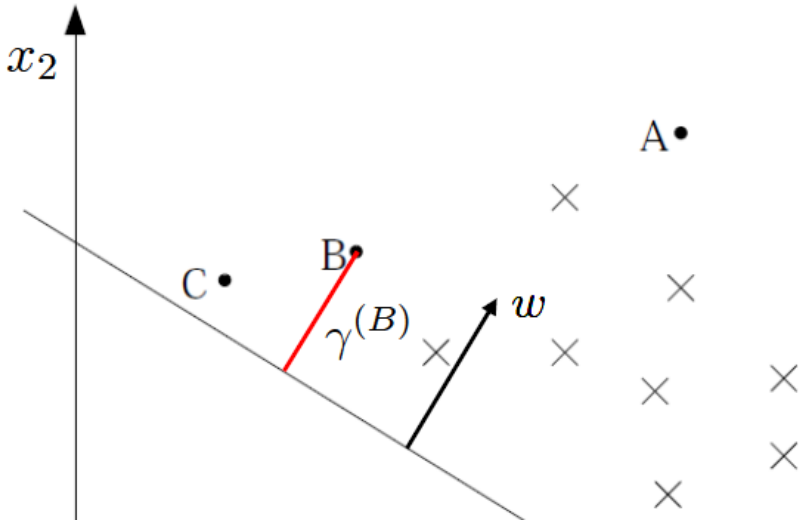
Support Vector Machines

The intuitive optimal decision boundary: the largest margin



$$h_{w,b}(x) = g(w^\top x + b)$$

$$g(z) = \begin{cases} +1 & z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



Functional margin

$$\hat{\gamma}^{(i)} = y^{(i)}(w^\top x^{(i)} + b)$$

Geometric margin

$$\gamma^{(i)} = y^{(i)}(w^\top x^{(i)} + b)$$

where $\|w\|^2 = 1$

Support Vector Machines

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{\|w\|} \right)^\top x^{(i)} + \frac{b}{\|w\|} \right)$$

Given a training set $S = \{(x_i, y_i)\}_{i=1...m}$

The smallest geometric margin $\gamma = \min_{i=1...m} \gamma^{(i)}$

Objective of an SVM

Find a separable hyperplane that maximizes the minimum geometric margin

$$\max_{\gamma,w,b} \ \gamma$$
$$\text{s.t.} \ \ y^{(i)}(w^\top x^{(i)} + b) \geq \gamma, \ \ i = 1, \ldots, m$$
$$\|w\| = 1 \quad \text{(non-convex constraint)}$$

$$\max_{\hat{\gamma},w,b} \ \frac{\hat{\gamma}}{\|w\|} \quad \text{(non-convex objective)}$$
$$\text{s.t.} \ \ y^{(i)}(w^\top x^{(i)} + b) \geq \hat{\gamma}, \ \ i = 1, \ldots, m$$

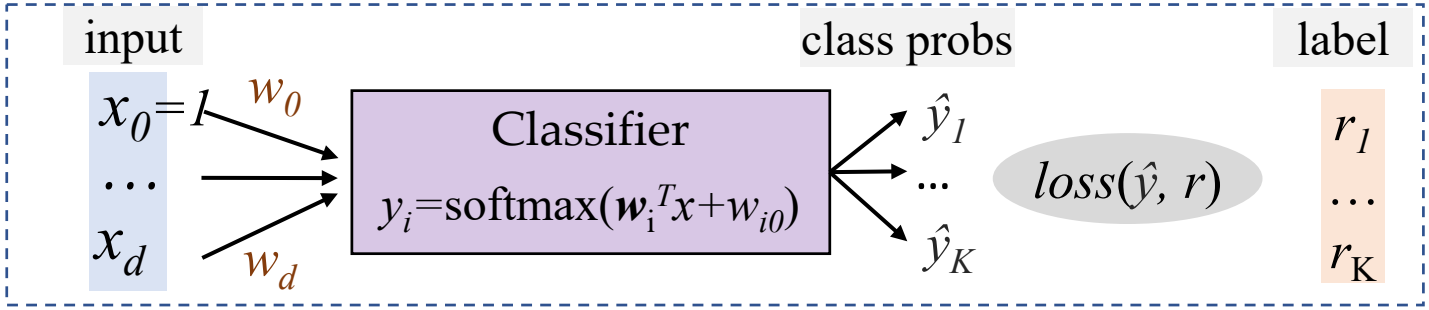
$$\max_{w,b} \ \frac{1}{\|w\|}$$
$$\text{s.t.} \ \ y^{(i)}(w^\top x^{(i)} + b) \geq 1, \ \ i = 1, \ldots, m$$

$$\min_{w,b} \ \frac{1}{2}\|w\|^2$$
$$\text{s.t.} \ \ y^{(i)}(w^\top x^{(i)} + b) \geq 1, \ \ i = 1, \ldots, m$$

Softmax Regression

A classifier that estimates the decision boundaries as Softmax functions:

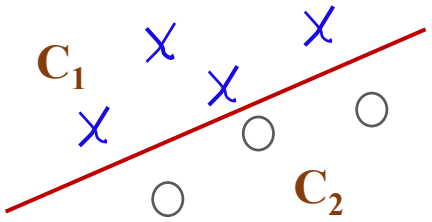
$$y_i = \text{softmax}\,(\boldsymbol{w}_i^{\mathrm{T}} x + w_{i0}) = \frac{\exp\,[\boldsymbol{w}_i^T x + w_{i0}]}{\Sigma_{j=1}^{K} \exp\,[\boldsymbol{w}_j^T x + w_{j0}]} \quad (i = 1,..,K)$$

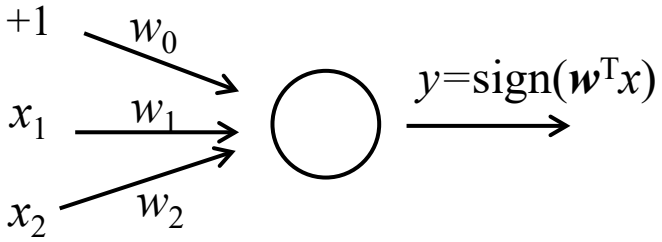


- Train:
  - estimate the parameters $\boldsymbol{w}_i$ and w_{i0} (i=1:K) from data
- Test:
  - calculate $y_i = \text{softmax}\,(\boldsymbol{w}_i^{\mathrm{T}} x + w_0)$ and choose C_i if $y_i = \max\{y_{1:K}\}$ (y can be interpreted as a posterior probability).

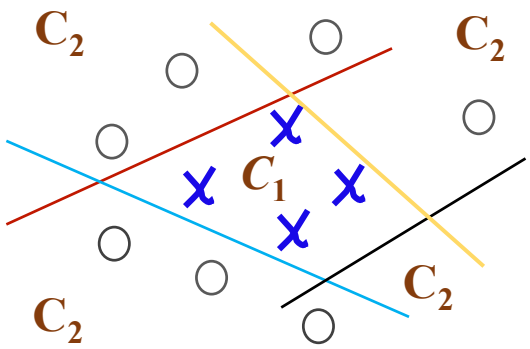
The Magic of Hidden Layers !

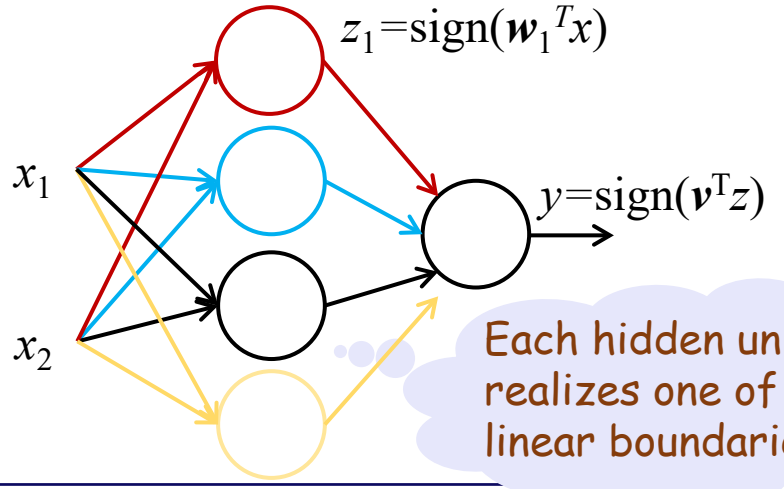
- **But,** adding hidden layer(s) (internal presentation) allows to learn a mapping that is not constrained by linear separability.





decision boundary: $x_1w_1 + x_2w_2 + w_0 = 0$

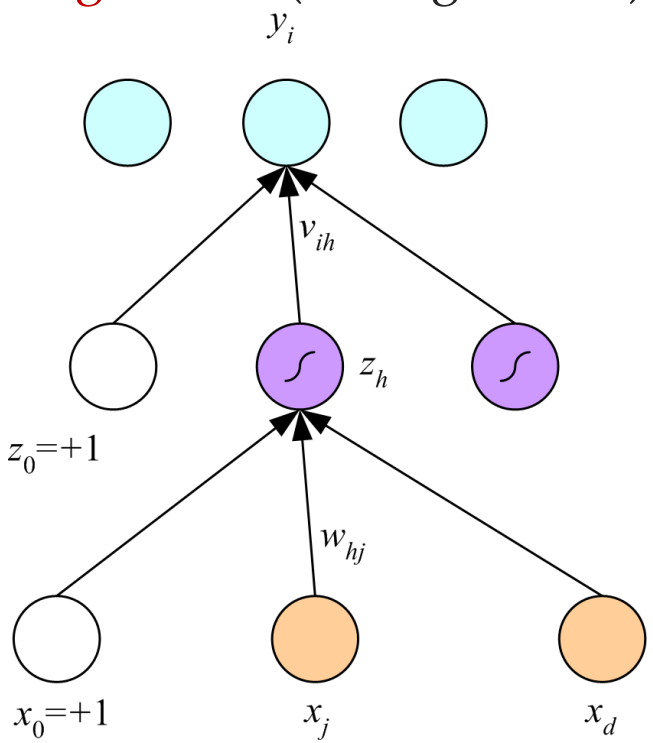




Each hidden unit realizes one of the linear boundaries

Multilayer Perceptrons

- A multilayer perceptron (MLP) has a hidden layer between the input and output layers.
- Can implement nonlinear discriminants (for classification) and regression (for regression) functions.



Output a linear function in the new H-dim space.

$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

Hidden a **nonlinear** transformation from the d-dim input space to the H-dim space spanned by the hidden units.

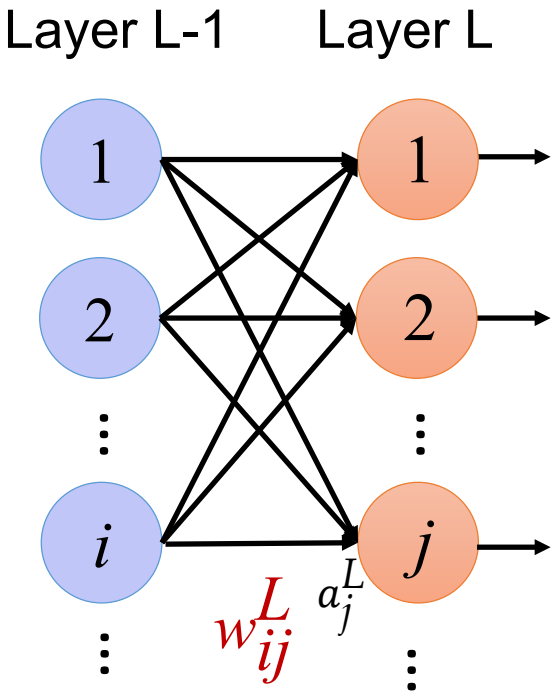
$$z_h = \text{sigmoid}(\mathbf{w}_h^T \mathbf{x}) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

Input performs no computation.

Perspective from Error-Propagation

$$\frac{\partial l}{\partial w_{ij}^{L}} = \frac{\partial a_j^{l}}{\partial w_{ij}^{l}} \frac{\partial l}{\partial a_j^{L}}$$

Error signal



$$\begin{cases} z_i^{L-1} & L > 1 \\ x_i & L = 1 \end{cases}$$

δ_j^{L}

Forward Pass

$$z^1 = W^1 x$$
$$a^1 = \sigma(z^1)$$
$$\cdots\cdots$$
$$z^{l-1} = W^{l-1} a^{l-2}$$
$$a^{l-1} = \sigma(z^{l-1})$$

Backward Pass

$$\delta^L = \sigma'(z^L) \bullet \nabla_y Loss$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$
$$\cdots\cdots$$
$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$
$$\cdots\cdots$$

The Algorithm

1. Forward propagation:

- apply the input vector to the network and evaluate the activations of all hidden and output units.

$$a_j = \sum_i w_{ji} z_i \qquad z_j = \sigma(a_j)$$

2. Backward propagation:

- evaluate the derivatives of the loss function with respect to the weights (errors).
- errors are propagated backwards through the network. $\delta_j^L = \frac{\partial l}{\partial a_j^L}$

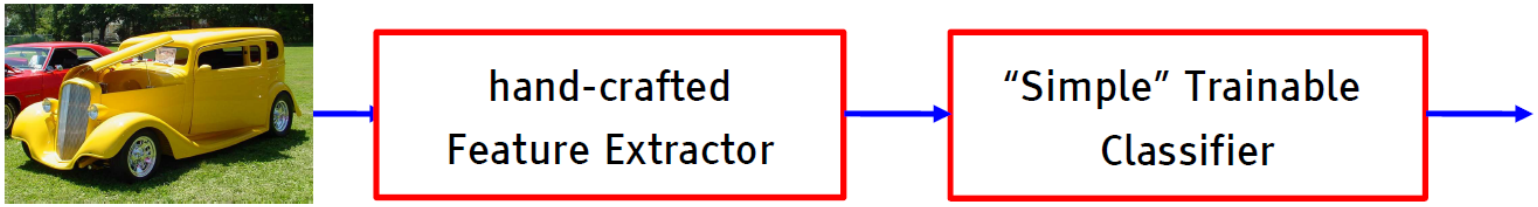
3. Parameter update:

- the evaluated derivatives (errors) are then used to compute the adjustments to be made to the parameters.

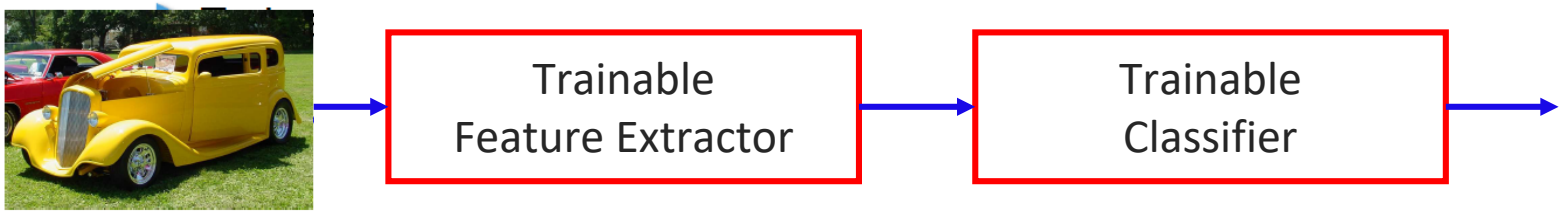
$$w'_{ij} = w_{ij} + \eta \delta_j \sigma'_j(a_j) z_i$$

Deep Learning = Learning Representations/Features of Data

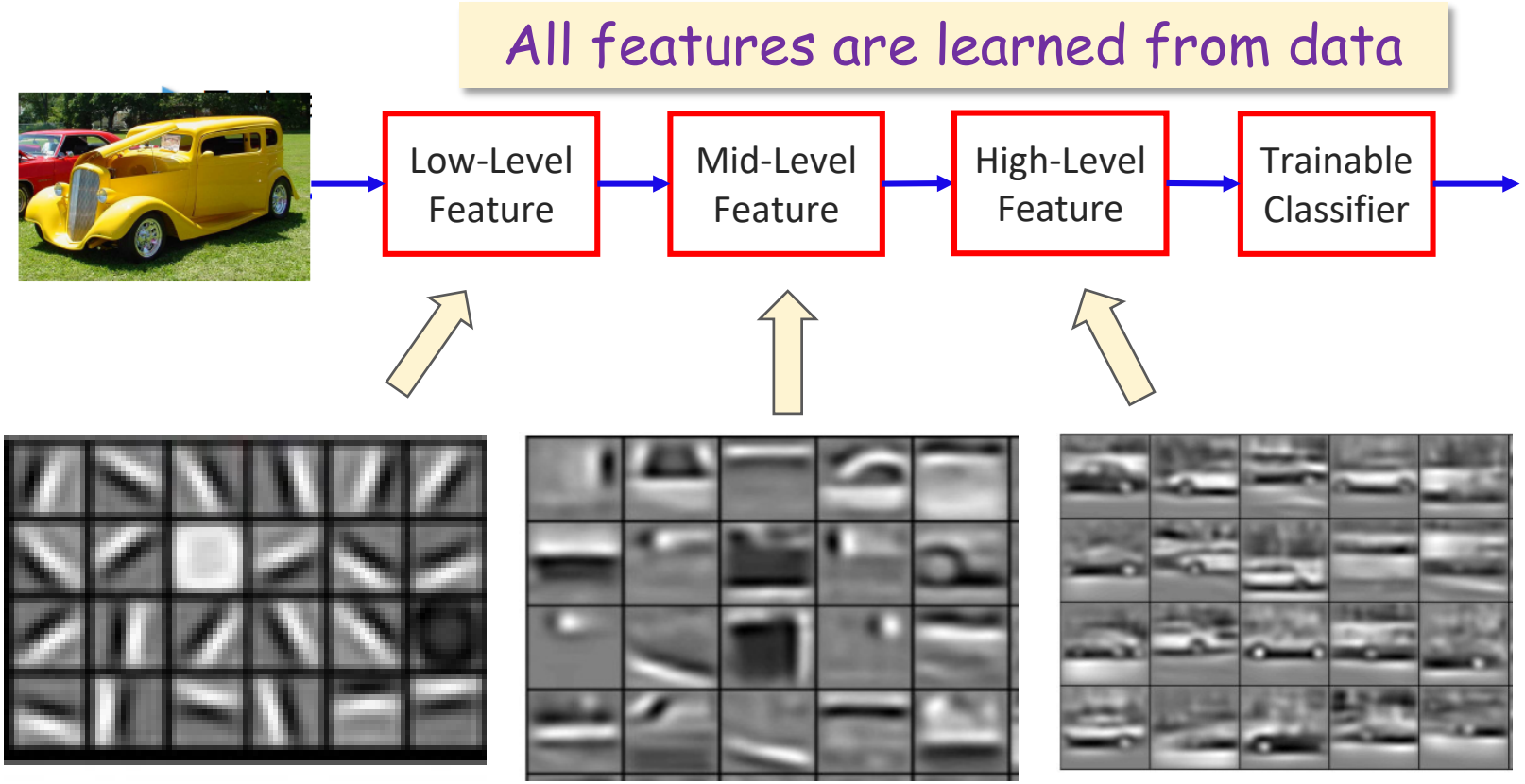
- The traditional model of pattern recognition (since the late 50's)
  - **fixed/engineered** features + **trainable** classifier



- Deep Learning/ Feature learning / End-to-end Learning
  - **trainable** features + **trainable** classifier



Deep Learning = Learning Hierarchical Representations



How to obtain word embeddings?

- A word can be understood by its context

"A bottle of ***tesgüino*** is on the table"
"Everybody likes ***tesgüino***"
"***Tesgüino*** makes you drunk"
"We make ***tesgüino*** out of corn"

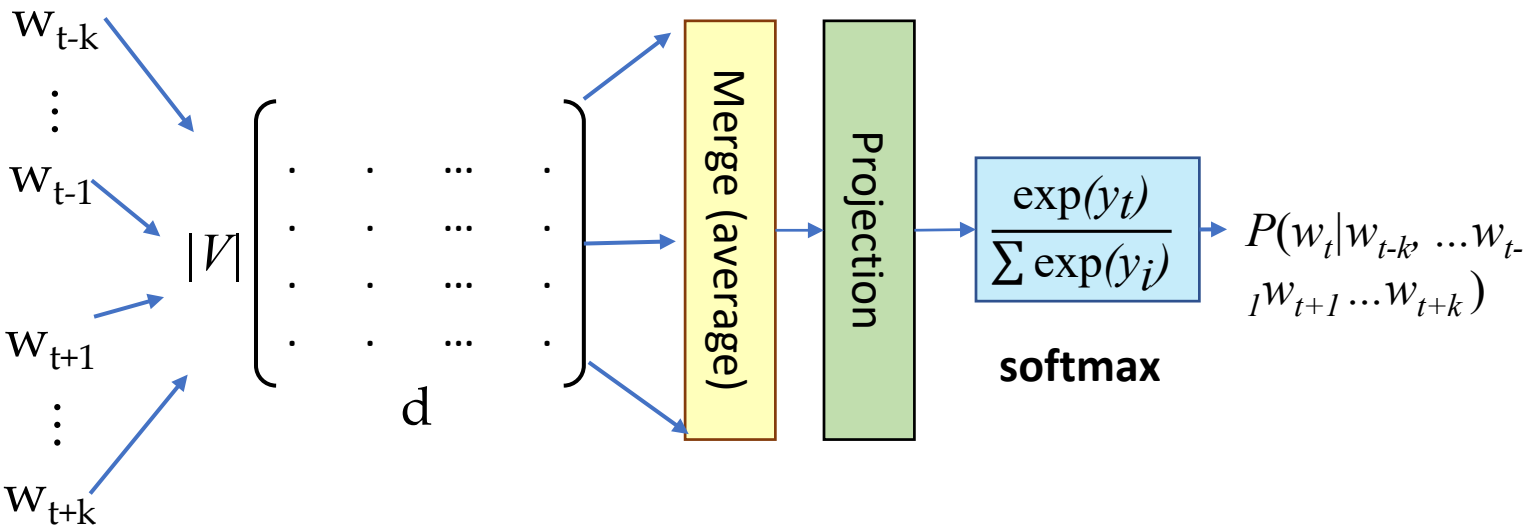
"You shall know a word by the company it keeps" (J. R. Firth 1957)

From context words we can guess tesgüino means an alcoholic beverage such as beer.

- **Intuition for an algorithm:**

Two words are similar if they have similar word contexts

Architecture: the big picture



Language Models

- A probabilistic model of how likely a given string appear in a given "language".
- For a given sequence $x = (w_1, w_2, \ldots, w_N)$. A language model can be defined as:

$$p(x) = p(w_1, w_2, \ldots, w_N)$$

Example:

P_1=P("我爱机器学习")
P_2=P("我爱学习机器")
P_3=P("机器我爱学习")
P_4=P("爱我机学习器")

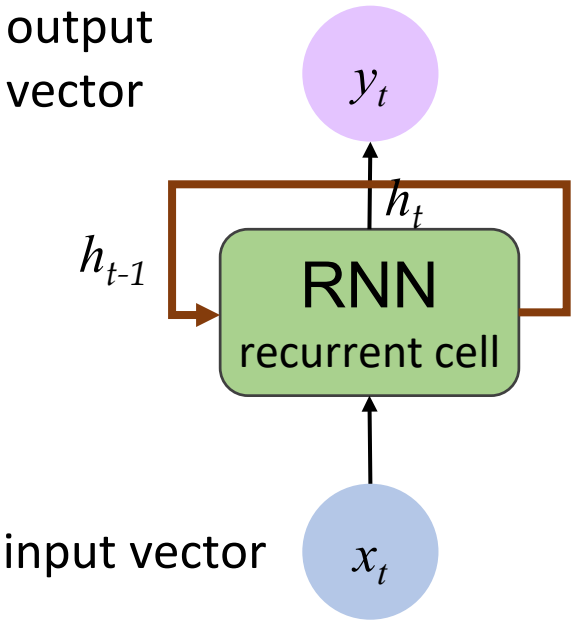
Chinese: $P_1 > P_2 > P_3 > P_4$

- Applications:

message suggestion; document generation; spelling correction; machine translation; speech recognition;…

我爱机器学__?__

Recurrent Neural Network (RNN)



Apply a recurrent relation at every time step to process a sequence:

$$h_t = f_W(h_{t-1}, x_t)$$

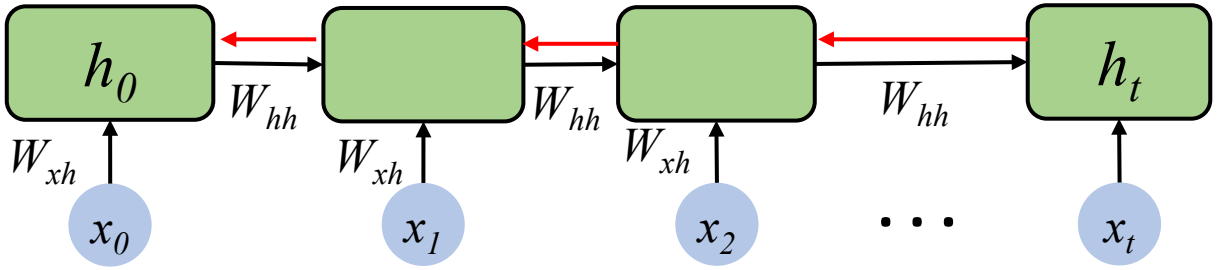
cell state — function parameterized by W — old state — Input vector at time step t

Note: the same function and set of parameters are used at every time step

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed.

The Problems of Standard RNNs

- **Gradient Flow of Standard RNNs:**



Computing the gradient w.r.t. h_0 involves many factors of W_{hh} + repeated gradient computation!

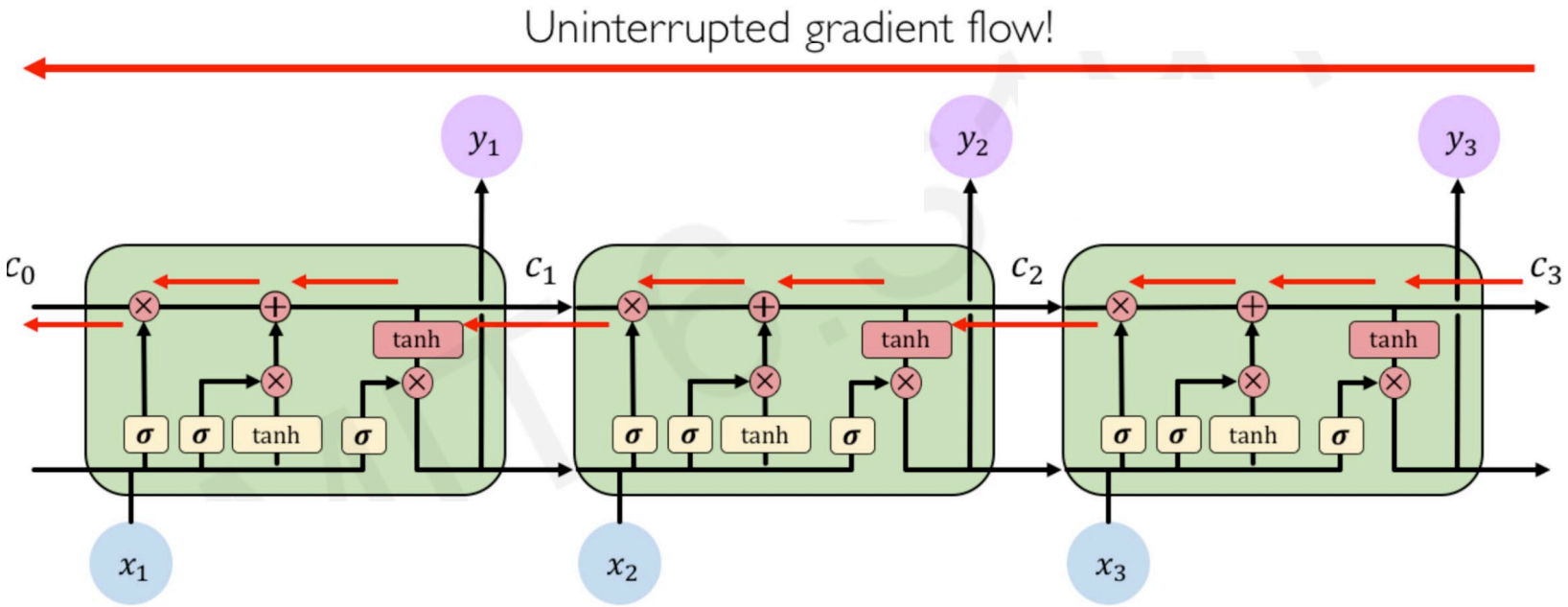
Many values > 1:
Exploding gradients

Gradient clipping to scale big gradients

Many values <1:
Vanishing gradients

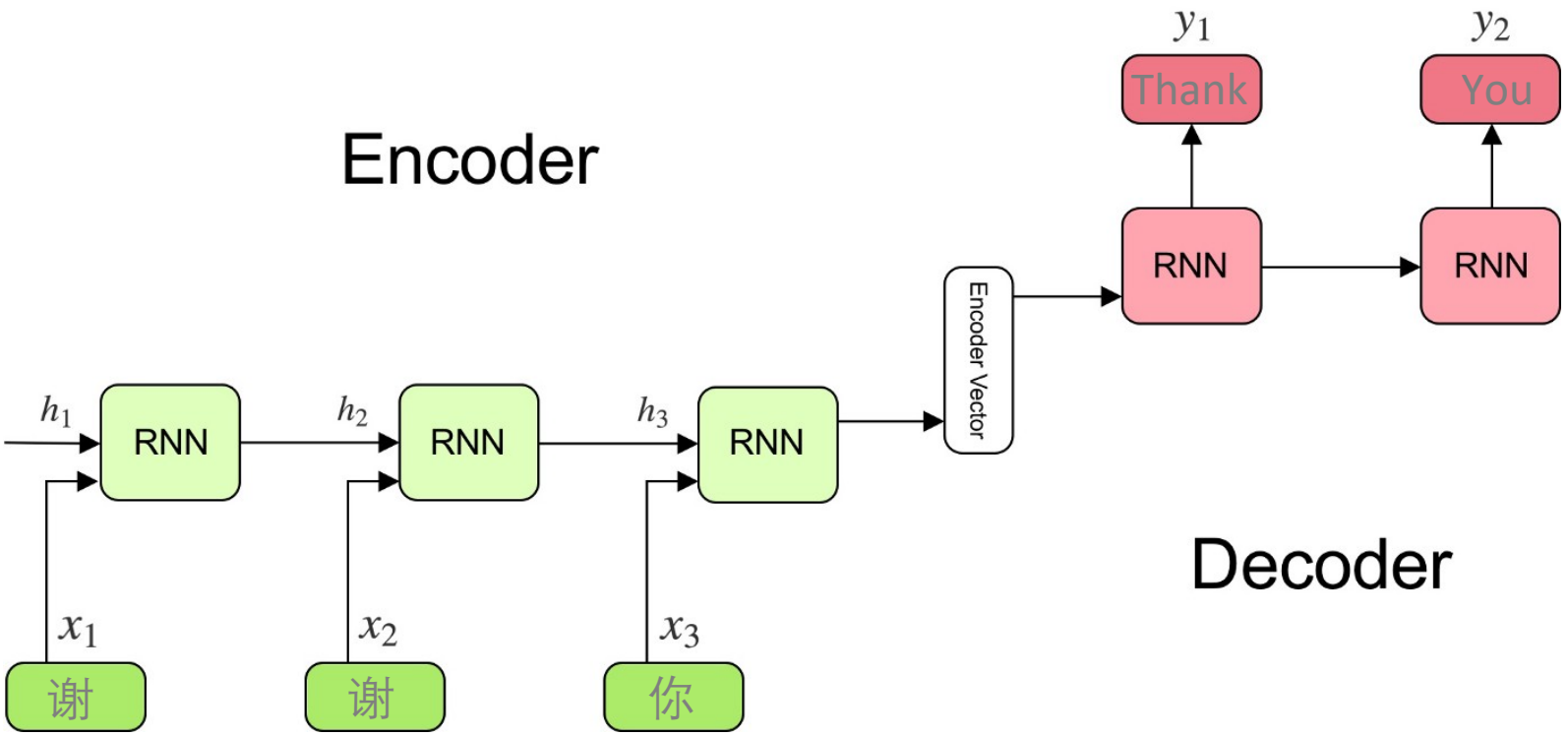
Gradient clipping to 0 gradients

LSTM Gradient Flow



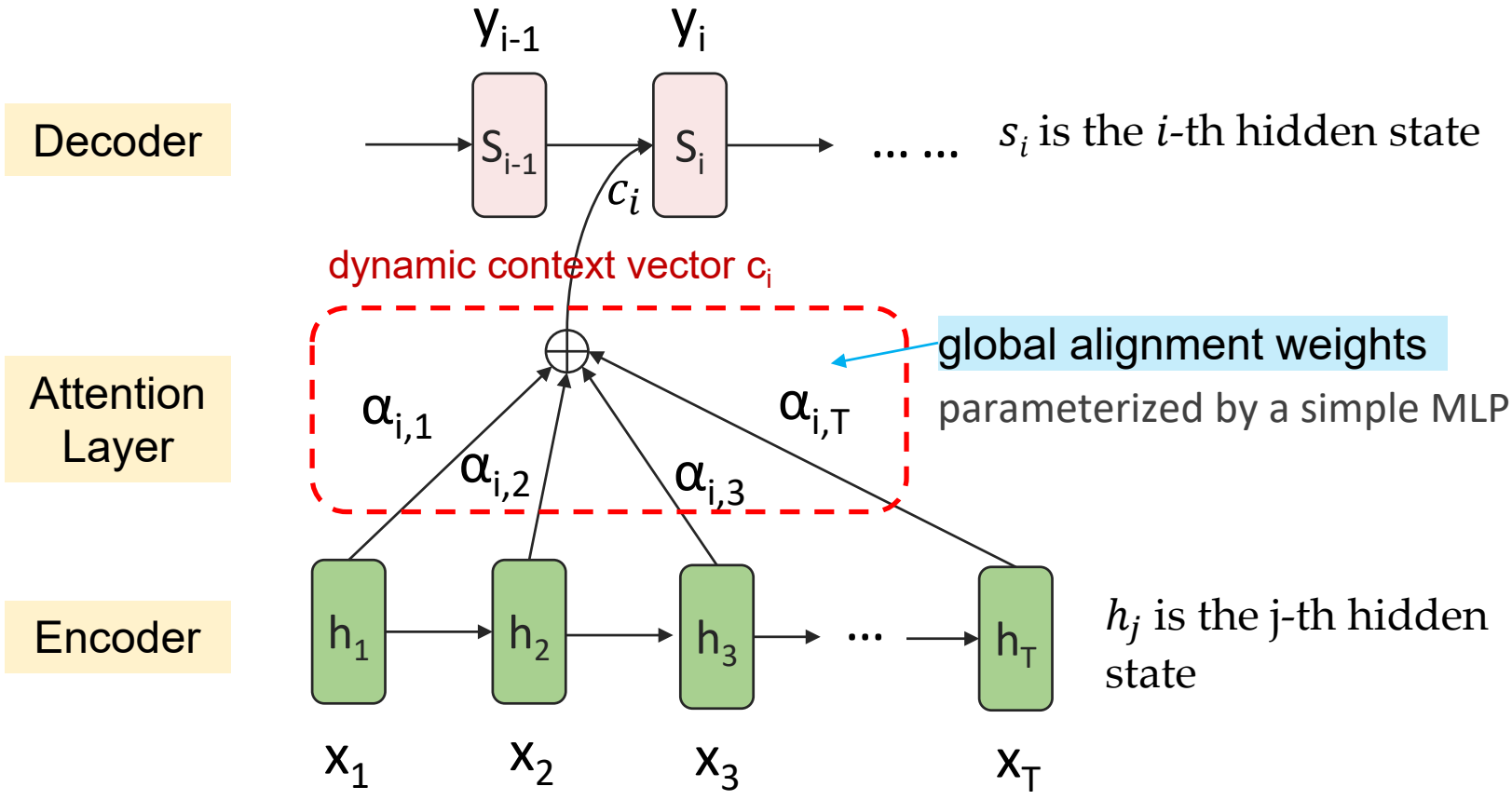
RNN Encoder-Decoder

- given x = $(x_1, \cdots, x_{T_x})$, generate y = $(y_1, \cdots, y_{T_y})$



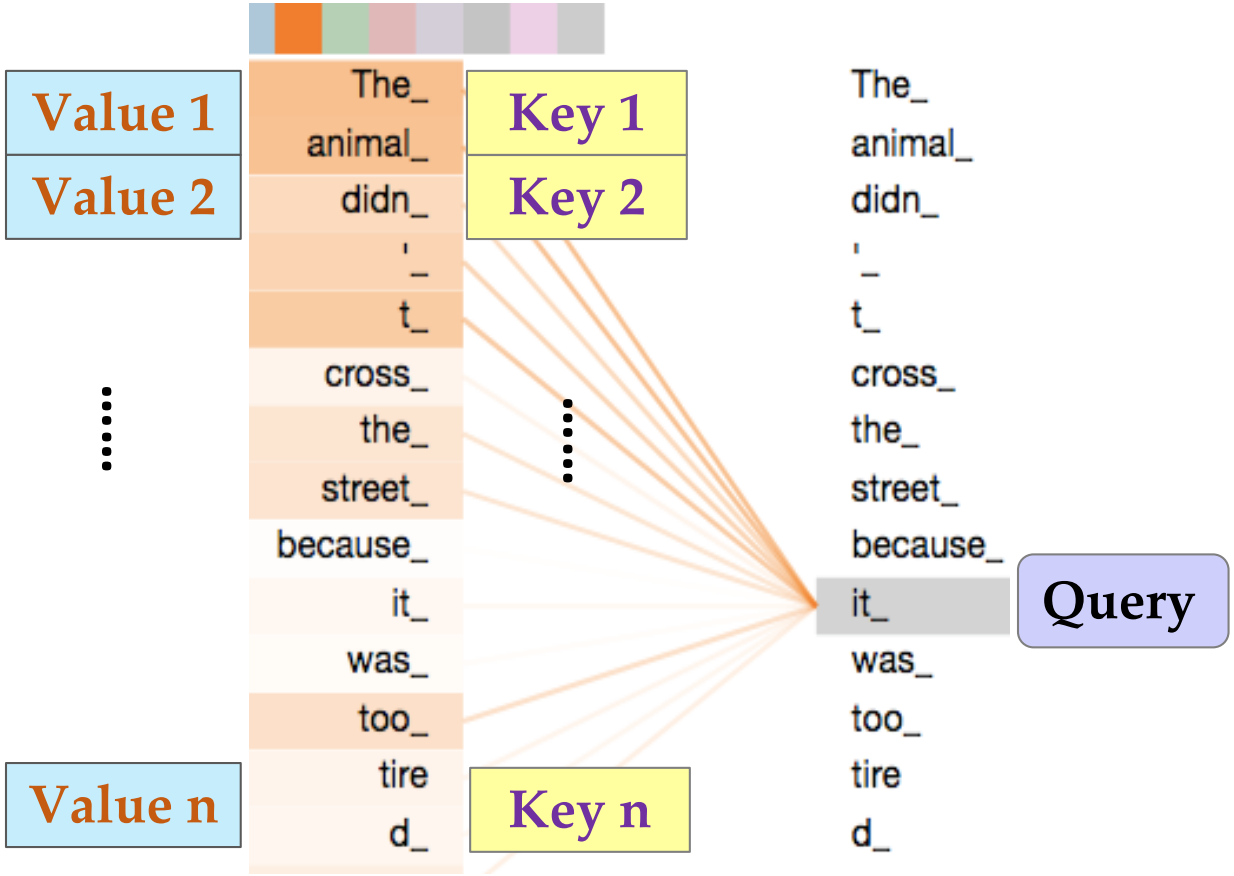
Sequence-to-Sequence with Attention

The decoder dynamically pays attention to different tokens in the source sentences during decoding.



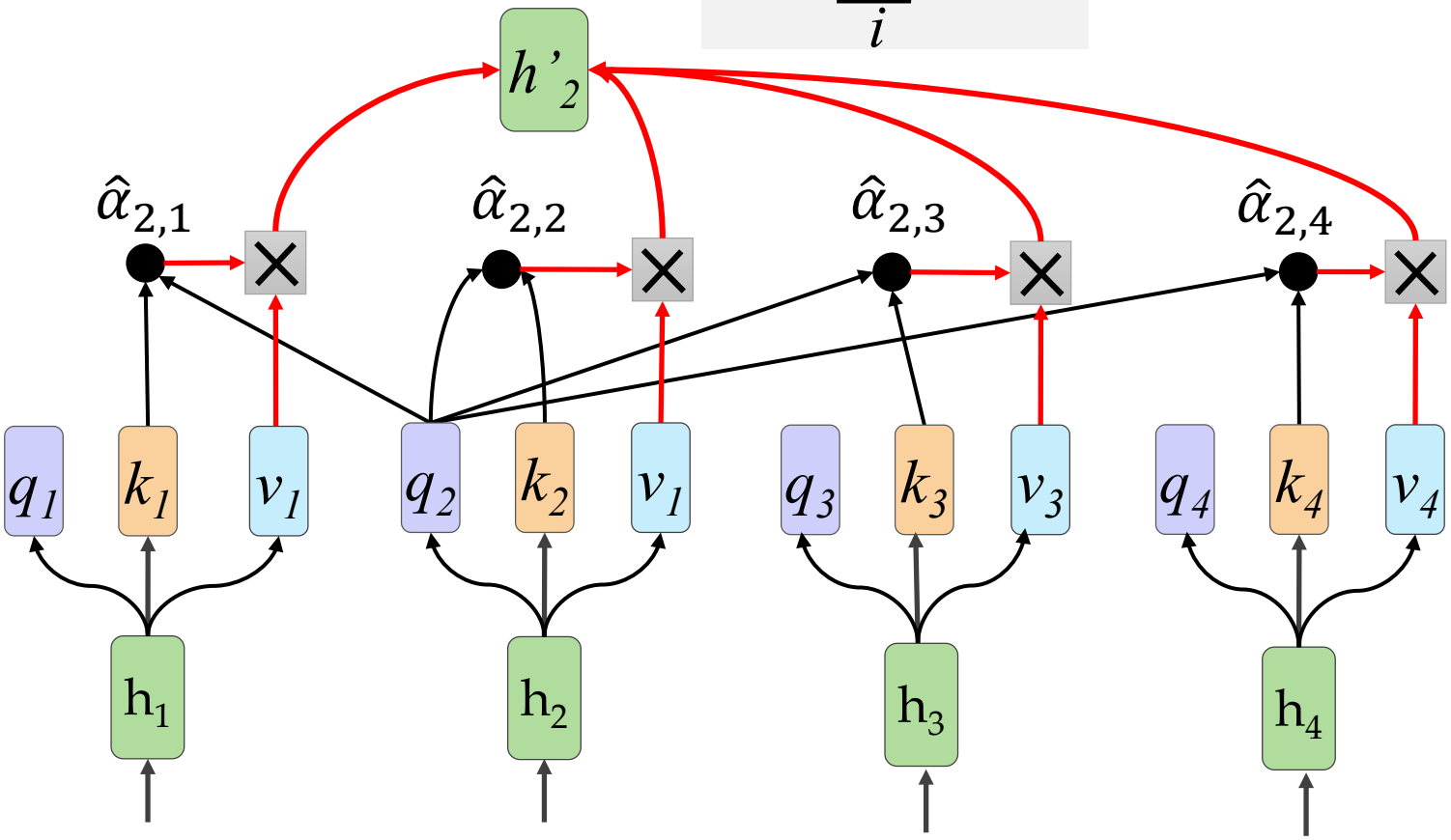
Self-Attention: The Idea

- Let each word pay attention to all other words.

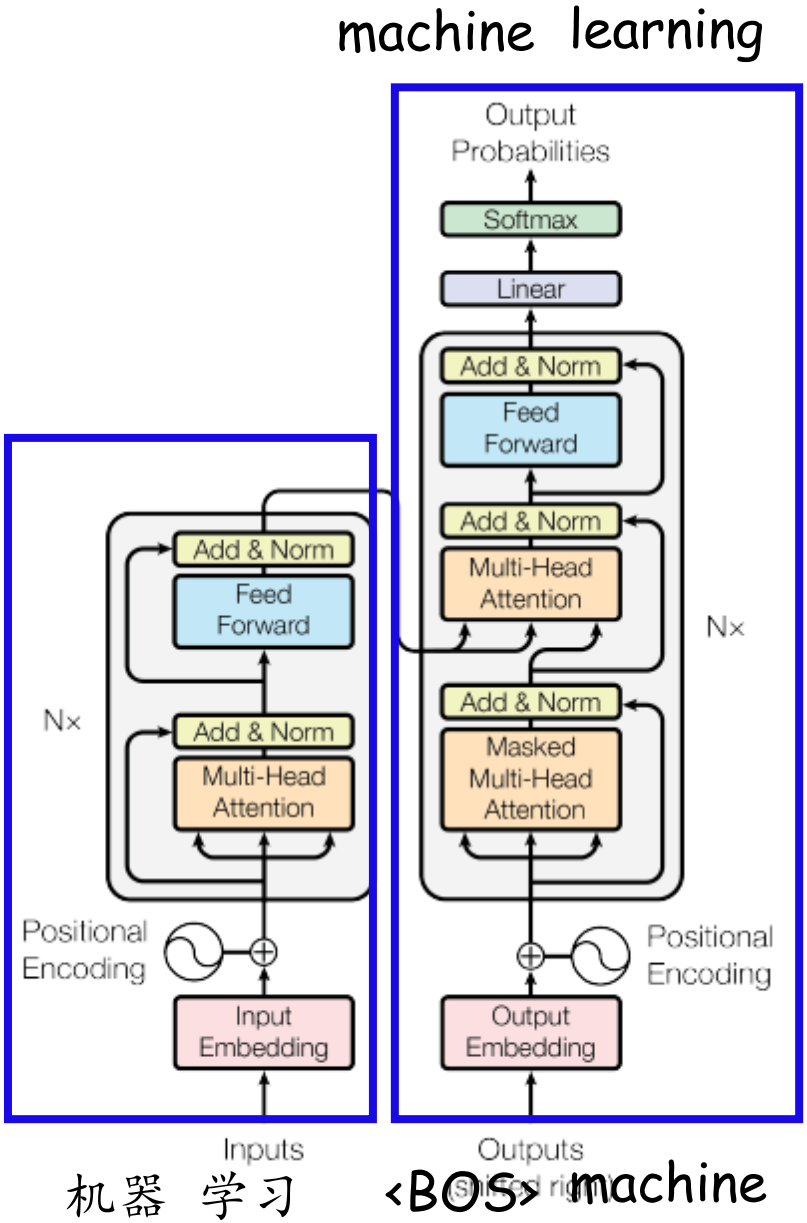


Step 4: Summarize Values According to Attention Weights

$$h'_2 = \sum_i \hat{\alpha}_{2,i} v_i$$



Transformer

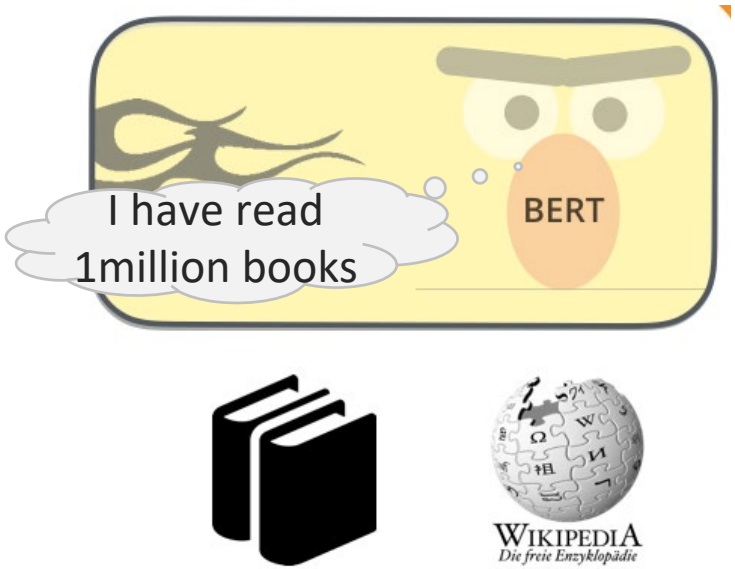


Encoder

Decoder

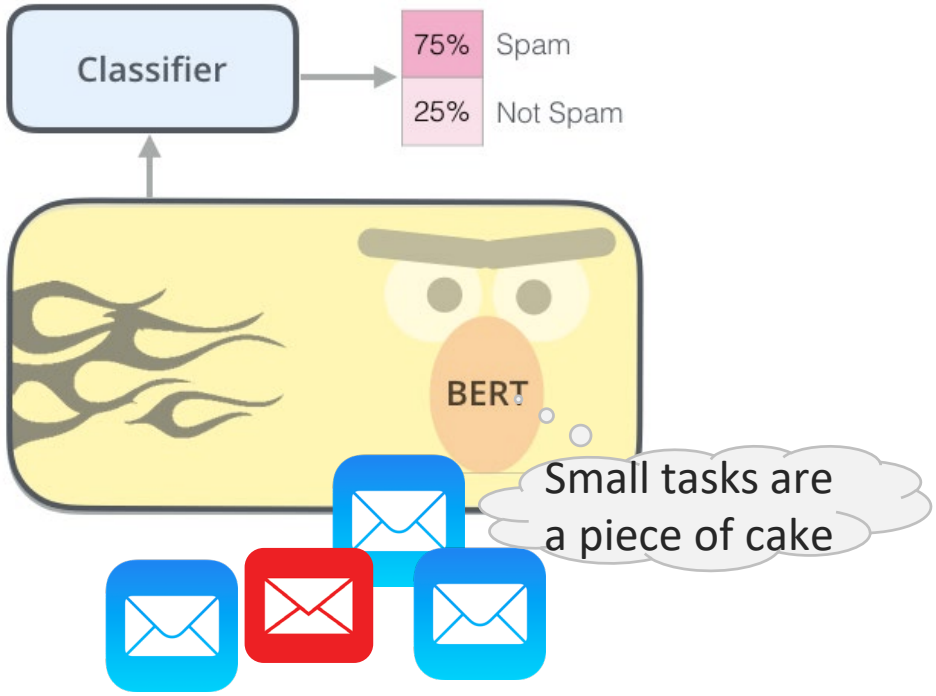
BERT

Unsupervised training on large amounts of text (e.g., books, Wikipedia, etc)



Phase 1: Pre-Training

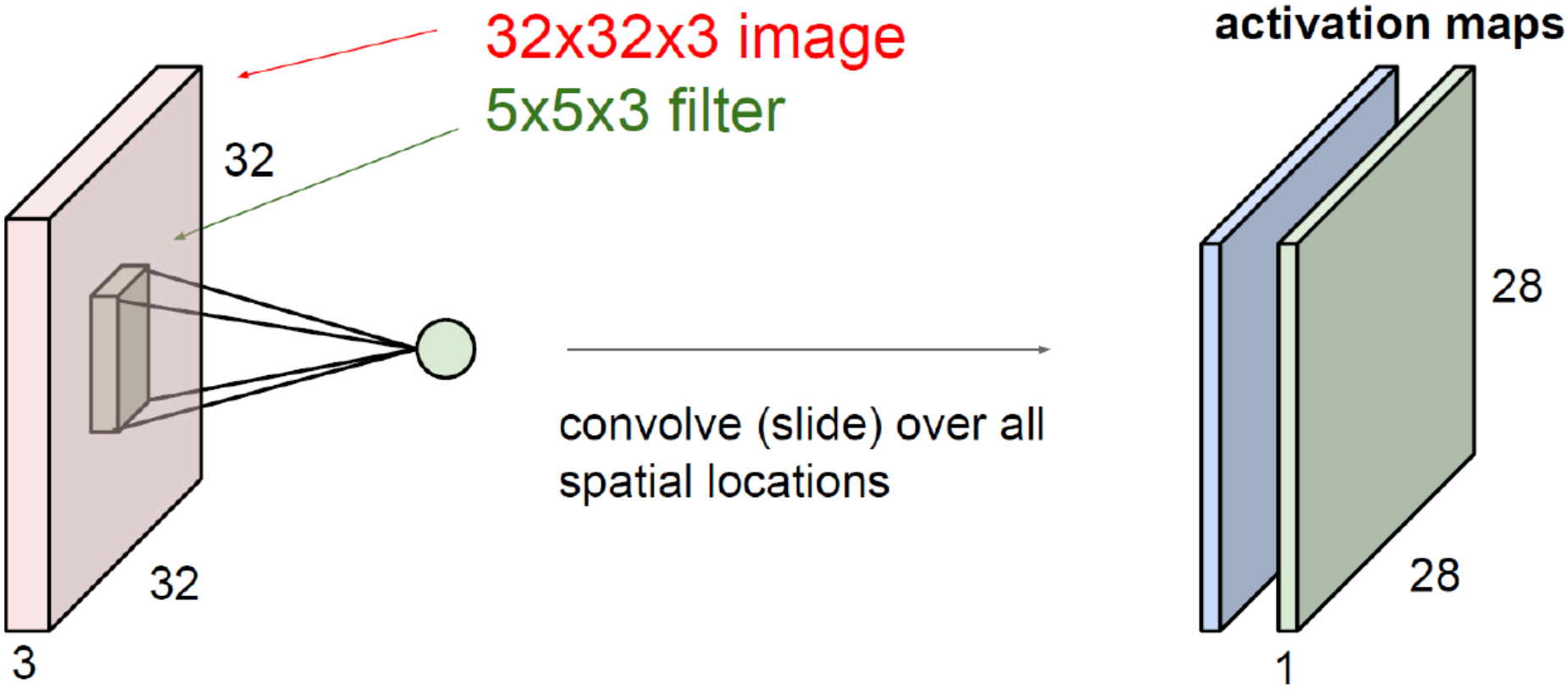
Supervised training on a specific task with a labeled dataset. (e.g., spam detection)



Phase 2: Fine-Tuning

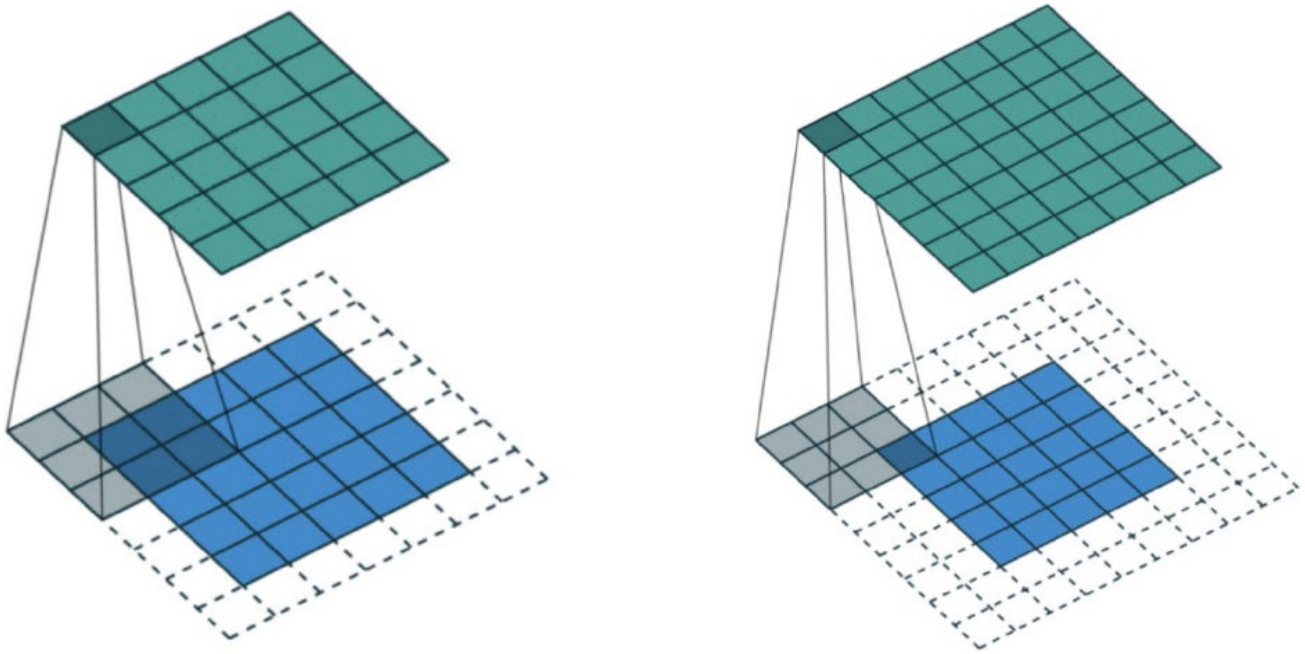
Convolutional Neural Networks

Convolution and filters



Convolutional Neural Networks

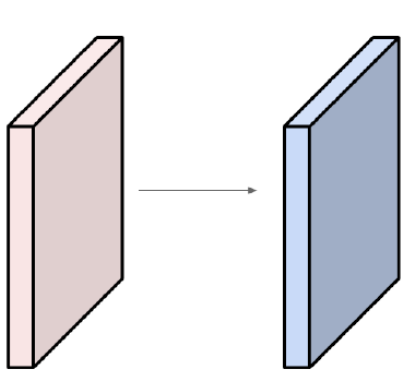
(Zero-)Padding refers to the process of symmetrically adding zeroes to the input matrix. It's a commonly used modification that allows the size of the input to be adjusted to our requirement.



Convolutional Neural Networks

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



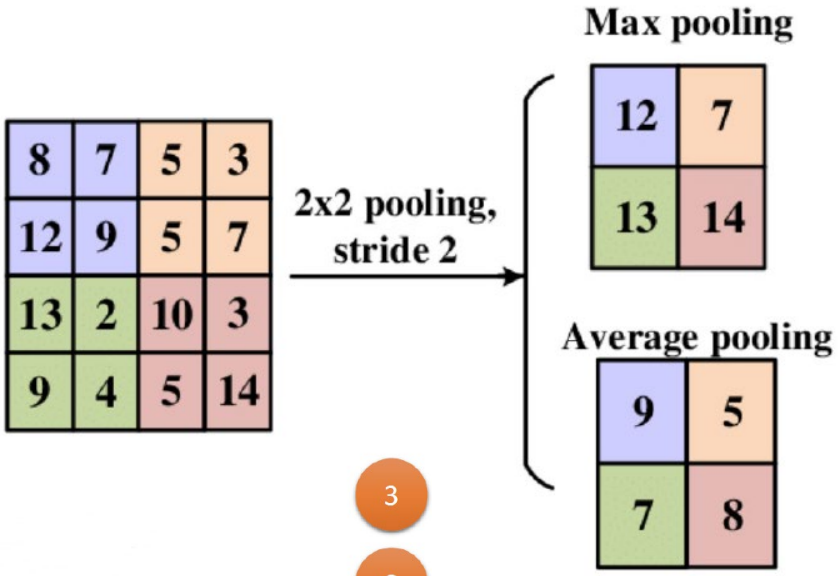
Convolutional Neural Networks

Summary. To summarize, the Conv Layer:

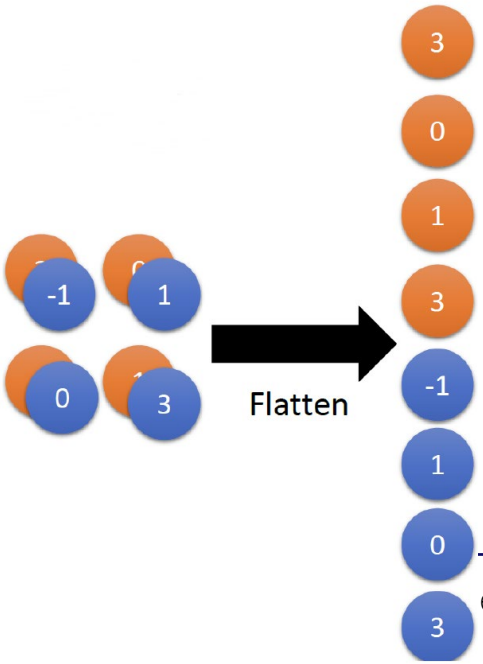
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolutional Neural Networks

Pooling

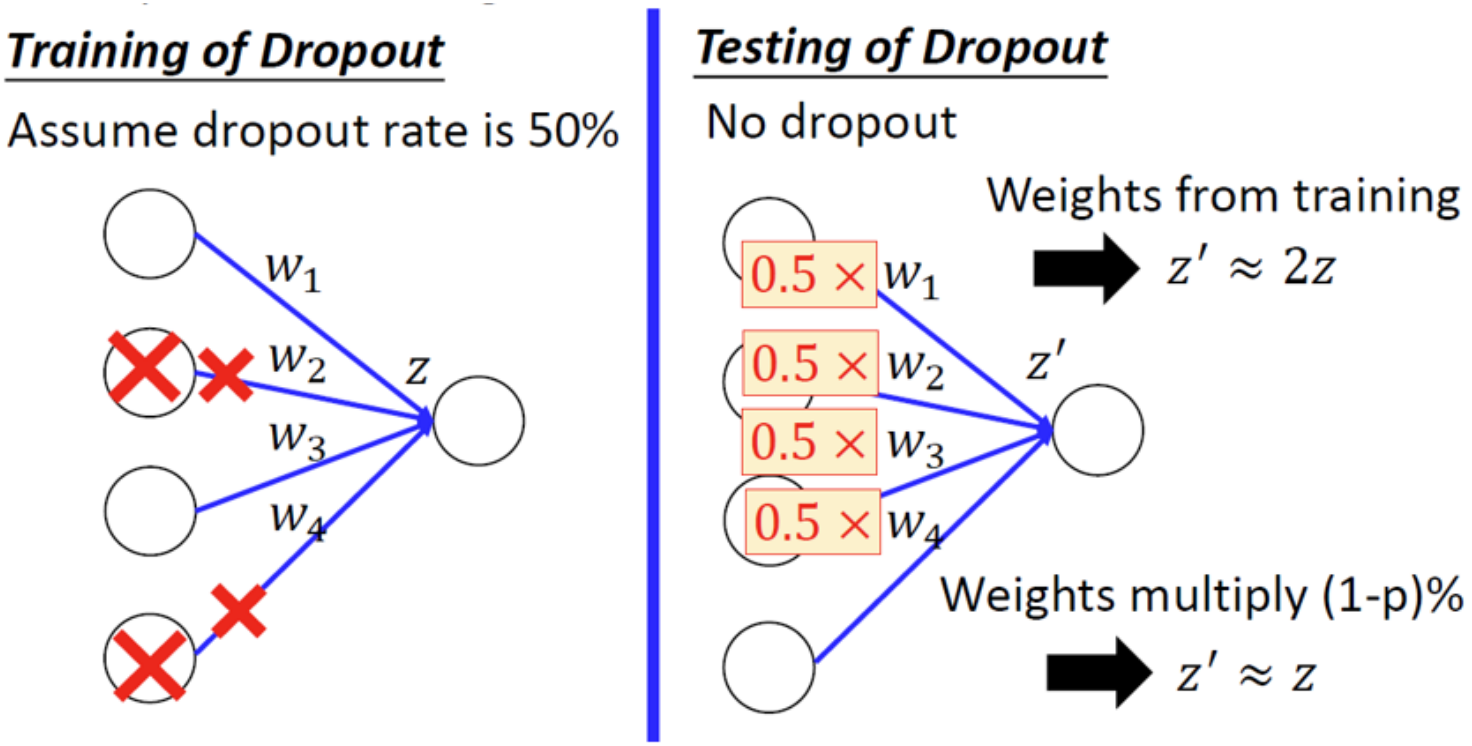


Flatten



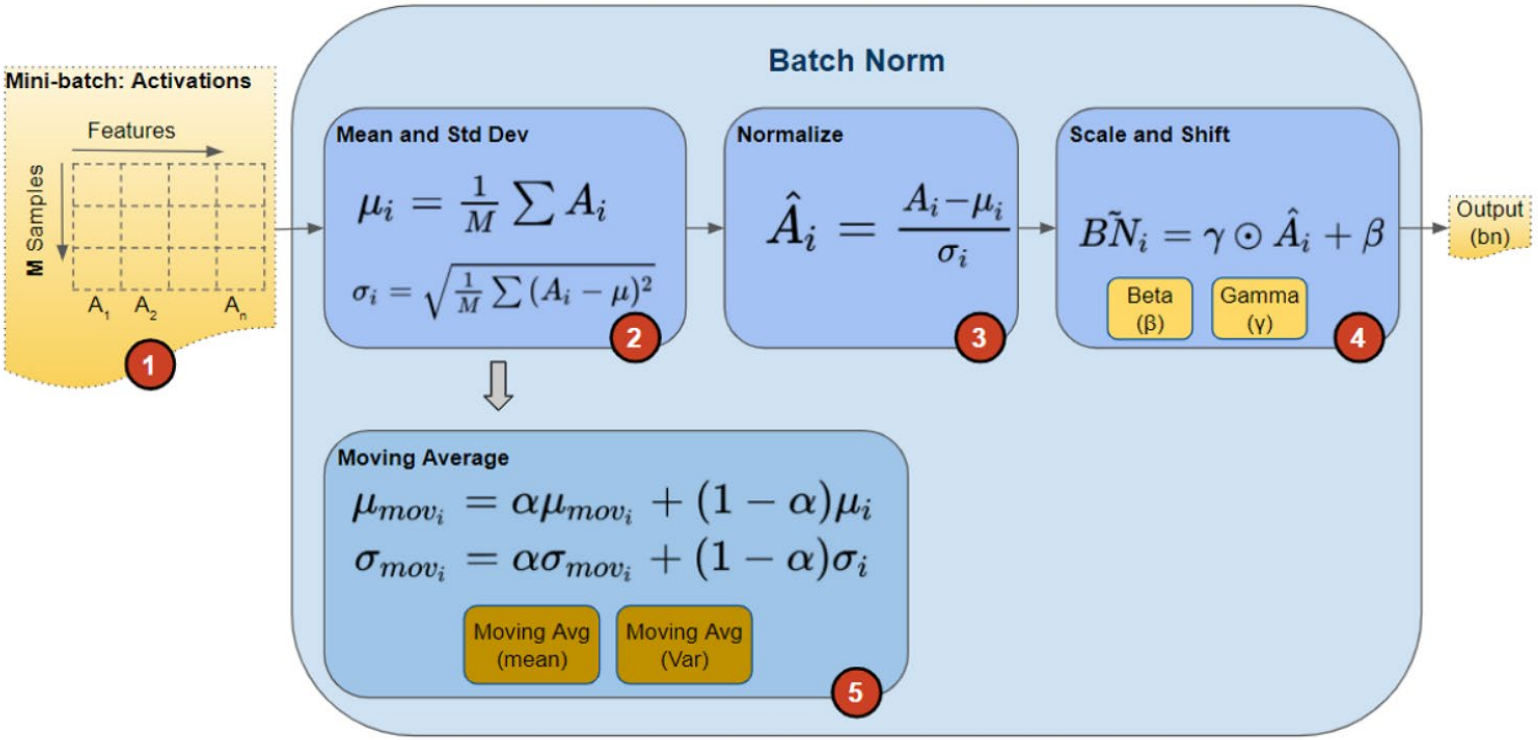
Convolutional Neural Networks

Dropout



Convolutional Neural Networks

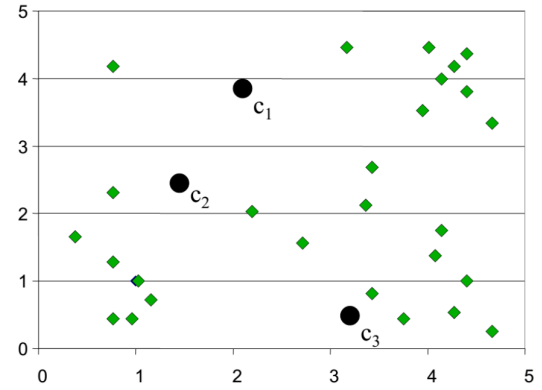
Batch Normalization



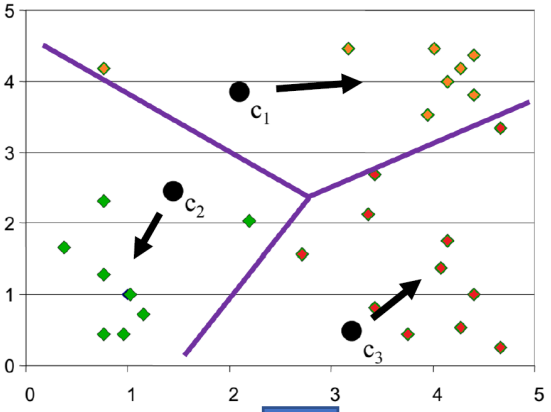
Unsupervised Learning

K-means clustering

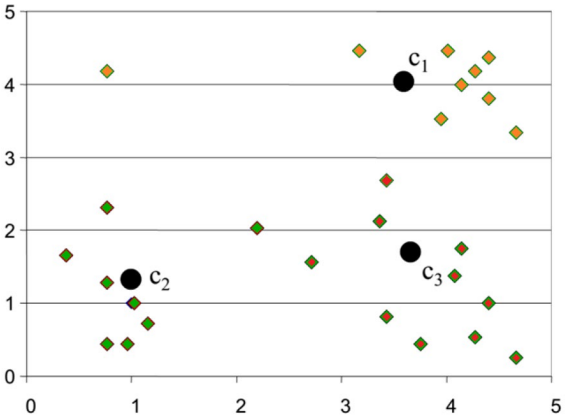
Randomly initialize the cluster centers (synaptic weights)



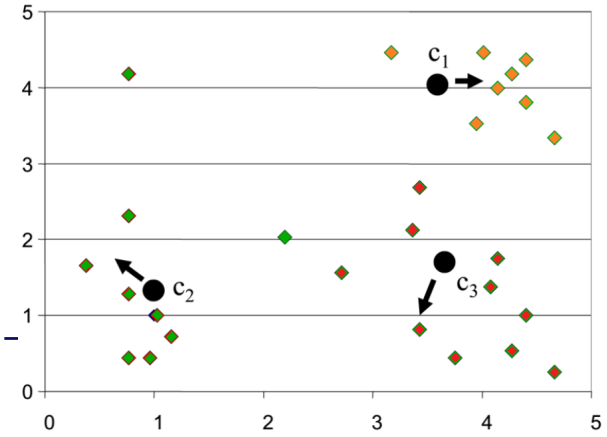
Re-estimate cluster centers (adapt synaptic weights)



Result of first iteration



Second iteration

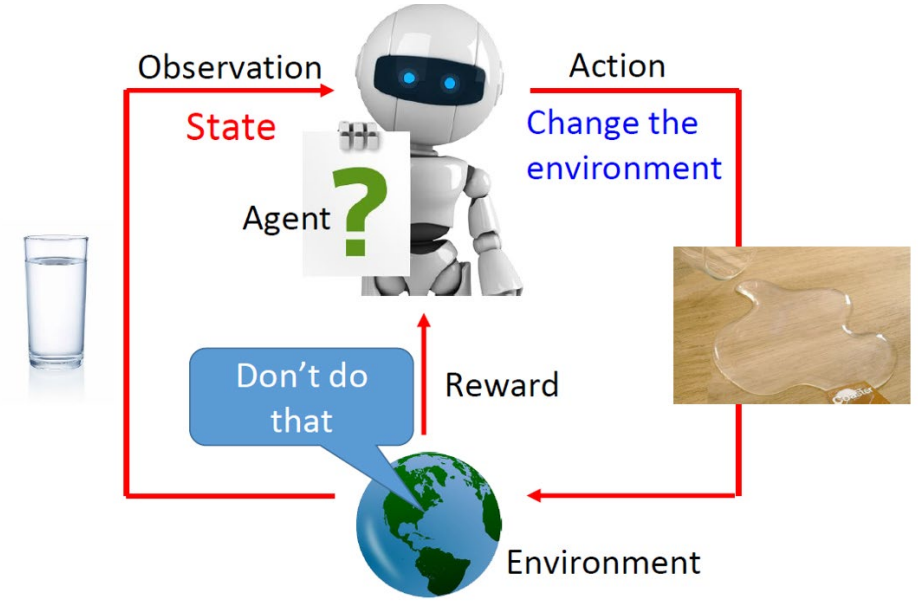


Reinforcement Learning

RL is a general-purpose framework for decision-making

- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- Goal: select actions to maximize future reward
- An episode is considered as a trajectory

$$\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$$
$$R(\tau) = \sum_{t=1}^{T} r_t$$



Reinforcement Learning

Policy gradient

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau)\nabla log p_\theta(\tau)]$$