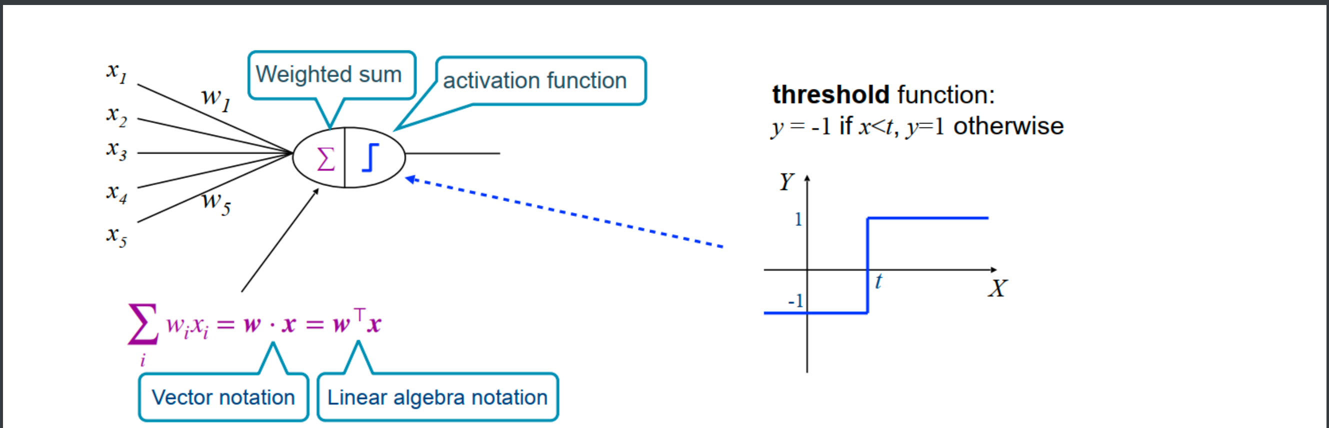
neural networks

为什么**深度**网络好于**宽度**网络

1. 逐层提取从低级到高级的特征
2. 深度比宽度计算效率高（宽度需要单层有指数级更多的node）
3. 更强的函数表示能力

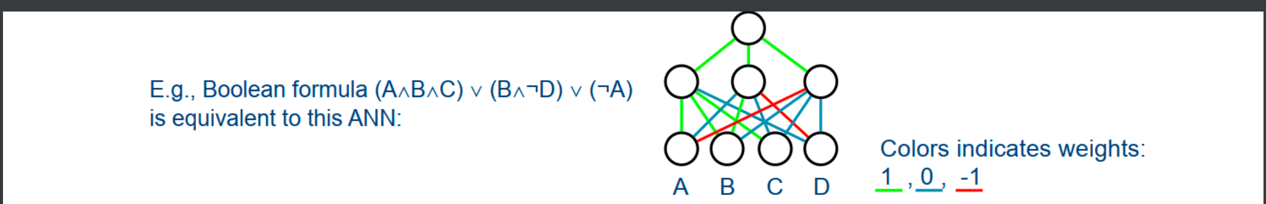
ANN: the basics

- perceptron: activation function, weighted sum

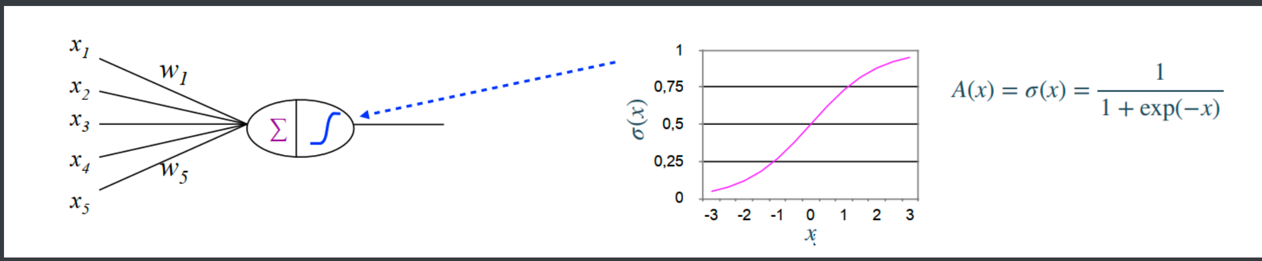
  

  if classes are not linearly separable, perceptron cannot represent a correct separator, for example, boolean functions (such as `and`)
- MLPs (multi-layer perceptrons)
  - hidden layer: all except the output layer
  - no longer restricted to linear separations, can express `and`
  - theorem on boolean functions: every boolean function can be represented by a 2-layer ANN (with enough neurons in the hidden layer)

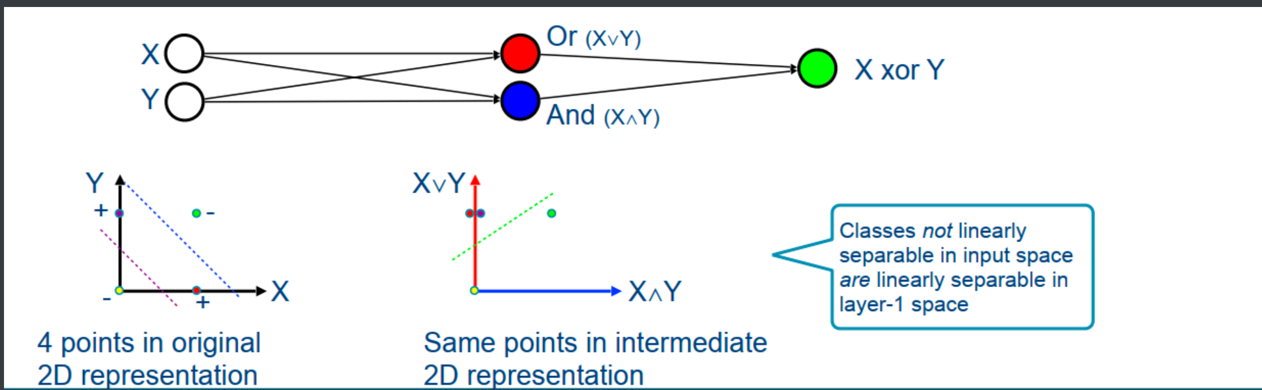
    proof: disjunctive normal form, first layer for "and", second layer for "or"

    2-layer perceptrons (2LP) are universal approximators
  - sigmoid threshold functions

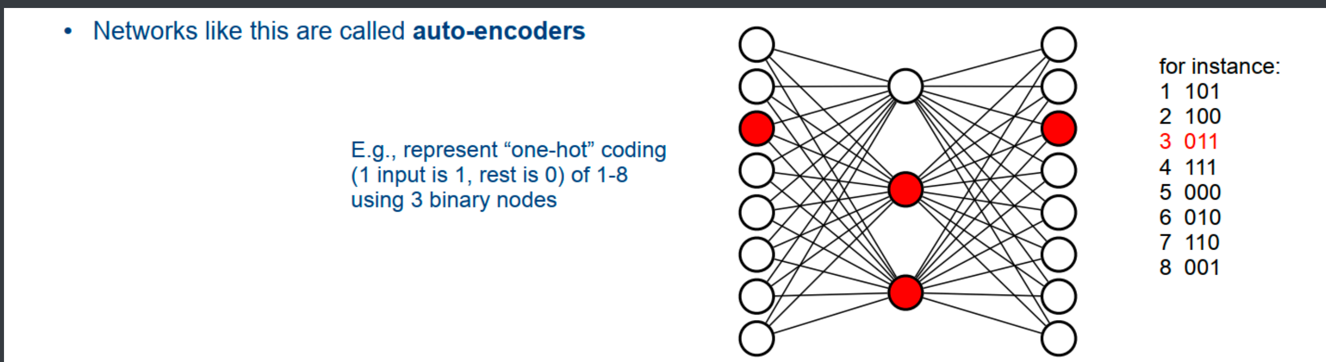


- 通过hidden layer的intermediate表示，ANN可以解决输入空间中原本无法线性分离的问题；



- auto-encoders

  compress & reconstruct



- interpretation of weights

  权重w和输入x的点积最大化时，可以视为它们最相似 -> 权重可以被解释为一个神经元“最理想的输入”

- activation functions

  typically for output layer,

  for classification -> sigmoid, for regression -> linear

  ReLU

- MLP 的应用功能是**映射**!

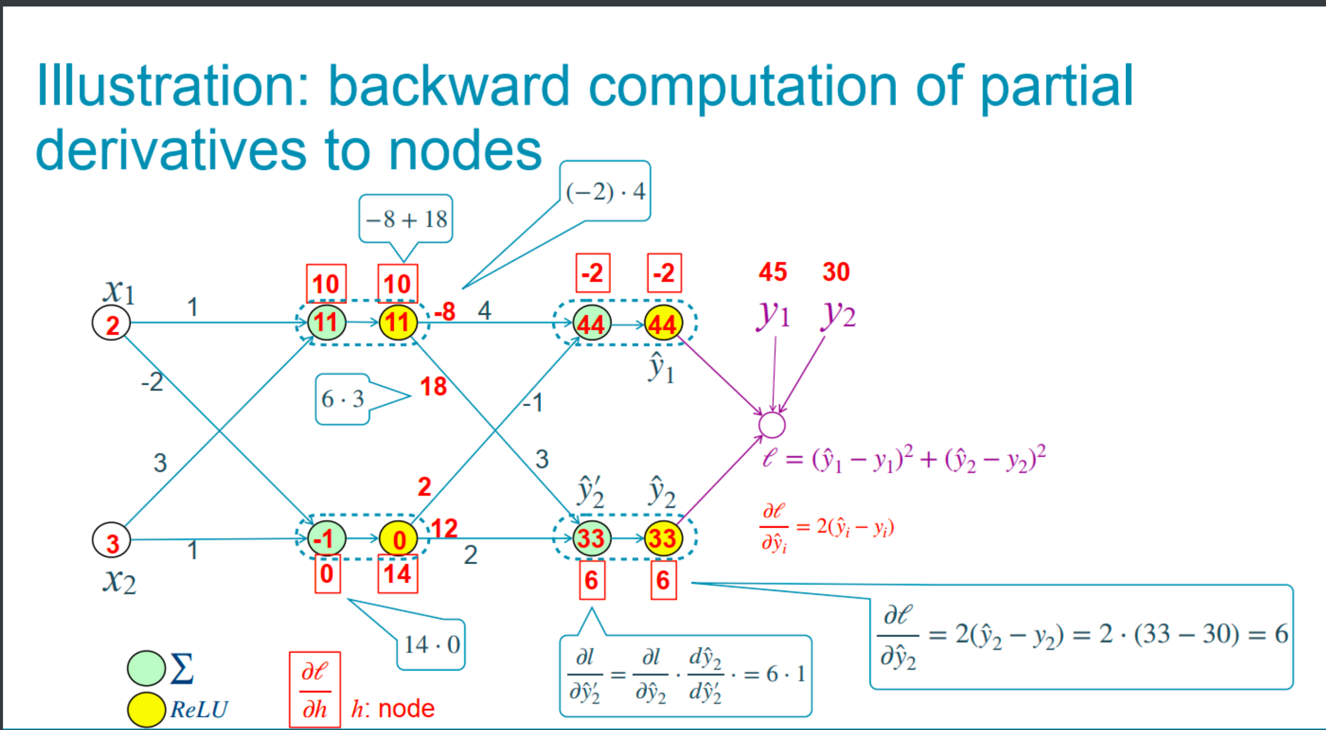
training

terminologies

- cost function
- loss function
- risk: expected loss value over a dataset of a model
- empirical risk: average loss over the dataset (用平均值估计期望)
- regularization terms: 用empirical risk可能导致overfitting。所以cost function通常要加上一个正则化项来限制模型的复杂度；

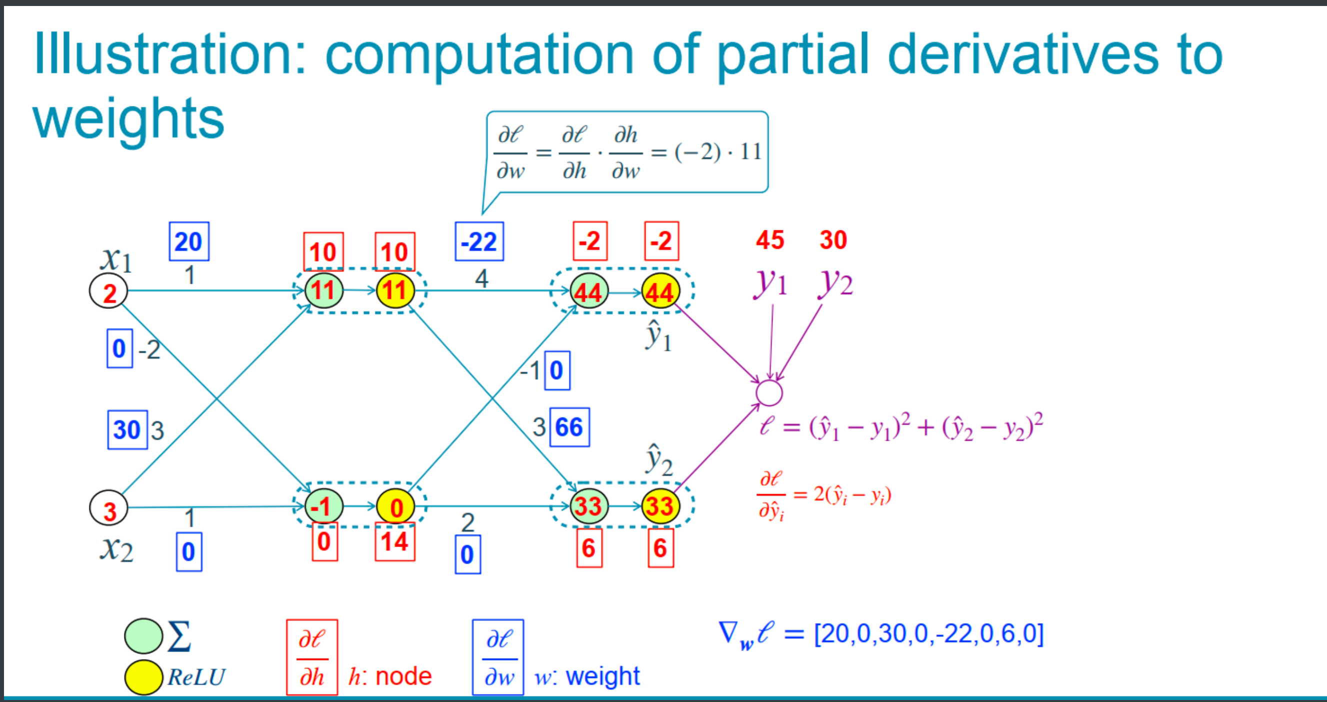
backpropagation 反向传播

- 节点的梯度反向传播



（图中对每个perceptron把加权和激活函数分开了）

- 权重的梯度反向传播



每个权重值的偏导数需要出发节点的函数值和到达节点的偏导数值；

backpropagation over many layers

层数非常多可能导致vanishing gradients（梯度消失）

ReLU 作为激活函数可以有效解决这个问题，所以逐渐取代了sigmoid

stochastic gradient descent（随机梯度下降法）

不需要每次使用整个数据集计算梯度下降，随机抽取一部分样本

---

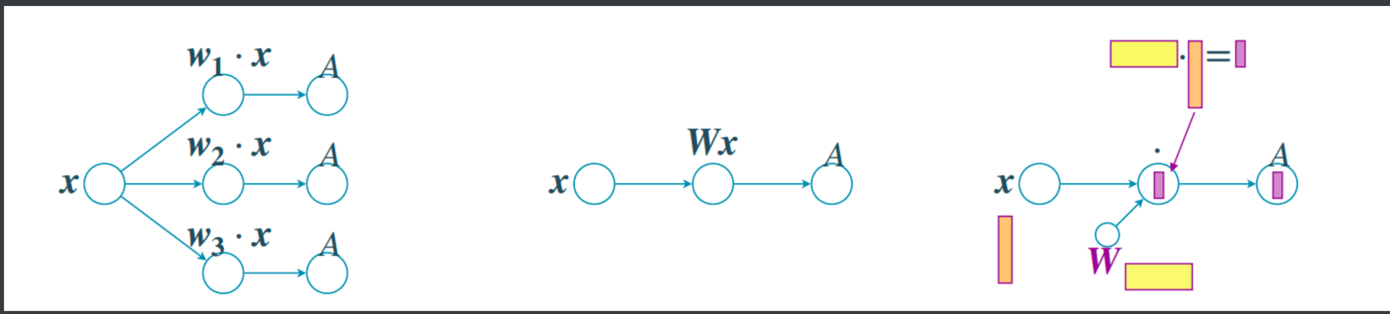
computational graphs

有向无环图

vector, matrix, tensor

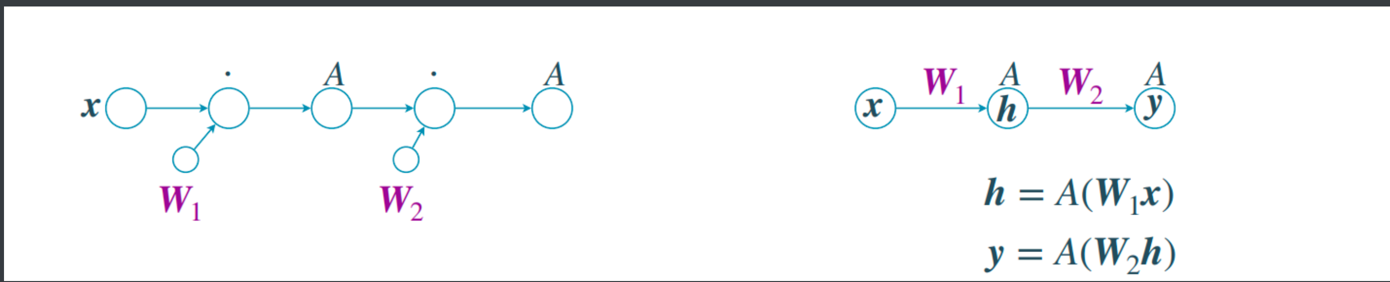
向量点积 -> 矩阵运算

单层感知机的矩阵运算&计算图表示：



（A可以理解为对一个竖向的转置的vector逐个元素ReLU）

多层感知机



实际应用中，一个节点表示aggregating the inputs + activation

activation function A is also left implicit in the picture

---

fitting computational graphs to data

Jacobian matrix

$$J_{\boldsymbol{x}}^{\boldsymbol{y}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ & \vdots & & \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} (\nabla y_1)^{\top} \\ (\nabla y_2)^{\top} \\ \vdots \\ (\nabla y_m)^{\top} \end{bmatrix}$$

For scalar y, $J_{\boldsymbol{x}}^{y}$ is essentially ∇y

For scalar x and y, J_x^y is essentially $\frac{dy}{dx}$

For $\boldsymbol{y} = f(\boldsymbol{x})$, we also write $J_{\boldsymbol{x}}^{\boldsymbol{y}}$ as J_f

first-order approximations 一阶近似

对于标量x，有（泰勒展开） $f(x+\varepsilon) \approx f(x) + \frac{df(x)}{dx}\varepsilon$

对于向量x，推广到Jacobian 矩阵： $f(\boldsymbol{x}+\boldsymbol{\varepsilon}) \approx f(\boldsymbol{x}) + J_f(\boldsymbol{x}) \cdot \boldsymbol{\varepsilon}$

Jacobian version 链式法则

- If $\boldsymbol{y} = f(\boldsymbol{x})$ and $\boldsymbol{z} = g(\boldsymbol{y})$, then $J_{\boldsymbol{x}}^{\boldsymbol{z}} = J_{\boldsymbol{y}}^{\boldsymbol{z}} \cdot J_{\boldsymbol{x}}^{\boldsymbol{y}}$

kronecker product

Question: if $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$, what is $\dfrac{\partial y_k}{\partial W_{ij}}$ for all i, j, k ?

$\boldsymbol{W}$ is a $m \times n$ matrix. We "flatten" it into a single vector with mn entries. Let $J_{\boldsymbol{W}}^{\boldsymbol{y}}$ denote the Jacobian matrix for this flattened version of $\boldsymbol{W}$

Since $y_i = \sum_j W_{ij} x_j$, we have $\dfrac{\partial y_k}{\partial W_{ij}} = x_j$ if $k = i$, 0 otherwise. In matrix form, this gives:

$$J_{\boldsymbol{W}}^{\boldsymbol{y}} = \underbrace{\begin{bmatrix} \boldsymbol{x}^\top & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{x}^\top & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{x}^\top \end{bmatrix}}_{m \times n \text{ columns}} \text{ with } \mathbf{0} = 0 \cdot \boldsymbol{x}^\top$$

Example:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \end{bmatrix}$$

$$J_{\boldsymbol{W}}^{\boldsymbol{y}} = \begin{bmatrix} x_1 & x_2 & x_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & x_2 & x_3 \end{bmatrix}$$

Applied to $\boldsymbol{w} = \boldsymbol{W}_2 \boldsymbol{v}$: $J_{\boldsymbol{W}_2}^{\boldsymbol{w}} = \begin{bmatrix} \boldsymbol{v}^\top & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{v}^\top & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & & & & \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{v}^\top \end{bmatrix}$

Multiplication with $J_{\boldsymbol{w}}^{\boldsymbol{y}}$ gives: $J_{\boldsymbol{W}_2}^{\boldsymbol{y}} = \begin{bmatrix} \frac{\partial y_1}{\partial w_1}\boldsymbol{v}^\top & \frac{\partial y_1}{\partial w_2}\boldsymbol{v}^\top & \cdots & \frac{\partial y_1}{\partial w_n}\boldsymbol{v}^\top \\ \frac{\partial y_2}{\partial w_1}\boldsymbol{v}^\top & \frac{\partial y_2}{\partial w_2}\boldsymbol{v}^\top & \cdots & \frac{\partial y_2}{\partial w_n}\boldsymbol{v}^\top \\ \vdots & & & \\ \frac{\partial y_n}{\partial w_1}\boldsymbol{v}^\top & \frac{\partial y_n}{\partial w_2}\boldsymbol{v}^\top & \cdots & \frac{\partial y_n}{\partial w_n}\boldsymbol{v}^\top \end{bmatrix} = J_{\boldsymbol{w}}^{\boldsymbol{y}} \otimes \boldsymbol{v}^\top$ ("Kronecker product")

(a.k.a. "tensor product")

Similarly, $J_{\boldsymbol{W}_1}^{\boldsymbol{y}} = J_{\boldsymbol{u}}^{\boldsymbol{y}} \otimes \boldsymbol{x}^\top$

the general case of computational graphs

- 训练流程：

  option 1: perform GD for the cost function on the whole dataset

  option 2: SGD

option 3: cycle through the dataset with a single instance at a time

- learning rate
- stopping criterion: no more substantial reduction of cost

---

convolutional neural networks (CNN)

前馈神经网络 (feedforward layered networks), 常用于 CV

卷积层 convolution layers & 池化层 pooling layers

network structure

input:

an input matrix X & a kernel K

output:

a map M where M_{ij} indicates how well K matches the submatrix of X starting at position (i, j)

(M is a map of "where K occurs" in X)

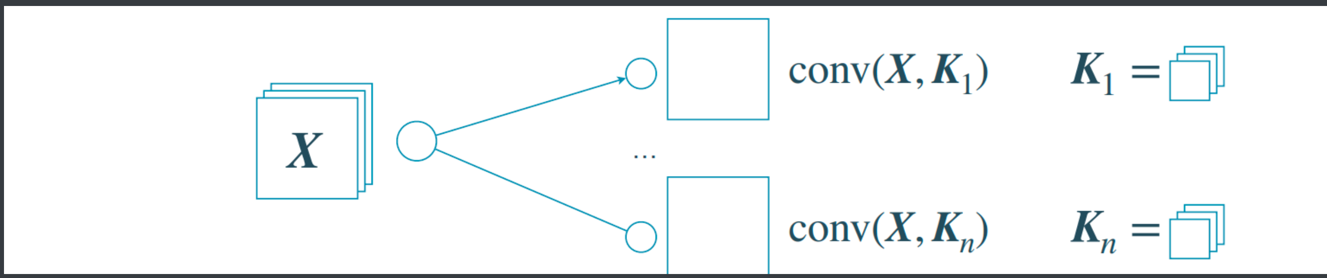
- convolution 卷积运算

$$5 \cdot 2 + 2 \cdot 3 + 1 \cdot 4 + 4 \cdot 2 = 28$$

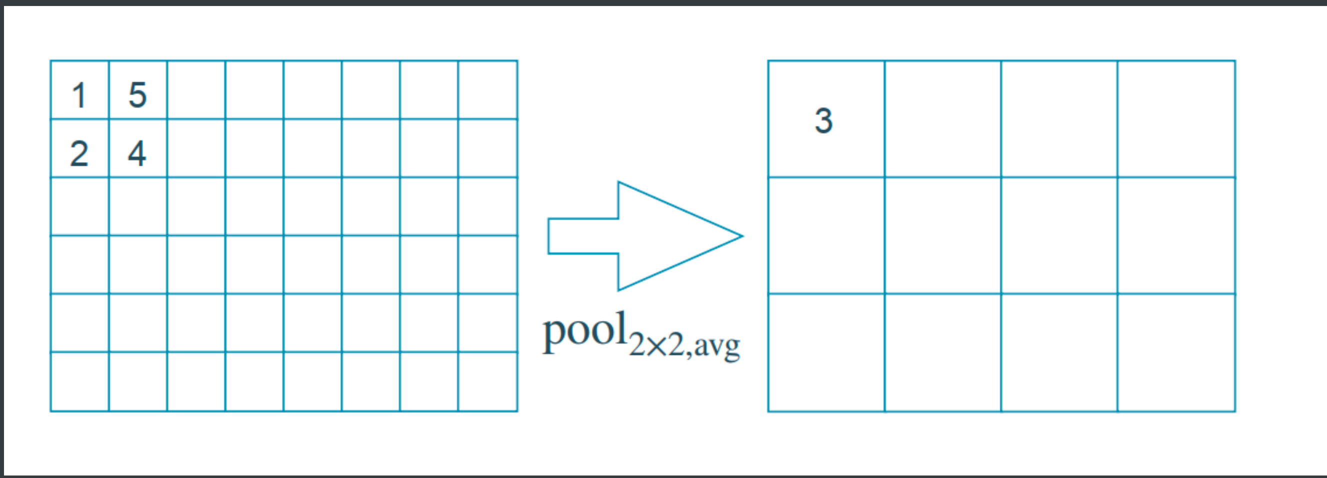
$$X = \begin{bmatrix} 5 & 2 & 3 & 0 & 2 \\ 1 & 4 & 2 & 5 & 1 \\ 2 & 1 & 4 & 6 & 0 \\ 0 & 1 & 8 & 4 & 3 \end{bmatrix} \qquad K = \begin{bmatrix} 2 & 3 \\ 4 & 2 \end{bmatrix} \qquad \mathrm{conv}(X, K) = \begin{bmatrix} \boxed{} & & \\ & & \\ & & \end{bmatrix}$$

表示两个矩阵有多相似；

输入形式：



- pooling layers 池化层

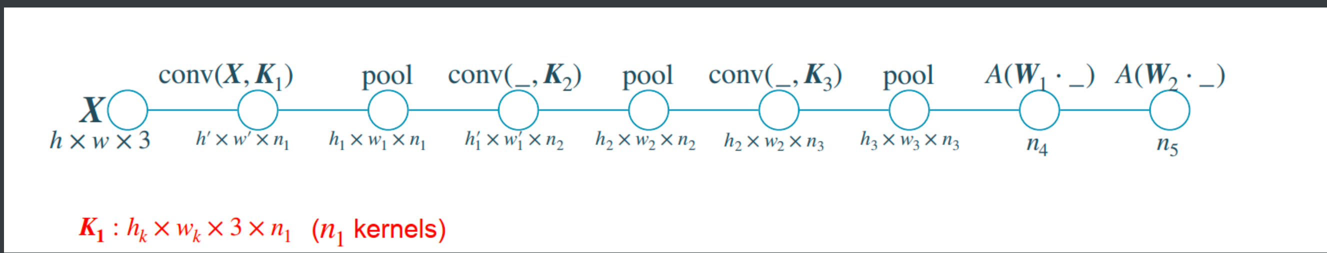


average池化 / max 池化

<u>池化层的目的</u>：

1. **降低计算复杂度**：缩小特征图尺寸
2. **提取最重要的特征**：在局部区域抑制噪声，忽略微小变化
3. **robust&泛化**：减少位置敏感性，更容易适应平移/缩放/旋转

- network structure



第一个卷积层n1个卷积核，每个卷积核有3个通道，在原始输入数据的3个通道上分别卷，然后加和得到单通道数据；最后n1个卷积核的结果concatenate，获得这个层的n1维输出数据；

通常，CNN的最后几层的构成是MLP（多层感知器），负责将提取的**high-level features**转化为具体的决策输出；

size和步长不一定总是相等的（重叠）

- how many layers?

observation: with too many layers, performance goes down again, even on the training set! (not overfitting, but trainability)

reason:

1. degradation problem 更深的网络无法有效学习到恒等映射
2. vanishing gradient / exploding gradient
3. difficult to optimize (优化困难) 出现更多鞍点或局部极小值

solution: ResNet !!!

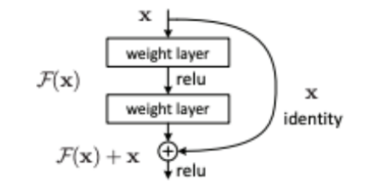


Figure 2. Residual learning: a building block.

至少可以学到恒等映射（identity function）

- pre-trained model

  priming the network with general-purpose visual features

pitfall: huskies vs. wolf

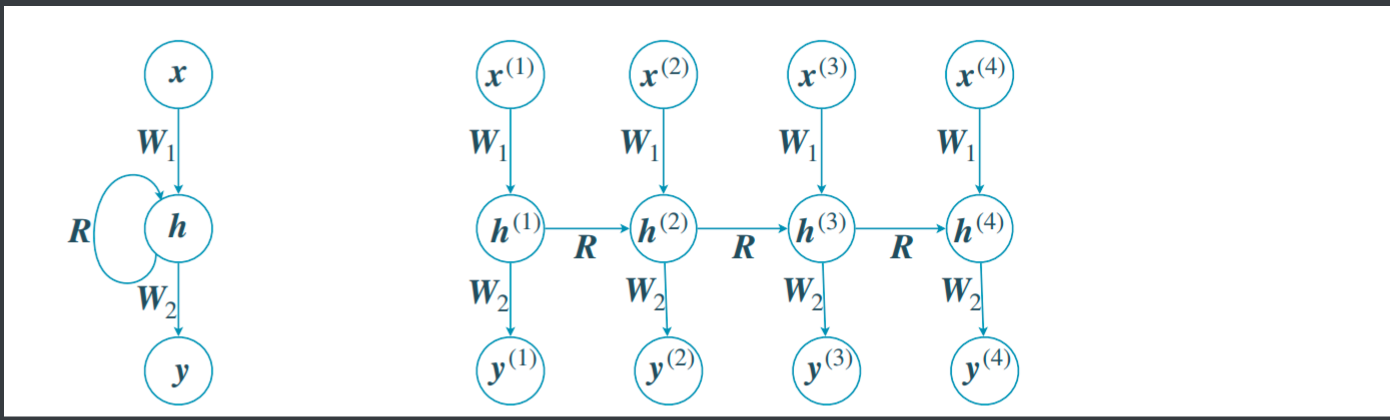
it is important to understand the features that the CNN uses

pitfall: low robustness

small sticker on traffic sign can totally destroy recognition

---

recurrent neural networks (RNN)



节点有记忆，可以记住自己之前的计算结果

backpropagation through time (BPTT)

vanishing gradient / exploding gradient 随序列变长而更加严重，solution是gated RNN或者LSTMs；

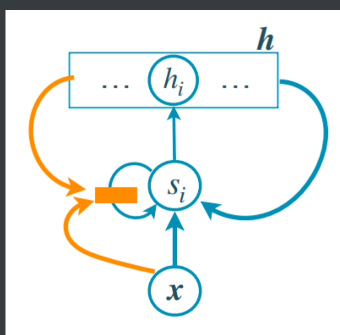
gated RNN

like "opening / closing the door", a selectvie way to process information

LSTMs

假设一个RNN的h节点有： $h_i^{(t)} = \tanh(b_i + \boldsymbol{w}_i \boldsymbol{x}^{(t)} + \boldsymbol{r}_i \boldsymbol{h}^{(t-1)})$

现在h节点和其相应的输入x节点中间加一个s节点：



s节点控制gate，

1. 当下面的函数 $f_i^{(t)} = 0$，节点 h_i reset（忘记记忆）

$$f_i^{(t)} = \sigma\left(b_i^f + \boldsymbol{w}_i^f \boldsymbol{x}^{(t)} + \boldsymbol{r}_i^f \boldsymbol{h}^{(t-1)}\right)$$

2. 当下面的函数 $g_i^{(t)} = 0$，节点 h_i 忽略当前输入

$$g_i^{(t)} = \sigma\left(b_i^g + \boldsymbol{w}_i^g \boldsymbol{x}^{(t)} + \boldsymbol{r}_i^g \boldsymbol{h}^{(t-1)}\right)$$

3. in summary, LSTM的 s 节点计算如下：

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \boldsymbol{w}_i \boldsymbol{x} + \boldsymbol{r}_i \boldsymbol{h}^{(t-1)}\right)$$

从 s 节点到 h 节点还有一个输出控制：

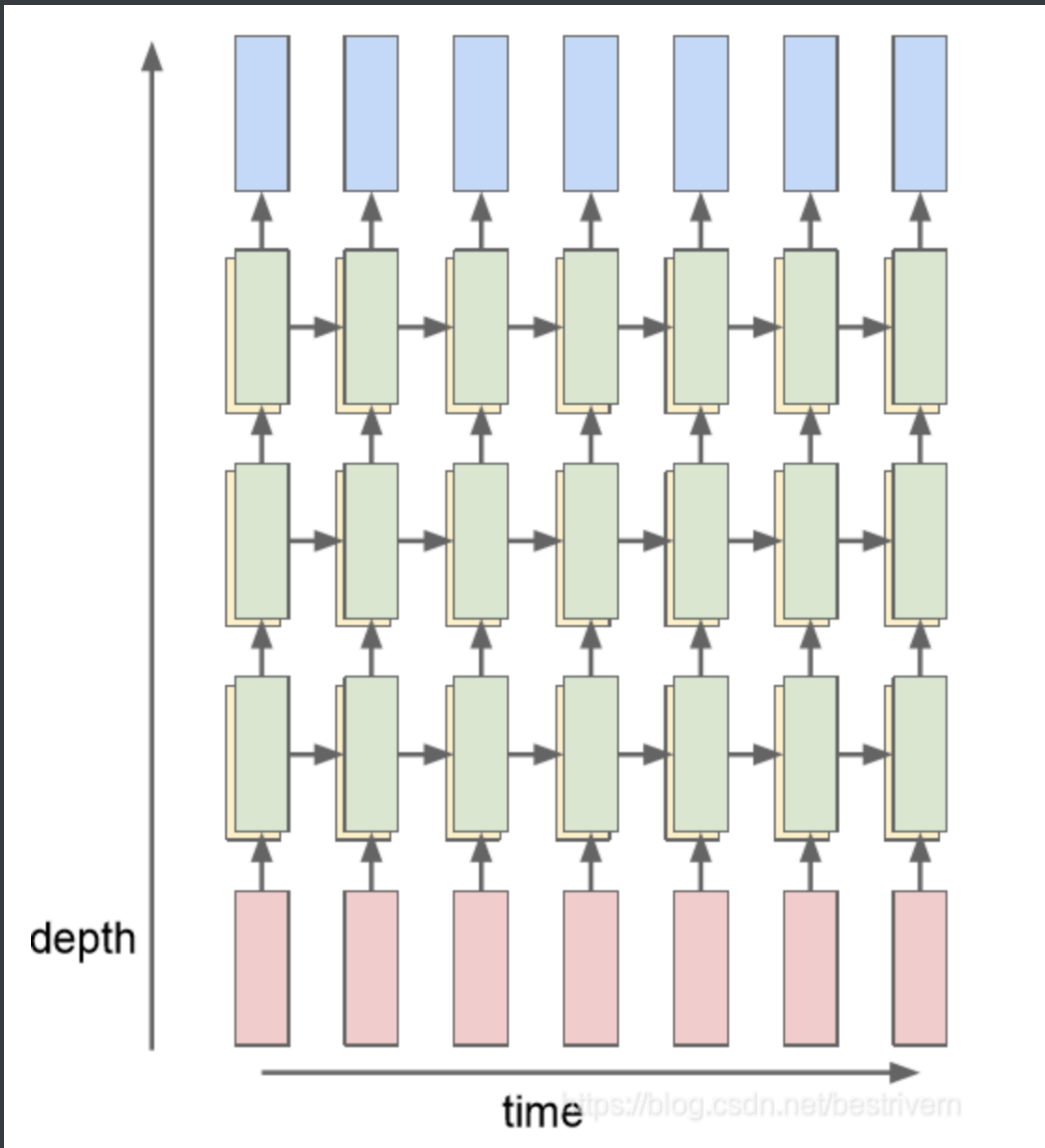
$$h_i\text{: } h_i^{(t)} = o_i^{(t)} \tanh(s_i^{(t)})$$

$$o_i^{(t)} = \sigma\left(b_i^o + \boldsymbol{w}_i^o \boldsymbol{x}^{(t)} + \boldsymbol{r}_i^o \boldsymbol{h}^{(t-1)}\right)$$

LSTMs can

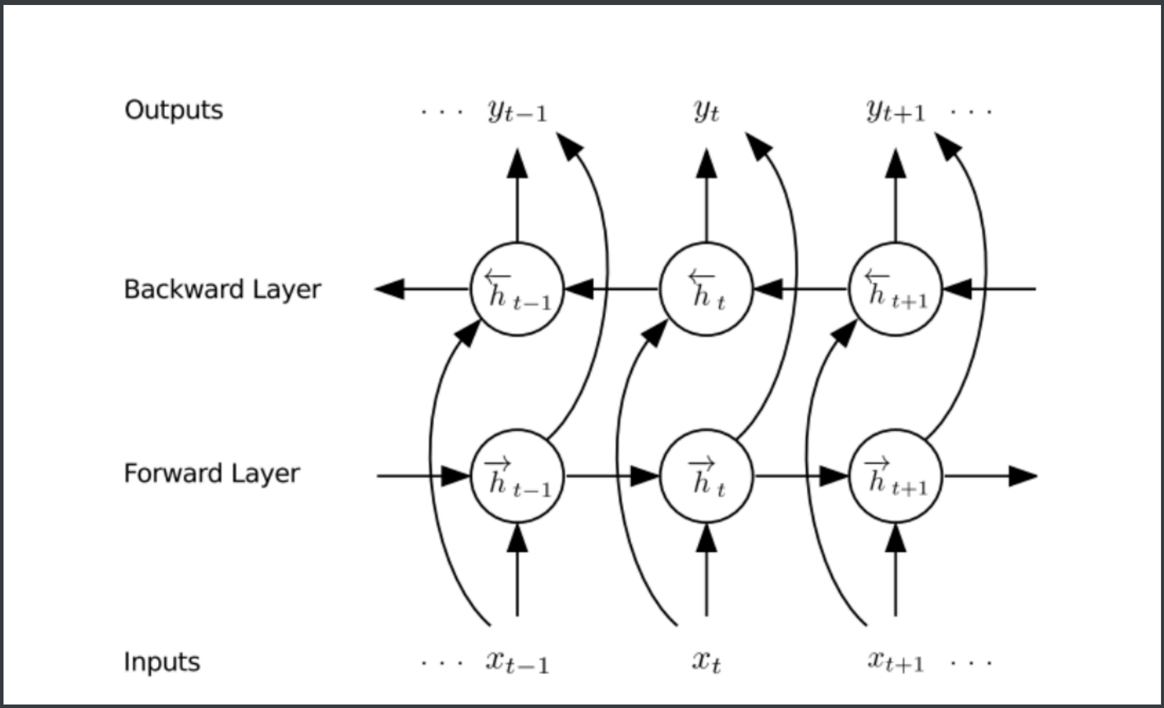
- learn the right time scale for looking at a sequence
- remember sth. as long as necessary, then immediately forget

deep LSTMs



turns out to work much better than single-layer LSTMs

bidirectional RNNs



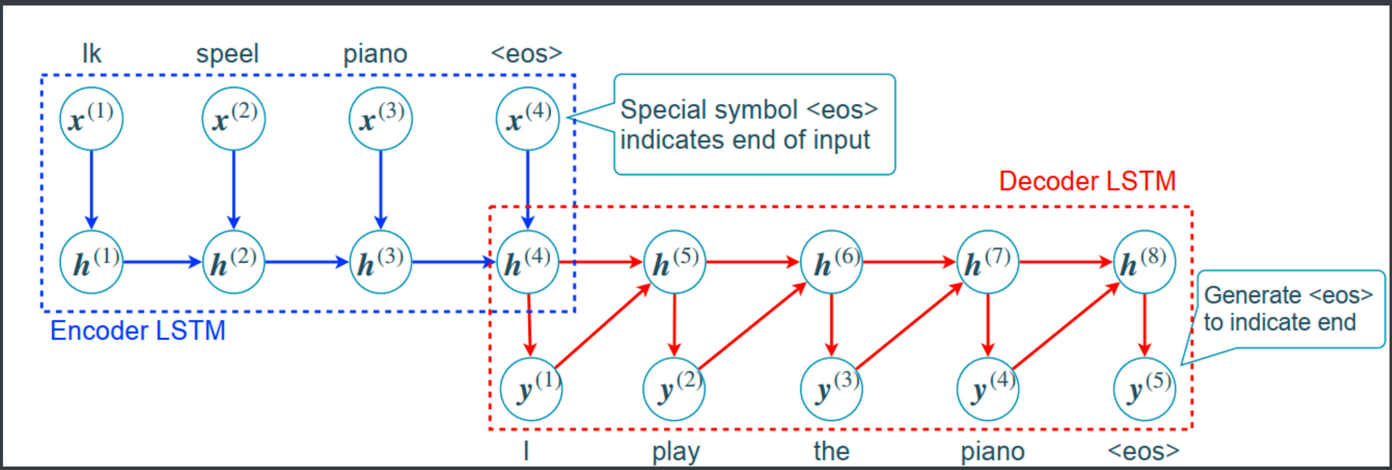
forward layer and backward layer can contain complementary info.

---

seq 2 seq, natural language translation

input和output都是序列，且不一定等长

encoder-decoder



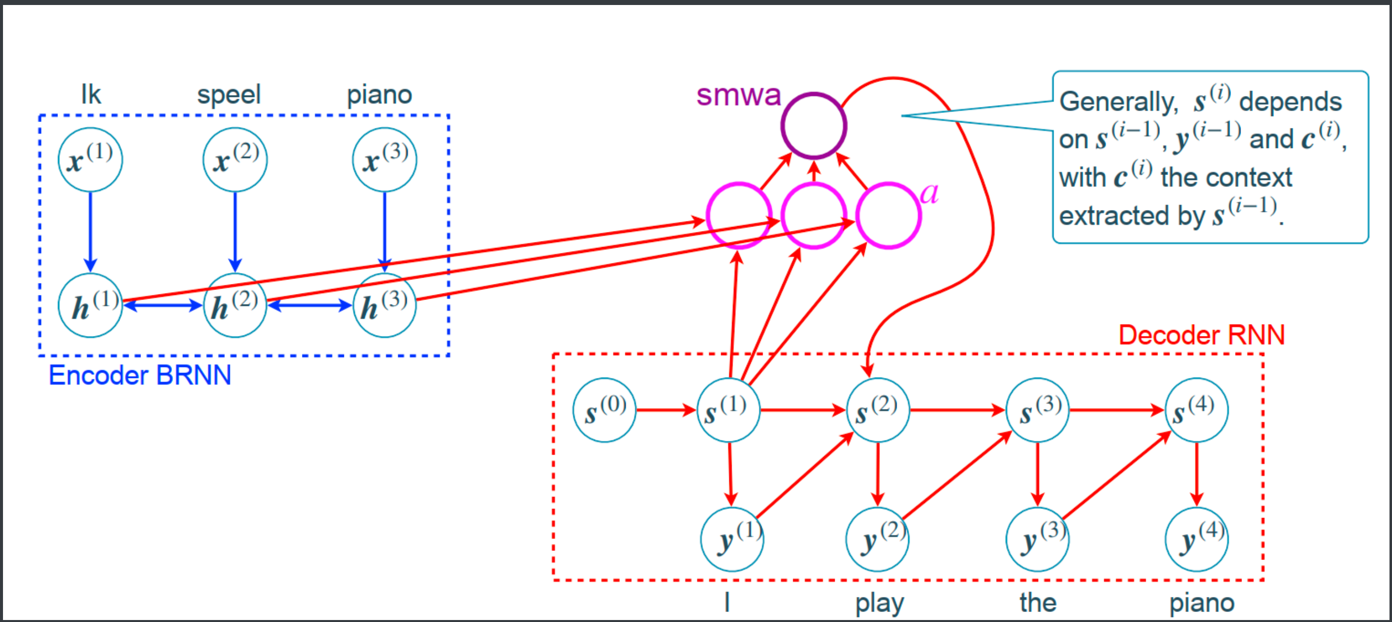
attention

$h^{(j)}$ 是 encoder 在读到输入句子第 j 个单词时的状态（bidirectional RNN 表示该位置的输入的“上下文”信息），

$s^{(i)}$ 是 decoder 在输出第 i 个单词后所处的状态；

attention 的核心是函数 $a(s^{(i-1)}, h^{(j)})$ 表示decoder在生成第 i 个输出之前的状态 $s^{(i-1)}$ 与 和输入中第 j 个位置的上下文状态 $h^{(j)}$ 有多匹配；

decoder 用smwa（softmax-weighted-average）选出matching程度最高的vector



与seq2seq相比，attention相当于提供了 $s^{(i)}$ 访问所有 $h^{(j)}$ 的shortcut，the earlier $h^{(j)}$ are never far away

s 可以被视为从序列 $h^{(1)}$ ... $h^{(n)}$ 中extract information的 **query**

transformer

drop the RNN aspect entirely, keeping only attention

- Consider a “query” vector $\boldsymbol{q}$
- We define a mapping as follows: $m(\boldsymbol{q}) = \dfrac{\sum_i \exp(\boldsymbol{q} \cdot \boldsymbol{k}_i) \cdot \boldsymbol{v}_i}{\sum_i \exp(\boldsymbol{q} \cdot \boldsymbol{k}_i)}$.
- Note: $m(\boldsymbol{q})$ is a softmax-weighted average of all $\boldsymbol{v}_i$, using $\boldsymbol{q} \cdot \boldsymbol{k}_i$ as the “match” between $\boldsymbol{q}$ and $\boldsymbol{k}_i$

对输入做线性变化，得到K, Q, V矩阵

$$\text{Attention}(Q, K, V) = V \cdot \text{softmax}\left(\frac{K^{\top}Q}{\sqrt{d_k}}\right)$$

BERT

---

auto-encoders: learns a new representation of the inputs

encoder, decoder

lossless compression : is the latent space is lower-dimensional && output = input

denoise

- auto-encoder & dimensionality reduction

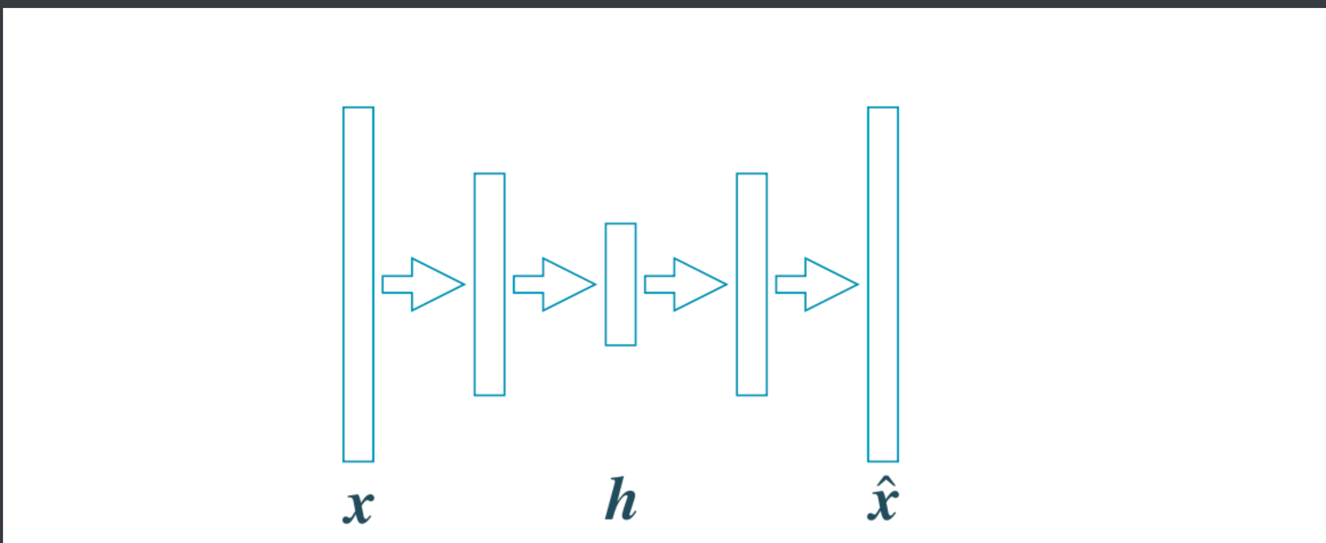
  类似pca的降维过程，但autoencoder是一种非线性降维，PCA是线性

  PCA只能发现数据中的线性关系，且主成分之间是正交的；

  AE的隐藏层向量不一定是正交的

  AE的隐藏层向量不一定是低维的，为了防止其维度过高导致过拟合，可以用正则化 regularization 强制隐藏层向量变得稀疏

- multi-layer encoders and decoders



- denoising auto-encoders

  - A simple approach to learn to denoise inputs:
    - Collect pairs $(\tilde{\boldsymbol{x}}, \boldsymbol{x})$, with $\tilde{\boldsymbol{x}}$ a noisy version of the clean input $\boldsymbol{x}$
    - Train an encoder on the $(\tilde{\boldsymbol{x}}, \boldsymbol{x})$ pairs, instead of $(\boldsymbol{x}, \boldsymbol{x})$ pairs

  可以训练用于数据降噪 / 缺省插值的网络；

  如果只能拿到clean data，可以人工生成noise；

---

generative adversarial networks

GANs train a decoder to generate realistic *x* from random *h*

GAN包括两个网络：

- A generator G (essentially the decoder part of an auto-encoder) that aims to generate realistic data (by decoding a randomly sampled $\boldsymbol{h}$)
- A discriminator D that tries to distinguish real data $\boldsymbol{x}$ (from a given dataset) from "fake" data $G(\boldsymbol{h})$ generated by G from a randomly chosen $\boldsymbol{h}$

D的训练目标：最大化以下value function：

$$V(G, D) = \underbrace{\mathsf{E}_{x \sim T}[\log D(\boldsymbol{x})]}_{\text{Try to predict numbers close to 1 for training data}} + \underbrace{\mathsf{E}_{h \sim p_h}[\log(1 - D(G(\boldsymbol{h})))]}_{\text{Try to predict numbers close to 0 for "fake" (generated) data}}$$

即，对于真实数据，D(x)应该尽可能接近1，对于伪造数据G(h)，D(G(h))应该尽可能接近0；

G的训练目标是最小化这个value function；但实际上第一项与它无关，第二项容易导致梯度消失，所以G等价地maximize log D(G(h));

deep generative networks：

e.g.

a "deconvolutional neural network", reconstruct images from lower-dimensional representations

generating pictures / complete a picture with holes in it

---

restricted Boltzmann machines

---

variational auto-encoders

---

others