Introduction.

Machine learning- Capability of a system to acquire patterns from raw data

Representation learning- learning representation of a required object from images. This algorithm can discover good set of features
Eg: Autoencoders

factors of variation - concepts that help us make sense of rich variability of data

Many of these factors are understandable only with human level knowledge. This is where deep learning comes in.

- learning right representation of data
- depth enables computer to learn a multistep program.

It represents world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones

Linear Algebra

Tensors:

Array with more than 2 axes

vector, matrix addition

C = A + b

A -> Matrix

b -> vector

vector b is added to each row.
This is called broadcasting.

Hadamard product
element-wise multiplication of 2 matrices

span

~~Set of vectors~~

The span of a set of vectors is the set of all points obtainable by linear combination of original vectors.

Ax = b is has a solution if b is in span of columns of A or column space / range of A

Square matrix with linearly dependant columns is called singular. Eigen values are zero

Norm of a vector measures the distance from origin

L^2 norm - Euclidean Norm.

Euclidian distance from the point to the origin given by x^Tx

squared L^2 norm is more convinient to use.

But sometimes it may be undesirable because it increase very slowly near origin.

L^1 norm grows at same rate everywhere.

It is used in ML when diff b/w zero & non-zero elements are very important.

$= \sum |x_i|$

L^∞ norm, max norm

Simplifies to the absolute value of the element with the largest magnitude in the vector

$= \max|x_i|$

Frobenius norm

used to measure size of matrix.

$= \sqrt{\sum A^2_{i,j}}$

diag(v) - square diagonal matrix

using this is computationally efficient.

* Rectangular diag matrix exist, they need not have inverse but can still be multiplied cheaply.

orthogonal matrix

Square matrix whose rows are mutually orthonormal and whose colocumns are mutually orthonormal.

$$A^TA = AA^T = I$$

$$A^{-1} = A^T$$

It is easy to find their inverses.

Eigen decomposition

decompose a matrix to a set of eigen vectors, eigen values.

$$Av = \lambda v$$

λ - Eigen value
v - an Eigen vector

$$A = V \text{diag}(\lambda) V^{-1}$$

every real symmetric matrix can be decomposed into an expression using only real v. & λ.

$$A = Q \Lambda Q^T$$

Eigen decompositions are unique only if all Eigen values are unique.

matrix with all +ve λ → positive definite.

matrix with +ve | 0 λ → positive semi-definite

matrix with -ve λ → negative definite

matrix with -ve | 0 → negative semi-definite

Singular Value Decomposition

Every real matrix has a SVD

$A = UDV^T$ $\quad U, V \rightarrow$ orthogonal

elements in $D \rightarrow$ singular value

Columns of $U \rightarrow$ left singular vectors

columns of $V \rightarrow$ Right singular vectors

Moore - Penrose Pseudo inverse

$$Ax = y$$
$$x = By$$

If A is tall than it is wide, mostly no soln.

If A is wide than it is tall, could b multiple soln.

Pseudo inverse of A,

$$A^+ = \lim_{\alpha \to 0} (A^T A + \alpha I)^{-1} A^T$$

or

$$A^+ = V D^+ U^T \qquad U, D, V \rightarrow \text{SVD of } A$$

D^+ – reciprocal of nonzero of D, then transpose

So when A has more columns than rows, MPP is applicable to find inverse.

Trace operator

Sum of all diagonal entries

used for alternate Frobenius norm

$$= \sqrt{Tr(AA^T)}$$

$Tr(A) = Tr(A^T)$

$Tr(ABC) = Tr(CAB) = Tr(BCA)$

Determinant

product of all eigen values

can be thought as a measure of how much multiplication by which matrix expands or contracts space.

PCA can be derived using simple LA.

Probability and Information theory.

probability allows us to make uncertain statements and to reason in presence of uncertainty.

Information theory allows us to quantify the amount of uncertainty in a PD

Frequentist probability

related to rate at which events occur

Bayesian probability

related to qualitative level of certainty.

Marginal probability

PD over a subset

Conditional probability

Probability of an event given that some other event has happened.

$$P(y|x) = \frac{P(y,x)}{P(x)}$$

Expectation

$$E = \sum P(x) f(x)$$

$$= \int P(x) f(x) \, dx$$

Variance

$$E[\ f(x) - E[f(x)]^2]$$

S.D

$$SD : \sqrt{V}$$

Covariance

how 2 values are linearly related to each other.

Bernoulli Distribution.

Distribution over a single binary random variable

Gaussian Distribution

$$\sqrt{\frac{1}{2\pi\sigma^2}}\ e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

$\beta = \frac{1}{\sigma^2}$: precision

Central limit theorem

sum of many independent random is approximately normally distributed.

normal distribution needs least amount of prior knowledge of the data.

Exponential distribution

sharp point at $x = 0$

$$p(x; \lambda) = \lambda e^{-\lambda x}$$

Laplace distribution

Sharp peak at an arbitrary point μ.

$$L(x; \mu, \gamma) = \frac{1}{2\gamma} e^{\left(\frac{-|x-\mu|}{\gamma}\right)}$$

Dirac delta function

to specify all the mass in a PDF around a single point.

$$p(x) = \delta(x - \mu)$$

Limit point of a series of fns that put less & less mass on all points other than zero.

empirical distribution

$$p(x) = \frac{1}{m} \sum \delta(x - x^{(i)})$$

It is the probability density which maximises likelihood of training data. One dirac component for each x^i.

Mixtures of distributions

On each trial, choice of which component distribution should generate the sample is determined by sampling a component identity from a multinoulli distribution.

latent variable

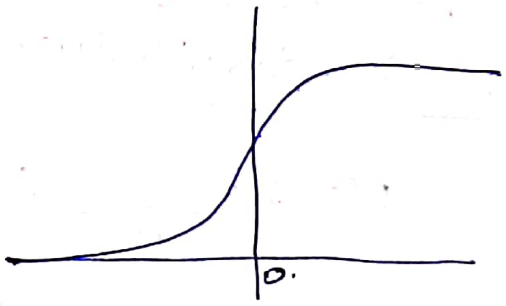
Random variable which can't be observed directly. They may be related to x through the joint distribution.

Gaussian Mixture Model

Each component are gaussians.
separately parameterized mean & covariance.
The parameter also specify the prior probability given to each component i.
This model is universal approximator of densities.

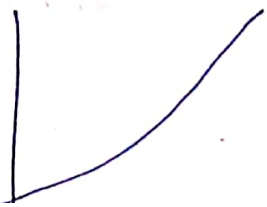
logistic sigmoid

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



Soft plus function

$$S(x) = \log(1+e^{x})$$



Bayes rule

$$P(x|y) = \frac{P(x)\,P(y|x)}{P(y)}$$

computing $P(x|y)$ when we know $P(y|x)$.

we can find $P(y) = \sum P(y|x)\,P(x)$

Some terms from Measure theory.

measure zero

a set of measure zero occupies no volume in the space we are measuring.

Eg: line has zero measure zero.
polygon has positive measure.

almost everywhere

holds throughout all space except on a set of measure zero. Because the exceptions occupy a negligible amount of space, they can be safely ignored for many applications.

Information theory

Quantifies how much information is present in a signal.

An unlikely event has occured is more informative than learning a likely event.

Also independent events should have additive info.

Self information

$$= -\log P(x)$$ unit = nat.

1 nat : amount of info gained by observing an event of probability $\frac{1}{e}$

base-2 unit → bit / shannons.

Shannon Entropy

amount of uncertainty over an entire PD

$$H(P) = -\mathbb{E}[\log(P(x))]$$

when x is continous, it is called Differential Entropy.

Kullback-Leibler Divergence

two separate PD over same random variable.

$$D_{KL}(P||Q) = E_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right]$$

often conceptualised as measuring some sort of difference b/w 2 distributions.

It is not true distance because it is not symmetric.

Cross. Entropy

$$= H(P) + D_{KL}(P||Q)$$

$$= -E_{x \sim P} \log Q(x)$$

Structured probabilistic Models.

Instead of a single fn to represent a PD, we can split a PD into many factors that we multiply together.

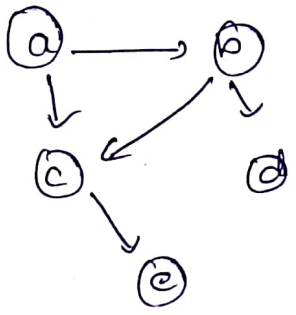
Eg: $P(a,b,c) = P(a)\,P(b|a)\,P(c|b)$

This greatly reduces no of parameters needed.

factorisation of a PD with a graph -> SPM or graphical model.

Directed Models

uses graphs with directed edges & they represent factorisations into conditional PD.



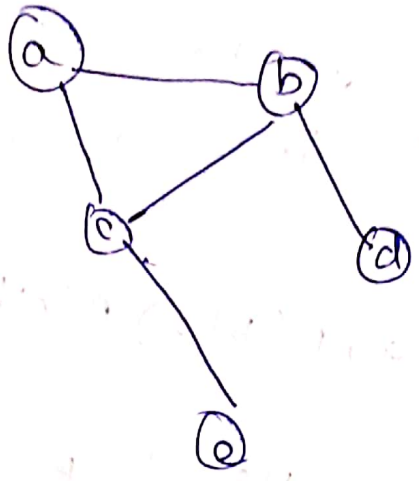
$P(a,b,c,d,e) =$

$P(a).P(b|a).P(d|b)P(e|c)$

$.P(c|a)$

Undirected Models

uses graphs with undirected edges
they represent factorisations into
set of functions.



$$P(a,b,c,d,e) = \frac{1}{Z}\phi^{(1)}(a,b,c)\,\phi^{(2)}(b,d)\,\phi^{(3)}(c$$

underflow

numbers near zero are rounded to zero

overflow

numbers with large magnitude approximated to ∞ or $-\infty$.

overflow & underflow affects softmax function.

$$\text{softmax}(x)_i = \frac{e^{(x)_i}}{\sum e^{(x)_j}}$$

when $\sum e^{(x)_j} \to 0$, due to underflow, it becomes not defined.

This can be resolved by finding

$$\text{softmax}(z) = z, \quad z = x - \max_i x_i$$

This doesn't change value of the function but eliminates the difficulties faced.

But underflow can still happen in numerator.

conditioning

Refers to how rapidly a fn changes with respect to small changes in its inputs.

Poorly conditioned matrices amplify pre existing errors.

Optimisation

minimising or maximising a factor some function $f(x)$ by altering x.

gradient based optimisation

when $f'(x) = 0$, derivative provide no info on which direction to move. These are called critical points or stationary points.

directional derivative

The DD in direction u is the slope of the function f in direction u.

Jacobian matrix

matrix containing partial derivatives of all a fn whose i/p, o/p are both vectors.

second derivative can be thought of measuring curvature.

Hessian matrix

when fn has multiple i/p dimensions, there are many second derivatives. These in a matrix is called Hessian matrix.

Hessian is the Jacobian of the gradient.

Since Hessian is symmetric, Eigen values and Eigen vectors can be found for them.

The second derivative tells how we can expect a gradient descent step to perform.

using second order Taylor series approximation

$$f(x) \approx f(x^0) + (x - x^0)^T g + \frac{1}{2}(x - x^0)^T H (x - x^0)$$

with learning rate ϵ

$$x^{(0)} = \epsilon g$$

$$f(x^{(0)} - \epsilon g) \approx f(x^0) - \epsilon g^T g + \frac{1}{2}\epsilon^2 g^T H g$$

- $f(x^0)$ ↓ original value
- $-\epsilon g^T g$ ↓ expected improvement due to the slope
- $\frac{1}{2}\epsilon^2 g^T H g$ ← correction to be applied due to curvature

optimisation algorithms that use only gradient is called first order optimisation algorithms.

which use Hessian matrix, such as Newton's method are called 2nd order opti algo.

Lipschitz continous function is a fn f whose rate of change is bound by Lipschitz constant L:

convex optimisation

These are applicable for convex fns, Hessians is positive semidefinite everywhere

These fns lack saddle point & all local minima are global

Constrained Optimisation

maximal & minimal values of x in some set. Points of x which lie within S is called feasible points.

Eg: $\|x\| \leq 1$

gradient descent can be modified taking the constraint into account

Karush-Kuhn-Tucker (KKT)

new function called generalized Lagrangian or general Lagrange function.

define S in terms of m functions g^i and n functions h^j

$$S = \{x \mid \forall i, g^i(x) = 0 \text{ \& } \forall j, h^j(x) \leq 0\}$$

(g^i: equality constraints; h^j: inequality constraints)

$$L(x, \lambda, \alpha) = f(x) + \sum \lambda_i g^{(i)}(x) + \sum \alpha_j h^j(x)$$

λ, α → KKT multipliers

$\min_{x} \max_{\lambda} \max_{\alpha > 0} L(x, \lambda, \alpha)$ has same optimal objective for value & set of optimal points x as $\min_{x \in S} f(x)$

Machine Learning Basics

Transcription

transcribing info into textual form from some unstructured data

Eg. Extract text from image

generalisation

The ability to perform well on previously unobserved inputs.

The factors determining how well a machine learning algorithm will perform are its ability to

1. make the training error small
2. make the gap between training & test error small.

underfitting occurs when model is not able to obtain a low value on training set.

overfitting occurs when gap b/w training and test error is too large.

model's capacity is its ability to fit a wide variety of functions.

One way to control capacity is choosing its hypothesis space, the set of functions that the learning algorithm is allowed to select as being the solution

Occam's Razor : simplest hypothesis is the best

Vapnik-Chervonenkis dimension: (VC dimension) measures the capacity of binary classifier.

discrepancy b/w training error & generalization error is bound from above by a quantity that grows as model capacity grows but shrinks as model capacity training example increases.

machine learning are rarely used with deep learning because it is difficult to determine capacity of dl algorithms.

Bayes error:

The error occured by making a prediction from the true distribution.

no free lunch theorem:

averaged over all possible data generation distributions, every classification algorithm has same error rate when classifying previously unknown points.

This theorem implies that we must design our ml algorithms to perform well on a specific task.

Regularisation

A penalty called a regulariser is added to the cost function.

This is intended to reduce the generalisation error but not the train error.

cross-validation

It is important that no test examples are used in any way to make choices about the model, including its hyper parameters, so validation set is necessary.

Point Estimation:

attempt to provide single best estimation of some quantity of interest.

Function Estimation:

predicting a variable given an input vector x.

Bias:

The bias of an estimator

$$\text{bias}(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta$$

for bernoulli distribution,
estimator θ is unbiased (mean)

for gaussian distribution,
estimator sample mean is unbiased.

for gaussian,

Sample variance is biased

unbiased estimators are desirable but biased are used when they possess other properties.

standard error of mean : $\frac{\sigma}{\sqrt{m}}$

we can use this to compute the probability that the true expectation falls in any chosen interval.

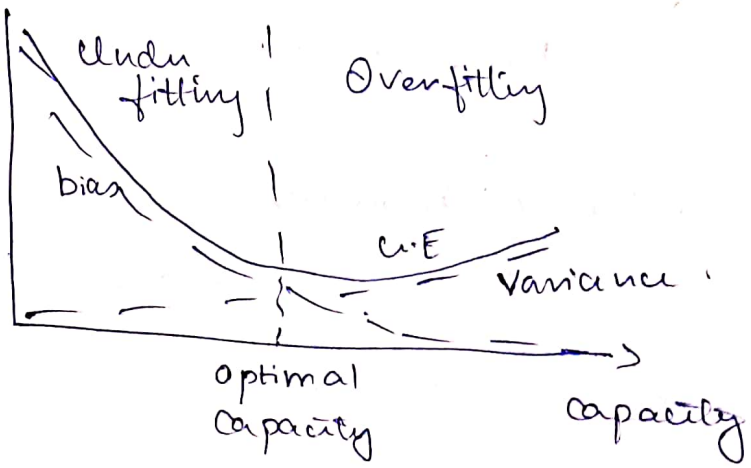
Bias- variance tradeoff

Bias measures expected variation from true value

Variance measures deviation from expected estimator value any data can have.

cross validation is used to tradeoff.

mean square errors can also be compared.



Consistency

as the no of data point grows, we wish that our estimates converge to true value.

Consistency ensures bias goes low as no of data grows.

Maximum Likelihood Estimation

principle from which we can derive specific functions that are good estimators.

It minimises the dissimilarity b/w emperical distribution and model distribution, with degree of dissimilarity measured by KL divergence.

minimising KL divergence ≡ minimising cross entry b/w distris.

Maximum Likelihood becomes minimisation of negative-log-likelihood or minimising cross entropy.

Properties of ML

under proper conditions, it is consistent.

- true distribution must lie within model family. Otherwise no estimator can recover true data
- true distri must correspond to 1 value of θ, otherwise ML can recover one value of correct true data

but will not be able to determine which value of θ was used

When no of examples are small enough to yield overfitting behaviour, regularisation strategies like weight decay can be used to obtain a biased version of MLE that has less var when Training data is limited

Bayesian Statistics

It uses probability to reflect degrees of certainty in state of knowledge.

Before observing data, our knowledge of θ is given using prior probability distribution.

The prior usually begins as a relatively uniform or Gaussian distribution with high entropy

It makes predictions using full distribution of θ

The prior often often has an influence on models making it smoother & simpler.

Maximum a Posteriori Estimation (MAP)

when we need point estimate using bayesian statistics. MAP estimate chooses the point of maximal posterior probability

Same as Bayesian inference, MAP Bayesian inference has the advantage of leveraging info that is brought by the prior & cannot be found in training data.

This additional info helps to reduce the variance in MAP point estimate.

MLE can be thought as MAP estimate with Bayesian inference.

Support Vector Machines

It outputs a class identity.

Kernal trick consists of observing many ML algorithms can be written exclusively in terms of dot products b/w examples.

It enables us to learn models that are non-linear as a function of using convex optimization techniques that are guaranteed to converge efficiently.

It also admits an implementation that is computationally more efficient.

The most commonly used kernal is Gaussian kernal. It is also called radial basis function.

Drawback to kernal machines is that the cost of evaluating decision fn is linear in the no of training examples.

K-n-n

can be used for reg & classi
It can achieve very high capacity

Our weakness is that it can't learn that our feature is more discriminative than other.

PCA

Can view PCA as unsupervised learning which learns representation of data.
It learns a representation that has lower dimension to that of original data.
It also learns which elements have no linear correlation to each other.
So it is used as a simple data reduction to keep as much data as possible.

Stochastic gradient descent.

used to train large data sets
insight of SGD is that it is that gradient is an expectation. In each step of algorithm, we can sample a minibatch of examples

Curse of dimensionality.

Our challenge is statistical challenge.
Many machine learning algorithms assume that output at new point should be approximately the same as the output at the nearest training point ...

the can in high dimensional data.

dupleaming introduces additional priors to reduce generalisation error

Manifold learning

A manifold is a connected region.
It tends to be used more loosely to designate a connected set of points that can be approx well by considering only a small no of degrees of freedom.
this assumes that interesting inputs occur only along a collection of manifolds containing a small set of points.

Deep Feedforward nlws.

Feed forward nlws are called networks because they are typically represented by composing together many different functions.

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

$f^{(1)}$ → first layer

$f^{(2)}$ → second layer

overall length gives depth.

The dimensionality of hidden layers set the width of the model.

To extend linear models to represent nonlinear functions of x, we apply linear model to a transformed input $\phi(x)$ where ϕ is a non-linear transformation

The strategy of deep learning is to learn ϕ.

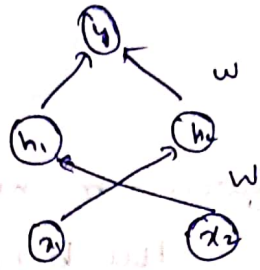
we have $y = f(x, \theta, w) = \phi(x, \theta)^T w$.

ϕ defines the hidden layers

we optimisation algorithm to find θ that corresponds to a good representation of x.

XOR implementation

linear model can't do this.



$h = g(x^T W + c)$

The output layer is still linear model, but it is applied to h.

Default activation function is ReLU.

For feedforward nlws, it is important to set small random values for weights.
Bias may be zero or small +ve value.

Cost functions

In most cases, our parametric model defines a distribution $p(y|x;\theta)$ and we simply use principle of maximum likelihood. So cross entropy b/w training data & model's predictions is used as cost function. (negative log likelihood)

sometimes, we merely predict some statistic of y conditioned on x. Specialised loss functions enable us to train a predictor of these estimates.

weight decay regularisation is often used with the cost function.

$$J(\theta) = -E_{x,y_{data}} \log P_{model}(y|x)$$

An advantage of deriving cf from maximum likelihood is that it removes the burden of designing cost fns for each model. Specifing a model $p(y|x)$ automatically determines $\log p(y|x)$

gradient of the c.f must be large & predictable enough to serve as a good guide for the learning algorithms.

Functions that saturate (gradient very small) undermine this.

The negative log likelihood helps in avoiding this problem.

Several o/p units involves an exp function that can saturate at negative arguement. log likelihood undoes this.

Learning Conditional Statistics.

Instead of learning full probability distribution $P(y|x;\theta)$, we often want to learn just one conditional statistic of y given x.

A tool called calculation of variations can be used.

The optimisation problem

$$f^* = \arg\min E_{x,y\sim p_{data}} \|y - f(x)\|^2$$

yields

$$f^*(x) = E_{x\sim p_{data}(y|x)}[y]$$

If we could train on infinitly many samples from the data generating distribution, minimising the mean squared error cost function would give a function that predicts the mean of y for each value of x.

$$f^* = \arg\min E_{x,y\sim p_{data}} \|y - f(x)\|,$$

yields a fn which predicts median value of y for each x. This is called mean absolute error function

But mean squared error & mean absolute error often lead to poor results when used with gradient based optimisation. This is one reason why ~~grd~~ cross entropy is preferred.

Output units

The choice of cost function is tightly coupled with the output unit. The units are mostly the same in hidden layer aswell.

Linear Units for Gaussian Output Distributions

$$\hat{y} = W^T h + b$$

linear output units are often used to produce mean of a conditional Gaussian distribution.

$$p(y|x) = N(y; \hat{y}, I)$$

The ML framework makes it straightforward to learn covariance of Gaussian, to make covariance of the Gaussian be a function of the input. However, covariance must be constrained to be a +ve define matrix, which is difficult with linear unit, so other units are used as well.

Linear units do not saturate, they pose little difficulty for gradient-based optimisation.

Sigmoid Unit for Bernoulli Output Distributions

The ML approach is to define a Bernoulli distribution over y conditioned on x.

A Bernoulli distribution is defined by just a number. Eg: $P(y=1|x)$

This wouldn't be efficient with a linear unit.

$$\hat{y} = \sigma(w^T h + b)$$

Because the C.F used is $-\log(y|x)$, the log undoes the (exp) in sigmoid.

So it doesn't allow it to saturate.

But it saturates when i/p is extremly -ve & +ve

Softmax Units for Multinoulli Output Distributions

for probability distribution over a variable with n values, we may use softmax fn.

$$\hat{y}_i = P(y=i|x)$$

$$z_i = \log P(y=i|x)$$

$$\log \text{softmax}(z)_i = z_i - \log \sum_j \exp(z_j)$$

input z_i always has a direct contribution to the cost function. This term can't saturate, so the learning can proceed even if the rest terms contribute very less.

* The negative log likelihood function always strongly penalises the most active incorrect prediction.

Many objective fns other than log-likelihood do not work as well with softmax. It can saturate when diff between input values vary largely.

numerically stable variant

$$\text{softmax}(z) = \text{softmax}(z - \max_i z_i)$$

The softmax outputs always sum up to 1.

Other Output types

Neural Networks with gaussian mixtures as their output is often called mixture density networks.

They are particularly effective in generating models of speech and movement of physical objects.

Hidden Units

ReLU are excellent default choice.

$g(z) = \max(0, z)$ is not differentiable at $z = 0$. In practice, gradient still works because it goes to a very low value, not necessarily 0.

Hidden units that are not differentiable are usually nondifferentiable at only a few points.

In general, a function $g(z)$ has a left derivative defined by the slope of the fn immediately to the left of z & a right derivative defined by the slope of the fn immediately to right of z. A fn is differentiable at z only if left d and right d are equal and are defined.

Software implementation returns usually one of the derivative.

So we can disregard nondifferentiability in practice.

Rectified Linear Units and their generalisations

$$g(z) = \max\{0, z\}$$

ReLU returns 0 for half of its domain.

This makes derivative through a ReLU remain large whenever the unit is active.

2nd derivative is almost 0 everywhere.

derivative of rectifying unit is 1 everywhere that the unit is active.

$$h = g(W^T x + b)$$

all elements of b to small +ve, 0.1.

This makes it very likely the ReLU will be initially active for most i/ps in training and allows derivative to pass through.

They can't learn via gradient-based on examples for which their activation is ReLU.

Three generalisation of ReLU are based on a nonzero slope α_i when $z_i < 0$

$$h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

Absolute value Rectification fixes $\alpha_i = -1$ to get $g(z) = |z|$.

Used for object detection from images.

Leaky ReLU fixes $\alpha_i = 0.01$

Parametric ReLU treats α_i as learning para.

Maxout Units

Instead of applying element wise function $g(z)$, maxout units divide z into group of k value Each maxout unit then outputs the max element of one of them groups:

$$g(z_i) = \max_{j \in G^{(i)}} z_j$$

maxout units need more regularisations (as they are parameterised by k weights)
Not needed if training set is large and no of units pieces per unit is low.

can gain some statistical and computational advantages by requiring fewer parameters.

As they are driven by multiple filters, they have some redundancy that helps them resist catastrophic forgetting - they forget how to perform tasks they were trained in the past.

ReLU and their generalisations are based on the principle models are easier to optimise if their behavior is close to linear.

Logistic sigmoid & hyperbolic tangent

$\sigma(z)$ $\tanh(z)$

$\tanh(z) = 2\sigma(2z) - 1$

They are used in hidden units is limited due to saturation.

hyperbolic tangent performs better.

Other hidden units.

$\cos(w^T x + b)$ on MNIST gives error rate of 1%.

Some layers can be purely linear.

Radical basis function:

$$h_i = e^{\left(-\frac{1}{\sigma_i^2}\|W_{:,i} - x\|^2\right)}$$

Saturates to 0 for most x.

Softplus

$$g(a) = \zeta(a) = \log(1 + e^a)$$

generally discouraged as rectifier is better

Hard tanh

$$g(a) = \max(-1, \min(1, a))$$

Similar to tanh, rectifier but bounded.

Architecture Design.

depth & width of each layer found via experimentation by monitoring validation set error.

Universal Approximation theorem.

a ffn with a linear output layer and atleast one hidden layer with any 'squashing' activation fn, can approximate any Borel measurable fn, from one finite dimensional space to another with any desired nonzero amount of error, provided the nw is given enough hidden units.

The derivatives of the fn can approx derivatives of function arbitarily well.

Basically, any continous fn on a closed and bounded subset of R^n may be approx by nn. A nn may also approx any fn mapping any finite dimensional discrete space to another.

Complications in Back propagation

Most implementations require return of more than one tensors.

They also need to handle computations involving multi tensors, making it computation heavy.

They also need to handle 32-bit, 64-bit floating point & integer value types.

Some ops have undefined gradient.

Higher Order Derivatives.

it is rare to see a single second derivative of a scalar fn. Instead properties of Hessian matrix is seen, of size $n \times n$ where n can run into billions.

So Krylov methods are used. They are a set of iterative methods for performing various tasks without any op other than matrix vector products.

To use them on Hessian, we only need to compute product b/w Hessian & an arbitary vector v.

$$Hv = \nabla_x [(\nabla_x b(x))^T v]$$

Regularisation in DL

They extra constraints and penalties can lead to improved performance on the test set. Sometimes they are designed to encode prior knowledge.
Ensemble methods combine multiple hypothesis that explain the training data.

Most strategies are based on regularising estimators. It works by trading increased bias for reduced variance.

Parameter norm penalties

parameter norm penalty added to the objective function.

$$J = J + \alpha \Omega(\theta) \qquad \alpha \rightarrow \text{hyperparameter}$$

we typically choose to use a parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer & leaves bias unchanged. The bias typically require less data than the weights to fit accurately Regularising the bias can bring in underfitting

So w is used to denote weights that should be affected by a norm penalty and θ denotes all the parameters.

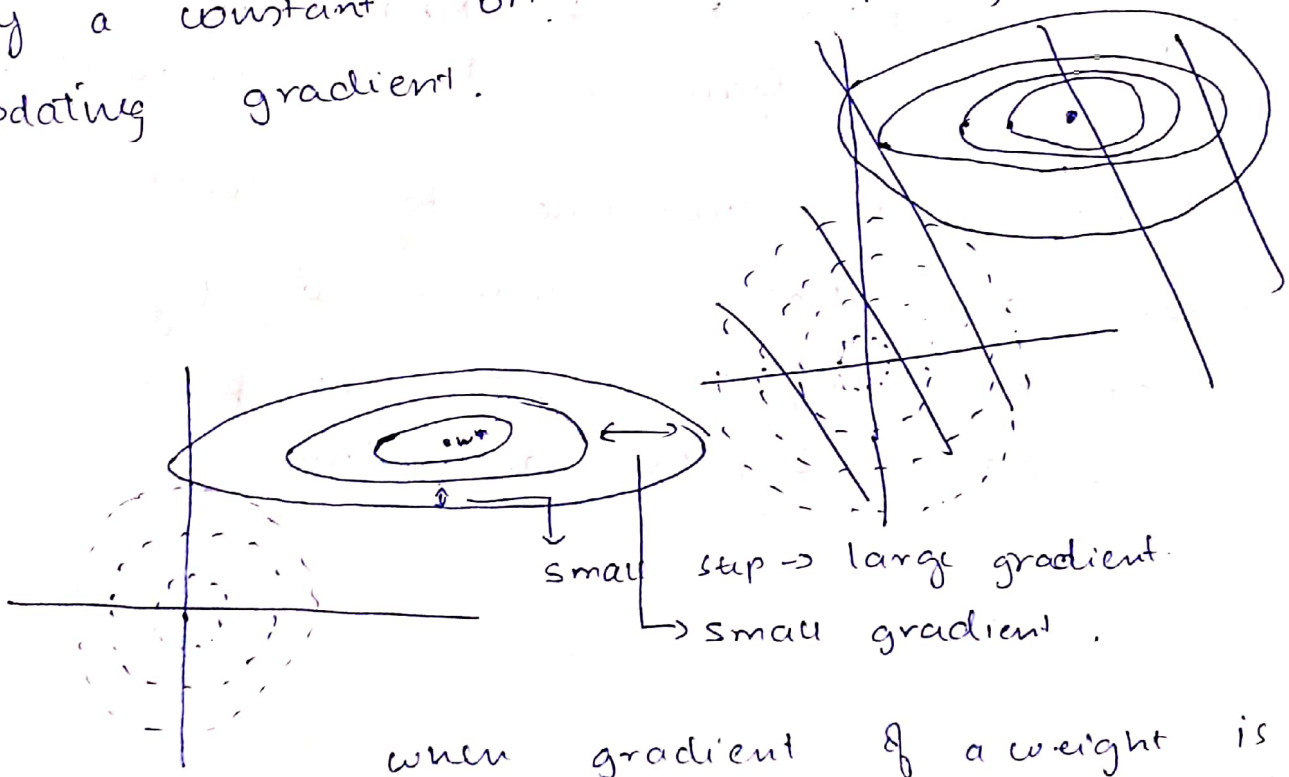
It is sometimes desirable to use a separate penalty with a different α coefficient for each layer of the network.

L^2 parameter Regularisation - weight decay

It drives the weights closer to the origin by adding a Reg term $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$.

It is also known as Ridge regression and Tikhonov Regularisation.

This multiplicatively shrinks the weight vector by a constant on each step just before updating gradient.



when gradient of a weight is small, the Reg term pulls it to 0. when it is large, there is little change

Only the directions along which the parameters contribute significantly to reducing the objective function are preserved relatively intact

L^1 Regularisation

$$\Omega(\theta) = ||w||_1$$

Reg contribution to the gradient no longer scales linearly, instead a constant factor $sign(w_i)$

Some parameters take value of zero.

This is used for feature selection.

L^2 Reg is equivalent to MAP Bayesian Influence with a Gaussian Prior on weights

For L^1 Reg, it is equivalent to the log prior term that is maximized by MAP Bayesian inference when the prior is an isotropic Laplace distribution.

Norm penalties as constrained Optimisation

parameter norm penalty can be thought as imposing a constraint on weights.

If Ω is L^2, weights are constrained to lie in L^2 ball.

If Ω is L^1, weights are constrained to lie in L^1 region.

The relationship between α & this region depends on J.

Large α → small region.

Sometimes explicit constraints can be used rather than penalties, like modifying algorithms eg: s.g.d instead of objective fn.

Penalties can cause nonconvex optimisation to get stuck in local minima corresponding to small θ. This usually manifests as nn that train with several dead units.

In these cases, explicit constraints might work better because they do not encourage weights to approach origin.

Also explicit constraints bring stability. When using high α, it can cause large weights, in turn large gradients, inturn θ moves away from origin till numerical overflow. Explicit constraints prevents this feedback loop

Hinton suggests constraints with high α which maintain stability

He suggests constraining norm of each column of weight rather than Frobenius norm of entire weight matrix. This prevents any one hidden unit from having very large weights.

Regularisation and Under Constrained problems

In many cases X^TX needs to be inverted, but not possible when it is singular.

Many forms of Reg correspond to inverting $X^TX + \alpha I$. This matrix is invertible.

Moore Penrose pseudo inverse can be interpreted as stabilising under determined problems using regularisation.

Data set Augmentation

very effective for object recognition, translating training images a few pixels in each direction can improve generelisation rotating, scaling image also works.

Effective for even speech recognition.
Injecting noise can also be considered.

Noise Robustness

generally, adding noise can be more powerful than shrinking parameters.

Injecting noise at output

Data can be wrongly labeled, so giving mislabeled data during training and giving training set label probabilities might work.

label smoothing regularises model by replacing hard 1, 0 classification instead.

$\frac{\epsilon}{k-1}$ & $1-\epsilon$.

Semi Supervised learning

Both labeled examples from $P(x, y)$, unlabeled from $P(x)$ are used to find $P(y|x)$ or predict y.

learning a representation $h = f(x)$

one variant is applying PCA first.

Multi-task learning

pooling same examples from different tasks.

Eg: different supervised tasks share same input as well as some intermediate level representation

- Task specific parameters, these are upper layers. (not shared)
- generic parameters, shared across all (lower levels)

training Error dicreams over time but validation Error begin to rise again

The algorithm terminals when no paramtrs have improved over the best recorded validation error for some iteralions.

This is a form of Regularisalion

Our idea is to train for a small set and stop early. Then retrain for full data & stop at the same ~~no~~ no of steps.

In case of linear model with Quadric error fn, & grid, ES is = L^2.

Parametr tying & parameter sharing.

A common type of dependancy that need to be expresnd is that certain paramtrs should be clos to one another.

Adjusting paramtrs of multipe models which are similar by eg

keep a Rg. $\Omega(w^A, w^B) = \|w^A - w^B\|_2^2$.

In Paramtr sharing, weights are forced to be equal. Only few subsets need to be stored

parameter sharing happens in CNN

Sparse Representations

penalty on the activations of units in NN, encouraging their activation to be sparse

$$J = J + \alpha \Omega(h)$$

where h is a function of x, which does Representational reg.

orthorgonal matching pursuit encodes input x with h

$$\arg\min_{h, \|h\|_0 < k} \|x - Wh\|^2 \quad \rightarrow \text{OMP-k}$$

$\|h\|_0$ → no of nonzero entries of h.

k → no of nonzero features allowed.

OMP-1 is a very effective feature extractor.

Bagging

Combining different models

Model averaging trains several models & all models vote on output.

This reduces gen error

def in k models, k datasets, train i models on i datasets

Boosting incrementally adds nn to ensemble.

Dropout

This trains ensemble consisting of all subnetworks that can be formed by removing non output units.

The units which are to be removed, its output is multiplied by 0, among other methods.

minibatch based algorithm is small steps (sgd)

parameter sharing happens here as multiple models might have same units.

Computationally cheap.

works well with any model with sgd.

~~works~~ not effective with less labled examples.

Adversial training

training an adversially perturbed examples, one main cause is excessive linearity

It accomplish' S- supervised learning.

Tangent Distance, Tangent Prog, Manifold Tangent Classifier.

tangent distance alg is a nonparametric nearest neighbour algo where metric is not Euclidian but one that is derived from knowledge of the manifolds. near probability centres.

Tangent prop algorithm trains with an extra penalty to make each output locally invariant of factors of variation. Closely related to Dataset Augmentation. It needs tangent vectors as prior.

Manifold tangent classifier doesn't need tangent vectors as priori.

use autoencoder to learn manifold structure use tangents to regularise a nn as in tangent prop.

Optimisation for Training Deep Models.

Emperical Risk Minimisation

goal of ML algorithm is to reduce the expected generalisation error. This quantity is known as the risk.

The simplest way to convert a ml problem back into an optimisation problem is to minimize the expected loss on training set. i.e, replacing true distribution $p(x, y)$ with emperical distribution $\hat{p}(x, y)$ defined by the training set, emperical risk.

$$\mathbb{E}_{x,y \sim \hat{p}_{data}(x,y)} [L(f(x;\theta), y)] = \frac{1}{m} \sum L(f(x^i;\theta), y^i)$$

This is prone to overfitting. and is not feasible.

Surrogate loss functions.

acts as a proxy

Eg: -ve log-likelihood used as surrogate for 0-1 loss.

can improve robustness of the classifier by further pushing the class apart.

ml algorithm usually minimises a surrogate loss function but halts when a convergence criterion based on early stopping is satisfied. It is designed to halt when model begins to overfit.

Minibatch

- large batches provide a more accurate estimate of the gradient, but with less linear returns.
- Multicore Architectures are usually underutilized by extremely small batches. This motivates using some absolute min batch size below which time is no reduction in the time.
- small batches offer regularisation effect.

Challenges in Neural Network Optimisation

Ill Conditioning

While optimising convex function **, ill conditioning of Hessian matrix H is an issue.

This causes SGD to get stuck in the sense that even very small steps $-\epsilon g$ will add

$$\frac{1}{2}\epsilon^2 g^T H g - \epsilon g^T g$$ to the cost.

It becomes a problem when $\frac{1}{2}\epsilon^2 g^T H g > \epsilon g^T g$.
Newton's method is an excellent tool to avoid this but it needs lot of modifications.

Local Minima

One of the most prominent feature of convex optimisation problem is that it can be reduced to problem of finding local minima.

Neural nets can have many local minima. This is due to model identifiability problem. A model is said to be identifiable if a sufficiently large training set can rule out all but one setting to the model's parameter.

Model with latent variables are often not identifiable because we can obtain equivalent models by exchanging latent variables with each other. For m layers with n units, there are n^m ways of arranging the hidden units. This is called weight space symmetry.

There are other cases like scaling incoming weights by α and outgoing by $1/\alpha$.

However, all these local minima arising from non identifiability are equivalent to each other in cost fn value. So these local minima are not a problem.

Local minima can be problematic if they have high cost in comparision to global.

A test that can rule out local minima as the problem is plotting the norm of gradient over time. If it does not shrink to insignificant size, the problem is neither local minima nor any kind of critical point.

Saddle points:

At a saddle point, Hessian matrix has both +ve and -ve eigen values. Points lying along eigen vectors associated with +ve eigen values have greater cost than the saddle point -, while points lying along -ve eigen values have lower value.

In low dimension spaces, local m are common.
In high dimension space, local m are rare.
and saddle points are common.

as dimension increases, ratio of no of saddle points to local minima grows exponentially with n.

Hessian matrix at a local m has only +ve at saddle, it has +ve & -ve.

Hessian eigens are more to be +ve once cost goes down.

For newton's method, without appropriate changes, it'll result in saddle points.

Proliferation of saddle points in high dimensional spaces presumably explains why second order methods have not succeeded in replacing gradient descent.

Cliffs

layers often have extremely steep regions. Thus result from multiplication of several large weights together. On face of an extremly steep cliff structure, the gradient update step can move the parameters extremly far.

Its most serious consequences can be avoided using gradient clipping heuristic.

Basic idea is that, gradient specifies which direction to take and when step is large, this intervenes to reduce the step size.

Long Term dependencies.

computational graph becomes extremly deep.

Eg: Suppose a computational graph contains a path that consists of repeatedly multiplying W. A t steps, it becomes W^t.

if $W = V \text{diag}(\lambda) V^{-1}$

$W^t = V \text{diag}(\lambda)^t V^{-1}$

Any λ_i not near absolute value of 1 will either explode or vanish.

The vanishing and exploding gradient problem refers to the fact that gradients through such a graph are also scaled according to $\text{diag}(\lambda)^t$.

Vanishing gradient makes it difficult to know which side to move while exploding gradient makes learning unstable.

Inexact gradient

we have noisy / biased estimate of gradient or Hessian matrix; not fully accurate.

In some cases, objective fn to be minimized is intractable, so is its gradient. In that case, it should be approximated.

Poor Correspondence b/w local & global structure

Goodfellow argues that much of the runtime in training is due to length of trajectory needed to arrive at the solution.
In practice, NNs donot arrive at any global or local or any critical points, as how cost functions don't reach 0.

Basic Algorithms.

1) SGD

average gradient on a mini batch of m examples.
In practice, learning rate is increased with time denoted by ϵ_k, $k \to$ no of iterations.
It is common to decay learning rate linearly until iteration τ

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \qquad \alpha = \frac{k}{\tau}$$

After τ iterations, have ϵ constant.

Usually τ may be a set to # nos of iterations required to make a few hundred passes.
Usually ϵ_τ should by 1% of ϵ_0.

It is usually best to monitor the first several iterations & use a learning rate that is higher than the best performing learning rate at this time, but not so high as to cause instability.

The most important property of SGD & related minibatch is that computation time per update does not grow with no of training examples. For some large training set, it may converge to within some fixed tolerance of its final test set error before it has process the entire training set.

To study the convergence rate of an optimisation algorithm, it is common to measure the excess error $J(\theta) - \min_\theta J(\theta)$ which is the amount by which current cost exceeds min cost.

When SGD is applied to convex problem, it is $O(1/\sqrt{k})$ after k iterations, for strongly convex, it is $O(1/k)$.

2) Momentum

Designed to accelerate learning

It accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.

$$v = \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum L(f(x^i; \theta), y^i) \right)$$

$$\theta = \theta + v$$

$v \rightarrow$ velocity - direction & speed at which parameters move through parameter space.

SGD can be used with momentum.

the size of step depends on how large and how aligned a sequence of gradients are.

Position of the particle is given by $\theta(t)$. The particle experiences a net force $f(t)$. In this case particle to accelerate.

$$f(t) = \frac{\partial^2}{\partial t^2}\theta(t)$$

3) Nesterov Momentum.

$$v = \alpha v - \epsilon \nabla_\theta \left[\frac{1}{m}\sum L(f(x^i;\theta+\alpha v), y^i)\right]$$

$$\theta = \theta + v$$

with nesterov, gradient is applied after current velocity is evaluated. It can be interpreted as attempting to add a correction factor to standard method of mom.

Parameter Initialization Strategies

initial points can determine whether model can converge at all, it determine how fast it converges and whether to region high or low cost.

Initial parameter need to break symmetry between different units.

Random initialisation from a high entropy distribution over a high. d space is computationally cheaper.

biases for unit is set heuristically. almost always initialize all weights in the model to values drawn from a Gaussian / normal d.

Large initial weights → strong symmetry break, help avoid redundant units.

optimisation perspective suggests the weight to be large, but some regularizers suggest otherwise.

SGD suggests final parameters should be close to initial ones.

one of the normalized initialisation

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

↳ assumes no nonlinearities.

Another idea is initialising random orthogonal matrices with a carefully chosen scaling or gain factor.

↳ again no nonlinearities.

sparse initialisation: each unit is initialized to have k non zero weights. The idea is to help total no amount of i/p to unit independent from the no of inputs m without making the magnitude of individual weight elements shrink with m.

When computational resources allow it, it is usually a good idea to treat the initial scale of the weight of each scan as hyperparameter & using a hyperparameter search algorithm such as random search

A good rule of thumb for choosing initial scales is to look at range a S.D of activations / gradients on a single minibatch. If ~~they~~ weights are too small, range of activations will shrink as the activations propagate.

Setting bias to zero is compatible with most weight initialisation schemes.

exceptions:

- if bias is for output
- to avoid saturation of an activation
- when a unit controls whether other units are able to participate

variance of precision can be set to 1.

Algorithms with adaptive learning rates.

1) AdaGrad

individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all historical squared values of gradient.

great progress in gently sloped directions.
doesn't perform well for all.

2) RMSProp

modifies AdaGrad to perform better in nonconvex setting by changing gradient accumulation into an exponentially weighted moving average.

one of the go to optimisation method.

3) Adam

adding momentum to RMSprop.
momentum applied on rescaled gradients.
It includes bias correction.

learning rate sometimes needs to be changed from a selected default.

Approximate Second-Order Methods

1) Newton's Method

It is an opti scheme based on second order Taylor series expansion to approximate $J(\theta)$ near some point θ_0 ignoring derivatives of higher order.

$$J(\theta) = J(\theta_0) + (\theta-\theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta-\theta_0)^T H (\theta-\theta_0)$$

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$

for a quadratic fn (with +ve definite H), by rescaling the gradient by H^{-1}, Newton's method jumps directly to the minimum.

For surfaces not quadratic, it can be applied as long as H remains +ve definite. saddle points can make it move in wrong direction. This can be avoided by regularising the H.

One idea is to add a α to diag H.

$$\theta^* = \theta_0 - [H(f(\theta_0)) + \alpha I]^{-1} \nabla_\theta f(\theta_0)$$

4) Conjugate gradients.

efficiently avoid calculation of inverse H by iteratively descending conjugate directions.

we seek to find a search direction that is conjugate to the previous line search direction; i.e, it will not undo progress in the direction.

5) Broyden - Fletcher - Goldfarb - Shanno (BFGS)

tries to bring Newton's advantages w/o computational expenses.

similar to conjugate gradients, simplifies inverse gumration.

Limited Memory BFGS: avoids storing complete inverse of H.

Meta-Algorithms

1) Batch Normalisation

While computing gradient, we assume other values in layer remain same. But we update simultaneously so there can be some problem.

Batch normalisation provides an elegant way of reparameterising which reduces

Let H be a minibatch of activations of the layer to normalise.

To normalise,

$$H' = \frac{H - \mu}{\sigma}$$

at training time,

$$\mu = \frac{1}{m} \sum H_i$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum (H - \mu_i)^2}$$

we back-propagate thru these ops for computing μ & σ, & to apply them to normalise H. So gradient will not propagate an op that acts to increase σ or μ.

normalising μ & σ of a unit can reduce expressive power of nn of that unit. So it is common to replace batch of hidden unit activations H with $\gamma H' + \beta$.

2) Coordinate Descent.

optimising one coordinate at a time can be used when variables can be separated into groups. or when optimising with respect to one group of variables is significantly more efficient than optimisation with respect to all variables

5) Polyak Averaging

averaging several points in the trajectory through parameter space visited by opti algo. Idea is that opti algo may leap back & forth across a valley several times w/o ever visiting a point near the bottom. The average of all the locations on either side should be close to the bottom of the valley.

When applying for non convex, it is typical to use an exponentially decaying running average:

$$\hat{\theta}^{(t)} = \alpha \hat{\theta}^{(t-1)} + (1-\alpha)\theta^{(t)}.$$

6) Supervised Pretraining

training a simple model to perform a task, then making it more complex. These are called pre training.

greedy algos break a problem into many components. Though combining at the end need not give proper result. Mgs need fine-tuning in which an opti algo searches for a opti soln.

Convolutional Networks.

for processing data that has grid like topography (1D or 2D etc)

Neural networks that use convolution instead of multiplication in at least one of their layers.

Convolution

$$s(t) = x(t) * w(t)$$

w needs to be a valid p.d.f or output will not be a weighted avg.

& w should be 0 for -ve arguements.

x -> input

w -> kernal

s -> feature map.

Convolution over multiple axes,

$$s(i,j) = (I * K)(i,j) = \sum\sum I(m,n)\, K(i-m, j-n)$$

Many libraries, calculate cross correlation & call it convolution.

$$s(i,j) = (K * I)(i,j) = \sum\sum I(m+i, n+j) K(m,n)$$

Convolution leverages 3 important ideas.

- Sparse interactions
- Parameter sharing
- equivariant rep

it also provides a means for working with variables of diff sizes.

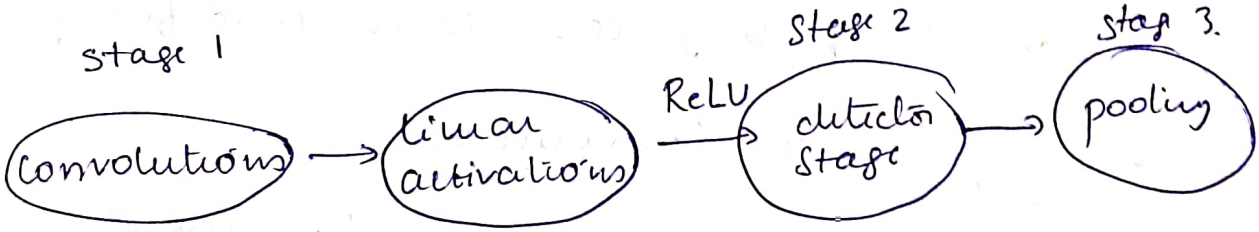
kernals are smaller than input. so important info can be stored with less computation.

parameter sharing - using same parameter for more than 1 fn in a model.

the particular form of parameter sharing causes layer to have property called equivariance. → if i/p changes, o/p changes in same way. $f(g(x)) = g(f(x))$.

Pooling.

A typical CNN consists of 3 stages.



A pooling fn replaces the output of the net at a certain location with a summary statistic of nearby outputs.

Max pooling op reports maximum op within a rectangular neighbourhood.

other ~~means~~ fns include avg, L^2, weighted avg etc

pooling helps to map rep approx invariant to small translations.

If we pool over the o/ps of separately parameterised convolutions, the features can learn which ~~parameters~~ transformation to become invariant to.

For various op, pooling is necessary to handle i/ps of varying sizes

We can imagine CNN as being a fully connected net, but with infinitely strong prior over its weights. This says weights of hidden should be equal to another but shifted in space.

convolution & pooling can cause underfitting.

Variants of Basic Convolution

Initially zero padding is done to input.

if there is no zero padding, kernal can shrink every layer

optimally enough zero padding is done so that is not zero of its not too much to make kernals work difficules.

In some cases, locally connected layers are used instead of convolution. This is called unshared convolution.

This is useful when we know that each feature should be a fn of a small part of space, but there is no reason to think that same feature occurs across all of space.

Tiled convolution offers a compromise blw a convolution layer and a locally connected layer. We learn a set of kernels that we rotate through as we move through space.

If the filters of locally connected layers & tiled learn to detect diff transformed versions of the same underlying feature, then the max pooled units become invariant to learned transformation.

Other operation ~~such~~ besides as convolution is necessary to compute CNNs. gradient is needed to be found wrt kernal, given the gradient wrt output.

Multiplication by the transpose of the matrix defined by convolution is one op. This is needed to back-propagate err derivatives.

This is also needed to reconstruct visible units from hidden units.

the 3 ops; convolution, backprop from o/p to weights and backprop from o/p to i/p are sufficient to train CNN.

Structured Outputs.

CNNs can be used to output a high dimensional structured object.

reduction in spatial dimension occurs due to large stride in pooling. One can simply emit a lower-resolution grid of labels instead of pooling process.

Data Types.

1-D audio single channel:

axis to convolve over is time.

discrete time & measure amplitude at each step

1-D multichannel skeleton animation data:

Animations of 3-D computer generated characters

1-D single channel audio data with Fourier transform:

audio waveform converted to 2-D tensor using FTs. and applying convolution op in freq domain, which is easier.

2-D multi channel colour image:

RGB, 3 different values for each pixel. convolution matrix moves horizontally & vertically over the image.

3-D single channel volumetric data:

CT scans etc

3-D multi channel video (colour):

extra axis for time, then height, width

Efficient convolution Algorithm.

sometimes easier to convert to some other domain & revert back instead of applying convolution directly.

when a d-dimensional kernal can be expressed as the outer product of d vectors, one vector per dimension, kernal is called separable. On this can direct convolution is inefficient.

Random or Unsupervised features.

- 3 strategies for obtaining convolution kernels w/o supervised training.
  Initialising randomly, setting by hand or by unsupervised training.

  greedy layer pretraining, to train first layer in isolation, then extract features.

Gabor functions.

It describes weight at a 2-D point in the image.

Sequence Modeling : Recurrent & Recursive Nets.

RNNs

family of NNs for processing sequential data. It can scale to much longer sequences, including sequences of variable length.

A RNN shares the same weight across different time steps.

Each member of the o/p is a fn of previous members of the output.

They operate on a series of vectors $x^{(t)}$ t ranging from 1 to τ. RNNs usually operate on minibatches of such sequences, with a diff τ for each memb of minibatch. They can also be applied to 2D data

Practical Methadology.

- determin goals - what error metric to use, target value for it etc.
- establish working end-to-end pipilin asap.
- Instrument the System will to determin bottlenecks.
- Repeatedly make incremental changes like gathering new data.

Performance Metrics.

impossible to achieve absolute zero error.

amount of training data can be limited.

Accuracy is a poor way to represent a System.

Precision & Recall is better.

Precision is the fraction of detections reported by the model that was correct.

Recall is the fraction of true events that was detected.

$$F\text{-score} = \frac{2pr}{p+r}.$$

Another option is to report total area lying beneath PR curve.

Coverage is the fraction of examples for which the machine learning system is able to produce a response

Default baseline Model.

- choose general architecture model based on structure of data.
- Reasonable choice for optimisation algorithm is SGD with momentum with a decaying learning rate. Other option is Adam.
  Batch Normalisation can help a lot.
- Unless training set has 10s of millions, include mild forms of regularisation from start.

Early stopping is used almost everytime.
Dropout is an excellent Regularizer.

Determine whether to gather more data:

if performance on training set is poor, that means algo is not using available data properly. Increase size of the model in that case.

if performance on training set is acceptable, check it on test set. if test set score is worser, then its a good idea to gather more data.

Alternative to gather more data is to reduce size of model & increase regularisation

Manual Hyperparameter Tuning -

goal is to find lowest generalisation error subject to some runtime & memory budget.

~~Effect~~ ~~Its~~

adjust the effective capacity of the model to match complexity of task. It is constrained by representational capacity, minimising cost function & the degree to which C.F & training regularise the model.

Learning rate can be the most important hyperparameter.

Grid Search

When there are 3 or fewer hyperparameters, common practise is grid search.

For each hyperparameter, user selects a finite set of values to explore. Then grid search perform for all combination

The experiment that yields the best validation set error is chosen as having the best hyperparameters.

Random Search.

much fast to get good values of hyperparametus.

First define a marginal distribution for each hyper paramete, eg: Bernoulli on a multinoulli. Do no discrete the values.. Random search has no wasted experiments

Model-band Hyper paramete Optimisation.

The search for good hyperparamete can be cast as an optimisation problem. This involves gradient of hyper peramete wrt validation error, but this might not be available practically.

To compensate for this, build a model of validation error and propose new hyper paramete guss by optimisation in this model.

Most model-band optimisation uses Bayesian regression model.

Contemporary approaches to hyper paramete optimisation include Spearmint, TDE & SMAC.

Debugging tools.

Visualise the model in action.

eg: view some images with detections proposed by the model displayed superimposed on the image

Visualize the worst mistake:

Most models output some sort of confidence

Compare back-propagated derivatives to numerical derivation.

< Applications >

Recurrent neural networks.

sequence of values.

RNN shares weight across several time steps.

$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$

this is used to represent value of hidden units

the nlw typically learns to use $h(t)$ as a kind of lossy summary of the task-relevant aspects of the past sequence.

Depending on the training criteria, summary might keep some aspects of the past sequence with more precision than other aspects.

unfolding:

each node is represented with particular time instance.

$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \ldots x^{(1)})$ → with unfolding

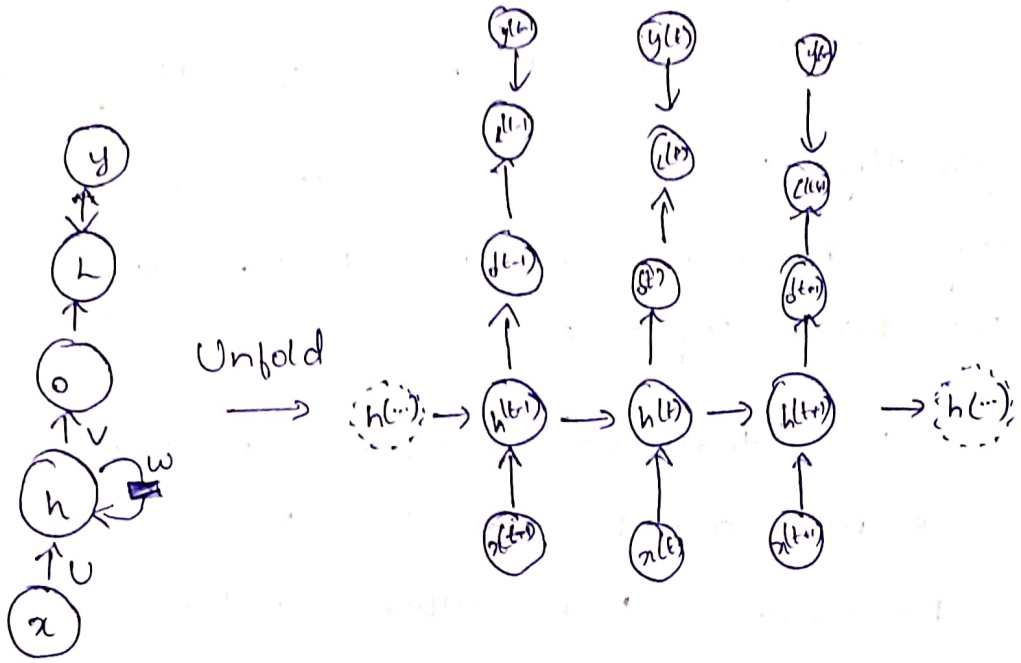
$= f(h^{(t-1)}, x^{(t)}; \theta)$

unfolded graph provides an explicit description of which decision to perform.

It also helps illustrate the idea of info flow forward in time & backward in time.

Important design patterns.

1. produce o/p at each time stp & have recurrent connections b/w hidden units.



Any fn computable by turing machine can be computed by such a recurrent nw of finite size.

RNN when used as a turing machine, takes binary sequence as inputs and its outputs must be discretised to provide a binary output.

x → input
h → hidden units
o → output
L → loss from each of

Assuming hyperbolic tan as activation, describe o/p for each time step,

$$a(t) = b + w(h(t-1)) + Ux(t)$$
$$h(t) = \tanh(a(t))$$
$$o(t) = c + Vh(t)$$
$$\hat{y}(t) = \text{softmax}(o(t))$$

$$L(\{x^{(1)}, \ldots, x^{(T)}\}, \{y^{(1)}, \ldots, y^{(T)}\})$$
$$= \sum_t L^{(t)}$$
$$= -\sum_t \log P_{model}(y^{(t)} \mid \{x^{(1)}, \ldots x^{(t)}\})$$

The Back propagation applied to the unrolled graph is called back propagation through time (BPTT).

2. produce o/p at each time step & have recurrent connection from o/p of one time step to hidden units at next time step.

Because it lacks h-h recurrence, it requires o/p units to capture all info about the past that the Nw needs to predict.

advantage is training can be parallised.
There is no need to compute o/p for previous time step.

teacher forcing

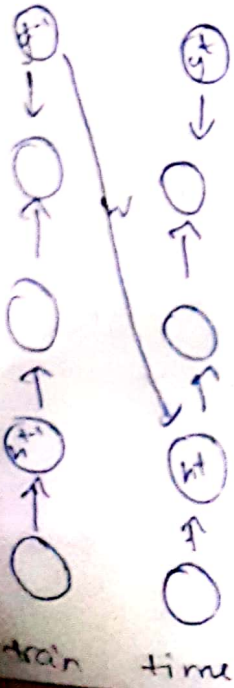
Models that have recurrent connections from their output leading back into the model may be trained with teacher forcing.

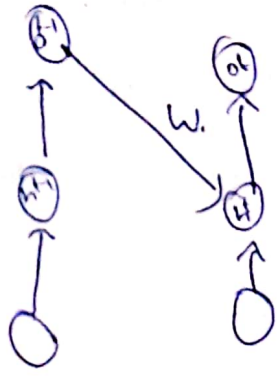
The model receives ground o/p $y^{(t)}$ as input time $t+1$.

This emerges from Maximum likelihood setup.

$$\log p(y^{(1)}, y^{(2)} \mid x^{(1)}, x^{(2)})$$

$$= \log p(y^{(2)} \mid y^{(1)}, x^{(1)}, x^{(2)}) + \log p(y^{(1)} \mid x^{(1)}, x^{(2)})$$





test time

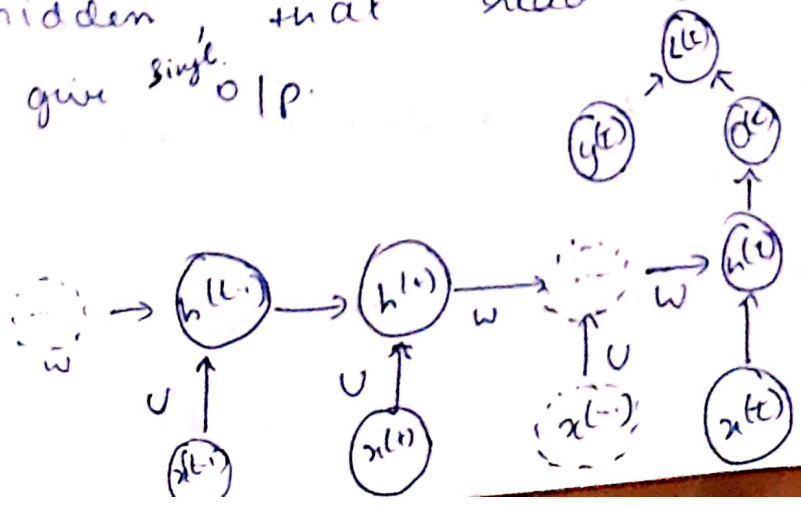
status that instead of feeding model's own o/p during training, the connections should be fed with target values specifying what correct o/p should be.

It can still be applied to models with h-h connections from o/p from one time step to values computed at next.

But as soon as h-h connections are there, BPTT is necessary.

Disadvantage arises when it is in closed loop. (o/p fed as i/p). One way to mitigate this is provide both TF, free running i/ps. Another approach to mitigate the gap b/w the i/ps seen at training time & i/ps seen at test time randomly choose to use generated values or actual values as i/p.

3) Recurrent n/ws with recurrent connection b/w hidden, that read entire sequence & then give single o/p.



Computing gradient in RNN

apply BP on unrolled graph., in this can BPTT.

final loss ~~at each~~ node $\frac{\partial L}{\partial L^{(t)}} = 1$ (~~first~~

(more equation)

Recurrent Networks as Directed graphical Models

we usually wish to interpret o/p of RNN as probability distribution & use cross entropy associated with the distribution to define the loss.

edges in a graphical model indicate whether variables directly depend on others.

Many graphical models omit edges that do not correspond to strong interactions

Markov assumption is that g.m should contain only edges from $\{y^{(t-k)}, \ldots y^{(t-1)}\}$ to $y^{(t)}$

In some cases, all i/ps are believed to have influence on next element.

The price RNNs pay for reduced no of parameters is optimisation.

extension of graphical model represents it as a joint distribution over y variables.

modelling sequences conditioned on context with RNNs

represented as a conditional distn over eg

given x.

represent $P(y|x)$ by $P(y|w)$ where

$w = f(x)$.

one option is to take only a single x as input. This can be given as extra i/p.

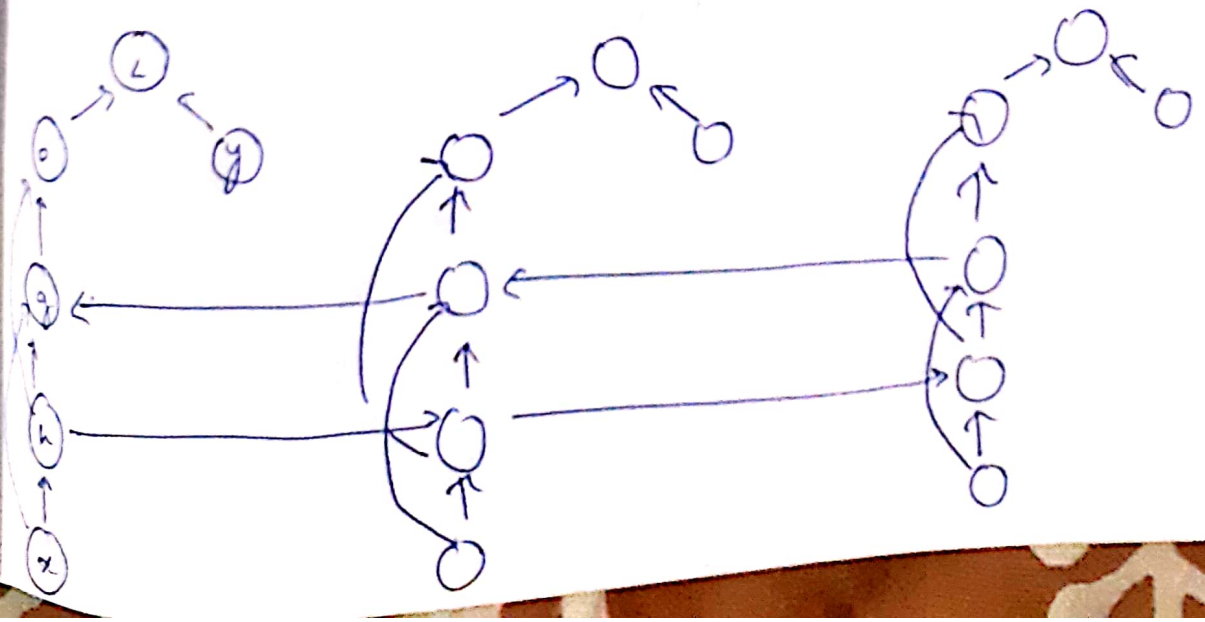
Two ways are.

- add extra i/p at each time step.
- th [illegible]
- both

Bidirectional RNNs.

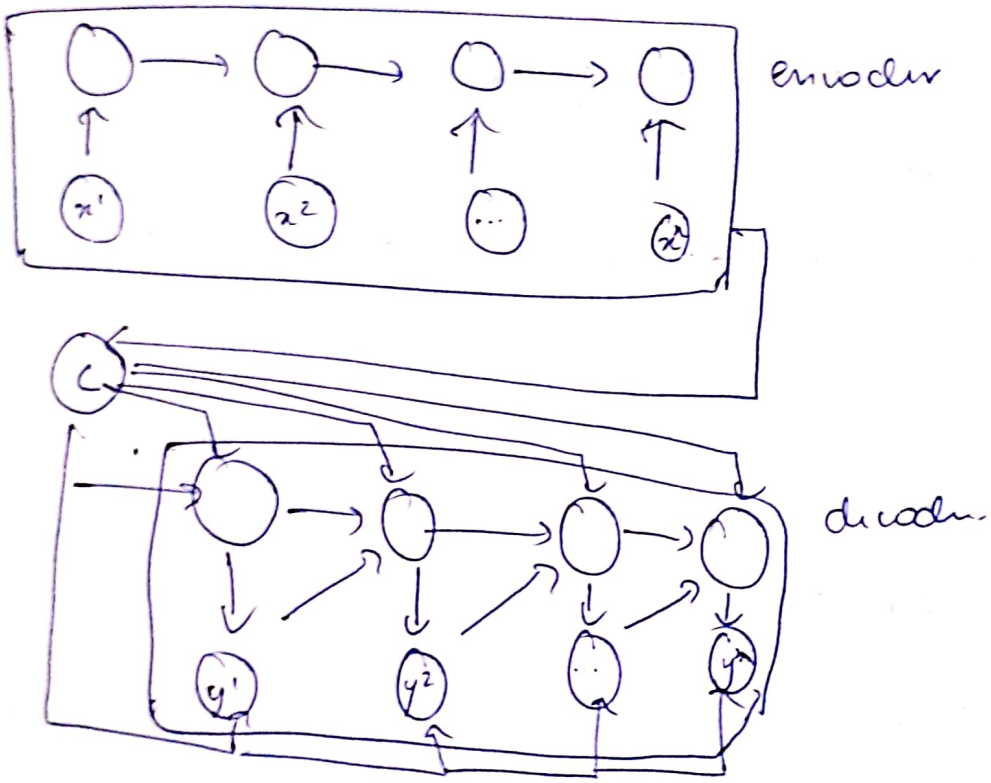
very successful in handwriting, speech, bioinformatics

* combines a RNN moving forward in time
& another RNN which moves backward in time



h propagates through time, g backwards. at each time step, output unit is built from relevent summary of the past in h & future in g

Encoder - Decoder Sequence to Sequence Architecture



needed when x & y have different variable length.

final state of encoder is used to compute C which represents summary of i/p & is given as i/p to decoder.

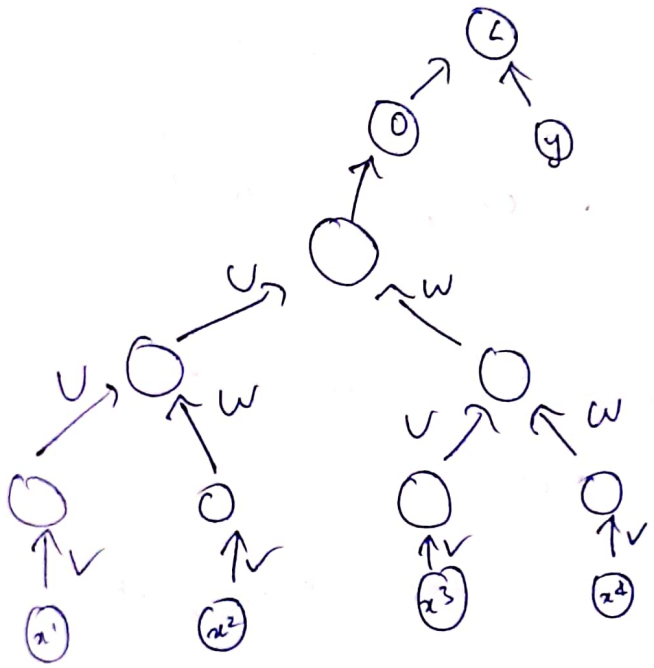
Deep Recurrent N/ws.

Experimental evidence suggests that introducing depth in each block (ip-h, h-o etc) can be helpful.

Recursive NNs

structured as deep tree.

depth can be reduced from τ (RNNs) to $O(\log \tau)$



many variants of this net are possible.

Challenge of long term dependencies

Can result in extremly non linear behavior.

Any component of $h^{(t)}$ not aligned with largest eigen vector will eventually be disregarded due to vanishing.

In order to store memories in a way that is robust to small perturbation, RNN must enter a region of parameter space when gradients vanish

Echo state N/ws.

$h^{(t-1)}$ to $h^{(t)}$ & $x^{(t)}$ to $h^{(t)}$ are most difficult parameters to learn.

Our proposed approach to avoid this is to set recurrent weights such that recurrent hidden units do a good job of capturing past inputs & only learn the output weights

In liquid state machines it same but has neurons with binary o/ps.

These two are termed reservoir computing

how to set weights?

make eigenvalues of Jacobian close to 1.

Leaky Units & other ideas

Some part of model operate at fine grained time scales & can handle small details & some part operate at coarse time scales & transfer info from part

future effectively.

Skip connections

One way to obtain coarse time scales is to add direct connections from distant past to variables in present.

Removing connections

removing length-one connections & adding longer ones.

One option is to make them leaky but to have different groups of units associated with different fixed time scales.

LSTM

introduce self loops to produce paths where gradient can flow for long durations, make weights conditioned on i/p.

LSTM has LSTM cells that has internal recurrence (self loop)

GRUs (gated RNN recurrent units)

main diff with LSTM is that a single gating unit simultaneously contr forgetting factor & decision to updat

Explicit Memory

To store key design aspects of the neural turing machine. Facts are stor The task n/w learns to control th memory, deciding when to read from and when to write to within memor

Linear Factor Models.

probabilistic models with latent variables is called linear factor models.

It uses a linear stochastic decoder fn. that generates x by adding noise to a linear transformation of h.

$$h \sim p(h) \qquad p(h) = \prod_i p(h_i)$$

$$x = Wh + b + \text{noise}.$$

Factor Analysis

latent variable is unit variance Gaussian.

$$h \sim N(h; 0, I)$$

$$x = Wh + b + \sigma z$$

Probabilistic PCA

most variance in data can be captured by h, up to small reconstruction error σ^2.

Independent component Analysis. (ICA)

Seeks to separate an observed signal into many underlying signals that are scaled & added together to form observed data.

to used to recover low level signals which are mixed together.
Many variants are possible.

Slow feature Analysis.

slowness is the idea that important characteristics of scenes change very slowly compared to individual measurement.

a term is added to C.F

$$\lambda \sum_t L(f(x^{(t+1)}), f(x^{(t)}))$$

f -> feature extractor.

Sparse coding.

It assumes that the linear factor have Gaussian noise with isotropic precision β
It's studied as an unsupervised feature learning & feature extraction.