COMP 540 Final Exam Review

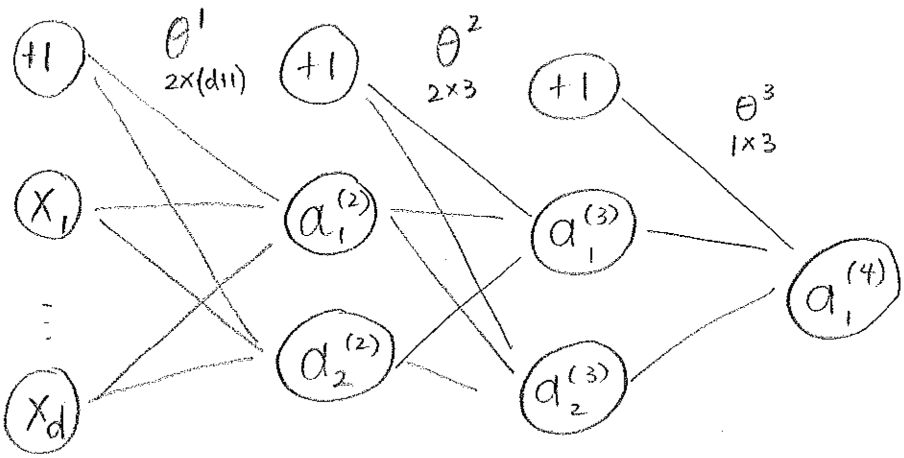
April 26

Basic Neural Networks

notation: $a_j^{(\ell)}$ - activation of j^{th} unit in ℓ^{th} layer

$\theta^{(\ell)}$ - parameters mapping layer ℓ to $\ell+1$

* the dimension of θ is always #nodes in next layer (no bias) $\times$ #nodes in previous layer (includes bias)



Forward Propogation

$a^{(1)} = X$ $\quad$ $z^{(2)} = \underset{2\times(d+1)}{\theta^{(1)}} * \underset{(d+1)\times1}{[1; X]}$ $\quad$ $z^{(3)} = \underset{2\times3}{\theta^{(2)}} * \underset{3\times1}{[1; a^{(2)}]}$

$a^{(2)} = \text{ReLU}(z^{(2)})$ $\quad$ $a^{(3)} = \text{ReLU}(z^{(3)})$

general step

$z^{(\ell+1)} = \theta^{(\ell)} * [1; a^{(\ell)}]$

$a^{(\ell+1)} = \text{ReLU}(z^{(\ell+1)})$

$$a^{(4)} = g(\theta^3 [1; g(\theta^2 [1; g(\theta^1 [1; x])])]) = g(h_\theta x)$$

* any continuous function can be approximated by 3 layers. 4 layers approximates any fxn

ost $\quad J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k \log(h_\theta(x))_k + (1-y_k)\log(1-h_\theta(x))_k + \frac{\lambda}{2m}\sum_{\ell=1}^{L-1}\sum_{k=1}^{S_\ell}\sum_{j=1}^{S_{\ell+1}} \theta_{ji}^{(\ell)\,2}$

$$\frac{\partial J(\theta)}{\partial \theta} = -\frac{1}{m}\sum_{i=1}^{m} \underbrace{(y^{(i)} - h_\theta(x^{(i)}))}_{\text{error in output}} x^{(i)}$$

K - # output units (final output)

S_ℓ - # units in layer ℓ

Backward Propogation

addendum: for $i = 1$ to m (training examples)

⓪ set $a^{(1)} = x^{(i)}$

Need: $J(\theta)$ and $\frac{\partial J}{\partial \theta_{ij}}$

① Compute forward prop $a^{(2)} \cdots a^{(L)}$

② δ = error of node j in layer ℓ, error in activation

ex: $\delta_j^{(4)} = a_j^{(4)} - y_j$, $\delta^{(3)} = (\theta^3)^T \delta^{(4)} .* \underbrace{g'(z^{(3)})}_{\text{derivative}}$

$a^{(3)} .* (1 - a^{(3)})$

- compute $\delta^{(L-1)} \cdots \delta^{(2)}$ using $\delta^{(\ell)}$

③ $\Delta_{ij}^{(\ell)} := \Delta_{ij}^{\ell} + a_j^{(\ell)} \delta_i^{(\ell+1)}$

because of the fact that $\frac{\partial J}{\partial \theta_{ij}^{(\ell)}} = a_j^{(\ell)} \delta_i^{(\ell+1)}$

* delta vals are derivative of cost fxn

(this is w/o regularization)

end loop

vectorized:

$\Delta^{(\ell)} := \Delta^{(\ell)} + \delta^{(\ell+1)} (a^{(\ell)})^T$

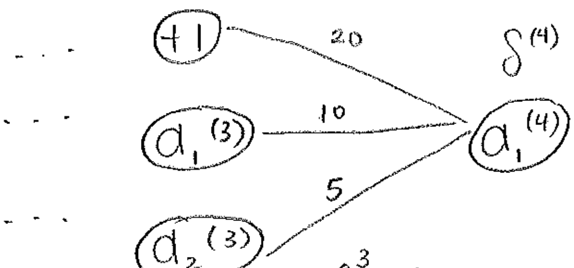
$D_{ij}^{(\ell)} := \frac{1}{m} \Delta_{ij}^{(\ell)} + \lambda \theta_{ij}^{(\ell)}$ if $j \neq 0$

$D_{ij}^{(\ell)} := \frac{1}{m} \Delta_{ij}^{(\ell)}$ if $j = 0$

} accumulates partial derivatives

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(\ell)}} = D_{ij}^{(\ell)}$$

Visually:



$\delta^{(4)} = a_1^{(4)} - y_1 = 7$

$\theta^3 = [20 \quad 10 \quad 5]$

1x3

$$\delta_1^{(3)} = \theta^{(3)T} \delta^{(4)} .* \underbrace{g'(z^{(3)})}_{a^{(3)} .* (1 - a^{(3)})}$$

$$\underset{3\times1}{\delta_1^{(3)}} = \underbrace{\begin{bmatrix} 20 \\ 10 \\ 5 \end{bmatrix} [7]}_{3\times1} .* \underset{3\times1}{[\;]} \times \underset{3\times1}{[\;]}$$

discard δ_0 for bias unit!!

NN Tactics, Specifications, Hyperparameters

Constructing a network:

- \# of input units = dimensions of $X^{(i)} \in 1$ to d
- \# of output units = # classes
- \# of hidden units/ layer
- recommend same # units in each hidden layer

Training a network:

1. randomly initialize weights
2. forward prop — ? unclear about steps
3. implement cost function
4. back prop to compute partial derivatives
5. grad checking
6. gradient descent to min cost function

* for $i = 1$ to m (do for each example)

FP
BP

Tips:

- Mean-subtract, zero center, normalize (div by std)
  - get mean/std from train set and <u>USE THEM</u> for val/test
- Batch Normalization: normalize activation of units to have $\sim N(0,1)$ dist at beginning of training $a_j^{(l)} \sim N(0,1)$
  - reduces dependence of gradients on scale of parameters or init values
  - can regularize
  - can lead to faster convergence
- L2 reg
  - sum of squared errors
- L1 reg
  - sparse weight vectors
  - used for feature selection
- overfit on small subset of data
  100% on small train

- moniter learning rate
- dropout: unit active w/ prob p
- dropconnect: some weights set to 0 during forward pass
- gradient checks!!
  numerical appoximations
  $\frac{f(x+h)-f(x-h)}{2h}$

Update momentums (SGD, nesterov, ADAM, etc)

SGD $X \mathrel{+}= -\alpha \frac{\partial J}{\partial \theta}$ $\qquad \alpha$ = learning rate

Momentum $\mu * v - \alpha \frac{\partial J}{\partial \theta}$

$X \mathrel{+}= \mu * v - \alpha \frac{\partial J}{\partial \theta}$

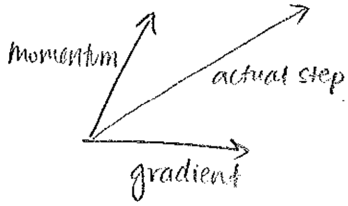
get over bumps in loss functions

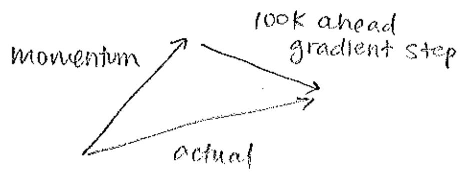
Nesterov momentum

$X_{ahead} = X + \mu v$

$v = \mu v - \alpha \frac{\partial J}{\partial \theta}$ (ahead)

$X \mathrel{+}= v$





Tips (more):

- annealing the learning rate
  - reduce step by 0.1 every 20 epoch
  - exponential decay

Adaptive Learning Rates

1) RMSProp: moving avg of squared gradients
2) Adam: smoothed RMS Prop w/ momentum
3) Adagrad: also depends on gradients

Convolutional Neural Networks

- a network learns filters that are local in width/height but full in depth
- hyperparams: depth, stride, and zero-padding

CONV

1) depth: # units in CONV that connects to one same input region
2) stride: small stride = largely overlapping receptive fields / large output vols
3) padding: pad input w/ zeros all around → control spatial size

- input volume of size $W1 \times H1 \times D1$
  - # filters K
  - receptive field size F
  - stride S
  - zero-pad P

- produces output volume of size $W2 \times H2 \times D2$

$$W2 = \left[\frac{W1 - F + 2P}{S}\right] + 1$$

$$H2 = \left[\frac{H1 - F + 2P}{S}\right] + 1$$

$$D2 = K$$

$F \times F \times D1$ weights per filter

total $F \times F \times D1 \times K$ weights and K bias units

* can dramatically reduce # params by making assumption that the same weights and biases are used across depth slices

---

Common to insert pool layer after conv.

POOL

- 2×2 filter w/ size 2 → take max over 4 numbers
- depth is same

params: · receptive field size F
· stride S

pooling is downsampling!!

input vol: $W1 \times H1 \times D1$

output: $W2 \times H2 \times D2$

$$W2 = \frac{(W1 - F)}{S} + 1$$

$$H2 = \frac{(H1 - F)}{S} + 1$$

$$D2 = D1$$

More Notes about NN

activation functions:

1) sigmoid (σ)
   - saturate & kill gradients at 0 or 1
   - not zero centered outputs (they are 0.5)

2) tanh
   - also saturate activations
   - squash [-1, 1]
   - zero-centered !! ✓

3) ReLU
   - $f(x) = \max(0, x)$
   - linear, non saturating
   - accelerates SGD convergence
   - gradients could die b/c of gradients that are huge ⟶ push to zero

4) Leaky ReLU
   - ReLU w/ a $x<0$ slope component

Conv Net Tips

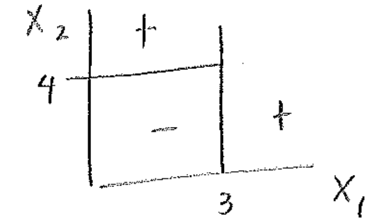
- Data augmentation
- ReLU
- dropout
- mini-batch grad descent
- take out layers and see what impact it has on accuracy
- too deep: weights at beginning become meaningless + small

April 27

Decision Trees

- type of adaptive basis function

ex:



$h_\theta(X) = \theta^T \phi(X)$ where $\phi_1(X) = I[X_1 < 3 \text{ and } X_2 > 4] = +$

$\phi_2(x) = I[X_1 < 3 \text{ and } X_2 \leq 4] = -$

$\phi_3(X) = I[X \geq 3] = +$

$$h_\theta(X) = \theta_1\, \phi_1(X) + \theta_2\, \phi_2(x) + \theta_3\, \phi_3(x)$$

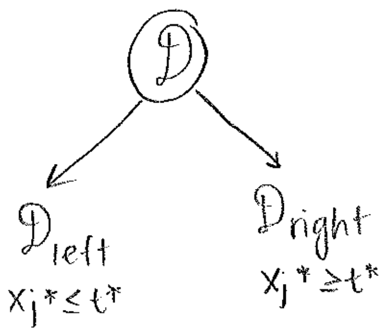
- must pick attribute that allows greatest reduction in cost
- too deep of a tree → overfitting
- highly unstable
- depth = hyperparameter. use validation to prune on overfit model

X_j^*'s − features

t^*'s − thresholds

feature selection: (maximizing reduction in cost due to split)

$$j^*, t^* = \underset{\substack{j \in \{1, \cdots d\} \\ t \in \text{values of } X_j}}{\operatorname{argmax}} \underbrace{\left\{ \operatorname{cost}(\mathcal{D}) - \underbrace{\left[\frac{|\mathcal{D}_{left}|}{|\mathcal{D}|} \operatorname{cost}(\mathcal{D}_{left}) + \frac{|\mathcal{D}_{right}|}{|\mathcal{D}|} \operatorname{cost}(\mathcal{D}_{right}) \right]}_{\text{cost of split}} \right\}}_{\text{reduction in cost due to split = info gain due to split}}$$



<u>Regression cost</u> $y \in \mathbb{R}$

$$\operatorname{cost}(\mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \bar{y})^2$$

every feature = $O(D)$ in computation

<u>classification cost</u>

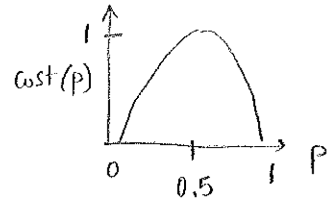
$y \in \{0, 1\}$

$$\operatorname{cost}(\mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} I(y^{(i)} = \hat{y})$$

$$\hat{y} = \underset{c}{\operatorname{argmax}}\ I(y^{(i)} = c)$$

Entropy

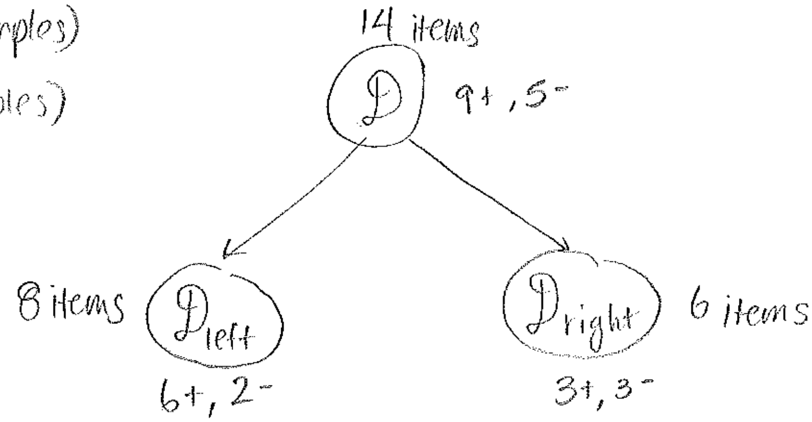
$$cost(\mathcal{D}) = -p\log p - (1-p)\log(1-p)$$



p = # pos examples

example: 9 + (pos examples)

5 − (neg examples)



$$cost(\mathcal{D}) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$$

$$cost(\mathcal{D}_{left}) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} = 0.811$$

$$cost(\mathcal{D}_{right}) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1$$

Information gain: $cost(\mathcal{D} - \left[\frac{|\mathcal{D}_{left}|}{|\mathcal{D}|}cost(\mathcal{D}_{left}) + \frac{|\mathcal{D}_{right}|}{|\mathcal{D}|}cost(\mathcal{D}_{right})\right])$

$$= 0.94 - \underbrace{\left[\frac{8}{14}(0.811) + \frac{6}{14}(1)\right]}_{\text{cost of split}} = \text{reduction in cost } \underline{\text{OR}} \text{ info gain}$$

Gini Index

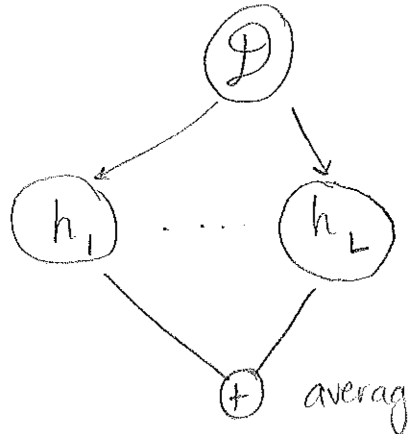
$$cost(\mathcal{D}) = 2p(1-p)$$

$p = \frac{9}{14}$

* read more on this!!

$$= \frac{18}{14}\left(\frac{5}{14}\right)$$

Ensemble Models



bootstrapping/ bagging
$\Downarrow$
Sample w/ replacement
m times

averaging over h

$$h_{bag}(x) = \frac{1}{L}\sum_{\ell=1}^{L} h_\ell(x)$$

$$h_\ell(x) = \theta_\ell^T X$$

serves to uncorrelate samples and errors (not always perfect in reality)

$h_{bag}(x) \rightarrow$ majority vote among $h_1(x) \cdots h_L(x)$

* Error of bagged ensemble is lower than average expected error

classification:

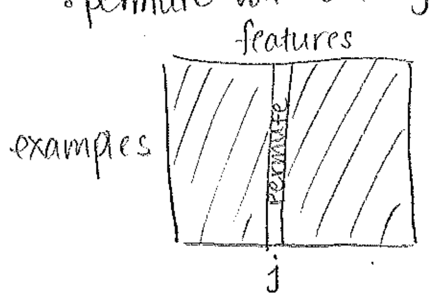
$$P\left(\substack{\text{ensemble}\\\text{error}}\right) = \sum_{K=\frac{L+1}{2}}^{L} \binom{L}{K} \varepsilon^K (1-\varepsilon)^{(L-K)}$$ } error of bagged ensemble w/ L members

regression:

$$P\left(\substack{\text{ensemble}\\\text{error}}\right) = \underbrace{\left[\frac{1}{L}\sum_{\ell=1}^{L}\left(f(x)+\varepsilon_\ell(x)\right)\right]}_{\frac{1}{L}\sum_{\ell=1}^{L} h_\ell(x) \,=\, h_{bag}(x)} - f(x)$$

Random Forests

- decision tree of depth $\mathcal{D}$
- $\sqrt{d}$ randomly chosen features; to see which features are most important, do:
    - permute values in j^{th} column



$\rightarrow$ if permuting j has no effect on decision tree...

$\rightarrow$ PRUNE! don't need to split on that feature

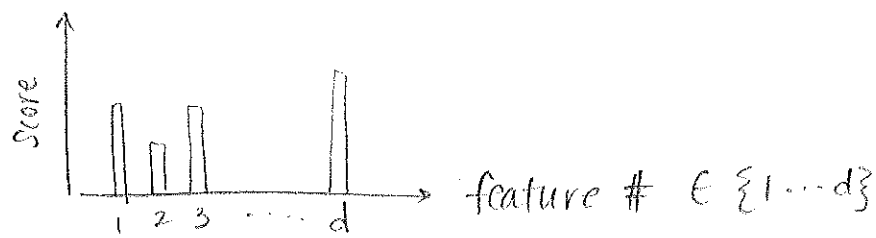
Feature Importance cont'd

OOB error of an ensemble: (out of bag)

1. cycle over examples
2. find decision trees where example is NOT used (out of bag, OOB)
3. for $i = 1$ to m
   - use OOB classifier tree to vote on x
   - check if majority vote $== y^{(i)}$
   - this is OOB error
4. $\text{Score}(f) = \text{OOBerror}(X) - \text{OOBerror}(X \mid f \text{ permute})$

   $f \in \{1 \cdots d\}$ features

   higher score → feature more important



Boosting: another method of uncorrelating groups of examples
- associate a weight w/ each example

classify $\mathcal{D} \cdot w_1 \quad \mathcal{D} w_2 \quad \cdots \quad \mathcal{D} w_L$

$\downarrow \qquad \downarrow \qquad \downarrow$

predict $\alpha_1 h_1 \quad \alpha_2 h_2 \quad \alpha_L h_L$

↑ weights on mistakes

↓ weights on correctly classified examples

$$h_{\text{ensemble}}(x) = \text{sgn}\left(\sum_{\ell=1}^{L} \alpha_\ell h_\ell(x)\right)$$ weighted vote of h from each classifier

Boosting con't

algorithm:

1. initialize $w_1^{(i)} = \frac{1}{m}$ for $i = 1 \cdots m$ (all examples have same weight)
2. for $l = 1$ to L do:
   - learn h_l to minimize $J_l = \frac{1}{m} \sum_{i=1}^{m} W_l^{(i)} \mathbb{I}(h_l(X^{(i)}) \neq y^{(i)})$

     [minimize the # of wrongly classified examples]
   - calculate error rate of h_l: $\varepsilon_l = \dfrac{\sum_{i=1}^{m} W_l^{(i)} \mathbb{I}(h_l(X^{(i)}) \neq y^{(i)})}{\sum_{i=1}^{m} W_l^{(i)}}$ → [this is just normalization]

   ★ if $\varepsilon_l \geq 0.5$, break out of loop! (termination criteria)

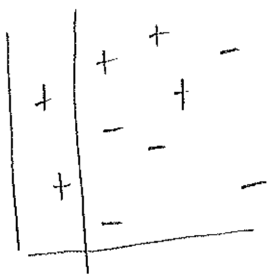
   - calculate $\alpha_l = \frac{1}{2} \ln \left\{ \frac{1 - \varepsilon_l}{\varepsilon_l} \right\}$
   - update example $1 \cdots m$ weights!

     → $W_{l+1} = \begin{cases} W_{l+1}^{(i)} = W_l^{(i)} \exp(-\alpha_l) & \text{if } X^{(i)} \text{ correctly classified} \\ W_{l+1}^{(i)} = W_l^{(i)} \exp(+\alpha_l) & \text{if } X^{(i)} \text{ incorrectly classified} \end{cases}$

Working out an example:

10 "examples"

2 iterations



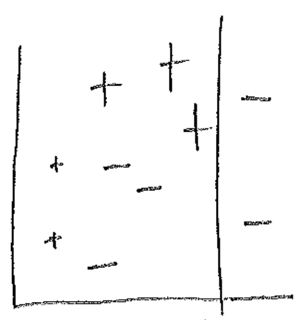
h_1

all $W_l = 0.1$

$\varepsilon_l = 0.3$

$\alpha_l = \frac{1}{2} \ln \left(\frac{0.7}{0.3} \right) = 0.42$

$\Longrightarrow$



h_2

$\varepsilon_l = 0.21$

$\alpha_l = \frac{1}{2} \ln \left(\frac{0.79}{0.21} \right) = 0.65$

updates:

$W_{l+1} \begin{cases} W_l e^{-0.42} & \text{if } x \text{ correct} \\ W_l e^{0.42} & \text{if } x \text{ incorrect} \end{cases}$

$\mathrm{sgn}(\alpha_1 h_1 + \alpha_2 h_2 \cdots)$

Adaboost (notes are spotty b/c I was PMS-ing that day ...)
... aka in a bad mood :)

$$J = \sum_{i=1}^{m} \exp(-y^{(i)} H_\ell(x^{(i)})) \quad \text{where } H_\ell(x) = \sum_{j=1}^{\ell} \alpha_j h_j(x) \quad \text{where } h_j(x) \in \{+1, -1\}$$

We keep $\alpha_1 \cdots \alpha_{\ell-1}$, $h_1 \cdots h_{\ell-1}$ constant and learn α_ℓ, h_ℓ to min. J

$$J = \sum_{i=1}^{m} \underbrace{\boxed{\exp(-y^{(i)} H_{\ell-1}(x^{(i)}))}}_{w_\ell^{(i)}} \cdot \exp(-y^{(i)} \alpha_\ell h_\ell(x^{(i)}))$$

$$J = \sum_{i=1}^{m} w_\ell^{(i)} \exp(-y^{(i)} \alpha_\ell h_\ell(x^{(i)}))$$

$$J = \sum_{i \text{ correctly classified}} w_\ell^{(i)} \exp(-\alpha_\ell) + \sum_{i \text{ incorrectly classified}} w_\ell^{(i)} \exp(\alpha_\ell)$$

$$|||$$

$$J = (\exp(+\alpha_\ell) - \exp(-\alpha_\ell)) \underbrace{\boxed{\sum_{i=1}^{m} w_\ell^{(i)} \mathbb{I}(h_\ell(x^{(i)}) \neq y^{(i)})}}_{A} + \underbrace{\boxed{\sum_{i=1}^{m} w_\ell^{(i)}}}_{B} \exp(-\alpha_\ell)$$

$$J = (\exp(+\alpha_\ell) - \exp(-\alpha_\ell)) A + \exp(-\alpha_\ell) B \qquad \frac{A}{B} \stackrel{def}{=} \varepsilon_\ell \quad \text{whoa!}$$

now take gradient and set to 0!!

$$\frac{\partial J}{\partial \alpha_\ell} = 0 \implies [\exp(\alpha_\ell) + \exp(-\alpha_\ell)] A - B \exp(-\alpha_\ell) = 0$$

$$[\exp(\alpha_\ell) + \exp(-\alpha_\ell)] \varepsilon_\ell - \exp(-\alpha_\ell) = 0$$

$$\therefore \quad \varepsilon_\ell = \frac{\exp(-\alpha_\ell)}{\exp(\alpha_\ell) + \exp(-\alpha_\ell)} \qquad 1 - \varepsilon_\ell = \frac{\exp(\alpha_\ell)}{\exp(\alpha_\ell) + \exp(-\alpha_\ell)} \qquad \frac{1 - \varepsilon_\ell}{\varepsilon_\ell} = \exp(2\alpha_\ell)$$

<u>Gradient Boosting</u> (apparently Adaboost is a special case of GB)

regression example:

$\mathcal{D} = \{ (x^{(i)}, y^{(i)}) \mid 1 \le i \le m,\ x^{(i)} \in \mathbb{R},\ y^{(i)} \in \mathbb{R} \}$ $\quad f(x)$ is true fxn

goal: find $h(x)$ to min $\sum_{i=1}^{m} L(y^{(i)}, h_1(x^{(i)}))$

$$\hat{h}(x) = h_1(x) + \underbrace{h_2(x)}_{y - h_1(x)}$$

$y - h_1(x)$ the residual!!

→ $\mathcal{D}_2 = \{ (x^{(i)}, y^{(i)} - h_1(x)) \mid 1 \le i \le m;\ x^{(i)} \in \mathbb{R}^d \}$

→ find h_2 to min squared err loss over $\mathcal{D}_2$

→ ☆ focusing on errors made by last classifier

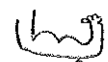
the gradient <u>is</u> the residual!

ex: $J = \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - h_1(x^{(i)}))^2 \qquad \frac{\partial J}{\partial h_1} = -\sum_{i=1}^{m} (y^{(i)} - h_1(x^{(i)}))$

$h_1 \leftarrow h_1 - \frac{\partial J}{\partial h_1}$ tada!

Gradient Boost	vs Adaboost
- generates learners during learning process - 1st learner predicts $y^{(i)}$'s 2nd learner predicts loss .... predict loss until threshold is hit!	- users specify a set of learners at start - it learns weights of how to add learners together to be stronger! - if learner classifies incorrectly → lower it's weight

learners = classifiers



Unsupervised Learning

$\boxed{\text{PCA}}$

- project to lower dimensionality
- max variance in data

find $u_1^T u_1 = 1$ that max var in direction

Short proof:

$$var(y_1) = \frac{1}{m}\sum_{i=1}^{m}(y_1^{(i)} - \bar{y}_1)^2 \qquad \bar{Y}_1 = u_1^T X$$

$$= \frac{1}{m}\sum_{i=1}^{m}(u_1^T x^{(i)} - u_1^T\bar{x})^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(u_1^T(x^{(i)} - \bar{x}))^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(u_1^T(x^{(i)} - \bar{x})(x^{(i)} - \bar{x})^T u_1)$$

$$u_1^T \underbrace{\left(\frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \bar{x})(x^{(i)} - \bar{x})^T\right)}_{cov(x)} u_1$$

$$\underset{u_1}{\text{argmax}} \quad u_1^T S u_1 \quad \text{st. } u_1^T u_1 = 1$$

$$\mathcal{L} = u_1^T S u_1 + \lambda_1(1 - u_1^T u_1)$$

$$\frac{\partial \mathcal{L}}{\partial u_1} = 2Su_1 - 2\lambda_1 u_1 = 0$$

$Su_1 = \lambda_1 u_1$ wow! eigenvectors

K-means

- goal: seperate into K clusters
- min inter-point distances within clusters
- max distance between cluster means
- min distance between point and mean

$$\arg\min J = \sum_{i=1}^{m}\sum_{K=1}^{K} z_K^{(i)} \|x^{(i)} - \mu_K\|^2$$

μ_K cluster mean

z_K latent var

Sweep across z_K and μ_K's

Algorithm:

- choose values for μ
- choose clusters $\longrightarrow$ assign $x^{(i)}$s to cluster k (E-step)
  - min J wrt. z w/ μ fixed
- relocate means $\longrightarrow$ update value of each μ (M-step)
  - min J wrt. μ w/ z fixed
- repeat til stable

E-step: assign points

- calculate J for each val of K for each point x
- select K w/ smallest J

M-step: relocate means

- each $x^{(i)}$ is independent
- compute means of the data points assigned to K

Details

- slow, $O(mK)$
- local min possible, convg gaurenteed
- run multiple times
- initialize cluster means w/ data points (sample)
- best for convex shapes

- cluster means too close? $\rightarrow$ overfitting

Gaussian Mixture Models

- K-means is deterministic $\quad z \to [0|0|0|0|1|0|0|0]$
- GMM is probabalistic $\quad z \to [0.1|0.5|0.3|0.1]$

$$p(x) = \sum_{k=1}^{K} \underbrace{P(z=k)}_{\pi_k} \underbrace{P(x|z=k)}_{\substack{\text{Gaussian} \\ \sim N(x|\mu_k, \sigma_k)}}$$

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \sigma_k) \qquad \sum_{k=1}^{K} \pi_k = 1 \quad \text{constraint}$$

k is # Gaussians $\longrightarrow$ can sweep over

GMMs are generative models! Given $K, \pi_k, \mu_k, \Sigma_k$

$$p(z=k|x) = \frac{p(x|z=k)p(z=k)}{p(x)}$$

$$\arg\max_k P(z=k|x) = \frac{N(x|\mu_k, \Sigma_k)\pi_k}{\sum_{k=1}^{K} P(x|z=k)P(z=k)}$$

$$\mathcal{L}(\mathcal{D}; \pi, \mu, \Sigma) = \prod_{i=1}^{m} P(x^{(i)}; \pi, \mu, \Sigma)$$

$$= \prod_{i=1}^{m} \sum_{k=1}^{K} P(x^{(i)} z^{(i)}; \pi, \mu, \Sigma)$$

$$= \prod_{i=1}^{m} \sum_{k=1}^{K} \underbrace{P(z^{(i)}=k; \pi)}_{\pi_k} \underbrace{P(x^{(i)}|z^{(i)}=k; \mu, \Sigma)}_{N(x^{(i)}; \mu_k, \Sigma_k)}$$

- Use EM! $\boxed{\pi, \mu, \Sigma}$ assume

$\downarrow$ E-step

$P(z^{(i)}=k|x)$

(pick max label)

re-estimate π, μ, Σ

M-step

$$\pi_k = \frac{\sum_{i=1}^{m} \mathbb{I}(z^{(i)}=k)}{m}$$

$$\mu_k = \frac{\sum_{i=1}^{m} \mathbb{I}(z^{(i)}=k)x^{(i)}}{\sum_{i=1}^{m} \mathbb{I}(z^{(i)}=k)}$$

$$\Sigma_k = \frac{\sum_{i=1}^{m} \mathbb{I}(z^{(i)}=k)(x^{(i)}-\mu_k)(x^{(i)}-\mu_k)^T}{\sum_{i=1}^{m} \mathbb{I}(z^{(i)}=k)}$$

GMMs. cont'd

"Hard variant" $\rightarrow$ assign z as one k instead of probability

"Soft variant" $\rightarrow$ complete log-likelihood

$$Q(\theta, \theta^{(t-1)}) = E\left[\ell_c(\mathcal{D}; \theta^{(t-1)})\right] \quad \text{E-step}$$

$\uparrow$ old parameters

$$\theta^{(t)} = \underset{\theta}{\operatorname{argmax}}\ Q(\theta, \theta^{(t-1)}) \quad \text{M-step}$$

1. guess $\theta = [\pi, \mu, \Sigma]$ $k = 1 \cdots K$
2. $r_k^{(i)} = P(z^{(i)} = k \mid x^{(i)}; \theta)$
3. $\theta^{(t)} = \underset{\theta}{\operatorname{argmax}}\ Q(\theta, \theta^{(t-1)})$

for GMMs

$$Q(\theta, \theta^{(t-1)}) = E\left[\sum_{i=1}^{m} \log P(x^{(i)}, z^{(i)} \mid \theta^{(t-1)})\right]$$

$$= E\left[\sum_{i=1}^{m} \log \prod_{k=1}^{K} \left(\pi_k N(x^{(i)} \mid \mu_k, \Sigma_k)\right)^{\mathbb{I}(z^{(i)} = k)}\right]$$

$$Q(\theta, \theta^{(t-1)}) = E\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \mathbb{I}(z^{(i)} = k)\left[\log \pi_k + \log N(x^{(i)} \mid \mu_k, \Sigma_k)\right]\right]$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{K} r_k^{(i)} \log(\pi_k) + \sum_{i=1}^{m}\sum_{k=1}^{K} r_k^{(i)} \log\left(N(x^{(i)} \mid \mu_k, \Sigma_k)\right)$$

$$\frac{\partial Q}{\partial \pi_k} = 0 \qquad \frac{\partial Q}{\partial \mu_k} = 0 \qquad \frac{\partial Q}{\partial \Sigma_k} = 0$$

you know the drill!