Artificial Neural Networks II

STAT 27725/CMSC 25400: Machine Learning

Shubhendu Trivedi

University of Chicago

November 2015

- Things we will look at today

- Things we will look at today
  - Regularization in Neural Networks

- Things we will look at today
  - Regularization in Neural Networks
  - Drop Out

- Things we will look at today
  - Regularization in Neural Networks
  - Drop Out
  - Sequence to Sequence Learning using Recurrent Neural Networks

- Things we will look at today
  - Regularization in Neural Networks
  - Drop Out
  - Sequence to Sequence Learning using Recurrent Neural Networks
  - Generative Neural Methods

A Short Primer on Regularization: Empirical Risk

- Assume that the data are sampled from an unknown distribution $p(\mathbf{x}, y)$

A Short Primer on Regularization: Empirical Risk

- Assume that the data are sampled from an unknown distribution $p(\mathbf{x}, y)$
- Next we choose the loss function L, and a parametric model family $f(\mathbf{x}; \mathbf{w})$

A Short Primer on Regularization: Empirical Risk

- Assume that the data are sampled from an unknown distribution $p(\mathbf{x}, y)$
- Next we choose the loss function L, and a parametric model family $f(\mathbf{x}; \mathbf{w})$
- Ideally, our goal is to minimize the *expected loss*, called the *risk*

$$R(\mathbf{w}) = \mathbb{E}_{(x_0,y_0)\sim p(x,y)}[L(f(x_0;\mathbf{w}), y_0)]$$

A Short Primer on Regularization: Empirical Risk

- Assume that the data are sampled from an unknown distribution $p(\mathbf{x}, y)$
- Next we choose the loss function L, and a parametric model family $f(\mathbf{x}; \mathbf{w})$
- Ideally, our goal is to minimize the *expected loss*, called the *risk*

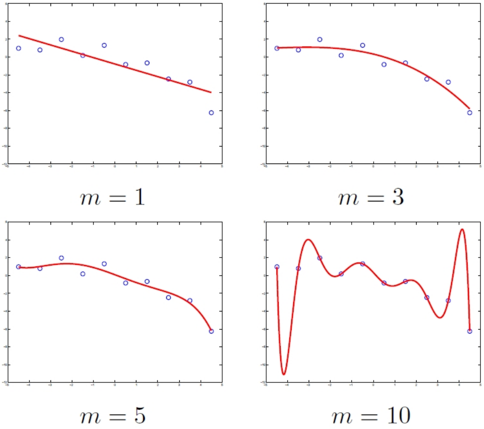
$$R(\mathbf{w}) = \mathbb{E}_{(x_0,y_0)\sim p(x,y)}[L(f(x_0; \mathbf{w}), y_0)]$$

- The true distribution is unknown. So, we instead work with a proxy that is measurable: Empirical loss on the training set

$$L(\mathbf{w}, X, y) = \frac{1}{N}\sum_{i=1}^{N} L(f(x_i; \mathbf{w}), y_i)$$

Model Complexity and Overfitting

Consider data drawn from a 3rd order model:



How to avoid overfitting?

- If a model overfits (is too sensitive to the data), it would be unstable and will not generalize well.

How to avoid overfitting?

- If a model overfits (is too sensitive to the data), it would be unstable and will not generalize well.
- Intuitively, the complexity of the model can be measured by the number of "degrees of freedom" (independent parameters) (previous example?)

How to avoid overfitting?

- If a model overfits (is too sensitive to the data), it would be unstable and will not generalize well.
- Intuitively, the complexity of the model can be measured by the number of "degrees of freedom" (independent parameters) (previous example?)
- Idea: Directly penalize by the number of parameters (called the Akaike Information criterion): minimize

$$\sum_{i=1}^{N} L(f(x_i; \mathbf{w}), y_i) + \#\text{params}$$

Description Length

- Intuition: Should not penalize the parameters, but the number of bits needed to encode the parameters

Description Length

- Intuition: Should not penalize the parameters, but the number of bits needed to encode the parameters
- With a finite set of parameter values, these are equivalent. With an infinite set, we can limit the effective number of degrees of freedom by restricting the value of the parameters.

Description Length

- Intuition: Should not penalize the parameters, but the number of bits needed to encode the parameters
- With a finite set of parameter values, these are equivalent. With an infinite set, we can limit the effective number of degrees of freedom by restricting the value of the parameters.
- Then we can have Regularized Risk minimization:

$$\sum_{i=1}^{N} L(f(x_i; \mathbf{w}), y_i) + \Omega(w)$$

Description Length

- Intuition: Should not penalize the parameters, but the number of bits needed to encode the parameters
- With a finite set of parameter values, these are equivalent. With an infinite set, we can limit the effective number of degrees of freedom by restricting the value of the parameters.
- Then we can have Regularized Risk minimization:

$$\sum_{i=1}^{N} L(f(x_i; \mathbf{w}), y_i) + \Omega(w)$$

- We can measure "size" in different ways: L1, L2 norms

Description Length

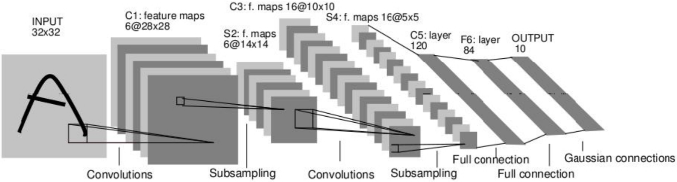
- Intuition: Should not penalize the parameters, but the number of bits needed to encode the parameters
- With a finite set of parameter values, these are equivalent. With an infinite set, we can limit the effective number of degrees of freedom by restricting the value of the parameters.
- Then we can have Regularized Risk minimization:

$$\sum_{i=1}^{N} L(f(x_i; \mathbf{w}), y_i) + \Omega(w)$$

- We can measure "size" in different ways: L1, L2 norms
- Regularization is basically a way to implement Occam's Razor

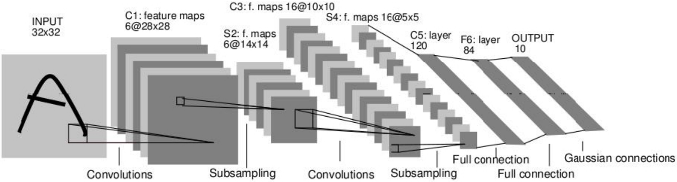
Regularization in Neural Networks

- We have infact already looked at one method (for vision tasks)



Regularization in Neural Networks

- We have infact already looked at one method (for vision tasks)



- How is this a form of regularization?

Regularization in Neural Networks

- **Weight decay**: Penalize $\|W^l\|_2$ or $\|W^l\|_1$ in every layer

Regularization in Neural Networks

- **Weight decay**: Penalize $\|W^l\|_2$ or $\|W^l\|_1$ in every layer
- Why is it called Weight decay?

Regularization in Neural Networks

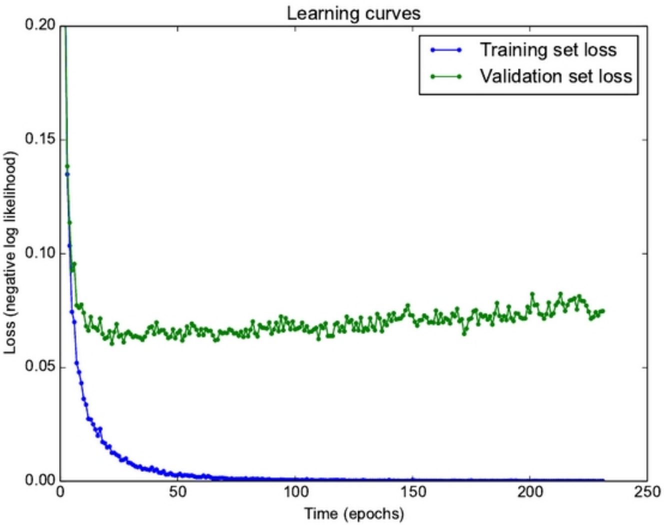
- **Weight decay**: Penalize $\|W^l\|_2$ or $\|W^l\|_1$ in every layer
- Why is it called Weight decay?
- **Parameter sharing** (CNNs, RNNs)

Regularization in Neural Networks

- **Weight decay**: Penalize $\|W^l\|_2$ or $\|W^l\|_1$ in every layer
- Why is it called Weight decay?
- **Parameter sharing** (CNNs, RNNs)
- **Dataset Augmentation** ImageNet 2012, discussed last time was won by significant dataset augmentation

Regularization in Neural Networks

- **Early Stoppping:**



Dropout

- A more exotic regularization technique. Introduced in 2012 and one of the factors in the recent Neural Net successes

Dropout

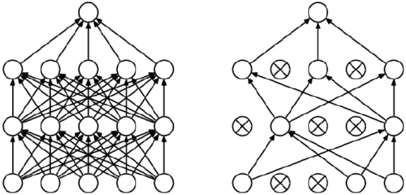
- A more exotic regularization technique. Introduced in 2012 and one of the factors in the recent Neural Net successes
- Every sample is processed by a decimated neural network

Dropout

- A more exotic regularization technique. Introduced in 2012 and one of the factors in the recent Neural Net successes
- Every sample is processed by a decimated neural network
- But, they all do the same job, and share weights

Dropout

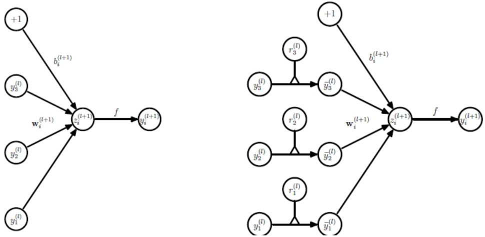
- A more exotic regularization technique. Introduced in 2012 and one of the factors in the recent Neural Net successes
- Every sample is processed by a decimated neural network
- But, they all do the same job, and share weights



(a) Standard Neural Net (b) After applying dropout.

Dropout: A simple way to prevent neural networks from overfitting, N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov, JMLR 2014

Dropout: Feedforward Operation



Without dropout: $z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)}$, and $y_i^{l+1} = f(z_i^{(l+1)})$

With dropout:

$r_j^{(l)} =$ Bernoulli(p)

$\tilde{y}^{(l)} = r^{(l)} * y^{(l)}$

$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^l + b_i^{(l+1)}$

$y_i^{l+1} = f(z_i^{(l+1)})$

Dropout: At Test time

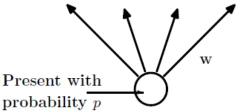
- Use a single neural net with weights scaled down

Dropout: At Test time

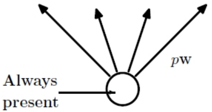
- Use a single neural net with weights scaled down
- By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time

Dropout: At Test time

- Use a single neural net with weights scaled down
- By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time



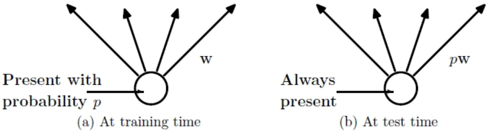
(a) At training time



(b) At test time

Dropout: At Test time

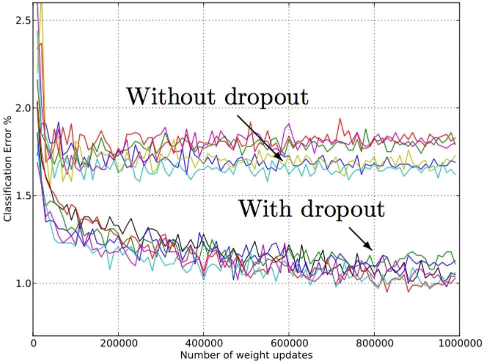
- Use a single neural net with weights scaled down
- By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time



- Extreme form of bagging

Dropout: Performance

These architectures have 2 to 4 hidden layers with 1024 to 2048 hidden units

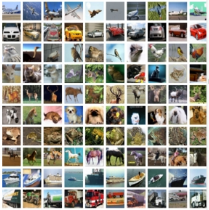


Dropout: Performance

(a) Street View House Numbers (SVHN)

Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.



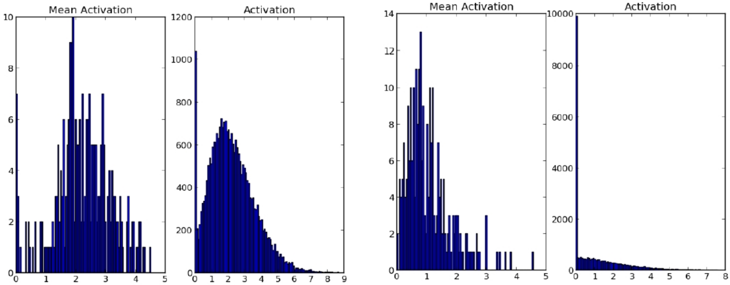
(b) CIFAR-10

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

Dropout: A simple way to prevent neural networks from overfitting, N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov, JMLR 2014

Dropout: Effect on Sparsity



(a) Without dropout (b) Dropout with $p = 0.5$.

Dropout: A simple way to prevent neural networks from overfitting, N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov, JMLR 2014

Dropout for Linear Regression

- Objective: $\|y - Xw\|_2^2$

Dropout for Linear Regression

- Objective: $\|y - Xw\|_2^2$
- When input is dropped out such that any input dimension is retained with probability p. The input can be expressed as $R * X$ where $R \in \{0,1\}^{N \times D}$ is a random matrix with $R_{ij} \sim$ Bernoulli(p)

Dropout for Linear Regression

- Objective: $\|y - Xw\|_2^2$
- When input is dropped out such that any input dimension is retained with probability p. The input can be expressed as $R * X$ where $R \in \{0,1\}^{N \times D}$ is a random matrix with $R_{ij} \sim$ Bernoulli(p)
- Marginalizing the noise, the objective becomes:

$$\min_w \mathbb{E}_{R \sim \text{ Bernoulli}(p)} \|y - (R * X)w\|_2^2$$

Dropout for Linear Regression

- Objective: $\|y - Xw\|_2^2$
- When input is dropped out such that any input dimension is retained with probability p. The input can be expressed as $R * X$ where $R \in \{0,1\}^{N \times D}$ is a random matrix with $R_{ij} \sim$ Bernoulli(p)
- Marginalizing the noise, the objective becomes:

$$\min_w \mathbb{E}_{R \sim \text{ Bernoulli}(p)} \|y - (R * X)w\|_2^2$$

- This is the same as:

$$\min_w \|y - pXw\|_2^2 + p(1-p)\|\Gamma w\|_2^2 \text{ where } \Gamma = (diag(X^T X))^{1/2}$$

Dropout for Linear Regression

- Objective: $\|y - Xw\|_2^2$
- When input is dropped out such that any input dimension is retained with probability p. The input can be expressed as $R * X$ where $R \in \{0,1\}^{N \times D}$ is a random matrix with $R_{ij} \sim$ Bernoulli(p)
- Marginalizing the noise, the objective becomes:

$$\min_w \mathbb{E}_{R \sim \text{ Bernoulli}(p)} \|y - (R * X)w\|_2^2$$

- This is the same as:

$$\min_w \|y - pXw\|_2^2 + p(1-p)\|\Gamma w\|_2^2 \text{ where } \Gamma = (diag(X^T X))^{1/2}$$

- Thus, dropout with linear regression is equivalent, in expectation to ridge regression with a particular form of Γ

Why does this make sense?

- Bagging is always good if models are diverse enough

Why does this make sense?

- Bagging is always good if models are diverse enough
- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"

Why does this make sense?

- Bagging is always good if models are diverse enough
- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"
- **Motivation 2:** Comes from a theory for the superiority of sexual reproduction in evolution (Livnat, Papadimitriou, PNAS, 2010).

Why does this make sense?

- Bagging is always good if models are diverse enough
- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"
- **Motivation 2:** Comes from a theory for the superiority of sexual reproduction in evolution (Livnat, Papadimitriou, PNAS, 2010).
- Seems plausible that asexual reproduction should be a better way to optimize for individual fitness (in sexual reproduction if a good combination is found, it's split again)

Why does this make sense?

- Bagging is always good if models are diverse enough
- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"
- **Motivation 2:** Comes from a theory for the superiority of sexual reproduction in evolution (Livnat, Papadimitriou, PNAS, 2010).
- Seems plausible that asexual reproduction should be a better way to optimize for individual fitness (in sexual reproduction if a good combination is found, it's split again)
- Criterion for natural selection may not be individual fitness but mixability. Thus role of sexual reproduction is not just to allow useful new genes to propagate but also to ensure that complex coadaptations between genes are broken

Sequence Learning with Neural Networks

Problems with MLPs for Sequence Tasks

- The "API" is too limited. They only accept an input of a fixed dimensionality and map it to an output that is again of a fixed dimensionality

Problems with MLPs for Sequence Tasks

- The "API" is too limited. They only accept an input of a fixed dimensionality and map it to an output that is again of a fixed dimensionality
- This is great when working (for example) with images, and the output is an encoding of the category

Problems with MLPs for Sequence Tasks

- The "API" is too limited. They only accept an input of a fixed dimensionality and map it to an output that is again of a fixed dimensionality
- This is great when working (for example) with images, and the output is an encoding of the category
- This is bad when if we are interested in Machine Translation or Speech Recognition

Problems with MLPs for Sequence Tasks

- The "API" is too limited. They only accept an input of a fixed dimensionality and map it to an output that is again of a fixed dimensionality
- This is great when working (for example) with images, and the output is an encoding of the category
- This is bad when if we are interested in Machine Translation or Speech Recognition
- Traditional Neural Networks treat every example independently. Imagine the task is to classify events at every fixed point in the movie. A plain vanilla neural network would not be able to use its knowledge about the previous events to help in classifying the current.

Problems with MLPs for Sequence Tasks

- The "API" is too limited. They only accept an input of a fixed dimensionality and map it to an output that is again of a fixed dimensionality
- This is great when working (for example) with images, and the output is an encoding of the category
- This is bad when if we are interested in Machine Translation or Speech Recognition
- Traditional Neural Networks treat every example independently. Imagine the task is to classify events at every fixed point in the movie. A plain vanilla neural network would not be able to use its knowledge about the previous events to help in classifying the current.
- Recurrent Neural Networks address this issue by having loops.

Some Sequence Tasks

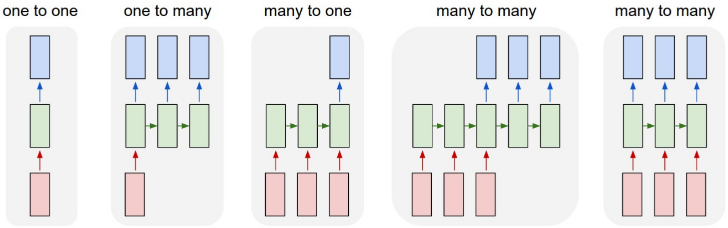
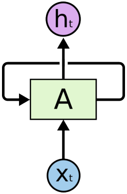


Figure credit: Andrej Karpathy

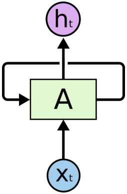
Recurrent Neural Networks

- The loops in them allow the information to persist



Recurrent Neural Networks

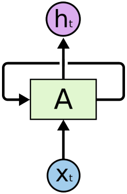
- The loops in them allow the information to persist



- For some input x_i, we pass it through a hidden state A and then output a value h_i. The loop allows information to be passed from one time step to another

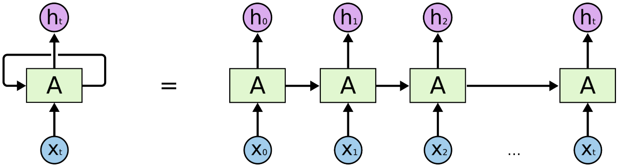
Recurrent Neural Networks

- The loops in them allow the information to persist



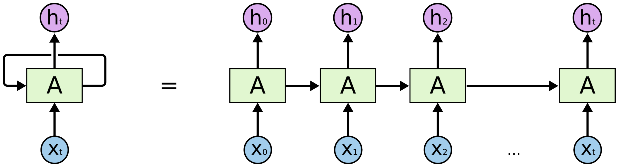
- For some input x_i, we pass it through a hidden state A and then output a value h_i. The loop allows information to be passed from one time step to another
- A RNN can be thought of as multiple copies of the same network, each of which passes a message to its successor

Recurrent Neural Networks



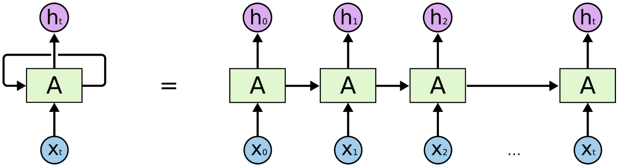
- More generally, a RNN can be thought of as arranging hidden state vectors h_t^l in a 2-D grid, with $t = 1, \ldots, T$ being time and $l = 1, \ldots, L$ being the depth

Recurrent Neural Networks



- More generally, a RNN can be thought of as arranging hidden state vectors h_t^l in a 2-D grid, with $t = 1, \ldots, T$ being time and $l = 1, \ldots, L$ being the depth
- $h_t^0 = x_t$ and h_t^L is used to predict the output vector y_t. All intermediate vectors h_t^l are computed as a function of h_{t-1}^l h_t^{l-1}

Recurrent Neural Networks



- More generally, a RNN can be thought of as arranging hidden state vectors h_t^l in a 2-D grid, with $t = 1, \ldots, T$ being time and $l = 1, \ldots, L$ being the depth
- $h_t^0 = x_t$ and h_t^L is used to predict the output vector y_t. All intermediate vectors h_t^l are computed as a function of h_{t-1}^l h_t^{l-1}
- RNN is a recurrence of the form:

$$h_t^l = tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

Illustration credit: Chris Olah

Recurrent Neural Networks

- The chain like structure enables sequence modeling

Recurrent Neural Networks

- The chain like structure enables sequence modeling
- W varies between layers but is shared through time

Recurrent Neural Networks

- The chain like structure enables sequence modeling
- W varies between layers but is shared through time
- Basically the inputs from the layer below and before in time are transformed by a non-linearity after an additive interaction (weak coupling)

Recurrent Neural Networks

- The chain like structure enables sequence modeling
- W varies between layers but is shared through time
- Basically the inputs from the layer below and before in time are transformed by a non-linearity after an additive interaction (weak coupling)
- The plain vanilla RNN described is infact Turing Complete with the right size and weight matrix

Recurrent Neural Networks

- The chain like structure enables sequence modeling
- W varies between layers but is shared through time
- Basically the inputs from the layer below and before in time are transformed by a non-linearity after an additive interaction (weak coupling)
- The plain vanilla RNN described is infact Turing Complete with the right size and weight matrix
- "If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs"

Recurrent Neural Networks

- Training RNNs might seem daunting.
- Infact, we can simply adopt the backpropagation algorithm after unrolling the RNN

Recurrent Neural Networks

- Training RNNs might seem daunting.
- Infact, we can simply adopt the backpropagation algorithm after unrolling the RNN
- If we have to look at sequences of size s, we unroll each loop into s steps, and treat it as a normal neural network to train using backpropagation

Recurrent Neural Networks

- Training RNNs might seem daunting.
- Infact, we can simply adopt the backpropagation algorithm after unrolling the RNN
- If we have to look at sequences of size s, we unroll each loop into s steps, and treat it as a normal neural network to train using backpropagation
- This is called Backpropagation through time

Recurrent Neural Networks

- Training RNNs might seem daunting.
- Infact, we can simply adopt the backpropagation algorithm after unrolling the RNN
- If we have to look at sequences of size s, we unroll each loop into s steps, and treat it as a normal neural network to train using backpropagation
- This is called Backpropagation through time
- But weights are shared across different time stamps? How is this constraint enforced?

Recurrent Neural Networks

- Training RNNs might seem daunting.
- Infact, we can simply adopt the backpropagation algorithm after unrolling the RNN
- If we have to look at sequences of size s, we unroll each loop into s steps, and treat it as a normal neural network to train using backpropagation
- This is called Backpropagation through time
- But weights are shared across different time stamps? How is this constraint enforced?
- Train the network as if there were no constraints, obtain weights at different time stamps, average them

Problems

- Recurrent Neural Networks have trouble learning long term dependencies (Hochreiter and Schmidhuber, 1991 and Bengio *et al*, 1994)
- Consider a language model in which the task is to predict the next word based on the previous

Problems

- Recurrent Neural Networks have trouble learning long term dependencies (Hochreiter and Schmidhuber, 1991 and Bengio *et al*, 1994)
- Consider a language model in which the task is to predict the next word based on the previous
- Sometimes the context can be clear immediately: "The clouds are in the *sky*"

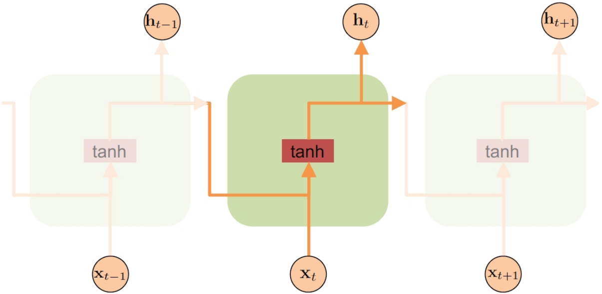
Problems

- Recurrent Neural Networks have trouble learning long term dependencies (Hochreiter and Schmidhuber, 1991 and Bengio *et al*, 1994)
- Consider a language model in which the task is to predict the next word based on the previous
- Sometimes the context can be clear immediately: "The clouds are in the *sky*"
- Sometimes the dependency is more long term: "We are basically from Transylvania, although I grew up in Spain, but I can still speak fluent *Romanian*."
- In principle, RNNs should be able to learn long term dependencies with the right parameter choices, but learning those parameters is hard.

Problems

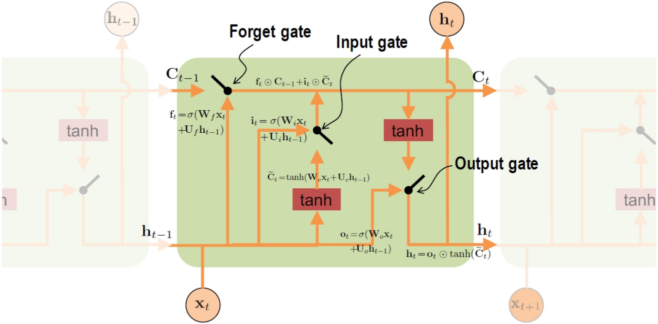
- Recurrent Neural Networks have trouble learning long term dependencies (Hochreiter and Schmidhuber, 1991 and Bengio *et al*, 1994)
- Consider a language model in which the task is to predict the next word based on the previous
- Sometimes the context can be clear immediately: "The clouds are in the *sky*"
- Sometimes the dependency is more long term: "We are basically from Transylvania, although I grew up in Spain, but I can still speak fluent *Romanian*."
- In principle, RNNs should be able to learn long term dependencies with the right parameter choices, but learning those parameters is hard.
- The Long Short Term Memory was proposed to solve this problem (Hochreiter and Schmidhuber, 1997)

Long Short Term Memory Networks



Vanilla RNN: Error propagation is blocked by a non-linearity *Illustration credit: Chris Olah*

Long Short Term Memory Networks



Long Short Term Memory

- One of the main points about LSTM is the cell state C_t, which runs across time and can travel unchanged only with minor linear interactions

Long Short Term Memory

- One of the main points about LSTM is the cell state C_t, which runs across time and can travel unchanged only with minor linear interactions
- The LSTM regulates the cell state by various gates, which gives the ability to remove or add information to the cell state.

Long Short Term Memory

- One of the main points about LSTM is the cell state C_t, which runs across time and can travel unchanged only with minor linear interactions
- The LSTM regulates the cell state by various gates, which gives the ability to remove or add information to the cell state.
- Each of the gates are composed of a sigmoid non-linearity followed by a pointwise multiplication

Long Short Term Memory

- One of the main points about LSTM is the cell state C_t, which runs across time and can travel unchanged only with minor linear interactions
- The LSTM regulates the cell state by various gates, which gives the ability to remove or add information to the cell state.
- Each of the gates are composed of a sigmoid non-linearity followed by a pointwise multiplication
- There are three types of gates in LSTM (e.g. forget gate helps the LSTM to learn to forget)

Long Short Term Memory

- Precise form of the LSTM update is:

$$\begin{pmatrix} i \\ f \\ o \\ \hat{C}_t \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot \hat{C}_t, \text{ and } h_t^l = o \odot tanh(c_t^l)$$

Some Applications: Caption Generation

Caption Generation (Karpathy and Li, 2014)

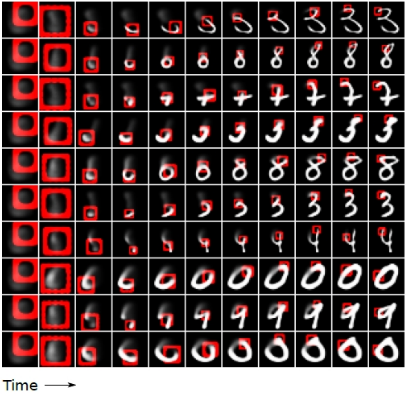
RNN Shakespeare

Using a character level language model trained on all of Shakespeare.

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

Image Generation



(Also uses attention mechanism - not discussed) DRAW: A Recurrent Neural Network For Image Generation (Gregor *et al.*, 2015)

Applications

- Acoustic Modeling

Applications

- Acoustic Modeling
- Natural Language Processing i.e. parsing etc

Applications

- Acoustic Modeling
- Natural Language Processing i.e. parsing etc
- Machine Translation (e.g. Google Translate uses RNNs)

Applications

- Acoustic Modeling
- Natural Language Processing i.e. parsing etc
- Machine Translation (e.g. Google Translate uses RNNs)
- Voice Transcription

Applications

- Acoustic Modeling
- Natural Language Processing i.e. parsing etc
- Machine Translation (e.g. Google Translate uses RNNs)
- Voice Transcription
- Video and Image understanding

Applications

- Acoustic Modeling
- Natural Language Processing i.e. parsing etc
- Machine Translation (e.g. Google Translate uses RNNs)
- Voice Transcription
- Video and Image understanding
- list goes on

Generative Neural Models

Recap: Multilayered Neural Networks

- Let layer k compute an output vector h^k using the output h^{k-1} of the previous layer.

Recap: Multilayered Neural Networks

- Let layer k compute an output vector h^k using the output h^{k-1} of the previous layer. Note that the input $\mathbf{x} = h^0$

Recap: Multilayered Neural Networks

- Let layer k compute an output vector h^k using the output h^{k-1} of the previous layer. Note that the input $\mathbf{x} = h^0$

$$h^k = tanh(b^k + W^k h^{k-1})$$

Recap: Multilayered Neural Networks

- Let layer k compute an output vector h^k using the output h^{k-1} of the previous layer. Note that the input $\mathbf{x} = h^0$

$$h^k = tanh(b^k + W^k h^{k-1})$$

- Top layer output h^l is used for making a prediction. If the target is given by y, then we define a loss $L(h^l, y)$ (convex in $b^l + W^l h^{l-1}$)

Recap: Multilayered Neural Networks

- Let layer k compute an output vector h^k using the output h^{k-1} of the previous layer. Note that the input $\mathbf{x} = h^0$

$$h^k = tanh(b^k + W^k h^{k-1})$$

- Top layer output h^l is used for making a prediction. If the target is given by y, then we define a loss $L(h^l, y)$ (convex in $b^l + W^l h^{l-1}$)
- We might have the output layer return the following non-linearity

$$h_i^l = \frac{e^{b_i^l + W_i^l h^{l-1}}}{\sum_j e^{b_j^l + W_j^l h^{l-1}}}$$

Recap: Multilayered Neural Networks

- Let layer k compute an output vector h^k using the output h^{k-1} of the previous layer. Note that the input $\mathbf{x} = h^0$

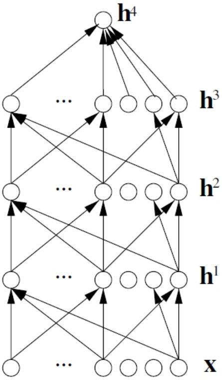
$$h^k = tanh(b^k + W^k h^{k-1})$$

- Top layer output h^l is used for making a prediction. If the target is given by y, then we define a loss $L(h^l, y)$ (convex in $b^l + W^l h^{l-1}$)
- We might have the output layer return the following non-linearity

$$h_i^l = \frac{e^{b_i^l + W_i^l h^{l-1}}}{\sum_j e^{b_j^l + W_j^l h^{l-1}}}$$

- This is called the softmax and can be used as an estimator of $p(Y = i|x)$

Recap: Multilayered Neural Networks



One loss to be considered: $L(h^l, y) = -\log P(Y = y|x)$

The Difficulty of Training Deep Networks

- Until 2006, deep architectures were not used extensively in Machine Learning

The Difficulty of Training Deep Networks

- Until 2006, deep architectures were not used extensively in Machine Learning
- Poor training and generalization errors using the standard random initialization (with the exception of convolutional neural networks)

The Difficulty of Training Deep Networks

- Until 2006, deep architectures were not used extensively in Machine Learning
- Poor training and generalization errors using the standard random initialization (with the exception of convolutional neural networks)
- Difficult to propagate gradients to lower layers. Too many connections in a deep architecture

The Difficulty of Training Deep Networks

- Until 2006, deep architectures were not used extensively in Machine Learning
- Poor training and generalization errors using the standard random initialization (with the exception of convolutional neural networks)
- Difficult to propagate gradients to lower layers. Too many connections in a deep architecture
- Purely discriminative. No generative model for the raw input features x (connections go upwards)

Initial Breakthrough: Layer-wise Training

- Unsupervised pre-training is possible in certain Deep Generative Models (Hinton, 2006)

Initial Breakthrough: Layer-wise Training

- Unsupervised pre-training is possible in certain Deep Generative Models (Hinton, 2006)
- Idea: Greedily train one layer at a time using a simple model (Restricted Boltzmann Machine)

Initial Breakthrough: Layer-wise Training

- Unsupervised pre-training is possible in certain Deep Generative Models (Hinton, 2006)
- Idea: Greedily train one layer at a time using a simple model (Restricted Boltzmann Machine)
- Use the parameters learned to initialize a feedforward neural network, and fine tune for classification

Sigmoid Belief Networks, 1992

- The generative model is decomposed as:

$$P(x, h^1, \ldots, h^l) = P(h^l)\Big(\prod_{k=1}^{l-1} P(h^k|h^{k+1})\Big)P(x|h^1)$$

Sigmoid Belief Networks, 1992

- The generative model is decomposed as:

$$P(x, h^1, \ldots, h^l) = P(h^l)\Big(\prod_{k=1}^{l-1} P(h^k|h^{k+1})\Big)P(x|h^1)$$

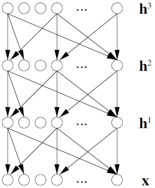
- Marginalization yields $P(x)$. Intractable in practice except for tiny models

Sigmoid Belief Networks, 1992

- The generative model is decomposed as:

$$P(x, h^1, \ldots, h^l) = P(h^l)\Big(\prod_{k=1}^{l-1} P(h^k|h^{k+1})\Big)P(x|h^1)$$

- Marginalization yields $P(x)$. Intractable in practice except for tiny models



R. Neal, Connectionist learning of belief networks, 1992

Dayan, P., Hinton, G. E., Neal, R., and Zemel, R. S. The Helmholtz Machine, 1995

L. Saul, T. Jaakkola, and M. Jordan, Mean field theory for sigmoid belief networks, 1996

Deep Belief Networks, 2006

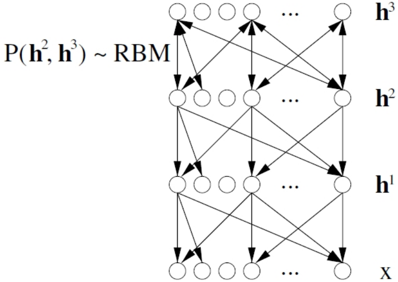
- Similar to Sigmoid Belief Networks, except the top two layers

$$P(x, h^1, \ldots, h^l) = P(h^{l-1}, h^l)\left(\prod_{k=1}^{l-2} P(h^k|h^{k+1})\right)P(x|h^1)$$

Deep Belief Networks, 2006

- Similar to Sigmoid Belief Networks, except the top two layers

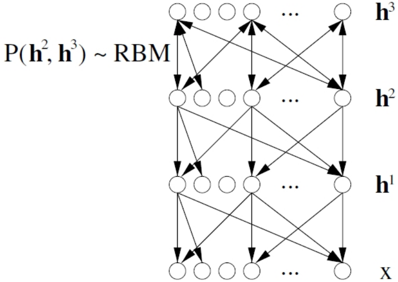
$$P(x, h^1, \ldots, h^l) = P(h^{l-1}, h^l)\Big(\prod_{k=1}^{l-2} P(h^k | h^{k+1})\Big) P(x | h^1)$$



Deep Belief Networks, 2006

- Similar to Sigmoid Belief Networks, except the top two layers

$$P(x, h^1, \ldots, h^l) = P(h^{l-1}, h^l)\Big(\prod_{k=1}^{l-2} P(h^k|h^{k+1})\Big)P(x|h^1)$$



- The joint distribution of the top two layers is a Restricted Boltzmann Machine

Energy Based Models

- Before looking at RBMs, let's look at the basics of Energy based models

Energy Based Models

- Before looking at RBMs, let's look at the basics of Energy based models
- Such models assign a scalar energy to each configuration of the variables of interest. Learning then corresponds to modifying the energy function so that its shape has desirable properties

$$P(x) = \frac{e^{-\text{Energy}(x)}}{Z} \text{ where } Z = \sum_x e^{-\text{Energy}(x)}$$

Energy Based Models

- Before looking at RBMs, let's look at the basics of Energy based models
- Such models assign a scalar energy to each configuration of the variables of interest. Learning then corresponds to modifying the energy function so that its shape has desirable properties

$$P(x) = \frac{e^{-\text{Energy}(x)}}{Z} \text{ where } Z = \sum_x e^{-\text{Energy}(x)}$$

- We only care about the marginal (since only x is observed)

Energy Based Models

- With hidden variables $P(x,h) = \frac{e^{-\text{Energy}(x,h)}}{Z}$

Energy Based Models

- With hidden variables $P(x,h) = \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We only care about the marginal (since only x is observed) $P(x) = \sum_h \frac{e^{-\text{Energy}(x,h)}}{Z}$

Energy Based Models

- With hidden variables $P(x,h) = \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We only care about the marginal (since only x is observed) $P(x) = \sum_h \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We can introduce the notion of free-energy

$$P(x) = \frac{e^{-\text{FreeEnergy}(x)}}{Z}, \text{ with } Z = \sum_x e^{-\text{FreeEnergy}(x)}$$

Energy Based Models

- With hidden variables $P(x,h) = \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We only care about the marginal (since only x is observed) $P(x) = \sum_h \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We can introduce the notion of free-energy

$$P(x) = \frac{e^{-\text{FreeEnergy}(x)}}{Z}, \text{ with } Z = \sum_x e^{-\text{FreeEnergy}(x)}$$

- Where

$$\text{FreeEnergy}(x) = -\log \sum_h e^{-\text{Energy}(x,h)}$$

Energy Based Models

- With hidden variables $P(x,h) = \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We only care about the marginal (since only x is observed) $P(x) = \sum_h \frac{e^{-\text{Energy}(x,h)}}{Z}$
- We can introduce the notion of free-energy

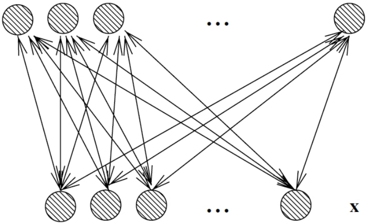
$$P(x) = \frac{e^{-\text{FreeEnergy}(x)}}{Z}, \text{ with } Z = \sum_x e^{-\text{FreeEnergy}(x)}$$

- Where

$$\text{FreeEnergy}(x) = -\log \sum_h e^{-\text{Energy}(x,h)}$$

- The data log-likelihood gradient has an interesting form (details skipped)

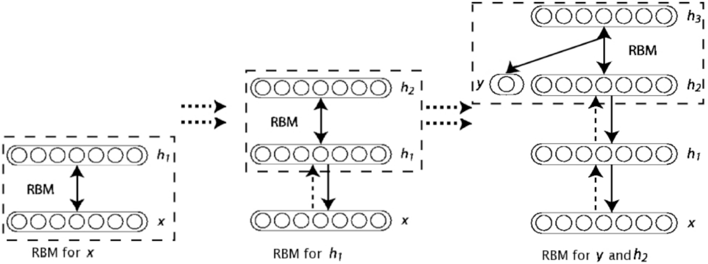
Restricted Boltzmann Machines



$$
\begin{aligned}
P(\mathbf{h}|\mathbf{x}) &= \frac{\exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\mathbf{h} + \mathbf{h}'W\mathbf{x})}{\sum_{\tilde{\mathbf{h}}} \exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\tilde{\mathbf{h}} + \tilde{\mathbf{h}}'W\mathbf{x})} \\
&= \frac{\prod_i \exp(\mathbf{c}_i\mathbf{h}_i + \mathbf{h}_i W_i \mathbf{x})}{\prod_i \sum_{\tilde{\mathbf{h}}_i} \exp(\mathbf{c}_i\tilde{\mathbf{h}}_i + \tilde{\mathbf{h}}_i W_i \mathbf{x})} \\
&= \prod_i \frac{\exp(\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x}))}{\sum_{\tilde{\mathbf{h}}_i} \exp(\tilde{\mathbf{h}}_i(\mathbf{c}_i + W_i\mathbf{x}))} \\
&= \prod_i P(\mathbf{h}_i|\mathbf{x}).
\end{aligned}
$$

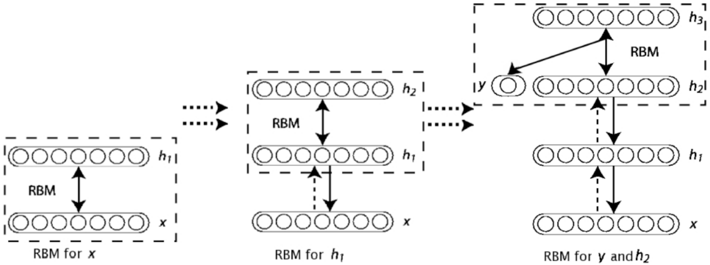
$$x_1 \to h_1 \sim P(h|x_1) \to x_2 \sim P(x|h_1) \to h_2 \sim P(h|x_2) \to \dots$$

Back to Deep Belief Networks



Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)
→ Supervised deep neural network

Back to Deep Belief Networks



Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)
→ Supervised deep neural network

- Everything is completely unsupervised till now. We can treat these weights learned as an initialization, treat the network as a feedword network and fine tune using backpropagation

Deep Belief Networks

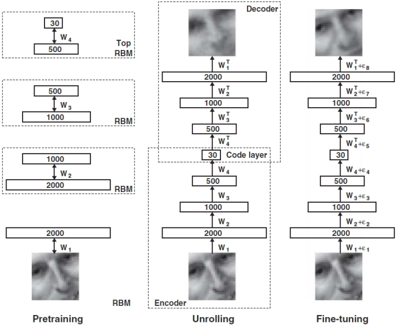
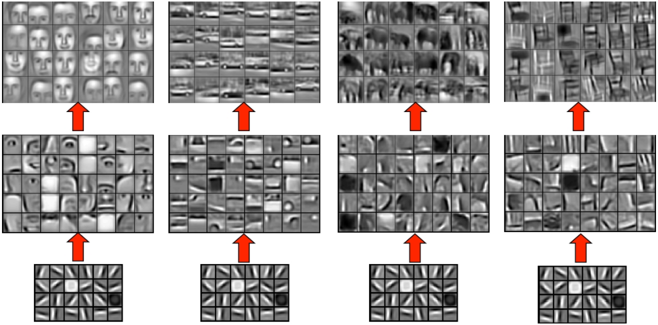


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, 2006

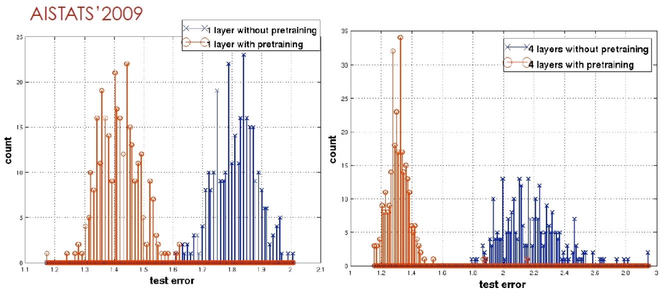
G. E. Hinton, S Osindero, YW Teh, A fast learning algorithm for deep belief nets, Neural Computation, 2006

Deep Belief Networks: Object Parts

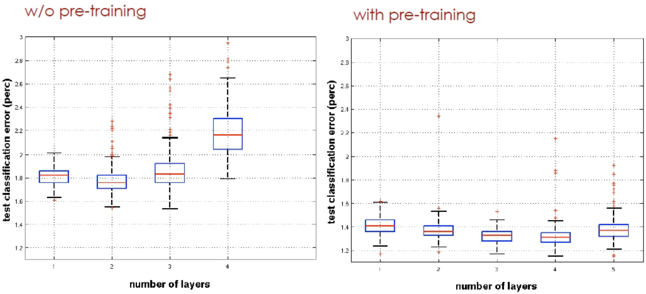


Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng

Effect of Unsupervised Pre-training



Effect of Unsupervised Pre-training



Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$

Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$
- Optimization: Unsupervised pre-training leads to better regions of the space i.e. better than random initialization

Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$
- Optimization: Unsupervised pre-training leads to better regions of the space i.e. better than random initialization

Autoencoders

- Main idea
- Sparse Autoencoders
- Denoising Autoencoders
- Pretraining using Autoencoders