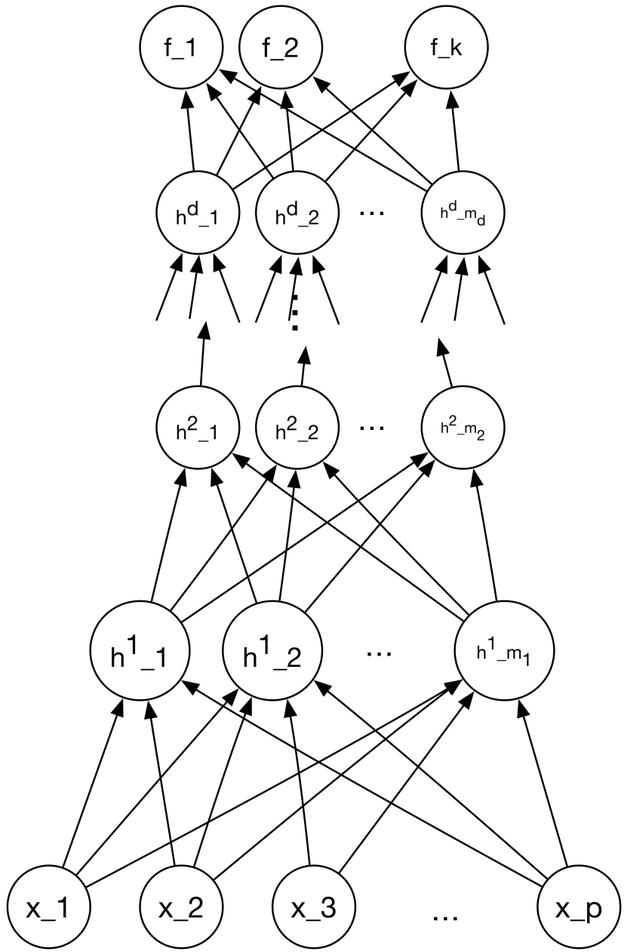
Deep Learning

Héctor Corrada Bravo

University of Maryland, College Park, USA

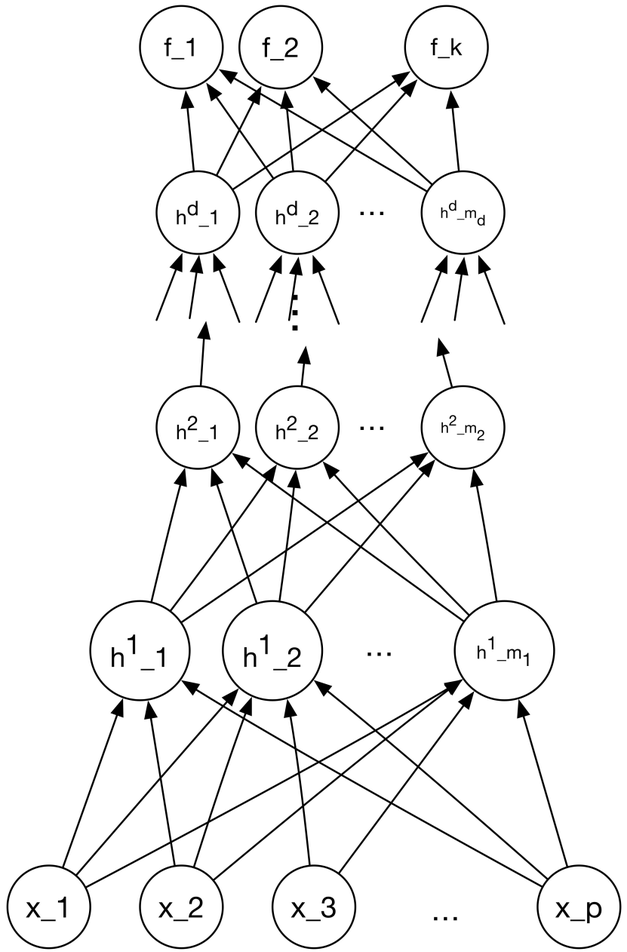
CMSC 643: 2017-11-07

Deep Feed-Forward Neural Networks



The general form of feed-forward network can be extended by adding additional hidden layers.

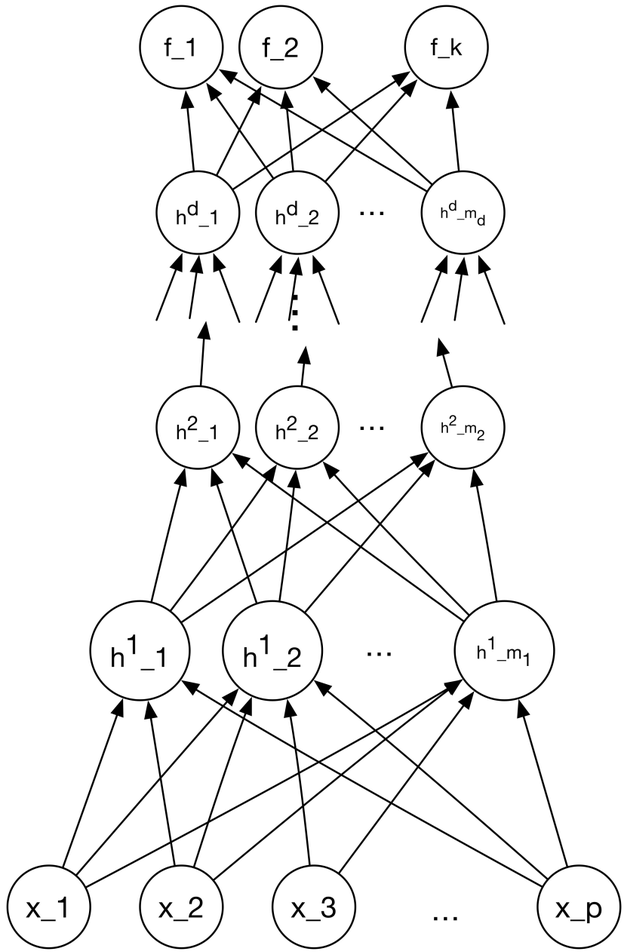
Deep Feed-Forward Neural Networks



The same principles we saw before:

- We arrange computation using a computing graph
- Use Stochastic Gradient Descent
- Use Backpropagation for gradient calculation along the computation graph.

Deep Feed-Forward Neural Networks

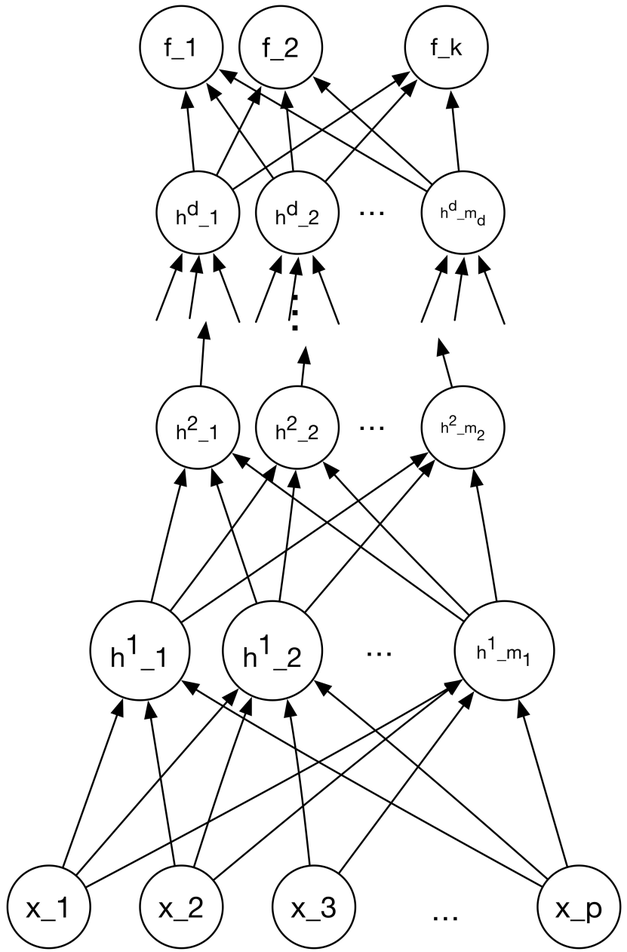


Empirically, it is found that by using more, thinner, layers, better expected prediction error is obtained.

However, each layer introduces more linearity into the network.

Making optimization markedly more difficult.

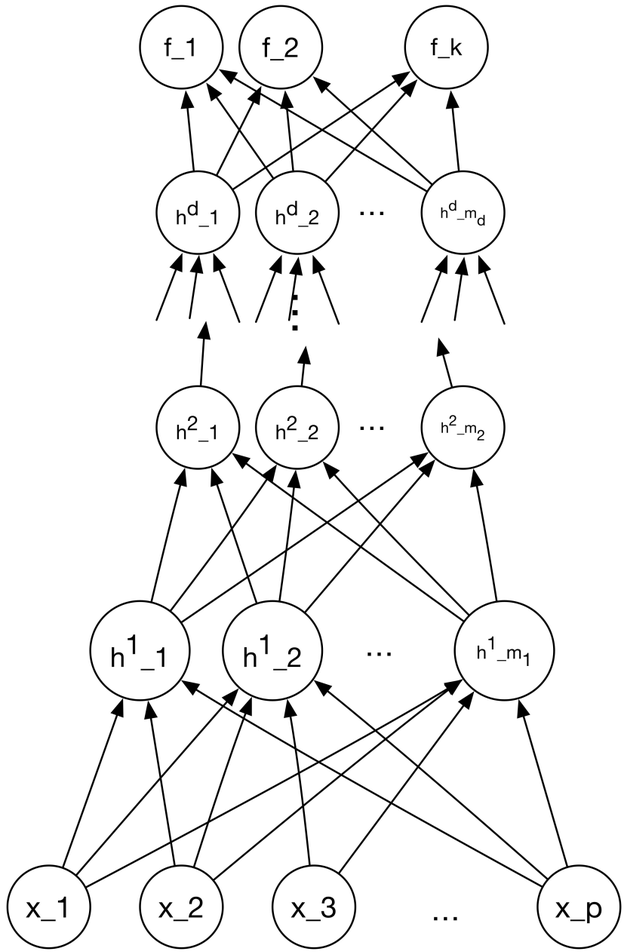
Deep Feed-Forward Neural Networks



We may interpret hidden layers as progressive derived representations of the input data.

Since we train based on a loss-function, these derived representations should make modeling the outcome of interest progressively easier.

Deep Feed-Forward Neural Networks



In many applications, these derived representations are used for model interpretation.

Deep Feed-Forward Neural Networks

Advanced parallel computation systems and methods are used in order to train these deep networks, with billions of connections.

The applications we discussed previously build this type of massive deep network.

Deep Feed-Forward Neural Networks

Advanced parallel computation systems and methods are used in order to train these deep networks, with billions of connections.

The applications we discussed previously build this type of massive deep network.

They also require massive amounts of data to train.

Deep Feed-Forward Neural Networks

Advanced parallel computation systems and methods are used in order to train these deep networks, with billions of connections.

The applications we discussed previously build this type of massive deep network.

They also require massive amounts of data to train.

However, this approach can still be applicable to moderate datasizes with careful network design, regularization and training.

Regularization of Deep NNs

Regularization by penalizing parameter norm is frequently used in these cases.

The obective function to minimize in this case is

$$J(\theta; X, y) = L(\theta; X, y) + \lambda\Omega(\theta)$$

Regularization of Deep NNs

Regularization by penalizing parameter norm is frequently used in these cases.

The obective function to minimize in this case is

$$J(\theta; X, y) = L(\theta; X, y) + \lambda\Omega(\theta)$$

- θ includes all parameters of the network
- L is a task appropriate loss function (e.g. least squares for regression)
- Ω is a penalty function on the parameters

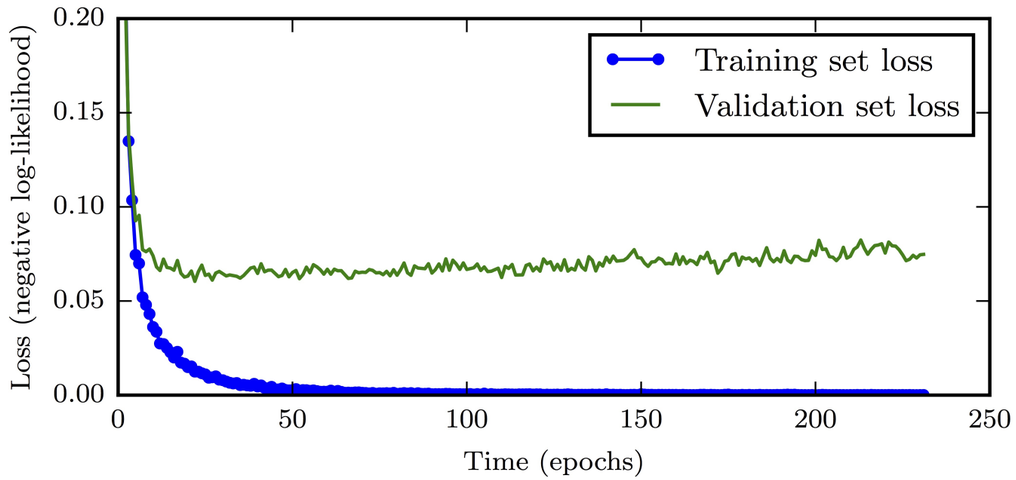
Regularization of Deep NNs

Commonly used functions are

- L2 normalization, $\Omega(\theta) = \sum w_{kl}^2$ for all weight parameters w_{kl}
- L1 normalization, $\Omega(\theta) = \sum |w_{kl}|$ for all weight parameters w_{kl}

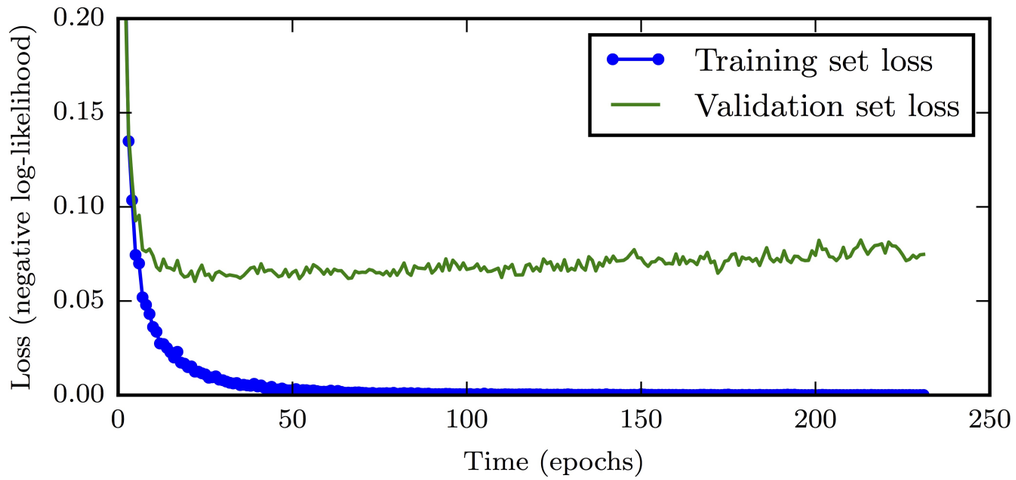
In both of these cases, SGD is easily adapted.

Early Stopping



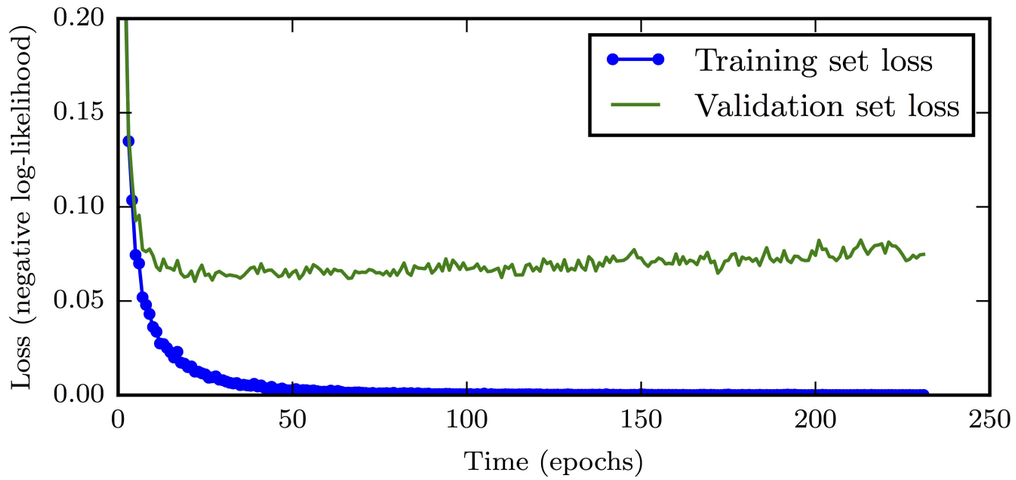
Another technique frequently used to regularize models is early stopping.

Early Stopping



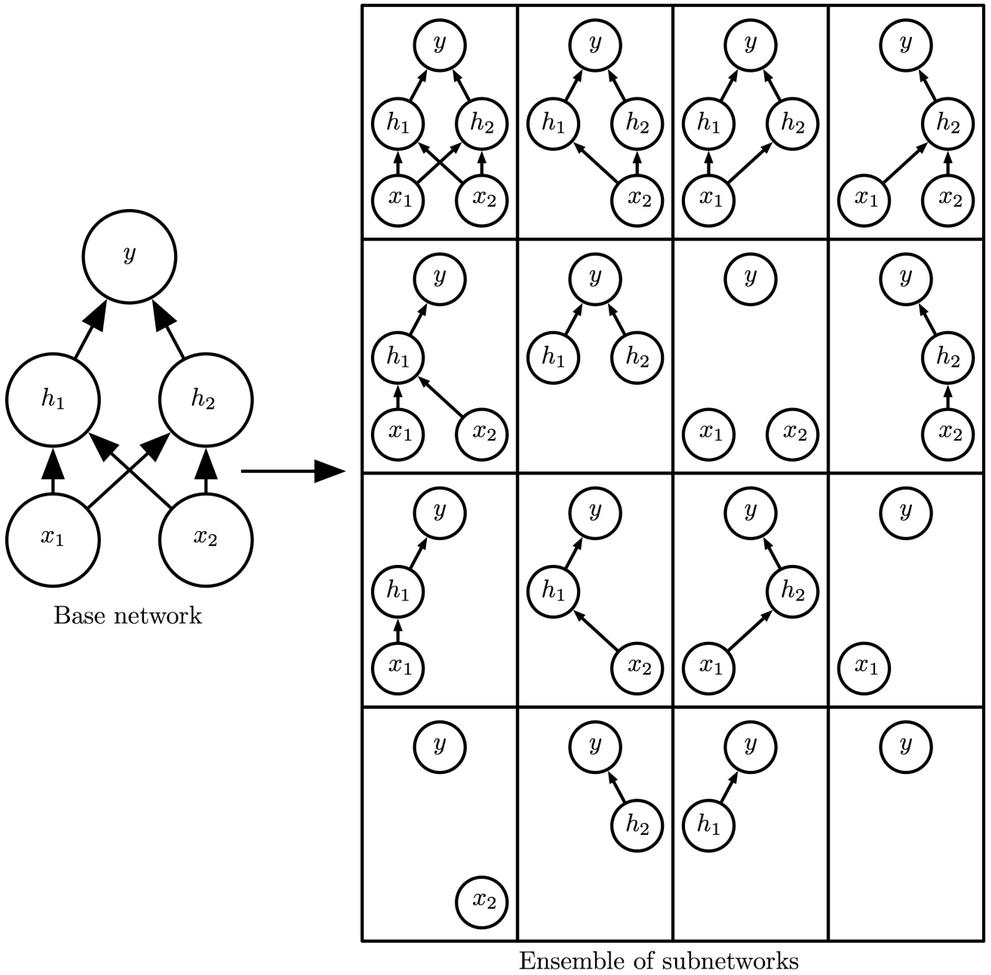
This is based on the empirical observation that as more training is done, overfitting becomes worse.

Early Stopping



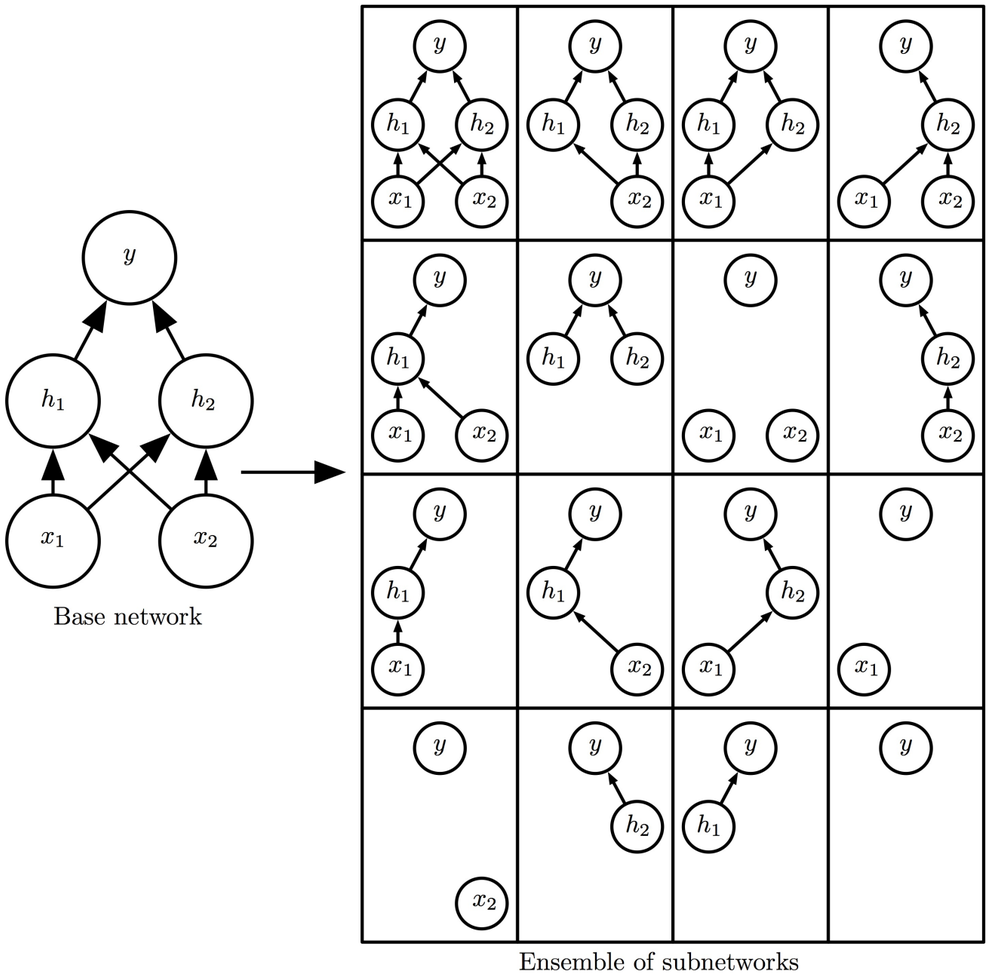
There are meta-algorithms designed to determine when to stop training based on improvement of expected prediction error, and the rate at which parameter models change between training iterations.

Dropout



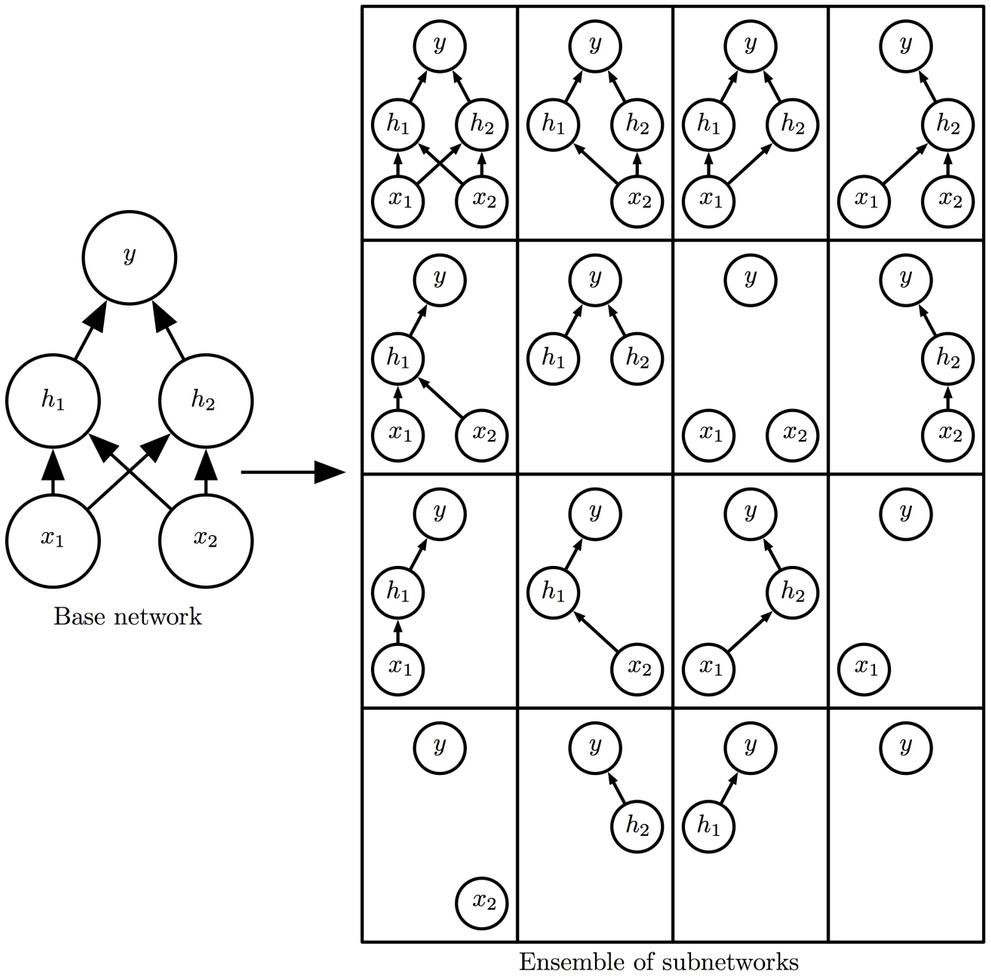
A popular method for regularization, that also addresses the multiple minimum problem is dropout.

Dropout



Bagging is used to build an ensemble of specifically constructed networks.

Dropout



In this case, subnetworks of the network being trained are selected randomly.

Each network is trained independently on a bootstrap sample of data.

Averaging is used to combine predictions.

Long-term Dependencies

A significant issue in training deep networks.

As deeper networks are used, multiplication of gradients causes major issues.

Long-term Dependencies

Consider the case of a computational graph where a weight matrix W is repeatedly multiplied.

Long-term Dependencies

Consider the case of a computational graph where a weight matrix W is repeatedly multiplied.

Suppose matrix W has eigen-value decomposition $W = V\text{diag}(\lambda)V'$

Long-term Dependencies

Consider the case of a computational graph where a weight matrix W is repeatedly multiplied.

Suppose matrix W has eigen-value decomposition $W = V\text{diag}(\lambda)V'$

After t steps, we obtain matrix $W^t = V\text{diag}(\lambda)^t V'$

Long-term Dependencies

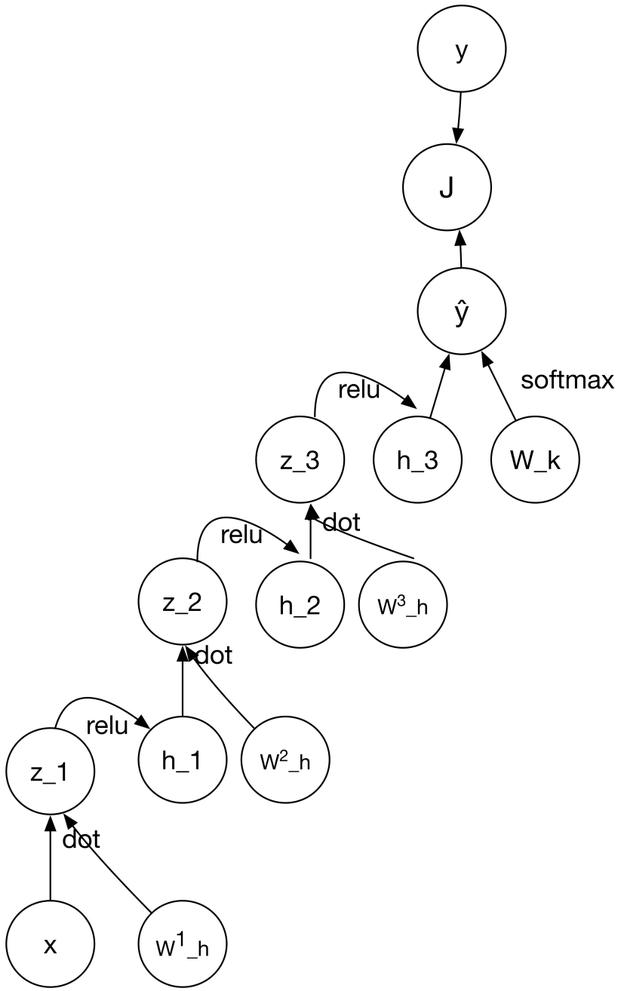
Consider the case of a computational graph where a weight matrix W is repeatedly multiplied.

Suppose matrix W has eigen-value decomposition $W = V\text{diag}(\lambda)V'$

After t steps, we obtain matrix $W^t = V\text{diag}(\lambda)^t V'$

What do we expect to happen to weights?

Supervised Pre-training

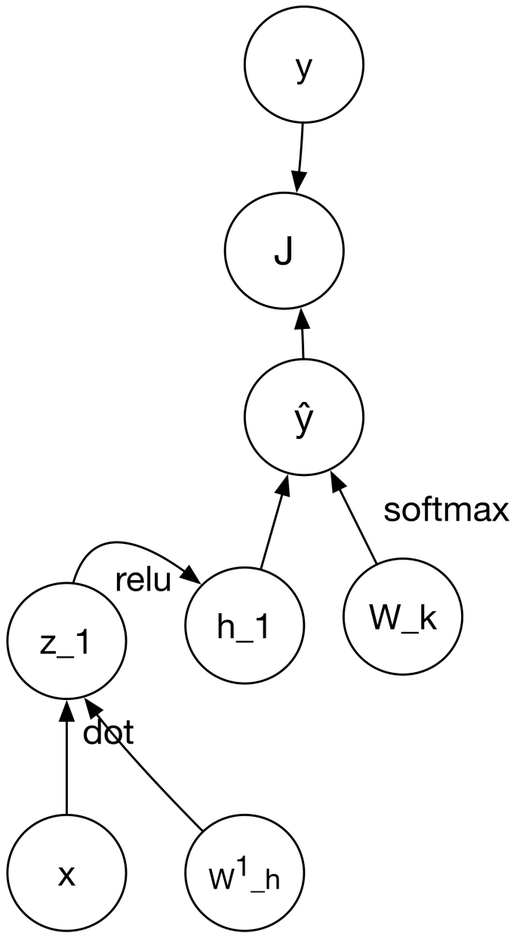


A clever idea for training deep networks.

Train each layer successively on the outcome of interest.

Use the resulting weights as initial weights for network with one more additional layer.

Supervised Pre-training

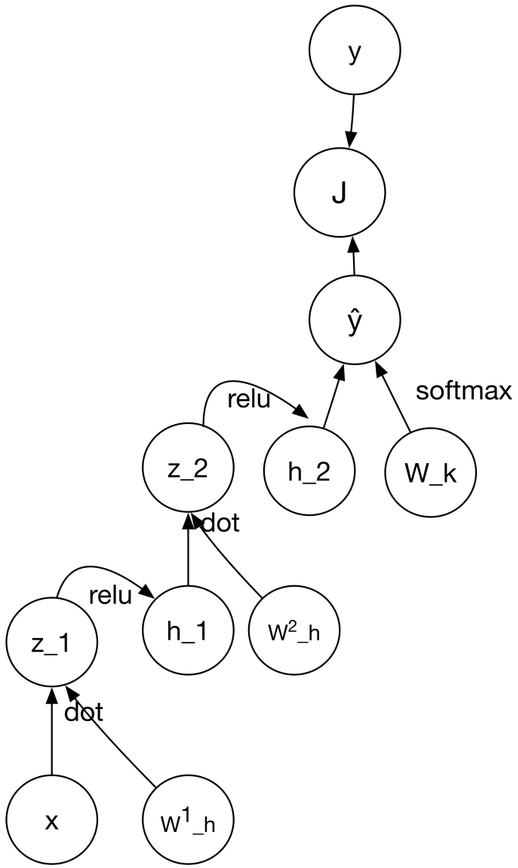


Train the first layer as a single layer feed forward network.

Weights initialized as standard practice.

This fits W_h^1.

Supervised Pre-training



Now train two layer network.

Weights W_h^1 are initialized to result of previous fit.

Supervised Pre-training

This procedure continues until all layers are trained.

Hypothesis is that training each layer on the outcome of interest moves the weights to parts of parameter space that lead to good performance.

Minimizing updates can ameliorate dependency problem.

Supervised Pre-training

This is one strategy others are popular and effective

- Train each layer as a single layer network using the hidden layer of the previous layer as inputs to the model.
- In this case, no long term dependencies occur at all.
- Performance may suffer.

Supervised Pre-training

This is one strategy others are popular and effective

- Train each layer as a single layer on the hidden layer of the previous layer, but also add the original input data as input to every layer of the network.
- No long-term dependency
- Performance improves
- Number of parameters increases.

Parameter Sharing

Another method for reducing the number of parameters in a deep learning model.

When predictors x exhibit some internal structure, parts of the model can then share parameters.

Parameter Sharing

Two important applications use this idea:

- Image processing: local structure of nearby pixels
- Sequence modeling: structure given by sequence

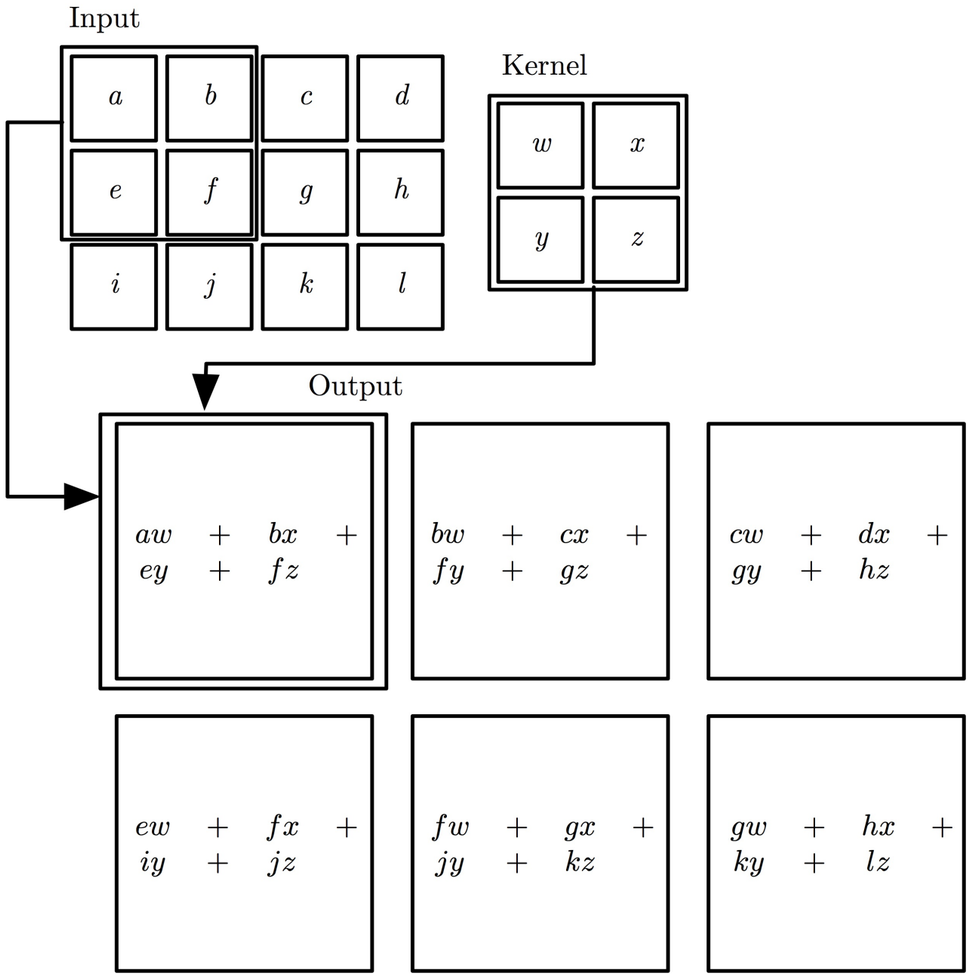
The latter includes modeling of time series data.

Parameter Sharing

Convolutional Networks are used in imaging applications.

Input is pixel data.

Parameters are shared across nearby parts of the image.

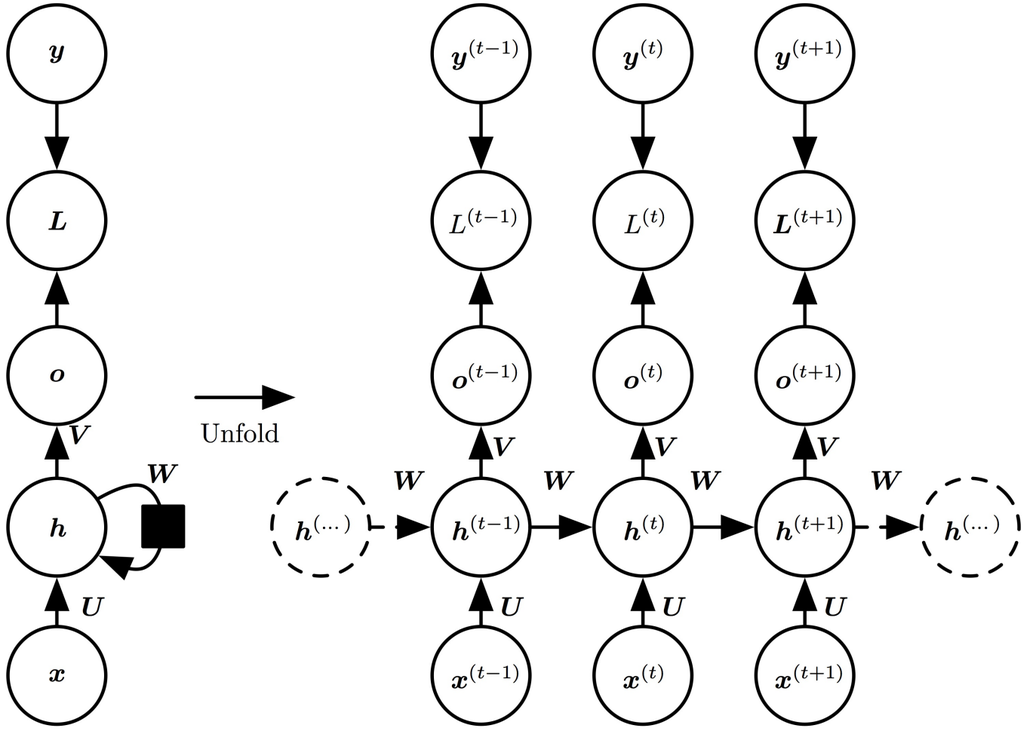


Recurrent Networks

Recurrent Networks are used in sequence modeling applications.

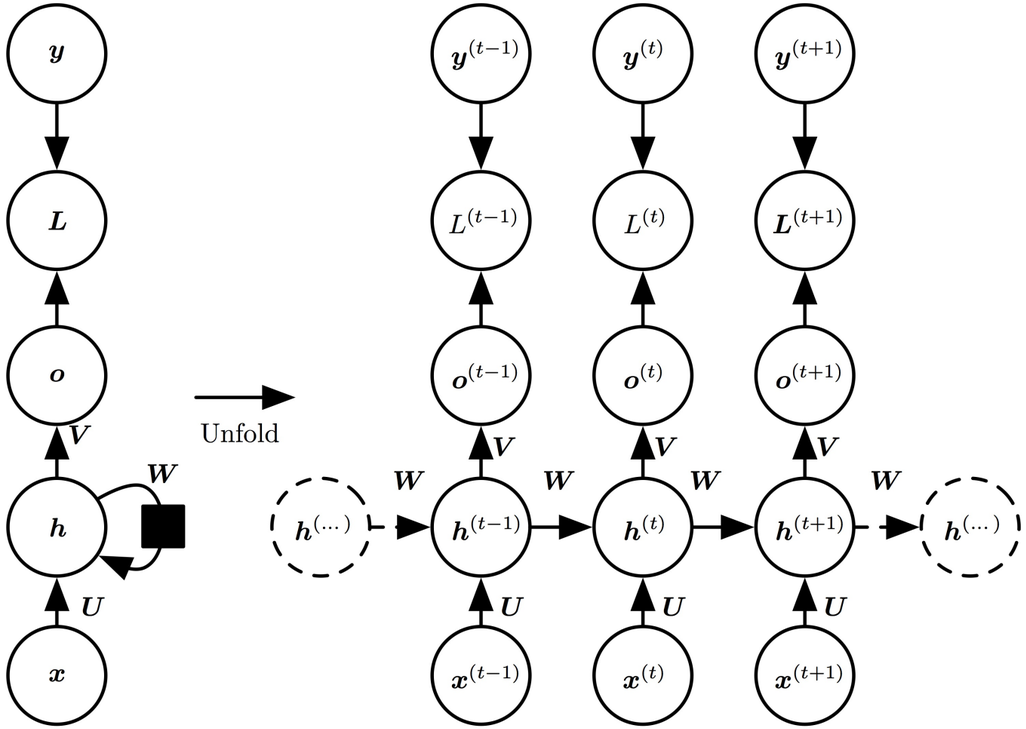
For instance, time series and forecasting.

Parameters are shared across a time lag.



Recurrent Networks

The long short-term memory (LSTM) model is very popular in time series analysis



Summary

Deep Learning is riding a big wave of popularity.

State-of-the-art results in many applications.

Best results in applications with massive amounts of data.

However, newer methods allow use in other situations.

Summary

Many of recent advances stem from computational and technical approaches to modeling.

Keeping track of these advances is hard, and many of them are ad-hoc.

Not straightforward to determine a-priori how these technical advances may help in a specific application.

Require significant amount of experimentation.

Summary

The interpretation of hidden units as representations can lead to insight.

There is current research on interpreting these to support some notion of statistical inference.

Excellent textbook: http://deeplearningbook.org