No.

Date

## Collaborative filtering

e.g. Movie preference $nu = 4$

| | A | B | C | D | x1 (romance) | x2 (actions) |
|---|---|---|---|---|---|---|
| m1 | 5 | 5 | 0 | 0 | 0.9 | 0 |
| m2 | 5 | ? | ? | 0 | 1.0 | 0.01 |
| m3 | ? | 4 | 0 | ? | 0.99 | 0 |
| m4 | 0 | 0 | 5 | 4 | 0.1 | 1.0 |
| m5 | 0 | 0 | 5 | ? | 0 | 0.9 |

$r(i,j) = 1$ if user $j$ has rated movie $i$ (0 otherwise)

$y^{(i,j)}$ = rating given by users $j$ on movie $i$ (if defined)

$w^{(j)}, b^{(j)}$ = parameters for user $j$

$x^{(i)}$ = feature vector for movie $i$

(similar to linear regression)

For user $j$ and movie $i$, predict rating: $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$m^{(j)}$ = number of movies rated by user $j$, $n$ = number of features.

To learn $w^{(j)}, b^{(j)}$

$$\text{Cost function} = \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (w_k^{(j)})^2$$

constant

minimize $J(w^{(j)}, b^{(j)})$

To learn parameters for all users $w^{(1)}, b^{(1)} \dots w^{(nu)}, b^{(nu)}$

$$J\begin{pmatrix} w^{(1)}, \dots w^{(nu)} \\ b^{(1)}, \dots b^{(nu)} \end{pmatrix} = \frac{1}{2} \sum_{j=1}^{nu} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{nu} \sum_{k=1}^{n} (w_k^{(j)})^2$$

Given $w^{(1)}, b^{(1)}, \ldots w^{(nu)}, b^{(nu)}$ to learn $x^{(i)}$

$$\frac{1}{2}\sum_{j:r(i,j)=1}(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)})^2+\frac{\lambda}{2}\sum_{k=1}^{n}(x_k^{(i)})^2$$

To learn $x^{(1)}, x^{(2)} \ldots x^{(nm)}$:

$$J(x^{(1)}, x^{(2)}, \ldots, x^{(nm)}) = \frac{1}{2}\sum_{i=1}^{nm}\sum_{j:r(i,j)=1}(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)})^2+\frac{\lambda}{2}\sum_{i=1}^{nm}\sum_{k=1}^{n}(x_k^{(i)})^2$$

Collabrative filtering : multiple users, overlapped item reviews.

$$J(w,b,x) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)})^2+\frac{\lambda}{2}\sum_{j=1}^{nu}\sum_{k=1}^{n}(w_k^{(j)})^2+\frac{\lambda}{2}\sum_{i=1}^{nm}\sum_{k=1}^{n}(x_k^{(i)})^2$$

Gradient descent

repeat {

$$w_i^{(j)} = w_i^{(j)} - \alpha\frac{\partial}{\partial w_i^{(j)}}J(w,b,x)$$

$$b^{(j)} = b^{(j)} - \alpha\frac{\partial}{\partial b^{(j)}}J(w,b,x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha\frac{\partial}{\partial x_k^{(i)}}J(w,b,x)$$

$x$ is also parameter.

☆ For binary babels:

Predict the probability of $y^{(i,j)}=1$. : $g(z) = \frac{1}{1+e^{-z}}$

Loss for binary label: $y^{(i,j)}$: $f_{(w,b,x)}(x) = g(w^{(j)}\cdot x^{(i)}+b^{(j)})$

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)}\log(f_{(w,b,x)}(x)) - (1-y^{(i,j)})\log(1-f_{(w,b,x)}(x))$$

$$J(w,b,x) = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

No.

Date

Mean Normalization

For user j, on movie i, predict

$w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$ $\quad y_{norm}(i,j) = y(i,j) - \mu$

User 5 (haven't rated any yet)

$w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $b^{(5)} = 0$ $\quad \underbrace{w^{(5)} \cdot x^{(i)} + b^{(5)}}_{0} + \mu_i = 2.5$

Normalization is mostly used for a user haven't rated any movies but not a movie hasn't rated by anyone

Custom Training Loop for GD

```
w = tf.Variable(3.0)
x = 1.0
y = 1.0
alpha = 0.01
iterations = 30
for iter in range(iterations):
    with tf.GradientTape() as tape:
        fwb = w*x
        costJ = (fwb-y)**2
    [dJdw] = tape.gradient(costJ, [w])
    w.assign_add(-alpha*dJdw)
```

variables are the parameters we want to optimize

tf.variables require special function to modify

↑ Auto Diff / Grad

No.

Date

```
optimizer = keras.optimizers.Adam(learning_rate = 1e-1)
iterations = 200
for iter in range(iterations):
    with tf.GradientTape() as tape:
        cost_value = cofiCostFuncV(X, W, b, Ynorm, R, num_users,
                                   num_movies, lambda)
    grads = tape.gradient(cost_value, [X, W, b])
    optimizer.apply_gradients(zip(grads, [X, W, b]))
```

$\frac{\partial}{\partial w}, \frac{\partial}{\partial b}, \frac{\partial}{\partial x} J(w, b, x)$ ←

Not limited to gradient descent.

Find related items:

find item k with $x^{(k)}$ similar to $x^{(i)}$

i.e. with smallest distance.

$$\sum_{l=1}^{n} (x_l^{(k)} - x_l^{(i)})^2$$

Limitation of collaborative filtering

cold start problem

- rank new items that few users have rated.
- show something reasonable to new users who have rated few items

—— Use side information

- Item: Genre, studio
- User: Demographics (age, gender, location), expressed preferences

Campus

No.

Date

Collaborative filtering:

Recommend items to you based on ratings of users who gave similar ratings as you.

Content-based filtering

Recommend items based on features of user and item to find good match

User features: $x_u^{(j)}$ for user $j$
Movie features: $x_m^{(i)}$ for movie $i$
> vector size could be different.

Predict rating of user $j$ on movie $i$ as:

$w^{(j)} \cdot x^{(i)} + b^{(j)}$

$v_u^{(j)} \cdot v_m^{(i)}$ → prediction

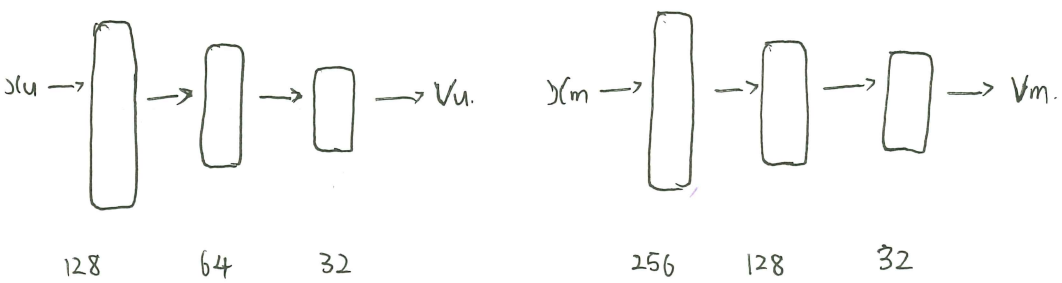
match for the same size

computed from $x_u^{(j)}$

computed from $x_m^{(i)}$

Neural network - combine user network and movie network

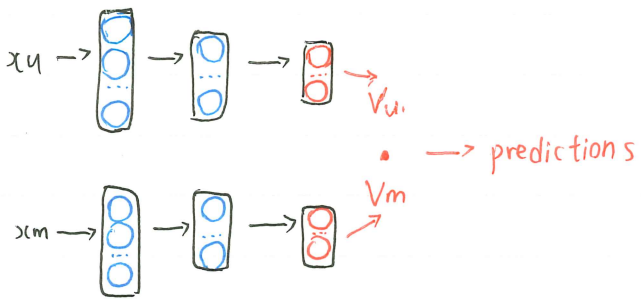
$x_u \rightarrow v_u$ $\quad$ $x_m \rightarrow v_m$



Cost Function: $J = \sum_{(i,j):r(i,j)=1} (v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2 + NN$ regularization term

To find movies similar to movie i: $||v_m^{(k)} - v_m^{(i)}||^2$ ↓

Can be pre-computed.




More efficient → Retrieval & Ranking

Retrieval:

- Generate large list of plausible item candidates.
  ensure broad coverage
- Combine items into list removing duplicates and items already watched

Ranking:

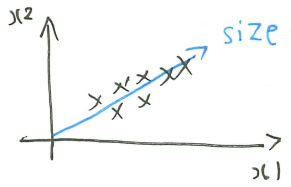
- Take the retrieved and rank using learned model
- Display ranked items to user

How many items to retrieve?

- To analyze the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations.
  e.g $P(y^{(i,j)}=1)$ of items displayed to user is higher

Principal Component Analysis:

find new axis and coordinates and user fewer numbers




2 -> 1 feature

n features → 2~3 features.
(many)

From 3D to 2D.

Data visualization

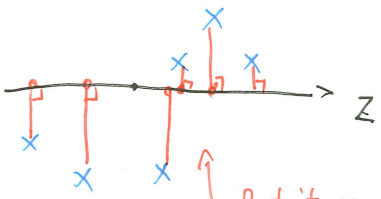
PCA algorithm.

preprocess: features:

Normalized to zero-mean

Feature scaling:

choose axis




"Project"

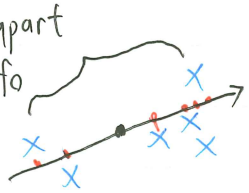
Capturing a lot of variance in original data set

But it may also squish data together

less variance

far apart

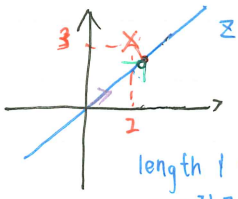
fore info



principal component

max variance

Coordinate on the new axis.

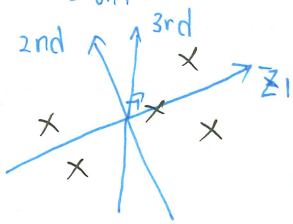



dot product

$\begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix} = 3.55$

length 1 vector

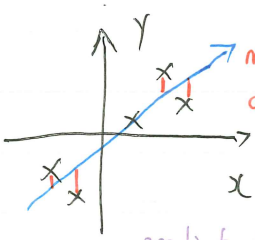
$\begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix}$




1st principal component

always perpendicular to each other.

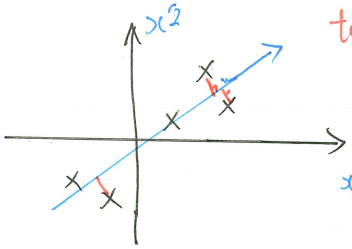
Linear regression




minimize distance along y axis

predict target output y.

PCA

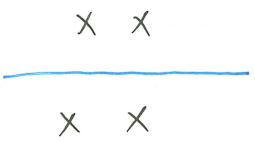



treat x1 and x2 equally!

find axis to retain variance.

---- x1 ---- x50

reduce number of x (features)



← lr

← pca.

given z = 3.55. can we find original (x1, x2)? reconstruction.

only approximately $3.55 \times \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix} \rightarrow \begin{bmatrix} 2.52 \\ 2.52 \end{bmatrix}$ (the point after projection)

↑ from $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$

No.

Date

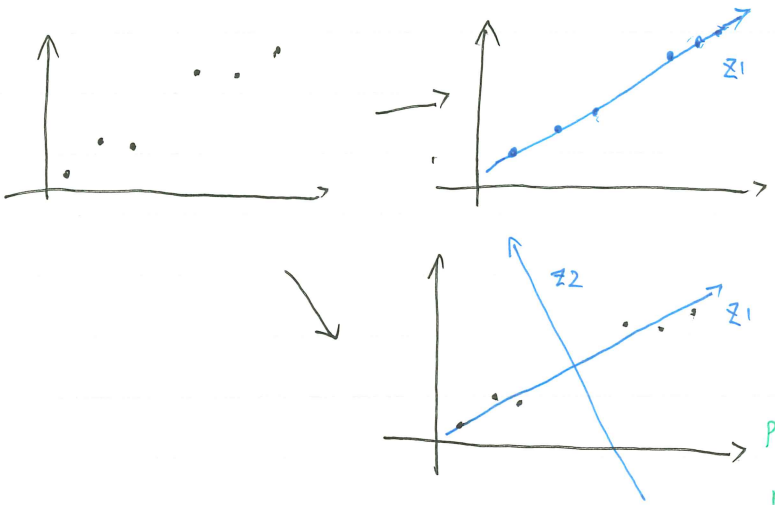
Application of PCA

Visualization (reduced to 2~3 features)

Previously:

Data compression

Speed up training of a supervised learning




pca_1.explained_variance_ratio = 0.992

pca_2.explained_variance_ratio = 0.008

Reinforcement Learning.

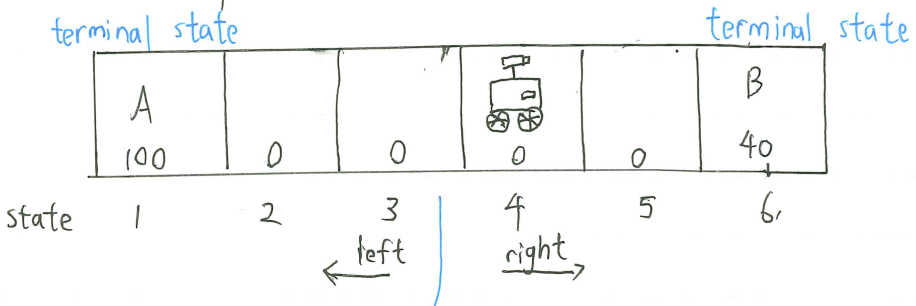
position of helicopter — control

state s — action a

supervised learning? ✗ hard to get dataset x and ideal action y

Reward functions.

Mars Rover Example.




(s, a, R(s), s')

e.g (4, ←, 0, 3)

first step isn't discounted!

Return = $0 + 0.9 \cdot 0 + 0.9^2 + 0.9^3 \cdot 100 = 72.9$

Return $= R_1 + r \cdot R_2 + \cdots$ (depend on actions)

Discount factor : $\boxed{r}$.

For system with negative rewards,

the algorithm will try to push the make of reward into future.

Goal: Find a policy $\pi$ that tells actions to take given state

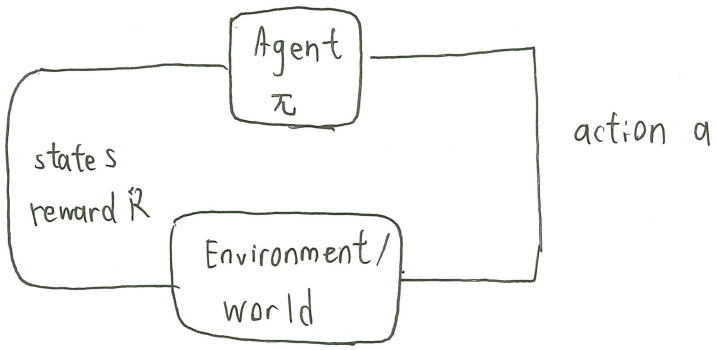
$a = \pi(s)$

No.

Date

Markov Decision Process (MDP):

Future only depends on where you are now,
but not how you've got here.




State action value function

$Q(s,a)$ = Return if you / $Q^*$ / optimal Q function

- start in state s
- take action a
- then behave optimally after that.

Measure how good is it to take a at s

The best possible return for state s is $\max_a Q(s,a)$
the best action.

Bellman Equation:

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

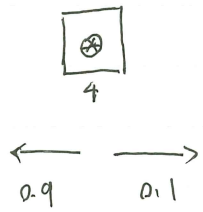
two key components for Reward Function

Reward you get right away

$a'$ action at $s'$

The broken down of $R_1 + \gamma R_2 + \gamma^2 R_3 \ldots$

Random (stochastic) environment




In stochastic event
Maximizing return will be random

Expected return: = Average $(R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots)$

$= E[R_1 + \gamma R_2 + \gamma^2 R_3 + \dots]$

Goal of R.L → choose a policy $\pi$ to maximize Average

$\pi(s) = a$

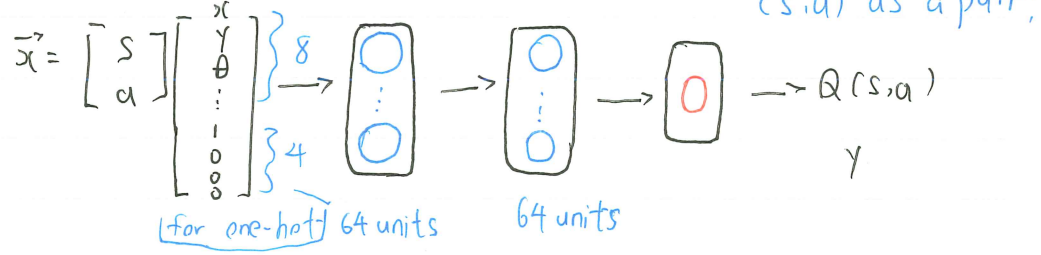
In reality, it's a continuous state.

Deep Reinforcement Learning

e.g landing on moon.

not inputs, output a.

(s, a) as a pair!




In states, use neural network to compute

Q(s, nothing), Q(s, left thruster), Q .....

Pick the action a that maximize Q(s, a)

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

| x | y |
|---|---|
| $x^{(1)} = (s^{(1)}, a^{(1)})$ (8 4) | $y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$ |
| $x^{(2)} = (s^{(2)}, a^{(2)})$ | $y^{(2)} = R(s^{(2)}) + \gamma \max_{a'} Q(s'^{(2)}, a')$ |
| ... | ... |
| $x^{10^5}$ | $y^{10^5}$ |

Then may use mean squared error loss to predict y

Learning Algorithm (DQN)

1. Initialize neural network randomly as guess of Q(s,a)
2. Repeat {.
   Take actions in the lunar lander, Get (s,a,R(s),s')
   Store 10,000 most recent (s,a,R(s),s') tuples Replay Buffer.

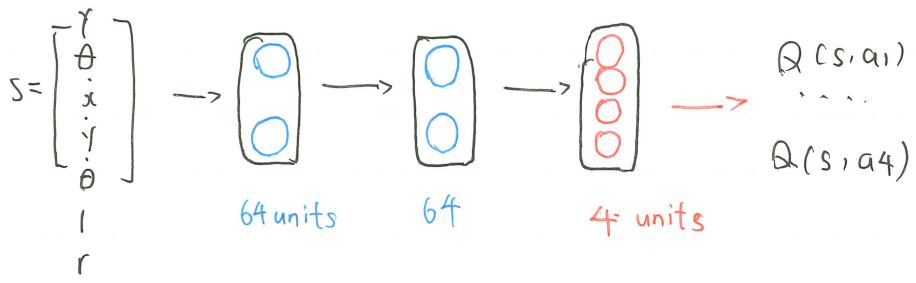
3. Train neural network:
   Create training set of 10,000 examples using
   $x = (s,a)$ and $y = R(s) + \gamma \max_{a'} Q(s',a')$

   Train $Q_{new}$ s.t $Q_{new} \approx y$ ($f_{w,B}(x) \approx y$)
4. Set $Q = Q_{new}$

Q(s,a) get's more accurate.

Refinement.

Architecture.




ε-greedy policy — pick actions while learning.

In state s: if $Q(s,a)$ is low, it will never try it.

Option 1: Pick the action a that maximizes $Q(s,a)$

Option 2: With Probability 0.95. pick the action that $\max Q(s,a)$ Greedy

Exploration → with probability 0.05, pick an action a randomly

$\varepsilon = 0.05$

Start ε high, and gradually decrease

ε: 1.00 → 0.01

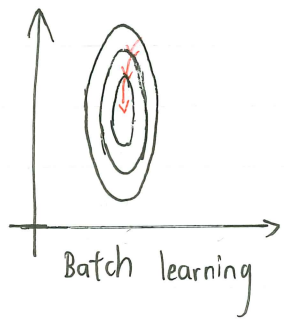
In RL, if you set ε and other parameters not quite well

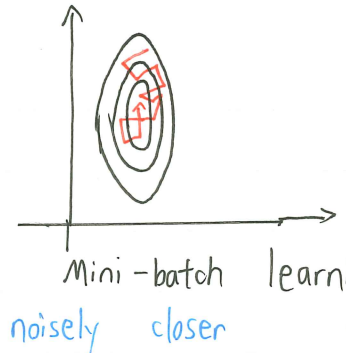
it will take 10 ~ 100 times to learn, very finicky

Mini-batch

if $m = 100{,}000$, the choose $m' = 1000$ to do gradient descent

| x | y | |
|---|---|---|
| ... | ... | } mini-batch 1 |
| ... | ... | } mini-batch 2 |
| ... | ... | |




noisely closer

but actually faster

To prevent from Qnew getting worse.

Soft Update.

how aggressive you update the Q.

$W = 0.01\, W_{new} + 0.99\, W$

$B = 0.01\, B_{new} + 0.99\, B$

Limitation of R.L

- Easier in simulation.
- Fewer applications for now. compared with supervised and unsupervised.

Target Network:

In practise. there will be Q-Network and target $\hat{Q}$-network

Error: $R + \gamma \max_{a'} \hat{Q}(s', a', w^-) - Q(s, a, w)$

$\gamma$ target

every C time steps we will use the $\hat{Q}$-Network to generate the $\gamma$ targets and update the weights of the target $\hat{Q}$-network using the weights of Q-network

```
q_network = Sequential([
    Input(shape=state_size),
    ...
])

target_q_network= ...

optimizer =Adam(learning_rate=ALPHA)
```

Experience Replay

a crucial component of off-policy deep reinforcement, improving the sample efficiency and stability of training by storing the previous environment interactions experienced by an agent.

- Generate uncorrelated experience instead of consecutive.

LSTM: Long Short-term Memory

GRU: Gated Recurrent Neural Network

No.

Date

Transformer 架构:

Based solely on attention mechanisms, dispensing with recurrence and convolutions entirely

开始于 机器翻译 — NLP — 图片, 视频

Replace recurrent layers most commonly used in encoder-decoder architectures with multi-head self-attention.

Another goal: make it less sequential 时序性

RNN: 从左往右一个词一个词看, 对于 词[t], 基于 $h_{t-1}$ 和 position t 生成 hidden states (隐层) $h_t$ 无法并行化! 但无法并行, 必须要先算出前置. 每一个 $h_t$ 都要存, 内存开销大 attention 机制带来高并行度

RNN对于比较长的序列难以建模 (它每次只看一个比较小的窗口)

e.g. 图片中若像素离的远则要很多层卷积才能将其融合

但用 注意力机制则可以在一层内看到整个序列

Multi-Head Attention: 模拟卷积网络多输出通道

Self-Attention: 联系单序列中不同位置, 以计算序列代表特征

x: sequence of symbol representations $(x_1, \dots x_n)$ ——> Encoder

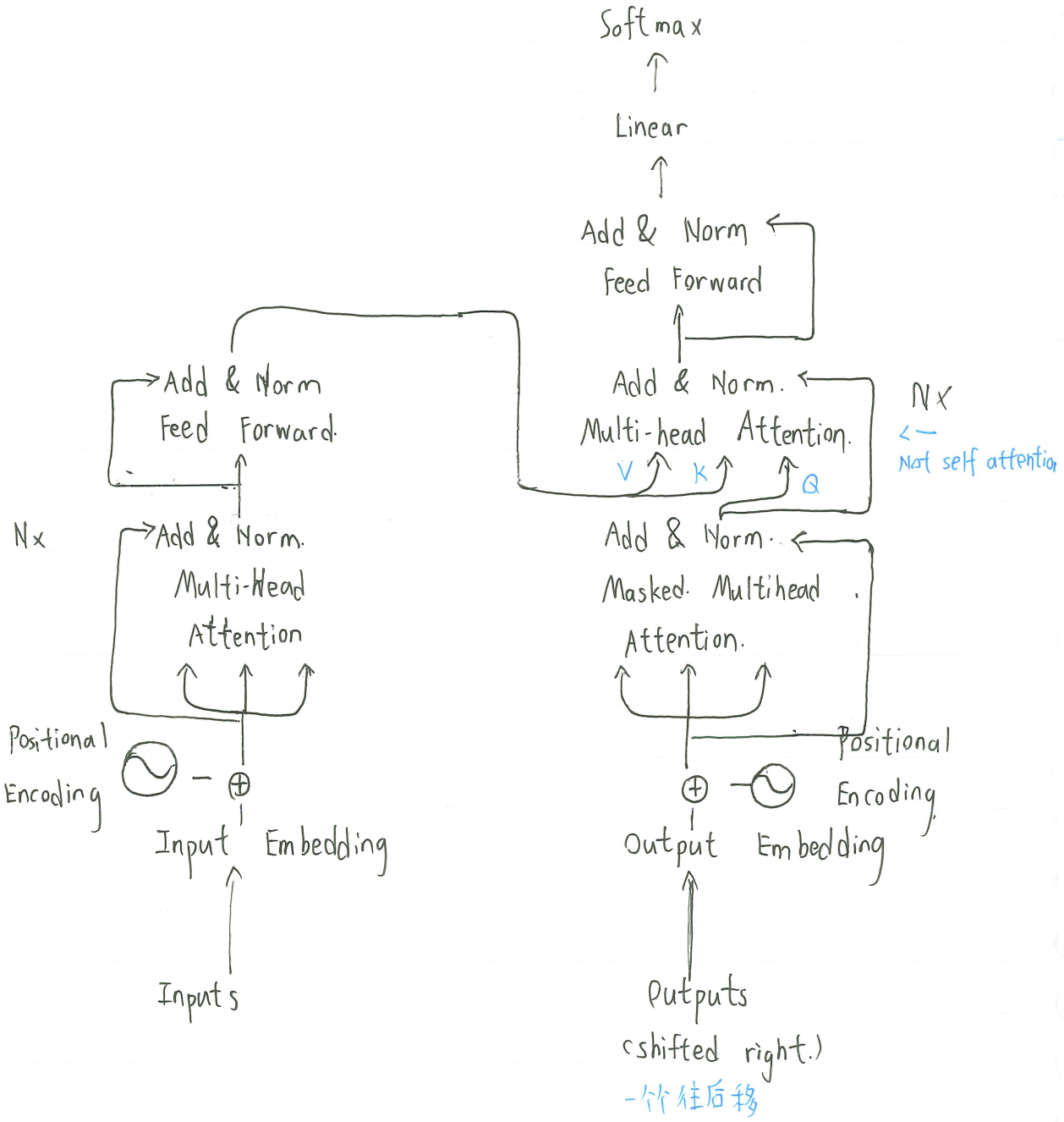
Encoder —> a sequence of continuous representations $z = (z_1, \dots z_n)$

z —> Decoder —> output sequence $(y_1 \dots y_m)$ of symbols one element at a time

Auto-regressive (自回归): 过去时刻的输出作为当前时刻输入的部分

Encoder

Decoder



Embedding layer (嵌入层): 将词表述成向量

Nx: N个层摞在一起

parameter

Encoder: a stack of N = 6 identical layers

2 sublayers: ① multi-head self-attention

② MLP

(残差连接) A residual connection around each of two sub-layers with layer normalization.

Output of each sub-layer: LayerNorm(x + Sublayer(x))

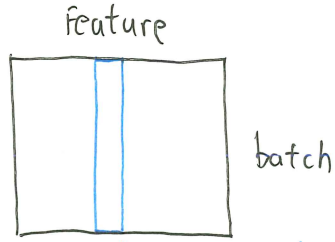
残差连接把输入和输出加在一起

all sub-layers and embedding layers produce outputs of dimension d = 512

Unlike CNN, MLP

to facilitate residual connections (or you need to do projection)

Batch Norm:




mini batch: make $\mu = 0$, $\sigma^2 = 1$ (by z-score)

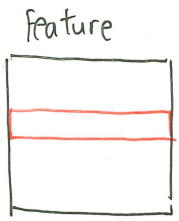
(one column)

训练时算小批量，预测时估效全局均值方差并存起来

先扫一遍

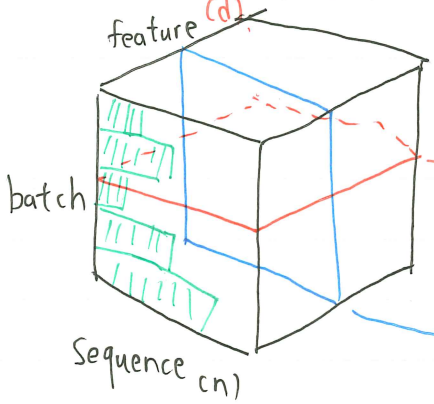
通过学习 MP 向量放成一个任意方差为某个值均值为某个值的东西?

Layer norm:




相当于整个把数据转置一下，放到 Batch Norm 里再转置回去

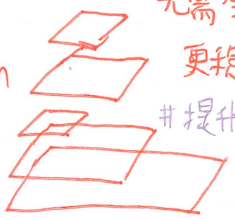
Transformer / RNN 里输入的是三维的



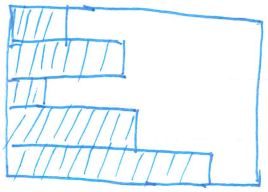

Layer Norm

每个样本自己算，无需全局均值方差更稳定

# 提升 lipschitz 常数



Batch norm 切一下拉成向量作前面的运算



问题：① 小批量计算时 $\mu, \sigma^2$ 抖动大

② 当碰到极端新样本时原 $\mu, \sigma^2$ 失效

对小批内的数据做归一化易受 Batchsize 影响

No.

Date

Decoder: a stack of N=6 identical layer.

3 sub-layers ① multi-head self-attention

② MLP

③ multi-head attention over the output of encoder.

Residual connections around sub-layers with layer normalization

解码器训练中做预测时不应看到后面的输出: prediction for $i$ depends on $k<i$

— 带掩码的注意力机制 (Masked), 保持训练和预测时的行为一致

Attention

Attention function: query $\vec{q}$ and key-value pairs $\vec{k}$ $\vec{v}$ ⟶ output $\vec{o}$

output is a weighted sum of values ($d\vec{o} = d\vec{v}$)

weight is assigned to each value computed by a compatibility function (相似度) between query and corresponding value

Scaled Dot-Product Attention

$d_q = d_k = d_v = d_{output}$

dot product can measure whether they are similar. θ↓ Product ↑

divide each by $\sqrt{d_k}$ - length of vector.

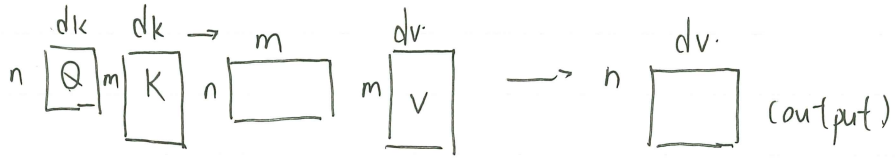
put n products of query and key into softmax → get weight sum = 1

make queries, keys, values into matrix

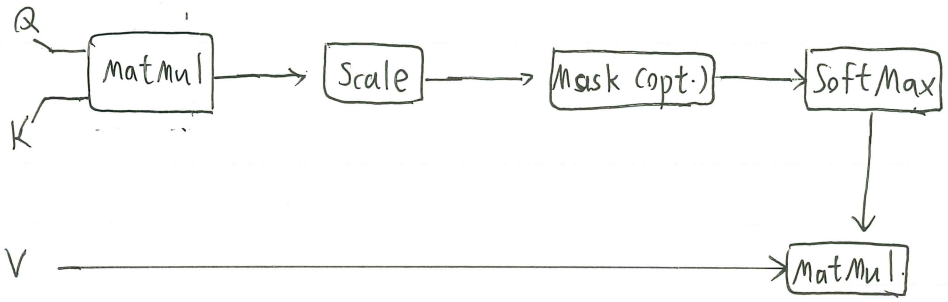
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

相比于 additive attention，dot-product 更高效

用矩阵方便并行




why $\frac{1}{\sqrt{dk}}$？: when dk个，it pushes the differences more extreme.
值会向两端靠拢，使梯度变得比较小，模型跑不动，
softmax 置信处 → 1，不置信处 → 0 则视为 convergence.




Mask：对于 $q_t \cdot k_t$ 大于 t 时的积换成很大的负数，（$10^{-11}$）
在 softmax 中变成 0，权重变成 0

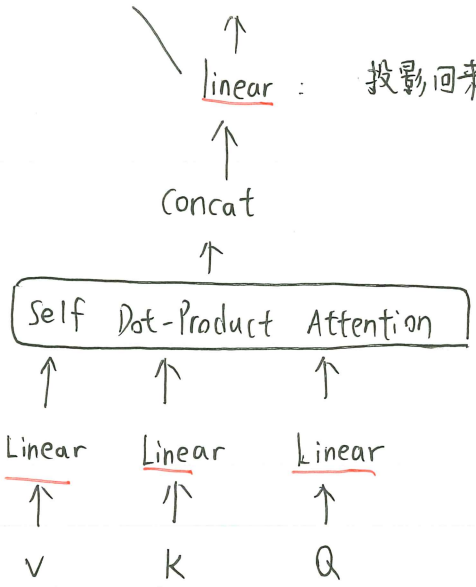
Self-attention（自注意力机制）：输入时 Q、K、V 相同
输入 Q、K、V 后的输出是 value 的加权和，权重来自 Query 和 Value 各个向量间的相似度

多头会稍微改变结果（因为生成了 h 个距离空间）

Multi-Head attention

Instead of performing a single attention function with dmodel-dimensional keys, values and queries, we linearly project the queries, keys and values h times with learned linear projections

Linear 投影到低维




投影到低维 可学习投影的方法
为了识别不同模式
不一样的计算相似性的方法

类似RNN中多输出通道

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$W_i^Q \in R^{d_{model} \times d_k}$, $W_i^K \in R^{d_{model} \times d_k}$, $W_i^V = R^{d_{model} \times d_v}$, $W^O \in R^{hd_v \times d_{model}}$

In the work we deploy $h = 8$ parallel attention layers/heads

$$d_k = d_v = \frac{d_{model}}{h} = 64$$

最后将h个区块连在一起(非求和,以便突出异同)

① Decoder 中的 非Masked Multi-Head 层中.

Value 和 Key 来自 Encoder, Query 来自 Masked-Multi Head.

通过比较他们的相似度计算权重

相当于把编码器中的信息拎出来"注意到"感兴趣的东西

Allow every position in the decoder to attend over all positions in the input sequence.

(mimics typical encoder-decoder attention mechanisms in sequence-to-sequence models)

②. Encoder 中的 self-attention layers..

, Each position in the encoder can attend to all positions in the previous layer of the encoder (like recursion?)

③ Decoder 中的 Masked Multi-Head 层.

Implement this inside of scaled dot-product attention by masking out (set to $-\infty$) all values of illegal connections