A good representation makes a learning task easier

Unsupervised data can be leveraged to get a good start on learning

**Greedy Layer-wise Unsupervised Pretraining**

A representation learned for one unsupervised task may be useful for a supervised task

Greedy layer-wise unsupervised pretraining relies on a single layer approach such an RBM or autoencoder with latent capabilities; each layer takes the output of the previous layer and produces a new representation, whose distribution is hopefully simpler

This was the first method which allowed fully-connected non-recurrent networks to be trained

The algorithm is greedy because it optimizes the network one piece at a time

It is called layer-wise because the individual pieces are layers

The k-th layer is trained while the previous ones are fixed; specifically, the lower layers' weights are fixed after the upper layers are introduced

Each layer is trained using an unsupervised representation learning algorithm

It is also called pre-training because the FC network is fine-tuned jointly afterwards

In the context of a supervised task, this process can be thought of as regularization or parameter initialization

Pretraining usually refers both to the pretraining and supervised learning phase

Supervised learning can simply introduce a classifier on top of the unsupervised features or it can refine the whole network to change the features entirely

Greedy layer-wise supervise pretraining is another option which works well due to the premise that shallow networks are easier to optimize than deep networks

*When and why does unsupervised pretraining work?*

Unsupervised pretraining performs inconsistently on many tasks, being useful for some and detrimental to others

It makes use of the idea that parameter initialization can have a significant impact on the model's regularization

Furthermore, it makes use of the idea that that learning about the input distribution can make the mapping from inputs to outputs easier

These only apply to unsupervised pretraining - other semi-supervised methods (such as combining autoencoders when calculating the loss, or semi-supervision within the network itself) do not necessarily use these principles

Parameter initialization may work well because it provides a good start into a region which would otherwise be inaccessible due to a noisy gradient estimation, or a region where the Hessian is poorly conditioned so grad. descent must take small steps

It is hard to tell which aspects of pre-trained parameters are retained, which is a motivation for using unsupervised and supervised learning simultaneously rather than two sequential stages

This issue may also be avoided by freezing learning of the feature extractor, and using supervised learning to only add a classifier on top of the learned features

The output constraints are naturally included from the start when using unsupervised training

* For example, the unsupervised training may provide linearly separable data for a linear classifier

Unsupervised pre-training can be useful for when the initial representation is poor

For example, in word embeddings, when using one-hot encoding, the L2 squared distance is always 2 between two distinct word vectors

However, when using distributed representations for the embedding, the distance is based on the similarity

Unsupervised pretraining is not as good for images, probably because the images are already in a rich vector space where distances provide little information about similarity

Unsupervised learning is effective when the number of training examples is small

It is also useful when the function is ver complicaated

Unsupervised learning is a different type of regularizer than weight decay because rather than biasing the parameters, it biases f(x) to make good use of the features already learned

If the true underlying features are shaped by patterns in the underlying input distribution, then unsupervised learning is the more appropriate regularizer

Unsupervised learning has been used to improve classifiers, and is best understood from a point of view of reducign test error

It also helps with overall optimization, reducing both training and test set error

Unsupervised training takes the parameters into a region which would be otherwise inaccessible

NN training is non-deterministic and differs every tme the algorithm is run

Training may hault due to the gradient approaching 0, or to early stopping, or due to a large gradient which cannot be improved due to a poorly conditioned Hessian

Pretraining consistently brings the parameters into a certain region, while no pretraining consistently brings it into another region

The pretraining region of possibilities is smaller, indicating that the variance is less when using unsupervised pretraining causing more consistent (and less very bad) results

Pretraining for deep networks best reduces the mean and variance of the test set error

An important question is how pretraining works as a regularizer

One hypothesis for why pretraining works well with deep networks is that it discovers the underlying causes of the observed data

Since there are 2 training phases, pretraining and then supervised learning, it is basically impossible to predict the effects of unsupervised training; there are very many hyperparameters due to the pre-training layers whose regularization strength is intractable to choose

In contrast, a normal regularizer would have one penalty hyperparameter coefficient

Training with both unsupervised and supervised training simultaneously, however, is straightforward

The loss can depend on both the unsupervised and supervised loss, with a coefficient affecting how much the objective function is regularized

In the case of unsupervised pre-training, there is no flexible way of adapting regularization - either the supervisd model is initialized to pretrained parameters, or it is not

Another disadvantage is that each phase has its own hyperparameters

Training and choosing hyperparameters for the supervised phase cannot proceed until the hyperparameters have been chosen for the unsupervised phase, and until after the unsupervised phase is done (which may be a long delay)

The most principled way to choose the hyperparameters for the unsupervised phase is using validation error

In practice, certain unsupervised hyperparameters such as the number of pre-training iterations can be chosen by applying early stopping; while it is not ideal, it is cheaper than using the supervised objective

Unsupervised pre-training has largely been abandoned except for natural language tasks where OHE vectors do not provide much info about distance, and a good representation can be learned for a huge dataset before supervised training is applied

Unsupervised pretraining is particularly effective because it can be applied to a huge dictionary of words to generate embeddings whihc can then be fine-tuned for a supervised task where there are fewer examples

On large datasets, and medium (around 5,000) examples, batch normalization and dropout tend to outperform unsupervised learning

Also when the number of training examples is very small, Bayesian methods tend to output unsupervised learning

For these reasons, the popularity of unsupervised larning has declined

Supervised pre-training can be applied in the form of transfer learning, where the weights for related tasks are re-used

**Transfer Learning and Domain Adaptation**

In transfer learning, the factors that explain variation in the distribution P1 are relevant to the variations needed to understand P2

This makes the assumption that P1 and P2 are at least loosely related

In particular, this works well in supervised learning tasks with different targets

When the number of training examples for the second task is much less than for the first task, the information learned about the distribution P1 can help the model generalize about P2 very quickly with just a few examples

This is because there are low-level things shared in common which are useful for both tasks

Basically, the model learns underlying factors which are used in more than one setting (both distributions), but has more training data to do so if task 1's data is used

Sometimes what is shared about different tasks is the semantics of the output

For example, a speech recognition algorithm may need to recognize the same phonemes which different people pronounce very differently

In this case, it makes sense to share the upper layers of the neural networks and use the lower levels for task specific pre-processing

Fig 15.2

Shows an example of how the lower level translates the task-specific input into a set of generic features which can then be used to produce the corresponding labels at a higher level

In domain adaptation, the task remains the same between each setting but the input distribution slightly changes

For example, in sentiment analysis, where reviews are classified as positive or negative, a model w hich is trained on reviews about books may be used to predict reviews about TV shows

It may be hard to generalize due to the different vocabulary / style in the different domains

One solution which has worked well is unsupervised pretraining with denoising autoencoders

A related problem is oncept drift where the input distribution changes over time

In general, these kinds of tasks are called "multi-task learning"

In ML competitions, usually a transformation of the distribution P1 is learned to transform it into a representation space; the same transformation is applied to P2 which then adds a simple classifier on top of the representation of P2

If multi-task learning was successful, then this could be achieved with few labeled examples

Deeper architectures when generating the unsupervised hidden space results in a better learning curve for new categories of the new distribution P2

Furthermore, an increasing depth requires fewer examples

One shot learning uses unsupervised learning to separate the underlying causes of P1 and then uses one example of the transfer task P2 to refine it

Zero shot learning, or zero data learning, does not give any examples from P2

One shot learning works because all of the data clusters around the sample when converted to representation space

This works when the factors of variation have been isolated in the representation space and we have learned which factors matter

An example of zero-shot learning is attempting to detect an object you've never seen visually based off a description of it

Zero data learning can be thought of with 3 variables: x, y, and T which is a variable describing the task

The model estimates the conditional distribution P(y | x, T) where T is a task such as answering questions about the image (is a cat in this picture?)

Zero shot learning requires T to be represented in a way that allows for some sort of generalization; we cannot use OHE and must use embeddings

In machine translation, we can apply 0 shot learning (or at least something similar)

Given two languages X and Y, we can find distributed representations for them in their own language representation space, and then create a link relating the two spaces

This link can be trained based on matching sentences in the different languages

This architecture works best if all 3 components (the link and representations) are learned jointly

The same principle can explain how one can perform multi-modal learning: learning a representation in one modality x, learning another representation in modality y, and then learning a mapping from x to y and finally generalizing on new pairs

Fig 15.3

Shows a mapping from x to hx and from y to hy

We see bidirectional mappings between hx and hy

Now one can associate an image xtest to a word ytest even if no image of that word was presented during training, since there is a new generic representation in the transformed space

Their respective feature vectors have been related to each other

**Semi-Supervised Disentangling of Causal Factors**

One hypothesis is that an ideal representation captures the underlying causes of x, with different directions / features corresponding to different causes

This representation will be effective when computing p(y | x) if y is a strong cause of x

This type of representation may be hard to achieve

Generally, capturing the underlying causes of x coincides with siolating the attributes

If h captures the underlying causes of x, it makes it easy to predict y

However, unsupervised learning of p(x) may result in no information about p(y|x)

Suppose we are trying to learn when p(x) is uniformly distribute

Observing a set of x training values gives no info about the modeling goal

Consider the situation where x arises from a mixture, with one mixture component (e.g object classes in image data) per y-value

Fig 15.4

Shows an example of a density over x which is a mixture over 3 components

The component factor is an underlying explanatory factor, y

Since the mixture components are statistically visible, modeling p(x) in an unsupervised manner without a labeled example reveals the factor y

If the mxiture components are well separated, then modeling p(x) reveals exactly where the mixture component is located

Then, we can train the model with just one example (the features for that object will already be well known, we only need the training example to actually get the corresponding)

If y is related to the causal factors of x, then p(y | x) and p(x) are tied together, and unsupervised learning works well

Suppose y is a causal factor or at least very closely related to them, and h generated from x is a representation which captures all causal factors (including y)

Then, the generative process can be structured using this graphical model with 

The data, as a result, has marginal probability



The best model of x is then a latent representation h which represents variation in x

The ideal representation should be able to recover the latent representation

If y is one of these latent factors or related to them, it should be easy to predict

Bayes rule shows how the conditional distribution of y given x is tied to the components in the other equations



In situations where the relationship assumptions are true, semi-supervised learning works well

If the number of underlying causes is very large (which it always is), and y is only one underlying cause hi , then the brute force method of disentangling h to figure out which cause is correct is impractical

Instead, we must figure out what to encode, which can be done in 2 ways:

* Use a supervised signal simultaneously during unsupervised training to capture only the most important features
* Have a larger representation

To find out what to encode, we need to figure out what is salient

Autoencoders do this well, due to their limited capacity with a need to capture significant features as required by the cost function

Fig 15.5

Shows the autoencoder deciding not to reproduce a ping pong ball due to limited capacity

If a group of pixels form a recognizable pattern, then that may be considered salient

These patterns may be captured by a generative adversarial network which attempts to fool a feed forward classifier (which attempts to distinguish real and fake)

If the FFN recognizes a pattern, then it can be considered "salient"

Fig 15.6

Shows how using a GAN may be beneficial since using an autoencoder with MSE does not reconstruct ears, while using a GAN does reconstruct ears do to a different criterion

Learning the underlying factors with x as an effect and y as a cause will make p(x | y) robust to changes in y

This is effective since in different domains the causal mechanisms remain invariant while only the marginal distribution changes

Therefore, better generalization by a model which attempts to recover h and p(x | h) can result in better generalization

**Distributed Representations**

Concepts are components of a representation whose elements can be set separately from one another

Distributed representations can use n features with k values to represent kn unique concepts

Each direction in the representation space can correspond to a different underlying variable

A vector of n binary features can have 2n representations

In a symbolic representation, n symbols have n detectors, one for each

* There are n representations of the input space
* This can be called a one-hot representation since it can be captured by a binary vector with n mutually exclusive bits (only one active bit at a time)

Non-distributed learning algorithms include clustering, KNN, decision trees, Gaussian mixtures, kernel machines, and language / n-grams where the set of contexts (sequence of symbols) is partitioned according to a tree structure of suffixes

While each input may be represented with multiple values for some of these, these values cannot be controlled separately from oen another

For some of those non-distributed algorithms, the output can be interpolated but the number of parameters remains linearly proportional to the number of regions, so we consider it symbolic

Another property of distributed representations is the ability to generalize via shared features; embeddings create a sort of "similarity space", while one-hot representations do not

A distributed representation can have a statistical advantage when a complicated structure needs to be learned, but only with a few parameters

Most non-distributed algorithms benefit only due to the smoothness assumption which is that if u ~= v, then f(u) ~= f(v)

The end result is that if f is our model, then f(x) ~= y ~= f(x + 𝟄)

This causes the curse of dimensionality where the number of examples must be at least as large as the dimensionality of our data

Having a separate degree of freedom for each region in this case is useful for mapping from regions of symbols to a value, but gives no ability to generalize to new symbols for new regions

If we are lucky, there is some regularity in the target function besides being smooth

For example, CNNs have the property of spatial invariance even though this corresponds to a non-smooth transformation in the input space of x

Consider a distributed representations case which extracts binary features by thresholding linear functions of the input

Each binary input divides Rd into half-spaces

The exponentially large number of intersections (w.r.t # of half-spaces) of n of the half-spaces determines how many regions this distributed representation can distinguish

The number of regions formed by the intersection of n hyperplanes in Rd is given by



Therefore, we see a growth which is polynomial in the number of hidden units and exponential in the input size

With O(nd) parameters for n linear threshold features in Rd we can represent nd regions in input space

If we used a representtion with one unique symbol for each region with separate parameters, then O(nd) regions would require O(nd) examples

If a parametric transformation with k parameters can learn about r regions where k << r, then we can probably generalize better than using non-distributed representations which would require O(r) examples to obtain the same partitioning into r regions

Using fewer parameters means having fewer parameters to fit the model with, which requires far fewer training examples to generalize well

Despite being able to capture many regions, models based on distributed representations have a limited capacity, which can cause them to perform well

The [VC dimension](https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis_dimension) of a NN of linear threshold units is O(w log w), where w is the # of weights

This limitation arises since while we can use many unique codes in the representation space, we cannot use the entire code space, or learn an arbitrary mapping from h to y with a linear classifier

Therefore, using a prior distribution and a linear classifier expresses a prior belief that the classes are linearly separable as a function of the underlying causes h

These ideas can be seen experimentally in a CNN where the higher layers correspond to high level features that even humans would associate between (e.g cat to whiskers)

One advantage these features have is that it is possible to learn about each one of them without learning about all configurations of the other features

Gradint descent will find useful features on its own; for example, the directions in the representation space capturing factors of variations in a face recognition task will correspond to features such as male or female

The property of being able to learn about one feature without the others is known as statistical separability which allows to generalize to examples not seen during training

**Exponential Gains from Depth**

While the universal approximation theorem states that a shallow NN can approximate any function, a deep network can do so with fewer neurons

This is because higher level features correspond to higher level layers of the network, and these higher level layers require a large depth with stacked nonlinearities

Higher level features tend not to be linearly related to the input

A hierarchy of re-used features with many nonlinearities can give an exponential boost in statistical efficiency

A sum product network uses polynomial circuits to compute the probability distributions over a set of random variables

It has been shown for SPNs that a minimum depth is needed in order to avoid exponential computations

Also, in order to make an SPN tractable at a lower depth (tractable here means work the same as a higher depth), it comes at the cost of representational power

A shallow circuit which approximates a deep circuit cannot do so as well (it has less representational power than the deep circuit)

**Providing Clues to Discover Underlying Causes**

Supervised learning provides a strong clue - a label y presented with each x specifies the value of one of the factors of variation directly

Representation learning makes use of other less direct hints about the underlying factors

These hints take the form of implicit priors we impose as the creators of the learning algorithm

Regularization is needed for generalization, and one of the goals of deep learning is to find widely applicable regularization strategies

Some generic regularization strategies:

* Smoothness: f(x + 𝟄d) =~ f(x) => allows for generalization through interpolation
* Assumption of linearity between variables allows the model to make predictions far from the rest of the data, but may sometimes be mutually exclusive with smoothness
* Multiple explanatory variables, which are represented by h; these is useful since p(x) and p(y | x) can learn something from this space, providing a motivation for semi-supervised learning
* Causal factors h (we assume h is a cause for x), which is good for when the distribution over h changes or we need to apply the model to a new task
* Shared factors among tasks: if we have yi different tasks which each have the same global input x, then the yi tasks are different subsets of the underlying factors h; due to overlap between these subsets, having a shared intermediate representation P(h | x) allows for statistical strength
* Manifolds: probability mass concentrates to form locally connected regions which occupy a small volume; a manifold is an instance of this with a much lower dimensionality, which many models try and learn (e.g autoencoders)
* Natural clustering: assumption that each connected manifold in the input space is assigned to a single class; class remains constant on each one of the disconnected manifolds
* Temporal and spatial coherence: most important explanatory factors change slowly over time, or at the very least, it is easier to predict the explanatory factors h than raw inputs
* Sparsity: most features are not relevant to describing the inputs since there are so many features; therefore, if a feature can be classified as "present" or "absent", it should most of the time be classified as absent
* Simplicity of factor dependencies: in good high-level representations, factors are related to each other through simple dependencies => marginal independence; this is assumed when plugging a linear predictor on top of a distributed representation

(Marginal independence)