L1:

Supervised Learning. Unsupervised Learning. Reinforcement Learning.

SL: · residence has x square feet, predict its price

· More features, we also know the lot size.

(size, lot size) → price

↑ feature/input ↑ label/output

$x \in \mathbb{R}^2$ $y \in \mathbb{R}$

· Regression VS Classification

↑ ↑

$y \in \mathbb{R}$ is a continuous the label is discrete variable.

Unsupervised Learning: Data set contains no labels

⇒ to find interesting structures of data ⇒ Clustering

Reinforcement Learning: . Can collect data interactively.

· Try the strategy and collect feedbacks.
Improve strategy based on the feedbacks.

L2

$$h(x) \equiv \sum_{i=0}^{d} \theta_i x_i = \theta^T x$$

cost function: $J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$

1. LMS algrithm: to minimize $J(\theta)$, start with some initial guess of θ, and repeatedly change to make $J(\theta)$ smaller.

⇓

Gradient Decent: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

α: learning rate.

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 = 2 \times \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y)$$

$$= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{d} \theta_i x_i - y \right)$$

$$= (h_\theta(x) - y) \cdot x_j$$

update rule: $\theta_j = \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$

LMS update rule: least Mean square rule

每个 sample 都会对 θ_j 产生影响. 有个 sample 都要 update 一下 parameter.

- The magnitude of the update is proportional to the error item.

⇓ modify for a training set.

Repeat until convergence: {

$$\theta_j := \theta_j + \alpha \sum_{i=0}^{n} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad \text{(for every } j\text{)}$$

}.

- This method looks at every example in the entire training set on every step. is called batch gradient descent.

- Suceptibe to local minima in general

- Loop {

  for i = 1 to n {

  $$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)}) x_j^{(i)} \quad \text{(for every } j\text{)}$$

  }

}.

⇓

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)}) x_j^{(i)}$$

. update the parameters according to the gradient of the error with respect to that single traning example only. called stochastic gradient descent (incremental gradient descent) Where batch gradient descent has to scan through the entire training set before taking a single step — costly when n is large

· Stochastic Gradient Descent much faster than Batch Gradient Descent But it may never 'converge' to the minimum, it may oscillates around.
↳ can be solved by descreasing α to nearly zero

2. The normal equations — Second way to decrease J. ⇒ to find θ.

2.1 Matrix derivatives.

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1d}} \\ \vdots & & \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{nd}} \end{bmatrix} \quad n \times d$$

eg. $f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}$

$$\Rightarrow \nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

2.2. Least Square Revisited.

$$X = \begin{bmatrix} -(x^1)^T- \\ -(x^2)^T- \\ \vdots \\ -(x^n)^T- \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

$$h_\theta(x^{(i)}) = (x^{(i)})^T\theta \quad \Rightarrow X\theta - \vec{y} = \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(n)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(n)}) - y^{(n)} \end{bmatrix}$$

$$z^T z = \Sigma_i z_i^2$$

$$\Rightarrow \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 = J(\theta)$$

3. Probabilistic interpretation. ⇒ find the maximum likelihood estimate of θ.

4. locally weighted linear regression. (LWR): assuming there is sufficient training data,

underfitting: the data clearly shows the structure not captured by the model.
overfitting:

Original Linear Regression: ① fit θ to minimize $\Sigma_i(y^{(i)} - \theta^T x^{(i)})^2$
② Output $\theta^T x$.

locally weighted linear regression: ① fit θ to minimize $\Sigma_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$.
② Output $\theta^T x$.

$w^{(i)}$ are non-negative valued weights.

If $w^{(i)}$ is large for a particular value of i, then in picking θ, we'll try hard to make $(y^{(i)} - \theta^T x^{(i)})^2$ small. if $w^{(i)}$ is small, the $(y^{(i)} - \theta(x^{(i)}))^2$ is not important.

LWR is non-parametric

Part II: Classification. and logistic regression.

5. Logistic Regression:

- Can use Linear Regression. but parly when $h_\theta(x) < 0$ or $h_\theta(x) > 1$

so we choose

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad \text{where} \quad g(z) = \frac{1}{1+e^{-z}}.$$

↑↑ sigmoid function.

sigmoid function $g(z) = \frac{1}{1+e^{-z}}$: $z \to \infty \Rightarrow g(z) \to 1$, $z \to -\infty \Rightarrow g(z) \to 0$

$$g'(z) = \frac{d}{dz}\frac{1}{1+e^{-z}} = g(z)(1 - g(z))$$

⇓

$$P(y=1 \mid x;\theta) = h_\theta(x) \quad \Rightarrow \quad P(y \mid x;\theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$
$$P(y=0 \mid x;\theta) = 1 - h_\theta(x)$$

⇓ Likelihood of the parameters:

$$L(\theta) = p(\vec{y} \mid X;\theta) = \prod_{i=1}^{n} p(y^i \mid x^i;\theta) = \prod_{i=1}^{n} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

⇓ Gradient Ascent:

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

6. Digression: The perceptron learning algorithm.

- modify g to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0. \end{cases}$$

if we then let $h_\theta(x) = g(\theta^T x)$, and use update rule:

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

Then we have Perceptron Learning Algorithm.

7. Another algorithm to maximize $\ell(\theta)$.

newtown: $\theta := \theta - \frac{f(\theta)}{f'(\theta)}$

⇓

to maximize $\ell(\theta)$. we need $\ell'(\theta) = 0 \Rightarrow \theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)} = \theta - H^{-1} \nabla_\theta \ell(\theta)$

Also called fisher scoring

Part III. Generalized Linear Models.

8. The exponential family:

$$p(y;\eta) = b(y) \exp(\eta^T T(y) - a(\eta))$$

9.3. Softmax Regression.

$y \in \{1, 2, \cdots k\}$.

Parametric learning algorithm: fixed set of parameters to data.

Nonparametric " ": parameters grows with size of data

PART IV Generative Learning Algorithm.

Discriminative Learning : Given x, classify $y \in \{0, 1\}$. $\Rightarrow p(y|x)$

Generative Learning : $p(x|y=0)$: the distribution of dogs features.
$p(x|y=1)$: the distribution of elephants features.

$p(y)$: class priors

$\Rightarrow$ Bayes : $p(y|x) = \frac{p(x|y)\,p(y)}{p(x)}$

I. Gaussian discriminant analysis. (GDA)

Assume $p(x|y)$ is distributed according to a multivariate normal distribution.

II. Multivariate Gaussian Distribution.

mean vector : $\mu \in \mathbb{R}^d$
covariance matrix : $\Sigma \in \mathbb{R}^{d \times d}$
$\} \; \mathcal{N}(\mu, \Sigma)$

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

covariance $\quad cov(XY) = E((X-\mu)(Y-\nu)) = E(X \cdot Y) - \mu\nu$, AB 相互独立 $cov(AB) = \mu\nu$

correlation : $\eta = \frac{cov(XY)}{\sqrt{var(X)\,var(Y)}}$

$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$

$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$

1.2. Gaussian Discriminant Analysis Model (GDA)

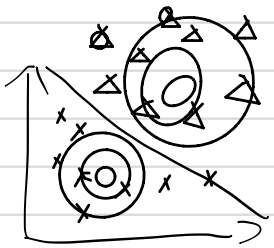
- Classification. x - continuous.
- model $P(x|y)$ using a multivariate normal distribution.

- $y \sim \text{Bernoulli}(\phi) \Rightarrow P(y) = \phi^y (1-\phi)^{1-y}$
  $x|y=0 \sim \mathcal{N}(\mu_0, \Sigma)$
  $x|y=1 \sim \mathcal{N}(\mu_1, \Sigma)$

- Parameter : $\phi, \Sigma, \mu_0, \mu_1$

$$\ell(\phi, \Sigma, \mu_0, \mu_1) = \log \prod_{i=1}^{n} P(x^{(i)}, y^{(i)}; \phi, \Sigma, \mu_0, \mu_1)$$

$$= \log \prod_{i=1}^{n} P(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) P(y^{(i)}; \phi)$$



1.3. GDA vs Logistic Regression.

- GDA assumes $P(x|y)$ is Gaussian Multivariate Distribution.
  it makes stronger modeling assumptions than does logistic regression.

  ⇓

- When data is indeed Gaussian, or dataset is large enough, GDA is better.
- When making significantly weaker assumptions. logistic regression is more robust and less sensitive to incorrect modeling assumptions. eg: $x|y=0 \sim \text{Poisson}(\lambda_0)$

2. Naive Bayes:

- x_j's are discrete-valued.

- Spam email filter.

- Process:
  - features: $x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$ a / aardvark / aardwolf / ⋮ / buy ⋮ ⇒ vocabulary.

实际上只会记录 training set 里的词汇，并除去非常高频词 (stop words) like the, of ... 以减少运算和空间占用.

- build generative model. have to model $P(X|y)$. if size = 50000. then $x \in \{0,1\}^{50000}$ ⇒ $2^{50000}-1$ dimensional parameter vector. ⇒ too many.
- Make very strong assumption.: x_i's are conditionally independent given y.

⇓

Naive Bayes assumption.

⇓

Naive Bayes Classifier.

say. $p(x_{2087}|y) = P(x_{2087}|y, x_{3451})$.

⇓

$P(x_1 \cdots x_{50000}|y) = \prod_{j=1}^{d} P(x_j|y)$

- When the original continuous valued attributes are not well modeled by a multivariate normal distribution, discretizing the features and use Naive Bayes rather than GDA will result better.

2.1. Laplaces smoothing. : make Naive Bayes better, especially for text classification

- It's bad to estimate the probability of some event to be zero just because you haven't seen it before

2.2 Event models for text classification.

Naive Bayes. Bernoulli event model. (Multivariate Bernoulli event model)

↳ probability of choosing i^{th} word in the dictionary = $P(x_j=1|y) = \phi_{j|y}$

↳ probability of a message was given by $p(y)\prod_{j=1}^{d} P(x_j|y)$

Multinomial event model.

↳ x_j the identity of the jth word in the email.

↳ assume spam/non-spam is determined fist, and choose of the word is indepedet.

↳ probability of a message is given by $p(y)\prod_{j=1}^{d} P(x_j|y)$

Part IV: Kernel Methods.

1.1 Feature Maps.

original input value: attributes. $\rightarrow x$

when attributes mapped to some new set of quantities: $\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix}$: feature variables.

$\Downarrow$

ϕ : feature map.

1.2. LMS (Least Mean Square) with features.

$$\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - h_\theta(x^{(i)}))\, x^{(i)}$$

$$:= \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})\, x^{(i)}$$

$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$

$$\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T \phi(x^{(i)})\, \phi(x^{(i)})$$

stochastic gradient descent:

$$\theta := \theta + \alpha (y^{(i)} - \theta^T \phi(x^{(i)})\, \phi(x^{(i)})$$

1.3. LMS with kernel tricks.

Can significatly reduce computing complexity and storage.

kernel tricks: can get the inner product of higher dimension without mapping data to high dimension.

θ can be represented as $\theta = \sum_{i=1}^{n} \beta_i \phi(x^{(i)})$ ← scaler (n variables instead of p)

$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$. $x \rightarrow \phi(x)$ $h_\theta(x) = \theta^T \phi(x)$ using gradient descent.

loop: $\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T \phi(x^{(i)}))\, \phi(x^{(i)})$

ISSUE: $\phi(x)$ is very high in dimension, high time complexity.

$\Downarrow$

Solv: improve to $O(n^2)$ per iteration.

at time t: $\theta = \sum_{i=1}^{n} \beta_i \phi(x^{(i)})$

next iter: $\theta_i := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T \phi(x^{(i)})\, \phi(x^{(i)}) \Rightarrow \theta := \sum_{i=1}^{n} (\beta_i + \alpha(y^{(i)} - \theta^T(x^{(i)}))\, \phi(x^{(i)})$

θ: $\sum_{i=1}^{n} \beta_i \phi(x^{(i)})$

new β_i

$$\beta_i = \beta_i + \alpha(y^{(i)} - \theta^T \phi(x^{(i)}))$$

$$= \beta_i + \alpha(y^{(i)} - (\sum_{j=1}^{n} \beta_j \phi(x^{(j)})^T)\phi(x^{(i)}))$$

$$= \beta_i + \alpha(y^{(i)} - \sum_{j=1}^{n} \beta_j \langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$$

$$\theta^T = (\sum_{j=1}^{n} \beta_j \phi(x^{(j)}))^T$$

$$= \sum_{j=1}^{1} \beta_j \phi(x^{(j)})^T$$

$$a^T b = \langle a, b \rangle$$

Part II Support Vector Machine.

2. Margins.

- separating hyperplane : decision boundary. $\theta^T x = 0$
- find θ so that $\theta^T x^{(i)} >> 0$ when $y^{(i)} = 1$
  $\theta^T x^{(i)} << 0$ when $y^{(i)} = 0$

3. Notation.

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

$h_{w,b}(x) = g(w^T x + b)$

$$\begin{cases} g(z) = 1 & \text{if } z \geq 0 \\ g(z) = -1 & \text{if } z < 0 \end{cases}$$

$$\begin{cases} b = \theta_0 \\ w = [\theta_1 \cdots \theta_d]^T \end{cases}$$

4. Functional and geometric margins.

- functional margin : $\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$

$y^{(i)} = 1$. $w^T x^{(i)} + b$ large positive
$y^{(i)} = -1$, $w^T x^{(i)} + b$ large negative
when $\hat{\gamma}^{(i)} < 0$, goes wrong.

⇓

large functional margins represents a confident and correct prediction.

- geometric margin :

$$\gamma^{(i)} = y^{(i)}\left(\left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}\right)$$

func m = geo m when $\|w\| = 1$

7. Optimal margin classifiers.

- $\min_{w,b} \frac{1}{2}\|w\|^2$

  s.t. $y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1 \cdots n.$

- support vectors.

- $\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]$

Deep Learning.

1. Supervised Learning with non-Linear Models.

- Predict y from x, hypothesis is $h_\theta(x)$. $h_\theta(x) = \theta^T x$, $h_\theta(x) = \theta^T \phi(x)$.
  parameters are θ, linear

- Next, non-linear in both parameters and inputs x. ⇒ neural networks

- Suppose $y \in \mathbb{R}$, $x \in \mathbb{R}$. least square cost function: $j^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$

  mean square cost function: $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta)$

- Optimizers (SGD)
  - GD: $\theta := \theta - \alpha \nabla_\theta J(\theta)$
  - SGD: learning rate α, iterations n_{iter}.
    Initialize θ randomly.
    for $i = 1$ to n_{iter} do:
    sample j uniformly from $\{1, 2 \cdots n\}$ and update θ by
    $\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta)$
  - Mini-batch SGD: often times computing the gradient of B examples simultaneously.
    can be faster due to hardware parallelization.
    α, n_{iter}, batch size B.
    for $i = 1$ to n_{iter} do:
    Sample B examples $j_1 \cdots j_B$ (without replacement) uniformly from $\{1 \sim n\}$
    and update θ by:
    $\theta := \theta - \frac{\alpha}{B}\sum_{k=1}^{B} \nabla_\theta J^{(j_k)}(\theta)$

① Define a neural network parameterization $h_\theta(x)$.
② Write backpropagation algorithm to compute the gradient of loss function $J^{(i)}(\theta)$ efficiently.
③ Run SGD or mini-batch SGD. with the loss function.

2. Neural Networks.

– refer to broad type of non-linear models that involves of combinations of matrix multiplications and other entrywise non-linear operations.

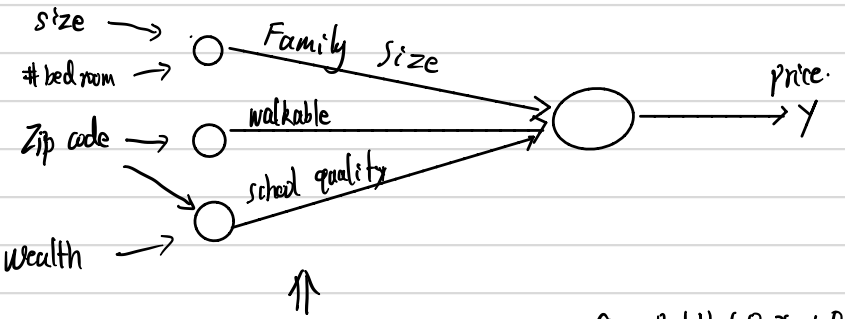
2.1. Nural network with single neuron:

$$h_\theta(x) = \max(wx+b, 0) \quad \text{where} \quad \theta = (w, b) \in \mathbb{R}^2.$$

* $\text{ReLU} \triangleq \max\{t, 0\}$.
* Activation function: a one-dimensional non-linear function that maps $\mathbb{R}$ to $\mathbb{R}$.
  $h_\theta(x)$ is said to have a single neuron because it has a single non-linear function.
* when $x \in \mathbb{R}^d$, a neural network with single neuron
  $h_\theta(x) = \text{ReLU}(w^T x + b)$, where $w \in \mathbb{R}^d$, $b \in \mathbb{R}$, $\theta = (w, b)$
  b: bias
  w: weight vector.

2.2. Stacking neurons.

One neuron passes its output as input into the next neuron.



input features.
x_1, x_2, x_3
x_4

hidden layer
hidden neuron
a_1, a_2, a_3

$$a_1 = \text{ReLU}(\theta_1 x_1 + \theta_2 x_2 + \theta_3)$$
$$a_2 = \text{ReLU}(\theta_4 x_3 + \theta_5)$$
$$a_3 = \text{ReLU}(\theta_6 x_3 + \theta_7 x_4 + \theta_8)$$

$$h_\theta(x) = \theta_9 a_1 + \theta_{10} a_2 + \theta_{11} a_3$$

2.4.

Two-layer Fully-Connected Neural Network.

- Fully-connected neural network: all the intermediate variables a_i's depend on all the inputs x_i's.

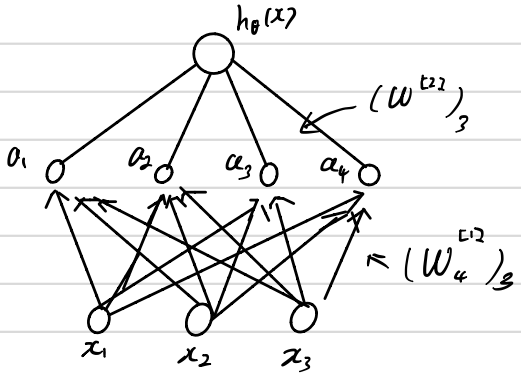
$a_1 = ReLU(w_1^T x + b)$, $w_1 \in \mathbb{R}^d$, $b_1 \in \mathbb{R}$.

$a_2 = ReLU(w_2^T x + b)$ - -

$a_3 = ReLU(w_3^T x + b)$ - -

- a two-layer fully-connected neural network with m hidden units and d dimensional input $x \in \mathbb{R}^d$

$\forall j \in [1, \cdots m]$, $z_j = w_j^{[1]T} x + b_j^{[1]}$ where $w_j^{[1]} \in \mathbb{R}^d$, $b_j^{[1]} \in \mathbb{R}$.



edge from x_i to a_j is associated with the weight $(w_j^{[1]})_i$, which denotes the i^{th} coordinate of vector $w_j^{[1]}$

$$a_j = ReLU\left(\sum_{i=1}^{d} (w_j^{[1]})_i \cdot x_i\right) \Rightarrow a_j = ReLU(z_i)$$

$h_\theta(x) = w^{[2]T} a + b$, $w^{[2]} \in \mathbb{R}^m$, $b^{[2]} \in \mathbb{R}$.

2.5. Vectorization.

reason: for loop is too slow, we need to parallelize the computation using GPU

$$W^{[1]} = \begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[1]T} - \\ \vdots \\ - w_m^{[1]T} - \end{bmatrix} \in \mathbb{R}^{m \times d} \Rightarrow$$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[1]T} - \\ \vdots \\ - w_m^{[1]T} - \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \Rightarrow z = W^{[1]} x + b^{[1]}$$

$\mathbb{R}^{m \times 1}$ $\mathbb{R}^{m \times d}$ $\mathbb{R}^{d \times 1}$ $\mathbb{R}^{m \times 1}$

2.6. Multi-layer full-connected neural networks.

r : number of layers. $W^{[1]} \cdots W^{[r]}$ weight matrices. $b^{[1]} \cdots b^{[r]}$ biases.

$$\Rightarrow a^{[1]} = ReLU(W^{[1]}x + b^{[1]})$$

$$\vdots$$

$$a^{[r-1]} = ReLU[W^{[r-1]}a^{[r-2]} + b^{[r-1]})$$

$$h_\theta(x) = W^{[r]}a^{[r-1]} + b^{[r]}$$

$$a^{[k]} \in \mathbb{R}^{m_k} \Rightarrow \begin{matrix} W^{[k]} \in \mathbb{R}^{m_k \times m_{k-1}} \\ b^{[k]} \in \mathbb{R}^{m_k} \end{matrix} \Rightarrow \begin{matrix} W^{[1]} \in \mathbb{R}^{m_1 \times d} \\ W^{[r]} \in \mathbb{R}^{1 \times m_{r-1}} \end{matrix}$$

$$a_0 = x$$

$$a^{[r]} = h_\theta(x)$$

$$a^{[k]} = ReLU(W^{[k]}a^{[k-1]} + b^{[k]}), \quad \forall k = 1, 2 \cdots r-1$$

2.7 other activation functions:

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{(sigmoid)}$$

$$\sigma(z) = \frac{e^z + e^{-z}}{e^z - e^{-z}} \quad \text{(tanh)}$$

2.8. Why not use identity function for $\sigma(x)$? ie. $\sigma(z) = z$?

that's: $h_\theta(x) = W^{[2]}a^{[1]}$ $\qquad a^{[1]} = \sigma(z^{[1]}) = z^{[1]}$

$$= W^{[2]}z^{[1]}$$

$$= W^{[2]}W^{[1]}x$$

$$= \tilde{W}x.$$

This will become simple linear regression which losses much of the representational power of the neural network as often times we are trying to predict has a non-linear relationship with the inputs.

2.9. Connect to the kernel method.

- The process of choosing the feature maps is Feature Engineering.
- We can view deep learning as a way to automatically learn the right feature map

β: collection of parameters in a fully connected neural network except last layer.
$a^{[r-1]}$: function of the input x.

$\Rightarrow \beta : a^{[r-1]} = \phi_\beta(x)$

$\Rightarrow$ the model : $h_\theta(x) = W^{[r]} \phi_\beta(x) + b^{[r]}$

β is fixed, $\phi_\beta(\cdot)$ can be viewed as a feature map.

: $h_\theta(x)$ is just a linear model over the features $\phi_\beta(x)$
- We will train the neural networks, both the parameters β and $W^{[r]}, b^{[r]}$.
- We are not only learning a linear model in the feature space, but also learning a good feature map $\phi_\beta(\cdot)$

⇓

- Therefore deep learning tends to depend less on the domain knowledge and requires less feature engineering.

3. Backpropagation.

- Computes the gradient of the loss $\nabla J^{(i)}(\theta)$ efficiently

Evaluation Metrics

1. Binary Classification.

- Two types of models:
  - output categorical class directly (kNN, decision tree)
  - output real valued score (SVM, Logistic Regression)
    - Score would be margin (SVM), probability (LR, NN)
    - Need to pick a threshold.

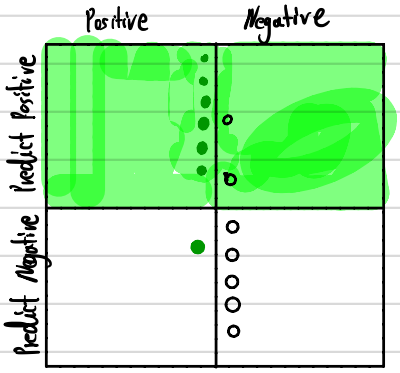
- Threshold → Classifier → Point Metrics.

1.1. Confusion Matrix

TP	FP
FN	TN

→ · Want diagonal to be heavy, off diagonals to be light.

FP : Type-1 error
FN : Type-2 error



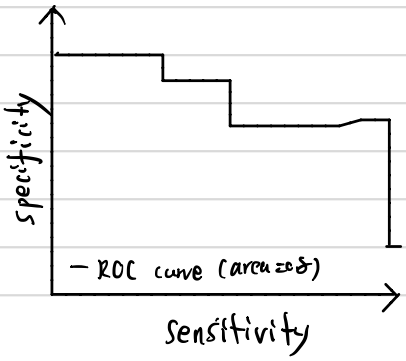
- Accuracy $= \frac{TP+TN}{total}$ Precision: $\frac{TP}{TP+FP}$

- Recall (Sensitivity) $= \frac{TP}{TP+FN}$
  - Trivial 100% recall = pull every one above the threshold.
  - Trivial 100% precision = push everyone below the threshold except 1 green on top.

- F1-Score $= \left(\frac{2}{recall^{-1}+precision^{-1}}\right) = 2 \cdot \frac{Precision \cdot recall}{precision + recall}$

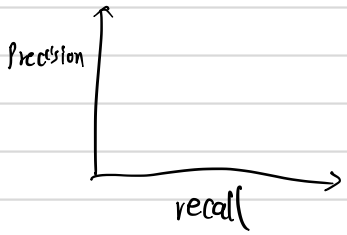
- Negative recall (specificity) $= \frac{TN}{TN+FP}$.

1.2 Summary metrics: Rotated ROC (Sen vs Specificity)



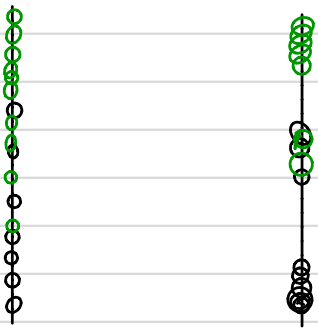
AUROC = Area Under ROC
= Prob [Random Pos ranked higher than random Neg]

- PRC (recall vs Precision)



AUPRC = Expected precision for random threshold

- Log-Loss vs Brier Score



model A model B which is better

- Same ranking, and therefore same AUROC, AUPRC
- Log Loss $= \frac{1}{N}\sum_{i=1}^{N} -y_i \log \hat{y}_i - (1-y_i) \log (1-\hat{y}_i)$
  - Rewards confident correct answers, heavily penalize confident wrong answers.
  - One perfectly confident wrong prediction is fatal
  -

2. Class imbalance
   - sympton : Prevalence < 5%

Prevalence $= \frac{P}{P+N}$

3. Choosing Metrics:
   - High precision is hard constraint; do best at recall
     - metric : recall at Precision = xx %
   - High recall ... ; do best at precision
     - metric : Precision at Recall = xx %