**PYTHON MISC**

convert row vector into column vector

```
Y = Y.reshape(-1,1)
```

**sigmoid**

```
from scipy.stats import logistic
logistic.cdf(x)
```

## LINEAR CLASSIFIERS

boundary line: $\theta \cdot x + \theta_0 = 0$

$\theta \cdot x$ is positive if $x$ is on the right side of the decision boundary

## PERCEPTRON

$y^{(i)} \in \{-1, 1\}$

training error: $\varepsilon(\theta, \theta_0) = \frac{1}{n} \sum 1\left\{ y^{(i)} \left( \theta \cdot x^{(i)} + \theta_0 \right) \leq 0 \right\}$

**algorithm**

```
$\theta = \vec{0}$
for t = 1...T
  for i = 1...n
    if $y^{(i)}\left(\theta \cdot x^{(i)} + \theta_0 \leq 0\right.$ then
      $\theta = \theta + y^{(i)} x^{(i)}$
      $\theta_0 = \theta_0 + y^{(i)}$
```

## REGULARIZATION

we want large boundaries: $\vdots \quad | \quad \vdots$

where $\vdots$ are $\theta \cdot x^{(i)} + \theta_0 = \{1, -1\}$ and $|$ is $\theta \cdot x^{(i)} + \theta_0 = 0$
· prevents overfitting
· the bigger $\|\theta\|$ the faster (= shorter distance) these reach -1 and 1
· we can control the margins distance with $\|\theta\|$

**signed distance**

$\gamma_i = y^{(i)} \frac{\theta \cdot x^{(i)} + \theta_0}{\|\theta\|} = \frac{1}{\|\theta\|}$ or $-\frac{1}{\|\theta\|}$

to maximise $\frac{1}{\|\theta\|}$ we minimise $\frac{1}{2}\|\theta\|^2$

**hinge loss**

$$Loss_h(z) = Loss_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) = max(0, 1 - y^{(i)}(\theta * x^{(i)} + \theta_0))$$
$$= \begin{cases} 0 & \text{if} \quad z \geq 1 \quad \text{correct side, outside margins} \\ 1 - z & \text{if} \quad z < 1 \end{cases}$$

hinge loss improves by moving $\theta$ towards example

**objective**

$J(\theta, \theta_0) = \frac{1}{n} \sum Loss_h\left( y^{(i)} \left( \theta \cdot x^{(i)} + \theta_0 \right) \right) + \frac{\lambda}{2}\|\theta\|^2$

where $\lambda$ is regularisation loss and minimising that term increases margins

$\rightarrow$ by minimising J we balance maximising margins and minimising loss

**gradient descent**

$\theta \Leftarrow \theta - \eta \nabla J(\theta)$

## SOLVING QUADRATIC

if linearly separable and don't allow any errors, can solve quadratic:

$\Rightarrow J(\theta, \theta_0) = \frac{1}{2}\|\theta\|^2$

## STOCHASTIC GRADIENT DESCENT

**update:** sample i at random

**decreasing learning rate** $\eta_T = \frac{1}{1+T}$

$$\theta \Leftarrow \theta - \eta_T \nabla_\theta \left[ Loss_h\left( y^{(i)} \theta \cdot x^{(i)} \right) + \frac{\lambda}{2}\|\theta\|^2 \right]$$

$$= \theta \left[ \begin{cases} 0 & \text{if} \quad Loss_h = 0 \\ -y^{(i)} \cdot x^{(i)} & \text{if} \quad Loss_h > 0 \end{cases} + \lambda\theta \right]$$

## LINEAR REGRESSION

**empirical risk**

$R_n(\theta) = \frac{1}{n} \sum \frac{\left( y^{(t)} - \theta x^{(t)} \right)^2}{2}$

small deviations less penalised than large ones

$\nabla_\theta R_n(\theta) = -\left( y^{(i)} - \theta x^{(t)} \right) \cdot x^{(t)}$

**gradient descent**

```
$\theta = 0$
  for:
    random $t = \{1...n\}$ # stochastic, could also loop
      $\theta = \theta + \eta_k \left( y^{(t)} - \theta \cdot x^{(t)} \right) \cdot x^{(t)}$
```

with $\eta_k = \frac{1}{1+k}$

## NONLINEAR CLASSIFICATION

not linearly separable $\Rightarrow$ map $x$ to higher dimension

**kernels**

i.e. $K(x, x') = \phi(x) \cdot \phi(X') = (x \cdot x') + (x \cdot x')^2$

where $\phi(x) = (x_1, x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$

$\Rightarrow$ dot product is cheap to compute

## COLLABORATIVE FILTERING

assumes X has low rank

$$J(u, v) = \sum_{a,i \in D} \frac{(Y_{ai} - u_a v_i)^2}{2} + \frac{\lambda}{2} \sum^n u_a^2 + \frac{\lambda}{2} \sum^m v_i^2$$

$$X = \begin{bmatrix} \text{user1rating1} & \text{user1rating2} \\ \text{user2rating1} & \text{user2rating2} \end{bmatrix} = u \cdot v^T$$

$\Rightarrow$ u: user's rating tendency
$\Rightarrow$ v: info about movie

**alternating projections** set v, optimise u, set u...
continue until local convergence
(exercise link)

## KERNEL PERCEPTRON

```
$\theta = 0$
for $i = 1...n$
  if $y^{(i)}$ $\theta \cdot \phi\left(x^{(i)}\right)$ $\leq 0$
    $\theta \Leftarrow \theta + y^{(i)} \phi\left(x^{(i)}\right)$
```

$\theta \cdot \phi\left(x^{(i)}\right) = \sum_j^n \alpha_j y^{(j)} \phi\left(x^{(j)}\right) \phi\left(x^{(i)}\right)$

where $\alpha_j$ is nb of mistakes made on $j^{th}$ example

we can then rewrite the perceptron algorithm without having to transform $x \Rightarrow \phi(x)$ anymore

```
$\theta = 0$
for $i = 1...n$
  if $y^{(i)} \sum_j^n \alpha_j y^{(j)} K\left(x^{(j)}, x^{(i)}\right) \leq 0$
    $\alpha_i = \alpha_i + 1$
```

**kernel composition**

· $f: \mathbb{R}^d \rightarrow R$ and $K(x, x')$
$\widetilde{K}(x, x') = f(x) K(x, x') f(x')$
· $K(x, x') = K_1(x, x') + K_2(x, x')$
· $K(x, x') = K_1(x, x') + K_2(x, x')$

**radial basis kernel**

$K(x, x') = \exp\left( -\frac{1}{2}\|x - x'\|^2 \right)$

$\Rightarrow$ correctly classifies every point
$\Rightarrow$ radial in `phi(x) space

**graph classification**
look at number of bends in figure to determine dimension

## K-NEAREST NEIGHBOURS

$\widehat{Y}_{ai} = \frac{1}{K} \sum_{b \in \text{KNN}(a,i)} Y_{bi}$

or weigh by similarity

$\widehat{Y}_{ai} = \frac{\sum_{b \in \text{KNN}(a,i)} sim(a, b) Y_{bi}}{\sum_{b \in \text{KNN}} sim(a, b)}$

**euclidian distance** $\|x_a - x_b\|$

**cosine similarity** $\cos(\theta) = \frac{x_a \cdot x_b}{\|x_a\|\|x_b\|}$

**limitations** my combination of tastes maybe uncommon

## ACTIVATION FUNCTIONS

**ReLU** $f(z) = \max(0, z)$
**tanh** $f(z) = \tanh(z)$
**linear** $f(z) = z$

## MISC

$\vec{a} \cdot \vec{b} = \sum a_i b_i$
$a \cdot a = \|a\|^2 \quad \Rightarrow$
$\theta_z \cdot Ax = \theta_z^T Ax = \left( A^T \theta_z \right) \cdot x$

**distance from point to line**
$d = \frac{|\theta \cdot x + \theta_0|}{\|\theta\|}$

**derivative of norm squared**
$\left( \|h(x)\|^2 \right)' = 2h(x)h'(x)$

**derivative** $[y(\theta \cdot x)]' = yx$
because we derive for each $\theta_i$

**L1 norm** $\|w\|_1 = \sum_i^n |w_i|$

**L2 norm** $\|w\|_2 = \sqrt{\sum_i^n w_i^2}$

**cosine similarity**
$\cos\left( x^{(i)}, x^{(j)} \right) = \frac{x^{(i)} \cdot x^{(j)}}{\|x^{(i)}\|\|x^{(j)}\|}$

**euclidean distance**
$dist\left( x^{(i)}, x^{(j)} \right) = \left\| x^{(i)} - x^{(j)} \right\|$

## DEEP LEARNING

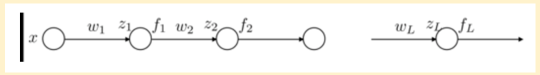
· intermediate layer is like feature representation: "learning $\phi$"
· activation layer acts like linear classifier

**backpropagation**
compute partial derivatives using chain rule to do gradient descent

$$w_1 \Leftarrow w_1 - \eta\left(\frac{\delta \text{Loss}}{\delta w_1}\right)$$

$$\frac{\delta \text{Loss}}{\delta w_1} = \frac{\delta \text{Loss}}{\delta f_1} \cdot \frac{\delta f_1}{\delta z_1} \cdot \frac{\delta z_1}{\delta w_1}$$

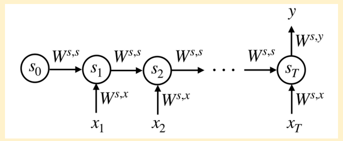
risks are vanishing or exploding gradients with long chains



## RNN

· same params for each layer
· variable number of layers
⇒ encodes sentences

$s_t = \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$



RNN have some gradient issues

**gating network**
in default RNN we overwrite $s_t$ but sometimes it's useful to retain control over what's overwritten at each step

$g_t = \text{sigmoid}(W^{g,s} s_{t-1} + W^{g,x} x_t)$
$s_t = (1 - g_t) \odot s_{t-1} + g_t \odot \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$
where $\odot$ is element-wise multiplication

$(1 - g_t): \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ i.e. $g_t$ is use to control what we retain from previous layer

## MARKOV MODELS

<beg> $w_1$ $w_2$ $w_3$ <end>
$w_0$ $w_L$

$P(w_1, \ldots, w_L) = P(w_1 \mid w_0) \ldots P(w_L \mid w_{L-1})$

with $P(w' \mid w) = \dfrac{\text{count}(w, w')}{\sum_{\tilde{w}} \text{count}(w, \tilde{w})}$

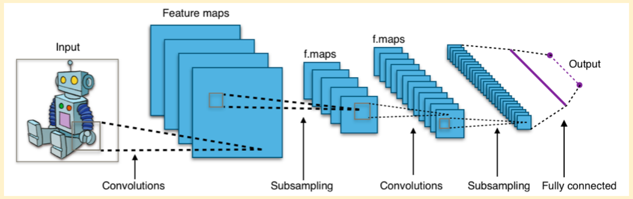
**RNNs for sequences**
variable history

$\phi(\text{word}) \;\vdots\; X^W \;\vdots^{s_t}\; X \;\vdots\; p$
$s_{t-1} \;\vdots\; X$

i.e. we're feeding the previous state as input to the next

## CNN

**convolution**
· classify by small patches
· slide over input
· use the same weights for each patch to build feature map
**pooling (subsampling)**
· stores *if* there's an activation for feature in that area
· stride



**cross-correlation/convolution**
we don't actually need to flip $g(t)$ as we would with convolution, because we're learning the filter, the values are just flipped
**1d** $f \odot g(t) = \sum f(T) g(T + t)$
**md** $f \odot g(x, y) = \sum_{T_1} \sum_{T_2} f(T_1, T_2) g(T_1 + x, T_2 + y)$

## K-MEANS

1) randomly assign $z^{(1)} \ldots z^{(2)}$
2) iterate
   a) assign x to closest z
   $$\text{Cost}\left(z^{(1)} \ldots z^{(k)}\right) = \sum_{i=1}^{n} \min_{j=1 \ldots k} \left\| x^{(i)} - z^{(j)} \right\|^2$$
   b) given $C_1 \ldots C_k$ find best rep. z in each C
   $$\text{Cost}(C_1 \ldots C_k) = \min_{z^{(1)} \ldots z^{(k)}} \sum_{j=1}^{k} \sum_{i \in C_j} \left\| x^{(i)} - z^{(j)} \right\|^2$$

**problems with K-means**
· points are not in set of Xs
· $(\text{euclidean distance})^2$ only
· K-means is $O(n \cdot k \cdot d)$ for one update step (2)

**find best $z_j$ for $C_j$**
$$\frac{\delta}{\delta z_j} \sum_{i \in C_j} \left\| x^{(i)} - z^{(j)} \right\|^2 = 0$$
$$z^{(j)} = \frac{\sum x^{(i)}}{|C_j|}$$

where $|C_j|$ is size of $C_j$

**impact of init**
· init matters
· we can end up with undesirable convergence

## K-MEDIANS

Like K-means but with medians. Compute median in each single dimension in the Manhattan distance formulation (see ex.)

## K-MEDOIDS

1) pick init $z^{(i)} \ldots z^{(j)}$ from points X
2) iterate until no change in cost
   a) for $i = 1 \ldots n$
   $$C_j = \left\{ i : z^{(j)} \text{ is closest to } x^{(i)} \right\}$$
   b) pick the most central $x^{(i)}$ in $C_j$ to be $z^{(j)}$

   for $j = 1 \ldots k$
   $$z_j = \left\{ x^{(i)} \ldots x^{(n)} : \sum_{i \in C_j} dist\left(x^{(i)}, z^{(j)}\right) \text{ is minimal} \right\}$$

· same idea as K-means but $z_j$ is in x and can use any distance measure
· K-medoids is $O\left(n^2 \cdot k \cdot d\right)$ for update step (2)

⚠ step 2 gets executed even if no change and cost, and order we cycle in matters and can update the medoid (see ex.)

## GENERATIVE MODELS

$p(w \mid \theta) = \theta w$: probability of generating word $w$ given all possibilities`

**probability of generating doc D**
$$p(D \mid \theta) = \prod_{i=1}^{n} \theta_{w_i} = \prod_{w \in W} \theta_w^{\text{count}(w)})$$

**MLE**
max $P(D \mid \theta) \rightarrow \sum_{w \in W} \text{count}(w) \log(\theta_w)$

$$\Rightarrow \tilde{\theta}_w = \frac{\text{count}(w)}{\sum_{w' \in W} \text{count}(w')}$$

**prediction**
essentially acts as a linear classifier through origin

$$\log \frac{p(D \mid \theta^+)}{p(D \mid \theta^-)} = \begin{cases} \geq 0 & + \\ < 0 & - \end{cases}$$

**prior, posterior, likelihood**
for when we have same prior knowledge of document classification

$$P(y = + \mid D) = \frac{P(D \mid \theta^+) P(y = +)}{P(D)}$$

⇒ linear classifier with offset influenced by prior

⚠ N-1 params in multinomial (i.e. remember the binomial uses 1 param for two values)

## GAUSSIAN GENERATIVE MODELS

$X \mid \mu, \sigma^2 \sim N\left(\mu, \sigma^2\right)$

want to find $\mu$ and $\sigma^2$ that give highest likelihood to training data.

$\Rightarrow \frac{\delta l_n}{\delta \mu} = 0$ and $\frac{\delta l_n}{\delta \sigma^2} = 0$

$\widehat{\mu} = \frac{1}{n} \sum^n x^{(i)}$ and $\widehat{\sigma^2} = \frac{1}{nd} \sum^n \left\| x^{(i)} - \mu \right\|^2$

## MIXTURE MODELS

**mixture components** K: $N\left(x, \mu_j, \sigma_j^2\right)$

**mixture weights** $p_1 ... p_k$ $\sum p_j = 1$

$\theta = p_1 ... p_k, \mu_1 ... \mu_k, \sigma_1^2 ... \sigma_k^2$

$p(x \mid \theta) = \sum_{j=1}^k p_j N\left(x, \mu_j, \sigma_j^2\right)$

$p(S_n \mid \theta) = \prod_{i=1}^n \prod_{j=1}^k p_j N\left(x, \mu_j, \sigma_j^2\right)$

## EM algorithm

"random" init of $\theta$: $\mu^{(i)}, \sigma^{(i)}, p_i$
init matters for speed and outcome

· converges locally
· good idea to use k-means for init

**expectation step**

$p(j \mid i) = \frac{p_j N\left(x_i, \mu_j, \sigma_j^2 I\right)}{p(x \mid \theta)}$

$\Rightarrow$ probability that point i in cluster j

$\Rightarrow$ $p(x \mid \theta) = \sum_{j=1}^k p_j N\left(x_i, \mu_j, \sigma_j^2 I\right)$

**maximisation step**
re-estimate parameters, (nb: cluster j, point i)

$\widehat{n}_j = \sum_{i=1}^n p(j \mid i)$

$\hat{p}_j = \frac{\widehat{n}_j}{n}$

$\widehat{\mu}_j = \frac{1}{\widehat{n}_j} \sum_{i=1}^n p(j \mid i) x^{(i)}$

$\widehat{\sigma}_j^2 = \frac{1}{\widehat{n}_j d} \sum_{i=1}^n p(j \mid i) \left\| x^{(i)} - \mu^{(j)} \right\|^2$

## MARKOV DECISION PROCESS

· states $s \in S$
· actions $a \in A$
· action dependent on transition probabilities

$T(s, a, s') = P(s' \mid a, s)$
$\sum_{s' \in S} T(s, a, s') = 1$

$R(s, a, s')$: reward for starting s, doing a, ending s'

## UTILITY FUNCTIONS

**finite horizon (not important)**

count reward for next n steps: $U(s_0 ... s_n) = \sum^n R(s_i)$

**infinite horizon/discounted reward**

$U(s_0 ... s_n) = \sum_{k=0}^\infty Y^k R(s_k) \leq \frac{R_{\max}}{1 - \gamma}$

**value function**

$V^\star(s)$: expected reward if agent acts optimally starting at s

$\pi: S \to A$: assigns action $\pi(s)$ to state s
optimal policy $\pi^\star$ assigns action at every state that maximises expected utility

$Q^\star(s, a)$: expected reward starting at s, taking action a and acting optimally

## BELLMAN EQUATIONS

$V^\star(s) = \max_a Q^\star(s, a) = Q^\star(s, \pi^\star(s))$

$Q^\star(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^\star(s')]$

## VALUE ITERATION ALGORITHM

$V_k^\star(s) \to_{k \to \infty} V(s)$

· init: $V_0^\star(s) = 0$
· iterate until $V_k^\star(s) \approx V_{k+1}^\star(s) \forall s$

$V_{k+1}^\star = \max_a \left[ \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k^\star(s') \right] \right]$

· compute $Q^\star(s, a)$

then $\pi^\star(s) = arg \max_a Q^\star(s, a)$

## Q-VALUE ITERATION

same as above but with Q and the update is:

$Q_{k+1}^\star = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k^\star(s', a') \right]$

$\Rightarrow$ we're summing over all possible states we may land in

## RL

typically won't have T and R ahead of time

**Q-value iteration**
want to calculate estimates for Q but don't have T and R.
we'll use **exponential running average** to get estimates for $Q(s, a)$:

$Q_{i+1}(s, a) = \alpha \cdot \text{sample} + (1 - \alpha) Q_i(s, a)$
sample: $R(s, a, s') + \gamma \max_{a'} Q_i(s', a')$

1) **initialisation** $Q(s, a) = 0 \forall a$

2) **iterate until convergence**

a) collect samples: s,a,s',R(s,a,s')

b) $Q_{i+1}(s, a) = \alpha \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] + (1 - \alpha) Q_i(s, a)$

**$\varepsilon$ greedy**

balance exploration and exploitation

· random action w.p. $\varepsilon$
· best current action w.p. $(1 - \varepsilon)$

$\varepsilon$ should decay over time