Machine Learning Notes

Probability:

1. marginal probability = ∫ (probability A , probability B) * d(probability B) = ∫ (probability A | probability B) * probability B * d(B)
2. ∑ P(Ai | B) = 1
3. p(Y | X) = ∫ p(Y | Z , X) p(Z | X) dZ
4. posterior probability = prior probability * marginal of the prior probability / marginal of the posterior probability
5. P(AB|C) P(C) = P(A|BC) P(BC)
   thus, **P(AB|C)** = P(A|BC) P(BC) / P(C) = P(A|BC) P(B|C)P(C) / P(C) = **P(A|BC) P(B|C)**
   https://www.youtube.com/watch?v=jY2E6ExLxaw
7. Taking the expectation over D, based on the expectation linearity,
   E[y(x;D) - E[y(x;D)]] = E[y(x;D)] - E[E[y(x;D)]] = E[y(x)] - E[E[y(x)] * 1] = E[y(x)] - E[y(x)] * E[1] = 0
8. Probability model is always working on random variables
9. For random variables X and Y, X is independent of Y if and only if:
   - P(X, Y) = P(X) * P(Y)
   - P(X | Y) = P(X)
   - P(Y | X) = P(Y)
10. For random variables X, Y and Z, X is conditionally independent of Y given Z if and only if:
    - P(X, Y | Z) = P(X | Z) * P(Y | Z)
    - P(X | Y , Z) = P(X | Z)
    - P(Y | X , Z) = P(Y | Z)
    - P(X, Y, Z) ∝ Φ1(X, Z) * Φ2(Y, Z)
11. Probability Mass Function (PMF for discrete random variable) =

Probability Density Function (PDF for concrete random variable) = Probability Distribution

12. If X1, X2, ..., Xn are independent Bernoulli(p) random variables, then the random variable X defined by X = X1 + X2 + X3 + ... + Xn has a Binomial(n, p) distribution

The law of large numbers:

Let X1, X2, X3 ... be a set of independent and identically distributed random samples. It can be shown that:

$$\lim_{n \to \infty} P(| Sn / n - E(X) | > \varepsilon) = 0$$

where Sn = X1 + X2 + X3 + ... + Xn

Central Limit Theorem:

Classical Definition:

Taking the mean of **n** samples that are i.i.d. sampled from the source distribution as a random variable **Sn** as the expectation of it according to **Law of Large Number**,

$$S_n := \frac{X_1 + \cdots + X_n}{n}$$

As **n** gets larger, the random variable

$$\sqrt{n}(S_n - \mu)$$

approximates the normal distribution with mean 0 and variance σ^2.

For large enough *n*, the distribution of S_n is close to the normal distribution with mean μ and variance σ^2 / n.

Lindeberg–Lévy Definition:

Let X1, X2, ..., Xn be i.i.d random variables with expected value E[Xi] = μ < ∞ and variance 0 < Var(Xi) = σ^2 < ∞. Then, the random variable

$$\sqrt{n}\left(\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) - \mu\right) \xrightarrow{d} N\left(0, \sigma^2\right).$$

converges in distribution to the standard normal random variable as n goes to infinity, that is

$$\lim_{n\to\infty} \Pr\left[\sqrt{n}(S_n - \mu) \leq z\right] = \Phi\left(\frac{z}{\sigma}\right),$$

where Φ(x) is the standard normal distribution.

As i goes infinity,

$$\text{Var}(\mu) = \text{Var}\left(\left(1/n\right)\sum_i X_i\right) = \left(1/n^2\right)\sum_i \text{Var}(X_i) = \sigma^2/n$$

The variance of the mean decreases when i increases.

Why is central limit theorem important ?

Many statistics have distributions that are approximately normal for large sample sizes, even when we are sampling from a distribution that is not normal.

https://www.khanacademy.org/math/ap-statistics/sampling-distribution-ap/sampling-distribution-mean/v/sampling-distribution-of-the-sample-mean
https://www.youtube.com/watch?v=Pujol1yC1_A

Law of the unconscious statistician (LOTUS) for continuous random variables:

if g(X) is a function of random variable X, then g(X) is also a random variable, thus,

E[g(X)] = ∫ g(X) f(X) dX,

where f is the probability density function of X

written in another way, let **Y = g(X)**,

$$\mathbb{E}_{Y\sim P_Y}[Y] = \mathbb{E}_{X\sim P_X}[g(X)]$$

Expectation:

1. Mean vs. Sample Mean

Let X1, X2, ... Xn be n i.i.d. random variables drawn from a distribution with mean μ and variance σ^2, then

E[Xi] = μ

While we define the random variable Sn is

Sn = (X1 + X2 + X3 + ... + Xn) / n = (1 / n) ∑ Xi

So the sample mean is

$$E[Sn] = E[(1 / n) \sum Xi] = (1 / n) \sum E[Xi] = (1 / n) * n * \mu = \mu$$

2. Let X and Y be random variables with finite expectations, g(X) and h(X) are also random variables, then

If g(X) >= h(X) for all real number x, then E[g(X)] >= E[h(X)]

E[aX + bY + c] = aE[X] + bE[Y] + c for any real number a, b, c

E[X1 + X2 + ... + Xn] = E[X1] + E[X2] + ... + E[Xn], for any set of random variables X1, X2, ... , Xn

3. Let X and Y be random variables with finite expectations, given X is known (which implies g(X) is known as well), taken out what is known,

$$E[g(X) h(Y) | X] = g(X) E[h(Y) | X]$$

4. The law of iterated (total) expectation,

$$E[Y] = \sum P(X = xi) E[Y | X = xi],$$

seeing X' = E[Y | X = xi], with LOTUS, we have

$$E[X'] = E[E[Y | X]] = \sum P(X = xi) E[Y | X = xi] = E[Y],$$

thus,

$$E[Y] = E[E[Y | X]]$$

for more details,
https://imai.princeton.edu/teaching/files/Expectation.pdf
https://www.probabilitycourse.com

Variance / Covariance:

1. variance definition
variance is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value.

$$
\begin{aligned}
\mathrm{Var}(X) &= \sigma^2 \\
&= E[(X - E[X])^2] \\
&= E[X^2 - 2 \cdot X \cdot E[X] + E[X]^2] \\
&= E[X^2] - 2 \cdot E[X] \cdot E[X] + E[X]^2 \\
&= E[X^2] - E[X]^2 \\
&= \int X^2 f(X)\, dX - \mu^2 \\
&= \mathrm{Cov}(X, X)
\end{aligned}
$$

$$\mathrm{Var}(aX) = a^2 \mathrm{Var}(X)$$

like entropy, variance could sort of present the uncertainty (information) of a distribution.
https://www.quora.com/Which-is-the-best-measure-of-uncertainty-variance-or-entropy-or-are-they-both-equivalent
the entropy of a Gaussian distribution is proportional to the determinant of it.

2. Variance vs. Sample Variance

Let X1, X2, ... Xn be n i.i.d. random variables drawn from a distribution with mean μ and variance σ^2, then

$$\mathrm{Var}[X_i] = \sigma^2$$

While we define the random variable Sn is

$$S_n = (X_1 + X_2 + X_3 + \ldots + X_n) / n = (1/n) \sum X_i$$

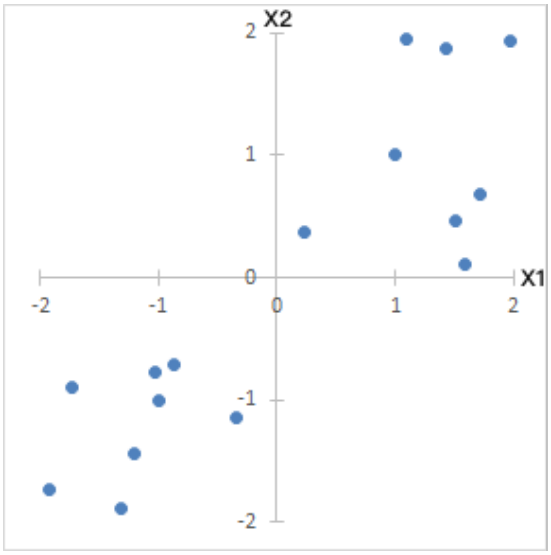
So the sample variance is

$$\mathbf{Var[\,Sn\,] = Var[\,(1/n)\sum Xi\,] = (1/n^2)\sum Var[\,Xi\,] = (1/n^2) * n * \sigma^2 = \sigma^2/n}$$

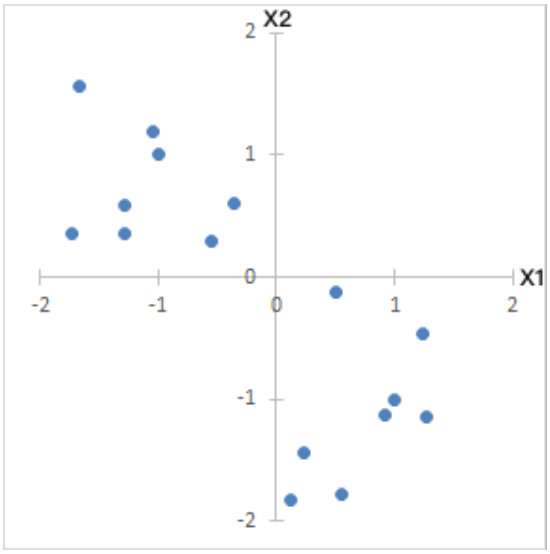
3. covariance definition

$$\mathbf{Cor(\,X, Y\,) = E[\,(x - \mu_x)(y - \mu_y)\,] = E[\,XY\,] - E[\,X\,]\,E[\,Y\,]}$$

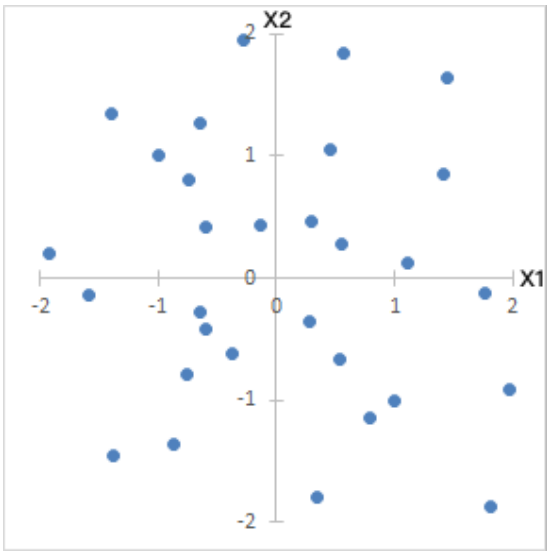
the covariance between X and Y indicates how the values of X and Y are more relative to each other.
a non-diagonal correlation matrix implies that the eigenvectors of the data distribution ***are not aligned with the axes.***
For example, Consider a 15 x 2 design matrix, which means 15 pairs of (x1, x2) data, with E[x] = 0,



$$Cov \approx \begin{bmatrix} \sum x_1^2 & \sum x_1 x_2 \\ \sum x_1 x_2 & \sum x_2^2 \end{bmatrix} \; where \sum x_1 x_2 > 0$$



$$Cov \approx \begin{bmatrix} \sum x_1^2 & \sum x_1 x_2 \\ \sum x_1 x_2 & \sum x_2^2 \end{bmatrix} \; where \sum x_1 x_2 < 0$$



$$Cov \approx \begin{bmatrix} \sum x_1^2 & \sum x_1 x_2 \\ \sum x_1 x_2 & \sum x_2^2 \end{bmatrix} \; where \sum x_1 x_2 = 0$$

4. correlation coefficient

ρ(X, Y) = Cov((X - EX) / σX, (Y - EY) / σY) = Cov(X, Y) / σXσY, -1 ≤ ρ(X, Y) ρ(X, Y) ≤ 1

5. variance decomposition(law of total variance)

Var(*X*)= *E*[Var(*X* | *Y*)] + Var(*E*[*X* | *Y*]).

proof: Let V = Var(X | Y), Z = E[X | Y]

$$V = E[X^2 | Y] - (E[X | Y])^2 = E[X^2 | Y] - Z^2$$

thus,

$$\begin{aligned} E[V] &= E[E[X^2 | Y] - Z^2] \\ &= E[E[X^2 | Y]] - E[Z^2] \quad \text{\# expectation linearity} \\ &= E[X^2] - E[Z^2] \quad \text{\# law of iterated expectation} \end{aligned}$$

next, we have

$$\begin{aligned} Var(Z) &= E[Z^2] - (E[Z])^2 \\ &= E[Z^2] - (E[E[X | Y]])^2 \\ &= E[Z^2] - (E[X])^2 \text{ \# law of iterated expectation} \end{aligned}$$

Combining 2 equations above, we have

$$E[V] + Var(Z) = E[X^2] - (E[X])^2 = Var(X)$$

that is,

$$Var(X) = E[Var(X | Y)] + Var(E[X | Y])$$

the variance decomposition implies that

$$Var(X) >= E[Var(X | Y)]$$

this states that when we condition on Y, the uncertainty (variance) of X reduces on average.

6. product of independent variables

$$\begin{aligned} Var(XY) &= E[X^2 Y^2] - E[XY]^2 \\ &= E[X^2] E[Y^2] - E[X]^2 E[Y]^2 \\ &= E[X]^2 E[Y^2] + E[Y]^2 E[X^2] - 2E[X]^2 E[Y]^2 + \\ &E[X^2]E[Y^2] - E[X^2]E[Y]^2 - E[X]^2 E[Y^2] + E[X]^2 E[Y]^2 \\ &= E[X]^2 (E[Y^2] - E[Y]^2) + E[Y]^2 (E[X^2] - E[X]^2) \end{aligned}$$

$$+ (E[X^2] - E[X]^2) (E[Y^2] - E[Y]^2)$$
$$= E[X]^2 Var(Y) + E[Y]^2 Var(X) + Var(X) Var(Y)$$

7. if X1, X2, ..., Xn are pairwise uncorrelated, i.e., $\rho(Xi, Xj) = 0$ when $i \neq j$, then

$$Var(X1 + X2 + ... + Xn) = Var(X1) + Var(X2) + ... + Var(Xn)$$

In case of 2 random variables X1 and X2,

$$Var(X1 + X2) = E[(X1 + X2)^2] - E[(X1 + X2)]^2$$
$$= E[X1^2 + 2X1X2 + X2^2] - (E[X1] + E[X2])^2$$
$$= E[X1^2] + 2E[X1X2] + E[X2^2] - E[X1]^2 - 2E[X1]E[X2] + E[X2]^2$$
$$= (E[X1^2] - E[X1]^2) + (E[X1^2] - E[X1]^2)$$
$$= Var(X1) + Var(X2)$$

8. variance of weighted sum random variables

$$Var(c^T X)$$
$$= Var(\sum_i c_i X_i)$$
$$= E[(\sum_i c_i X_i)^2] - E[(\sum_i c_i X_i)]^2$$

note that

$$(\sum_i c_i X_i)^2$$
$$= (c_1 X_1 + c_2 X_2 + ... + c_n X_n) \bullet (c_1 X_1 + c_2 X_2 + ... + c_n X_n)$$
$$= \sum_i \sum_j c_i c_i \bullet X_i X_j$$

thus

$$E[(\sum_i c_i X_i)^2]$$

$$= E[\sum_i \sum_j c_i c_i \bullet X_i X_j]$$
$$= \sum_i \sum_j c_i c_i \bullet E[X_i X_j]$$

similarly

$$E[(\sum_i c_i X_i)]^2$$
$$= [\sum_i c_i \bullet E[X_i]]^2$$
$$= \sum_i \sum_j c_i c_i \bullet E[X_i] E[X_j]$$

so

$$Var(c^T X_i)$$
$$= \sum_i \sum_j c_i c_i \bullet E[X_i X_j] - \sum_i \sum_j c_i c_i \bullet E[X_i] E[X_j]$$
$$= \sum_i \sum_j c_i c_i \bullet (E[X_i X_j] - E[X_i] E[X_j])$$
$$= \sum_i \sum_j c_i c_i \bullet Cov(X_i , X_j)$$
$$= c^T \sum c$$

where ∑ is the covariance matrix.

Furthermore, the *first principal component* is the linear combination of x-variables that has maximum variance (among all linear combinations). It accounts for as much variation in the data as possible.

https://stats.stackexchange.com/questions/31177/does-the-variance-of-a-sum-equal-the-sum-of-the-variances?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

9. variance **σ^2 = (∑ (x - μ) ^ 2) / N**, captures the spread among the outcomes

10. From the point view of vector, variance σ^2 is the dot product of the data over the number of the data, which means the variance is just the unique

length of the whole data distribution. $\sigma^2 = (1/n)\, XX^T$

11. Similarly, the covariance is defined as $\sigma^2_{XY} = (1/n)\, XY^T$
If X and Y are correlated, the dot product of XY^T would be non-zero. In contrast, it would be 0, or orthogonal otherwise.

for more details,
https://imai.princeton.edu/teaching/files/Expectation.pdf
https://www.probabilitycourse.com

Convolution of joint probability distribution:

If X and Y are two jointly continuous random variables and Z = X + Y, then

$$f_Z(Z) = \int f_{XY}(w, z-w)\,dw = \int f_{XY}(z-w, w)\,dw$$

If X and Y are also independent, then

$$\begin{aligned} f_Z(Z) &= f_X(Z) * f_Y(Z) \\ &= \int f_X(w) * f_Y(z-w) * dw \\ &= \int f_Y(w) * f_X(z-w) * dw \end{aligned}$$

for the X and Y be two independent standard normal random variables,
if $X \sim N(\mu_x, \sigma_x^2)$ and $Y \sim N(\mu_y, \sigma_y^2)$ are said to be jointly normal based on the central limit therom, if

$$X + Y \sim N(\mu_x + \mu_y,\ \sigma_x^2 + \sigma_y^2 + 2\rho(X, Y)\sigma_x\sigma_y)$$

Moment Generating Function:

Mx(s) = E[e^(sX)]
= E[1 + sX + (sX)^2 / 2! + (sX)^3 / 3! + (sX)^4 / 4! + ... + (sX)^k / k!] # Taylor Series
= E[∑ (X^k)(s^k) / k!]
= ∑ E[X^k] (s^k) / k! # expectation linearity

where **E[X^k]** is the **k-th moment** of random variable **X**.

MGF of X gives all moments of X which uniquely determines the distribution.
If two random variables have the same MGF, then they must have the same distribution.
Thus, if you find the MGF of a random variable, you have indeed determined its distribution.

Markov Inequality:

P(X ≥ α) ≤ E[X] / α

Chebyshev's Inequality:

$P(X - \mu \geq \alpha) \leq Var(X) / \alpha^2$

Proof: Based on Markov Inequality **P(X ≥ α) ≤ E[X] / α**,

$P(X - \mu \geq \alpha) = P((X - \mu)^2 \geq \alpha^2) \leq E[(X - E[X])^2] / \alpha^2 \leq Var(X) / \alpha^2$

Chernoff bounds:

Chernoff bounds use of moment generating functions in an essential way to

give exponential deviation bounds.

$$P(X - \mu \geq a) \leq E[e^{\lambda[X - \mu]}]e^{-\lambda a}$$

where $E[e^{\lambda[X - \mu]}]$ is the moment generating function of random variable $X - \mu$.

Proof:

$$P(X - \mu \geq a) = P(e^{\lambda[X - \mu]} \geq e^{-\lambda a}) \leq E[e^{\lambda[X - \mu]}]e^{-\lambda a}$$ **# Markov Inequality**

Hoeffding Lemma:

Let X be a bounded random variable with $X \in [a , b]$, then

$$E[e^{\lambda[X - \mu]}] \leq \exp(\lambda^2 (b - a)^2 / 8)$$ for all $\lambda \in R$

Hoeffding inequality:

Let X1 , ... , Xn be independent bounded random variables with $Xi \in [a , b]$ for all i, where $-\infty < a \leq b < \infty$, then

$$P((1 / n) \sum (Xi - \mu i) \geq t) \leq \exp(- 2nt^2 / (b - a)^2)$$

for all $t \geq 0$.

Proof:

$$P((1 / n) \sum (Xi - \mu i) \geq t) = P(\sum (Xi - \mu i) \geq nt)$$

$$\leq E[e^{\lambda \Sigma[X^i - \mu^i]}] e^{-\lambda nt}$$ **# Chernoff bounds**

$$\leq \exp(- 2nt^2 / (b - a)^2)$$ **# Hoeffding Lemma**

http://cs229.stanford.edu/extra-notes/hoeffding.pdf

Temperature definition in thermodynamics theory:

Temperature is the change of the expected energy in a thermodynamic equilibrium system, which is the sum of probability of each microstate times the energy required by the state, that increasing 1 bit of entropy to the system, which is the negative sum of the probability of each microstate times the log of it.

$$T = dE / dH$$

1st law of thermodynamics:
energy remains the same among systems

2nd law of thermodynamics:
the entropy of the whole system increases.

we introduced 2 energies here: E1 and E2:
first thing first, from the 1st law of thermodynamics:

$$dE1 + dE2 = 0$$

by the definition of temperature,

$$dE1 = T1 * dH1,\ \ dE2 = T2 * dH2$$

and since the entropy of the 2 systems each of which owns its own energy increases, we have

$$dH1 + dH2 > 0$$

thus,

$$dH1 - (T1 / T2) * dH1 > 0$$

which means, **If system didn't reach to the thermodynamic equilibrium, which means the temperature is not equal, entropy increases**. the

thermodynamic equilibrium is rarely happened.

Euler Number e:

1. e = lim (1 + 1/n)^n = ∫ (1 / n!) dn,
it could be explained as, in an other way, the Taylor Series of function e^x at the point x = 1
https://www.youtube.com/watch?v=3d6DsjIBzJ4

2. e^x = lim (1 + x/n)^n = ∫ (x / n!) dn

3. e is always being used in the equations as to represent the factorial facets like permutations etc.

Definition of ln:

$$\ln(x) = \int 1/x \, dx$$

Computing Log-Sum-Exp

Imagine that we have a set of N values and we want to compute the quantity

$$z = \log \sum_i \exp \{ x_i \}$$

For extremely large or extremely negative $\mathbf{x}_i$, $\mathbf{z}$ would be resulted in underflow or overflow. With some transformations we have,

$$\exp \{ z \} = \sum_i \exp \{ x_i \}$$
$$\exp \{ -a \} \bullet \exp \{ z \} = \exp \{ -a \} \bullet \sum_i \exp \{ x_i \}$$
$$\exp \{ z - a \} = \sum_i \exp \{ x_i - a \}$$

$$z - a = \log \sum_i \exp \{ x_i - a \}$$

$$z = a + \log \sum_i \exp \{ x_i - a \}$$

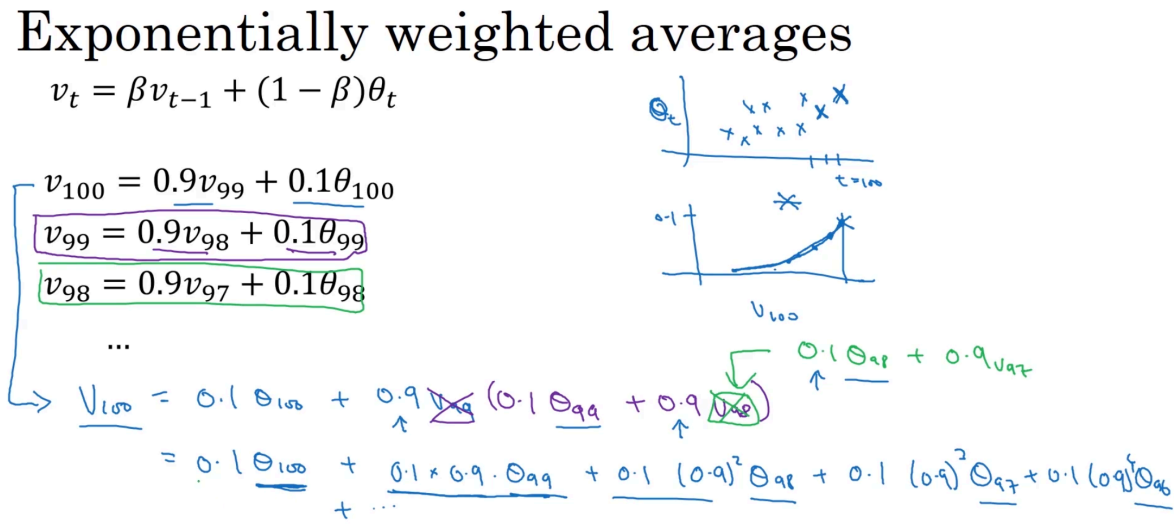
A typical choice of **a** is

$$a = \text{argmax}_i \, x_i$$

which ensures that the largest value you exponentiate will be zero.

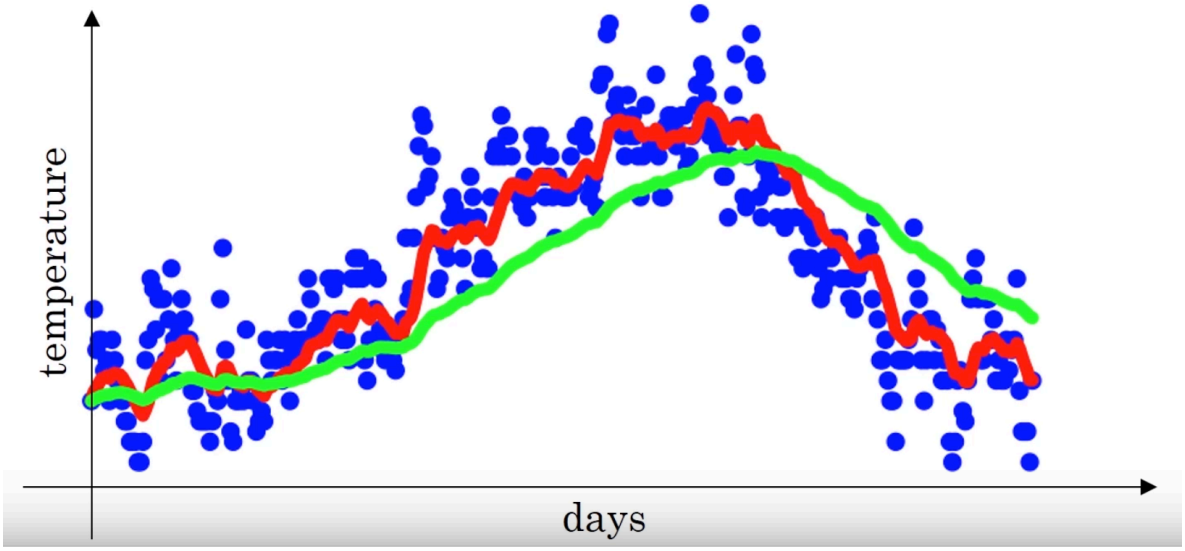
z can be roughly approximated by **a**, that is $\text{argmax}_i \, x_i$. This approximation is based on the idea that $\exp \{ x_k \}$ is insignificant for any x_k that is noticeably less than $\text{argmax}_i \, x_i$.

Exponentially Weighted Average



which is exponentially average **1 / (1 - ß)** historical data to compute the current data.

The larger the ß, the smoother the weighed average. As shown below, ß = 0.9 generates the red line, while ß = 0.98 generates the green line.



https://www.youtube.com/watch?v=NxTFIzBjS-4

Gaussian distribution:

1. $N(\mu, \sigma^2) = \mu + N(0, \sigma^2)$

2. for the linear transformation $\mathbf{y = w^T x}$ with some gaussian noise $\epsilon \sim N(0, \sigma^2)$

$$y = w^T x + \epsilon$$

recall that for a normally distributed random variable with mean **m** and variance **h**: $t \sim N(m, h)$, then a linear transformation of **t** given by $at + b$ is also a normally distributed random variable with mean $a * m + b$, and variance $h * a^2$.

thus, y is the linear transformation of random variable ϵ with mean $w^T x$, and variance σ^2, which is

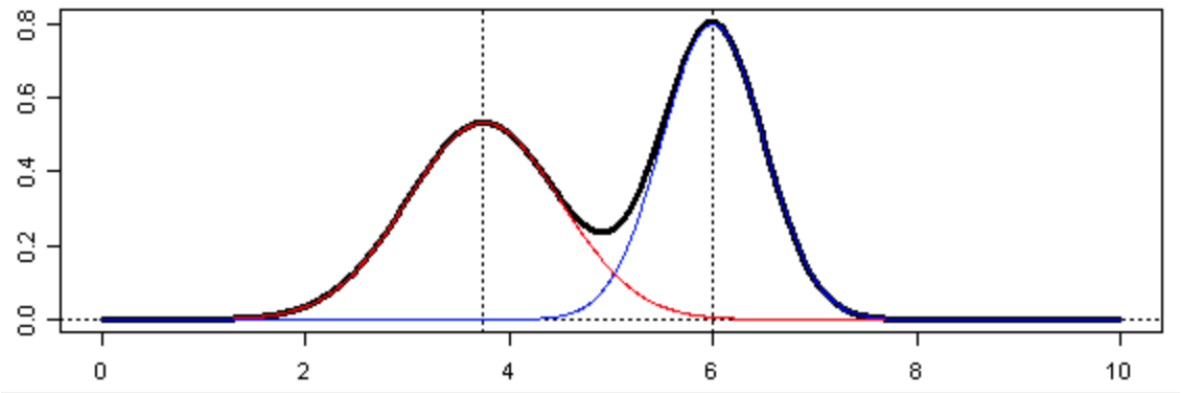
$$y \mid x \sim N(w^T x, \sigma^2)$$

Furthermore, if given that $\mathbf{x \sim N(0 , I)}$, thus

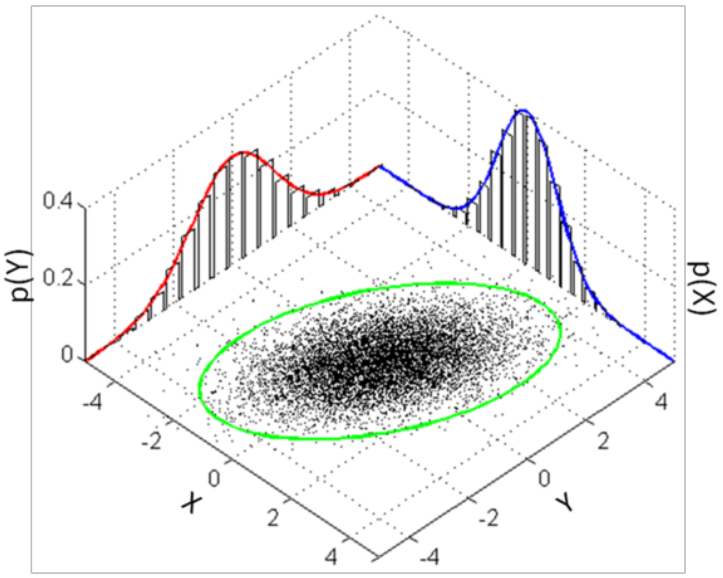
$$\mathbf{Var(y \mid x)}$$
$$\mathbf{= Var(w^T x + \epsilon)}$$
$$\mathbf{= w^T \bullet Var(x) \bullet w + Var(\epsilon)}$$
$$\mathbf{= w^T w + \sigma^2}$$

3. Gaussian Mixture Model vs. Multivariate Gaussian Model

$$\mathbf{GMM = \sum_i N(x ; \Phi_i)}$$



$$\mathbf{MGM = N(X_{\mid : N} ; \Phi)}$$



Bernoulli distribution:

1. $Bern(x ; \theta) = \theta^{x} * (1 - \theta)^{1 - x}$

for x belongs to { 0, 1 }, the common estimator (**sample mean**) for θ is **$(1 / m) \sum_i x_i$**

2. $E[x] = \sum_i x_i * p_i = (1 * \theta^{1} * (1 - \theta)^{1 - 1}) + (0 * \theta^{0} * (1 - \theta)^{1 - 0}) = \theta$

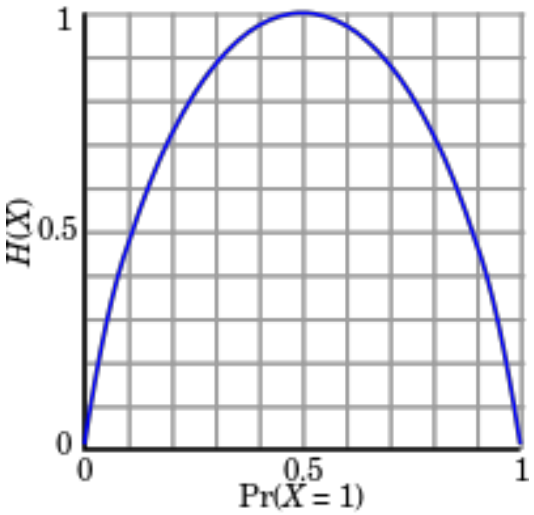
for x belongs to { 0, 1 }

3. $Var(x) = E[x^{2}] - E[x]^{2} = Bern(x = 0 ; \theta) * 0^{2} + Bern(x = 1 ; \theta) * 1^{2} - \theta^{2} = \theta(1 - \theta)$

for x belongs to { 0, 1 }

4. the entropy for a Bernoulli variable X

$H(x) = -\sum Bern(x ; \theta) \log(Bern(x ; \theta)) = -((1 - \theta) \log(1 - \theta) + \theta \log \theta)$



Binomial distribution:

1. $Bin(x ; n, \theta) = \binom{n}{x} \theta^{x} * (1 - \theta)^{n - x}$

https://www.youtube.com/watch?v=qIzC1-9PwQo

2. E[x] = nθ

https://www.youtube.com/watch?v=8fqkQRjcR1M

Proof:

$$E[x] = \sum x * p$$
$$= \sum x * [(n! / x! * (n - x)!) * \theta^{x} * (1 - \theta)^{n - x}]$$
$$= \sum [(n! / (x - 1)! * (n - x)!) * \theta^{x} * (1 - \theta)^{n - x}]$$
$$= n\theta * \sum [((n - 1)! / (x - 1)! * (n - x)!) * \theta^{x-1} * (1 - \theta)^{n - x}]$$
$$= n\theta * \sum [((n - 1)! / (x - 1)! * ((n - 1) - (x - 1))!) * \theta^{x-1} * (1 - \theta)^{(n - 1) - (x - 1)}]$$
$$= n\theta * \sum Bin(x - 1 ; n - 1, \theta)$$
$$= n\theta * 1$$
$$= n\theta$$

2. Var(x) = nθ(1 - θ)

https://www.youtube.com/watch?v=8fqkQRjcR1M

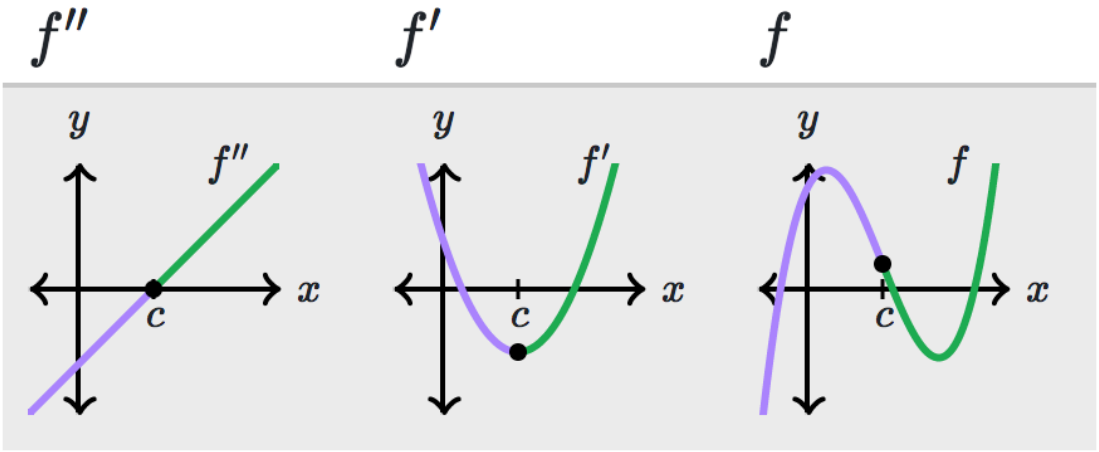
Proof:

$$Var(x) = E[x^{2}] - E[x]^{2}$$
$$= E[x^{2} - x + x] - E[x]^{2}$$
$$= E[x^{2} - x] + E[x] - E[x]^{2}$$
$$= E[x(x - 1)] + E[x] - E[x]^{2}$$
$$= \sum x(x - 1) * [(n! / x! * (n - x)!) * \theta^{x} * (1 - \theta)^{n - x}] + n\theta - (n\theta)^{2}$$
$$= \sum [(n! / (x - 2)! * (n - x)!) * \theta^{x} * (1 - \theta)^{n - x}] + n\theta - (n\theta)^{2}$$
$$= n * (n - 1) * \theta^{2} * \sum [((n - 2)! / (x - 2)! * (n - x)!) * \theta^{x - 2} * (1 - \theta)^{n - x}] + n\theta - (n\theta)^{2}$$
$$= ((n\theta)^{2} - n\theta^{2}) * \sum [((n - 2)! / (x - 2)! * ((n - 2) - (x - 2))!) * \theta^{x - 2} * (1 - \theta)^{(n - 2) - (x - 2)}] + n\theta - (n\theta)^{2}$$
$$= ((n\theta)^{2} - n\theta^{2}) * \sum Bin(x - 2 ; n - 2, \theta) + n\theta - (n\theta)^{2}$$
$$= ((n\theta)^{2} - n\theta^{2}) * 1 + n\theta - (n\theta)^{2}$$
$$= (n\theta)^{2} - n\theta^{2} + n\theta - (n\theta)^{2}$$

$$= n\theta - n\theta^2$$
$$= n\theta(1 - \theta)$$

How f'' informs about the concavity of f

When the second derivative f'' is positive, that means the first derivative f' is increasing, which means that f is concave up.
Similarly, a negative f'' means f' is decreasing and f is concave down.



Taylor Series:

1. https://www.youtube.com/watch?v=3d6DsjIBzJ4
2. the basic intuition is it **transform the derivative information at a point to the output information near that point**. Precisely, if we want to approximate the curve of **cos(θ)** at the point **0** with the polynomial function controlled by some parameters C_N,

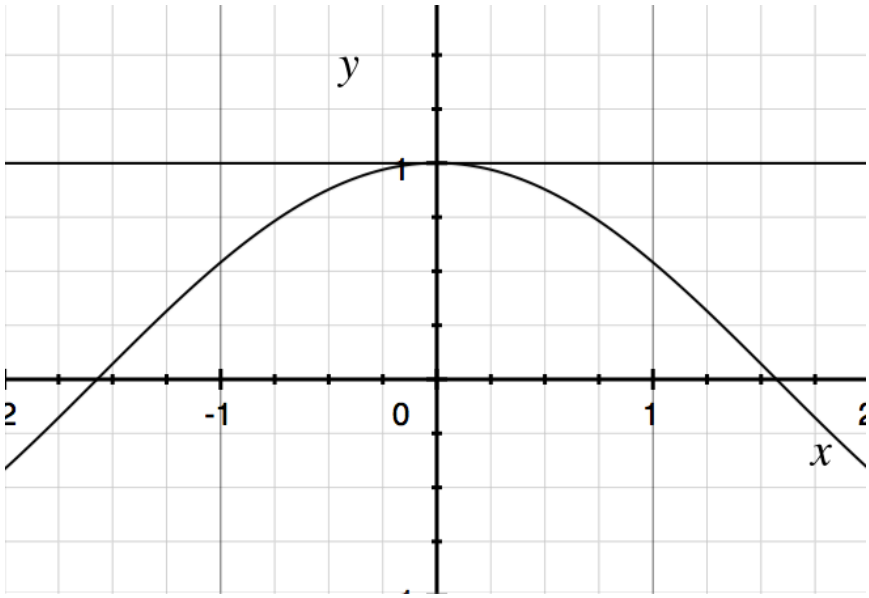
let's say

$$f(\theta) = C_0 + C_1\theta + C_2\theta^2 + C_3\theta^3 + \ldots + C_N\theta^N$$

The choice of $\mathbf{C_N}$ is determined by the derivative of **cos(θ)** and **f(θ)** at the point 0.

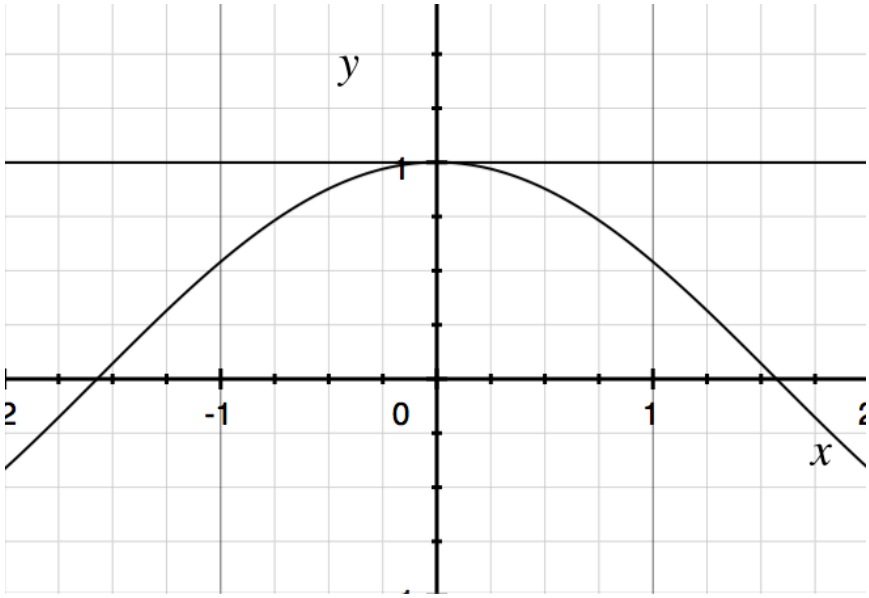
Let **N = 0**, $f(\theta) = C_0$
We approximate **cos(θ)** at the point 0 as a constant line. Let $\cos(0) = f(0)$, then $C_0 = 1$.

f(θ) = 1



Let **N = 1**, $f(\theta) = 1 + C_1\theta$
We approximate cos(θ) at the point 0 as a line with slop C_1. Let $\partial\cos(\theta) / \partial\theta \mid_{\theta=0} = \partial f(\theta) / \partial\theta \mid_{\theta=0}$, then $C_1 = 0$.

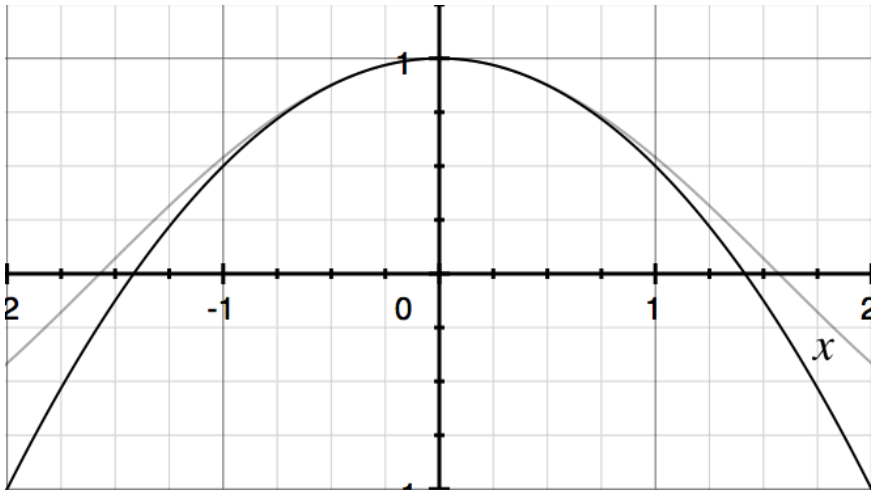
f(θ) = 1



Let **N = 2** , $f(\theta) = 1 + C_2\theta^2$

We approximate cos(θ) at the point 0 as a quadratic curve. Let $\partial^2\cos(\theta) / \partial\theta^2 |_{\theta=0} = \partial^2 f(\theta) / \partial\theta^2 |_{\theta=0}$, then $C_2 = -1/2$.

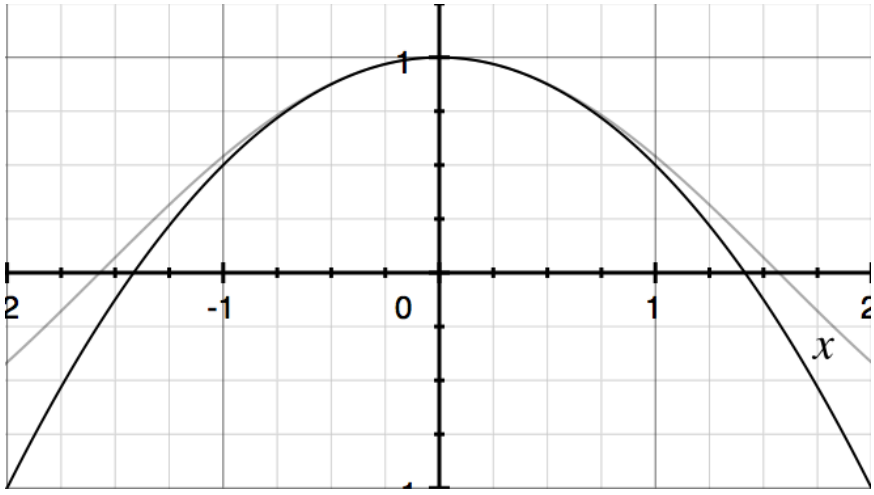
$f(\theta) = 1 - (1/2)\theta^2$



Let **N = 3** , $f(\theta) = 1 - (1/2)\theta^2 + C_3\theta^3$

Let $\partial^3\cos(\theta) / \partial\theta^3 |_{\theta=0} = \partial^3 f(\theta) / \partial\theta^3 |_{\theta=0}$, then $C_3 = 0$.

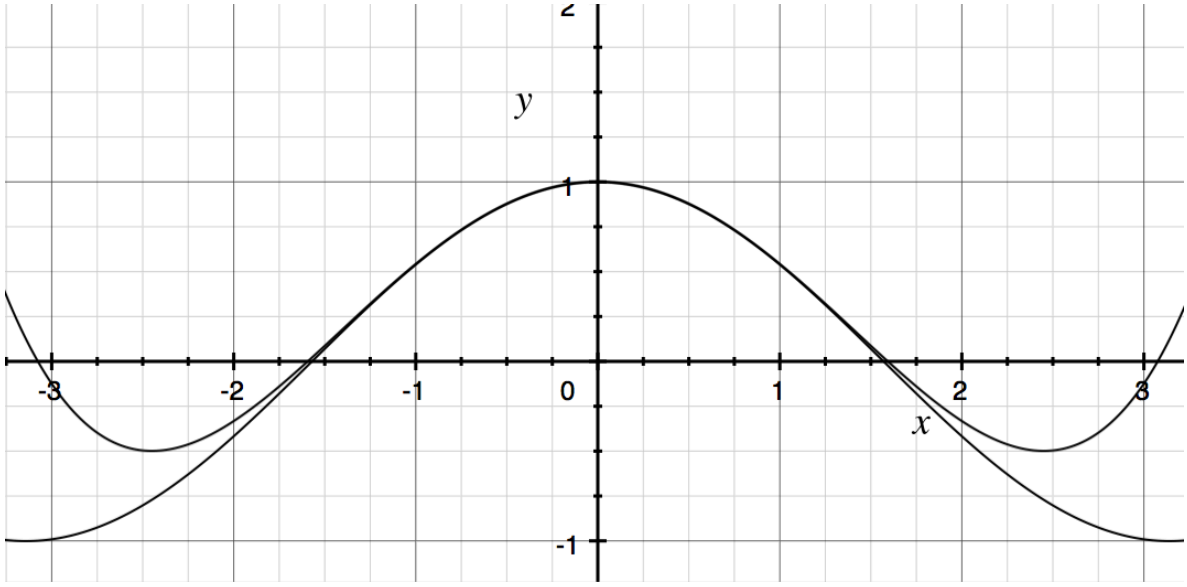
$f(\theta) = 1 - (1/2)\theta^2$



Let **N = 4** , $f(\theta) = 1 - 0.5\theta^2 + C_4\theta^4$

Let $\partial^4\cos(\theta) / \partial\theta^4 |_{\theta=0} = \partial^4 f(\theta) / \partial\theta^4 |_{\theta=0}$, then $C_4 = 4!$.

$f(\theta) = 1 - (1/2)\theta^2 + (1/4!)\theta^4$



...

The larger the exponent, the deeper the derivative matching, the more precise approximation near the point.

More generally, if **f(x)** is the approximation of the function **g(x)** at the point **ø**, with Taylor Series we have,

$$f(x) = g(ø) + (\partial g(x)/\partial x|_{x=ø}(x-ø)^1/1!) + (\partial^2 g(x)/\partial x^2|_{x=ø}(x-ø)^2/2!) + \ldots + (\partial^n g(x)/\partial x^n|_{x=ø}(x-ø)^n/n!)$$

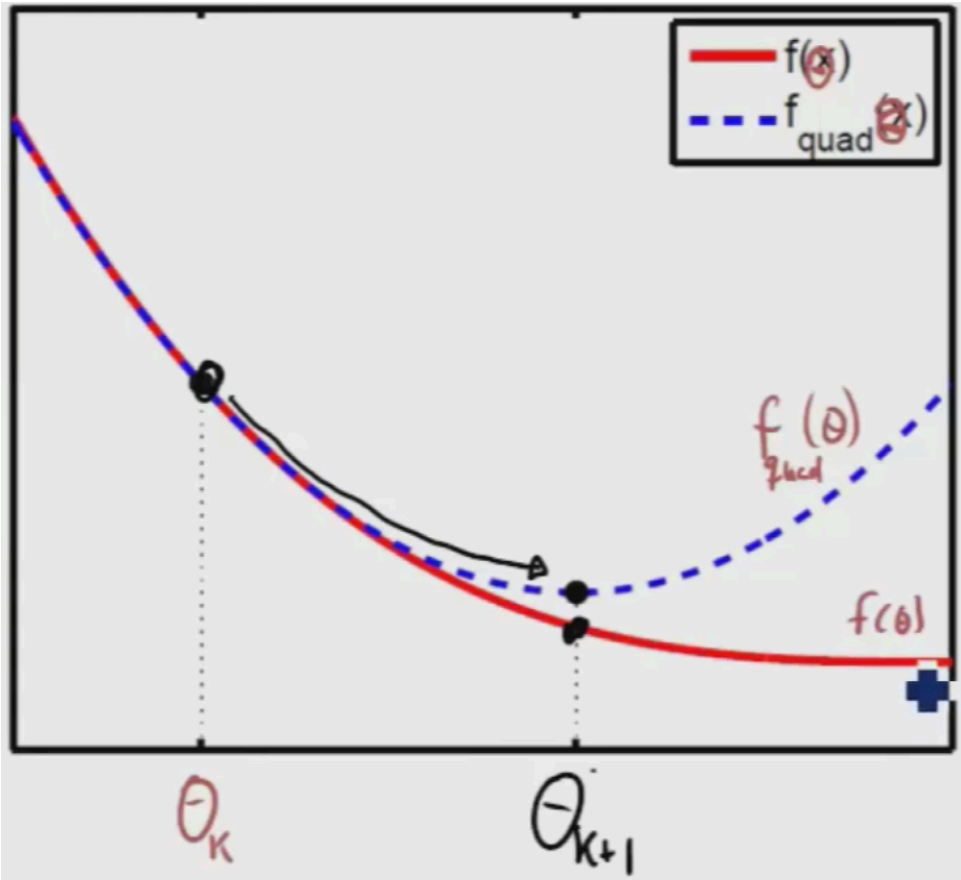
check out https://www.youtube.com/watch?v=ClFrIg0PpnM for multivariate Taylor Series.

Newton's Method for Optimization

Making the **2nd-order Taylor Series** approximation of the cost function **f(θ)** around θ_k, we will get a quadratic approximation of **f(θ)** near θ_k,

$$f_{quad}(\theta) = f(\theta_k) + (\theta - \theta_k)^T g_k + (1/2)(\theta - \theta_k)^T H_k (\theta - \theta_k)$$

where g_k and H_k are the gradient and hessian of **f(θ)** at the point **θ**.



So the next point θ_{k+1} for the steepest step of the cost function **f** can be calculated by finding the **zero gradient** point of the quadratic approximation

function, $\nabla f_{quad} = 0$, thus,

$$\theta_{k+1} = \theta_k - H_k^{-1} \bullet g_k$$

And that is the brilliant **Newton's Method**.

Definition of Residual:

In regression analysis, the difference between the observed value of the dependent variable (y) and the predicted value ($\hat{y}$) is called the **residual** (e). Each data point has one residual.

Definition of Gamma function:

1. Γ(n + 1) = n! = ∫ e^(-t) * t^n * dt [1] = (z - 1)!
2. using the Taylor Series, we can approximate e^(-t) * t^n ≈ (n^n / e^n) * e^(-(ε^2) / 2n) [2], and taking [2] into the integration [1], we get the Stirling Approximation Equation.
3. the Gamma prior is conjugated to the Poisson or Gaussian distribution.

Definition of Beta function:

1. Beta function is used for modeling the prior of Bernoulli distribution in **Bayesian learning procedure**.

$$
\begin{aligned}
f(x;\alpha,\beta) &= \text{constant} \cdot x^{\alpha-1}(1-x)^{\beta-1} \\
&= \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\,du} \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\,x^{\alpha-1}(1-x)^{\beta-1} \\
&= \frac{1}{\mathrm{B}(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1}
\end{aligned}
$$

2. as **α, ß** being determined, the posterior probability can be calculated with the conjugate of prior probability and likelihood, instead of the hard work of the marginal probability calculation using integral which might be a NP problem in high dimensional space.
3. Beta prior is conjugated to the Bernoulli distribution.

Representer Theorem

Suppose the regularized risk is defined as

$$J_\lambda(\theta) = (1/m) \sum_i \text{Loss}(\theta^T x_i, y_i) + (\lambda/2) \|\theta\|^2$$

Then there is a minimizer of the risk (0 gradient) that can be written as

$$\theta = \sum_i \alpha_i x_i$$

where $\alpha_i = -\text{Loss}'(\theta^T x_i, y_i) / \lambda$.

Linear Transformation

1. The **prime** form of a linear transformation

   $$\mathbf{y} = \boldsymbol{\theta}^T \mathbf{x}$$

   $\boldsymbol{\theta}$ is perpendicular to $\mathbf{y}$.

2. Based on the representer theorem, the linear transformation can be always written as the **dual** form

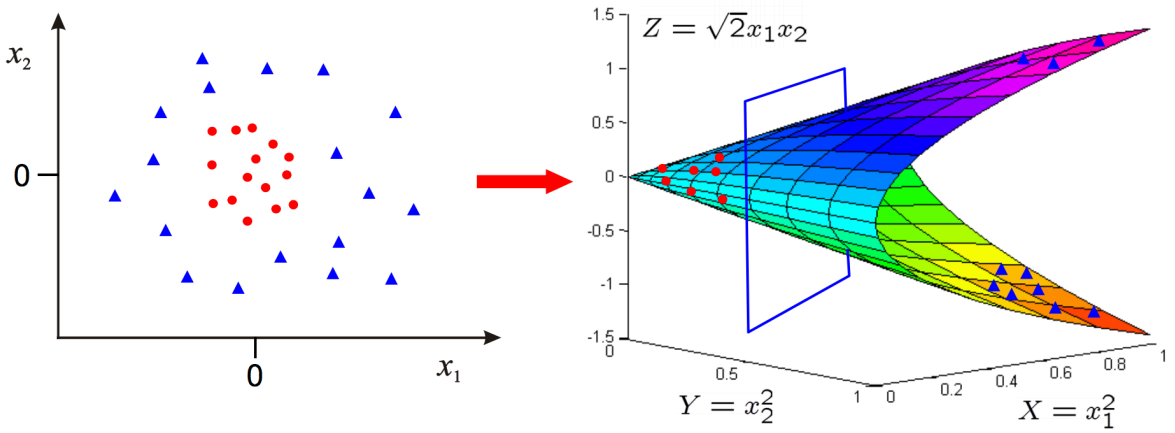
   $$\mathbf{y} = \sum_i \alpha_i \mathbf{x}^T \mathbf{x}_i$$

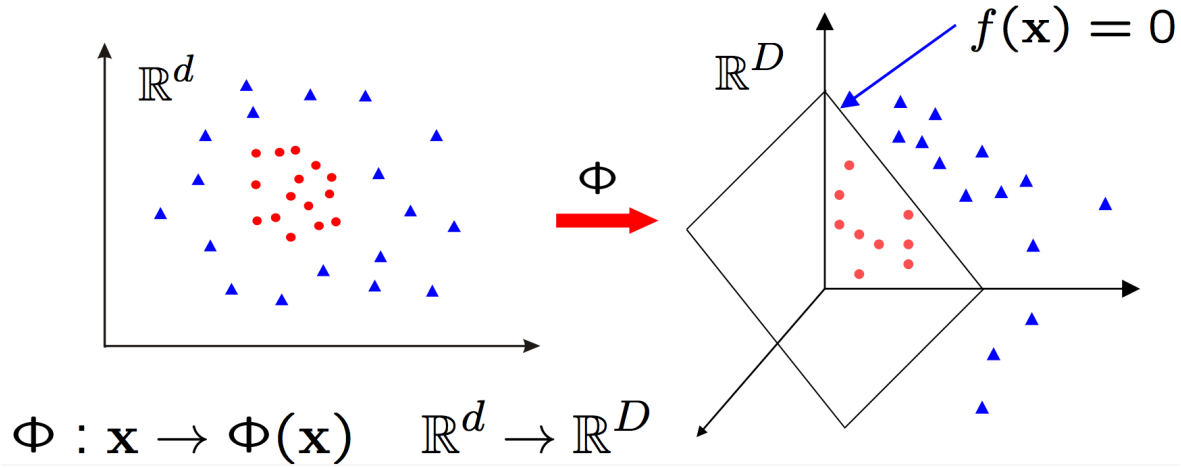
   That is, in any learning algorithm, minimize directly over $\boldsymbol{\alpha} \in R^m$

Kernel Trick

1. Sometimes data are not linearly separable in low dimensional space, it can be solved by mapping the data to higher dimension.

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$





the **Inner Product** of the mapping function **Φ** is called **Kernel**.

$$K(x, x) = \phi(x)^T \phi(x).$$

Then we can rewrite the regularized risk as

$$J_\lambda(\alpha) = (1 / m) \sum_i Loss(\sum_j \alpha_i K(x_i , x_j)) + (\lambda / 2) \sum_{ij} \alpha_i \alpha_j K(x_i , x_j)$$

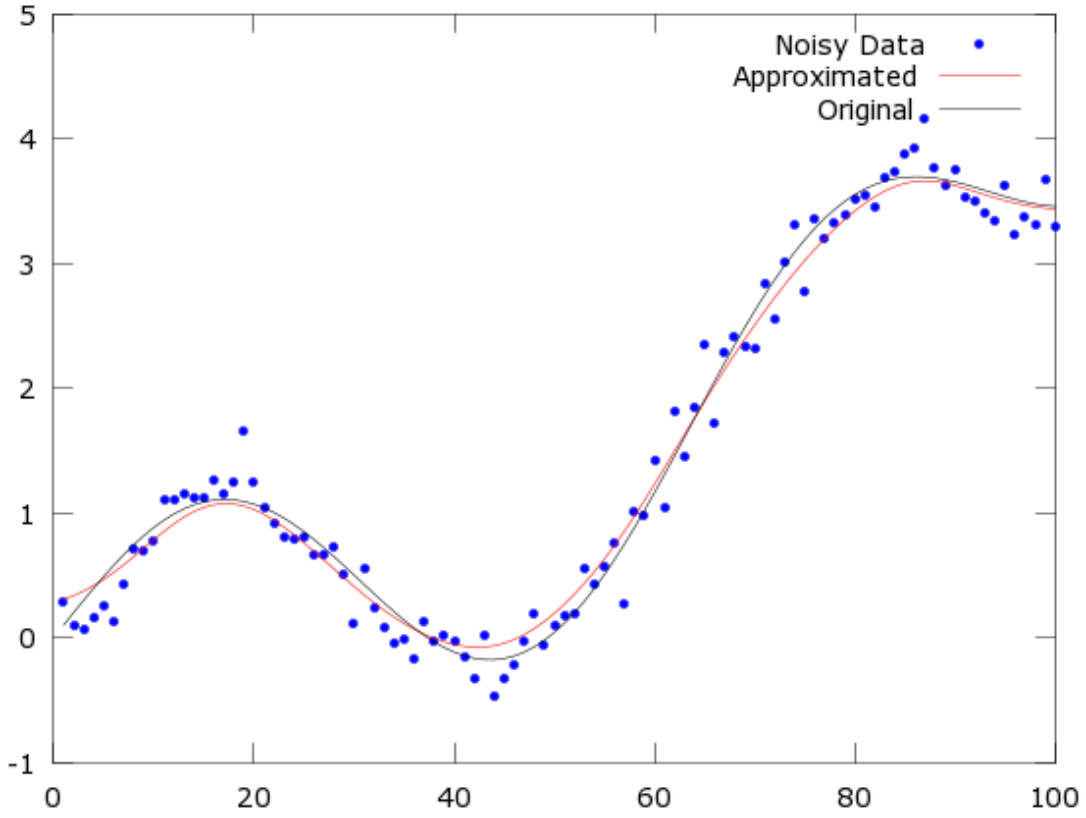
2. Kernel Trick

The kernel function **K(x , z)** may be very inexpensive to calculate, even though **ϕ(x)** itself may be very expensive to calculate (perhaps because it is an extremely high dimensional vector). In such settings, we can learn higher dimensional features spaces, without actually having to map the points into the high dimensional space, say calculate **ϕ(x)**.

For example, the Gaussian or Radial Basis Function (RBF) kernel is applicable to data in any dimension.

$$K(x, x0) = \exp (-|| x - x' ||^2 / 2\sigma^2) \text{ for } \sigma > 0$$

Gaussian Kernel Regression:



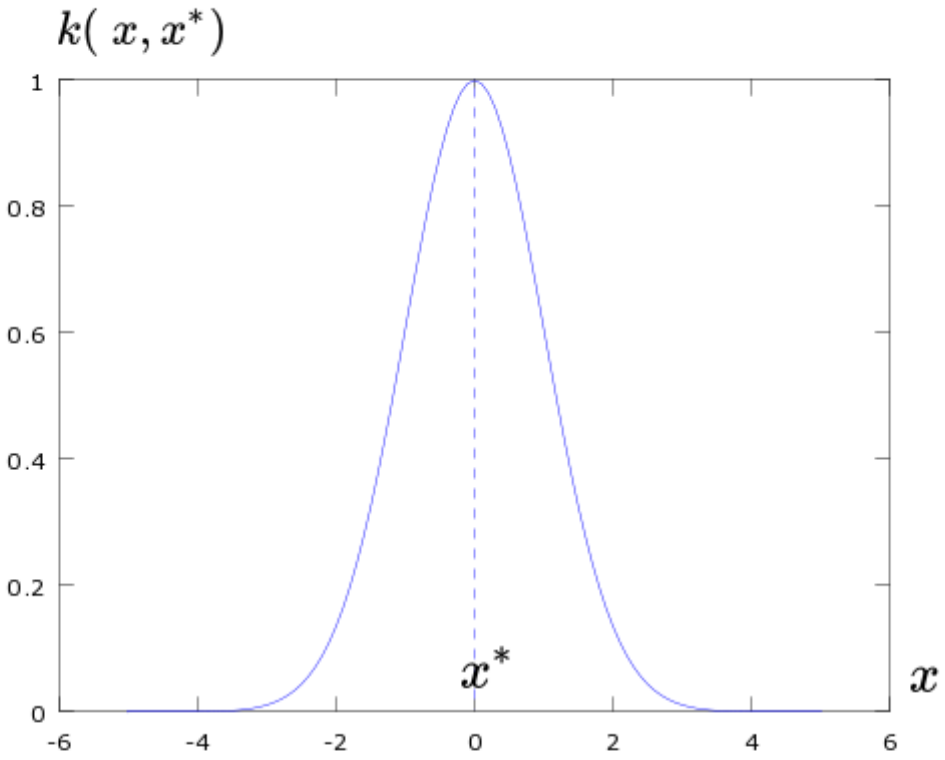
Gaussian Kernel Regression is a technique for **non-linear** regression.
Instead of the **Iterative Learning** method (such as gradient decent in DNN), Gaussian Kernel Regression approximate a function using **Weighted Average** method,
which is the prediction **y-hat** could be computed by **the history of y with some chosen weights** introduced.

$$\bar{y} = \frac{\sum_{i=1}^{m}(w_i y_i)}{\sum_{i=1}^{m} w_i}$$

So how to choose the weight ?

The intuition here is that if the history point is closer to the query point, it should be considered to be important to the prediction (a higher weight). Likewise, if the history point is far away from the query point, it should be considered to be negligible to the prediction (a lower weight).

The gaussian function has exactly the behavior we want for computing the weight values.



$$k(x, x^*) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-x^*)^2}{2\sigma^2}}$$

The denominator will cancel out when compute the weighted average, thus

$$k(x, x^*) = e^{-\frac{(x-x^*)^2}{2\sigma^2}}$$

which is called the **Gaussian Kernel**.

http://mccormickml.com/2014/02/26/kernel-regression/

Gaussian Process:

A GP is a Gaussian distribution over functions **f(x_i)**.

$$f(x) \sim N(e(x), k(x, x')) + \epsilon,$$

$$e(x) = E[f(x)],$$

$$\epsilon \sim N(0, \sigma^2),$$

$$k(x, x') = E[(f(x) - e(x))(f(x') - e(x'))^T] + \sigma^2 I,$$

k(x, x′) is called the **kernel function** which finds out the covariance of f(x) and f(x′) as given x and x′, plus the variance of the noise.

There are different ways to choose kernel function for different problems, here choose

$$k(x, x') = \exp(-(1 / 2\ell^2)(x - x')^2)$$

for GP.

Prediction:

As new data comes in, the joint probability of **f, f*** is

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right)$$

Then we can get **p(f_* | f)** with the formula of multivariate gaussian conditional probability.

$$\begin{aligned} p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f}) &= \mathcal{N}(\mathbf{f}_* | \mu_*, \Sigma_*) \\ \mu_* &= \mu(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{f} - \mu(\mathbf{X})) \\ \Sigma_* &= \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \end{aligned}$$

Gaussian Process is equivalent to **Ridge Regression**, which means introducing the regularization is introducing noise into the hypotheses actually.
Moreover, if you take a one-layer hidden neural network, and just make the number of neurons goes infinity, this NN actually is a Gaussian Process Model.

https://www.youtube.com/watch?v=MfHKW5z-OOA&index=9&list=PLE6Wd9FR--EdyJ5lbFl8UuGjecvVw66F6
https://www.youtube.com/watch?v=ewJ3AxKclOg

Exponential Family:

The Exponential Family of distributions of y as those having the form:

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

1. **η** is called the natural parameter (also called the canonical parameter) of the distribution
2. **T(y)** is the sufficient statistic (T(y) is accurate enough to describe the distribution. T(y) = y in general)
3. **a(η)** is the log *partition function*. The quantity $e^{-a(\eta)}$ essentially plays the role of normalization constant, that makes sure the distribution **p(y ; η)** sums / integrates over y to 1.

In thermodynamics, partition function is calculated by minimizing the entropy over Pi constrained by average energy, average momentum etc., with the LaGrange multiplier. It is the most important part for the Canonical ensemble.

4. $-\nabla a(\eta) = E[T(y)]$

Generalized Linear Model (GLM) :

Given a data set X (imagine the pixels input from training images), we want to understand the expectation of the (conditional) distribution of Y | X, in particular we would like to use the linear predictor $\eta = \beta^T x$. In the linear model, we use η as the predictor for $E[Y | X]$

A generalization of this is the Generalized Linear Model (GLM):

$$\eta = \beta^T x = g(E[Y | X])$$

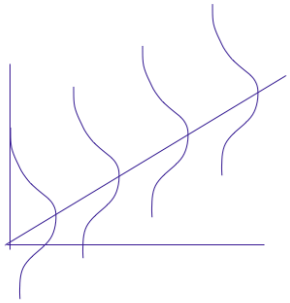
where $g(\bullet)$ is some link function (it links the **expectation** of the distribution of Y to the linear expression).

Generalised linear models – components

- **Distribution** of Y_i's:

  linear models $Y_i \sim N(\mu_i, \sigma^2)$

  GLMs $Y_i \sim$ exponential family



- **Linear predictor** = function of the covariates:

  linear models $\eta_i = \alpha + \beta x_i$

  GLMs eg $\eta_i = \alpha + \beta x_i + \gamma x_i^2$

- **Link function** = connection between the linear predictor and $\mu_i = E(Y_i)$:

  linear models $\eta_i = \mu_i \Rightarrow \mu_i = \alpha + \beta x_i$

  GLMs eg $\eta_i = \ln \mu_i \Rightarrow \mu_i = e^{\alpha + \beta x_i + \gamma x_i^2}$

Gradient:

1. The sum of all the partial derivative vectors of a function. It is a vector that always perpendicular to the contour line of the function.

$$\operatorname{div}\mathbf{F} = \nabla \cdot \mathbf{F} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot (U, V, W) = \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} + \frac{\partial W}{\partial z}.$$

From the definition of derivative, the ∂F / ∂x represent how much F change respective to the slitty change of x. It can be also interpreted as, the value of vector ∂F divided by vector ∂x, the result is also a vector.

2. The derivative of the gradient, also the second order derivative $\partial^2 F / \partial x^2$, represent the **sharpness** of the curve.

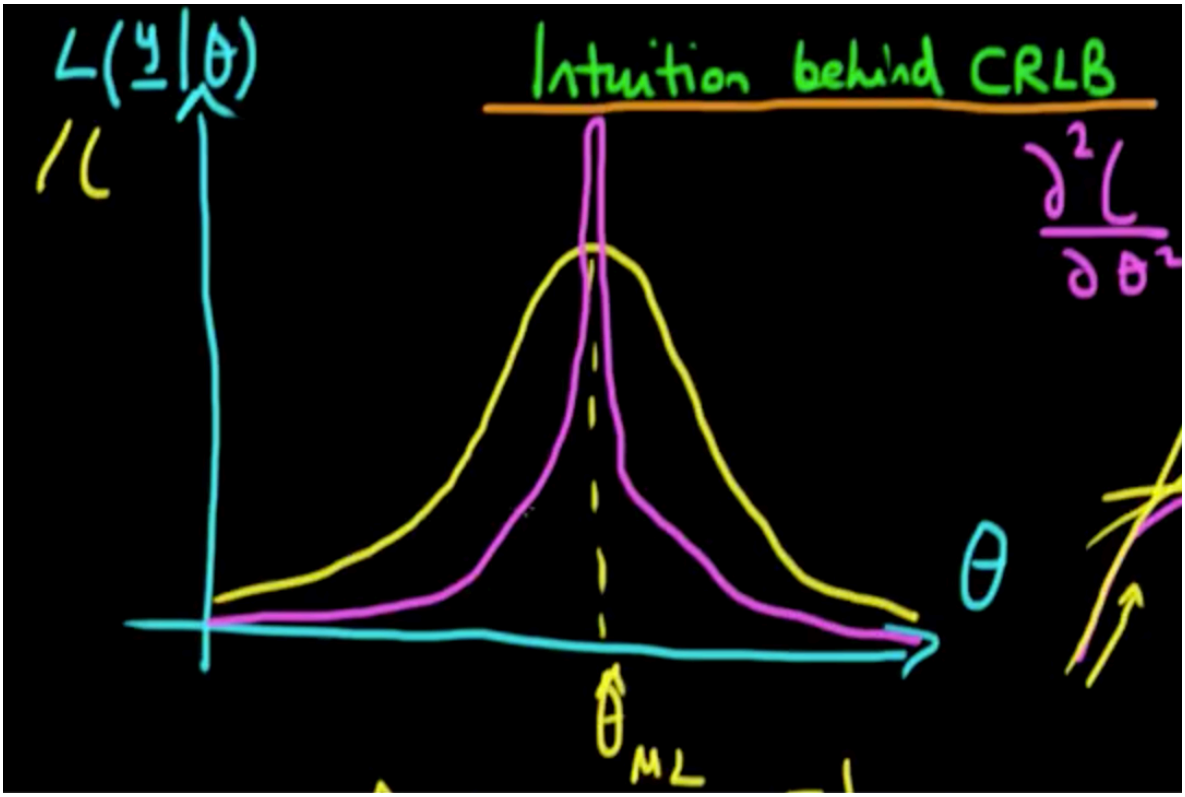
The higher the $\partial^2 F / \partial x^2$, the sharper the curve, thus the low variance of the distribution in probability distribution function (PDF).

The Relationship between the Curvature (2nd derivative, or Information matrix I) of Log-likelihood function and the Variance: Cramer-Rao Lower Bound

$$\text{Var}(\theta) \geq I^{-1}(\theta) = [\partial^2 L(\theta) / \partial \theta^2]^{-1}$$

Think of it in geometry, a gaussian distribution for instance,

more narrow the bell -> lower variance the distribution has -> larger the curvature -> less uncertainty



Furthermore, the Fisher Information Matrix is the negative Hessian Matrix:

$$I(\theta) = -H(\theta) = -\partial^2 L(\theta) / \partial\theta_i \partial\theta_j, \quad 1 \le i, j \le p$$

Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation is a **frequentist (discriminative) statistics** approach that can be used to derive parameter (point) estimators in a parametric estimation problem.

$$\theta_{ML} = \text{argmax}_\theta L(X ; \theta)$$
$$= \text{argmax}_\theta \prod_i L(x_i ; \theta)$$
$$= \text{argmax}_\theta \sum_i \log L(x_i ; \theta)$$
$$= \text{argmax}_\theta (1 / N) \sum_i \log L(x_i ; \theta) \text{ \# Monte Carlo estimate of the true cross entropy}$$
$$= \text{argmax}_\theta \sum_i p(x_i) \log L(x_i ; \theta)$$
$$= \text{argmin}_\theta - \sum_i p(x_i) \log L(x_i ; \theta) \text{ \# cross entropy}$$

We can thus see maximum likelihood as an attempt to make the model distribution match the empirical distribution . Any loss consisting of a negative log-likelihood is a cross entropy between the empirical distribution defined by the training set and the model. We often phrase it as minimizing a cost function.

More generally, MLE can be used to estimate a conditional probability **P(Y | X ; θ)** in order to predict y given x for **classification** and **regression** problems.

$$\theta_{ML} = \text{argmax}_{\theta} \sum_i \log L(y_i \mid x_i ; \theta)$$

For example, mean squared error is the cross-entropy between the empirical distribution and a gaussian model. That is each data point is seen as gaussian distributed.

Proof:

for the linear transformation $y = w^T x$ with some gaussian noise $\epsilon \sim N(0 , \sigma^2)$

$$y = w^T x + \epsilon$$

recall that for a normally distributed random variable with mean **m** and variance **h**: $t \sim N(m, h)$, then a linear transformation of **t** given by $at + b$ is also a normally distributed random variable with mean $a * m + b$, and variance $h * a^2$.

y is the linear transformation of random variable ϵ with mean $w^T x$, and variance σ^2, which is

$$y \mid x \sim N(w^T x , \sigma^2)$$

thus

$$P(y \mid x) \sim N(y ; w^T x , \sigma^2)$$

the estimator of **f(y | x)** using MLE is

$$w_{ML} = \text{argmax}_w \, (1 / N) \sum_i \log p(y_i \mid x_i ; w) \quad \# \text{ cross entropy between the empirical distribution and a gaussian likelihood}$$

$$= \text{argmax}_w \, - \log\sigma - (1 / 2) \log(2\pi) - (1 / N) \sum_i [|| w^T x_i - y_i ||^2 / 2\sigma^2]$$

here we assume that the variance is fixed to some constant σ^2 chosen by the user.
Maximizing the log-likelihood is equivalent to minimizing $(1 / N) \sum_i || w^T x_i - y_i ||^2$, which is equivalent to the **MSE**

$$MSE_{train} = (1 / N) \sum_i || w^T x_i - y_i ||^2$$

The MLE is asymptotically normal

That is, as **training size** goes infinity, we have

$$\theta_{ML} \sim N(\theta , I^{-1})$$

where **I** is the negative of the Hessian Matrix (Fisher Information Matrix) of the log-likelihood function. According to the Cramer-Rao Lower Bound (see above), the MLE attains the lowest variance, which is I^{-1}, and is often considered the preferred estimator to use for machine learning.

Maximum A Posteriori (MAP):

Maximum A Posteriori is a **bayesian (generative) statistics** approach that consider all possible values of θ when making a prediction.

$$\theta_{MAP} = \text{argmax}_\theta \log p(y | x , \theta) + \log p(\theta)$$

MAP has the same formulas as for the MLE (maximum likelihood estimator) for θ, except that MLE look at the true parameter θ as a unknown parameter but not a random variable, whereas MAP represents it as a random variable. So it put the prior of θ into account. In practical applications, a common choice for prior p(θ) is to assume that $\theta \sim N(0 , \tau I^2)$ with the highest entropy to reflect a high degree of uncertainty in the value of θ before any observation, or **Beta** function, or using the **Monte Carlo** to sample the prior distribution.

The functionality of **prior to the MAP** is sort of like the **regularization to the MLE,** that both can be used to **prevent overfitting**.

For example, if consider a linear regression model with a gaussian prior on the weights **w**. If this prior is given by

$$p (w) = N(w ; 0, \tau I^{2}),$$

then

$$\log p (w) \propto (1 / \tau) w^{T} w$$

which is a **L2-Norm** regularizer.

Logistic Regression with MLE

1. Deriving the Log-likelihood

Since the pdf of y is a Bernoulli distribution y | x ~ Bernoulli(θ), we assume the conditional likelihood of all data are complied with:

$$p(y | x) = \text{Bernoulli}(\theta) = \theta^{y} \bullet (1 - \theta)^{1 - y}$$

where **θ** is the expectation of the Bernoulli distribution.
In the form of exponential family **y | x ~ b(y) exp { η^{T} T(y) - α(η) }**, the distribution of y is:

$$p(y | x) = \exp \{ (\log(\theta / (1 - \theta)) \bullet y + \log(1 - \theta) \}$$

where

1. $\eta = \log(\theta / (1 - \theta))$
2. $T(y) = y$
3. $\alpha(\eta) = - \log(1 - \theta) = \log(1 + e^{\eta})$
4. $b(y) = 1$

then,

$$\theta = 1 / (1 + e^{-\eta})$$

thus the **link function** is a **logistic function**.

By definition of Generalized Linear Model, **η** is the **linear predictor** to the expectation **θ** which is $\mathbf{w^T x}$, then we have

$$\theta = 1 / (1 + \exp \{ - w^T x - b \})$$

Taking back to the **p(y | x)** we have

$$p(y | x ; w) = [1 / (1 + \exp \{ - w^T x - b \})]^y \bullet [1 - 1 / (1 + \exp \{ - w^T x - b \})]^y$$

2. Maximizing Likelihood Estimator:

$$w_{ML} = \operatorname{argmax}_w \log L(y | x ; w)$$
$$= \operatorname{argmax}_w \log \prod_i [1 / (1 + \exp \{ - w^T x_i - b \})]^y \bullet [1 - 1 / (1 + \exp \{ - w^T x_i - b \})]^y$$
$$= \operatorname{argmax}_w \sum [y_i * \log(1 / (1 + \exp \{ - w^T x_i - b \}))] + [(1 - y_i) * \log(1 - 1 / (1 + \exp \{ - w^T x_i - b \}))] \quad \text{\# cross-entropy, convex}$$

the derivative of log-likelihood function respect to **w** is

$$dw = \sum_i x_i^T (1 / (1 + \exp \{ - w^T x_i - b \}))$$

then update w by dw:

$$w = w + \xi \bullet dw$$

LaGrange Multiplier:

the basic idea is that as maximizing the original function with the constraint function, both of the contour line of these 2 functions at the maximized value should have the same gradient. thus, ∇f = λ ∇g. In other words, λ is the derivative of function f respect to

the function g.

1. adding the constrains to the target with laGrange parameter
2. take the partial derivatives of each variable, **∂f / ∂x = λ(∂g / ∂x), ∂f / ∂y = λ(∂g / ∂y)**, etc. Then using combined with the constraint function g to solve λ and the vector that maximizing the original function.
3. the Regularization is essential the LaGrange Multiplier

Stirling Approximation:

N! = 1 * 2 * 3 * ... * N
logN! = log1 + log2 + log3 + ... + logN
= ∑ logx
= ∫ logx dx
= xlogx - x ranged from N to 1
= NlogN

thus,

N! = N^N * e^(-N)

Jensen's Inequality:

Given random variable X, g(X) is also random variable, then,

E[g(X)] ≥ g(E[X])

if and only if g is a convex function **(the Hessian H(g(X)) ≥ 0 for all X)**

More generally, taken latent random variable t, p(t = a1) = ß1, p(t = a2) = ß2, p(t = a3) = ß3, we have

ß1 • f(a1) + ß2 • f(a2) + ß3 • f(a3) >= f(ß1a1 + ß2a2 + ß3a3)

where ß1 • f(a1) + ß2 • f(a2) + ß3 • f(a3) is $\mathbf{E_{t\sim p}[\, f(\, t\,)\,]}$, f(ß1a1 + ß2a2 + ß3a3) is $\mathbf{f(\, E_{t\sim p}[\, t\,]\,)}$

Mutual Information:

1. **entropy** is the munis of self information **-H(p , p)**.
If we think of a distribution as the tool we use to encode symbols, then entropy measures the number of bits we'll need for this particular distribution.
This is optimal, in that we can't encode the symbols using fewer bits on average.

2. **cross entropy** measures the number of ***extra* bits** we'll need on average if we encode symbols from y according to y-hat.
Think of it as a bit tax for encoding symbols from y with an inappropriate distribution y-hat. It's never negative, and it's 0 only when y and y-hat are the same.
minimizing cross entropy = minimizing the KL divergence = minimizing the negative log likelihood of our data.
http://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

3. KL-divergence: the distance of uncertainty between 2 probability distributions

$$
\begin{aligned}
D(P \,||\, Q) &= \sum P(x) \log\left(P(x)/Q(x)\right) \\
&= E[P \log(P/Q)] \\
&= \sum P(x) \log P(x) \;-\; \sum P(x) \log Q(x) \\
&= H(P,P) - H(P,Q) \quad \#\ \text{entropy - cross entropy} \\
&= E[\text{bits of model P}] - E[\text{bits from model P when using model Q}]
\end{aligned}
$$

4.

$$
\begin{aligned}
I(X;Y) &= D[\,p(x,y) \,||\, p(x)\,p(y)\,] \\
&= D[\,p(y \mid x) \,||\, p(y)\,] \\
&= D[\,p(x \mid y) \,||\, p(x)\,] \\
&= H(X) + H(Y) - H(X;Y) \\
&= H(X) - H(X \mid Y)
\end{aligned}
$$

Thus we can interpret the MI between X and Y as the reduction in uncertainty

about X after observing Y , or, by symmetry, the reduction in uncertainty about Y after observing X.

5. the mutual information can be used to discover interesting relationships between variables in a way that simpler measures, such as correlation coefficient , cannot.

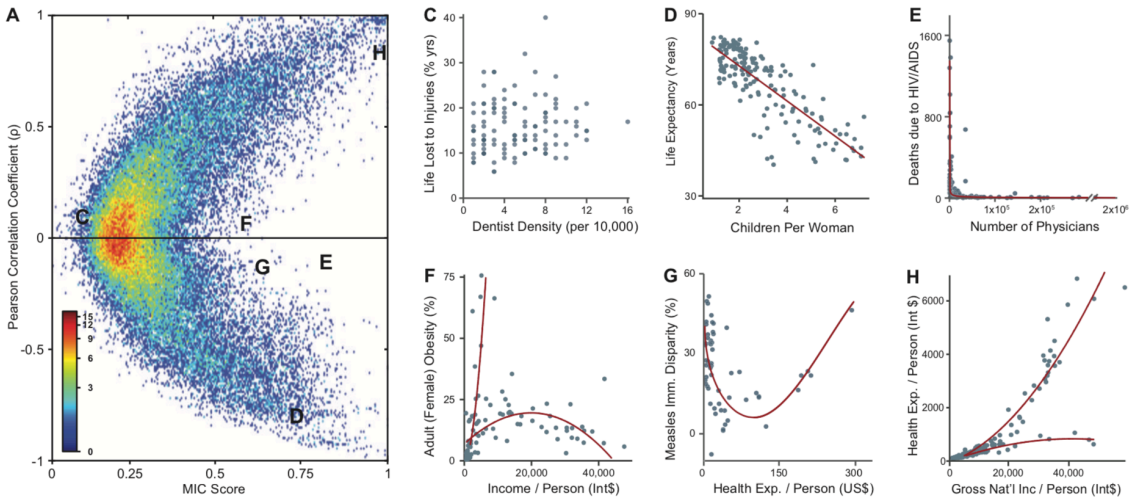


Figure 2.22 Left: Correlation coefficient vs maximal information criterion (MIC) for all pairwise relationships in the WHO data. Right: scatter plots of certain pairs of variables. The red lines are non-parametric smoothing regressions (Section 15.4.6) fit separately to each trend. Source: Figure 4 of (Reshed et al. 2011) . Used with kind permission of David Reshef and the American Association for the Advancement of Science.

Quick notes on Matrix

[v1, v2] is the vertical vector, and **[v1, v2]**^{T} is the horizontal vector.

Matrix Intuition

1. Matrix can be either interpreted as a collection of vector or a collection of transformation.
2. Matrix can be also interpreted as the relationship among the high dimensional data. If you dot product 2 vectors XY^T, you will get the covariance matrix of it.

Matrix Determinant

The determinant of a matrix is the **AREA** of the transformed **Identity** matrix transforming from it.

$$A = [\,[\,a\ b\,]^T\ [\,c\ d\,]^T\,] \quad \det|A| = ad - bc = \Pi\,\lambda i$$

Matrix Inversion

the inverted transformation respect to M,

$$M^{-1}M = MM^{-1} = I$$

Matrix Transpose

$$U^TV = V^TU$$

Span of Matrix

for R^{mxn} Matrix M, the span of M is all of the linear combinations of the column vector of M, denoted as

$$\{\, Mx \mid x \in R^n \,\}$$

where $Mx = \sum v_i x_i$

https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/null-column-space/v/column-space-of-a-matrix

Rank of Matrix

the number of **Linearly Independent Column Vectors**, aka basis, of a Matrix

https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/null-column-space/v/dimension-of-the-column-space-or-rank

Singular Matrices:

A Matrix is singular if A^{-1} is undefined.

$A = [\ [\ a\ b\ ]^T\ [\ c\ d\ ]^T\ ]$ $A^{-1} = 1 / \det|\ A\ | [\ [\ d\ -b\ ]^T\ [\ -c\ a\ ]^T\ ]$

A^{-1} is undefined IFF. the determinant of A ($\det|\ A\ |$) is 0.

It can be explained, in another way, as the identity vectors within the Identity matrix become parallel after the transformation.

Matrix Multiplication:

Matrix Multiplication is just the vector transformation.

$$M1\ M2 = [\ [\ a1\ ,\ a2\ ]\ [\ b1\ ,\ b2\ ]\ ]\ \ [\ [\ c1\ ,\ c2\ ]\ [\ d1\ ,\ d2\ ]\ ]$$

Think of the 2 X 2 matrix in 2D space with basic vector I = [i , j] = [[1 , 0] [0 , 1]].
In Matrix M1, the vector [a1, a2] represent scaling a1 and a2 represent to i and j, thus

$$a1 * [\ 1\ ,\ 0\ ] + a2 * [\ 0\ ,\ 1\ ] = [\ a1\ ,\ a2\ ]$$

Similarly, the vector [b1 , b2] represent scaling b1 and b2 represent to i and j, thus we have the Matrix M1.

Matrix can be seen as the transformation function, or the collection of transformed basic vector I.

Multiplied by M2, the transformed [i, j] would be scaled by another 2 vectors [c1 , c2] and [d1 , d2].

Matrix Dot Product:

$$[\ u1\ ,\ u2\ ]^T \bullet [\ v1\ ,\ v2\ ] = u1v1 + u2v2 = ||\ u\ ||\ ||\ v\ ||\ \cos\theta$$

The dot product project the 2D space vector [u1 , u2] on to a 1D line that the [v1 , v2] represented, and measure the direction (angle) through +/- of its result. $[\ u1\ ,\ u2\ ]^T$ -> [u1 , u2] is called the duality.

If the matrix with column vectors dot product equals 0 to each other,

orthogonal to each other in other words, is called the **Orthogonal Matrix**. Furthermore, if the length column vectors are 1, then the matrix is called **Orthonormal Matrix.**

See also https://www.youtube.com/watch?v=KDHuWxy53uM

Vector Projection

$$\mathbf{Proj_v(u) = || u || cos\theta = [(u^T v) / || v ||^2] v}$$

as projecting vector **u** onto vector **y**.

Eigenvector and Eigenvalue:

Any Symmetric Matrix ($M = M^T$) can be written as

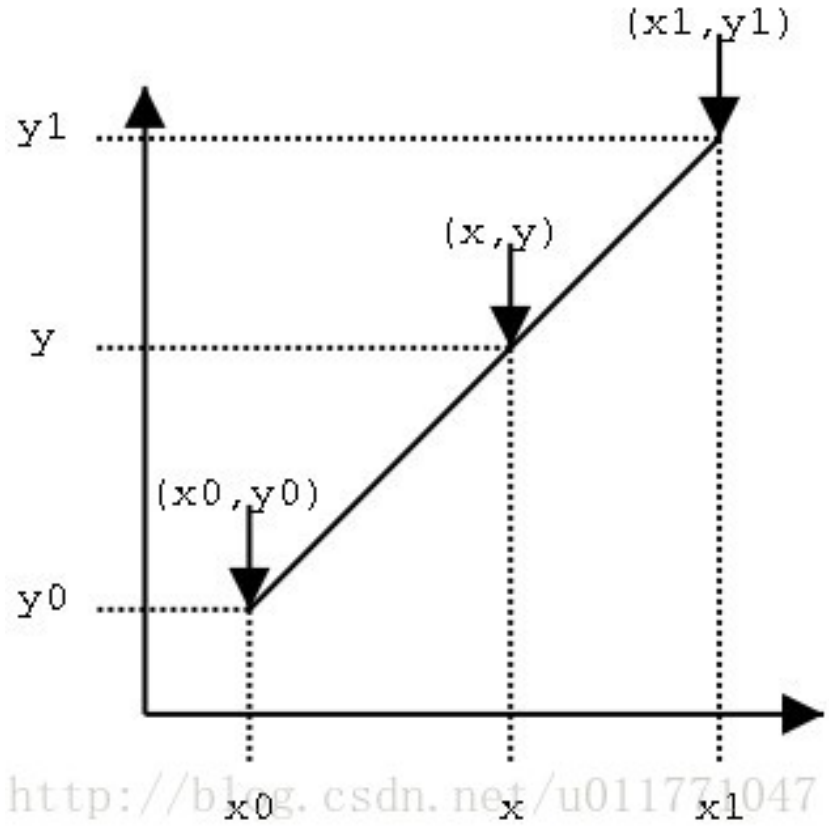
$$\mathbf{M = V\Lambda V^{-1} = \sum \lambda i \, vv^{-1}}$$

where
V is a collection of (orthogonal or so-called rotation) eigenvectors respect to eigenvalues Λ, $\mathbf{V^{-1} = V^T}$
Λ is the diagonal matrix of eigenvalues **λi**, which is

$$\mathbf{MV = \Lambda V}$$

https://www.youtube.com/watch?v=spNpfmWZBmg

Linear Interpolation



Given

function f that mapping from point x to its value f(x) = y,
points x0, x1,
and a query point x,

how to calculate

the value of x

Suppose **f(x) = ax + b**,

y1 = ax1 + b
y0 = ax0 + b

a = (y1 - y0) / (x1 - x0)
b = y0 - x0 [(y1 - y0) / (x1 - x0)]

y = x [(y1 - y0) / (x1 - x0)] + y0 - x0 [(y1 - y0) / (x1 - x0)]

= y0 + (x - x0) (y1 - y0) / (x1 - x0)

thus,

(x - x0) / (x1 - x0) = (y - y0) / (y1 - y0)

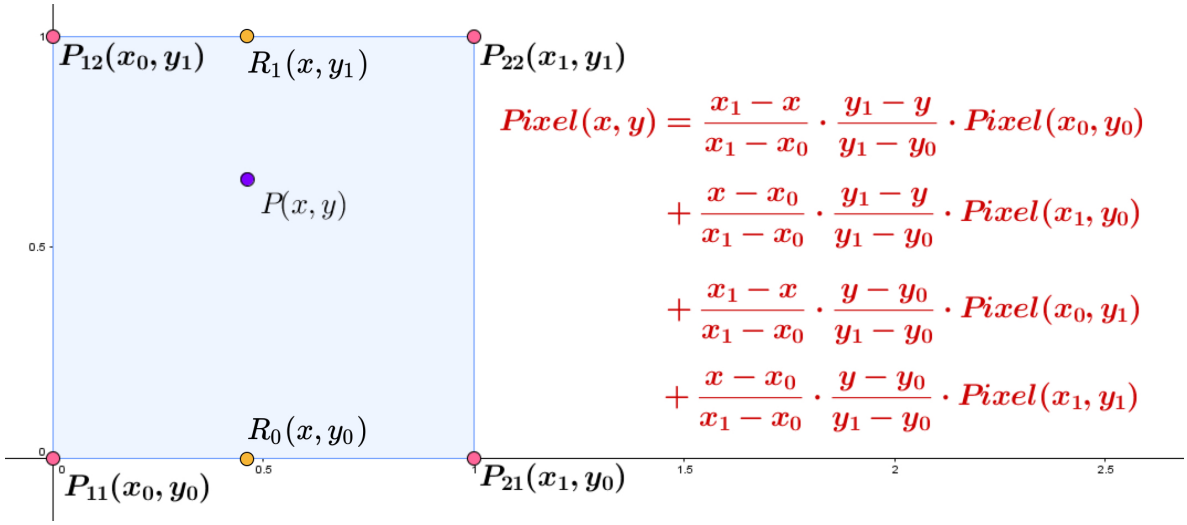
let

∂ = (x - x0) / (x1 - x0) = (y - y0) / (y1 - y0)

thus

y = f(x)
= y0 + ∂ (y1 - y0)
= (1 - ∂)y0 + ∂y1
= [1 - (x - x0) / (x1 - x0)] y0 + (x - x0) / (x1 - x0) y1
= y0 (x1 - x) / (x1 - x0) + y1 (x - x0) / (x1 - x0)
= f(x0) (x1 - x) / (x1 - x0) + f(x1) (x - x0) / (x1 - x0)
[Eq.1]

Bilinear Interpolation



Given

function f that mapping from point coordinates P(x, y) to its

value f(P(x, y)) = V,
points P11 = (x0, y0), P12 = (x0, y1), P21 = (x1, y0), P22 = (x1, y1)
and a query point P(x, y),

how to calculate

f(P(x, y))

1. Linear Interpolate along with **x** axis. From **[Eq.1]** above, we have,

f(R0) = f(P11) (x1 - x) / (x1 - x0) + f(P21) (x - x0) / (x1 - x0), where R0 = (x, y0) [Eq.2]
f(R1) = f(P12) (x1 - x) / (x1 - x0) + f(P22) (x - x0) / (x1 - x0), where R1 = (x, y1) [Eq.3]

which are the values of points **R1** and **R2**.

2. Linear Interpolate along with **y** axis,

f(P) = f(R0) (y1 - y) / (y1 - y0) + f(R0) (y - y0) / (y1 - y0) [Eq.4]

3. From **Eq.2**, **Eq.3** and **Eq.4**, we have,

f(P) = f(P11) (x1 - x) (y1 - y) / (x1 - x0) (y1 - y0)
+ f(P21) (x - x0) (y1 - y) / (x1 - x0) (y1 - y0)
+ f(P12) (x1 - x) (y - y0) / (x1 - x0) (y1 - y0)
+ f(P22) (x - x0) (y - y0) / (x1 - x0) (y1 - y0)

let

D = (x1 - x0) (y1 - y0)

we have,

f(P) = f(P11) (x1 - x) (y1 - y) / D
+ f(P21) (x - x0) (y1 - y) / D
+ f(P12) (x1 - x) (y - y0) / D
+ f(P22) (x - x0) (y - y0) / D

In image processing, **Q11**, **Q21**, **Q12**, **Q22** are neighboring pixels, so **D = 1**, thus

Pixel(P) = Pixel(P11) (x1 - x) (y1 - y)
+ Pixel(P21) (x - x0) (y1 - y)
+ Pixel(P12) (x1 - x) (y - y0)
+ Pixel(P22) (x - x0) (y - y0) [Eq.6]

where **Pixel** is the pixel value of some point **P**.

https://kevinzakka.github.io/2017/01/10/stn-part1/

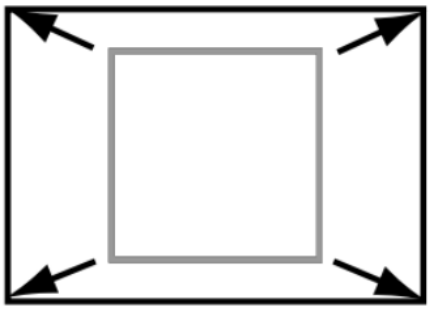
Affine (Linear) Transformation

1. Identity transform

$$K' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = K$$

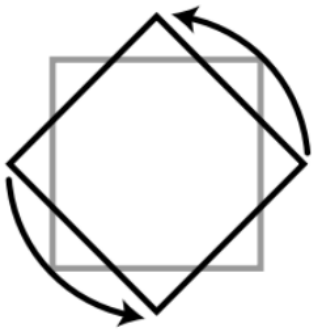
---

2. Scaling



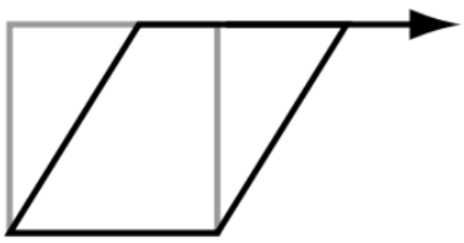
$$K' = \begin{bmatrix} p & 0 \\ 0 & q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} px \\ qy \end{bmatrix}$$

3. Rotation



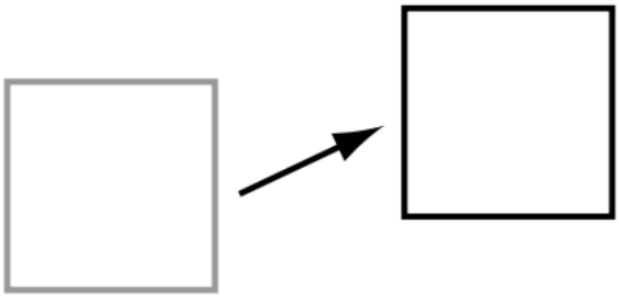
$$K' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

4. Sheer



$$K' = \begin{bmatrix} 1 & m \\ n & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + my \\ y + nx \end{bmatrix}$$

5. Translate



$$K' = \begin{bmatrix} 1 & 0 & \Delta \\ 0 & 1 & \Delta \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + \Delta \\ y + \Delta \end{bmatrix}$$

Affine Transformation + Bilinear Interpolation in Image Processing

An image processing affine transformation usually follows the 3-step pipeline below:

1. create a sampling grid composed of **(*x*, *y*)** coordinates. For example, given a 400x400 grayscale image, we create a grid of same dimension, that is, evenly spaced **x** ∈ **[0, W]** and **y** ∈ **[0, H]**.
2. apply the transformation matrix to the sampling grid generated in the step above.
3. sample the resulting grid from the original image using the desired interpolation technique.

https://kevinzakka.github.io/2017/01/10/stn-part1/

Singular Value Decomposition

For any matrix $A \in \mathbb{R}^{m \times n}$ there exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ and a 'diagonal' matrix $\Sigma \in \mathbb{R}^{m \times n}$, i.e.,

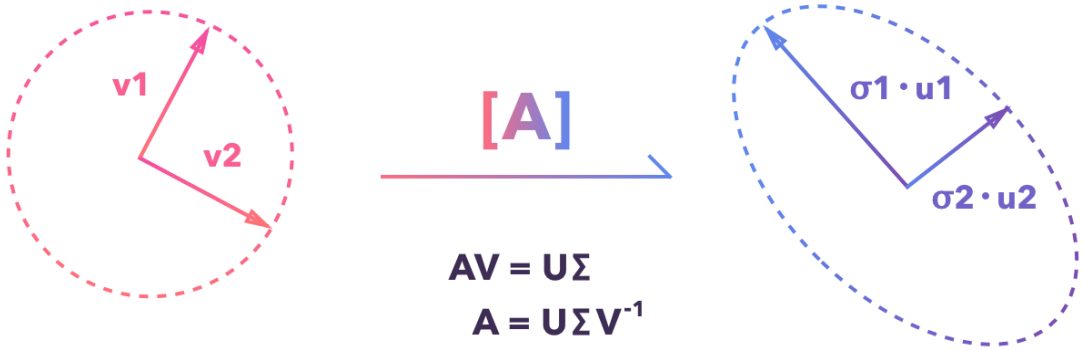
$$\Sigma = \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ 0 & & & & & 0 \\ \vdots & & & & & \vdots \\ 0 & & & & & 0 \end{pmatrix} \quad \text{for} \quad m \geq n$$

with diagonal entries

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_{\min\{m,n\}} = 0$$

such that $A = U\Sigma V^T$

Think of a matrix **A** that transforming from a orthogonal matrix **V = [v1, v2]** to another orthogonal matrix **U = [u1, u2]** as of transforming from a circle with radius **v1** (v1 = v2 in this special case) to a ellipse with axis **u1, u2**.



where

- **U is the eigenvectors respect to the squared root of eigenvalues ∑ for covariance matrix AA^T**
- **Say A is the data (design) matrix, the k-th principle components of A are $u1\sigma1v1^T$, $u2\sigma2v2^T$, ... , $uk\sigma kvk^T$**
- **To projecting N demential data matrix A onto k demential principle components, $A_k = U^TA\ [\ 1 : k\ ]$**
- **Say A is the data (design) matrix, recall that the unbiased estimator of Variance is $Var(\ A\ ) = (\ 1\ /\ (\ N - 1\ )\ )\ AA^T$, then $Var(\ A\ ) = (\ 1\ /\ (\ N - 1\ )\ )\ U\Sigma^2U^T$**

Principle Component Analysis and Mahalanobis Distance:

Principle Component Analysis:

PCA reduces the demotions of the data while remaining the most important information.

It attempt to disentangle the unknown factors of variation underlying the data , this disentangling takes the form of finding a rotation of the input space (described by U) that aligns the principal axes of variance with the basis of the new representation space associated with $\mathbf{z = x^TU}$

To perform PCA, you calculate the eigenvectors of the data's covariance matrix. Typically, we use SVD to decompose the data then retrieving the **singular vectors U and variance ∑** as the eigenvectors and squared root of

eigenvalues of the covariance matrix.

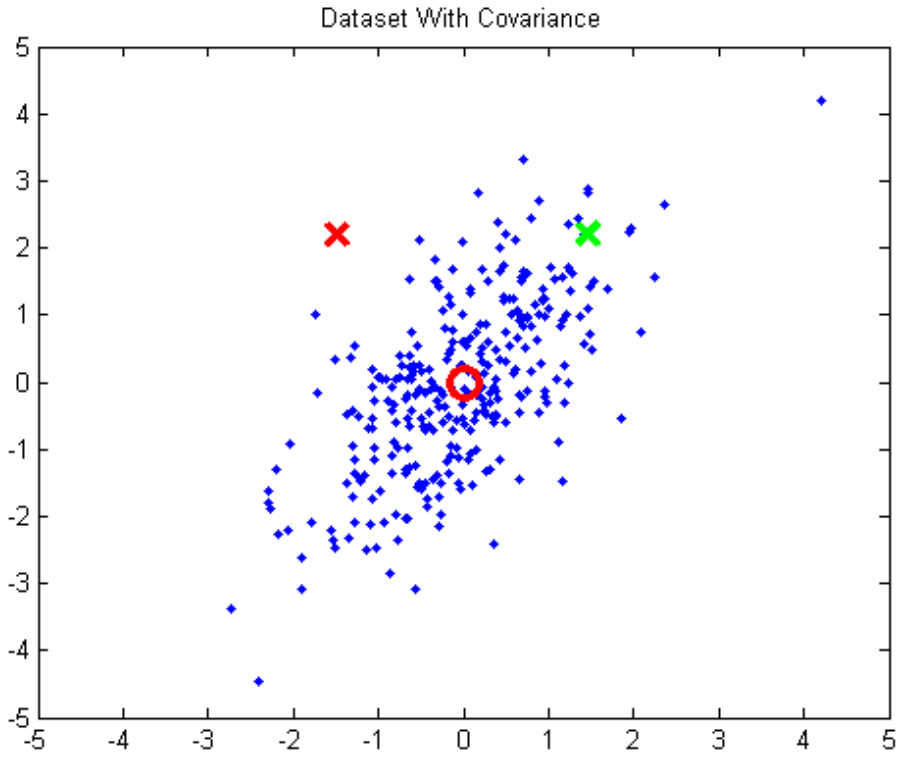
Mahalanobis Distance:

$$d_M(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)}$$

where S^{-1} is the inversion of the covariance matrix of X and Y.
[Side note]: the probability density function for a multivariate Gaussian distribution uses the Mahalanobis distance instead of the Euclidean.

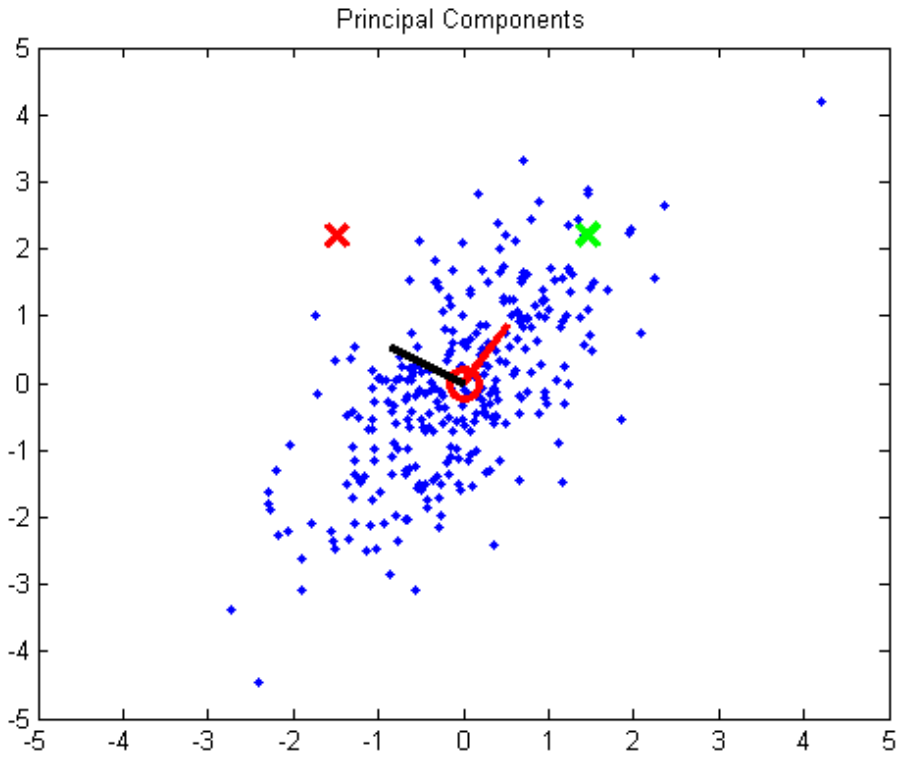
Mahalanobis Distance measuring the Euclidean Distance throughout the Non-Correlational data (wipe out the correlation through S^{-1}).

Intuitively, one can think of it as the following steps:

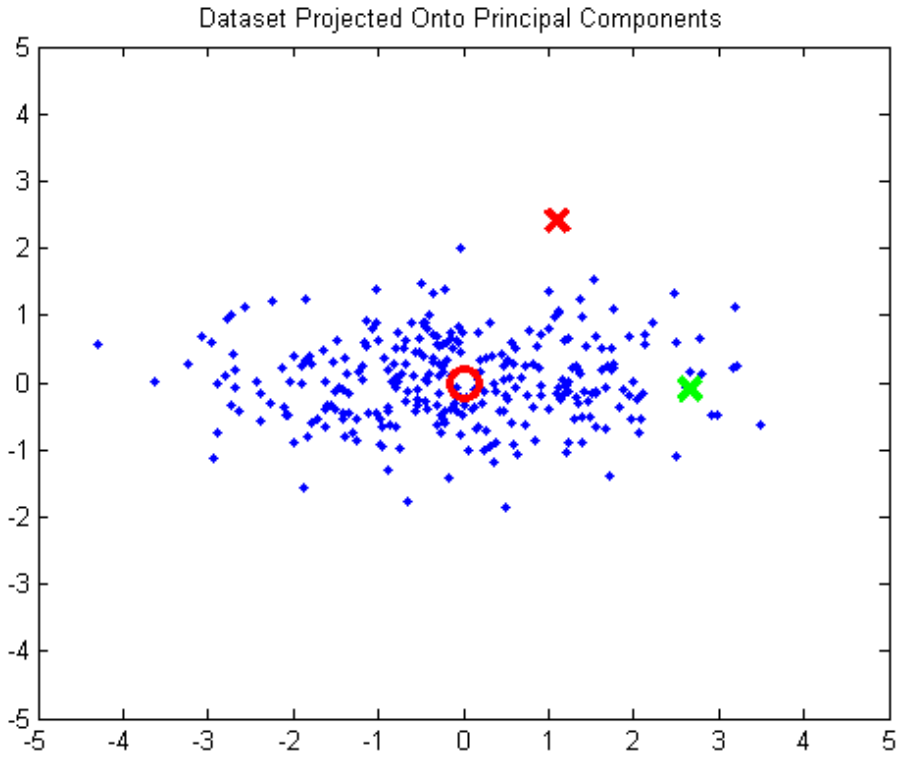


Here, the covariance of the data is non-zero since the principle components are **not aligned with axes**. Thus the RED and GREEN test points shouldn't be considered equally with the same similarity to the data distribution even though they have the same Euclidean Distance to the mean of the data.

The solution is to find the principle components of the data using SVD:

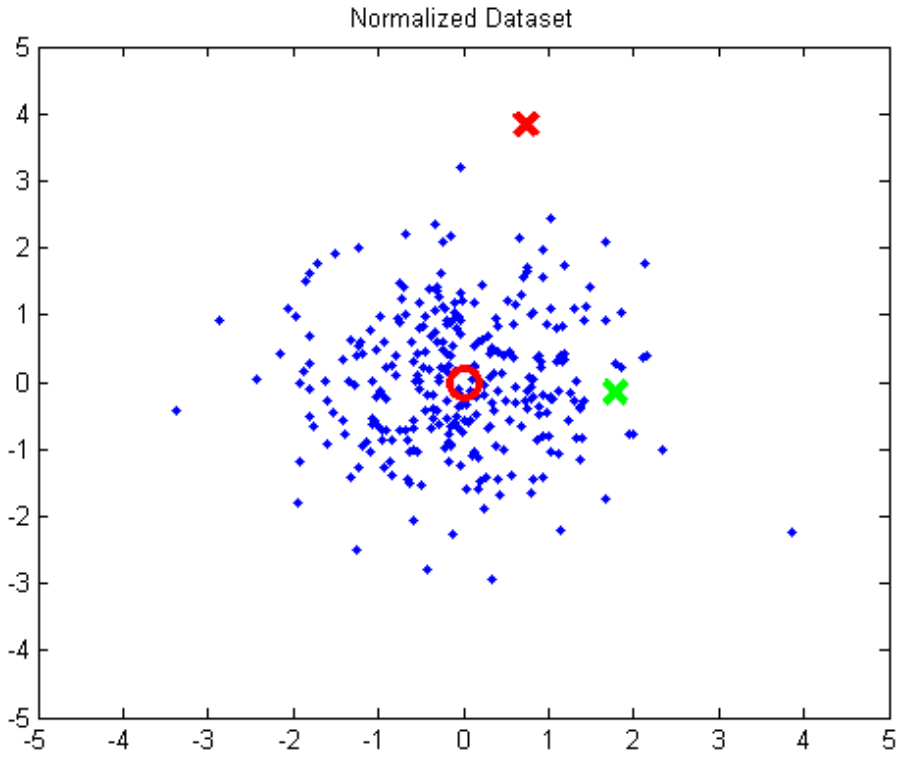


Then projecting data onto these principle components. This is the crucial step to wipe out the covariance of the data.



Finally, normalizing the data along with each of the principle components by

dividing the standard divination ∑ (the squared root of eigenvalues for the covariance matrix). The Euclidean Distance of GREEE point is lower than the RED one now.



The 3 steps above are exactly the PCA whitening.

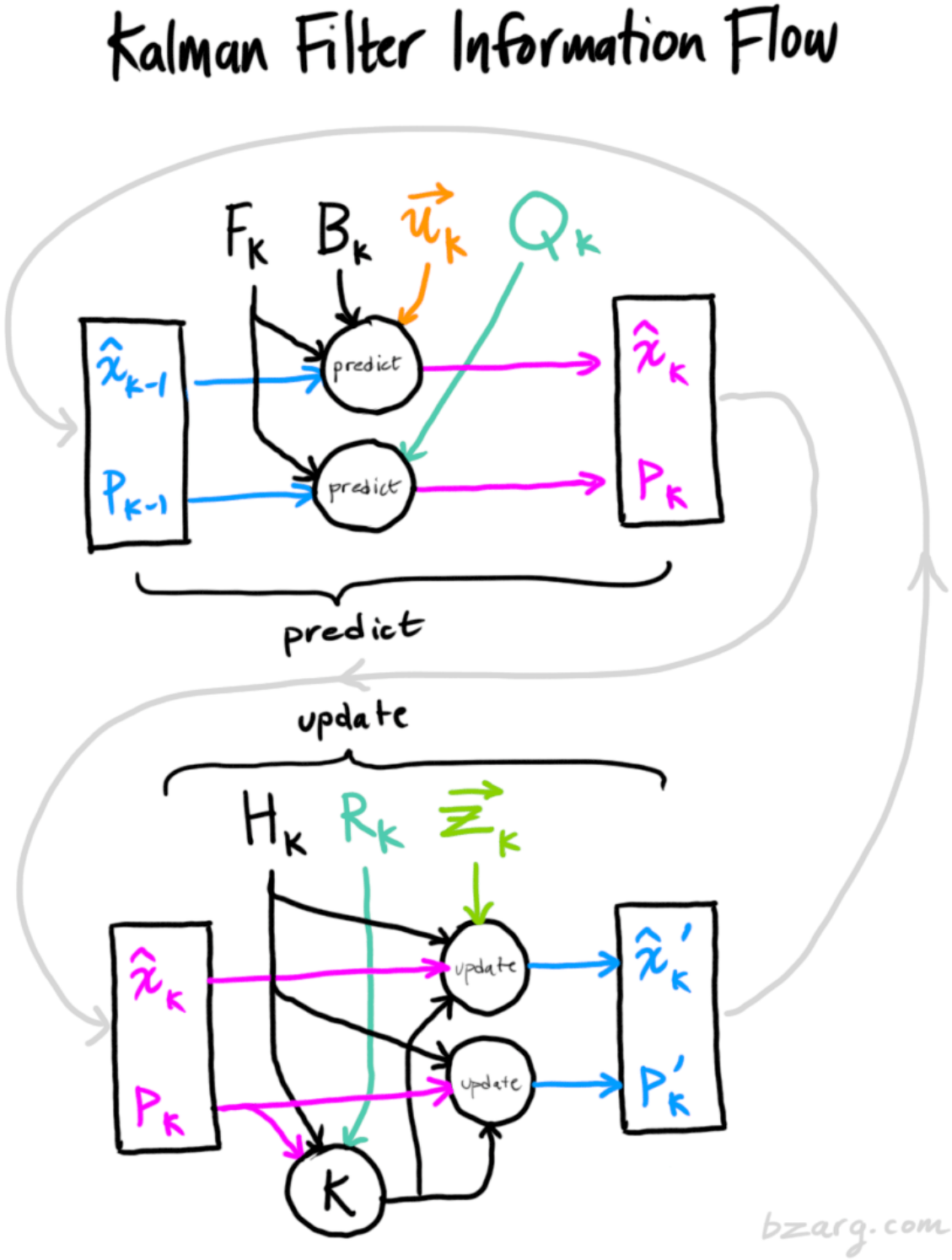
$$x_{\mathrm{PCAwhite},i} = \frac{x_{\mathrm{rot},i}}{\sqrt{\lambda_i + \epsilon}}.$$

Although calculating the Mahalanobis distance between our two example points yields a different value than calculating the Euclidean distance between the PCA Whitened example points, it's still the intuitive way of understanding how correlation needs to be taken into account for distances.

http://mccormickml.com/2014/07/21/mahalanobis-distance/

Kalman Filtering

http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/



What is the probability of data in machine learning

exactly ?

assuming all possible values of an input, i.e. the price of the house, fits into some probability distribution parameterized by θ, which is the probability of the data **p(x | θ)**. for a pixel from an image, it is always chosen from [0, 255] with some probability distribution underneath.

Generalization Error:

1. the error (**expected loss**) for the test data is the probability distribution that the hypothesis hθ is unequal to the true outcome y

$$\varepsilon(h) = P \sim D(h\theta \neq y)$$

2. minimizing the generalization error is equivalent to maximizing the likelihood.

Bias-Variance tradeoff

for vary range of loss function, taking the expectation over the data set, we have

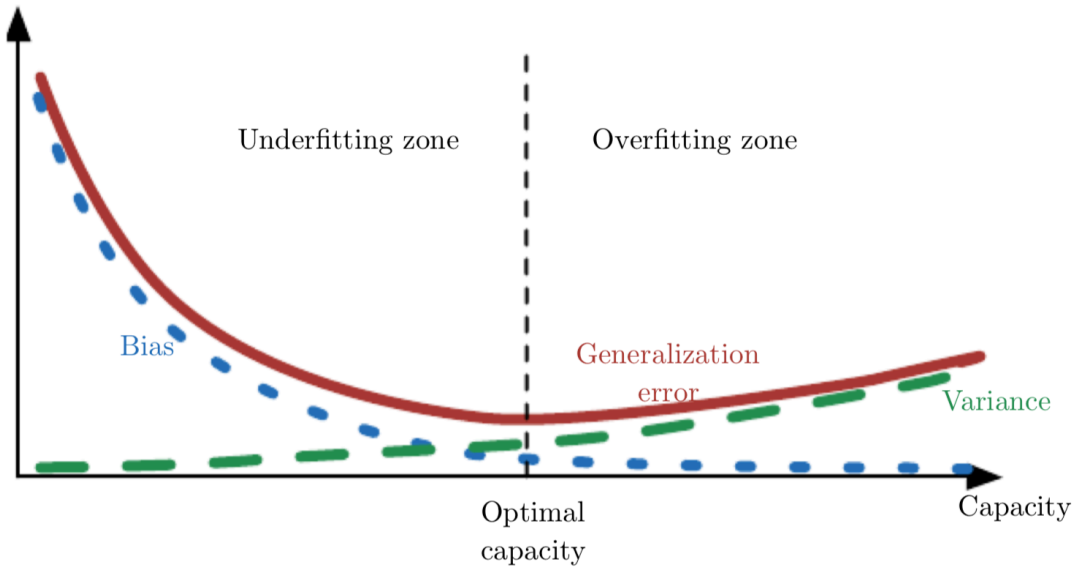
expected loss (Mean Square Error for example)

$$= E[(\theta m - \theta)^2]$$
$$= E[\theta m^2 - 2\theta \bullet \theta m + \theta^2]$$
$$= E[\theta m^2] - 2\theta \bullet E[\theta m] + \theta^2 \; \# \; E[\theta^2] = \theta^2$$
$$= (E[\theta m]^2 - 2\theta \bullet E[\theta m] + \theta^2) + (E[\theta m^2] - E[\theta m]^2)$$
$$= (E[\theta m] - \theta)^2 + (E[\theta m^2] - E[\theta m]^2)$$

where **θm** is the parameter of the sample distribution.

By the definition, Bias is just the distance between the expected empirical distribution and the true distribution, thus,

$$MSE = Bias(\theta m)^2 + Var(\theta m)$$



In the case where generalization error is measured by the MSE (where bias and variance are meaningful components of generalization error), **increasing capacity tends to increase variance and decrease bias**.

see also **Deep Learning Book 5.4.4**

from the perspective of learning theory, we have

ϵ(hypotheses-hat) ≤ (min ϵ(hypotheses)) + 2((1 / 2m) log (2k / γ))

where
ϵ is the generalization error,
m is the training size,
k is the number of hypotheses,
hypotheses-hat is the hypotheses of new example drawing from the training set,
min ϵ(hypotheses) is the bias,
2((1 / 2m) log (2k / δ)) is the variance.

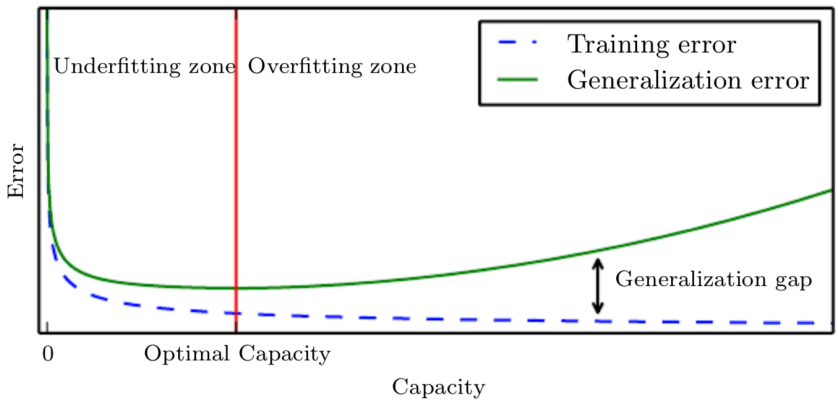


Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the *underfitting regime*. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the *overfitting regime*, where capacity is too large, above the *optimal capacity*.

Markov Chain:

0. Definition

Let
S = { s1, s2, ... ,sn } being the possible states,
p_{ij} = p(sj | si) being the state transition probability,
X1, X2, ..., Xn being the random processes,

then the Markov Chain P is defined as

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1r} \\ p_{21} & p_{22} & \cdots & p_{2r} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ p_{r1} & p_{r2} & \cdots & p_{rr} \end{bmatrix}.$$

Note that $p_{ij} >= 0$, and for all i, we have

$$\sum_{k=1}^{r} p_{ik} = \sum_{k=1}^{r} P(X_{m+1} = k | X_m = i) = 1.$$

This is because, given that we are in state i, the next state must be one of the possible states. Thus, when we sum over all the possible values of k, we should get one. That is, the rows of any state transition matrix must sum to one.

1. Markov Property

P(Xn+1 | Xn, Xn-1, Xn-2, ... , X1) = P(Xn+1 | Xn)

2. State Probability Distribution

$$\pi^{(n)} = \begin{bmatrix} P(X_n = 1) & P(X_n = 2) & \cdots & P(X_n = r) \end{bmatrix}$$

π^n is the probability distribution of **Xn**

Given π^0, we can use the law of total probability to obtain the probability

distribution of **X1**. More specifically, for any j ∈ S, we can write

$$P(X_1 = j) = \sum_{k=1}^{r} P(X_1 = j|X_0 = k)P(X_0 = k)$$
$$= \sum_{k=1}^{r} p_{kj}P(X_0 = k).$$

thus

$$\pi^{(1)} = \pi^{(0)}P,$$

$$\pi^{(2)} = \pi^{(1)}P = \pi^{(0)}P^2.$$

More generally, we have

$$\pi^{(n+1)} = \pi^{(n)}P, \quad \text{for } n = 0, 1, 2, \cdots;$$
$$\pi^{(n)} = \pi^{(0)}P^n, \quad \text{for } n = 0, 1, 2, \cdots.$$

3. n-Step Transition Probabilities

P(Xm + Xn) = P(Xm) P(Xn)

4. Limiting Distribution

The probability distribution π = [π_1, π_2, ..., π_n] is called the **limiting distribution** of the Markov chain Xn if

$$\pi_j = \lim_{n\to\infty} P(X_n = j | X_0 = i)$$

for all i, j ∈ S, we have

$$\sum_{j\in S} \pi_j = 1.$$

5. Stationary Distribution

How do we find the limiting distribution? The trick is to find a ***stationary distribution***.

If π = [π_1, π_2, ..., π_n] is a limiting distribution for a Markov chain, then we have

$$\begin{aligned}\pi &= \lim_{n\to\infty} \pi^{(n)} \\ &= \lim_{n\to\infty} \left[\pi^{(0)} P^n\right].\end{aligned}$$

Similarly, we have

$$
\begin{aligned}
\pi &= \lim_{n\to\infty} \pi^{(n+1)} \\
&= \lim_{n\to\infty} \left[\pi^{(0)} P^{n+1}\right] \\
&= \lim_{n\to\infty} \left[\pi^{(0)} P^{n} P\right] \\
&= \left[\lim_{n\to\infty} \pi^{(0)} P^{n}\right] P \\
&= \pi P.
\end{aligned}
$$

Suppose that ***Xn*** has distribution ***π***. As we saw before, ***πP*** gives the probability distribution of ***Xn*+1**.
If we have ***π* = *πP***, we conclude that ***Xn*** and ***Xn*+1** have the same distribution.
In other words, the chain has reached its ***stationary*** distribution.

From the perspective of derivative we have,

∂P(X) / ∂X
= lim $_{∂X->0}$ (P(X + ∂X) - P(X)) / ∂X
= lim $_{∂X->0}$ (P(X) P(∂X) - P(X)) / ∂X # P(Xm + Xn) = P(Xm) P(Xn)
= P(X) lim $_{∂X->0}$ (P(∂X) - I) / ∂X # I is the identity matrix
= P(X) Q(∂X)

As the derivative **∂P(X) / ∂X** going to be 0, **P(X) would not to be changed with the Time,** which means **P(X)** will going to be the stationary distribution **π**.

6. Ergodic

A Markov chain is **Ergodic** if it has those 2 properties,

1. ***Irreducible***: we must be able to reach any one state from any other state eventually (i.e. the expected number of steps is finite).
2. ***Aperiodic***: the system never returns to the same state with a fixed period (e.g. not returning to state deterministically every 5 steps).

Then it has a unique stationary distribution $\boldsymbol{\pi}$.

7. Detailed Balance

A Markov Chain is said to be reversible (also known as the detailed balance condition)
if there exists a probability distribution $\boldsymbol{\pi}$ that satisfies this condition

$$\pi_i P(X_{n+1} = j | X_n = i) = \pi_j P(X_{n+1} = i | X_n = j)$$

In other words, in the long run, the proportion of times that you transition from state ***i*** to state ***j*** is the same as the proportion of times you transition from state ***j*** to state ***i***.

Monte Carlo Approximation

The basic idea of Monte Carlo Approximation is that, for complex probability distribution, its unable to compute the expected value. Instead we can use Monte Carlo Approximation to approximate the expected value using the average (mean) value of samples that drawing from the distribution.

$$\mu = (1 / N) \sum f(xi), \quad x \text{ ranging from } 1 \ldots N$$

For the Bayesian Inferences, its usually unable to compute the marginal likelihood which is a NP hard problem, so instead another analyzable probability distribution could be introduced to address the problem.

Say we have the marginal likelihood **Z**, and we introduces a gaussian distribution **Q(θ)**

$$Z = \int P(y | \theta) P(\theta) d\theta$$
$$= \int P(y | \theta) P(\theta) Q(\theta) / Q(\theta) d\theta$$

Rewritten the ratio of the likelihood against the introduced distribution as **W(θ)**, we have

$$Z = \int W(\theta)\, Q(\theta)\, d\theta$$

then the marginal likelihood is converted to the Expectation of **W(θ)** with the probability of **Q(θ)** over **θ** based on **the LOTUS law.** Thus with the **Law of Large Number**, we have

$$Z = (1 / N) \sum_i W(\theta^i)$$

as **i** goes infinity.

Markov Chain Monte Carlo Method

Markov Chain Monte Carlo (MCMC) methods are simply a class of algorithms that use Markov Chains to sample from a particular probability distribution (the Monte Carlo part). They work by creating a Markov Chain where the limiting distribution (or stationary distribution) is simply the distribution we want to sample.

http://bjlkeng.github.io/posts/markov-chain-monte-carlo-mcmc-and-the-metropolis-hastings-algorithm/

Softmax function and Temperature:

Softmax function is the generalized function of Logistic Regression. It can be interpreted as the posterior probability p(y = c | x ; θ) that inferred by the likelihood of data x modeled by the exponential family distribution (1), Gaussian, Bernoulli, etc., divided by the marginal likelihood of (1), that is

$$p(y = c \mid x ; \theta) = S(\eta_i) = e^{\eta i} / \sum e^{\eta}$$

where η is the nature parameter $\eta_i = k(\theta^T x)$, .

In terms of thermodynamics statistical, **-η** is called the **Energy function**, and **$\sum e^{\eta}$** is called the **partition function**. There we can introduce another parameter **T** which represents the **Temperature** of the system.

$$S(\eta_i / T) = e^{\eta i / T} / \sum e^{\eta / T}$$

As T goes to infinity, every $e^{\eta i / T}$ goes to 1, then $S(\eta_i / T) \approx 1 / (1 + 1 + \dots 1)$ becomes a uniform distribution which owns the highest entropy,
which implies the higher the Temperature, the more uncertainty the system.

Otherwise as T goes to 0, $e^{\eta i / T}$ goes to infinity, then $S(\eta_i / T) \approx 1$, and every other $e^{\eta j / T}$ goes to 0 since the dinominater should sum to 1,
which implies the lower the Temperature, the more certainty the system.

The form of **$e^{\beta \eta i} / \sum e^{\beta \eta}$** is called the **Boltzmann Distribution**.

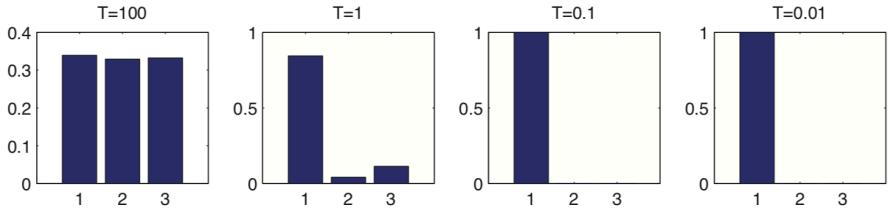
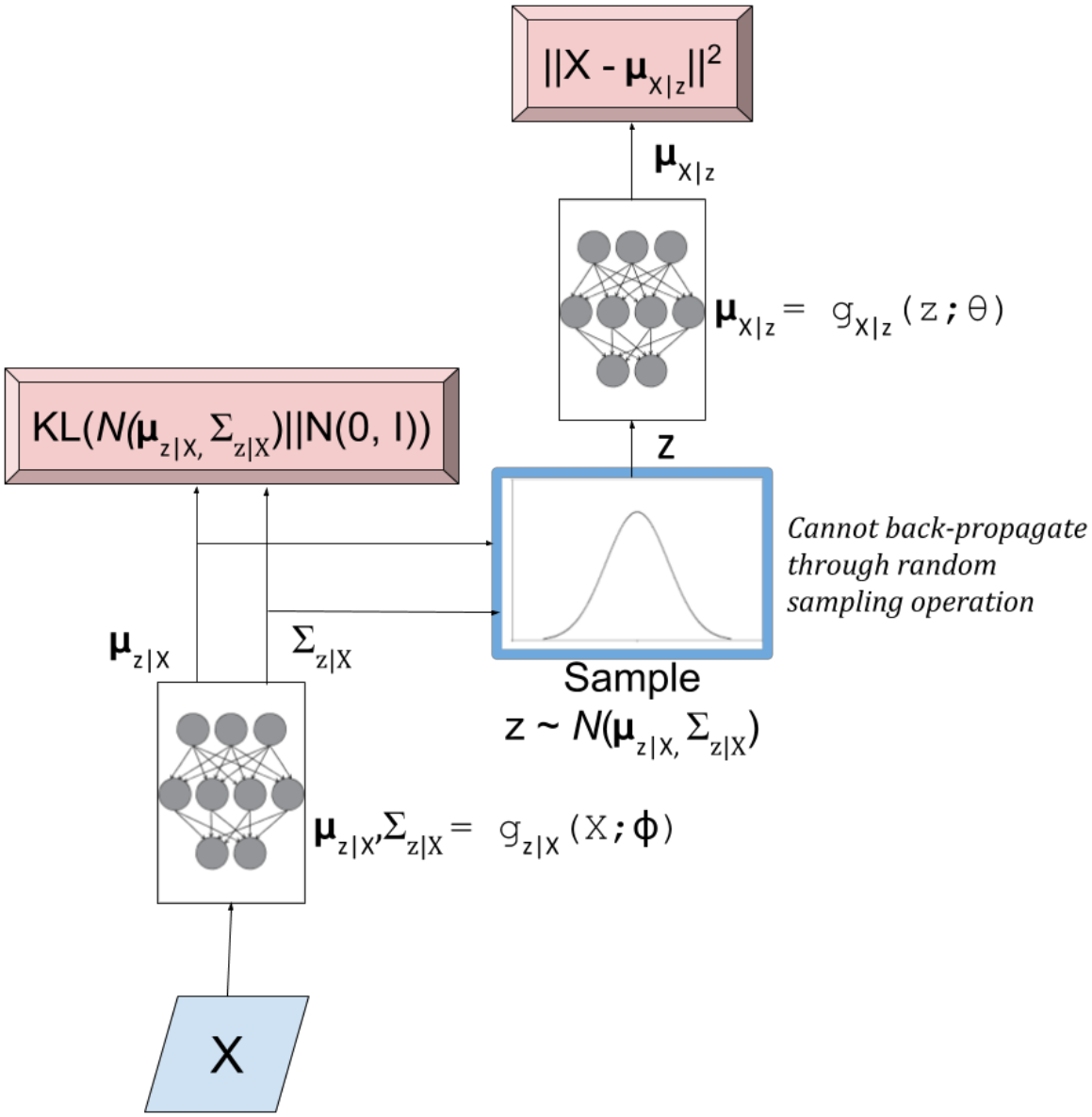


Figure 4.4 Softmax distribution $\mathcal{S}(\boldsymbol{\eta}/T)$, where $\boldsymbol{\eta} = (3, 0, 1)$, at different temperatures T. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is "spiky", with all its mass on the largest element. Figure generated by `softmaxDemo2`.

Support Vector Machine:

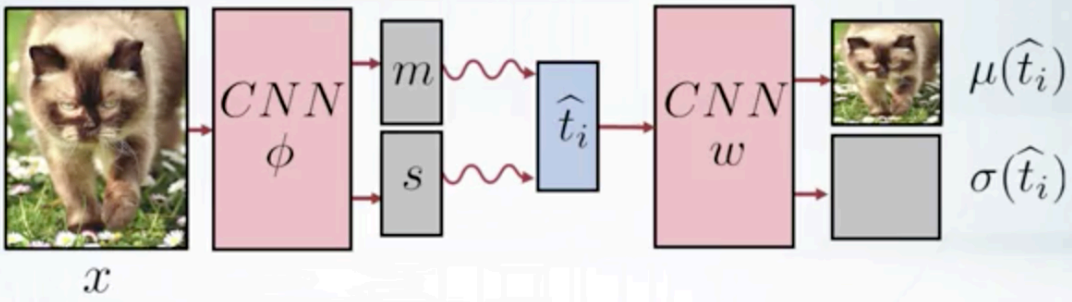
Variational Autoencoder:

http://bjlkeng.github.io/posts/variational-autoencoders/
https://www.coursera.org/learn/bayesian-methods-in-machine-learning/lecture/RC0Op/scaling-variational-em



$$\underset{w,\phi}{\text{maximize}} \quad \sum_i \mathbb{E}_{q_i} \log \frac{p(x_i \mid t_i, w)p(t_i)}{q_i(t_i)}$$

$$\text{subject to} \quad q_i(t_i) = \mathcal{N}(m(x_i, \phi), \text{diag}(s^2(x_i, \phi)))$$



$$\hat{t}_i \sim \mathcal{N}(m(x_i, \phi), \text{diag}(s^2(x_i, \phi))$$

$$\max. \sum_i \mathbb{E}_{q_i} \log \frac{p(x_i \mid t_i, w)p(t_i)}{q_i(t_i)}$$

$$= \sum_i \mathbb{E}_{q_i} \log p(x_i \mid t_i, w) + \underbrace{\mathbb{E}_{q_i} \log \frac{p(t_i)}{q_i(t_i)}}_{-\mathcal{KL}(q_i(t_i) \,\|\, p(t_i))}$$

Xavier Initialization

suppose a linear transformation neural network with inputs **X** and weights **W**,

$$\mathbf{Y = W_1X_1 + W_2X_2 + \ldots + W_nX_n}$$

the variance of the $\mathbf{W_iX_i}$ is

$$\mathbf{Var(W_iX_i) = E[X_i]^2 Var(W_i) + E[W_i]^2 Var(X_i) + Var(W_i) Var(X_i)}$$

if the expectation of all the $\mathbf{X_i}$ are 0, then

$$\mathbf{Var(W_iX_i) = Var(W_i) Var(X_i)}$$

thus,

$$\mathbf{Var(Y) = Var(W_1X_1 + W_2X_2 + ... + W_nX_n) = N \bullet Var(W_i) Var(X_i)}$$

Xavier Initialization is about to make the variance of the input $\mathbf{Var(X_i)}$ and output **Var(Y)** to be the same, then

$$\mathbf{Var(W_i) = 1 / N}$$

http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

Locality Sensitive Hashing

https://blog.csdn.net/orthocenterchocolate/article/details/38943491

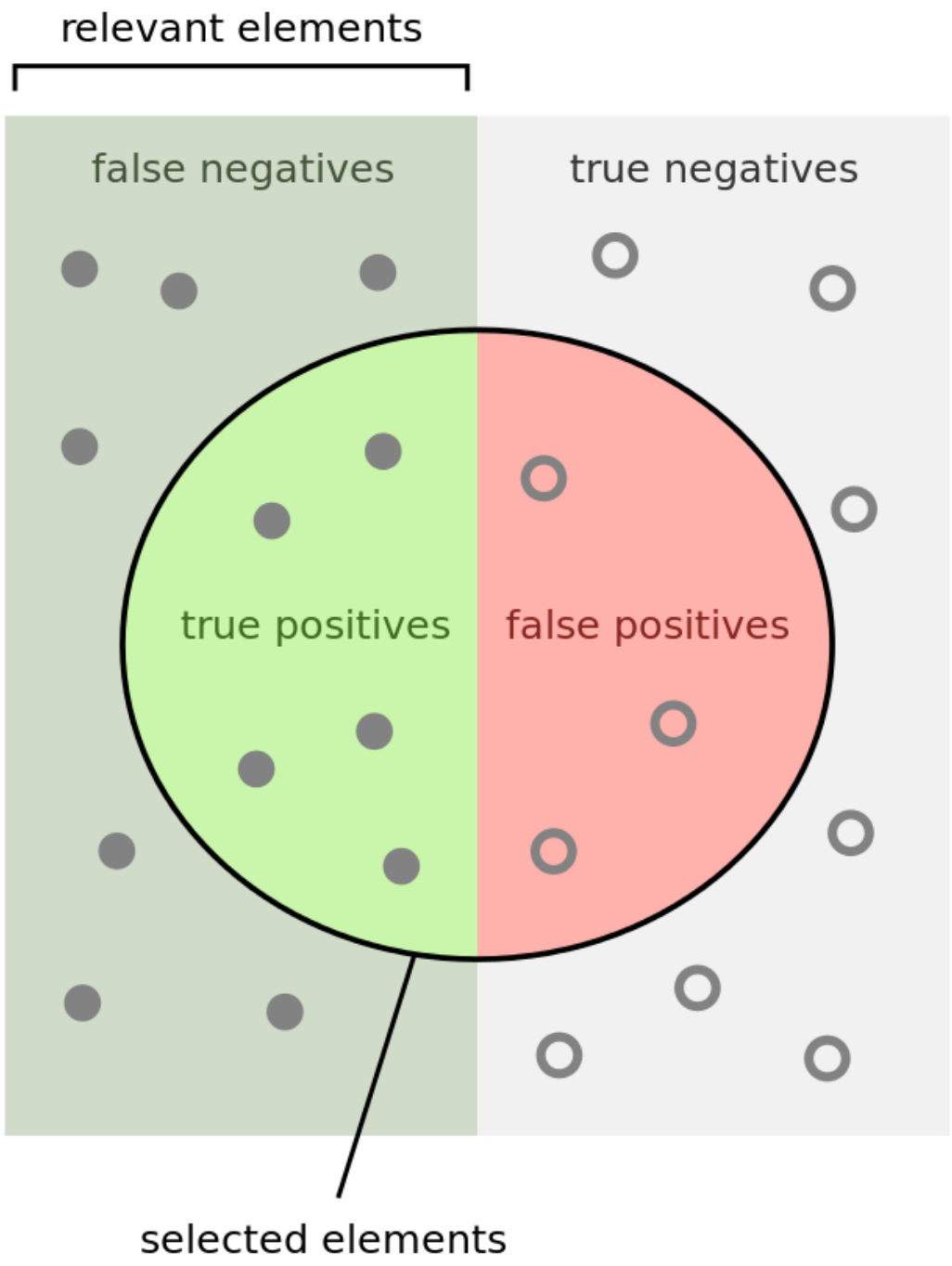
Precision / Recall:

Suppose a computer program for recognizing dogs in photographs identifies eight dogs in a picture containing 12 dogs and some cats. Of the eight dogs identified, five actually are dogs (true positives), while the rest are cats (false positives). The program's precision is 5/8 while its recall is 5/12.

When a search engine returns 30 pages only 20 of which were relevant while failing to return 40 additional relevant pages, its precision is 20/30 = 2/3 while its recall is 20/60 = 1/3.

Precision is "how useful the search results are", and recall is "how complete the results are"

If you want to increase the recall, the precision decreases.

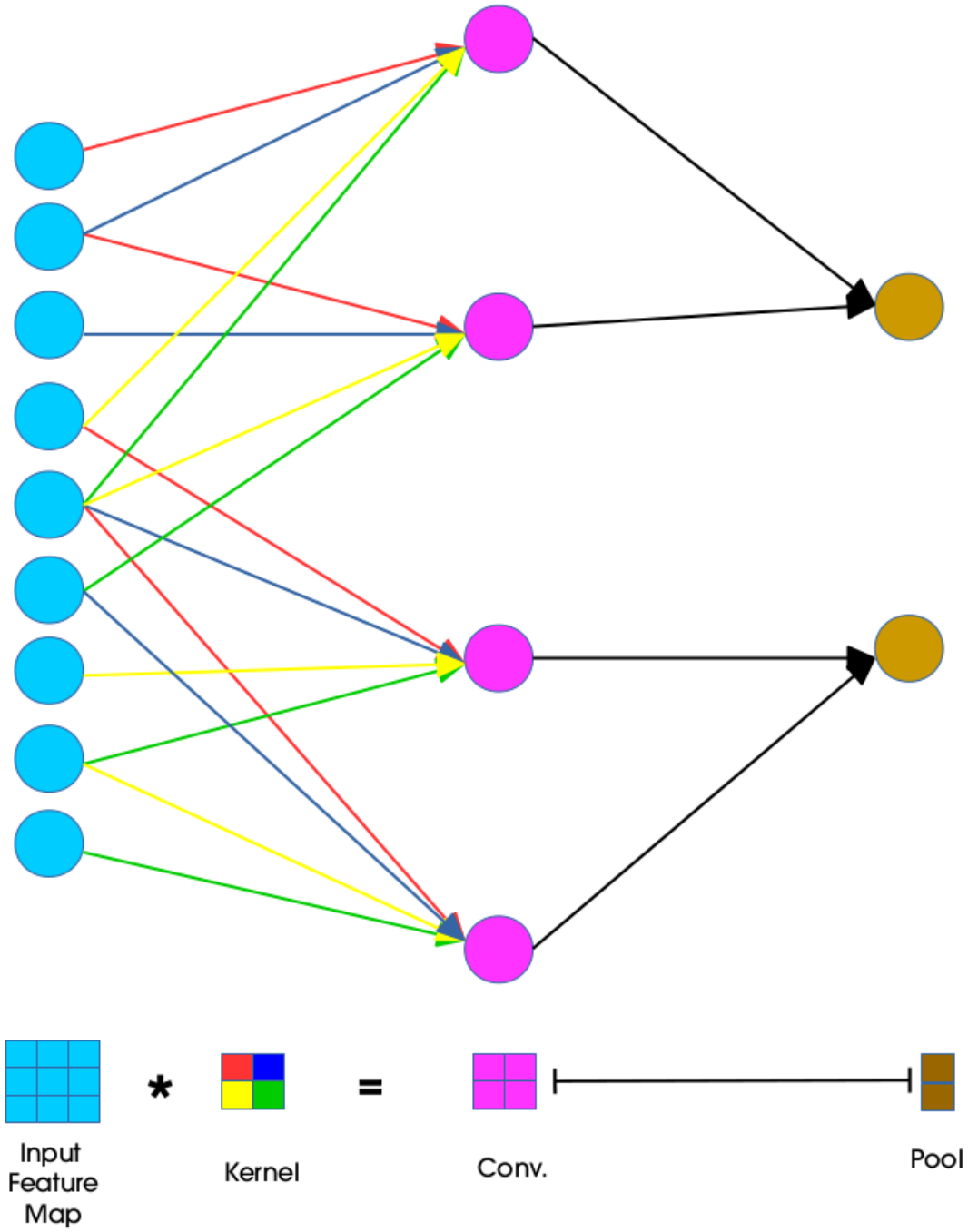


How many selected items are relevant?

How many relevant items are selected?

What is Precision-Recall (PR) curve?
https://www.quora.com/What-is-Precision-Recall-PR-curve

CNN:



When calculating the derivative of $\mathbf{W}_{red}$, the **4 red lines** should be taken into account in contrast with **1 line** for normal DNN.

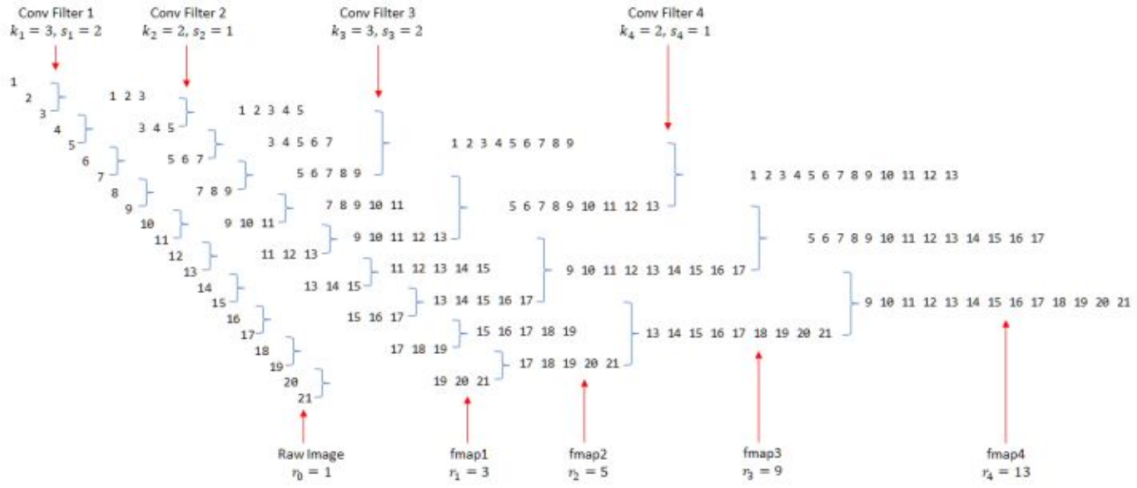
After the max pooling layer, some of the magenta neurons would be dismissed as of the Regularization or Dropout for downsampling or preventing overfitting, say the top and bottom neurons, thus the weights directed to those neurons would be dismissed as well.

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$
$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

for more details, Backpropagation In Convolutional Neural Networks

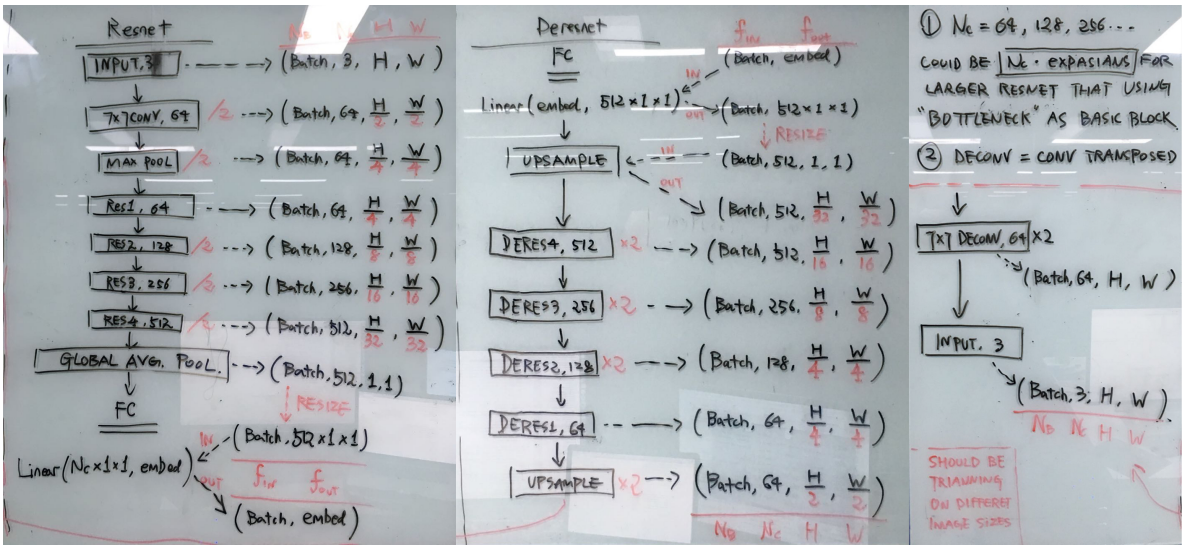
Receptive Field



https://zhuanlan.zhihu.com/p/28492837

Resnet and Reversed Resnet (Deresnet so to speak)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



Winner-Take-All Convolution

W_{ij}^C, h_{ij}^C *denotes* i^{th} *row and* j^{th} *column element of* C^{th} *channel*

*After **WTA** hashing and forward pass, **cell$_1$** reminds*

$$h_{11}^1 = W_{11}^1 X_{11} + W_{12}^1 X_{12} + W_{21}^1 X_{21} + W_{22}^1 X_{22}$$

$$h_{11}^3 = W_{11}^3 X_{11} + W_{12}^3 X_{12} + W_{21}^3 X_{21} + W_{22}^3 X_{22}$$

*After **WTA** hashing and forward pass, **cell$_2$** reminds*

$$h_{12}^1 = W_{11}^1 X_{12} + W_{12}^1 X_{13} + W_{21}^1 X_{22} + W_{22}^1 X_{23}$$

$$h_{12}^3 = W_{11}^3 X_{12} + W_{12}^3 X_{13} + W_{21}^3 X_{22} + W_{22}^3 X_{23}$$

*After **WTA** hashing and forward pass, **cell$_3$** reminds*

$$h_{21}^1 = W_{11}^1 X_{21} + W_{12}^1 X_{22} + W_{21}^1 X_{31} + W_{22}^1 X_{32}$$

*After **WTA** hashing and forward pass, **cell$_4$** reminds*

$$h_{22}^4 = W_{11}^4 X_{22} + W_{12}^4 X_{23} + W_{21}^4 X_{32} + W_{22}^4 X_{33}$$

Backpropogation w.r.t. each $\boldsymbol{W_{ij}^1}$

$$\frac{\partial L}{\partial W_{11}^1} = \frac{\partial L}{\partial h_{11}^1} X_{11} + \frac{\partial L}{\partial h_{12}^1} X_{12} + \frac{\partial L}{\partial h_{21}^1} X_{21}$$

$$\frac{\partial L}{\partial W_{12}^1} = \frac{\partial L}{\partial h_{11}^1} X_{12} + \frac{\partial L}{\partial h_{12}^1} X_{13} + \frac{\partial L}{\partial h_{21}^1} X_{22}$$

$$\frac{\partial L}{\partial W_{21}^1} = \frac{\partial L}{\partial h_{11}^1} X_{21} + \frac{\partial L}{\partial h_{12}^1} X_{22} + \frac{\partial L}{\partial h_{21}^1} X_{31}$$

$$\frac{\partial L}{\partial W_{22}^1} = \frac{\partial L}{\partial h_{11}^1} X_{22} + \frac{\partial L}{\partial h_{12}^1} X_{23} + \frac{\partial L}{\partial h_{21}^1} X_{32}$$

Backpropogation w.r.t. each $\boldsymbol{W_{ij}^3}$

$$\frac{\partial L}{\partial W_{11}^3} = \frac{\partial L}{\partial h_{11}^3} X_{11} + \frac{\partial L}{\partial h_{12}^3} X_{12}$$

$$\frac{\partial L}{\partial W_{12}^3} = \frac{\partial L}{\partial h_{11}^3} X_{12} + \frac{\partial L}{\partial h_{12}^3} X_{13}$$

$$\frac{\partial L}{\partial W_{21}^3} = \frac{\partial L}{\partial h_{11}^3} X_{21} + \frac{\partial L}{\partial h_{12}^3} X_{22}$$

$$\frac{\partial L}{\partial W_{22}^3} = \frac{\partial L}{\partial h_{11}^3} X_{22} + \frac{\partial L}{\partial h_{12}^3} X_{23}$$

Backpropogation w.r.t. $\boldsymbol{W_{ij}^4}$

$$\frac{\partial L}{\partial W_{11}^4} = \frac{\partial L}{\partial h_{22}^4} X_{22}$$

$$\frac{\partial L}{\partial W_{12}^4} = \frac{\partial L}{\partial h_{22}^4} X_{23}$$

$$\frac{\partial L}{\partial W_{21}^4} = \frac{\partial L}{\partial h_{22}^4} X_{32}$$

$$\frac{\partial L}{\partial W_{22}^4} = \frac{\partial L}{\partial h_{22}^4} X_{33}$$

Backpropogation w.r.t. $\boldsymbol{X_{11}}$

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial h_{11}^1} W_{11}^1 + \frac{\partial L}{\partial h_{11}^3} W_{11}^3$$

Backpropogation w.r.t. $\boldsymbol{X_{12}}$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial h_{11}^1} W_{12}^1 + \frac{\partial L}{\partial h_{11}^3} W_{12}^3 + \frac{\partial L}{\partial h_{12}^1} W_{11}^1 + \frac{\partial L}{\partial h_{12}^3} W_{11}^3$$

Backpropogation w.r.t. $\boldsymbol{X_{13}}$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial h_{12}^1} W_{12}^1 + \frac{\partial L}{\partial h_{12}^3} W_{12}^3$$

Backpropogation w.r.t. $\boldsymbol{X_{21}}$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial h_{11}^1} W_{21}^1 + \frac{\partial L}{\partial h_{11}^3} W_{21}^3 + \frac{\partial L}{\partial h_{21}^1} W_{11}^1$$

Backpropogation w.r.t. $\boldsymbol{X_{22}}$

$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial h_{11}^1} W_{22}^1 + \frac{\partial L}{\partial h_{11}^3} W_{22}^3 + \frac{\partial L}{\partial h_{12}^1} W_{21}^1 + \frac{\partial L}{\partial h_{12}^3} W_{21}^3 + \frac{\partial L}{\partial h_{21}^1} W_{12}^1 + \frac{\partial L}{\partial h_{22}^4} W_{11}^4$$

Backpropogation w.r.t. $\boldsymbol{X_{23}}$

$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial h_{12}^1} W_{22}^1 + \frac{\partial L}{\partial h_{12}^3} W_{22}^3 + \frac{\partial L}{\partial h_{22}^4} W_{12}^4$$

Backpropogation w.r.t. $\boldsymbol{X_{31}}$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial h_{21}^1} W_{21}^1$$

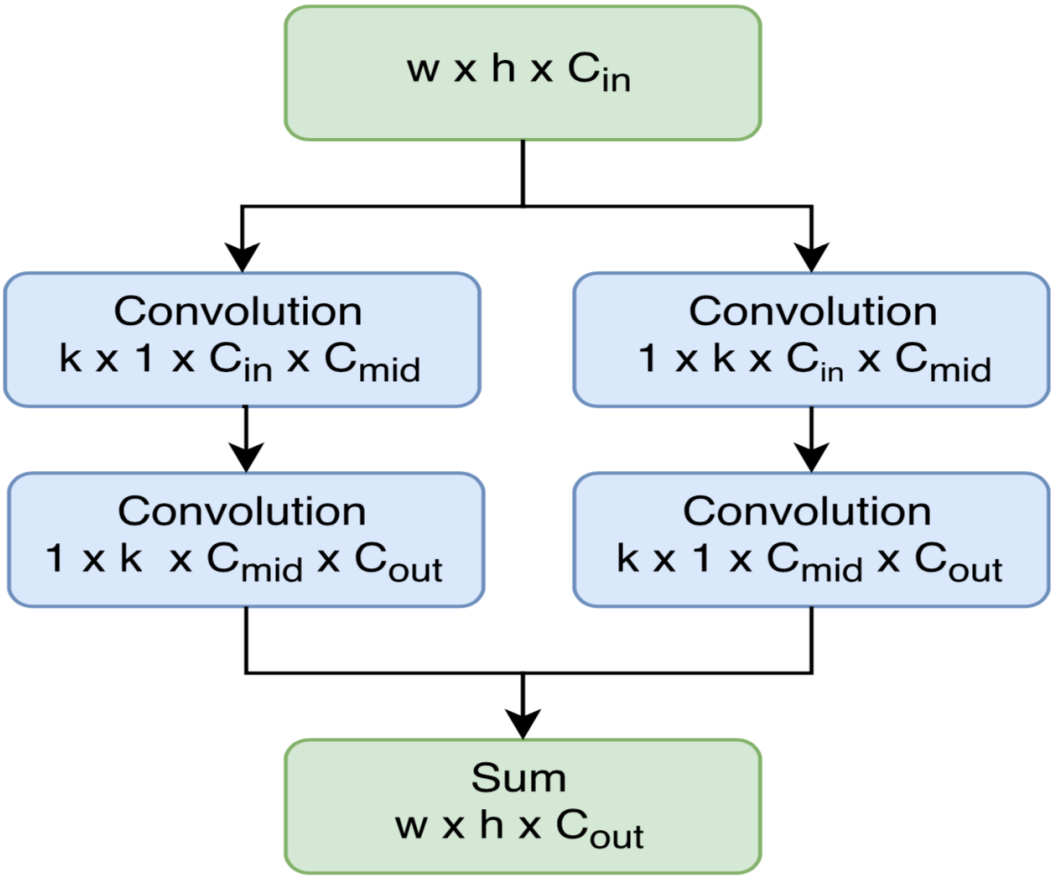
Backpropogation w.r.t. $\boldsymbol{X_{32}}$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial h_{21}^1} W_{22}^1 + \frac{\partial L}{\partial h_{22}^4} W_{21}^4$$

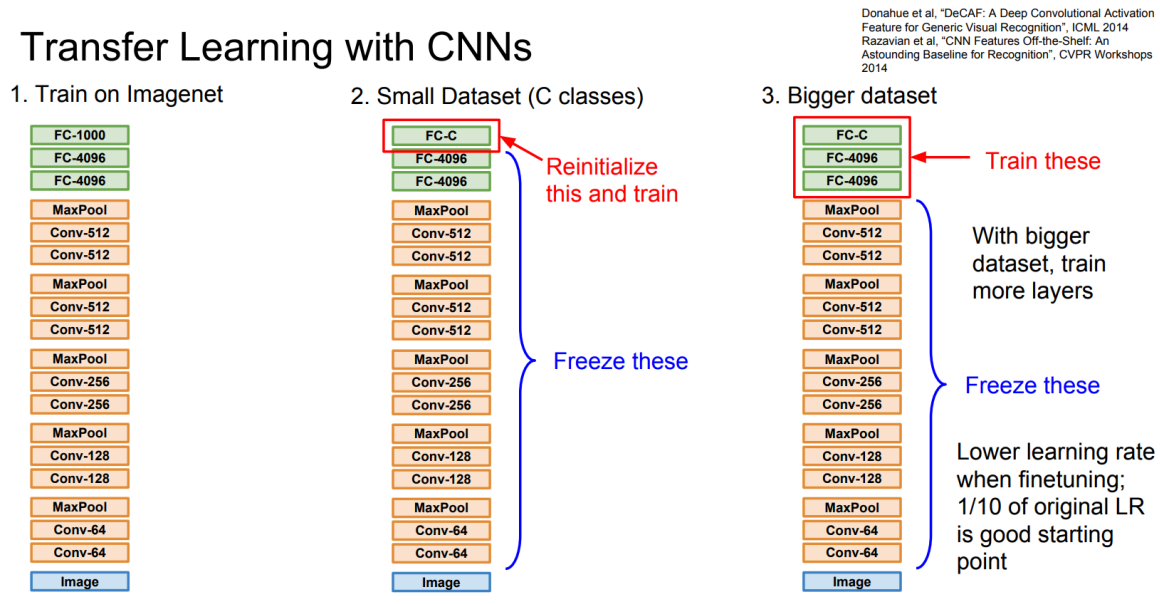
Backpropogation w.r.t. $\boldsymbol{X_{33}}$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial h_{22}^4} W_{22}^4$$

Large Separable Convolution



Transfer Learning



Questions:

1. What is the Universal Approximation Theorem? A: http://neuralnetworksanddeeplearning.com/chap4.html
2. what the relationship between Naive Bayesian and Logistic Regression?
3. parametric vs non-parametric. parametric: like linear regression with weight parameters w; non-parametric: like nearest neighbor regression.
4. We run it through l2- normalization, then an affine transformation step, then another normalization step for mapping the result to a normalized feature space. For the affine transform we consider a learned fully-connected layer. **[Paper]** *Scalable Object Detection for Stylized Objects*
5. The PCA projection can be implemented with a shifting and a fully connected (FC) layer **[Paper]** *Deep Image Retrieval - Learning global representations for image search*
6. *DATA - VECTOR - Probability*
7. relationship between svm and svd
8. reconstruct yolo with attention model

Paper Notes:

1. [Leveraging category-level labels for instance-level image retrieval] finetuning the pretrained models for retrieval can bring a significant improvement
2. [Learning fine-grained image similarity with deep ranking] One can distinguish offline hard negative mining, where a second model is trained to identify hard cases on the whole dataset.
3. [Facenet: A unified embedding for face recognition and clustering] suggest various methods for picking the negative samples.

Miscellaneous:

Big O Notation:

1. Definition: How **TIME** scales respect to **INPUT**.

For example, a for-loop takes **O(N)** amount of time, a for-loop inside the identical loop takes **O(N^2)** amount of time, etc.

2. Constant doesn't matter

O(2N) ≈ O(N)

as **N** goes infinity.

3. Dropping the non-dominate term

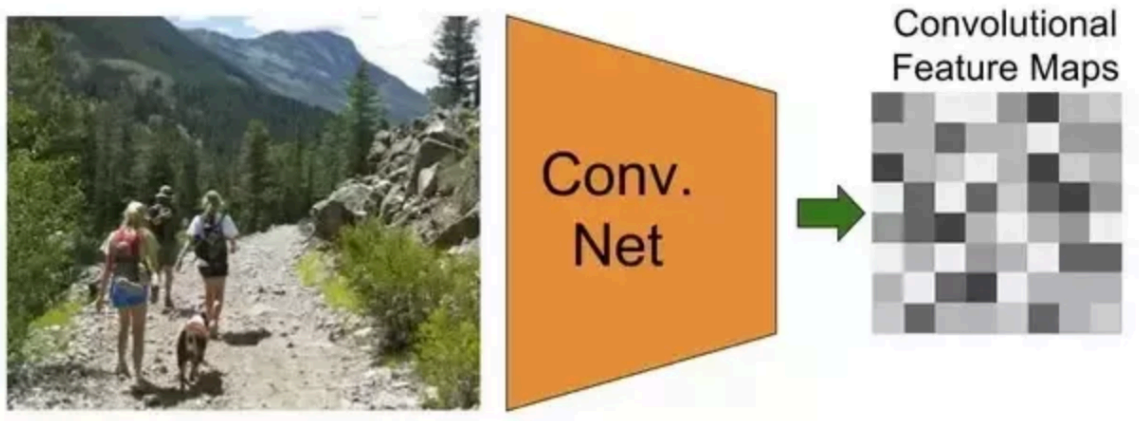
O(N) + O(N^2) = O(N + N^2) ≈ O(N^2)

since **O(N)** is non-dominate compared to **O(N^2)**

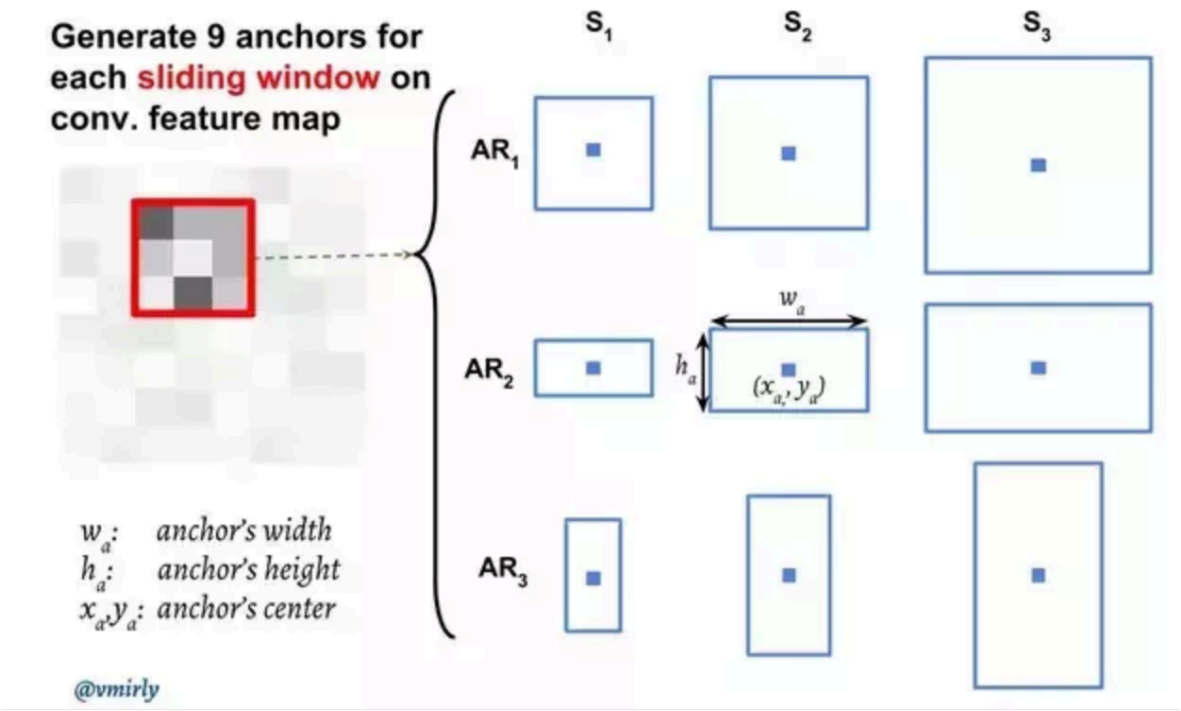
ROI Pooling

How does the region proposal network (RPN) in Faster R-CNN work?

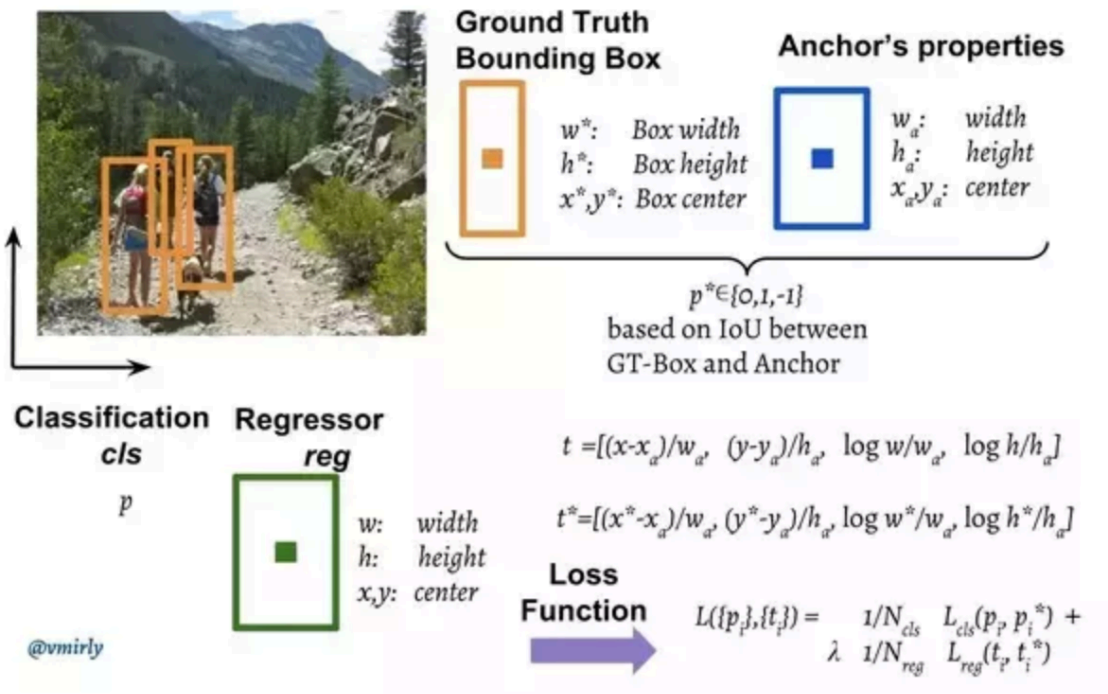
1. In the first step, the input image goes through a convolution network which will output a set of convlutional feature maps on the last convolutional layer:



2. Then a sliding window is run spatially on these feature maps. The size of sliding window is $n \times n$ (here 3×3). For each sliding window, a set of 9 anchors are generated which all have the same center (x_a, y_a) but with 3 different aspect ratios and 3 different scales as shown below. Note that all these coordinates are computed with respect to the original image.



3. Finally, the 3×3 spatial features extracted from those convolution feature maps (shown above within **red box**) are fed to a smaller network which has two tasks: classification (cls) and regression (reg). The output of regressor determines a a predicted bounding-box (x, y, w, h), The output of classification sub-network is a probability p indicating whether the the predicted box contains an object (1) or it is from background (0 for no object).



How to train Fast-RCNN ? First RPN is trained and regions obtained are used to train detection network. Then RPN is retrained but this time with the convolution part initialized with weights from convolution part of detection network.

Different between **latent variables** of probabilistic models and **hidden unit** of neural network. https://www.quora.com/How-are-latent-factor-neural-networks-trained

progressive resizing

Train on smaller images at the start of training, and gradually increase image

size as you train further.

It makes intuitive sense that you don't need large images to learn the general sense of what cats and dogs look like (for instance), but later on when you're trying to learn the difference between every breed of dog, you'll often need larger images.

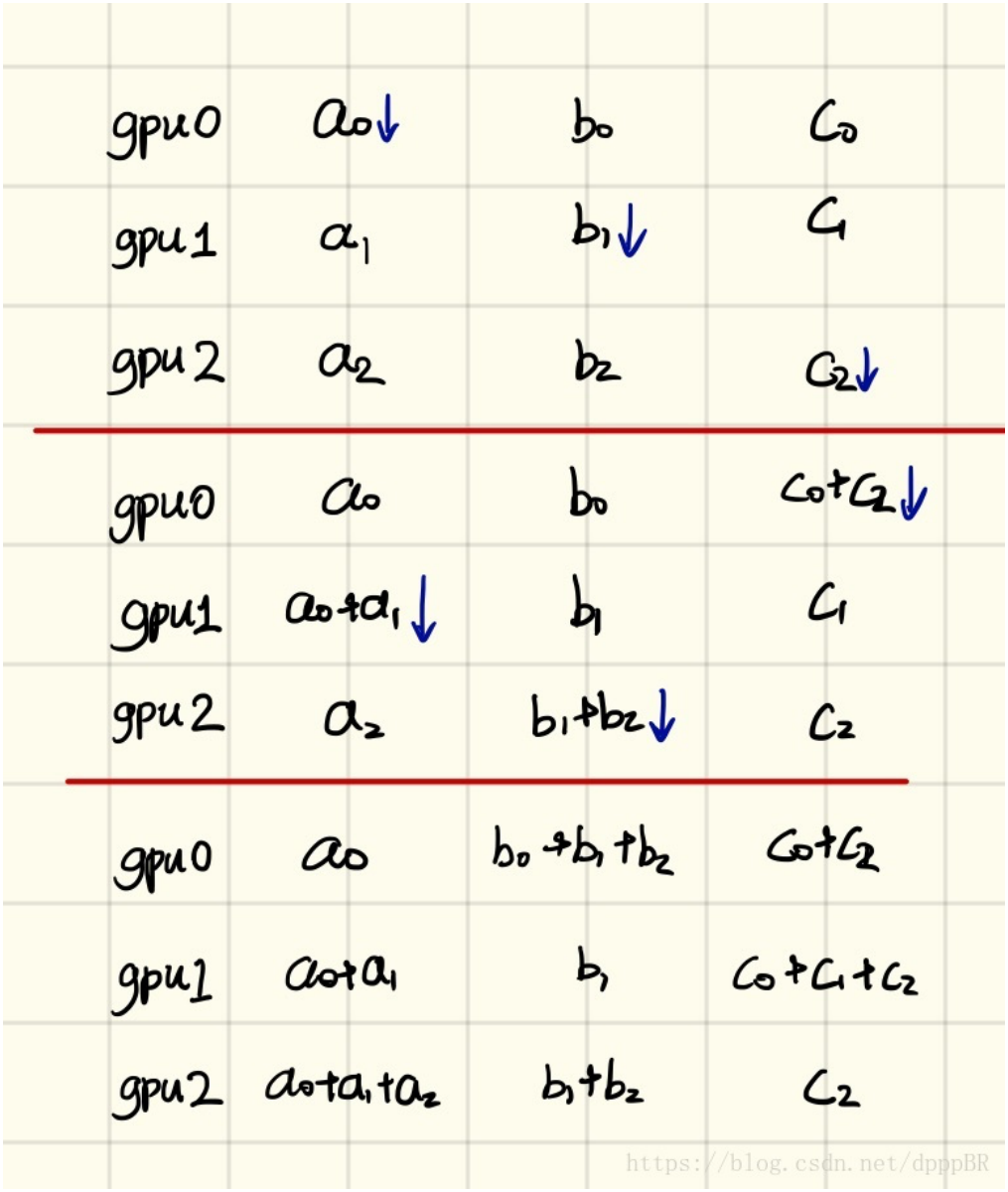
By using progressive resizing we were both able to make the initial epochs much faster than usual (using 128x128 images instead of the usual 224x224), but also make the final epochs more accurate (using 288x288 images for even higher accuracy). But performance was only half of the reason for this success.

The other impact is better generalization performance. By showing the network a wider variety of image sizes, it helps it to avoid over-fitting.

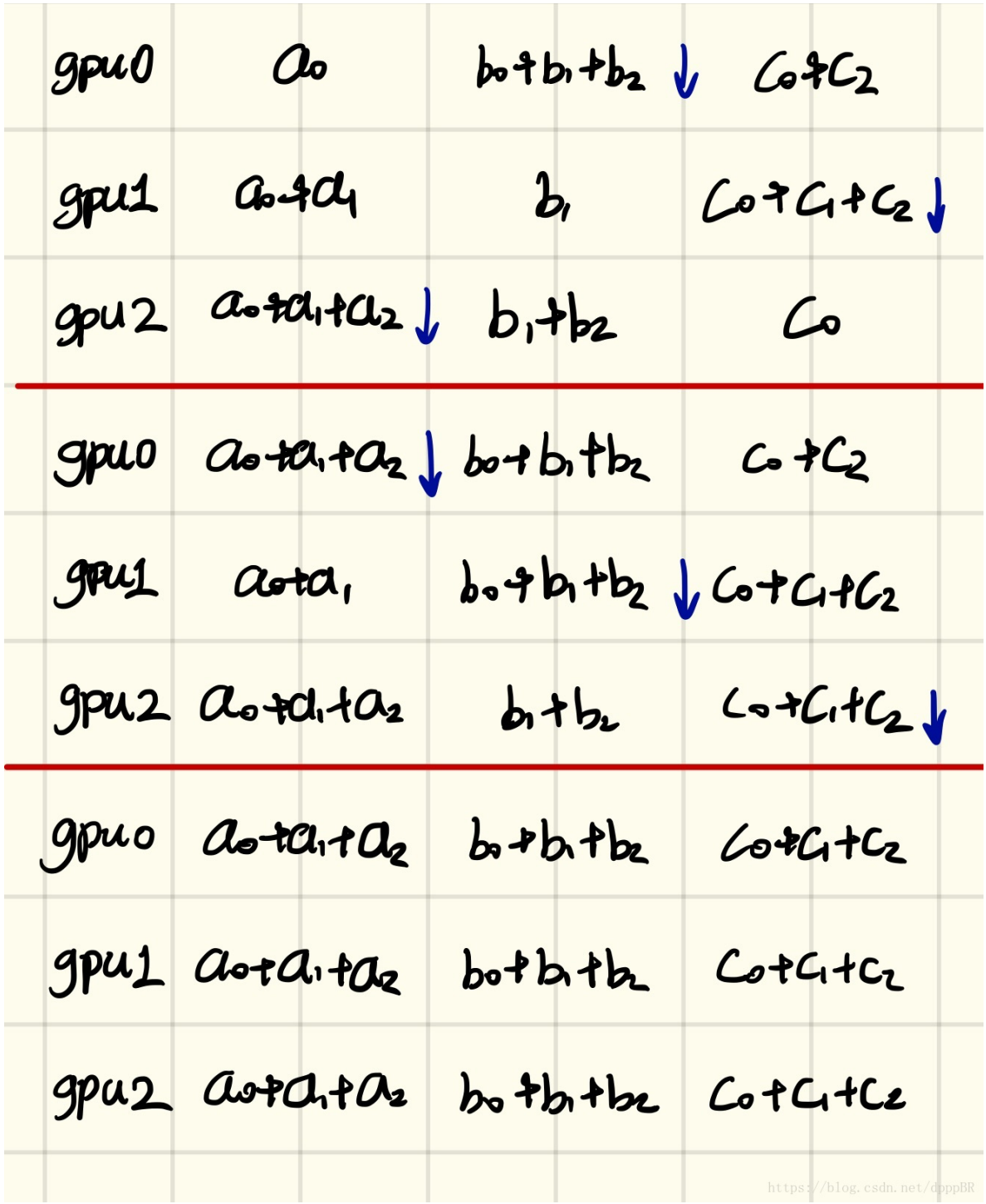
http://www.fast.ai/2018/04/30/dawnbench-fastai/

ring-allreduce

1. scatter-reduce


gpu0 a_0↓ b_0 c_0
gpu1 a_1 b_1↓ c_1
gpu2 a_2 b_2 c_2↓
gpu0 a_0 b_0 c_0+c_2↓
gpu1 a_0+a_1↓ b_1 c_1
gpu2 a_2 b_1+b_2↓ c_2
gpu0 a_0 $b_0+b_1+b_2$ c_0+c_2
gpu1 a_0+a_1 b_1 $c_0+c_1+c_2$
gpu2 $a_0+a_1+a_2$ b_1+b_2 c_2
https://blog.csdn.net/dpppBR

2. all-gather



https://blog.csdn.net/dpppBR/article/details/80445569

DetNet training strategy

Backbone training:

pretrained hyper-parameters and training settings provided by [Aggregated residual transformations for deep neural networks]:

- Imagenet
- 8 Pascal TITAN XP GPUs
- 256 total batch size

Detector training:

- training setting: https://github.com/facebookresearch/detectron
- COCO
- 8 Pascal TITAN XP GPUs
- Each mini-batch has 2 images, so the effective batch-size is 16
- synchronized SGD
- multi-gnu synchronized BN during training as in [Megdet: A large mini-batch object detector]
- weight decay: 0.0001; momentum: 0.9
- shorter edge: 800px; longer edge: 1333px
- pad the images within mini-batch to the same size by filling zeros into the right-bottom of the image
- 2x training setting used in detection
- learning rate:
  - 0 - 500: 0.02 * 0.3
  - 500 - 120k: 0.02
  - 120k - 160k: 0.002
  - 160k - 180k: 0.0002

Light-Head R-CNN training strategy

- 8 Pascal TITAN XP GPUs
- learning rate: 0.01 for 1st 1.5M iterations ; 0.001 for later 0.5M iterations (passing 1 image is regarded as 1 iteration)
- synchronized SGD
- weight decay: 1e-4
- momentum: 0.9
- batch-size: 2 / GPU
- training: 2000 RoIs / image
- testing: 1000 RoIs / Image
- OHEM (online hard example mining): 2000 RoIs for training; 256 samples for backpropagation
- RPN on stage 4
- atrous in stage 5
- fix parameters of stage1 and stage2
- fix batch normalization

- horizontal image flipping augmentation
- resize the shorter edge of image to 800 pixels, and restrict the max size of the longer edge to 1200
- pad images within mini-batch to the same size by filling zeros into the right-bottom of the image
- scale jitter approach
- classification loss and regression loss should be balanced
- pool with alignment
- NMS replace original 0.3 threshold to 0.5. It improves 0.6 of mmAP by improving the recall rate especially for the crowd cases.
- multi scale train: during training, randomly sample the scale from {600, 700, 800, 900, 1000} pixels, then resize the shorter edge of the image into the sampled scale. The max edge of the image is constrained within 1400 pixels because the shorter edge may reach to 1000 pixels. Multi-scale training brings newly 1.0 improvement on mmAP.

Receptive Field Block Net training strategy (COCO)

- batch size: 32
- warmup: progressively increases the learning rate from 1e-6 to 2e-3 at the first 5 epochs, then decrease it after 80 and 100 epochs by factor of 10, and end up at 120
- image size: 512