Deep learning notes

Tototititata

六大托科王

Lecture 1

Deep learning VS Machine Learning.

① DL：用神经元训练。
② ML：选feature，训练。

Tensor internal：内核？

Tensor 在储存的时候，都会被变作
1维向量。
用 .storage() 查看。
储存的时候也有 stride 和 offset 参数。

Lecture 2

2-1

机器学习3大类：（按照输出的类型）

1. Classification → 判断属于哪类
2. Regression → 预测值
3. Density Estimation → Capture the struct of data (e.g. outlier detection / image generation)

Expected Risk vs Empirical Risk

E.R：期望风险，模型f关于输入输出的真实的损失。

↓ 不知道真实输入（不知道P(X,Y)）

Emp.R.：经验风险：用合(X,Y)代替，模型f关于训练集的平均损失。

Training: Minimize the empirical risk

2-2.

KNN: k-nearest-neighbors

对于新输入，在训练集中与它最近的K个点，属于K中的多数类.

K太小，overfitting. 学习noise.
K太大，underfitting. 没学习

Capacity: 拟合函数的能力.
元素数多少

Regularization: 处理过拟合.

↓

保留所有features，减少特征数量级.
日↓. 越先背.

2-3 Bias - Variance.

Generalization error (泛化误差)

∴ 训练集的loss 和 一般化数据集的loss 之间的差异.

Bias. Variance. 用来描述学习的模型和真实模型之间的差距.

所有训练集可能训练出的 mean output 和 real data 的 output 之间的关系

trade off

不同训练集训练出的模型之间的差异.

2-5 Clustering and Embedding.

Embedding.：离散集合的元素映射到一个向量.

$x_1 \to \vec{v_1}$

$x_2 \to \vec{v_2}$

if x_1 is similar with x_2.

$\vec{v_1}$ is close to $\vec{v_2}$

PCA Algo.

Clustering: 类似元素在一起....

K-means.

Lecture 3

3-1 Perceptron

The basic mathmetical model for a neuron (神经元) was

Threshold Logic Unit.
(阈值逻辑单元)

$$f(x) = 1_{\{w\sum_i x_i + b \geq 0\}}$$

可以演变成例如

$$or(u,v) = 1_{\{u+v-0.5 \geq 0\}}$$

$w=1, b=-0.5$ 调参.

类似的基本逻辑门

o/w: otherwise.

Perceptron:

$$f(x) = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b \geq 0 \\ 0 & \text{o/w} \end{cases}$$

Rosenblatt

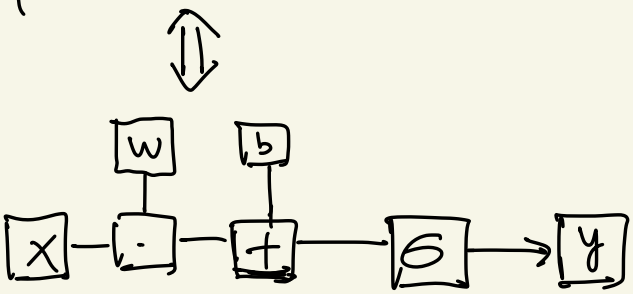
w: synaptic weights

또는 activation function σ

$$f(x) = \sigma(w \cdot x + b)$$

$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{o/w} \end{cases}$$

$$f(x) = \sigma(W \cdot x + b)$$



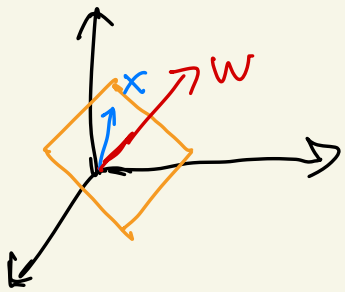
超平面：hyperplane

n维空间中n-1维的子平面，能把线性空间分割成不相交的两部分.

Ex：2维平面中的一条直线

3维空间中的一个平面.

法向量w：垂直于超平面的向量



□为过原点的超平面.

w为法向量

x为超平面上的任意一点

显然有 $w^T x = 0$ (法向量与平面上的向量积为0)

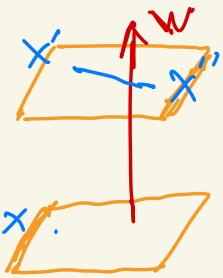
$w = (w_1, w_2, w_3)$

$x = (x_1, x_2, x_3)$

现将超平面沿着 w 方向移动.

此时 x 不再是超平面内的点,

$w^T x \neq 0$.



在新的超平面内有2点. x' 和 x''

$x' = (x_1', x_2', x_3')$

$x'' = (x_1'', x_2'', x_3'')$

可知 $(\vec{x'} - \vec{x''}) \cdot \vec{w} = 0$

$(x_1' - x_1'', x_2' - x_2'', x_3' - x_3'') \cdot (w_1, w_2, w_3) = 0$

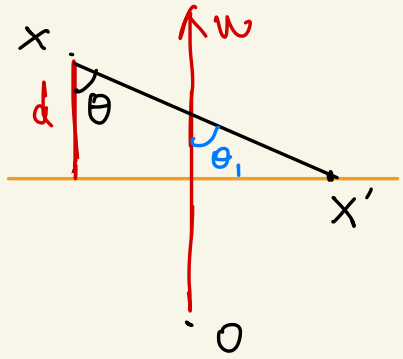
$x_1' w_1 + x_2' w_2 + x_3' w_3 = x_1'' w_1 + x_2'' w_2 + x_3'' w_3$

$w^T x' = w^T x''$

令 $b = -w^T x''$. $x' \to x$

可以得到 $w^T x + b = 0$ ← 超平面公式

点到超平面的公式



x是平面外一点，到平面距离为d.

$$\cos\theta = \frac{d}{\|x-x'\|}$$

w有两个方向，加||.

$$|(x-x')w| = \|x-x'\| \cdot \|w\| \cdot \cos\theta$$

因为 $d \parallel w$，有 $\theta = \theta_1$，可利用点乘求θ

$$d = \frac{|(x-x')\cdot w|}{\|w\|} = \frac{|wx-wx'|}{\|w\|}$$

在hyperplane内，$b = -w^T x'$

$$d = \frac{|wx+b|}{\|w\|}$$

补充. Perceptron.

Perceptron 本质上也是一个二分类的线性模线.

$$f(x)=\begin{cases}1 & \text{if } \sum_i w_i x_i + b \geq 0 \\ 0 & \end{cases}$$

w/w

↓ $w^T x + b$

超平面.

也就是说有一个超平面 $w^T x + b = 0$ 存在. 才可以二分类. ≥ 0 和 < 0. 即超平面的左右.

在 perceptron algo 中.

① initialize w with 0. ($w^0 = 0$)

② While $\exists n_k$ s.t. $y_{nk}(w^k \cdot x_{nk}) \leq 0$.

update $w^{k+1} = w^k + y_{nk} x_{nk}$

↓

y_{nk} 为 target output. -1 或 1. $w^k x_{nk}$ 为 prediction.

当 $y_{nk}(w^k \cdot x_{nk}) \leq 0$ 说明 target 与 pred 不一样!

所以要更新 w.

① 关于 $y(wx+b)$ 用来判定分类是否正确.

y 为观察到的值.

$wx+b$ 映射到 y 的关系是

$$\begin{cases} 1 & wx+b \geq 0 \\ -1 & wx-b < 0 \end{cases}$$

所以, $wx+b$ 与 y 理应是同号

若出现异号说明当前这个 x 被分错了.

② 同时. $wx+b$ 也代表 x 到超平面的距离.

依据点到超平面的公式.

$$d = \frac{|w^T x+b|}{\|w\|}$$

当对 w 作归一化. $\|w^*\|=1$ 时.

$$d = |w^T x+b|$$

(w^Tx+b) 则表示到超平面的距离和方向.

在更新 w 时, Loss function =

$$-\frac{1}{\|w\|}\sum_{x_i\in M} y_i(w^Tx+b)$$

M 为误差点

$-$ 保证 L.F. > 0

$\frac{1}{\|w\|}$ 距离

y_i: 错误的分类结果.

比如 $w^Tx+b=2$. y 应该 $=1$.

观察到的都是 -1. $\|w\|=1$.

$$-\frac{1}{\|w\|}\sum y(w^Tx+b)=2$$

算法优化的目的是. $\min \underbrace{L(w,b)}_{\text{Loss. Fun.}}$

用 L 分别对 w 和 b 求偏导.

$$\frac{\partial L(w,b)}{\partial w} = \frac{\partial -\sum_{x_i \in M} y_i (w^T x_i + b)}{\partial w} = -\sum_{x_i \in M} y_i x_i$$

$$\frac{\partial L(w,b)}{\partial b} = -\sum y_i$$

如果把 b 加进 w x 矩阵中.

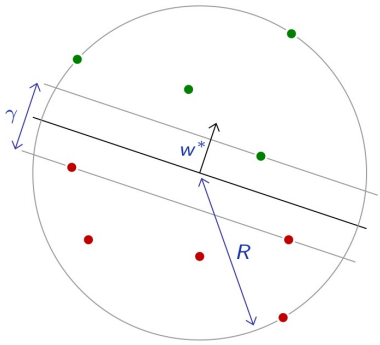
$$w = (b, w_1, w_2 \cdots\cdots, w_n)$$

$$x = (1, x_1, x_2 \cdots\cdots x_n)$$

得到 $w^{k+1} = w^k + y_i x_i$

$(-(-y_i x_i))$

We can get a convergence result under two assumptions:



1. The x_n are in a sphere of radius R:
$$\exists R > 0,\ \forall n,\ \|x_n\| \leq R.$$
2. The two populations can be separated with a margin γ:
$$\exists w^*,\ \|w^*\| = 1,\ \exists \gamma > 0,\ \forall n,\ y_n (x_n \cdot w^*) \geq \gamma/2.$$

1. 有半径为R的球，能把所有样本包含进去.

2. 有一个超平面满足 $\|\hat{w}^*\| = 1$.

$$\hat{w}^{*T}\hat{x} = 0$$

能将数据点完全分开，且最近的点到 hyperplane 的距离为 $\frac{\gamma}{2}$.

1. + 2. $\Rightarrow$ Algo convergence

Proof: Suppose there still is a misclassified sample at iteration k.

We have.

$$w^{k+1} w^* = (w^k + y_{nk} x_{nk}) \cdot w^*$$
$$= w^k \cdot w^* + y_{nk}(x_{nk} \cdot w^*)$$
$$\geq \underline{w^k \cdot w^*} + \frac{\gamma}{2}$$
$$\vdots \text{ by induction.}$$
$$\geq (k+1)\frac{\gamma}{2}$$

$\because \|w^k\| \|w^*\| \geq w^k \cdot w^*$

$\therefore \|w^k\|^2 \geq (w^k \cdot w^*)^2 / \|w^*\|^2$

$$\geq k^2 \gamma^2 / 4$$

若还有misclassified.

$$\|w^{k+1}\|^2 = w^{k+1} \cdot w^{k+1}$$
$$= (w^k + y_{nk} x_{nk})^2$$
$$= w^k \cdot w^k + \underbrace{2 y_{nk} w^k \cdot x_{nk}}_{\leq 0} + \underbrace{\|x_{nk}\|^2}_{\leq R^2}$$

$\because y_{nk} w^k \cdot x_{nk} < 0.$

$$\leq \|w^k\|^2 + R^2$$
$$\leq (k+1)R^2$$

$$\frac{k^2\gamma^2}{4} \leq \|w^k\|^2 \leq kR^2$$

$$k \leq \frac{4R^2}{\gamma^2}$$

在 $\frac{4R^2}{\gamma^2}$ 个 iteration 后，分类必然结束

3-2 Probabilistic view of a linear classifier

方差 Variance：度量单个随机变量的离散程度.

$$\sigma_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

n为样本量. $\bar{x}$为样本均值

协方差 covariance：

用来刻画两个随机变量的相似程度.

$$\sigma(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

两个变量同向反向(正)还是反方向变化?

$|\sigma(x,y)|$ ↑，同向/逆向程度↑

方差可以被视作x关于自身的协方差 $\sigma(x,x)$

方差是协方差的一种特殊形式!!!

⇒同向程度很大

方差→协方差矩阵

给 d 个随机变量 x_k, $k=1,\cdots,d$

x_k 的方差为 $\sigma(x_k,x_k)=\frac{1}{n-1}\sum_{i=1}^{n}(x_{ki}-\bar{x}_k)^2$

$k=1,\cdots,d$

where, x_{ki} : i-th sample

n : sample size

求这些随机变量两两之间的协方差

$$\sigma(x_m,x_k)=\frac{1}{n-1}\sum_{i=1}^{n}(x_{mi}-\bar{x}_m)(x_{ki}-\bar{x}_k)$$

得到协方差矩阵

$$\Sigma=\begin{bmatrix}\sigma(x_1,x_1) & \cdots & \sigma(x_1,x_d)\\ \vdots & & \vdots \\ \sigma(x_d,x_1) & \cdots & \sigma(x_d,x_d)\end{bmatrix}\in\mathbb{R}^{d\times d}$$

↑ symmetric matrix

↓ 对角线上为每个随机变量的方差

多元正态分布与线性转换

Multi-variate Gaussian distribution

假设一个向量X服从均值μ，协方差矩阵为Σ的多元正态分布，则

$$p(x) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

.....

Homoskedasticity 同方差性.

(回归的残差项是随机的) variance不变

↓

residual: (观察值 − 样本估计值)

$$f(x: w, b) = \sigma(w \cdot x + b)$$

$$1 - \sigma(x) = 1 - \frac{1}{1+e^{-x}} = \sigma(-x)$$

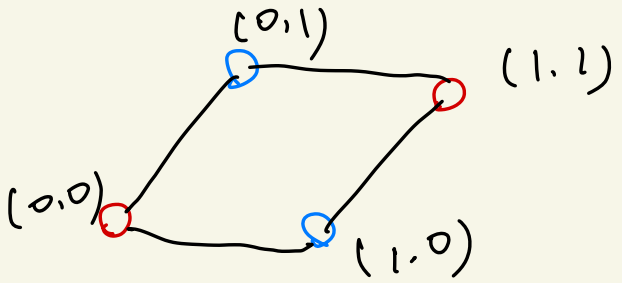
if $y \in \{-1, 1\}$

$$P(Y = y \mid X = x) = \sigma(y(wx+b))$$

3.3.

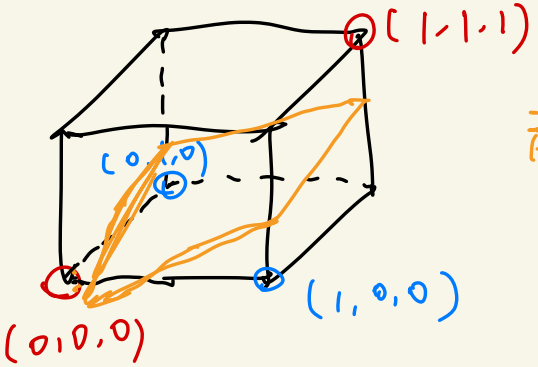
如果有时候数据不可线性分割，可以通过加一维的方式.

Ex.



找不到 linear separator

$\phi: (x_u, x_v) \to (x_u, x_v, x_u x_v)$



超平面.

MLP multi-layer Perceptrons

① A linear classifier:

$$\mathbb{R}^D \to \mathbb{R}$$

$$x \mapsto \sigma(w \cdot x + b)$$

with $w \in \mathbb{R}^D$, $b \in \mathbb{R}$, $\sigma: \mathbb{R} \to \mathbb{R}$

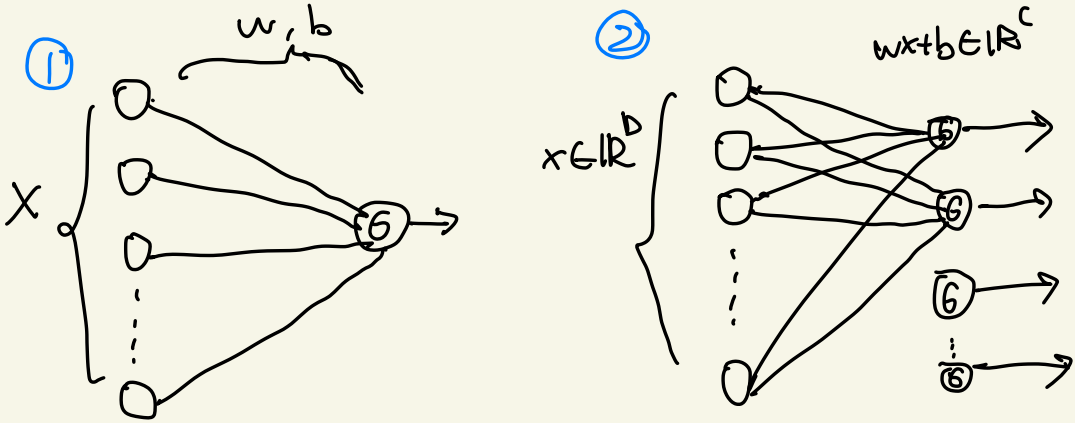
② Component-wise: ??? piece-wise 分段. 分块

$$\mathbb{R}^D \to \mathbb{R}^C$$

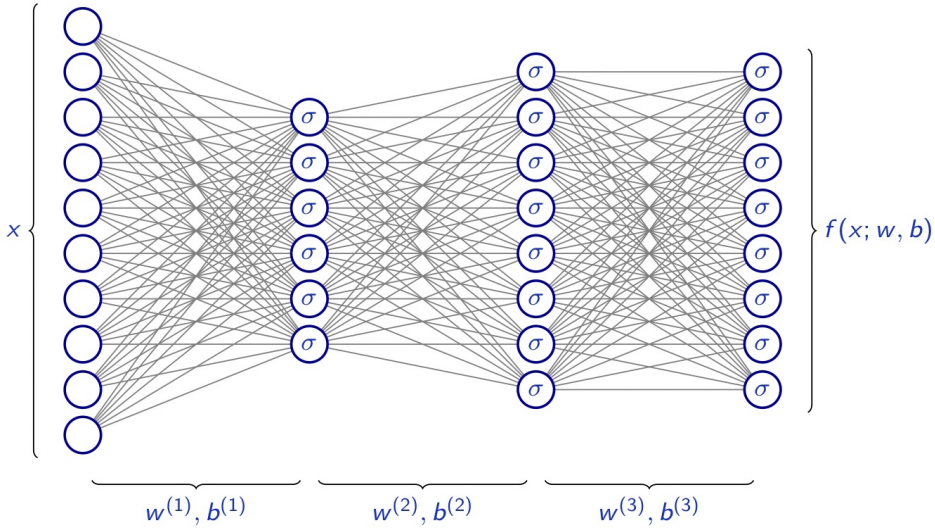
$$x \mapsto \sigma(wx + b)$$

with $w \in \mathbb{R}^{C \times D}$, $w_c, b \in \mathbb{R}^C$

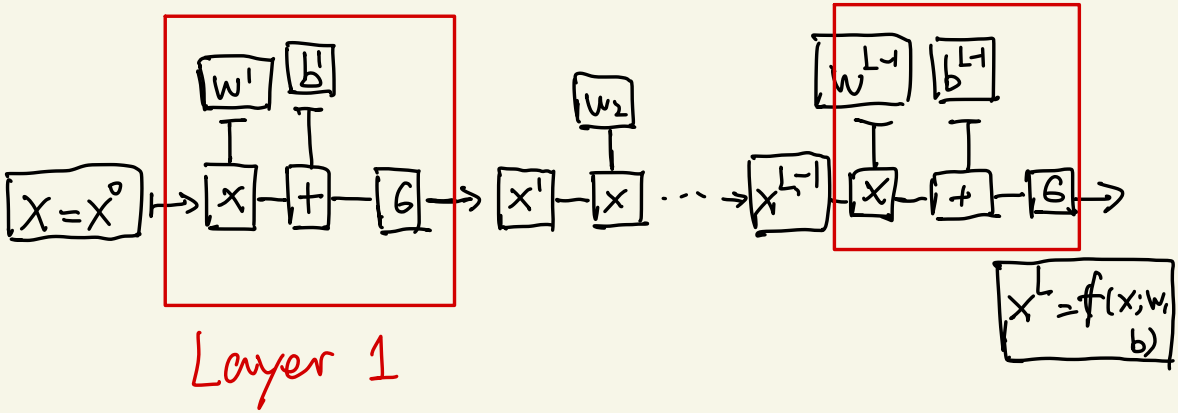
extended to multi-dimension output



Even though it has no practical value implementation-wise, we can represent such a model as a combination of units. More importantly, we can extend it.



MLP Model



$\forall I = 1, \cdots, L \quad x^{(I)} = G(W^I x^{I-1} + b^I)$

and $f(x; w, b) = x^L$

仿射变换＝线性变换+平移!

直线经过变换后仍旧是直线且平行

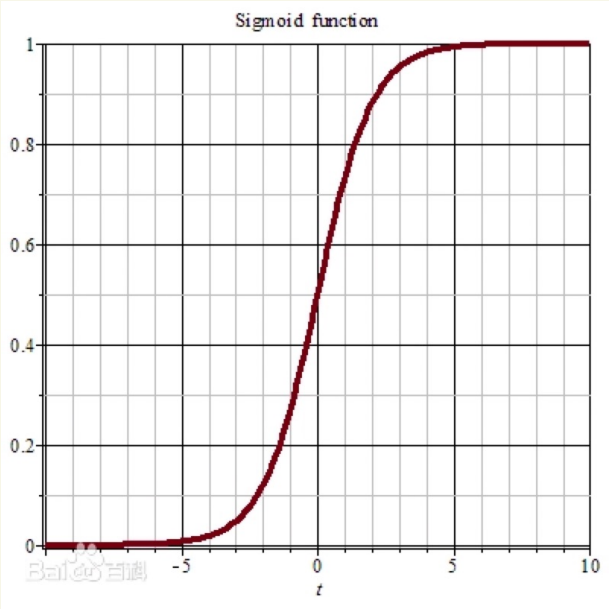
线段的长度比例不变

激活函数不应该是仿射的（通常是非线性的）.

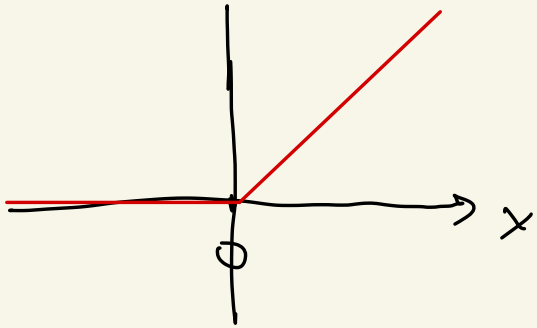
两个常见的激活函数:

1. Sigmoid
2. ReLU

Sigmoid: $S(x) = \frac{1}{1+e^{-x}}$



ReLU. $x \mapsto \max(0, x)$



Universal Approximation

U.A Theorem: 人工神经网络能够以任何精度逼近任何连续函数.

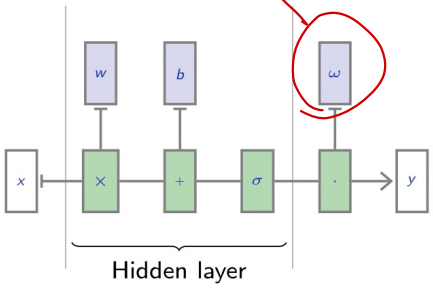
So we can approximate any continuous function

$$\psi : [0, 1]^D \to \mathbb{R}$$

with a one hidden layer perceptron

$$x \mapsto \omega \cdot \sigma(w x + b),$$

where $b \in \mathbb{R}^K$, $w \in \mathbb{R}^{K \times D}$, and $\omega \in \mathbb{R}^K$.



This is the **universal approximation theorem.**

Universal approximation theorem

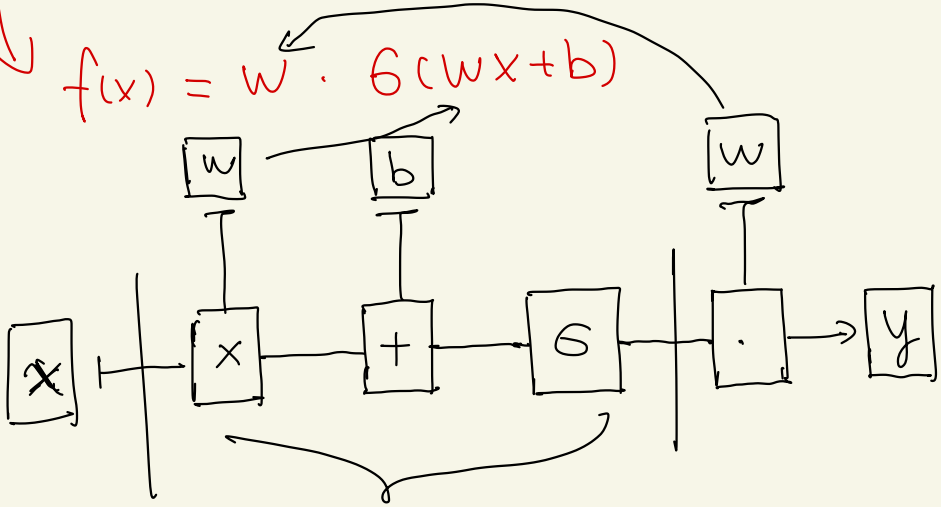
通用近似定理

We approximate any $\psi \in \varphi([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \cdots + \sigma(w_n x + b_n)$$

重复尝试 ReLU 单元的不同组合，直到 $f(x)$ 拟合目标函数.

$$f(x) = w \cdot \sigma(wx + b)$$



Hidden layer.

Hidden layer larger ↑

Approximation precision ↑.

training error ↓

No business with

test error

U.A.T states ↑

Gradient Descent

MSE:

$$L(w,b) = \frac{1}{N}\sum_n (f(x_n; w, b) - y_n)^2$$

← Loss

↓

min

通常是 $\frac{1}{2N}$

↑

For classification, certain regressions...

For logistic regression:

$$P_w = (Y=1 \mid X=x) = \sigma(wx+b)$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \leftrightarrow \text{Sigmoid}$$

$$L(w,b) = -\sum_n \log \sigma(y_n(w \cdot x_n + b))$$

逻辑回归找到分类概率 $P(Y=1)$ 与 x 的关系，比较概率值来判断.

与分界值.

Given a function:

$$f: \mathbb{R}^D \to \mathbb{R}$$
$$x \longmapsto f(x_1, x_2, \cdots, x_D)$$

its gradient:

$$\nabla f: \mathbb{R}^D \to \mathbb{R}$$
$$x \mapsto \left(\frac{\partial f}{\partial x_1}(x), \cdots, \frac{\partial f}{\partial x_D}(x) \right)$$

Back-propagation 反向传播.

假设现在要训练一个MLP模型.

目标: Minimize losses over the training set

Loss function: $L(w,b) = \sum_n l(f(x_n; w,b), y_n)$

如果用 Gradient descent, 那就对 $L(w,b)$ 求关于 w 和 b 的偏导.

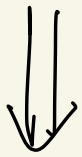
$$\frac{\partial l_n}{\partial w_{ij}^{(l)}} \quad \text{and} \quad \frac{\partial l_n}{\partial b_i^{(l)}}$$

Forward Pass 正向传播

首先引入 $s^{(1)}, s^{(2)}, \cdots, s^{(L)}$

记录在 activation function 前的值.

$$x^{(0)} = x \xrightarrow{w^{(1)} b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \longrightarrow \cdots \xrightarrow{w^{(L)} b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} \rightarrow f(x; w,b)$$



$$\forall I = 1, \cdots, L. \quad \begin{cases} s^{(I)} = w^{(I)} \cdot x^{(I-1)} + b^{(I)} \\ x^{(I)} = \sigma(s^{(I)}) \end{cases}$$

output: $x^{(L)} = f(x; w, b)$

Backpass:

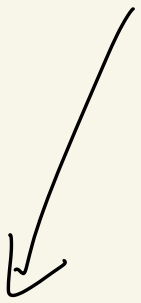
B-P 算法基于 "Chain Rule"

$$(g \circ f)' = (g' \circ f) f'$$

$$x^{(I-1)} \xrightarrow{w^{(I)}, b^{(I)}} s^{(I)} \xrightarrow{\sigma} x^{I}$$

Since $s_i^{(I)}$ influences f only through $x_i^{(I)}$ with

$$x_i^{(I)} = \sigma(s_i^{(I)})$$



We have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \cdot \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)})$$

And since $x_j^{(l-1)}$ influences ℓ only through the $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$$

We have

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} \cdot \frac{\partial s_i^{(l)}}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}$$

Since $w_{i,j}^{(l)}$ and $b_i^{(l)}$ influences ℓ only through $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}$$

We have

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \cdot \frac{\partial s_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)},$$

$$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}$$

Summary: We can compute $\frac{\partial \ell}{\partial x_i^{(L)}}$ from the def. of ℓ, and recursively propagate backward the derivatives of the loss

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'(s_i^{(l)})$$

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}$$

And then compute the derivate

$$\frac{\partial \ell}{\partial w_{ij}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)}$$

$$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}$$

Forward pass

Compute the activations.

$$x^{(0)} = x, \quad \forall l = 1, \dots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right) \end{cases}$$

Backward pass

Compute the derivatives of the loss w.r.t. the activations.

$$\begin{cases} \left[\frac{\partial \ell}{\partial x^{(L)}}\right] \text{ from the definition of } \ell \\ \text{if } l < L, \left[\frac{\partial \ell}{\partial x^{(l)}}\right] = \left(w^{(l+1)}\right)^\top \left[\frac{\partial \ell}{\partial s^{(l+1)}}\right] \end{cases} \qquad \left[\frac{\partial \ell}{\partial s^{(l)}}\right] = \left[\frac{\partial \ell}{\partial x^{(l)}}\right] \odot \sigma'\left(s^{(l)}\right)$$

Compute the derivatives of the loss w.r.t. the parameters.

$$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right] \left(x^{(l-1)}\right)^\top \qquad \left[\frac{\partial \ell}{\partial b^{(l)}}\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right].$$

Gradient step

Update the parameters.

$$w^{(l)} \leftarrow w^{(l)} - \eta \left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right] \qquad b^{(l)} \leftarrow b^{(l)} - \eta \left[\frac{\partial \ell}{\partial b^{(l)}}\right]$$

Pytorch 里，Tensor 有一个 requires_grad 的属性，默认为 False

⇒ 可以用来计算 gradient.

↓

手动设为 True.

↓

只有 floating point type 有梯度

x = [1., 10.]

[3]:
```
t = torch.tensor([1., 2., 4.]).requires_grad_()
u = torch.tensor([10., 20.]).requires_grad_()
a = t.pow(2).sum() + u.log().sum()
```

[4]:
```
torch.autograd.grad(a, (t, u))
```

→ inputs

$\frac{\partial a}{\partial t}$ $\frac{\partial a}{\partial u}$

outputs

```
/usr/local/lib/python3.8/dist-packages/torch/autograd/__init__
on your system. Please check that you have an NVIDIA GPU and i
(Triggered internally at  /pytorch/c10/cuda/CUDAFunctions.cpp:
  return Variable._execution_engine.run_backward(
```

[4]:
```
(tensor([2., 4., 8.]), tensor([0.1000, 0.0500]))
```

$t = [1., 2., 4.]$

$u = [10., 20.]$

$a = \sum t_i^2 + \sum \log(u_i)$

$(\ln x)' = \frac{1}{x}$

$\frac{\partial a}{\partial t} = \left[\frac{\partial a}{\partial t_1}, \frac{\partial a}{\partial t_2}, \frac{\partial a}{\partial t_3}\right]$

$= [2t_1, 2t_2, 2t_3]$

$= [2., 4., 8.]$

```
x = torch.tensor([-3., 2., 5.]).requires_grad_()
u = x.pow(3).sum()
x.grad
u.backward()
x.grad
```

u

→ ? 相当于关于x求导, $3x^2$

→ 当x=[-3, 2, 5]时的值

⇒ tensor([27., 12., 75.])

$x = [-3, 2, 5]$

$u = \sum x_i^3$

$$\frac{\partial u}{\partial x} = \left[\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \frac{\partial u}{x_3}\right]$$

$$= [3x_1^2, 3x_2^2, 3x_3^2]$$

$$= [27, 12, 75]$$

好久没学习了！
48小号，下半学期开始

4.4 Convolution 卷积神经网络

① 来源：有时要处理数据量非常大（多维）的信息，
例如图 image of RGB 256×256. → input
would require $(256 \times 256 \times 3)^2$ parameters.
发现信息在转换（transformation）时有一些共性（invariance）。

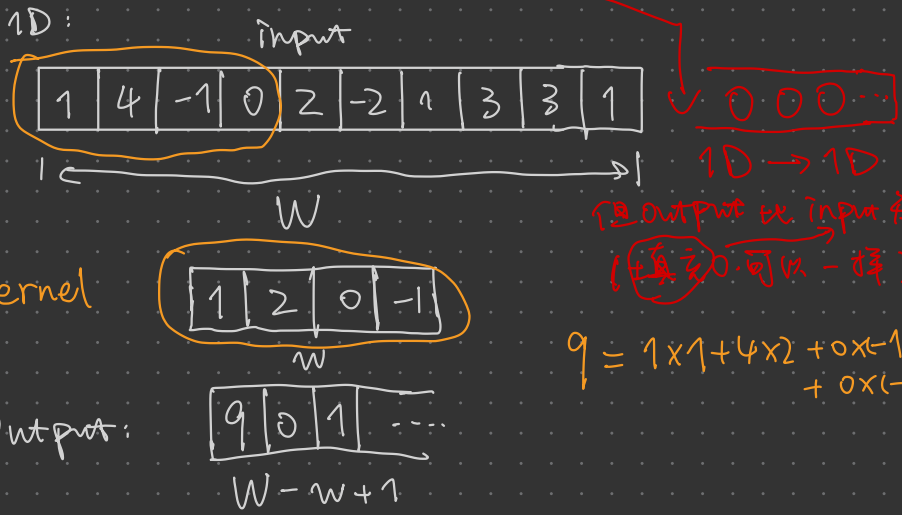
我想将局部共性推广到全局。

A transformation meaningful at a certain location can be applied everywhere!

P.S. convolution 保留进/出结构一致。
（the dimensions are same）.

2d → 2d
3d → 3d

+ 指明位置对应关系



$9 = 1\times1 + 4\times2 + 0\times(-1) + 0\times(-1)$

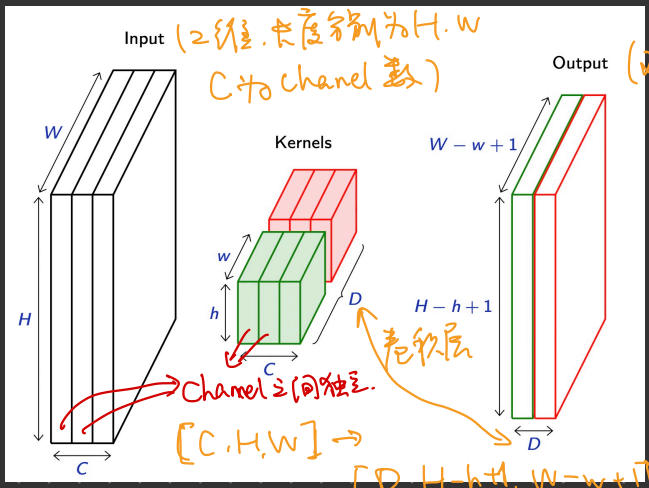
上图的数学表述

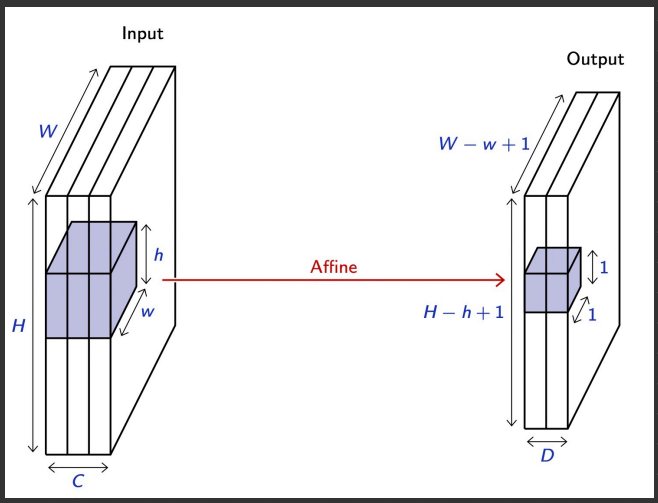
input: $x = (x_1, x_2, \cdots, x_W)$

kernel: $u = (u_1, u_2, \cdots, u_w)$

$(x \circledast u) = (x_i, \cdots, x_{i+w-1}) \cdot u$

② 用途:





1个 channel 叫为 activation map?

③函数：

1. F.conv2d

参数 weight : $D \times C \times h \times W$ （Convolution layer；channel；size of filters）

bias : D

input : $N \times C \times H \times W$

result : $N \times D \times (H-h+1) \times (W-w+1)$ （N：number of samples）

2. torch.nn.Conv2d

参数：

① padding：input 里在0.

(2,1)：上下各在2行，左右各1行0.

② stride：how coarsely the filter should be swiped across the signal.

(2,2)：每隔2个距离，映射一次

③ dilation：filter 的扩张，填0

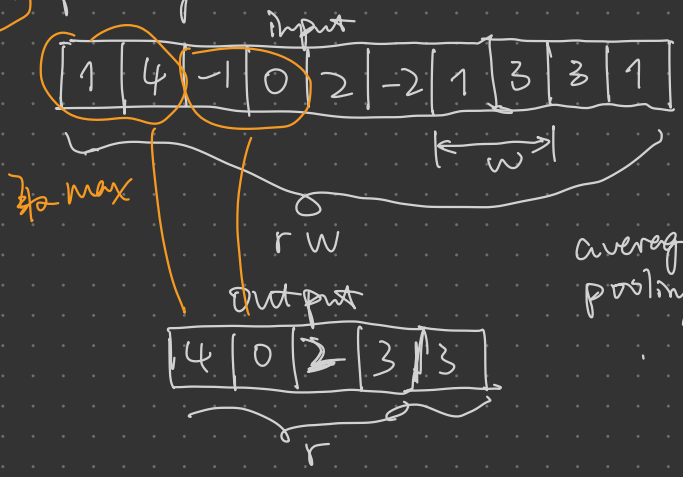
Dilation：1 → 2

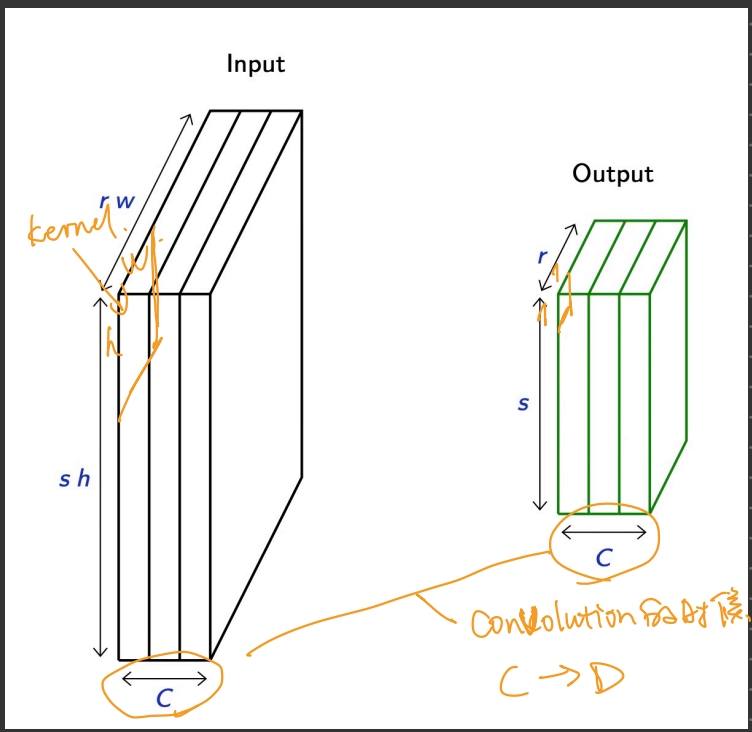
4.5 Pooling

簡化Signal：从高維信号中得到低維信号.

：Grouping → Pooling

① max-pooling：





② 函数：

1. F.max_pool2d：

   input：N × C × H × W

   kernel：(h, w)

   Output：N × C × ⌊H/h⌋ × ⌊W/w⌋

2. torch.nn.MaxPool2d

卷积神经网络 Convolution

Inspiration:

全连接神经网络 ⟺ 卷积 ···

- 全连接神经网络 (full connected) → 参数过多
- 卷积 C. → 解决方案.

卷积神经元:
- width
- height
- depth

不是网络深度, 激活数据体的第三维度.

卷积网络:
- 输入层
- 卷积层
- ReLU层
- 池化层
- 全连接层

(卷积层 = 卷积层 + ReLU层)

卷积层：产生网络中大部分的计算量！

作用：① 卷积层参数 = 可学习的滤波器集合。 不是参数

保证(宽、高较小) 深度一致

② 当作神经元的一个输出。

与左右神经元共享参数

③ 降低参数数量

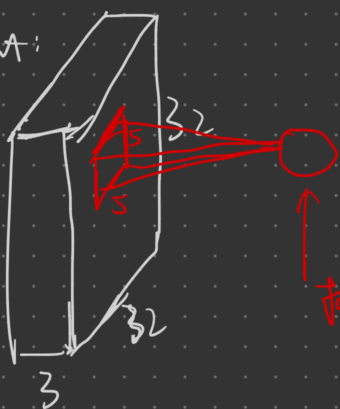
感受野 Receptive field

每个神经元和输入数据局部区域连接。

这个连接空间的大小 → 感受野。

Receptive filed 在深度方向和输入保持一致。

Input:



假设 input 为 RGB 图，大小 [32×32×3]。

R_filed：[5×5]

权重 5×5×3 个 + 1个 bias

MLP

Layers { Input. Hidden. Output. } ↕ Fully connected

5-1

One-hot Coding. 将类别变量转化为二进制等征.

Cross-entropy loss: 交叉熵损失.

描述了两个概率分布之间的距离.