Much research in deep learning involves building a probabilistic model of the input pmodel(x) which in principle can predict any of the variables in its environment given other variables

Many of these models also have latent variables h with pmodel(x) = Eh pmodel(x | h)

* This is the expected probability given h drawn from a distribution
* The latent variables h are sampled from a distribution and each unique combination of samples gives a different observation x

These latent vars provide another way of representing the data, and distributed representations based on latent vars can obtain all the benefits of representation learning

A linear factor model consists of a decoder f that generates x by adding noise to a linear transformation of h

A linear factor model breaks p(x) down into a joint distribution over explanatory latent variables

We sample explanatory factors h from a [factorial distribution](https://www.statisticshowto.datasciencecentral.com/factorial-distribution/) - where two variables don't interact at all -

(Condition for independence p(a, b) = p(a)p(b) )

The normal distribution for example: getting two means in a row is equal to p(𝝁)\* p(𝝁)

Sample real-valued observable variables given the factors: x = Wh + b + noise

* How do you know which latent variables correspond to which x?

The noise is independent across dimensions

Fig 13.1

Graphical model showing how the observed vector x is obtained by a linear combination of the independent latent variables plus some noise

**Summary - a linear factor model uses latent variables to generate a model for p(x), with the latent variables being sampled form an arbitrary factorial distribution**

**Probabilistic PCA and Factor Analysis**

PCA and factor analysis use the above equations but have a prior before x

Factor analysis has a latent prior where h is drawn from a Gaussian with unit variance: Note h is a latent variable vector

Q1: Why is a latent variable drawn from a distribution; shouldn't the vector have nonconstant values?

* It does have nonconstant values; each time h is drawn from the distribution, there is a new corresponding observation

The variables xi are assumed to be conditionally independent given h in factor analysis

Noise is drawn from the diagonal covariance Gaussian distribution with covariance matrix 

The vector of per-variable variances

The covariance of one variable is given by E[f(x) - E(f(x))]2 which equals sig2

The noise's covariance matrix means that it is drawn to account for the proper variance of each of the variables of x, where sigi2 corresponds to the variance of xi

Latent variables attempts to capture the dependencies between different variables xi

x is just a multivariate normal random variable with 

To cast PCA into a probablistic framework, modify the factor analysis model making the all of the conditional variances equal to each other

Now x just has a covariance of  where sig2 is a scalar, yielding the conditional distribution



where z is Gaussian noise: 

Probabilistic PCA takes advantage of the fact most variations in data can be captured by some latent variable h up to some residual reconstructor error sig2

As sig approaches 0, probabilistic PCA turns into regular PCA and the conditional expected value of h given x becomes x - b projected onto the column space of W

Model assigns very low likelihood if the data doesn't cluster near a hyperplane

**Summary - We have a diagonal covariance matrix, which makes sense since variables are independent from one another. x is drawn from a normal distribution, which is equivalent to matmul with h along with variance params. Probabilistic PCA gives common variance for every variable.**

**Independent Component Analysis**

Independent component analysis attempts to break down a signal into multiple underlying signals which are scaled and added together in order to form the observed data

These signals should be fully independent, rather than just decorrelated from one another

ICA method which is most similar to other generative models is a variant which trains a fully parametric generative model

The prior distribution over the underlying factors p(h) must be fixed ahead of time

The model can deterministically (non-randomly) generate x = Wh and then apply a nonlinear change of variables (like activations); then the model can be trained as normal using max likelihood

By choosing p(h) to be independent, we can recover the underlying factors which are independent (or at least essentially independent)

Each xi is one sensor's mixed signals, and hi is the estimate of one of the original signals

For example, if there are n different people speaking into n microphones, ICA can separate the signals so each hi only contains one person speaking clearly

Some ICA variants add noise in the decoder instead of making it deterministic

Other variants constrain the elements of h = W-1x to be independent from one another

Taking the determinant of W could be numerically unstable and expensive, however, so some models constrain W to be orthogonal

ICA requires p(h) be non-Gaussian since if it has a prior w/ Gaussian components, then W is not [identifiable](http://en.wikipedia.org/wiki/Identifiability) (it cannot be derived due to multiple parametrizations)

The same distribution can be obtained over p(x) for multiple values of W

Probablistic PCA and other linear factor models on the other hand requires p(h) to be Gaussian to guarantee that many operations have closed form solutions

In max likelihood where the user explicitly specifies the distribution, the typical choice is p(hi) = d(sig(hi)/dhi)

Non-gaussian distributions have more peaks near 0 than Gaussian distributions which lets ICA learn sparse features

Many ICA variants are not generative models in the sense of the phrase

A generative model either represents p(x) or can draw samples from it

ICA variants can transform between x and h, but do not have a representation for p(h), so they cannot form a distribution over p(x)

For example, many ICA variants try and increase the sample kurtosis of h = W-1x since it indicates a non-Gaussian p(h); however, p(h) is not explicitly represented

Just as PCA can be generalized to a nonlinear autoencoder, ICA can be generalized to a nonlinear generative model where a nonlinear function f generates the observed data

Nonlinear independent components estimation (NICE) stacks invertible transformations which have the property that the determinant of the Jacobian can be computed easily

The likelihood can be computed exactly, and just like ICA, the data is transformed into a space where it has a factorized marginal distribution (represented by the independent signals)

However, NICE is more likely to succeed due to a nonlinear encoder

Samples can be generated by sampling from p(h) and applying the decoder

ICA can also learn groups of features with statistical dependence allowed within a group but discouraged between groups

When groups of related units are chosen to be non-overlapping, this is independent subspace analysis

We can also have overlapping groups of spatially neighboring units, encouraging nearby units to learn similar transformations

Topographic ICA learns Gabor filters such that nearby features have the same orientation

**Summary - ICA trains a generative model which results in independent signals; it may or may not use an independent p(h), depending on the tradeoff between model complexity and independence between the derived underlying signals. Some ICA variants express the underlying signals as latent variables, and constrain them to be independent. ICA needs W to be identifiable (cannot have 2 different parametrizations which are equivalent), so it does not allow for Gaussian distributions**

**Also, the requirement of statistical independence means that 2 variables should not be correlated with the result in anyway, etc. so both should not have an effect on the input which depends on the other**

**Slow Feature Analysis**

Slow feature analysis is a linear factor model using info from time signals to learn invariant features

Slow feature analysis is based on the slowness principle which states that the important characteristics of scenes change very slowly relative to the individual measurements making up the scene

We may want to regularize our model to learn features which slowly change over time

Slowness principle can be applied to gradient-based learning by adding the following to the cost function:



ƛ is a hyperparameter, t is the index into the time sequence, f is a feature extractor to be regularized, and L is some loss (e.g MSE) measuring distance between f(xt+1) and f(xt)

SFA is efficient because when it is applied to a linear feature extractor, it can be trained in closed form

SFA is not a generative model, in the sense that it defines a mapping from the input to feature space, but does not have a prior over the feature space (h) and therefore doesn't impose a distribution p(x) over input space

SFA defines f(x; 𝚹) as a linear transformation and solves the optimization problem



Constrained to

, 

This means all expectations over time of f equal 0 and f2 equals 1

The constraint of the learned feature fi having a 0 mean is necessary to have 1 solution; otherwise, we can add a constant and get an equal value for the slowness feature objective

Having a unit variance is necessary to prevent the solution where features collapse to 0

To learn multiple features, we must also add the constraint 

This specifies the learned features must be linearly decorrelated from each other; otherwise, all the features would learn the slowest signal

We can learn nonlinear features through SFA by applying a nonlinear expansion to x before applying SFA (e.g we can transform the vector x into a vector which contains every xixj for all combinations of i and j)

We can build a deep nonlinear slow feature extractor by applying SFA and then a nonlinear basis expansion, and then keep on stacking SFAs and basis expansions

Deep SFA works well, as indicated by its ability to replicate biological neurons

We can predict the results of deep SFA if we know the dynamics of the input space (e.g the probability distribution of a video's angle)

Given how the underlying factors change, we can analytically solve for f\*

Deep SFA models have been shown to be able to recover these predicted functions

Other cost functions which depend on specific pixel values (which are highly variables) make it more difficult to predict what features the model will learn

The slowness prior may be too strong; rather than imposing a prior that the features are approximately constant, it may be better to impose a prior saying features should be easily predicted from one timestep to another

* How exactly does a prior work in this case? Does it give an initial probability distribution which would be too hard to modify, or is it just a constraint?
* Based on some research, having too strong a prior means that the initial / prior probability distribution is too hard to modify into the proper probability distribution

**Sparse Coding**

Sparse coding is a linear factor model which can be used as an unsupervised feature learning mechanism and as a feature extractor

Sparse coding atttempts to infer the value of h in the model while sparse modeling attempts to design and learn the model

Sparse coding assumes that the linear factors have Gaussian noise with isotropic precision B

Variance of one var is 𝜷

(Important: covariance matrix accounts for variance across the diagonal as well)

Recall isotropic distribution refers to the covariance matrix being equal to a scalar times IN

p(h) is chosen to be a distribution with peaks near 0; common choices are factorized Laplace, cauchy, or factorized student-t distributions

For example, Laplace in terms of the sparse penalty coefficient ƛ is and the student t-prior



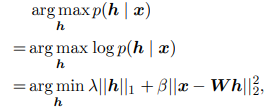
It is important to note that p(h) is its own distribution which generates the latent variables, while p(x | h) is dependent on h which is in turn dependent on the probability of h

Training sparse coding with max likelihood is intractable; instead, we use an encoder and then train the decoder to reconstruct the data given the encoding (this is an approx. to max likelihood)

In PCA, the parametric encoder function predicts h and consists of multiplication by a weight matrix; sparse encoding does not use a parametric encoder, but it is an optimization problem where we find the most likely code value



This yields the following optimization problem



Consider the L1 norm of h weighted by ƛ and the reconstruction error weighted by 𝜷

The L1 norm on h yields a sparse h\* (since norm of h isn't being squared, non-sparse values have a higher impact on the cost)

To train the model, we alternate between minimizing w.r.t h and w.r.t W

𝜷 is usually a hyperparameter with values set to 1, but it could be a learned parameter (if it is learned, then additional terms are there which need to be included)

Not all sparse encoding methods explicitly build a good p(h) or p(x | h); often, we just want a dictionary of features w/ activation values of 0 when extracted

* In other words, the practical uses based on the cost function when learning will disregard theoretical applications

If h is sampled from a Laplace prior, there is no chance of an element of h being 0

The generative model is not sparse, but the feature extractor is

The combinaiton of sparse coding wiht a non-parametric encoder (such as sampling from a distribution) can minimize the combination of reconstruction and log-prior error

Parametric encoders on the other hand, will generate a bad h if x is unusual, which can cause failure when creating an accurate reconstruction

When the inference problem is convex, the optimization problem will always find the optimal code when using sparse coding models

Sparsity and reconstruction costs may rise, but only due to the decoder's generalization error

This can result in better generalization when sparse coding is used as a feature extractor for a classifier than when a parametric function is used to predict the code (especially when there are very few labels)

Computing h given x with a non-parametric encoder takes a relatively long time since it uses an iterative process

Also, backprop is not straight forward with a nonparametric model, which makes unsupervised pretraining + fine tuning with supervised training hard

Sparse coding often results in poor reconstruction results since the factorial prior causes the model to have random subsets of all the features in each generated sample

This motivates a deeper model which can impose a non-facotrial distribution on the deepest code layer

Fig 13.2

Poor generations of MNIST digits

The model has learned to represent penstrokes and complete digits, but random subsets of features (which shouldn't go together to form a recognizable) are combined

**Manifold Interpretation of PCA**

Linear factor models such as PCA and factro analysis can be thought of as learning a manifold

Probabilistic PCA defines a thin-pancake like region of high probability - one which is very narrow among some axes (e.g vertically) but wide along others (horizontally)

PCA can be considered aligning this pancake-manifold with a linear manifold in a higher dimensional space

Fig 13.3

Shows a flat Gaussian capturing probability concentration near a low-dimensional manifold

The original 3D "pancake" has a plane which goes through it; this plane has a lower dimensionality, while the pancake has very small variance in the direction orthogonal to the plane (since this is how the ideal manifold plane is defined)

The variance in the direction orthogonal to the plane can be considered noise, while the variance in the directions captured by the plane could be considered useful signals

This applies to any linear autoencoder which learns matrices W and V to reconstruct x



In an autoencoder, the decoder computes the reconstruction

Linear encoder and decoder which minimize the erroroccur when V = W, 𝝁 = b = E[x], and when the column space of W forms an orthonormal basis with the same span as the eigenvectors of the covariance matrix:

Weights correspond to the directions where different variables move with each other

In PCA, the columns of W are eigenvectors ordered by the magnitude of their eigenvalues

The eigenvalue ƛi of C corresponds to the variance of x in the eigenvector direction vi

If x 𝟄 RD and h 𝟄 Rd and d < D, then the optimal reconstruction error is 

If the covariance matrix has rank d, the eigenvalues ƛd+1 to ƛD are 0 and the reconstruction error is D

The solution can also be achieved by maximizing variances of elements of h under an orthogonal W instead of minimizing reconstruction error