Deep Learning Tutorial

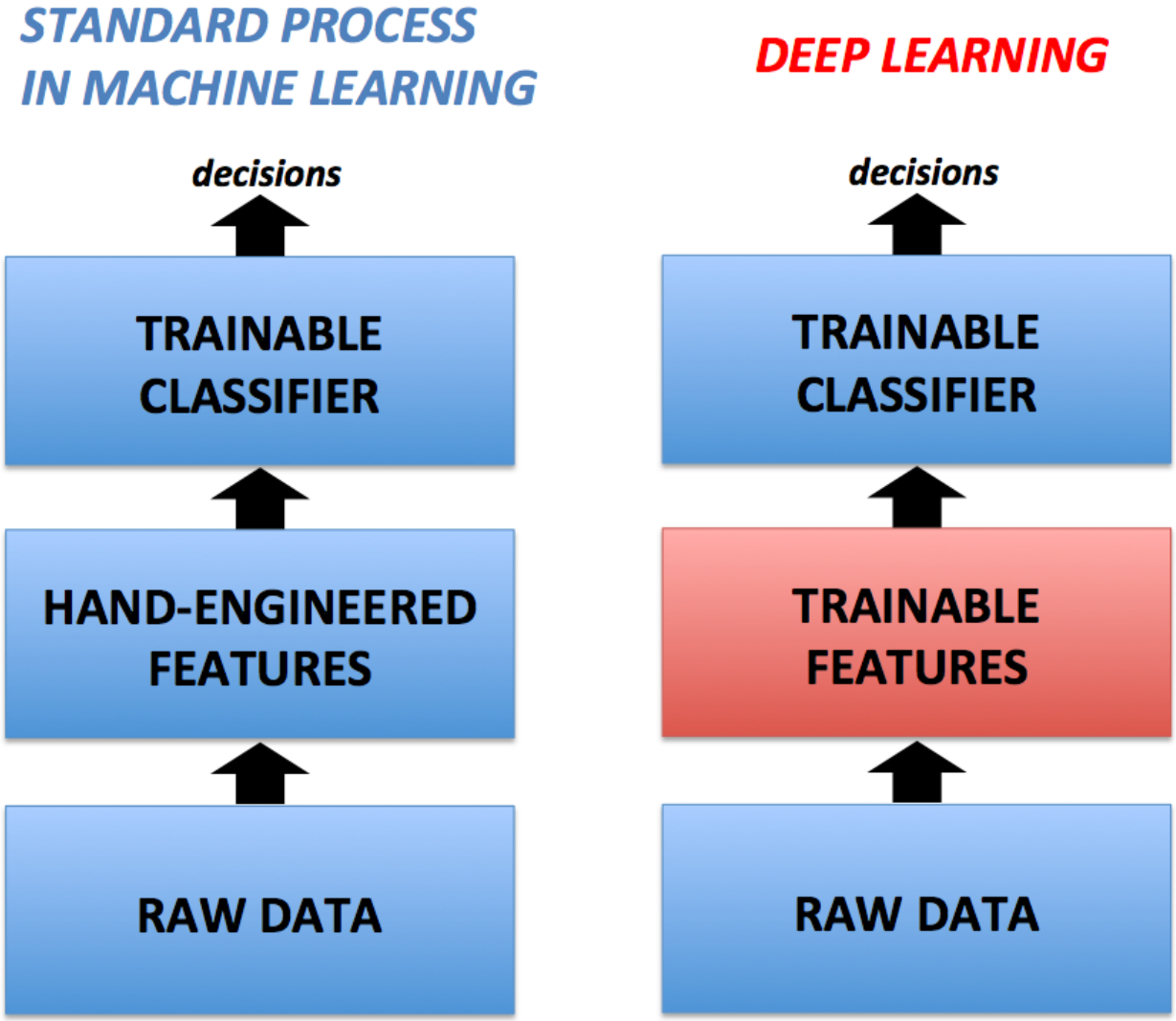
Kevin Duh

Nara Institute of Science & Technology / Johns Hopkins HLTCOE

Oct 18, 2015

What is Deep Learning?

A family of methods that uses deep architectures to learn high-level feature representations

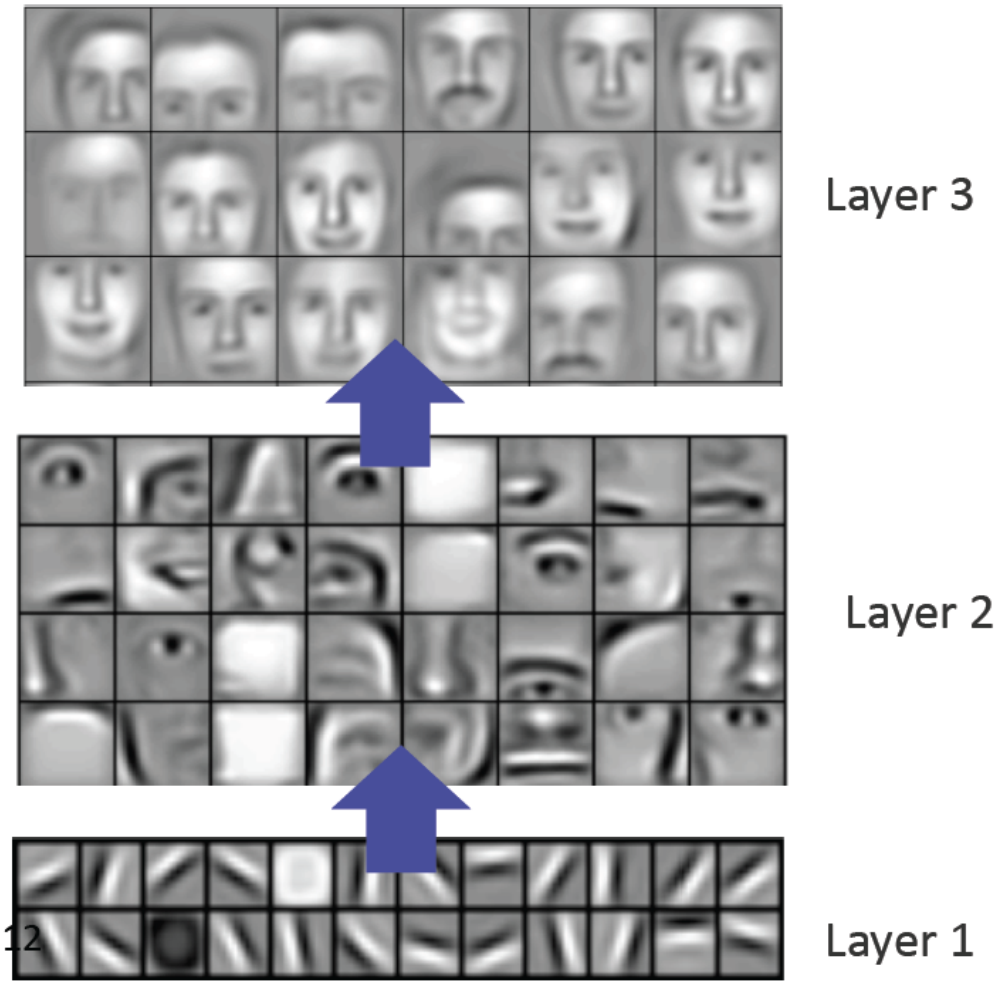


What got me excited in Deep Learning

Automatically trained features make sense! [Lee et al., 2009]

Input: Images (raw pixels)

$\rightarrow$ Output: Features of Edges, Body Parts, Full Faces



Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

This is a long talk, so...

- Please interrupt with questions/comments anytime.
- Feel free to come and go as desired.

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Problem Setup

- Training Data: a set of $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ pairs
  - Input $x^{(m)} \in R^d$
  - Output $y^{(m)} = \{0, 1\}$
- Goal: Learn function $f : x \rightarrow y$ to predict correctly on new inputs x.

Problem Setup

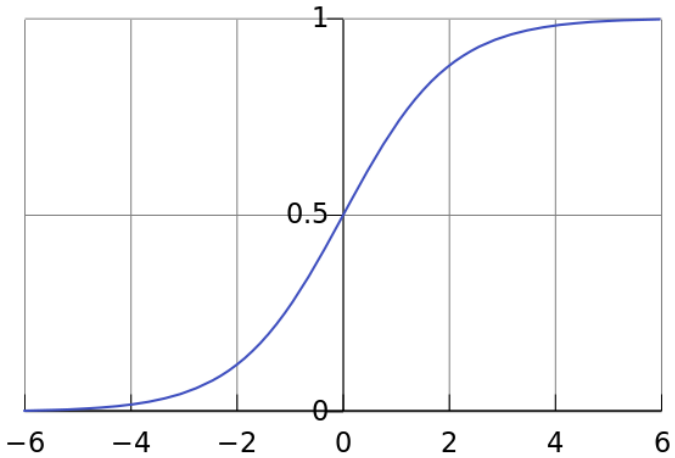
- Training Data: a set of $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ pairs
  - Input $x^{(m)} \in R^d$
  - Output $y^{(m)} = \{0, 1\}$
- Goal: Learn function $f : x \rightarrow y$ to predict correctly on new inputs x.
  - Step 1: Choose a function model family:
    - e.g. logistic regression, support vector machines, neural networks

Problem Setup

- Training Data: a set of $(x^{(m)}, y^{(m)})_{m=\{1,2,..M\}}$ pairs
  - Input $x^{(m)} \in R^d$
  - Output $y^{(m)} = \{0, 1\}$
- Goal: Learn function $f : x \rightarrow y$ to predict correctly on new inputs x.
  - Step 1: Choose a function model family:
    - e.g. logistic regression, support vector machines, neural networks
  - Step 2: Optimize parameters w on the Training Data
    - e.g. minimize loss function $\min_w \sum_{m=1}^{M} (f_w(x^{(m)}) - y^{(m)})^2$

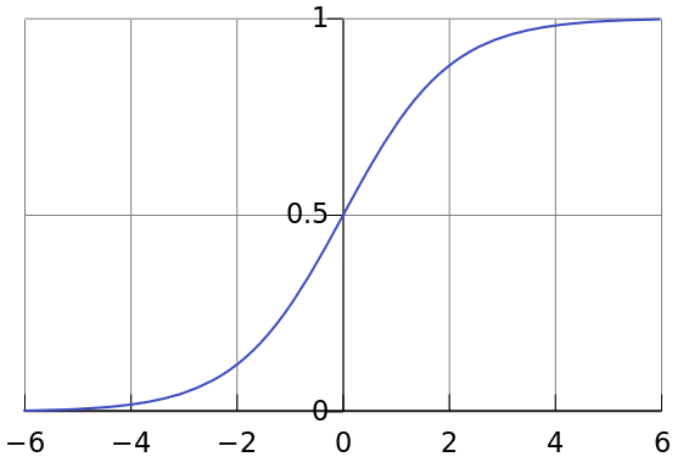
Logistic Regression (1-layer net)

- Function model: $f(x) = \sigma(w^T \cdot x)$
  - Parameters: vector $w \in R^d$
  - σ is a non-linearity, e.g. sigmoid: $\sigma(z) = 1/(1 + \exp(-z))$



Logistic Regression (1-layer net)

- Function model: $f(x) = \sigma(w^T \cdot x)$
  - Parameters: vector $w \in R^d$
  - σ is a non-linearity, e.g. sigmoid: $\sigma(z) = 1/(1 + \exp(-z))$



- Non-linearity will be important in expressiveness multi-layer nets. Other non-linearities, e.g., $tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$

Gradient Descent for Logistic Regression

- Assume Squared-Error*
  $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$

*An alternative is Cross-Entropy loss:
$\sum_m \; y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Gradient Descent for Logistic Regression

- Assume Squared-Error*
  $Loss(w) = \frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2$
- Gradient:
  $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right]\sigma'(w^T x^{(m)})x^{(m)}$

*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Gradient Descent for Logistic Regression

- Assume Squared-Error*
  $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
- Gradient:
  $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$
  - Define input into non-linearity $in^{(m)} = w^T x^{(m)}$
  - General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
  - Derivative of sigmoid $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Gradient Descent for Logistic Regression

- Assume Squared-Error*
  $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
- Gradient:
  $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$
  - Define input into non-linearity $in^{(m)} = w^T x^{(m)}$
  - General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$
  - Derivative of sigmoid $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Gradient Descent Algorithm:
  1. Initialize w randomly
  2. Update until convergence: $w \leftarrow w - \gamma(\nabla_w Loss)$
- Stochastic gradient descent (SGD) algorithm:
  1. Initialize w randomly
  2. Update until convergence:
     $w \leftarrow w - \gamma(Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$

---

*An alternative is Cross-Entropy loss:
$\sum_m y^{(m)} \log(\sigma(w^T x^{(m)})) + (1 - y^{(m)}) \log(1 - \sigma(w^T x^{(m)}))$

Stochastic Gradient Descent (SGD)

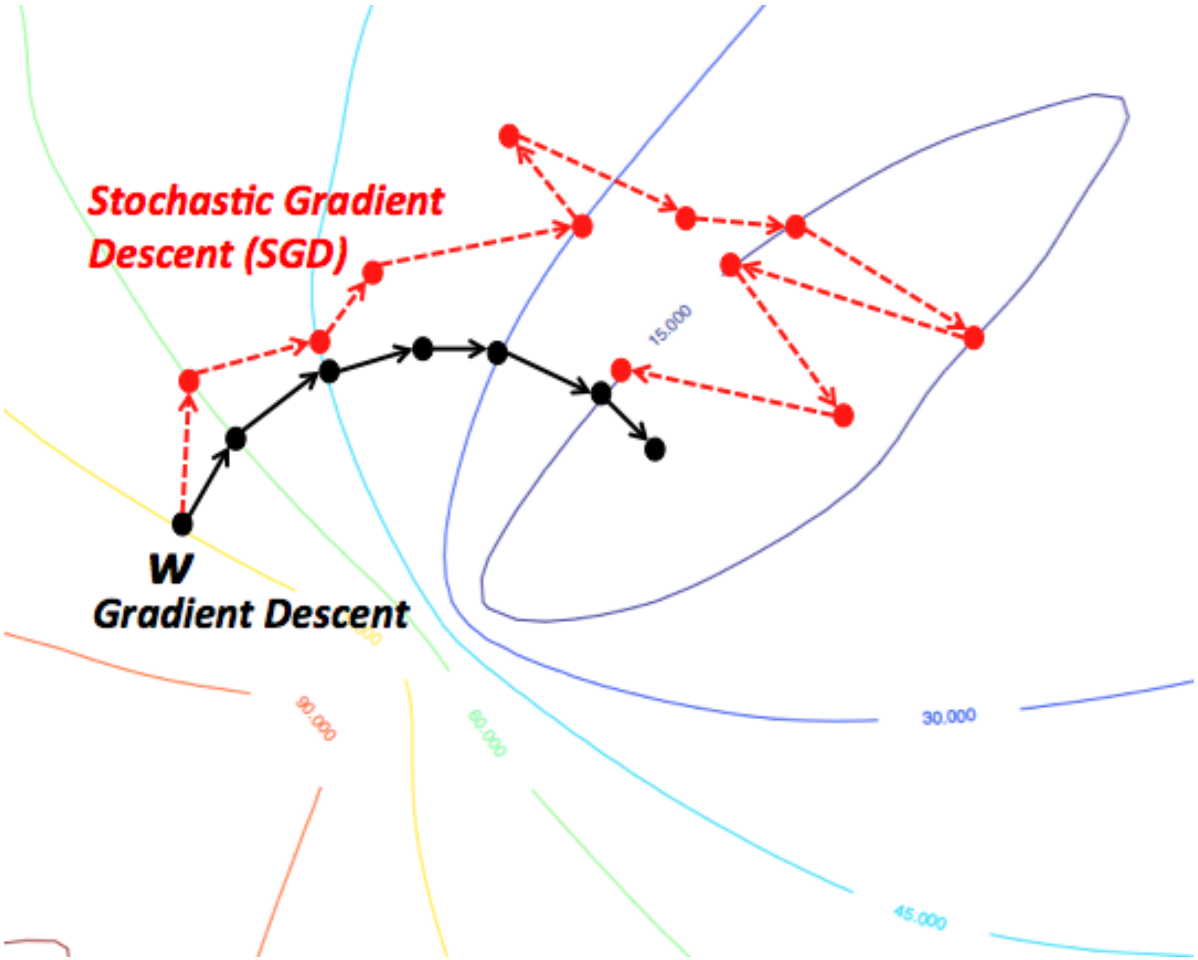
- Gradient Descent Algorithm:
  1. Initialize w randomly
  2. Update until convergence: $w \leftarrow w - \gamma(\nabla_w Loss)$
- Stochastic gradient descent (SGD) algorithm:
  1. Initialize w randomly
  2. Update until convergence:
     $w \leftarrow w - \gamma(\frac{1}{|B|} \sum_{m \in B} Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$
     where minibatch B ranges from e.g. 1-100 samples

Stochastic Gradient Descent (SGD)

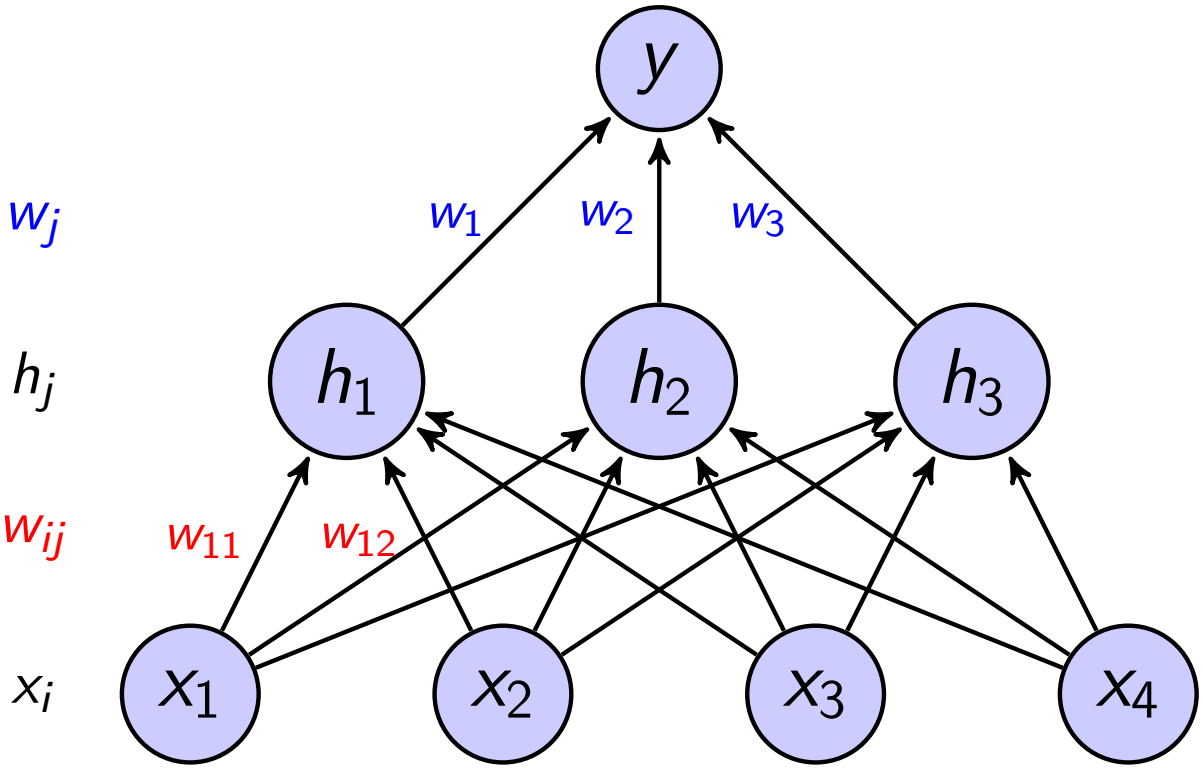
- Gradient Descent Algorithm:
  1. Initialize w randomly
  2. Update until convergence: $w \leftarrow w - \gamma(\nabla_w Loss)$
- Stochastic gradient descent (SGD) algorithm:
  1. Initialize w randomly
  2. Update until convergence:
     $w \leftarrow w - \gamma(\frac{1}{|B|}\sum_{m \in B} Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$
     where minibatch B ranges from e.g. 1-100 samples
- Learning rate γ:
  - For convergence, should decrease with each iteration t through samples
  - e.g. $\gamma_t = \frac{1}{\lambda * t}$ or $\gamma_t = \frac{\gamma_0}{1+\gamma_0 * \lambda * t}$

SGD Pictorial View

- Loss objective contour plot:
  $\frac{1}{2}\sum_m(\sigma(w^T x^{(m)}) - y^{(m)})^2 + ||w||$
  - Gradient descent goes in steepest descent direction
  - SGD is noisy descent (but faster per iteration)



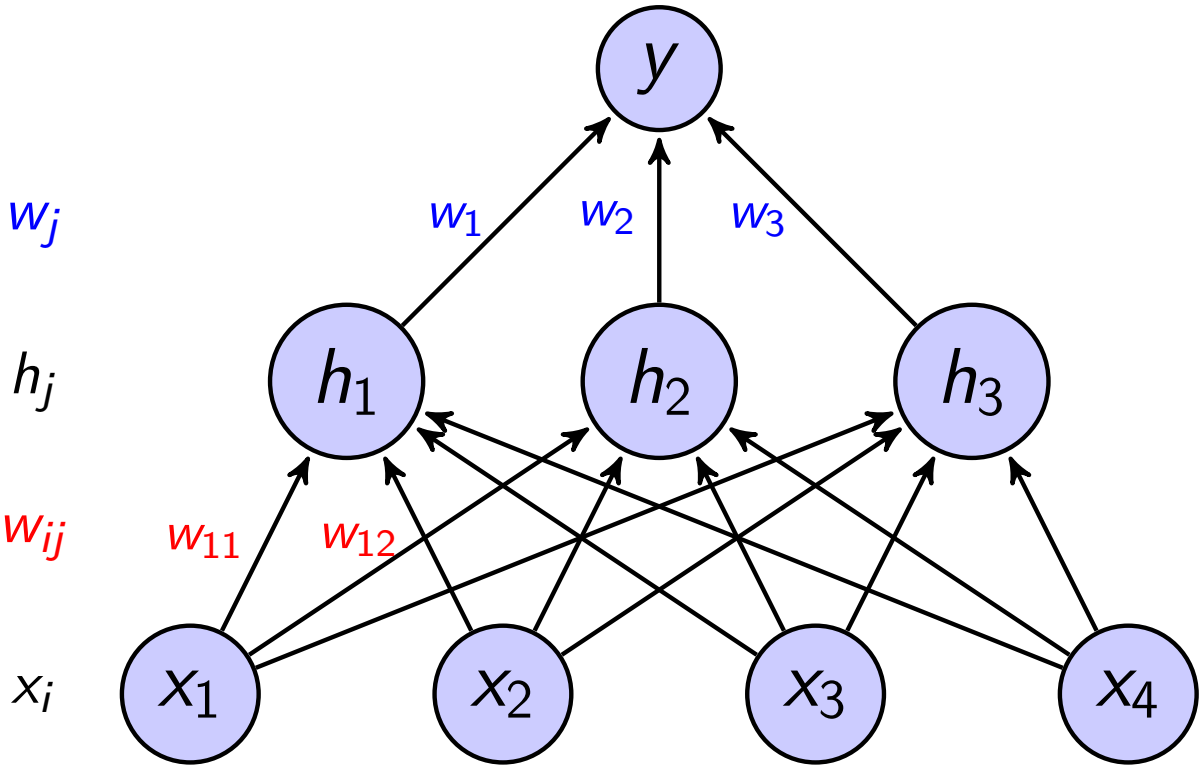
2-layer Neural Networks



$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

Called Multilayer Perceptron (MLP), but more like multilayer logistic regression

2-layer Neural Networks



$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

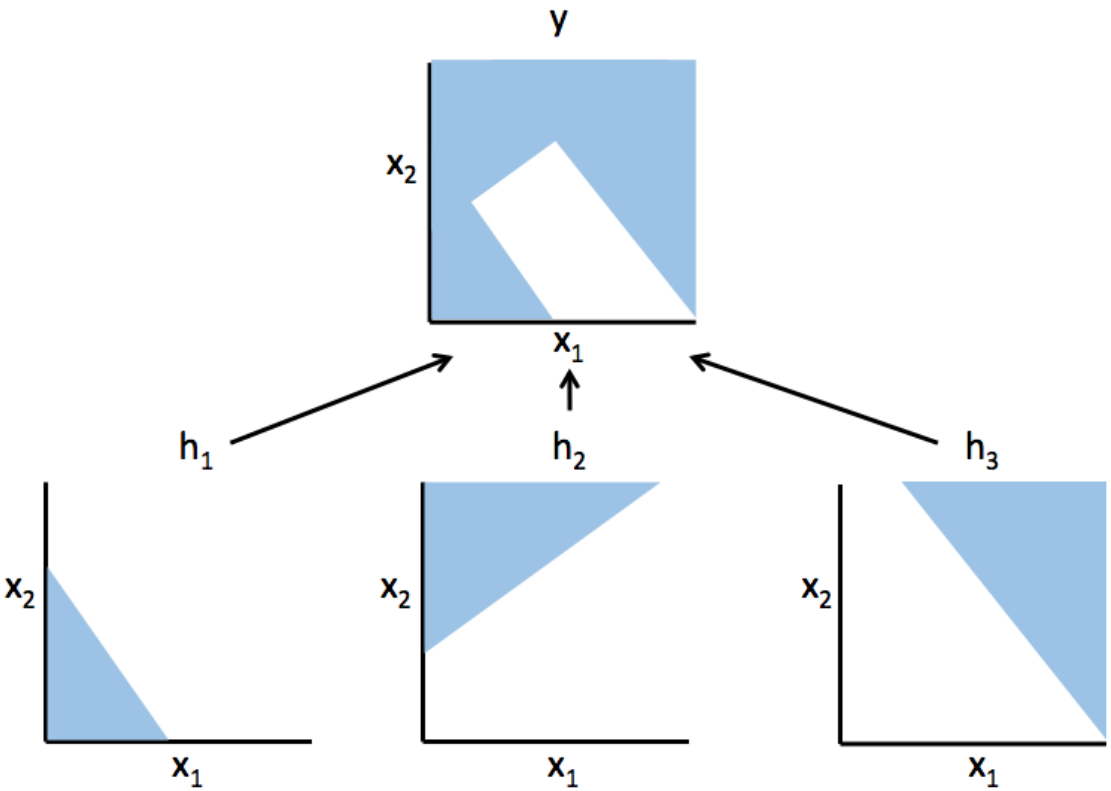
Hidden units h_j's can be viewed as new "features" from combining x_i's

---

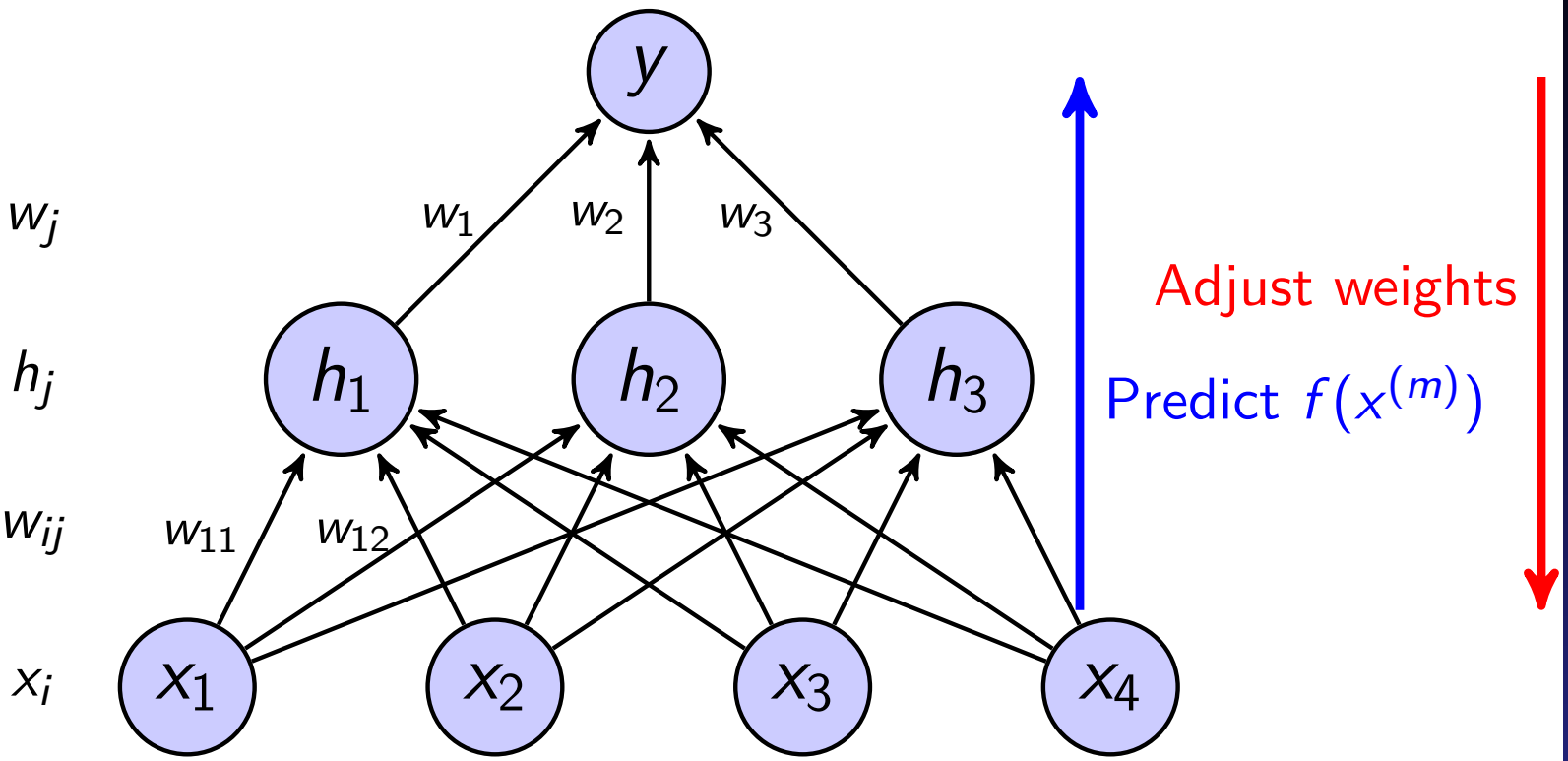
Called Multilayer Perceptron (MLP), but more like multilayer logistic regression

Expressive Power of Non-linearity

- A deeper architecture is more expressive than a shallow one given same number of nodes [Bishop, 1995]
  - 1-layer nets only model linear hyperplanes
  - 2-layer nets can model any continuous function (given sufficient nodes)
  - >3-layer nets can do so with fewer nodes



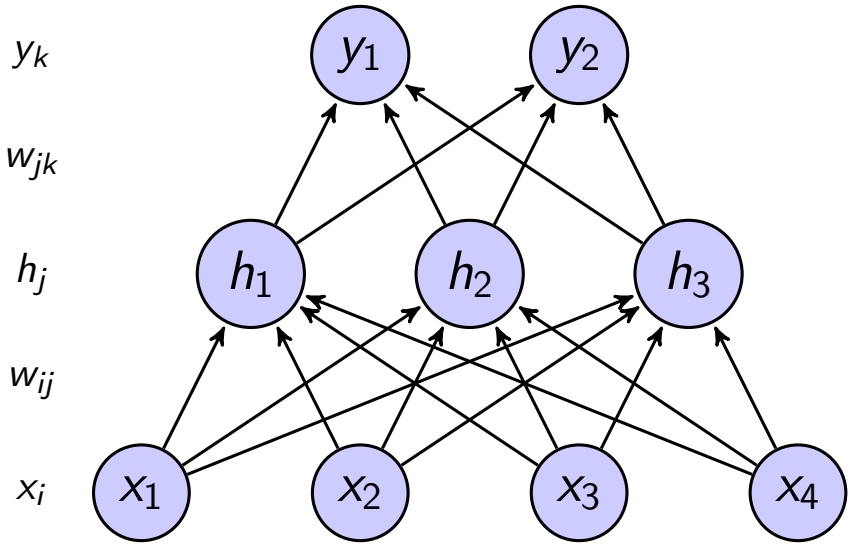
Training Neural Nets: Back-propagation



1. For each sample, compute
$f(x^{(m)}) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i^{(m)}))$
2. If $f(x^{(m)}) \neq y^{(m)}$, back-propagate error and adjust weights $\{w_{ij}, w_j\}$.

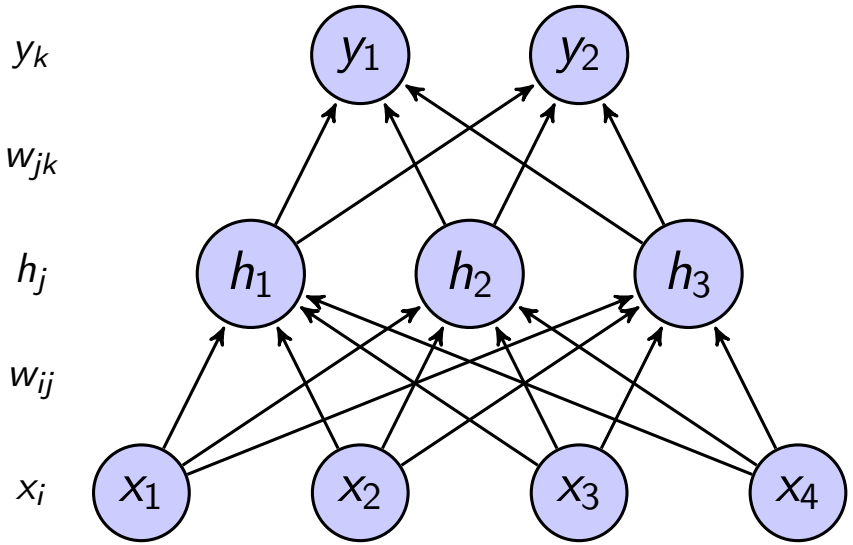
Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2$



Derivatives of the weights

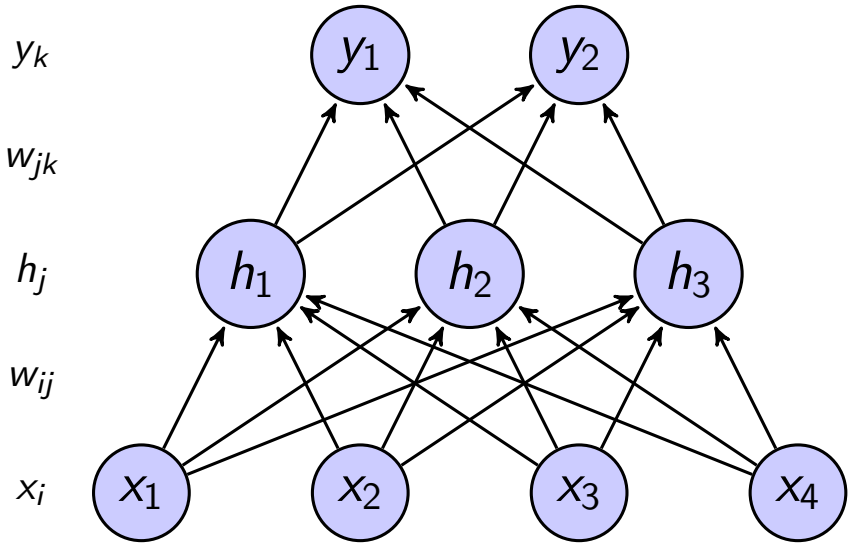
Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2$



$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2$

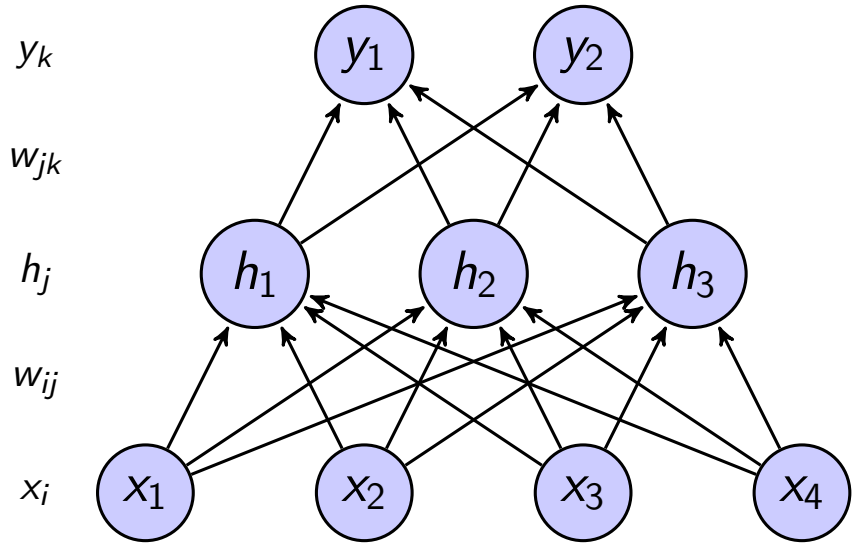


$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2$



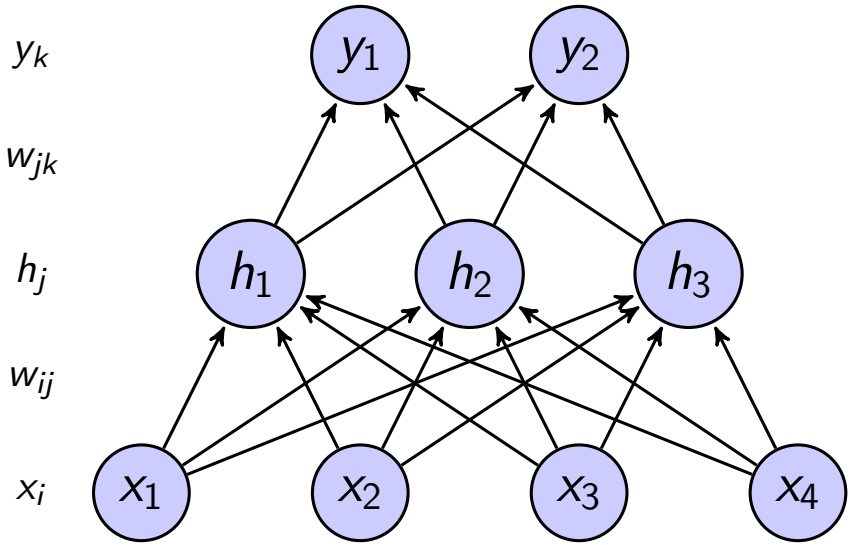
$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_k = \frac{\partial}{\partial in_k} \left(\sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2 \right) = [\sigma(in_k) - y_k] \, \sigma'(in_k)$$

Derivatives of the weights

Assume two outputs (y_1, y_2) per input x,
and loss per sample: $Loss = \sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2$



$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial (\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial (\sum_j w_{ij} x_i)}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_k = \frac{\partial}{\partial in_k} \left(\sum_k \frac{1}{2} [\sigma(in_k) - y_k]^2 \right) = [\sigma(in_k) - y_k] \sigma'(in_k)$$

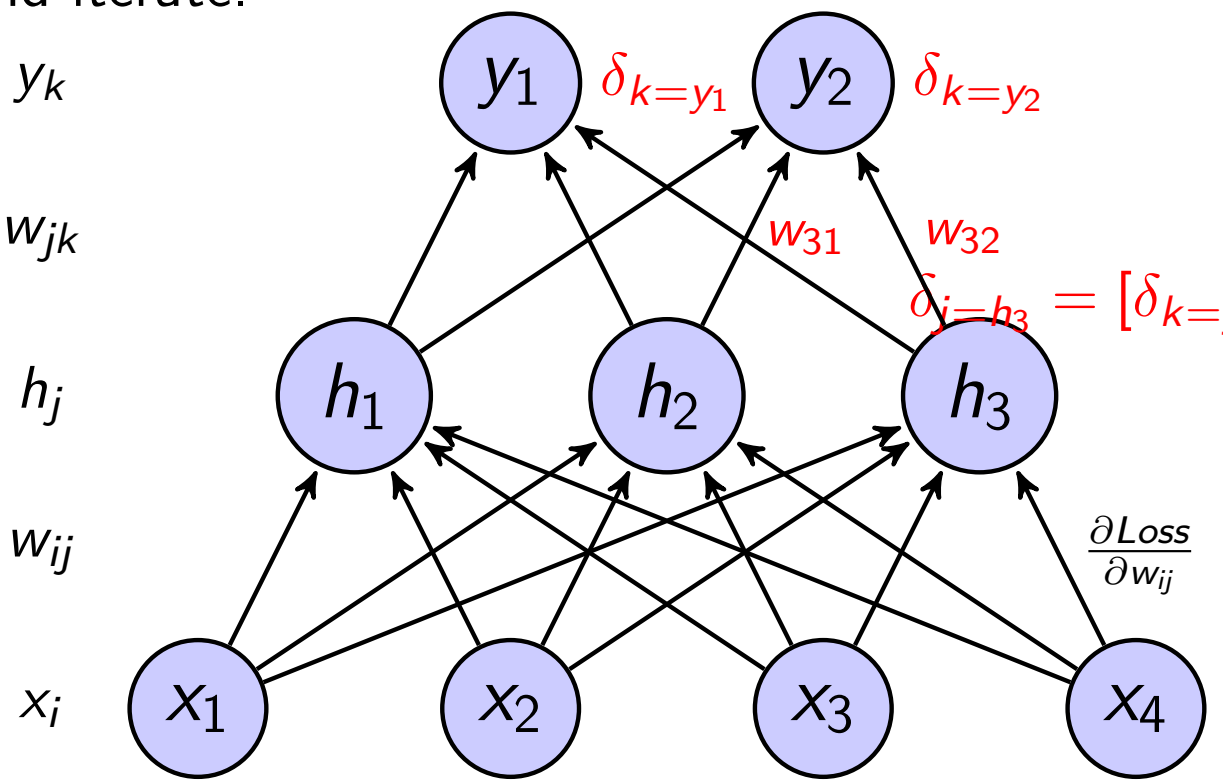
$$\delta_j = \sum_k \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial in_j} = \sum_k \delta_k \cdot \frac{\partial}{\partial in_j} \left(\sum_j w_{jk} \sigma(in_j) \right) = [\sum_k \delta_k w_{jk}] \sigma'(in_j)$$

Training Neural Nets: Back-propagation

All updates involve some scaled error from output $*$ input feature:

- $\frac{\partial Loss}{\partial w_{jk}} = \delta_k h_j$ where $\delta_k = [\sigma(in_k) - y_k]\,\sigma'(in_k)$
- $\frac{\partial Loss}{\partial w_{ij}} = \delta_j x_i$ where $\delta_j = [\sum_k \delta_k w_{jk}]\,\sigma'(in_j)$

First compute δ_k from final layer, then δ_j for previous layer and iterate.



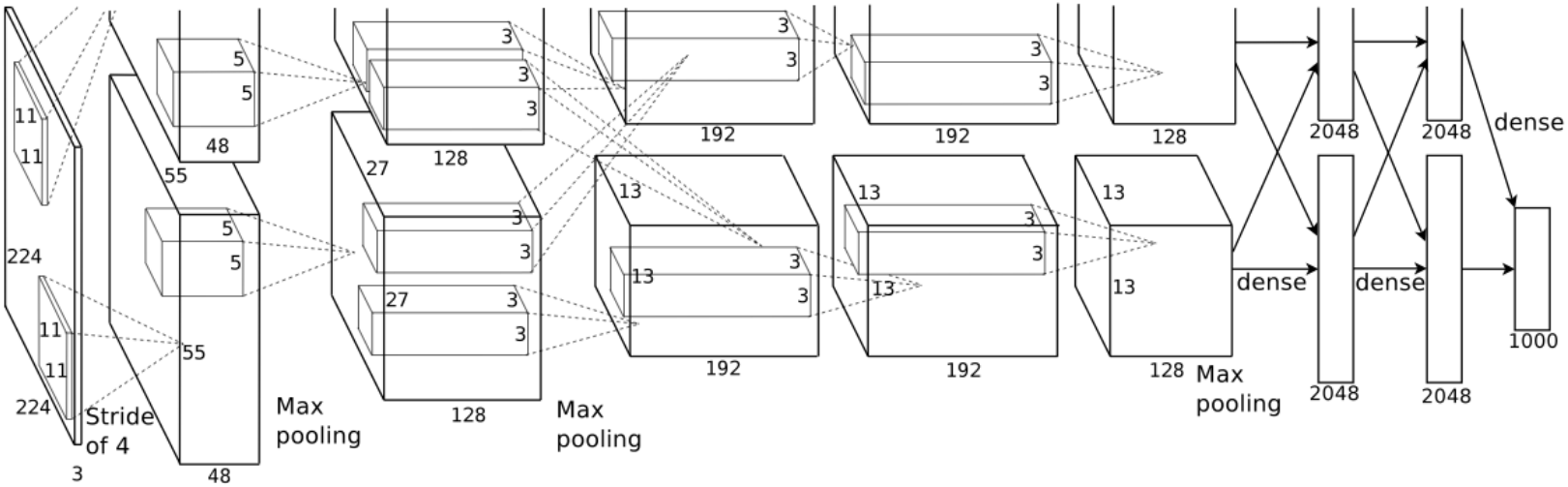
$$\delta_{j=h_3} = [\delta_{k=y_1} w_{31} + \delta_{k=y_2} w_{32}]\sigma'(in_{h_3})$$

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Current models are becoming more complex

Do you want to take the derivative of this?



- AlexNet for image classification [Krizhevsky et al., 2012]

Computation Graphs

- All computations can be viewed on a graph:
- e.g. 2-layer MLP:
  $y = softmax(W^{(2)} \cdot \sigma(W^{(1)} \cdot X + B^{(1)}) + B^{(2)})$

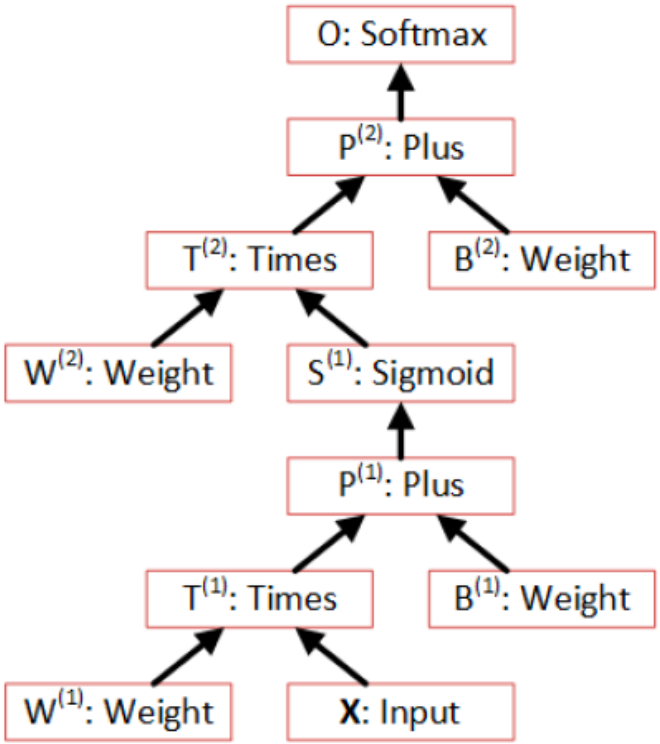


Figure from MSR CNTK tech report [Yu et al., 2014]

*$softmax(a_k) = \exp(a_k)/\sum_j \exp(a_j)$

Computation Graphs

- All computations can be viewed on a graph:
- e.g. 2-layer MLP with shared weights:
  $y = softmax(W^{(1)} \cdot \sigma(W^{(1)} \cdot X + B^{(1)}) + B^{(2)})$

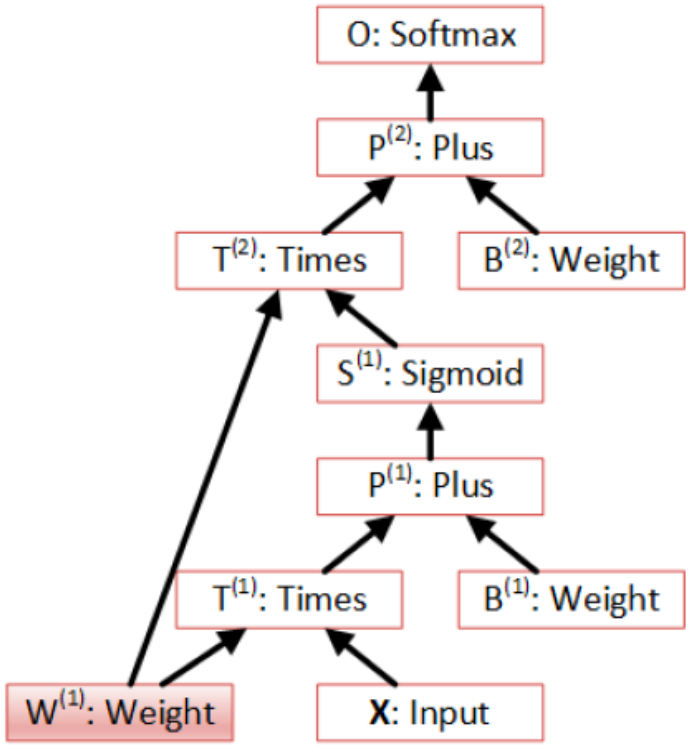


Figure from MSR CNTK tech report [Yu et al., 2014]

*$softmax(a_k) = \exp(a_k)/\sum_j \exp(a_j)$

Computation Graphs

- All computations can be viewed on a graph:
- e.g. Computing the loss term of the MLP

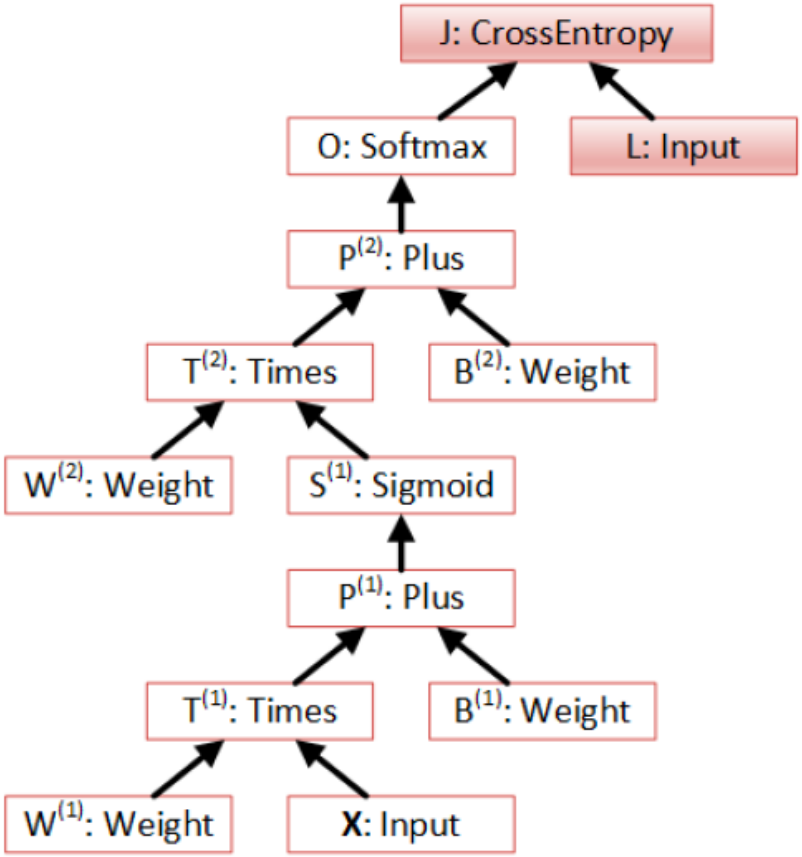


Figure from MSR CNTK tech report [Yu et al., 2014]

*$softmax(a_k) = \exp(a_k)/\sum_j \exp(a_j)$

Toolkits enable rapid experimentation of various architectures

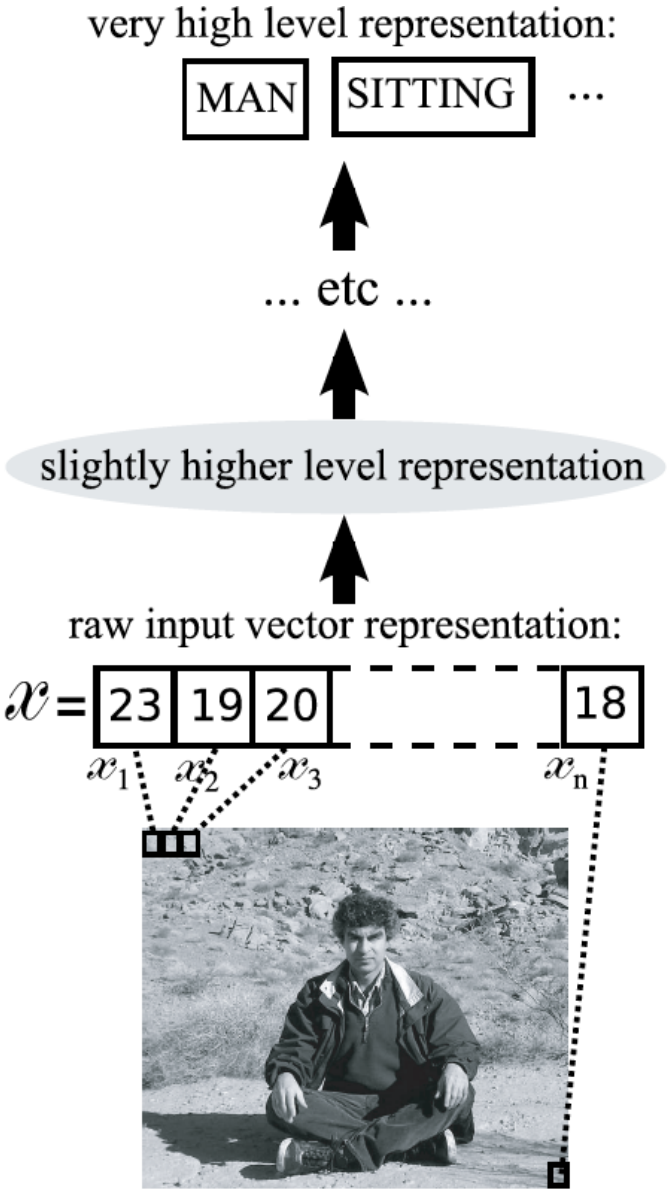
Popular toolkits: Theano, Torch, pylearn2, Caffe, CNTK, CNN, Kaldi, etc.

1. Compiles mathematical expression to graph
2. Implements basic operation for each node
3. Handles scheduling and batching of foward/backward computation
4. Also, GPU or fast math library support

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Potential of Deep Architecture



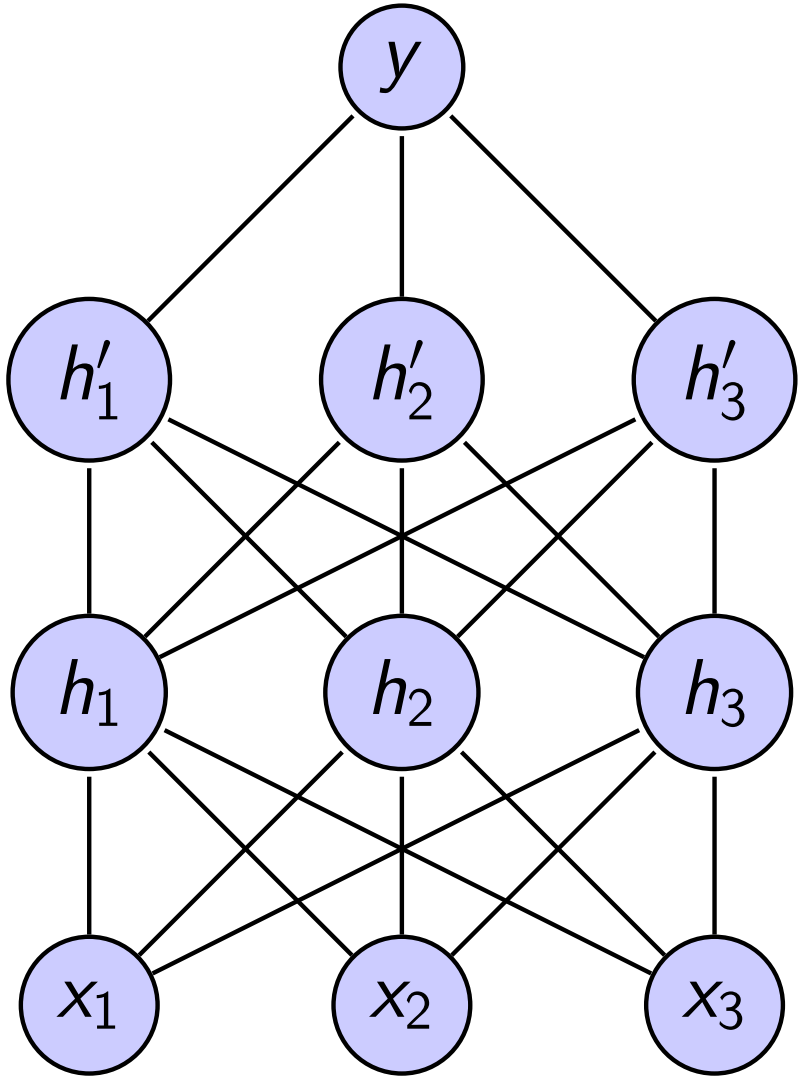
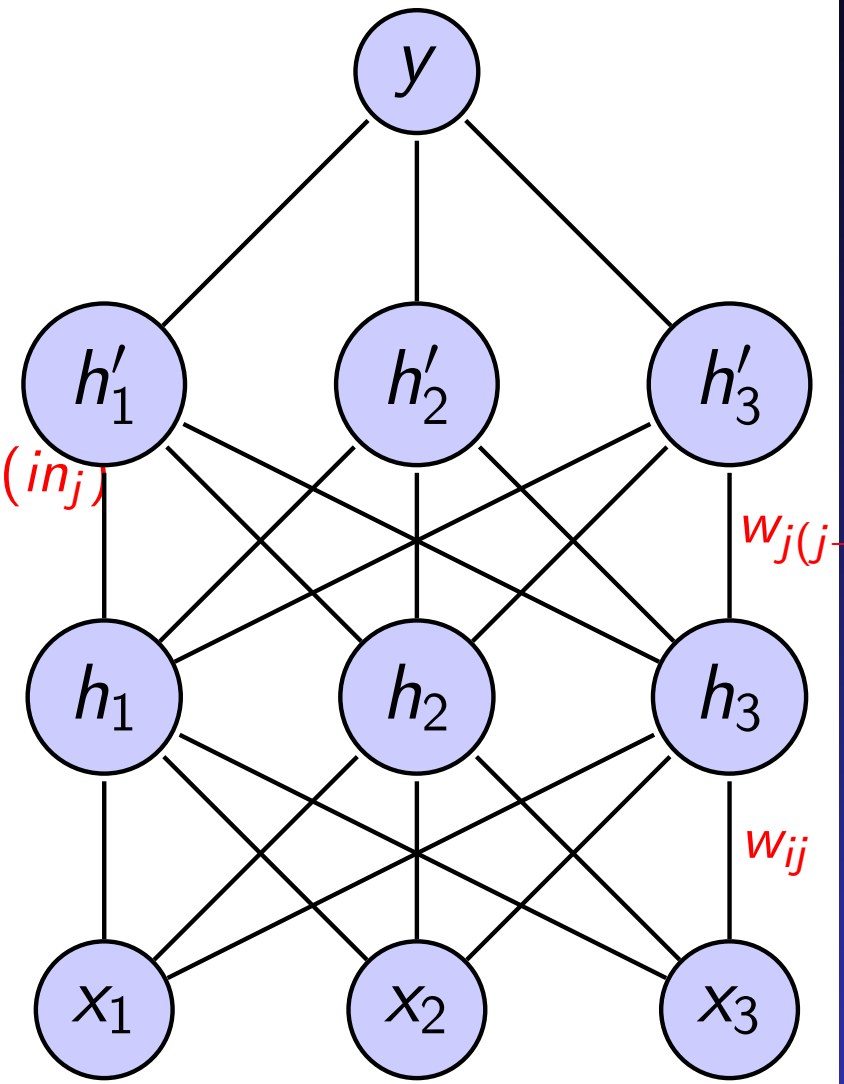


*Figure from [Bengio, 2009]

Difficulties of Deep Architecture

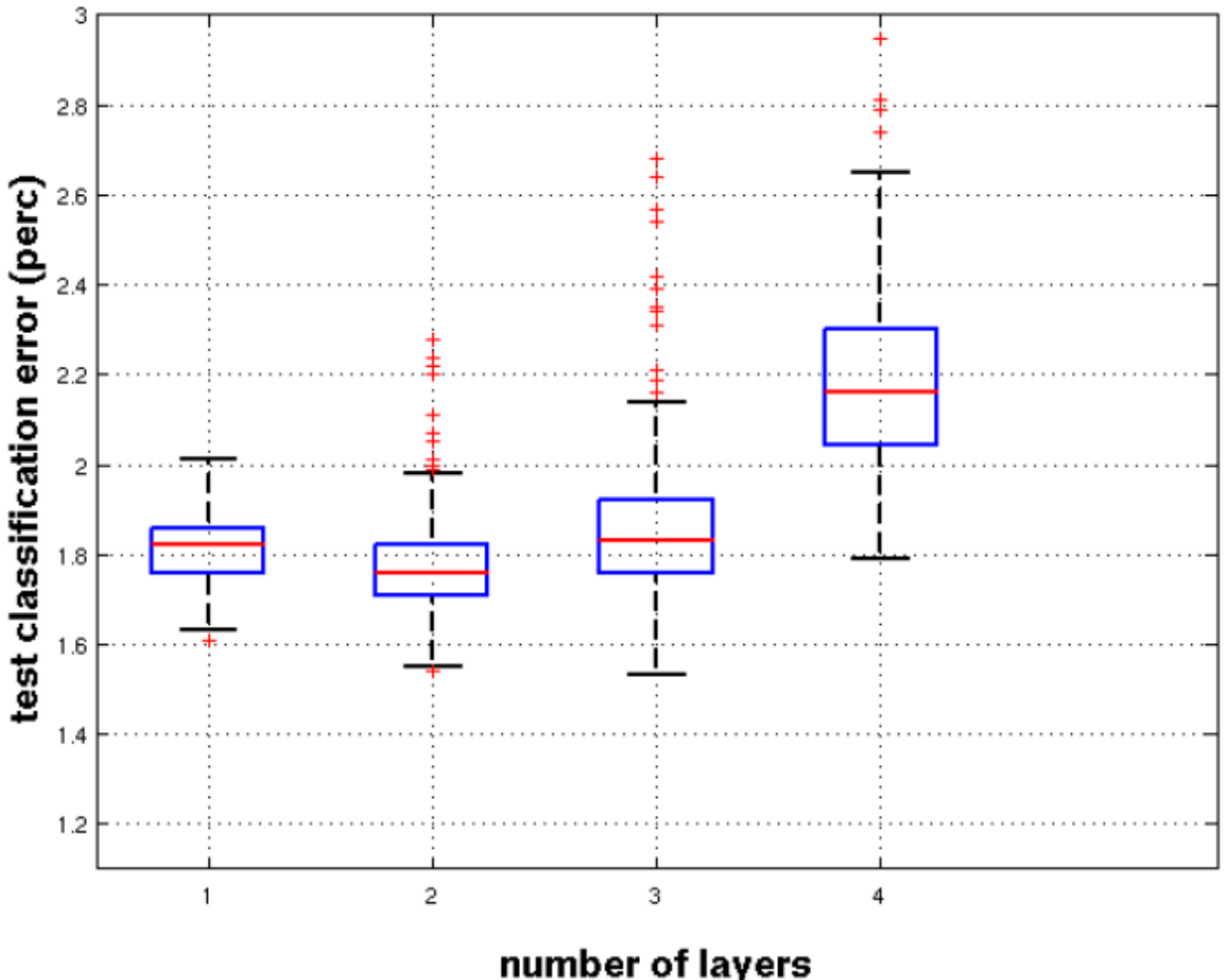
Vanishing gradient problem in Backpropagation

- $\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}} = \delta_j x_i$
- $\delta_j = \left[\sum_{j+1} \delta_{j+1} w_{j(j+1)}\right] \sigma'(in_j)$
- δ_j may vanish after repeated multiplication
- Also, exploding gradient problem!



Training Difficulties [Erhan et al., 2009]

- MNIST digit classification task
- Train neural net by Backpropagation (random initialization of w_{ij})

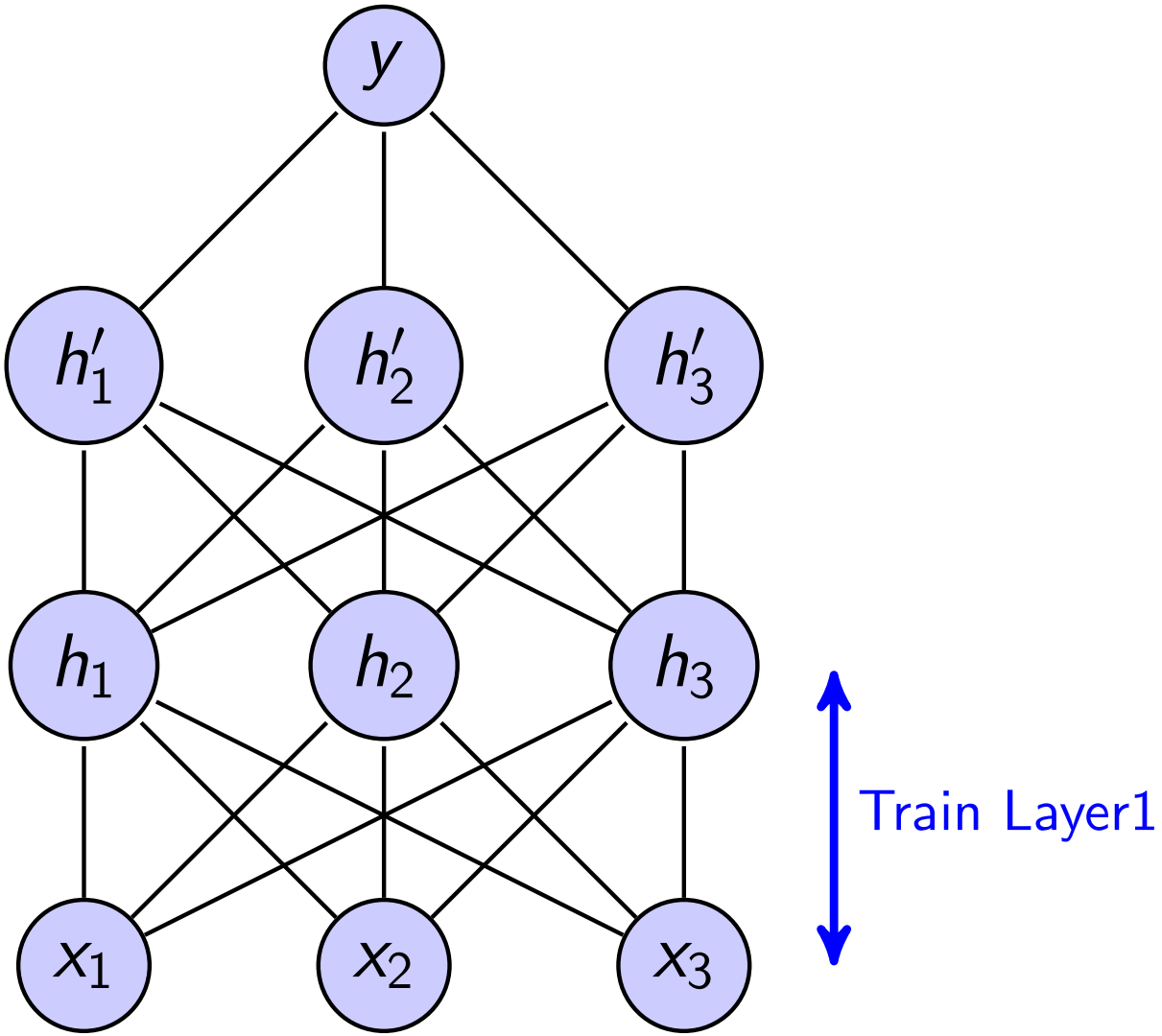


Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

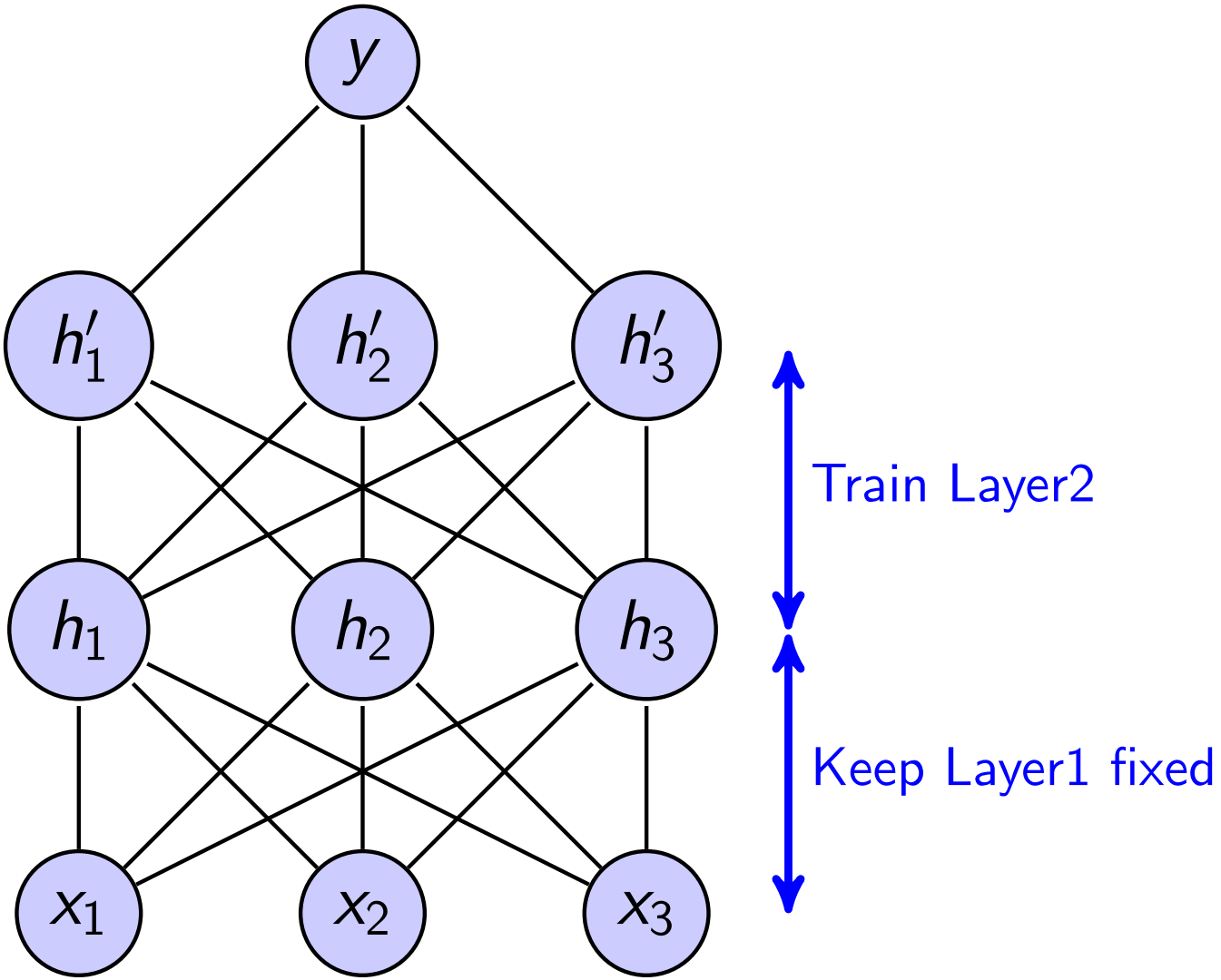
Layer-wise Pre-training [Hinton et al., 2006]

First, train one layer at a time, optimizing data-likelihood objective $P(x)$



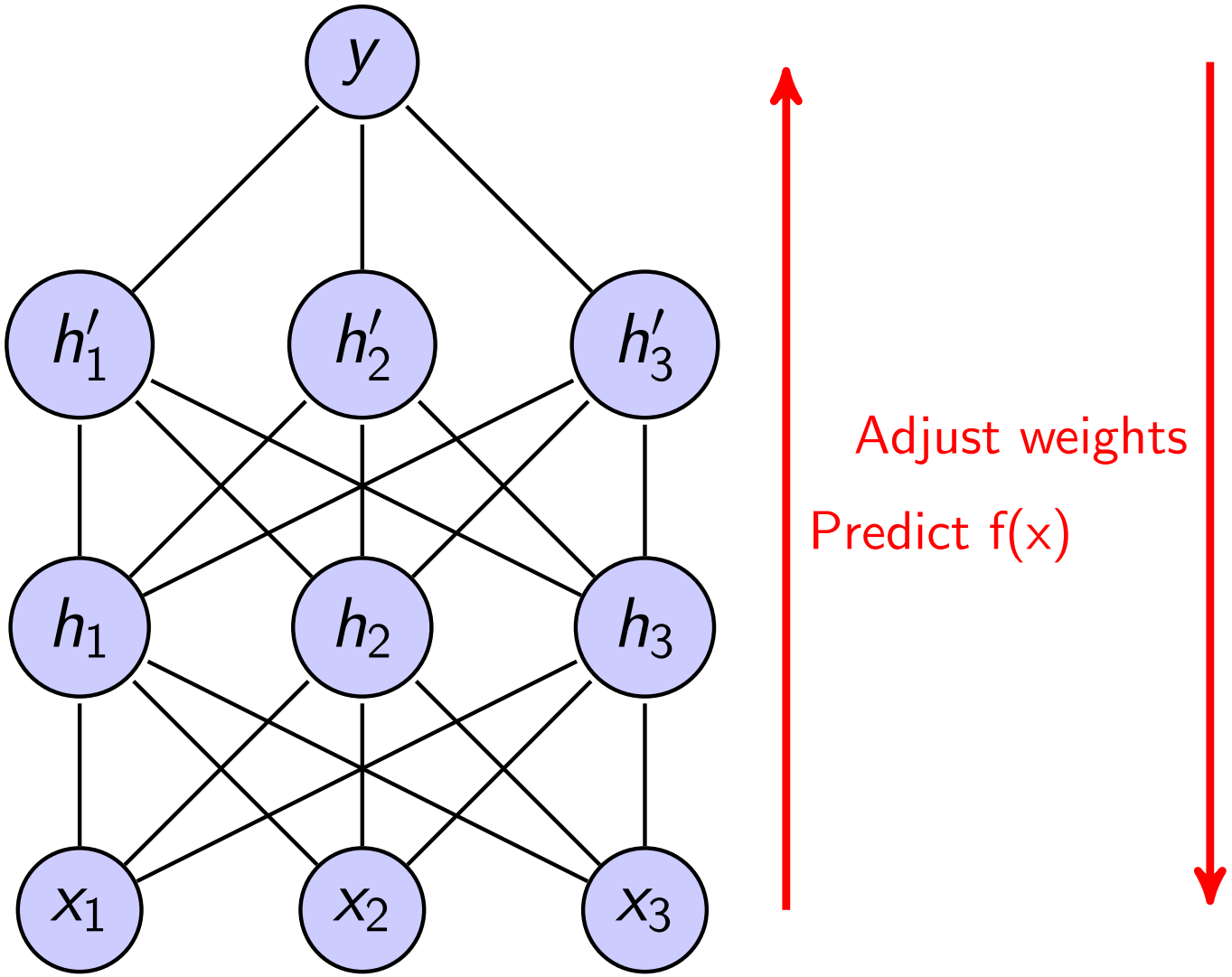
Layer-wise Pre-training [Hinton et al., 2006]

First, train one layer at a time, optimizing data-likelihood objective $P(x)$



Layer-wise Pre-training [Hinton et al., 2006]

Finally, fine-tune labeled objective $P(y|x)$ by Backpropagation



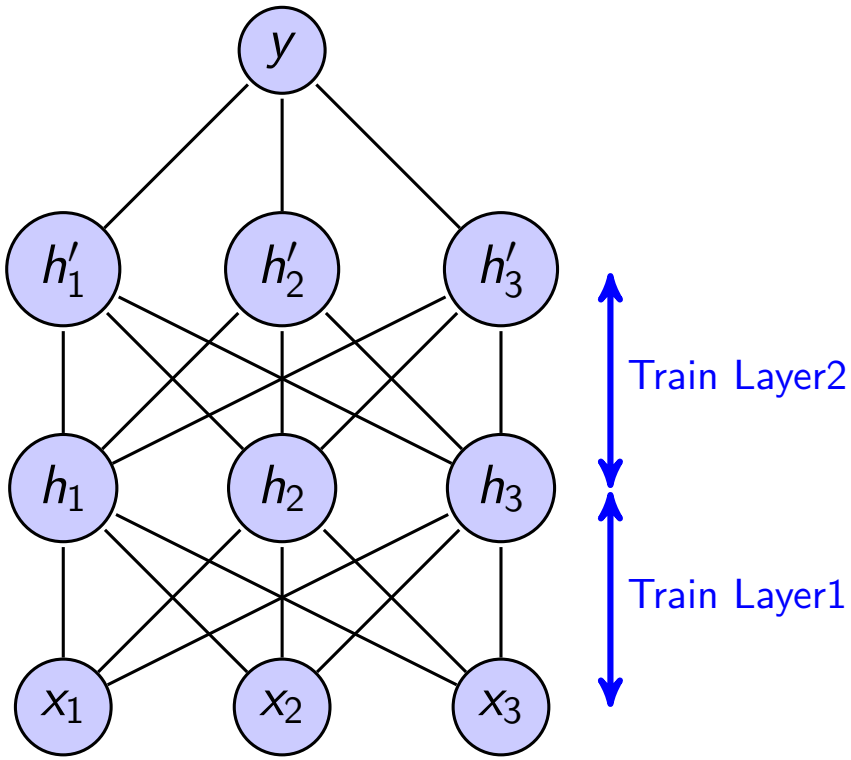
Layer-wise Pre-training [Hinton et al., 2006]

Key Idea:

Focus on modeling the input $P(X)$ better with each successive layer.

Worry about optimizing the task $P(Y|X)$ later.

> *"If you want to do computer vision, first learn computer graphics." – Geoff Hinton*



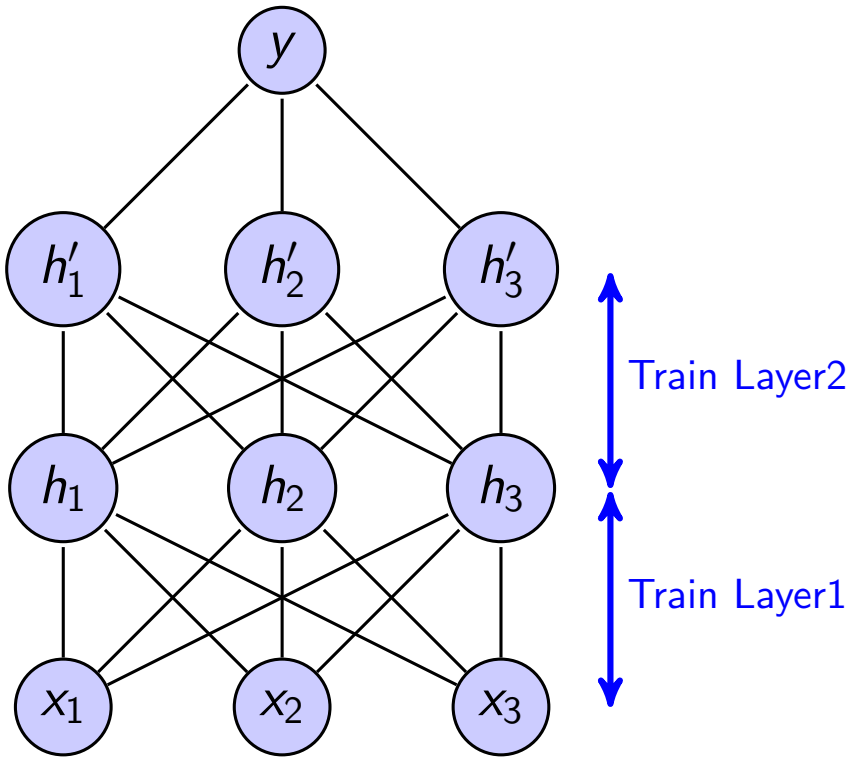
Layer-wise Pre-training [Hinton et al., 2006]

Key Idea:

Focus on modeling the input $P(X)$ better with each successive layer.

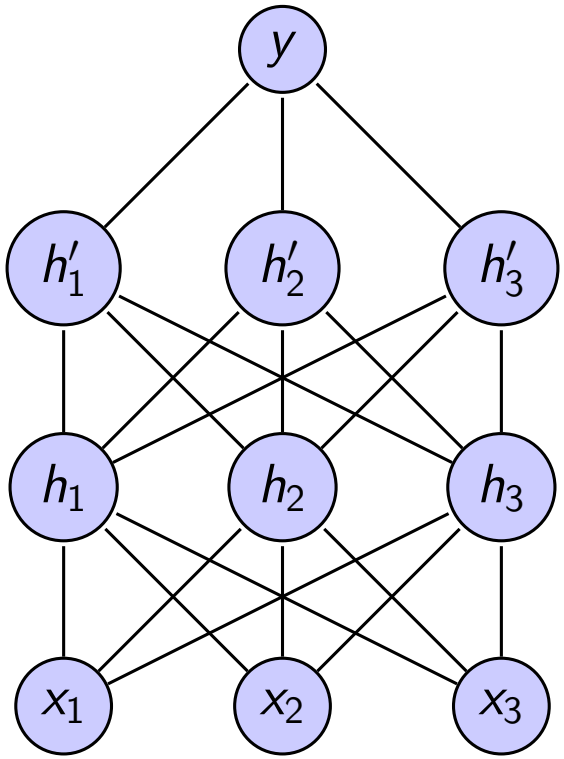
Worry about optimizing the task $P(Y|X)$ later.

> *"If you want to do computer vision, first learn computer graphics." – Geoff Hinton*



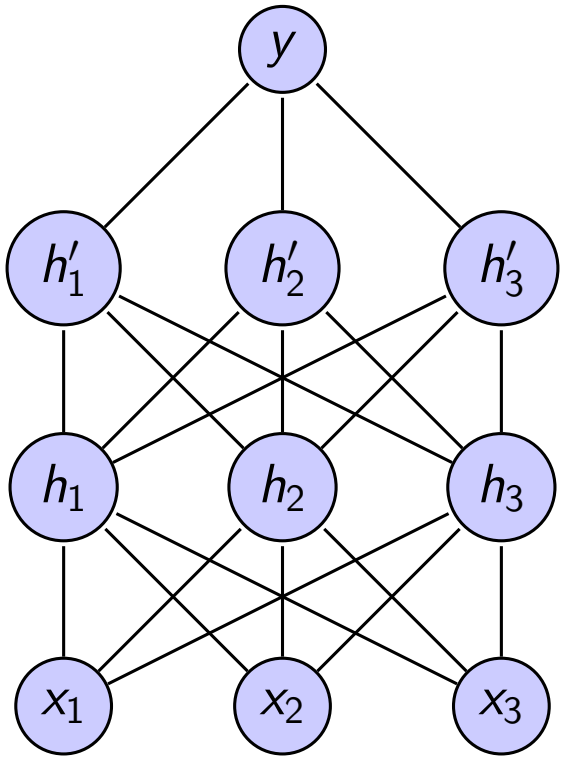
Extra advantage: Can exploit large amounts of unlabeled data!

Why does Pre-Training work?



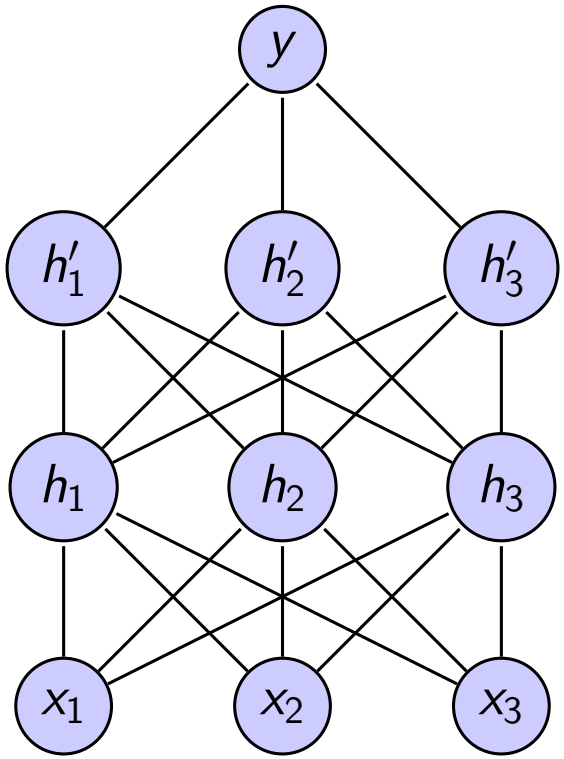
- [Erhan et al., 2010] - A deep net can fit the training data in many ways (non-convex):
  1. By optimizing upper-layers
  2. By optimizing lower-layers

Why does Pre-Training work?



- [Erhan et al., 2010] - A deep net can fit the training data in many ways (non-convex):
  1. By optimizing upper-layers
  2. By optimizing lower-layers
- Top-down vs. Bottom-up information
  1. Even if lower-layers are random weights, upper-layer may still fit well. But may not generalize to new data
  2. Pre-training with objective on $P(x)$ learns more generalizable features

Why does Pre-Training work?



- [Erhan et al., 2010] - A deep net can fit the training data in many ways (non-convex):
  1. By optimizing upper-layers
  2. By optimizing lower-layers
- Top-down vs. Bottom-up information
  1. Even if lower-layers are random weights, upper-layer may still fit well. But may not generalize to new data
  2. Pre-training with objective on $P(x)$ learns more generalizable features
- Pre-training maybe put weights at a better local optimum

Is Pre-Training really necessary?

Is Pre-Training really necessary?

Answer in 2006: Yes!

Is Pre-Training really necessary?

Answer in 2006: Yes!
Answer now: No!

- On large data, back-prop seems to work.
- Clever architectures avoid vanishing/exploding gradient
- While pre-training may not be needed for training deep architectures, they may help anyway (e.g. word embeddings for neural net parsers [Chen and Manning, 2014]).

Is Pre-Training really necessary?

Answer in 2006: Yes!
Answer now: No!

- On large data, back-prop seems to work.
- Clever architectures avoid vanishing/exploding gradient
- While pre-training may not be needed for training deep architectures, they may help anyway (e.g. word embeddings for neural net parsers [Chen and Manning, 2014]).

Lesson: What we believe true today may not be true tomorrow. Don't follow dogma.

Section Summary

- Neural Networks (1-layer, 2-layer)
- Computation Graphs and Deep Learning Toolkits
- Why Deep Architectures are Hard
- The Breakthrough in 2006

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Deep Learning Paradigm

- Recall problem setup: Learn function $f : x \rightarrow y$

Deep Learning Paradigm

- Recall problem setup: Learn function $f : x \rightarrow y$
- First learn hidden features h that model input, i.e. $x \rightarrow h \rightarrow y$

Deep Learning Paradigm

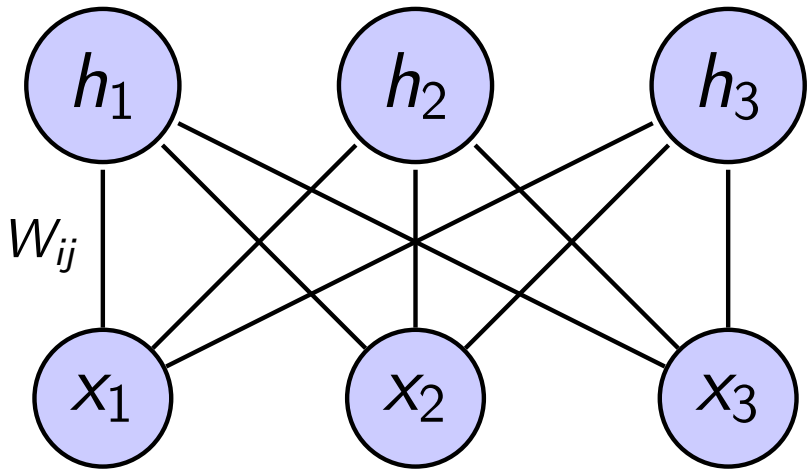
- Recall problem setup: Learn function $f : x \rightarrow y$
- First learn hidden features h that model input, i.e. $x \rightarrow h \rightarrow y$
- How to discover useful latent features h from data x?
  - We'll focus on unsupervised techniques, e.g. use Restricted Boltzmann Machines (RBMs)

Restricted Boltzmann Machine (RBM)

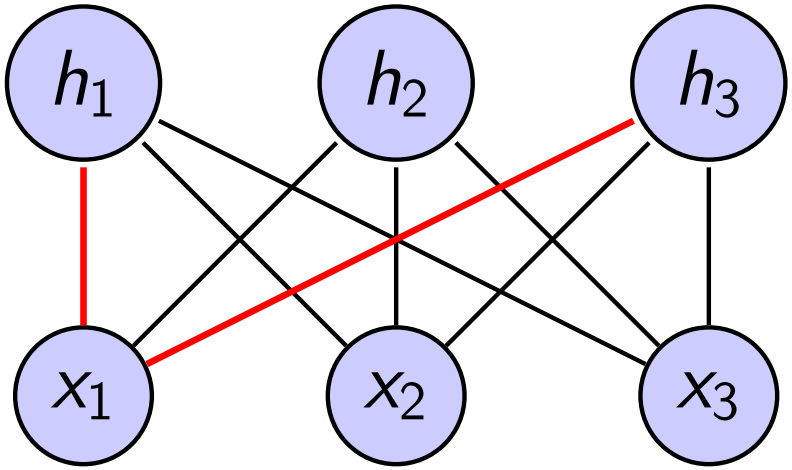
- RBM is a probabilistic model on binary variables h_j, x_i:

$$\begin{aligned} p(x,h) &= \frac{1}{Z_\theta}\exp\left(-E_\theta(x,h)\right) \\ &= \frac{1}{Z_\theta}\exp\left(x^T W h + b^T x + d^T h\right) \end{aligned}$$

  - W is a matrix; elements W_{ij} models correlation between x_i and h_j
  - b and d are bias terms; we'll assume $b = d = 0$ here.
  - normalizer (partition function): $Z_\theta = \sum_{(x,h)} \exp(-E_\theta(x,h))$



Restricted Boltzmann Machine (RBM): Example



Let W_{ij} on edges $(h_1, x_1), (h_3, x_1)$ be big positive, others be small negative.

x_1	x_2	x_3	h_1	h_2	h_3	$p(x,h) = \frac{1}{Z_\theta} \exp\left(x^T W h\right)$
1	0	0	1	0	1	highest
1	0	0	0	0	1	high
1	0	0	1	0	0	high
0	0	0	1	0	1	low
0	1	0	0	0	0	low
0	0	1	0	0	0	low

etc

RBM Posterior Distribution (Easy!)

- Computing $p(h|x)$ is easy due to factorization:

$$
\begin{aligned}
p(h|x) &= \frac{p(x,h)}{\sum_h p(x,h)} = \frac{1/Z_\theta \exp(-\mathrm{E}(\mathrm{x},\mathrm{h}))}{\sum_h 1/Z_\theta \exp(-\mathrm{E}(\mathrm{x},\mathrm{h}))} \\
&= \frac{\exp(x^T W h + b^T x + d^T h)}{\sum_h \exp(x^T W h + b^T x + d^T h)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)}{\sum_{h_1 \in \{0,1\}} \sum_{h_2 \in \{0,1\}} \cdots \sum_{h_j} \prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j)}{\prod_j \sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} \\
&= \prod_j \frac{\exp(x^T W_j h_j + d_j h_j)}{\sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} = \prod_j p(h_j|x)
\end{aligned}
$$

- $p(h_j = 1|x) = \exp(x^T W_j + d_j)/Z = \sigma(x^T W_j + d_j)$ is Logistic Regression!

RBM Posterior Distribution (Easy!)

- Computing $p(h|x)$ is easy due to factorization:

$$
\begin{aligned}
p(h|x) &= \frac{p(x,h)}{\sum_h p(x,h)} = \frac{1/Z_\theta \exp(-\mathrm{E}(\mathrm{x},\mathrm{h}))}{\sum_h 1/Z_\theta \exp(-\mathrm{E}(\mathrm{x},\mathrm{h}))} \\
&= \frac{\exp(x^T W h + b^T x + d^T h)}{\sum_h \exp(x^T W h + b^T x + d^T h)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)}{\sum_{h_1 \in \{0,1\}} \sum_{h_2 \in \{0,1\}} \cdots \sum_{h_j} \prod_j \exp(x^T W_j h_j + d_j h_j) \cdot \exp(b^T x)} \\
&= \frac{\prod_j \exp(x^T W_j h_j + d_j h_j)}{\prod_j \sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} \\
&= \prod_j \frac{\exp(x^T W_j h_j + d_j h_j)}{\sum_{h_j \in \{0,1\}} \exp(x^T W_j h_j + d_j h_j)} = \prod_j p(h_j|x)
\end{aligned}
$$

- $p(h_j = 1|x) = \exp(x^T W_j + d_j)/Z = \sigma(x^T W_j + d_j)$ is Logistic Regression!
- Similarly, computing $p(x|h) = \prod_i p(x_i|h)$ is easy

RBM Max-Likelihood Training (Hard!)

Derivative of the Log-Likelihood: $\partial_{w_{ij}} \log P_w(x = x^{(m)})$

$$= \partial_{w_{ij}} \log \sum_h P_w(x = x^{(m)}, h) \tag{1}$$

$$= \partial_{w_{ij}} \log \sum_h \frac{1}{Z_w} \exp\left(-E_w(x^{(m)}, h)\right) \tag{2}$$

$$= -\partial_{w_{ij}} \log Z_w + \partial_{w_{ij}} \log \sum_h \exp\left(-E_w(x^{(m)}, h)\right) \tag{3}$$

$$= \frac{1}{Z_w} \sum_{h,x} e^{(-E_w(x,h))} \partial_{w_{ij}} E_w(x, h) - \frac{1}{\sum_h e^{(-E_w(x^{(m)},h))}} \sum_h e^{(-E_w(x^{(m)},h))} \partial_{w_{ij}} E_w(x^{(m)}, h)$$

$$= \sum_{h,x} P_w(x, h)[\partial_{w_{ij}} E_w(x, h)] - \sum_h P_w(x^{(m)}, h)[\partial_{w_{ij}} E_w(x^{(m)}, h)] \tag{4}$$

$$= -\mathbb{E}_{p(x,h)}[x_i \cdot h_j] + \mathbb{E}_{p(h|x=x^{(m)})}[x_i^{(m)} \cdot h_j] \tag{5}$$

RBM Max-Likelihood Training (Hard!)

Derivative of the Log-Likelihood: $\partial_{w_{ij}} \log P_w(x = x^{(m)})$

$$= \partial_{w_{ij}} \log \sum_h P_w(x = x^{(m)}, h) \tag{1}$$

$$= \partial_{w_{ij}} \log \sum_h \frac{1}{Z_w} \exp\left(-E_w(x^{(m)}, h)\right) \tag{2}$$

$$= -\partial_{w_{ij}} \log Z_w + \partial_{w_{ij}} \log \sum_h \exp\left(-E_w(x^{(m)}, h)\right) \tag{3}$$

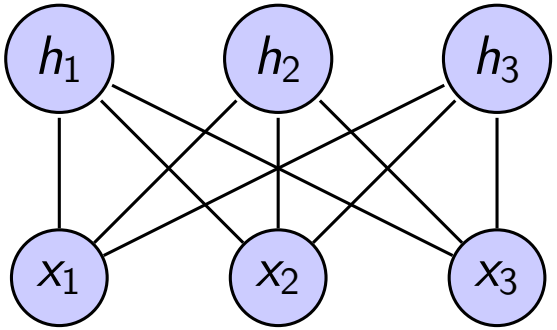
$$= \frac{1}{Z_w} \sum_{h,x} e^{(-E_w(x,h))} \partial_{w_{ij}} E_w(x, h) - \frac{1}{\sum_h e^{(-E_w(x^{(m)},h))}} \sum_h e^{(-E_w(x^{(m)},h))} \partial_{w_{ij}} E_w(x^{(m)}, h)$$

$$= \sum_{h,x} P_w(x, h)[\partial_{w_{ij}} E_w(x, h)] - \sum_h P_w(x^{(m)}, h)[\partial_{w_{ij}} E_w(x^{(m)}, h)] \tag{4}$$

$$= -\mathbb{E}_{p(x,h)}[x_i \cdot h_j] + \mathbb{E}_{p(h|x=x^{(m)})}[x_i^{(m)} \cdot h_j] \tag{5}$$

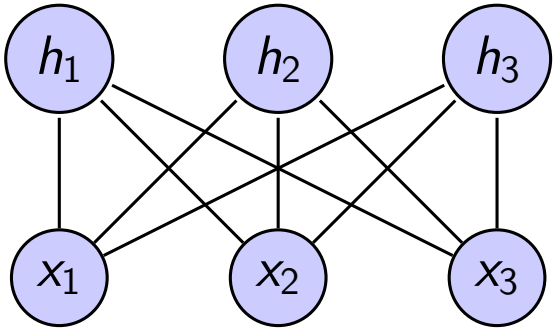
Second term (positive phase) increases probability of $x^{(m)}$;
First term (negative phase) decreases probability of samples generated by the model

Contrastive Divergence Algorithm



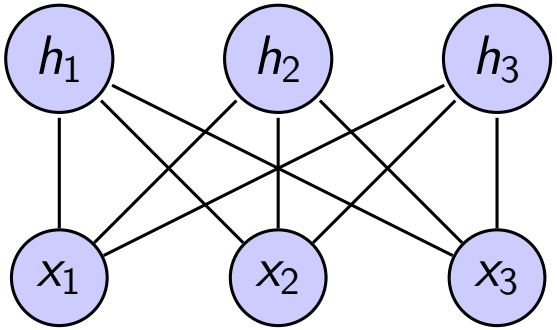
- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model

Contrastive Divergence Algorithm



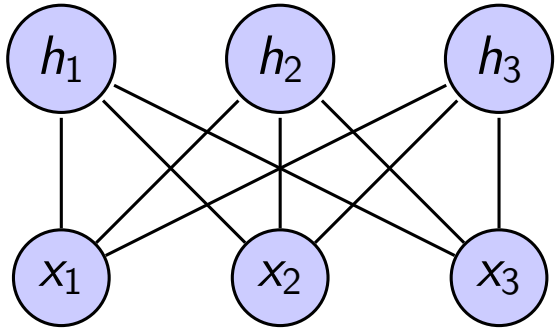
- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model
- Gibbs Sampling works but slow.

Contrastive Divergence Algorithm



- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model
- Gibbs Sampling works but slow.
- Contrastive Divergence (faster but biased):

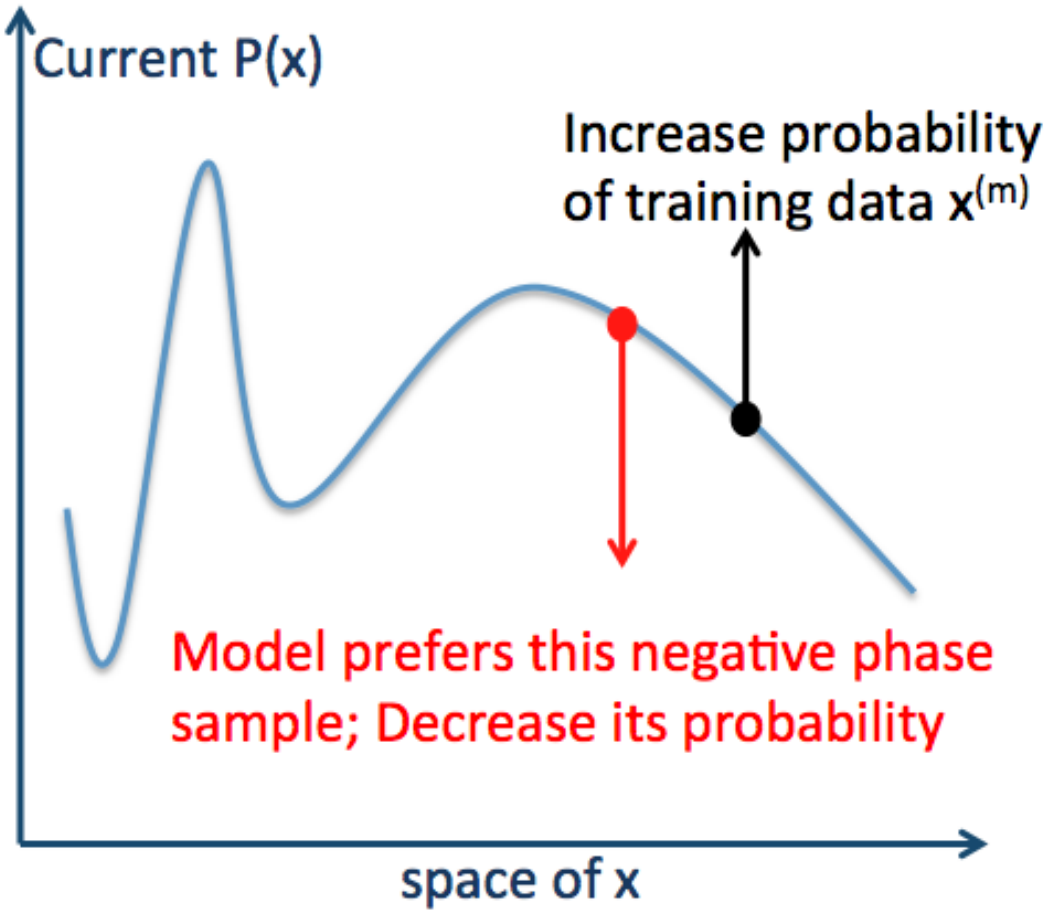
Contrastive Divergence Algorithm



- The negative phase term ($\mathbb{E}_{p(x,h)}[x_i \cdot h_j]$) is expensive because it requires sampling (x,h) from the model
- Gibbs Sampling works but slow.
- Contrastive Divergence (faster but biased):
  1. Let $x^{(m)}$ be training point, $W = [w_{ij}]$ be current model weights
  2. Sample $\hat{h}_j \in \{0, 1\}$ from $p(h_j | x = x^{(m)}) = \sigma(\sum_i w_{ij} x_i^{(m)} + d_j)$
  3. Sample $\tilde{x}_i \in \{0, 1\}$ from $p(x_i | h = \hat{h}) = \sigma(\sum_j w_{ij} \hat{h}_j + b_i)$
  4. Sample $\tilde{h}_j \in \{0, 1\}$ from $p(h_j | x = \tilde{x}) = \sigma(\sum_i w_{ij} \tilde{x}_i + d_j)$
  5. $w_{ij} \leftarrow w_{ij} + \gamma(x_i^{(m)} \cdot \hat{h}_j - \tilde{x}_i \cdot \tilde{h}_j)$

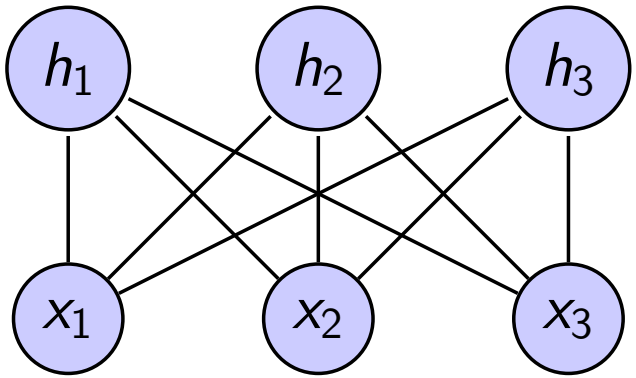
Contrastive Divergence Pictorial View

- Goal: Make RBM $p(x, h)$ have high probability on training samples
- To do so, we'll "steal" probability mass from nearby samples that incorrectly preferred by the model
- Analysis in [Carreira-Perpinan and Hinton, 2005]



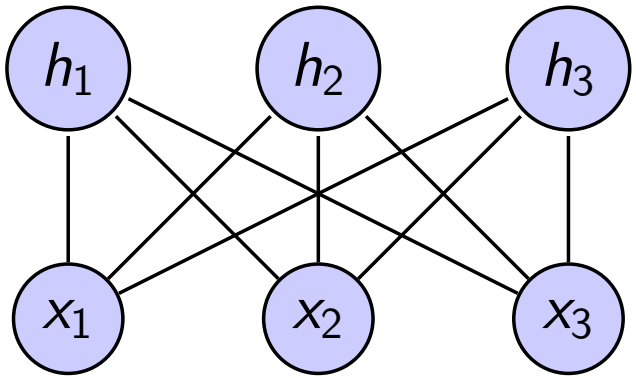
Distributed Representations learned by RBM

- Vector h act as **Distributed Representation** of data
  - Multiple h_j may be active simultaneously for a given x. (Multi-clustering)
  - $2^{|h|}$ possible representations with $|h| \times |x|$ parameters.

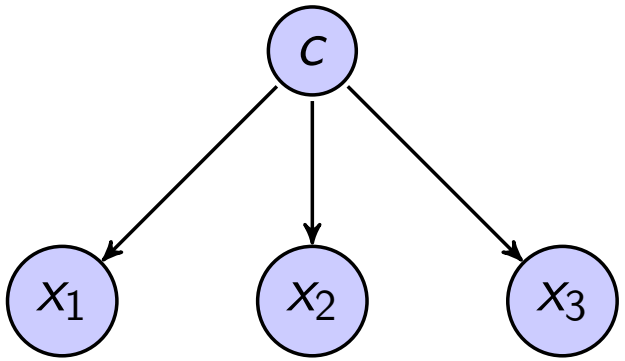


Distributed Representations learned by RBM

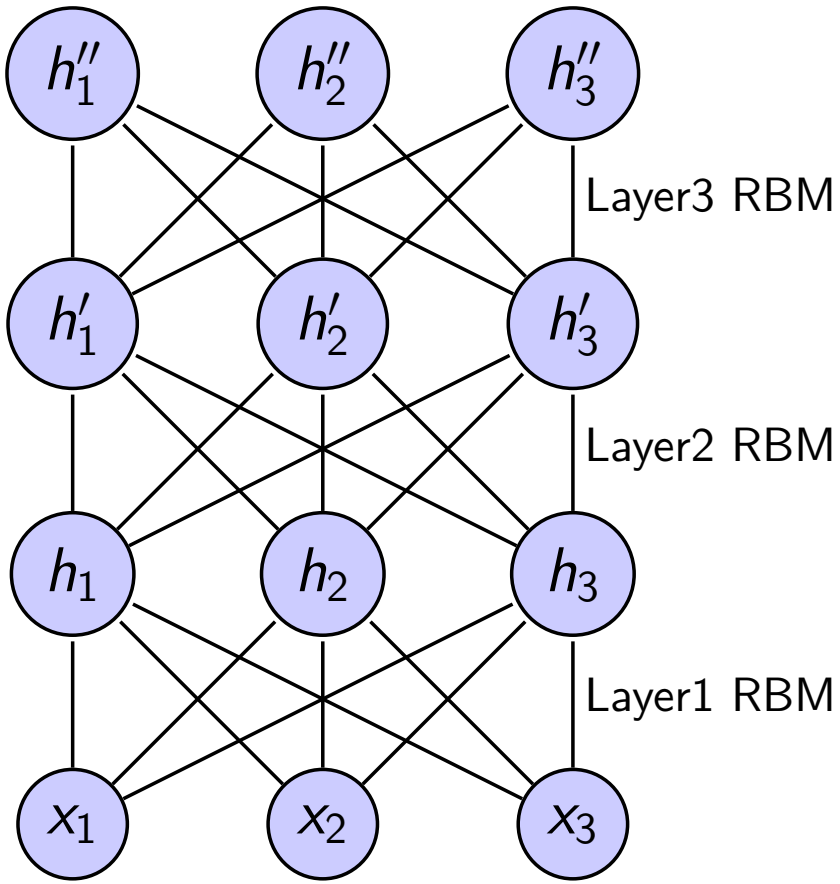
- Vector h act as **Distributed Representation** of data
  - Multiple h_j may be active simultaneously for a given x. (Multi-clustering)
  - $2^{|h|}$ possible representations with $|h| \times |x|$ parameters.



- A mixture model $p(x) = \sum_h p(c)p(x|c)$ would need $2^{|h|} \times |x|$ parameters:

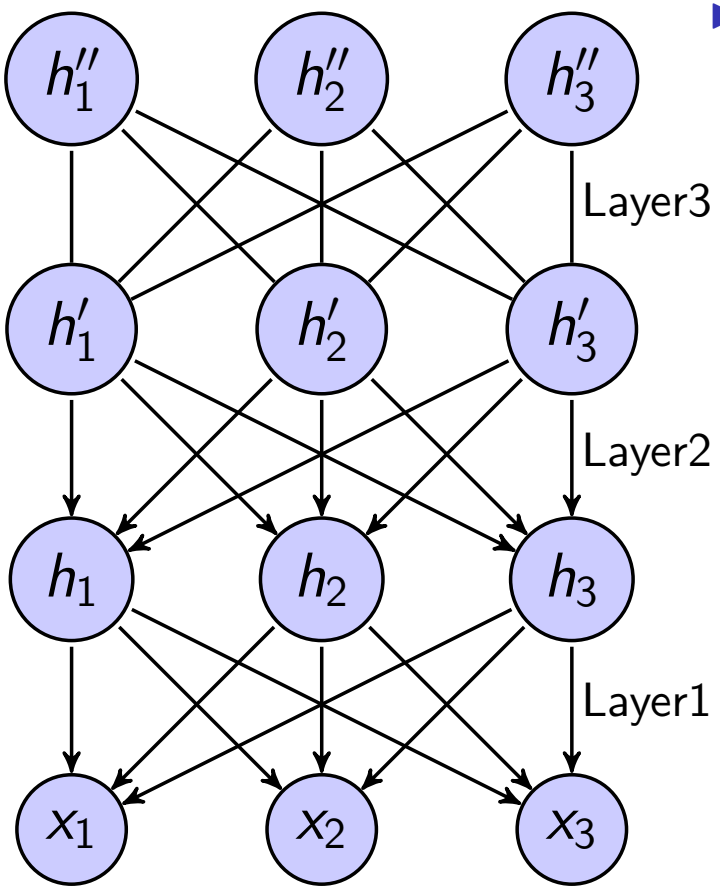


Layer-wise Pre-training of RBMs



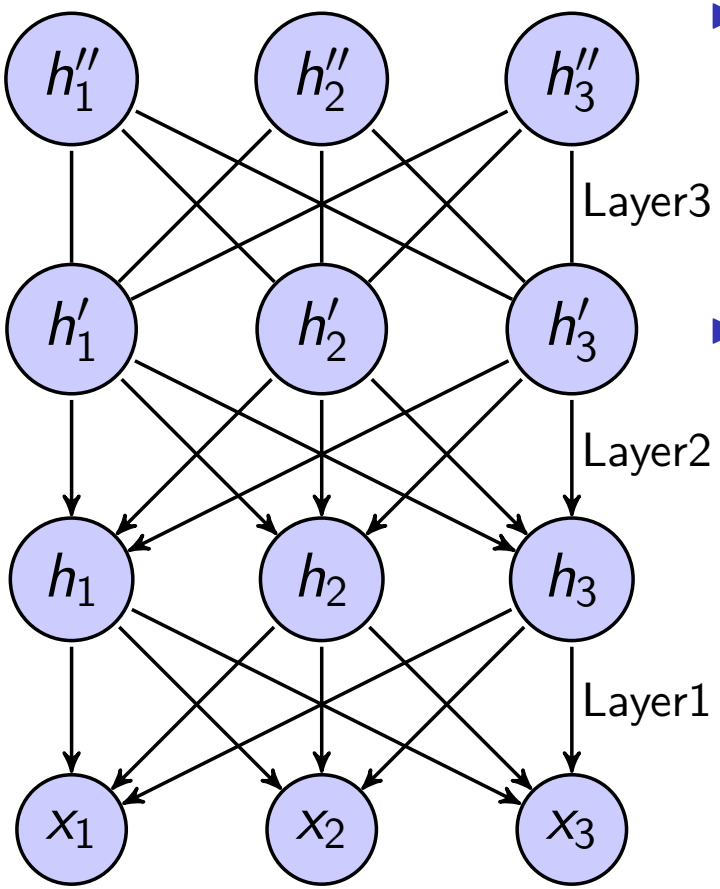
1. Train Layer 1 RBM
   max $\log P(x, h)$
2. Train Layer 2 RBM
   max $\log P(h, h')$
3. Train Layer 3 RBM
   max $\log P(h', h'')$

Deep Belief Nets (DBN) = Stacked RBM [Hinton et al., 2006]



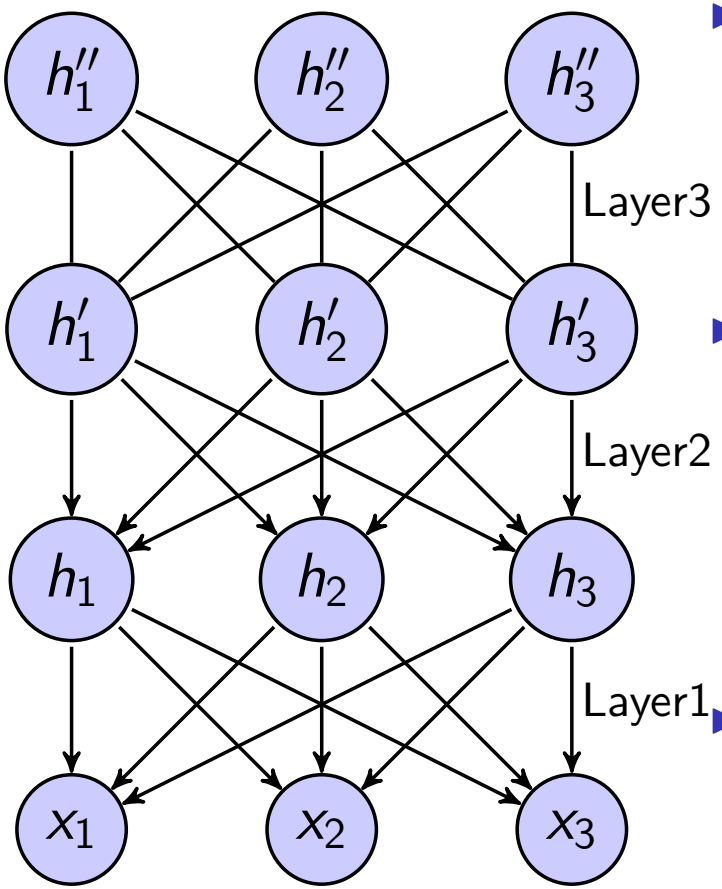
- After pre-training, stacked RBM can be converted to DBN, defined as: $p(x) = \sum_{h,h',h''} p(x|h)p(h|h')p(h',h'')$

Deep Belief Nets (DBN) = Stacked RBM [Hinton et al., 2006]



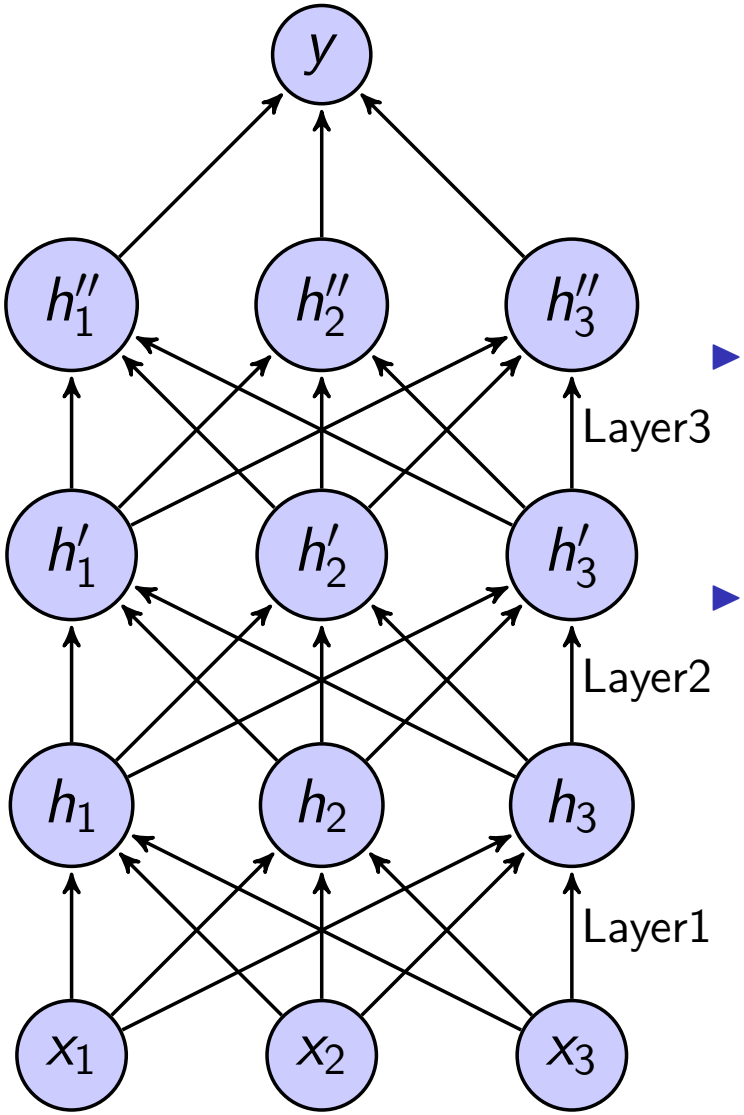
- After pre-training, stacked RBM can be converted to DBN, defined as: $p(x) = \sum_{h,h',h''} p(x|h)p(h|h')p(h',h'')$
- This is a probabilistic generative model: Can sample from RBM at Layer3, then generate data x via directed sigmoids

Deep Belief Nets (DBN) = Stacked RBM [Hinton et al., 2006]



- After pre-training, stacked RBM can be converted to DBN, defined as: $p(x) = \sum_{h,h',h''} p(x|h)p(h|h')p(h',h'')$
- This is a probabilistic generative model: Can sample from RBM at Layer3, then generate data x via directed sigmoids
- Upper layers can be viewed as prior; More layers = better prior

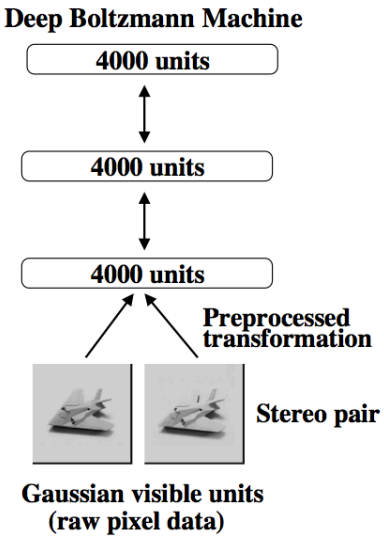
Popular alternative: Initialize MLP with RBMs

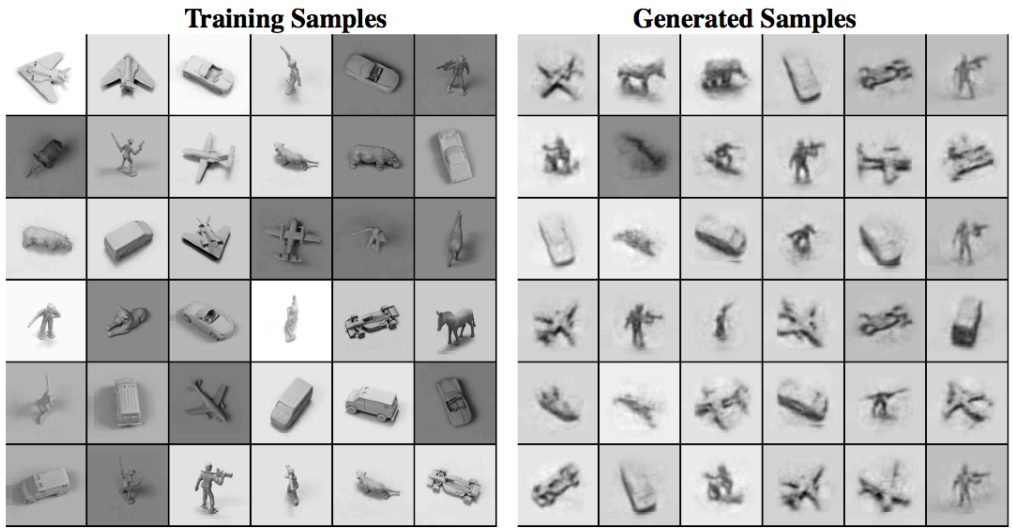


- Stacked RBMs can be used to initialize a Deep Neural Network (DNN), i.e. MLP
- Then, fine-tuned by back-propagation

What a Deep Generative Model can do

After training on 20k images, the generative model of [Salakhutdinov and Hinton, 2009]* can generate random images (dimension=8976) that are amazingly realistic!





This model is a Deep Boltzmann Machine (DBM), different from Deep Belief Nets (DBN) but also built by stacking RBMs.

RBM and DBN Summary

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.

RBM and DBN Summary

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not label. First model $p(x)$ to do better $p(y|x)$.

RBM and DBN Summary

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not label. First model $p(x)$ to do better $p(y|x)$.
3. Why RBM? $p(h|x)$ is tractable, so it's easy to stack.

RBM and DBN Summary

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not label. First model $p(x)$ to do better $p(y|x)$.
3. Why RBM? $p(h|x)$ is tractable, so it's easy to stack.
4. RBM training can be expensive. Solution: contrastive divergence

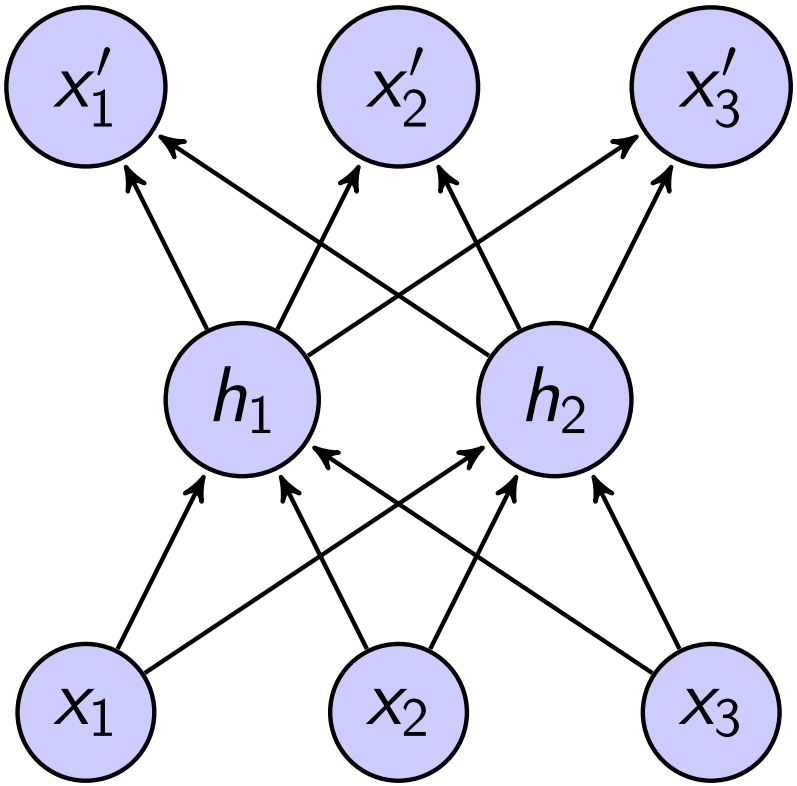
RBM and DBN Summary

1. Layer-wise pre-training is the innovation that rekindled interest in deep architectures.
2. Pre-training focuses on optimizing likelihood on the data, not label. First model $p(x)$ to do better $p(y|x)$.
3. Why RBM? $p(h|x)$ is tractable, so it's easy to stack.
4. RBM training can be expensive. Solution: contrastive divergence
5. We can stack RBMs to form a deep probabilistic generative model (DBN), or to initialize deep neural network (DNN)

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

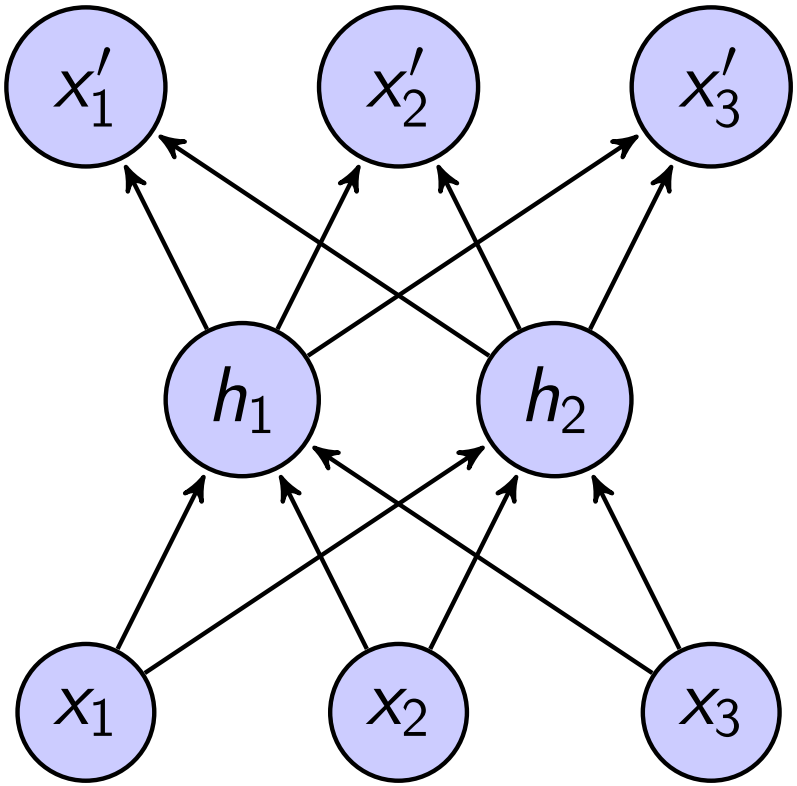
Auto-Encoders: Efficient replacement for RBM



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Auto-Encoders: Efficient replacement for RBM



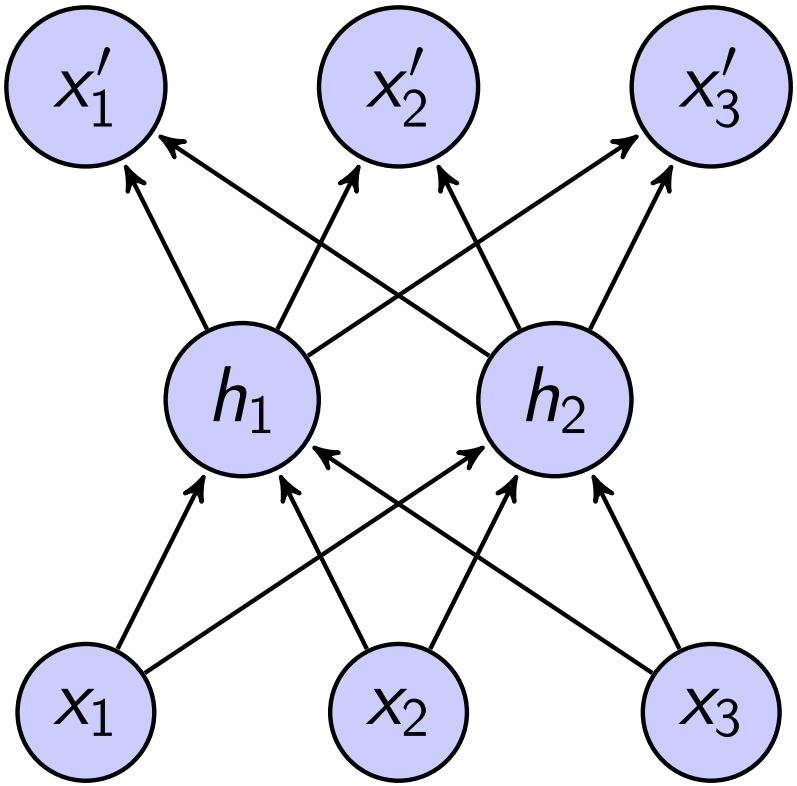
Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

- $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$

Auto-Encoders: Efficient replacement for RBM



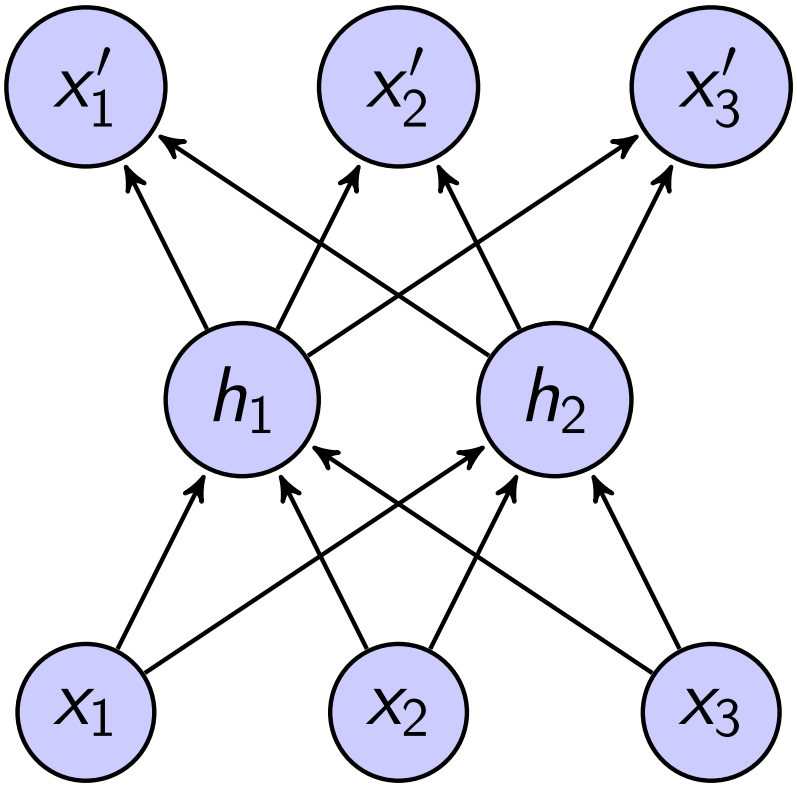
Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

- $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$

Auto-Encoders: Efficient replacement for RBM



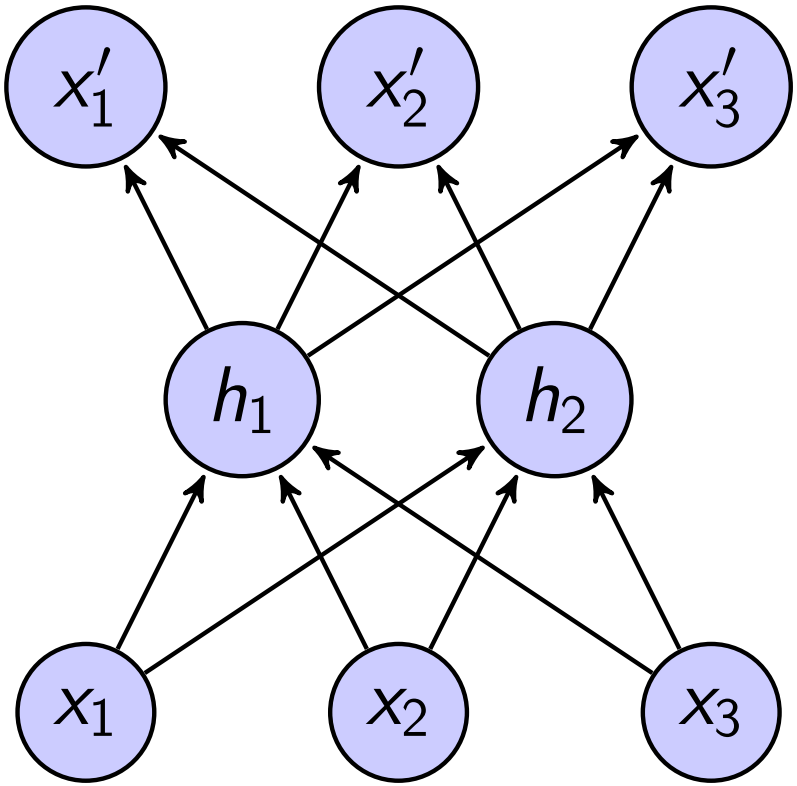
Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

- $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$
- $|h|$ is small to enforce "compression" of data

Auto-Encoders: Efficient replacement for RBM



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

Encourage h to give small reconstruction error:

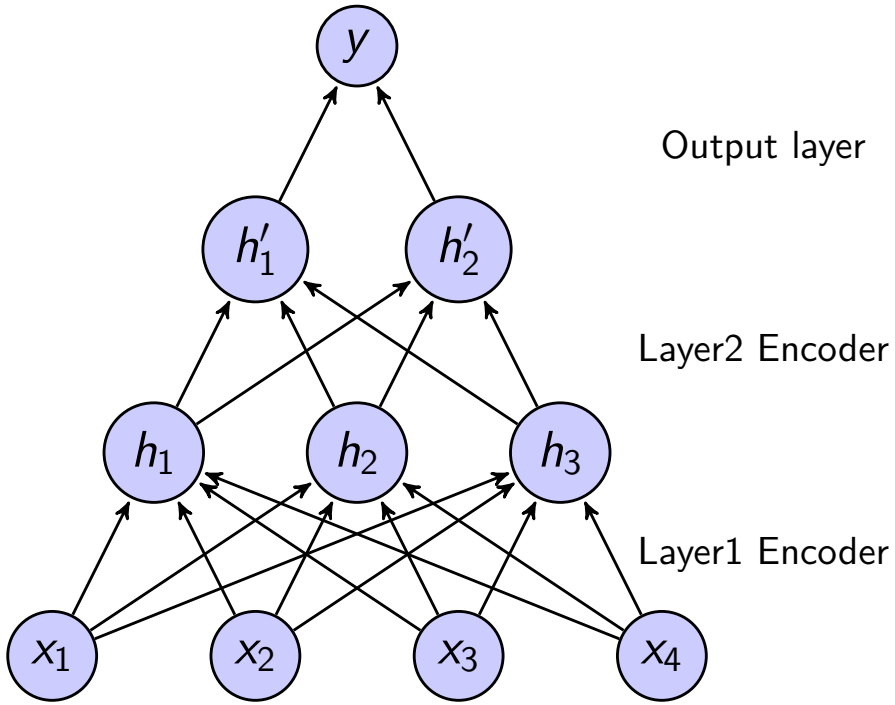
- $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$
- $|h|$ is small to enforce "compression" of data
- Back-prop for 2-layer nets, with $x^{(m)}$ as input & output

Stacked Auto-Encoders (SAE) [Bengio et al., 2006]

- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way

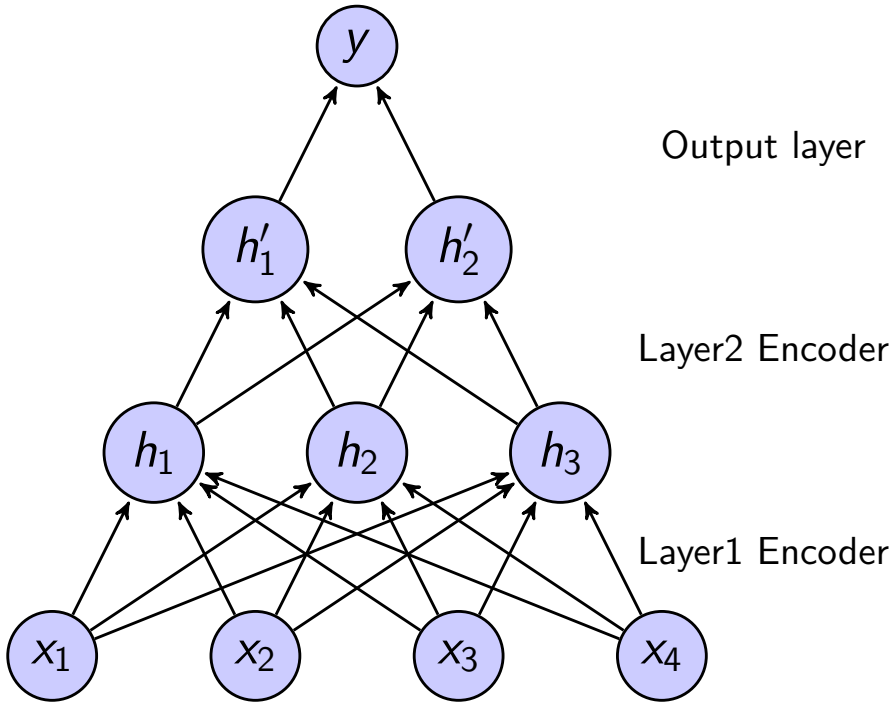
Stacked Auto-Encoders (SAE) [Bengio et al., 2006]

- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way



Stacked Auto-Encoders (SAE) [Bengio et al., 2006]

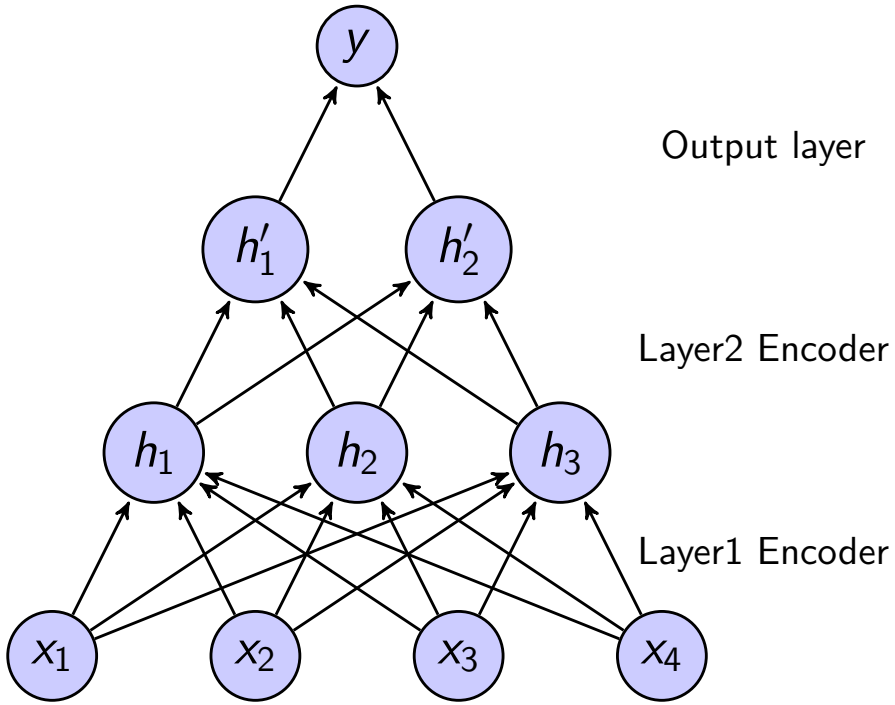
- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way



- Unlike RBMs, Auto-encoders are deterministic.
  - $h = \sigma(Wx + b)$, not $p(h = 1) = \sigma(Wx + b)$

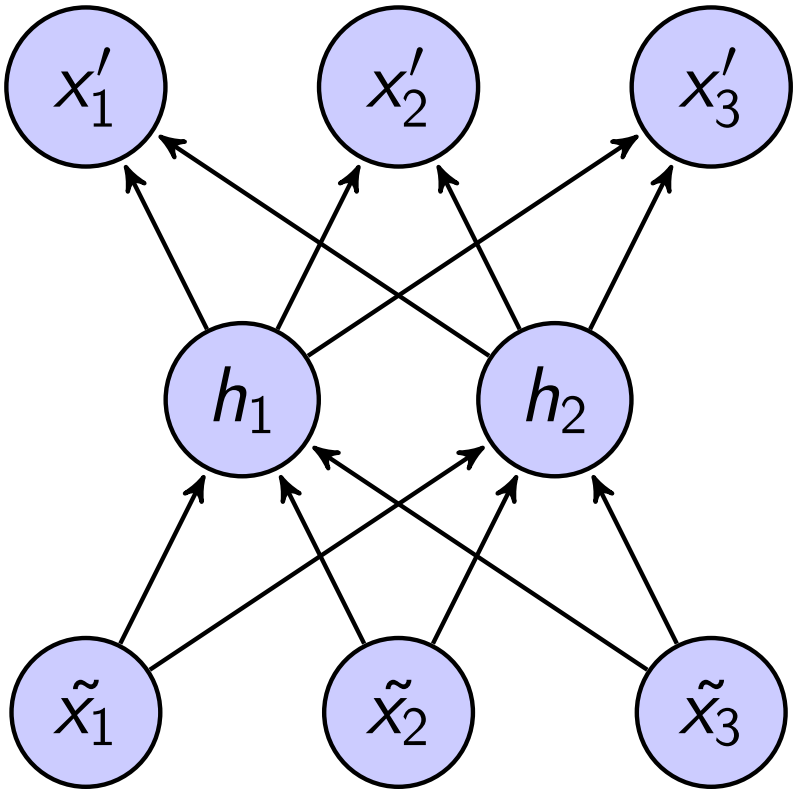
Stacked Auto-Encoders (SAE) [Bengio et al., 2006]

- The encoder/decoder gives same form $p(h|x)$, $p(x|h)$ as RBMs, so can be stacked in the same way



- Unlike RBMs, Auto-encoders are deterministic.
  - $h = \sigma(Wx + b)$, not $p(h = 1) = \sigma(Wx + b)$
  - Disadvantage: Can't form deep generative model
  - Advantage: Fast to train, useful still for DNN init
- Note similarities to RBM contrastive divergence

Variants: e.g. Denoising Auto-Encoder



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(W\tilde{x} + b)$

$\tilde{x} = x +$ noise

1. Perturb input data x to $\tilde{x}$ using invariance from domain knowledge.
2. Train weights to reduce reconstruction error with respect to original input: $||x - x'||$

Denoising Auto-Encoders

- Example: Randomly shift, rotate, and scale input image
- An image of "2" is a "2" no matter how you add noise, so the auto-encoder will try to cancel the variations that are not important.

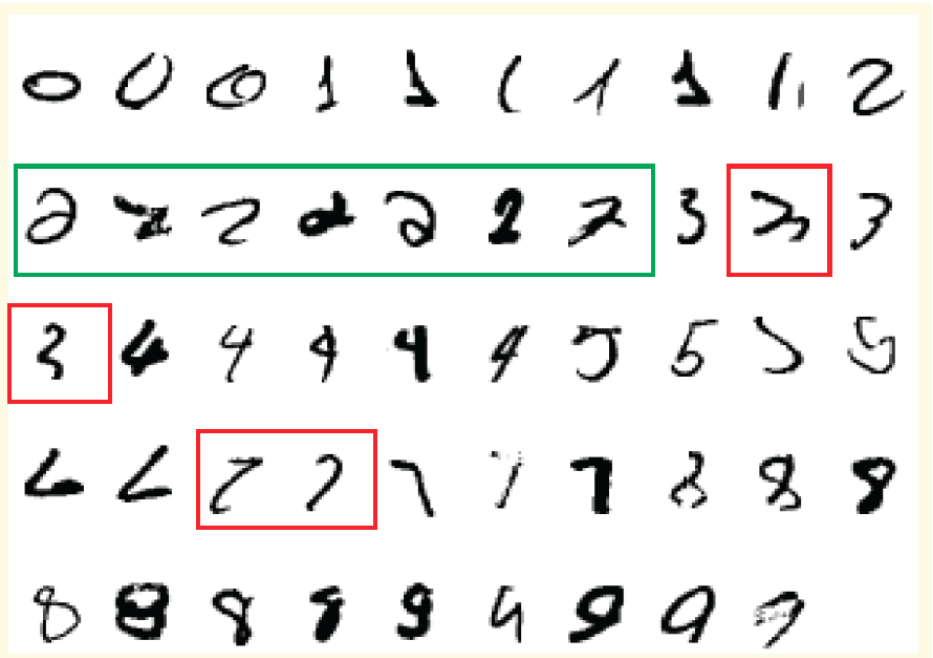


Figure from Geoff Hinton's 2012 Coursera course, lecture 1: https://www.coursera.org/course/neuralnets

Stacked Auto-Encoders (SAE): Summary

1. Auto-Encoders are cheaper alternatives to RBMs.
   - Not probabilistic, but fast to train using Backpropagation
   - Achieves similar accuracies as RBM [Bengio et al., 2006]

Stacked Auto-Encoders (SAE): Summary

1. Auto-Encoders are cheaper alternatives to RBMs.
   - Not probabilistic, but fast to train using Backpropagation
   - Achieves similar accuracies as RBM [Bengio et al., 2006]
2. Auto-Encoders learn to "compress" and "re-construct" input data. Again, the focus is on modeling $p(x)$ first.

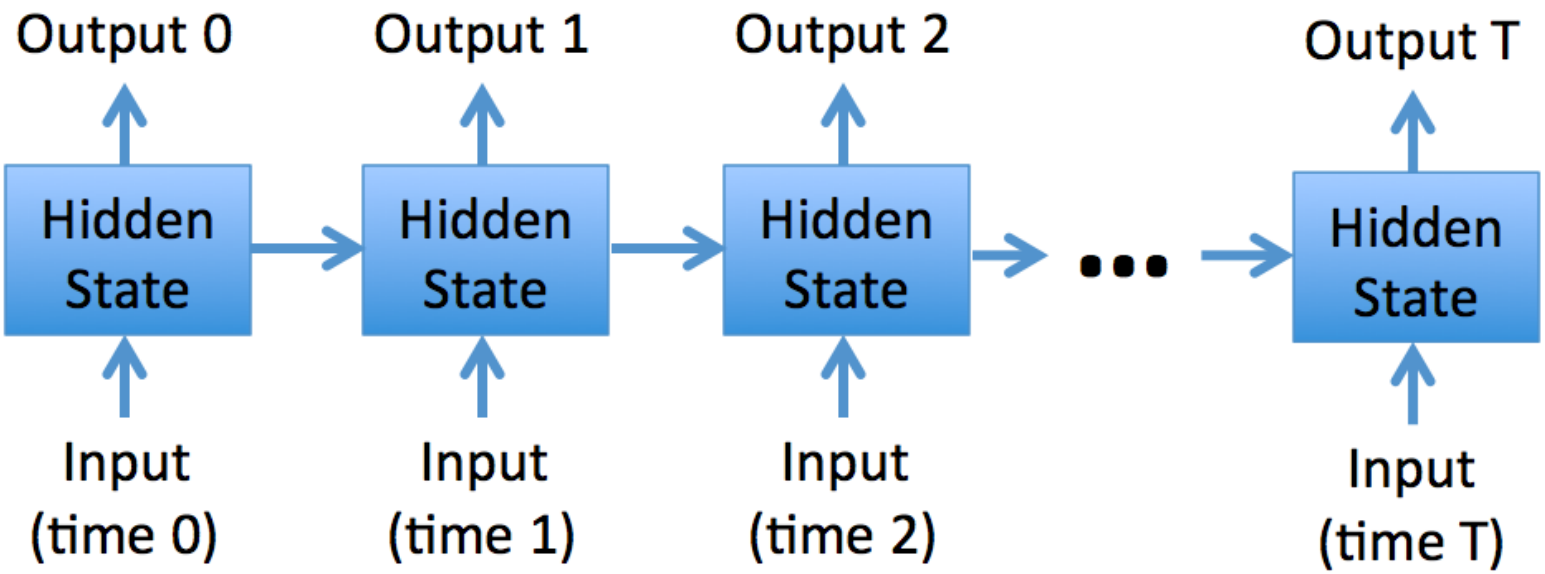
Stacked Auto-Encoders (SAE): Summary

1. Auto-Encoders are cheaper alternatives to RBMs.
   - Not probabilistic, but fast to train using Backpropagation
   - Achieves similar accuracies as RBM [Bengio et al., 2006]
2. Auto-Encoders learn to "compress" and "re-construct" input data. Again, the focus is on modeling $p(x)$ first.
3. Many variants, some provide ways to incorporate domain knowledge.

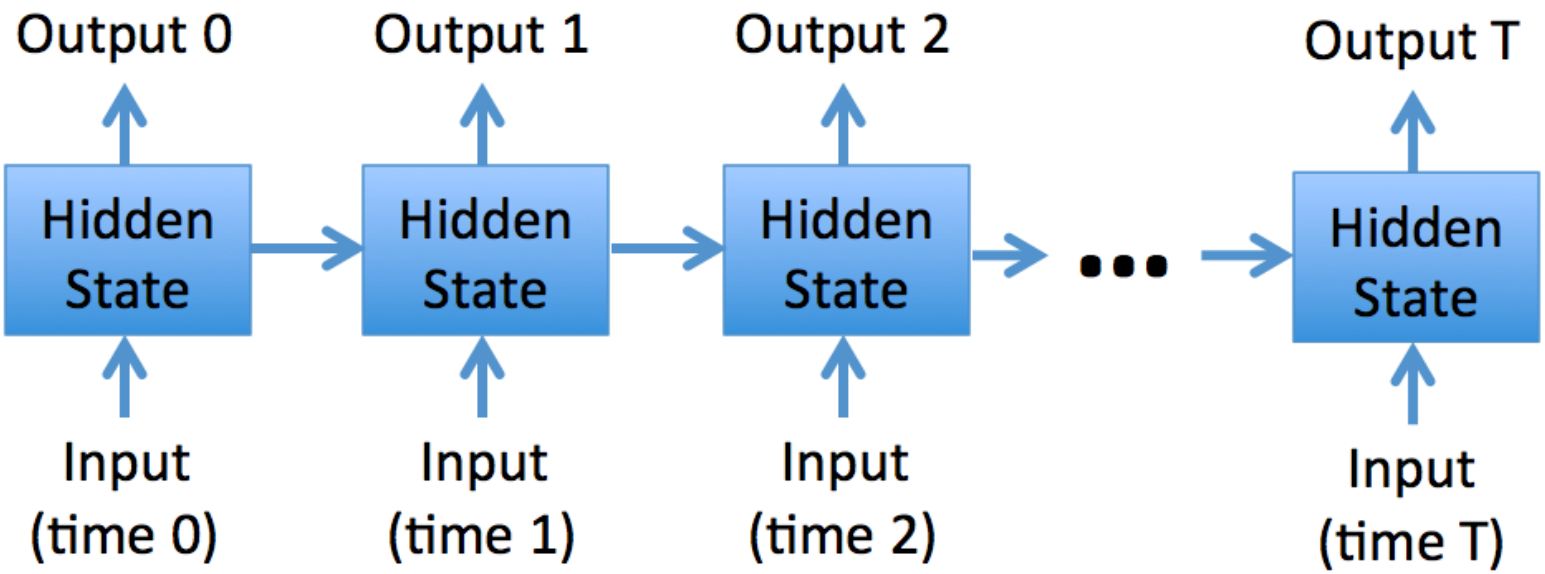
Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Modeling sequence data



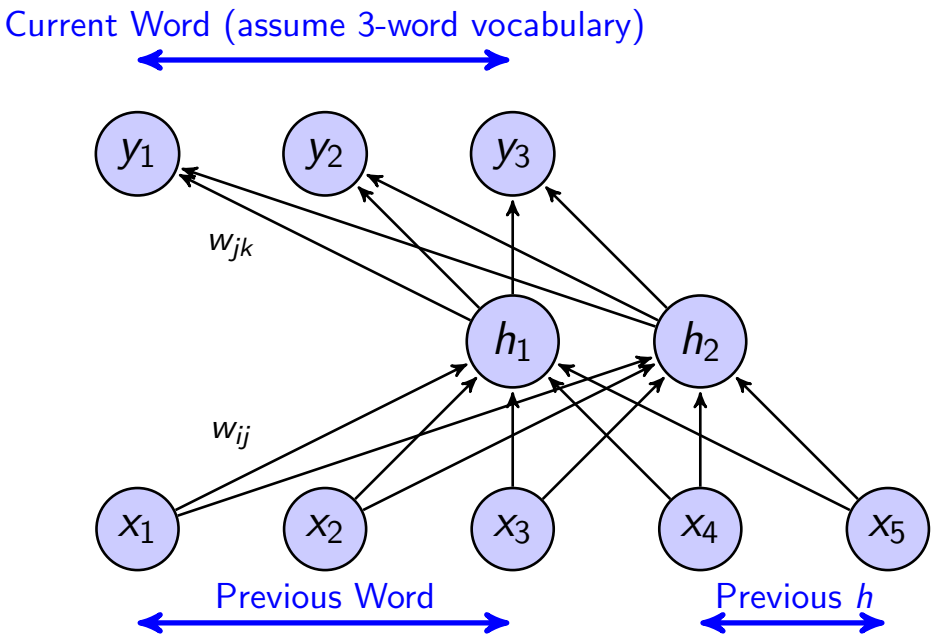
Modeling sequence data



- Example applications:
  - Time-series prediction
  - Sequence-to-sequence transduction
  - Language modeling $P(current\_word \mid previous\_words)$

Recurrent Neural Net Language Models

Model $p(\textit{current\_word}|\textit{previous\_words})$ with a recurrent hidden layer [Mikolov et al., 2010]

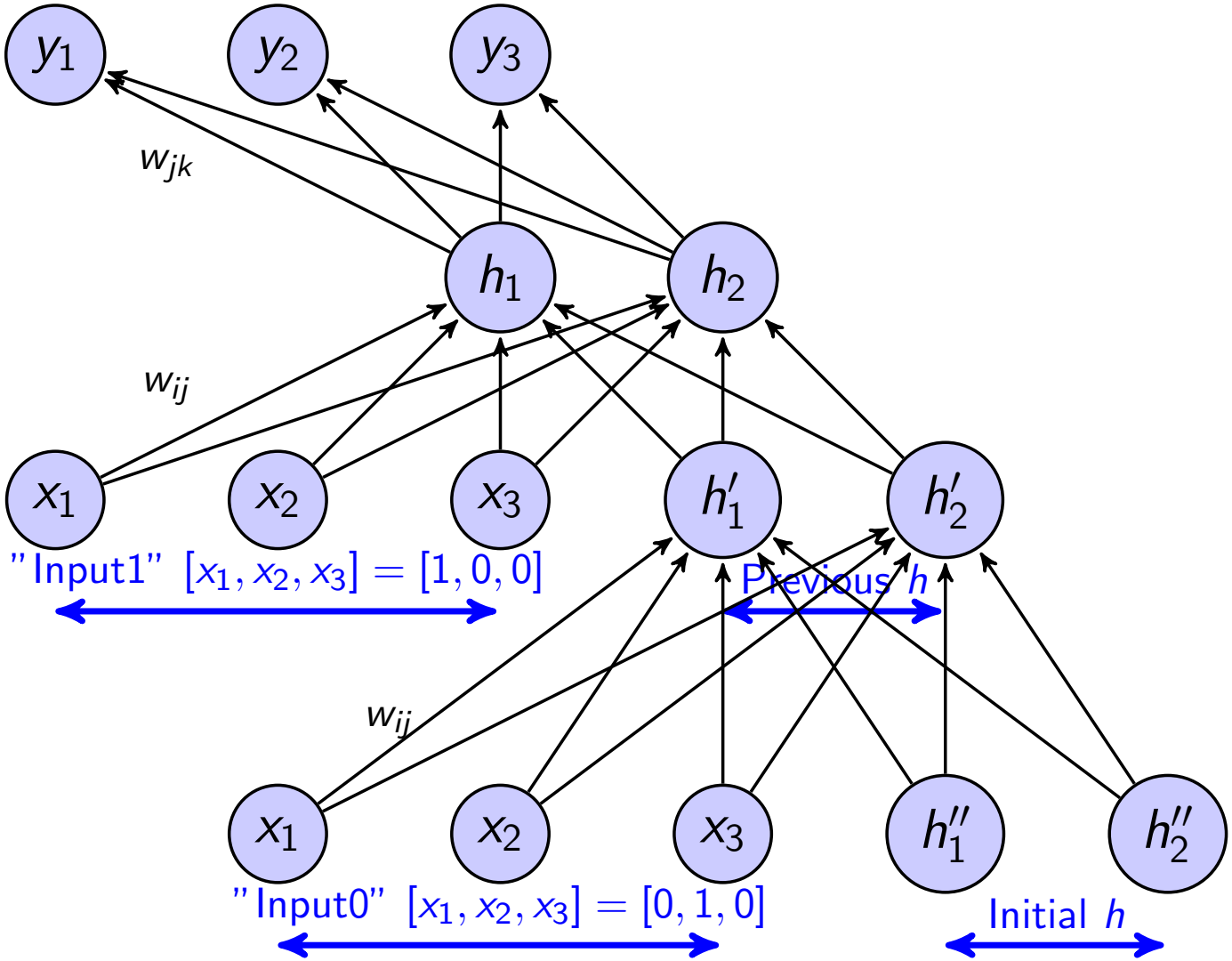


- Probability of word: $y_k = \frac{\exp(W_{jk}^T h)}{\sum_{k'} \exp(W_{jk'}^T h)}$
- $[x_4, x_5]$ is a copy of $[h_1, h_2]$ from the previous time-step
- $h_j = \sigma(W_{ij}^T x_i)$ is hidden state of partial sentence
- Arbitrarily-long history is (theoretically) kept through recurrence

Training: Backpropagation through Time

Unroll the hidden states for a few time-steps, then backprop.

Very deep network: vanishing/exploding gradients!



A popular recurrent unit: LSTM

Long Short-term Memory (LSTM):

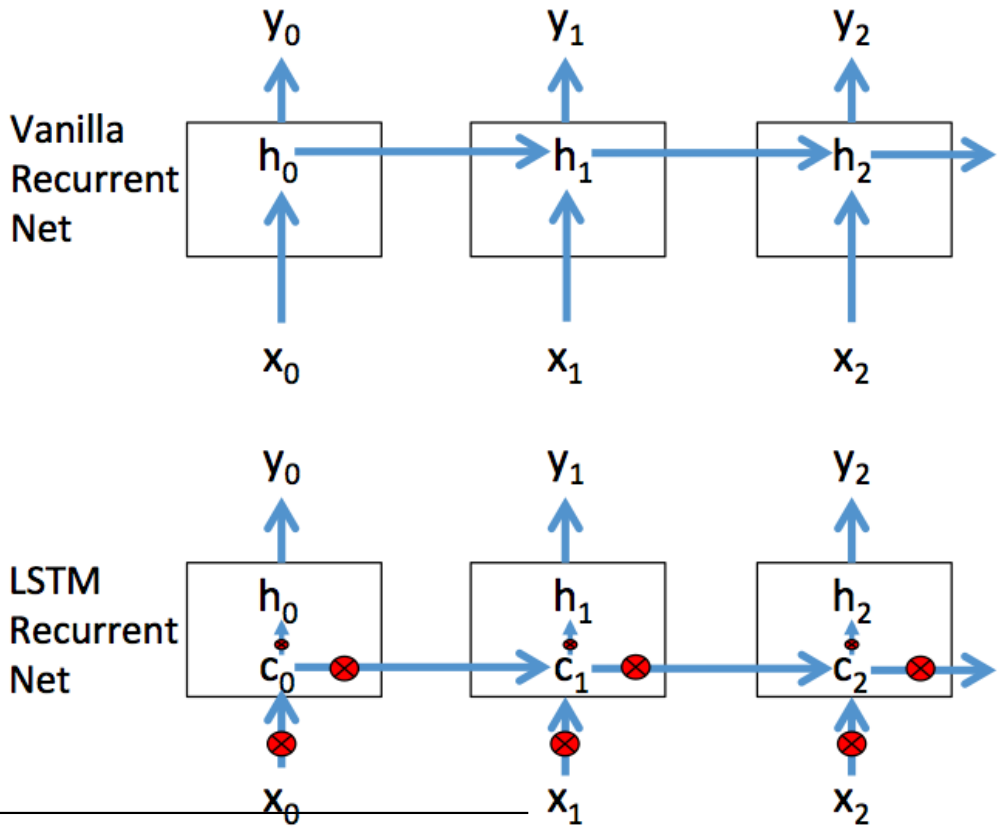
- Idea: Shouldn't need to back-prop to beginning of history. Just keep a memory of important bits.

*Nice explanation of LSTM and variants here:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

A popular recurrent unit: LSTM

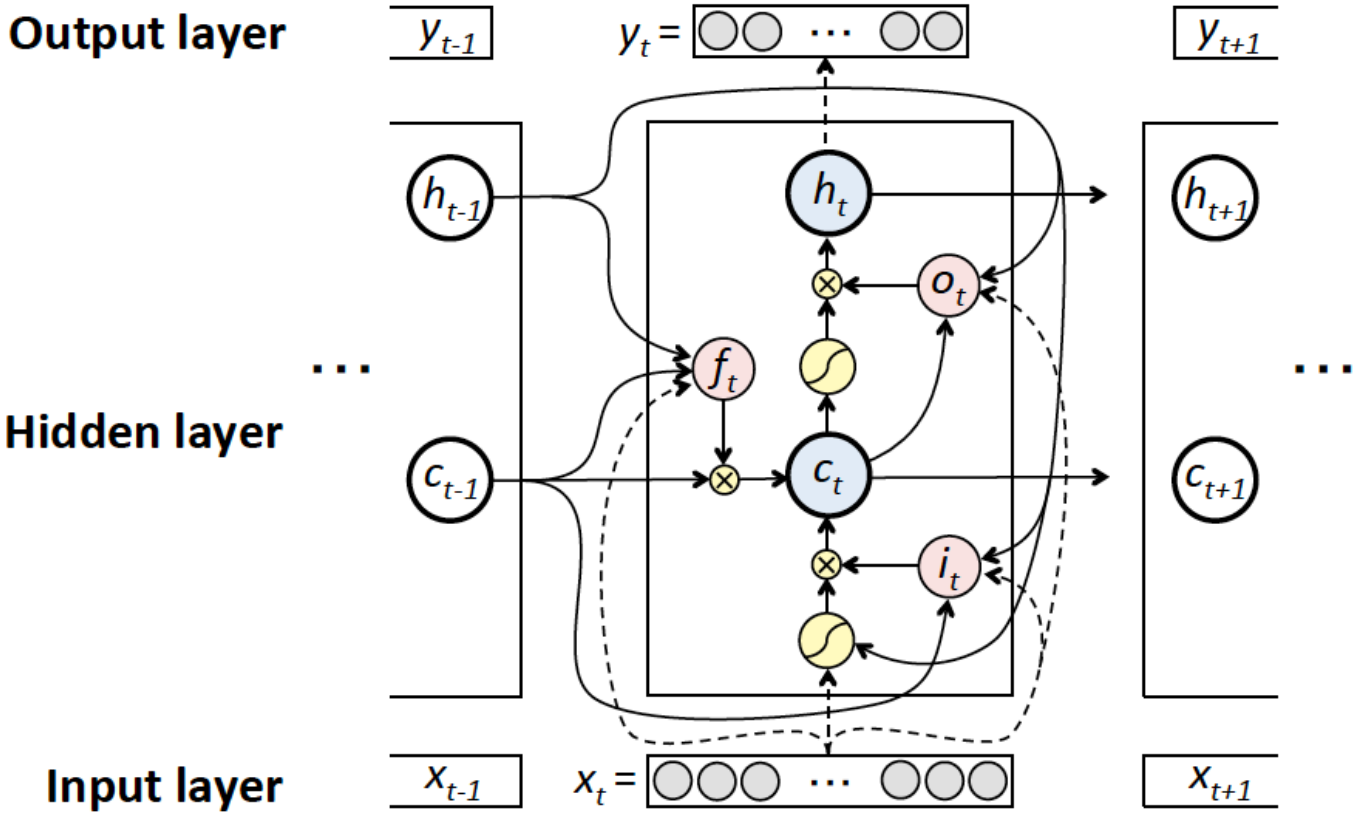
Long Short-term Memory (LSTM):

- Idea: Shouldn't need to back-prop to beginning of history. Just keep a memory of important bits.
- Introduces memory cell (c_t), with input/output/forget gates to keep track of what to remember and when



*Nice explanation of LSTM and variants here: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM in detail [Gers et al., 2002]



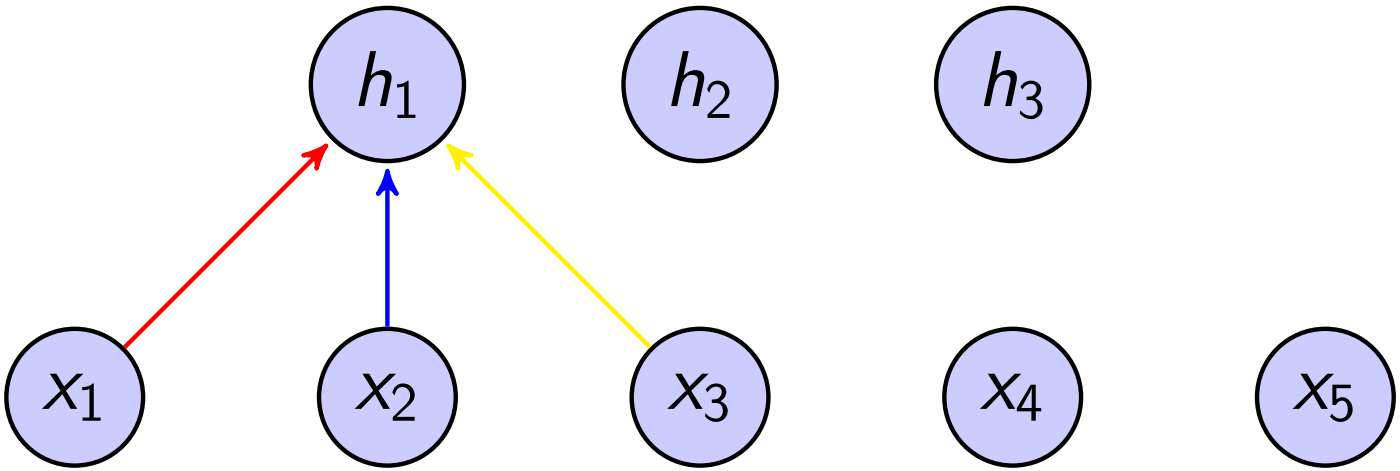
$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{ci}c_{t-1} + W_{hi}h_{t-1}) \\
f_t &= \sigma(W_{xf}x_t + W_{cf}c_{t-1} + W_{hf}h_{t-1}) \\
c_t &= f_t c_{t-1} + i_t \ tanh(W_{xc}x_t + W_{hc}h_{t-1}) \\
o_t &= \sigma(W_{xo}x_t + W_{co}c_t + W_{ho}h_{t-1}) \\
h_t &= o_t \tanh(c_t), \quad y_t = softmax(W_{hy}h_t)
\end{aligned}
$$

Figure courtesy of Adhi Kuncoro & Yuichiro Sawai

Outline

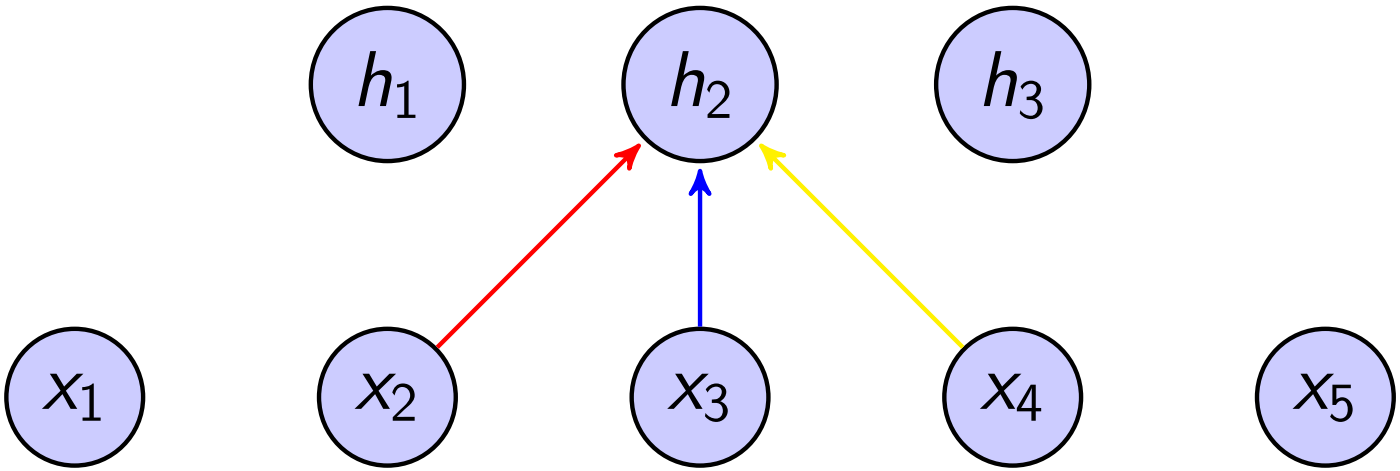
1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Convolution



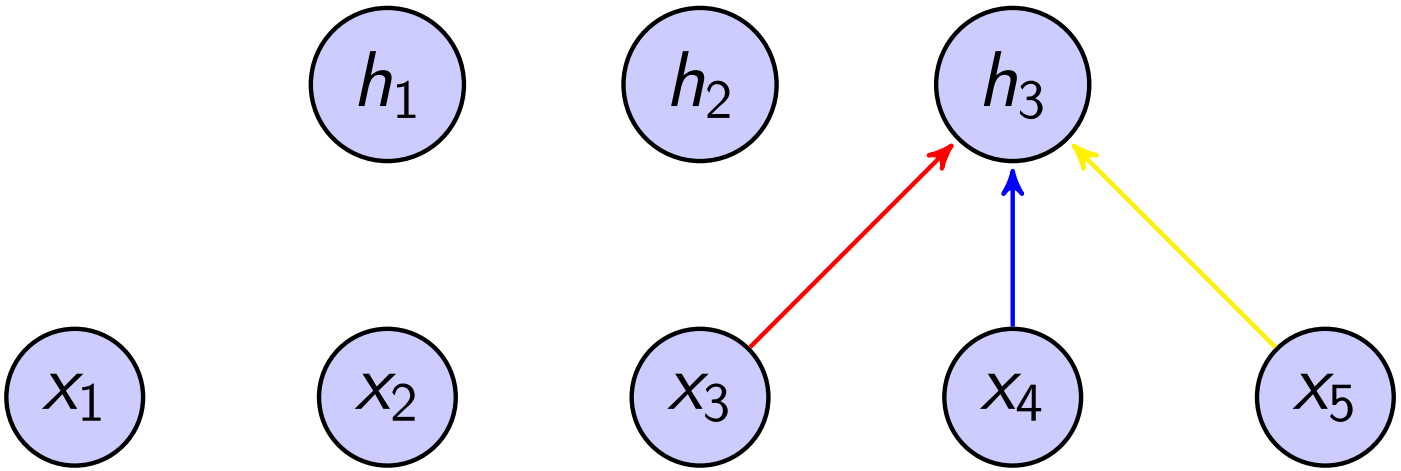
Sliding window of shared weights: $h_j = \sum_{\tau=0}^{2} w_\tau x_{j+\tau}$

Convolution



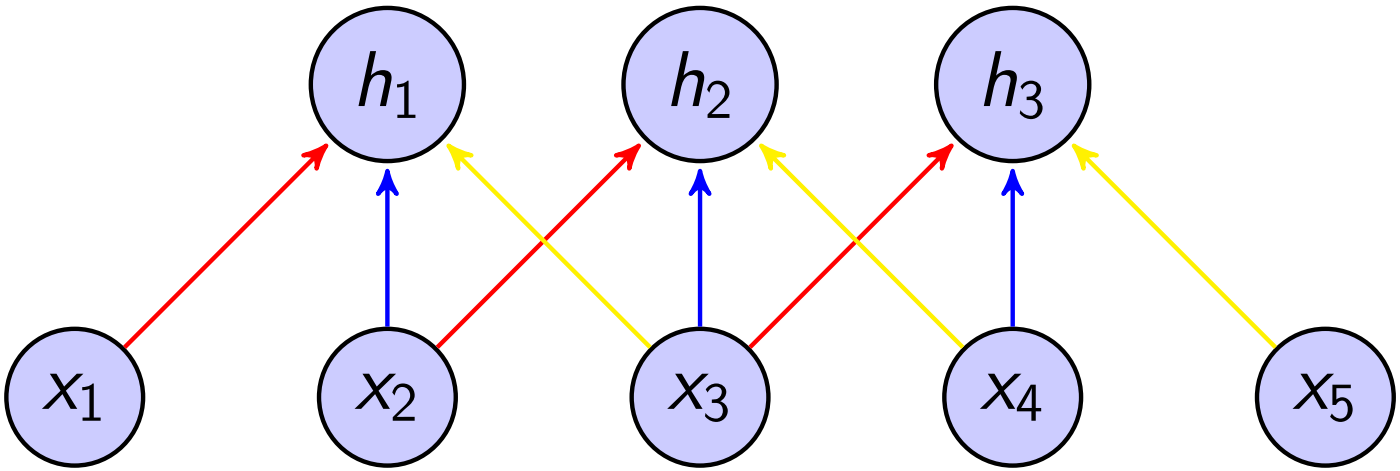
Sliding window of shared weights: $h_j = \sum_{\tau=0}^{2} w_\tau x_{j+\tau}$

Convolution



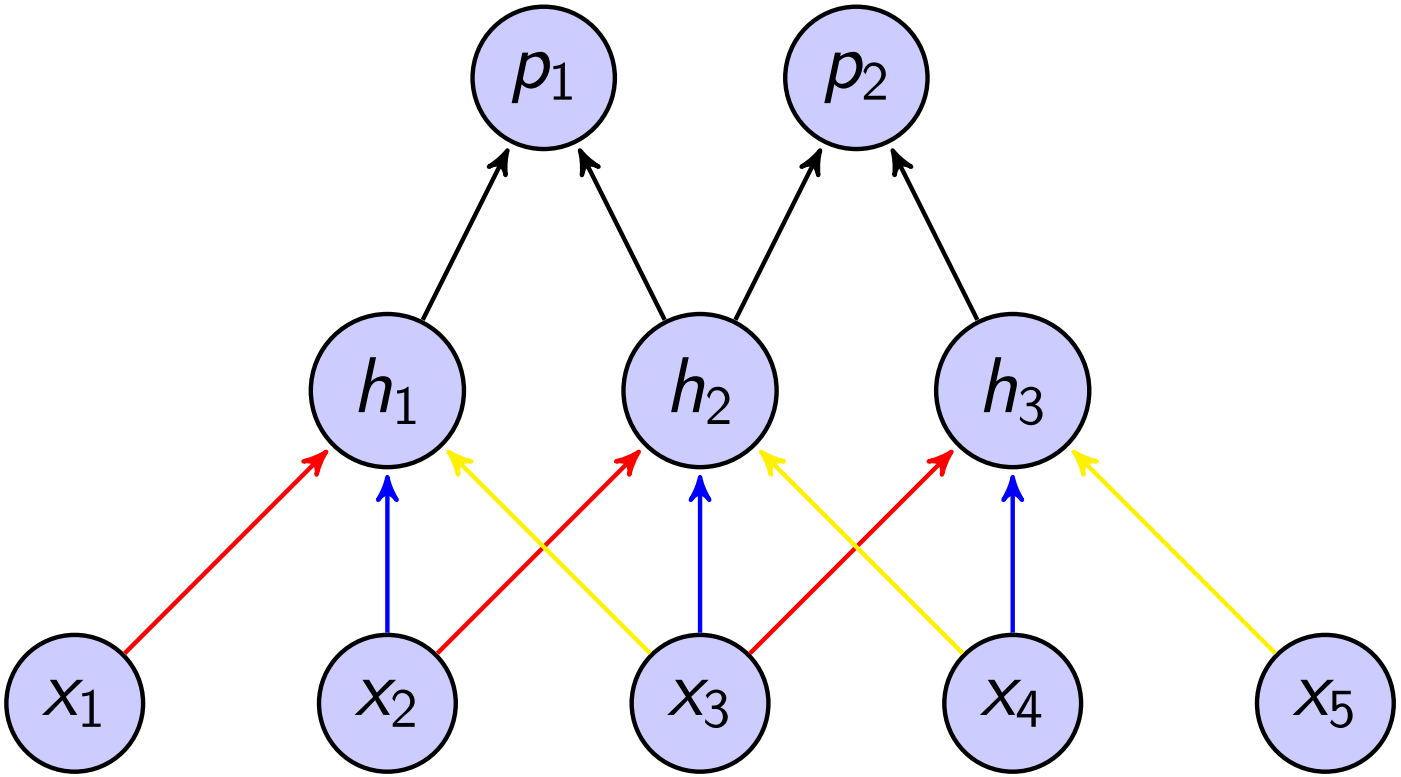
Sliding window of shared weights: $h_j = \sum_{\tau=0}^{2} w_\tau x_{j+\tau}$

Convolution



Sliding window of shared weights: $h_j = \sum_{\tau=0}^{2} w_\tau x_{j+\tau}$

Pooling

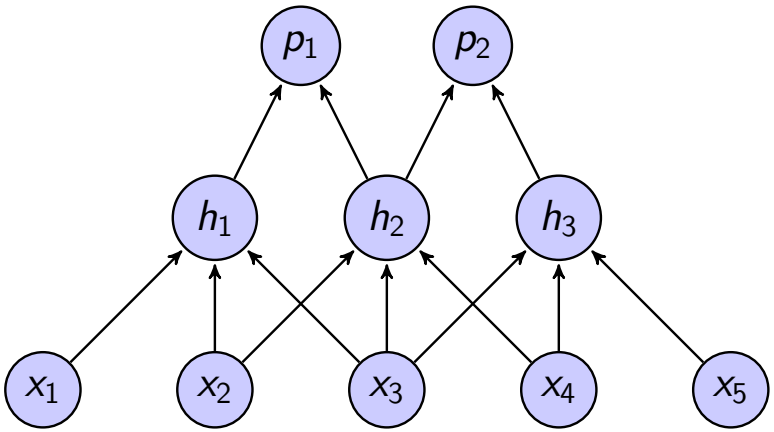


Max-Pooling nodes: $p_1 = max(h_1, h_2)$, $p_2 = max(h_2, h_3)$

Advantages of Convolution + Pooling:

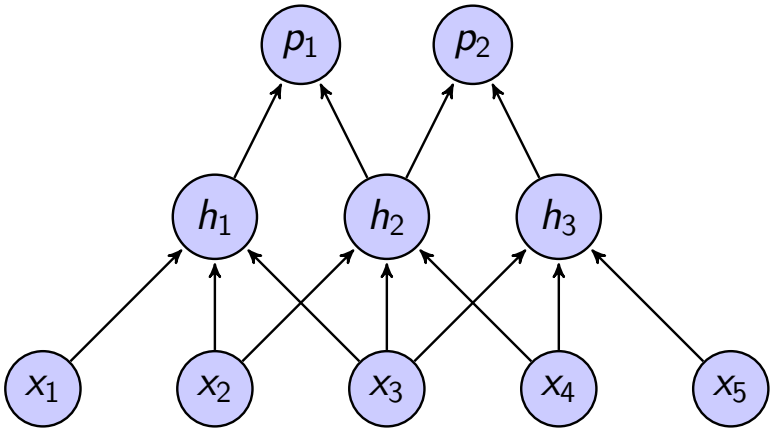
1. Fewer weights
2. Shift invariance

Convolutional Nets in vision [LeCun et al., 1998]

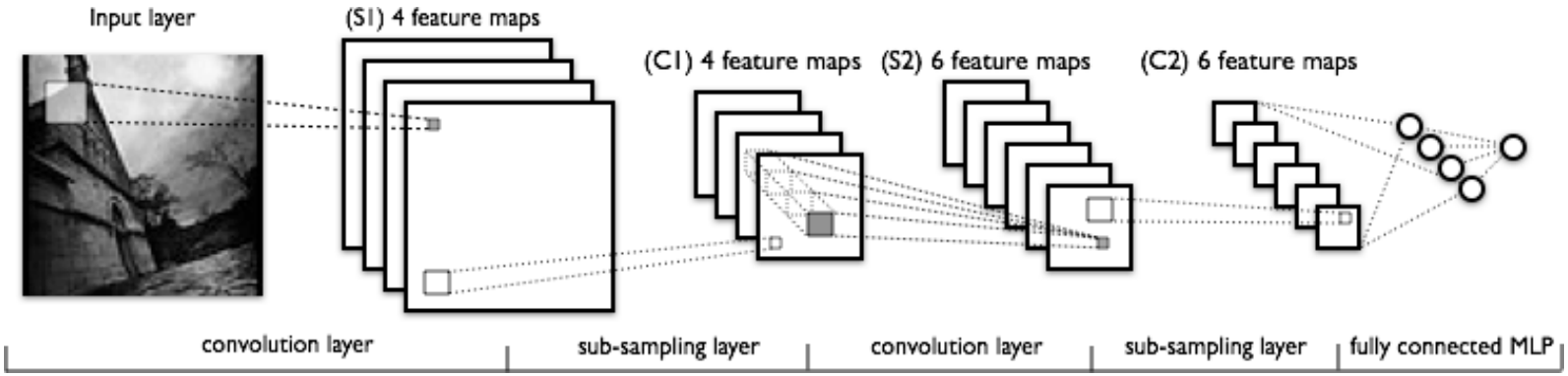


Receptive Field (RF): each h_j only connects to small input region via convolution.
Pooling: e.g.
$p_1 = max(h_1, h_2)$ or
$p_1 = \sqrt{h_1^2 + h_2^2}$

Convolutional Nets in vision [LeCun et al., 1998]



Receptive Field (RF): each h_j only connects to small input region via convolution.
Pooling: e.g.
$p_1 = max(h_1, h_2)$ or
$p_1 = \sqrt{h_1^2 + h_2^2}$



(Figure from `http://deeplearning.net/tutorial/lenet.html`)

Section Summary

- RBM
- Auto-Encoders
- Recurrent Units
- Convolution

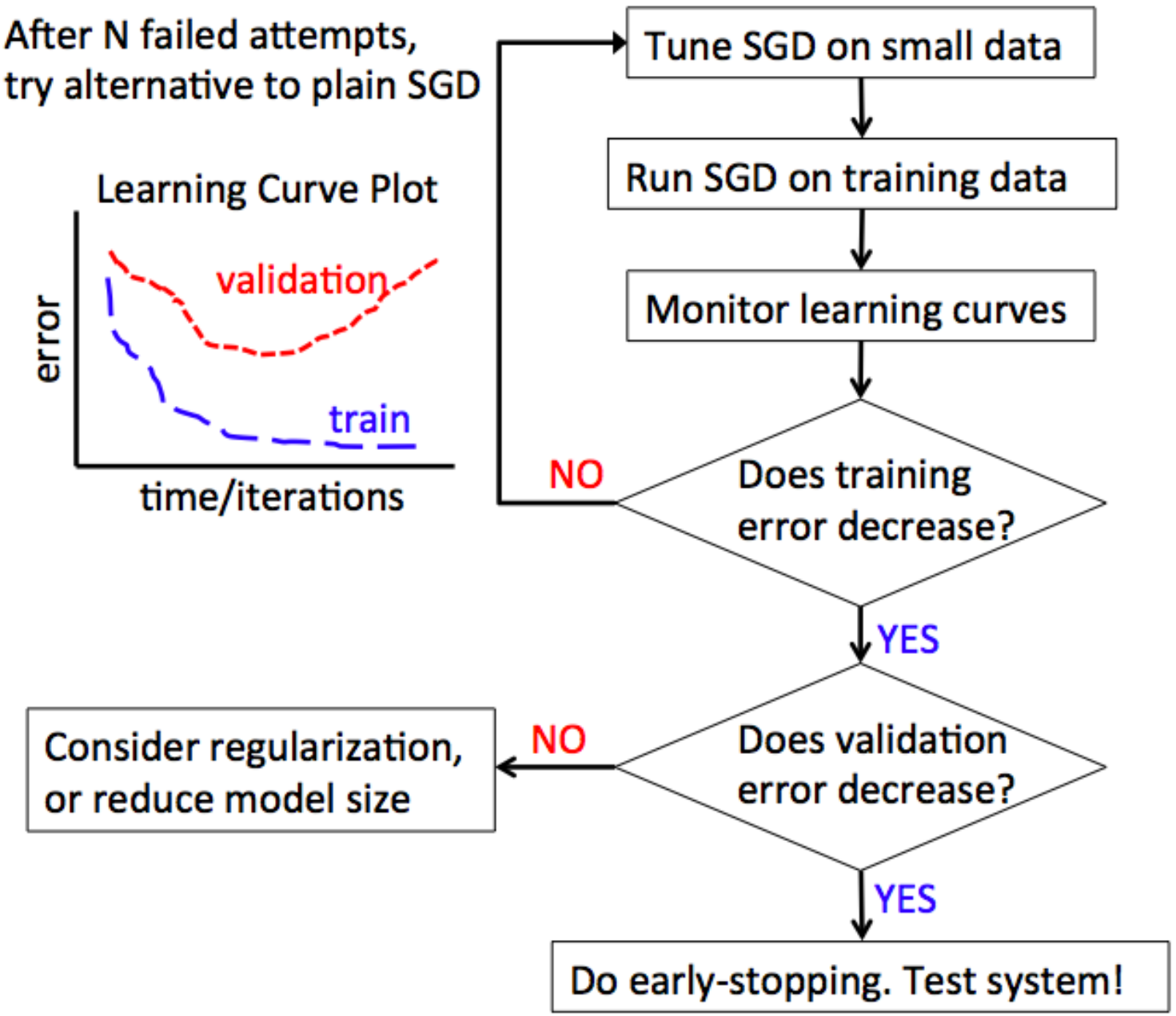
Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Neural Net Training Recipe



Making plain SGD Work

- With some tuning, SGD often works just as well as more advanced optimization methods!

Making plain SGD Work

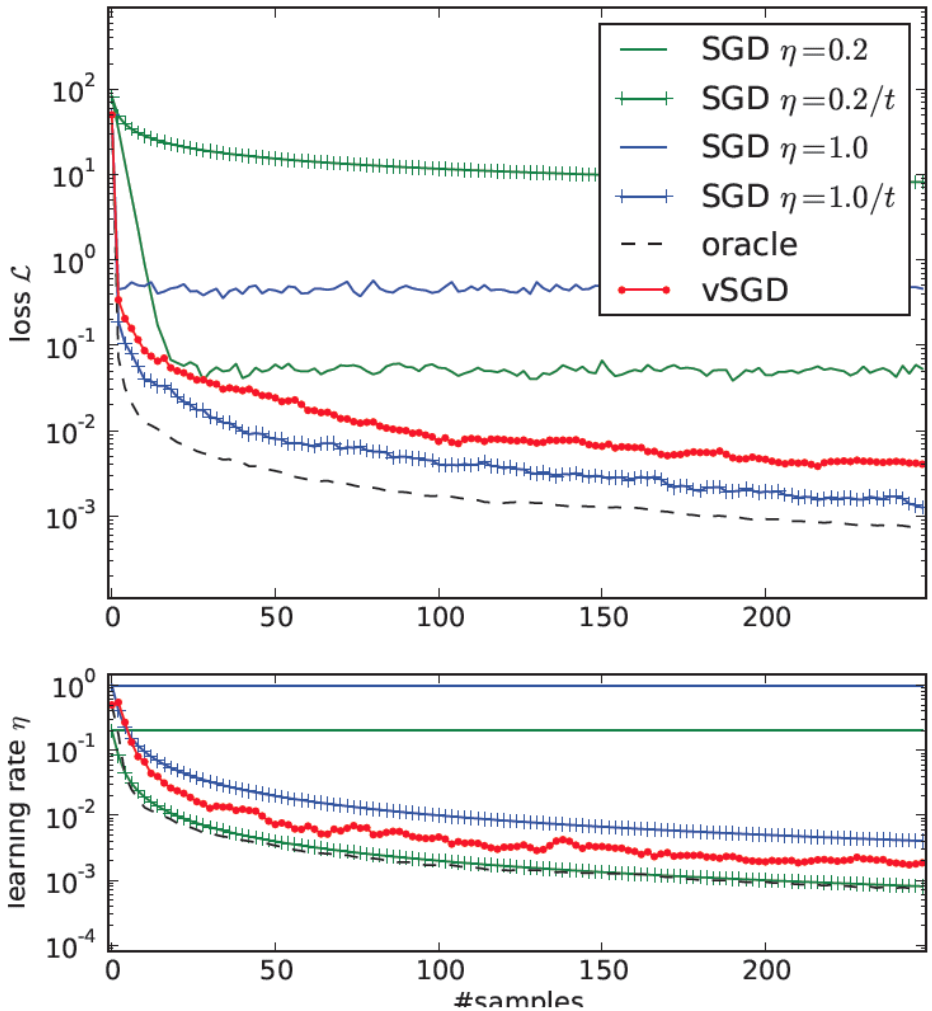
- With some tuning, SGD often works just as well as more advanced optimization methods!
- Important hyper-parameters:
  - Learning rate: tune on small data first
  - Mini-batch size: fit to computer's memory

Making plain SGD Work

- With some tuning, SGD often works just as well as more advanced optimization methods!
- Important hyper-parameters:
  - Learning rate: tune on small data first
  - Mini-batch size: fit to computer's memory
- General tips:
  - Shuffle training data
  - Scale training data within suitable range
  - Randomly initialize weights, e.g. uniform$[-1/\sqrt{(FanIn)}, 1/\sqrt{(FanIn)}]$

Effect of Learning Rate γ on SGD Convergence

- Update: $w \leftarrow w - \gamma(\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)})$
  - γ: try to be as large as possible without divergence.
  - Common heuristic: $\gamma_t = \frac{\gamma_0}{1+\gamma_0 * \lambda * t} = O(1/t)$
- Analysis by [Schaul et al., 2013] (in plot, $\eta \equiv \gamma$):



Difficulty of optimizing highly non-convex loss functions

- "Pathological curvature" is tough to navigate for SGD
- Many methods for avoiding this zig-zag problem.

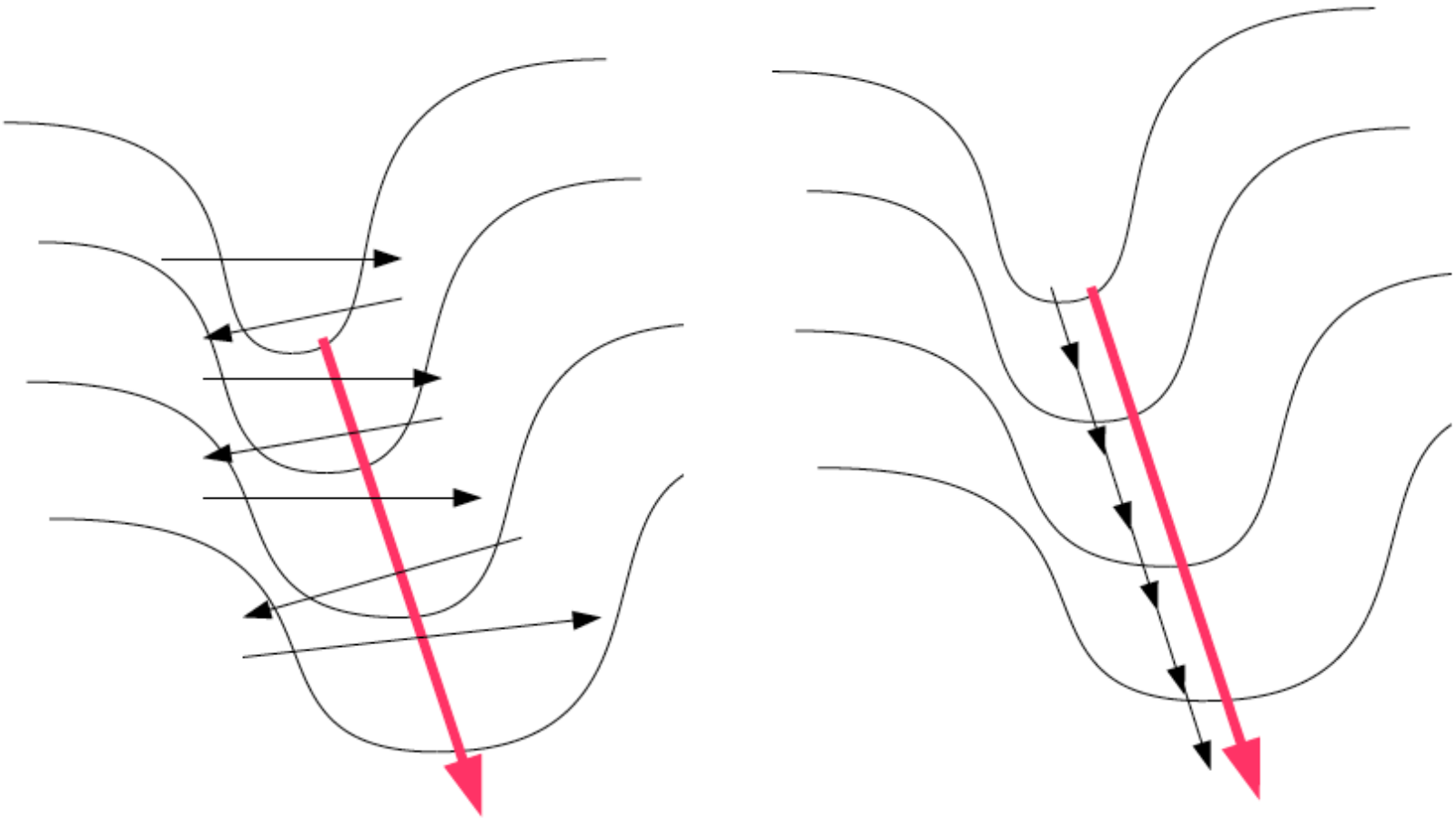


Figure from [Martens, 2010]

SGD + Momentum

- Idea: Accelerate in the direction that is consistently pointed to by successive gradients

Note slight change of notation in this section: w is a vector of weights, subscript t in w_t refers to weight at time t, not index as in previous sections

SGD + Momentum

- Idea: Accelerate in the direction that is consistently pointed to by successive gradients
- Let update equation be: $w_{t+1} = w_t + \Delta w_t$
  - SGD: $\Delta w_t = -\gamma g_t$, where g_t is gradient

Note slight change of notation in this section: w is a vector of weights, subscript t in w_t refers to weight at time t, not index as in previous sections

SGD + Momentum

- Idea: Accelerate in the direction that is consistently pointed to by successive gradients
- Let update equation be: $w_{t+1} = w_t + \Delta w_t$
  - SGD: $\Delta w_t = -\gamma g_t$, where g_t is gradient
  - SGD+Momemtum: $\Delta w_t = \rho \Delta w_{t-1} - \gamma g_t$

Note slight change of notation in this section: w is a vector of weights, subscript t in w_t refers to weight at time t, not index as in previous sections

SGD + Momentum

- Idea: Accelerate in the direction that is consistently pointed to by successive gradients
- Let update equation be: $w_{t+1} = w_t + \Delta w_t$
  - SGD: $\Delta w_t = -\gamma g_t$, where g_t is gradient
  - SGD+Momemtum: $\Delta w_t = \rho \Delta w_{t-1} - \gamma g_t$
- ρ: a constant that determines how much momentum from previous updates

---

Note slight change of notation in this section: w is a vector of weights, subscript t in w_t refers to weight at time t, not index as in previous sections

SGD + Momentum

- Idea: Accelerate in the direction that is consistently pointed to by successive gradients
- Let update equation be: $w_{t+1} = w_t + \Delta w_t$
  - SGD: $\Delta w_t = -\gamma g_t$, where g_t is gradient
  - SGD+Momemtum: $\Delta w_t = \rho \Delta w_{t-1} - \gamma g_t$
- ρ: a constant that determines how much momentum from previous updates
- Zig-zag directions will be cancelled out, while others will be accelerated.

Note slight change of notation in this section: w is a vector of weights, subscript t in w_t refers to weight at time t, not index as in previous sections

Adaptive Learning Rate: ADAGRAD [Duchi et al., 2011]

- Idea:
  - Each individual weight has it's own dynamic learning rate.
  - Weights with large derivative have smaller learning rate, weights with small derivative have large learning rate $\rightarrow$ different scales are balanced
- Let update equation be: $w_{t+1} = w_t + \Delta w_t$
  - SGD: $\Delta w_t = -\gamma g_t$, where g_t is gradient
  - ADAGRAD: $\Delta w_t = -\frac{\gamma}{diag(\sqrt{\sum_{\tau=1}^{t} g_\tau g_\tau^T})} g_t$
- Denominator accumulates the ℓ_2 norm of past gradients per dimension.

Adaptive Learning Rate: ADADELTA [Zeiler, 2012]

- Idea: Similar to ADAGRAD, but no learning rate.
  - ADAGRAD: $\Delta w_t = -\frac{\gamma}{diag(\sqrt{\sum_{\tau=1}^{t} g_\tau g_\tau^T})} g_t$
- ADADELTA:
  1. Accumulate gradient over sliding window:
     $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t g_t^T$
  2. $\Delta w_t = -\frac{RMS(E[\Delta w]_{t-1})}{RMS(E[g^2]_t)} g_t$

     where $RMS(E[g^2]_t) = \sqrt{E[g^2]_t + \epsilon}$
  3. Accumulate update over sliding window:
     $E[\Delta w^2]_t = \rho E[\Delta w^2]_{t-1} + (1-\rho) \Delta w_t \Delta w_t^T$

2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

$$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

where H is the Hessian (curvature matrix) at w

2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

$$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

where H is the Hessian (curvature matrix) at w

- Minimizing this gives the search direction: $z^* = -H^{-1} \nabla L(w)$
  - Intuitively, H^{-1} fixes any pathological curvature for $\nabla L(w)$
  - In practice, don't want to store nor invert H

2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

$$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

where H is the Hessian (curvature matrix) at w

- Minimizing this gives the search direction: $z^* = -H^{-1} \nabla L(w)$
  - Intuitively, H^{-1} fixes any pathological curvature for $\nabla L(w)$
  - In practice, don't want to store nor invert H
- Quasi-Newton methods
  - L-BFGS: uses low-rank approximation of H^{-1}

2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

$$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

where H is the Hessian (curvature matrix) at w

- Minimizing this gives the search direction: $z^* = -H^{-1} \nabla L(w)$
  - Intuitively, H^{-1} fixes any pathological curvature for $\nabla L(w)$
  - In practice, don't want to store nor invert H
- Quasi-Newton methods
  - L-BFGS: uses low-rank approximation of H^{-1}
  - Hessian-free (i.e. truncated Newton): (1) minimize $q_w(z)$ with conjugate gradient method; (2) computes Hz directly using finite-difference: $Hz = \lim_{\epsilon \to 0} \frac{\nabla L(w+\epsilon z) - \nabla L(w)}{\epsilon}$

Hessian-free optimization results

- Experiments in [Martens, 2010]

  (Random initialization + 2nd-order Hessian-free optimizer)

  gives lower training error than

  (Pre-training initialization + 1-order optimizer).

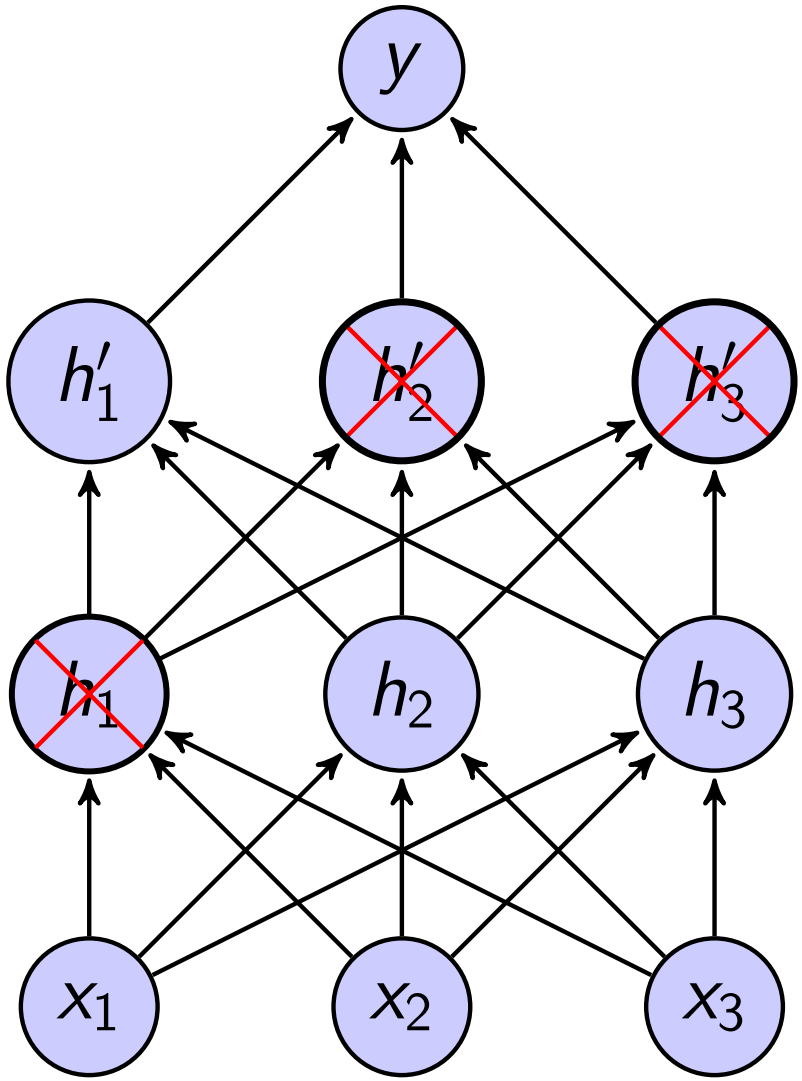
- Nice results in Recurrent Nets too
  [Martens and Sutskever, 2011]

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

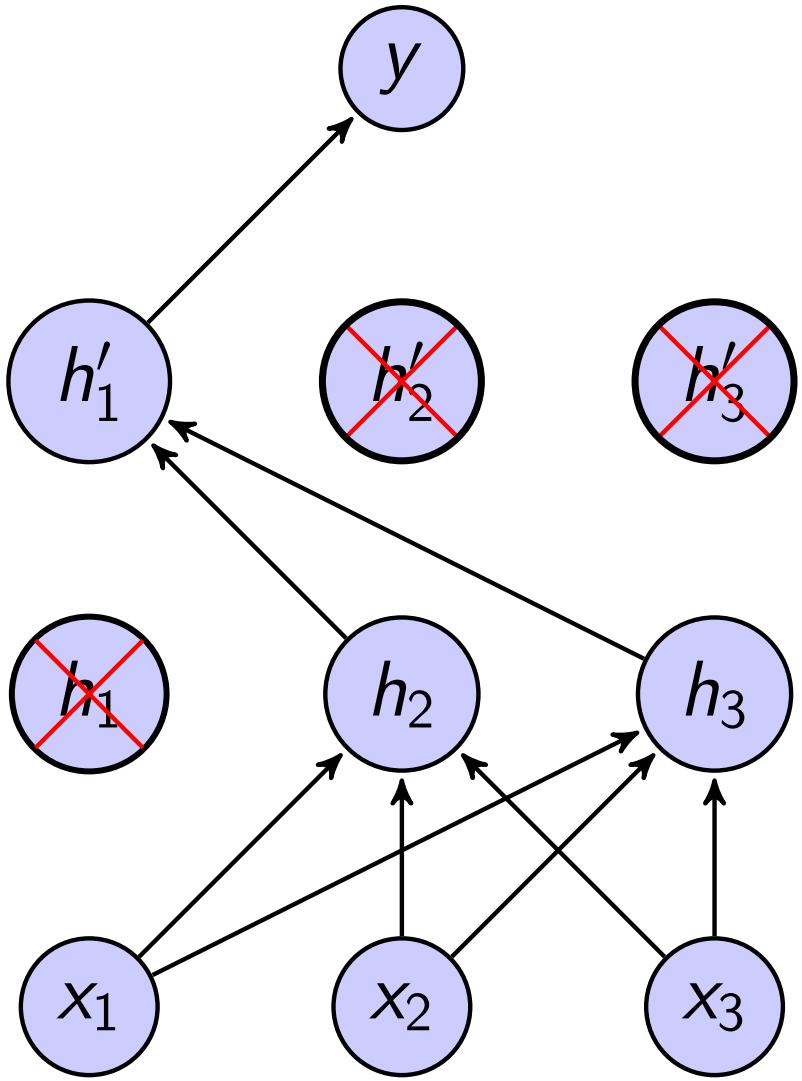
Dropout [Hinton et al., 2012]

- Each time we present $x^{(m)}$, randomly delete each hidden node with 0.5 probability
- This is like sampling from $2^{|h|}$ different architectures
- At test time, use all nodes but halve the weights
- Effect: Reduce overfitting by preventing "co-adaptation"; ensemble model averaging

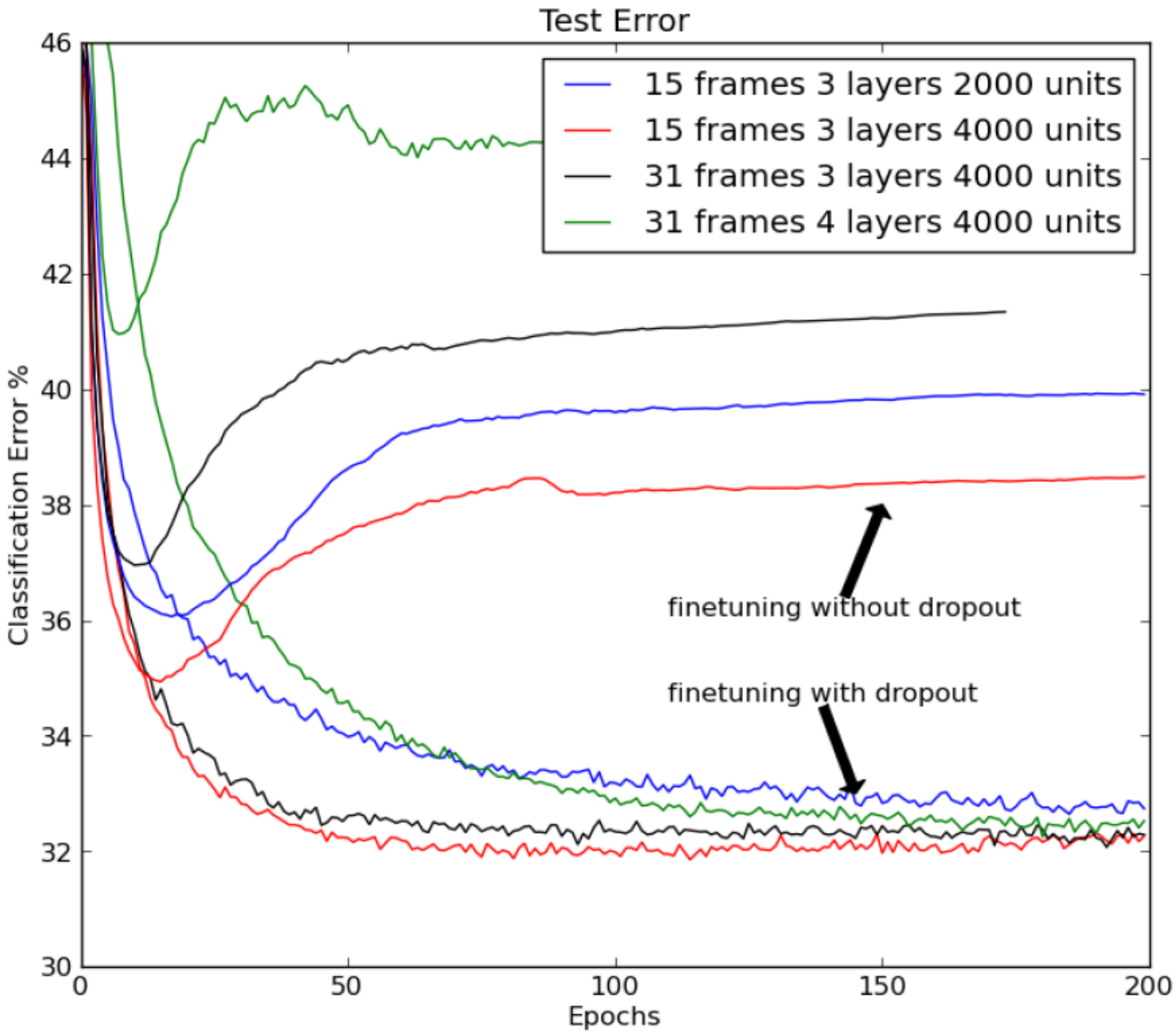


Dropout [Hinton et al., 2012]

- Each time we present $x^{(m)}$, randomly delete each hidden node with 0.5 probability
- This is like sampling from $2^{|h|}$ different architectures
- At test time, use all nodes but halve the weights
- Effect: Reduce overfitting by preventing "co-adaptation"; ensemble model averaging

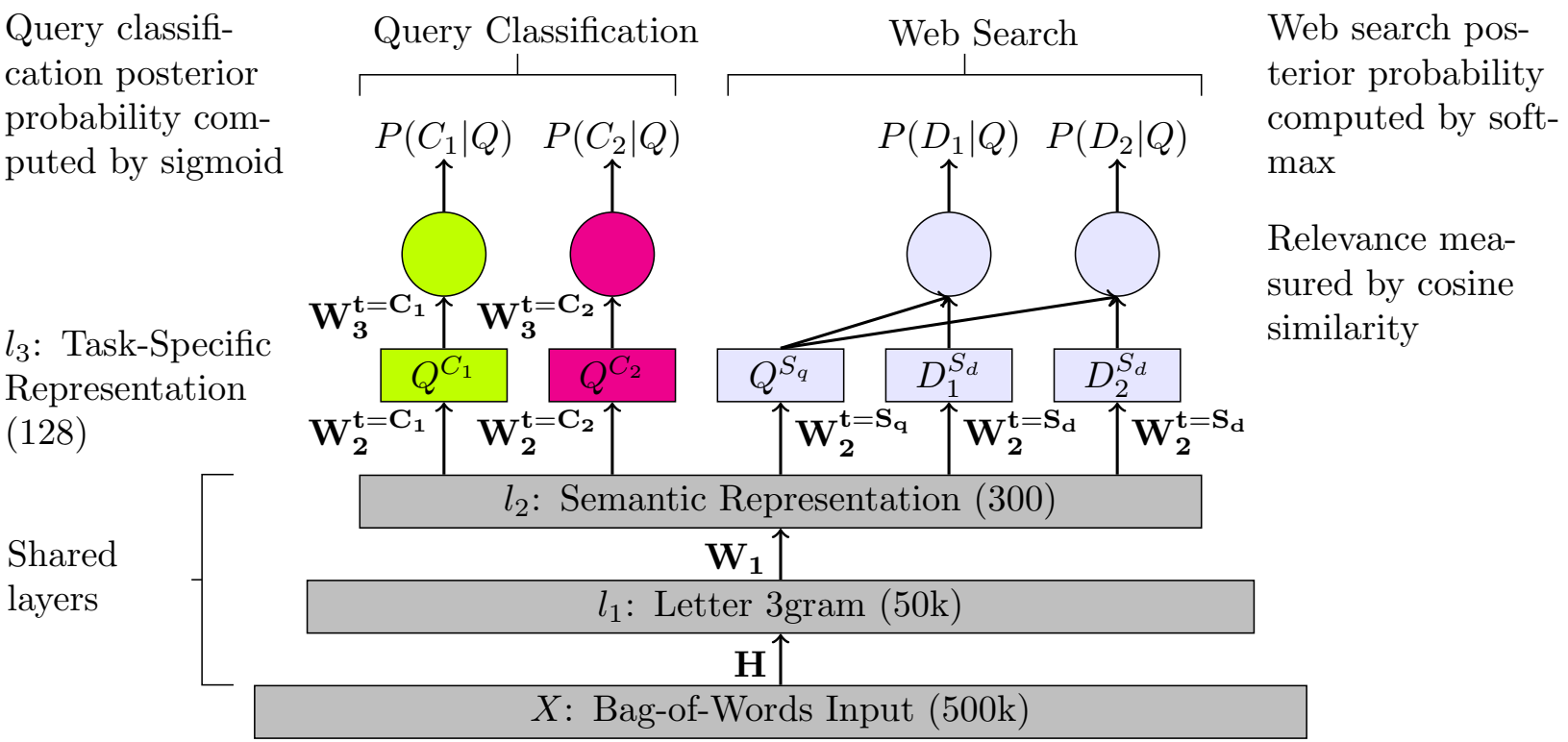


Some Results: TIMIT phone recognition



Multi-task Learning: sharing hidden layers

- If too data is insufficient for desired task:
  1. train jointly with related tasks with shared hidden layer
  2. use related tasks' hidden layer as feature representation
- e.g. [Liu et al., 2015]



See also [Caruana, 1997, Collobert et al., 2011, Bordes et al., 2012]

Multi-task learning results in [Liu et al., 2015]

Task	Query Classification				Web Search
	Restaurant	Hotel	Flight	Nightlife	
Train	1,585K	2,131K	1,880K	1,214K	4M queries, click-through docs
Test	3,074	6,307	6,199	298	12K queries / 897K docs

1. Train jointly with all 5 tasks with shared hidden layer

Model	Query Classification (Accuracy ↑)			
	Restaurant	Hotel	Flight	Nightlife
Single-task DNN	97.38	76.81	95.58	93.24
Multi-task DNN	**97.57**	**78.56**	**96.21**	**94.20**

Model	Web Search (NDCG ↑)		
	@1	@3	@10
Single-task DNN	0.327	0.359	0.432
Multi-task DNN	**0.334**	**0.363**	**0.434**

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

A note on Computational Infrastructure

A note on Computational Infrastructure

- We always do our best: use biggest computer and largest dataset

A note on Computational Infrastructure

- We always do our best: use biggest computer and largest dataset
- Don't pre-maturely performance-optimize your code/hardware
  - But be aware of ongoing efforts: [Coates et al., 2013, Dean et al., 2012]

A note on Computational Infrastructure

- We always do our best: use biggest computer and largest dataset
- Don't pre-maturely performance-optimize your code/hardware
  - But be aware of ongoing efforts: [Coates et al., 2013, Dean et al., 2012]
- Invest in hyper-parameter tuning
  - Personal preference: I'd choose 10 low-end GeForce GPUs in favor of 1 high-end Tesla GPU to run more tuning experiments in parallel

Lots of hyper-parameters!

1. Number of layers
2. Number of nodes per layer
3. SGD learning rate
4. Detailed configurations of optimization method
5. Regularization constant
6. Mini-batch size
7. Type of non-linearity
8. Type of distribution for random initialization

Important to invest in finding good settings for your data

- difference between a winning system vs. useless system

Approaches to hyper-parameter tuning

1. Exhaustive grid search

Approaches to hyper-parameter tuning

1. Exhaustive grid search
2. Intelligent manual search, a.k.a Graduate Student Descent (GSD)

Approaches to hyper-parameter tuning

1. Exhaustive grid search
2. Intelligent manual search, a.k.a Graduate Student Descent (GSD)
3. Treat as a meta machine learning problem [Bergstra et al., 2011]
   - Input x = space of hyper-parameters
   - Output y = validation error after training with given hyper-parameters

Approaches to hyper-parameter tuning

1. Exhaustive grid search
2. Intelligent manual search, a.k.a Graduate Student Descent (GSD)
3. Treat as a meta machine learning problem [Bergstra et al., 2011]
   - Input x = space of hyper-parameters
   - Output y = validation error after training with given hyper-parameters
   - Computing y is expensive, so we learn a function f(x) that can predict it based on past (x, y) pairs
     - e.g. Gaussian Process [Bergstra et al., 2011], Evolutionary Algorithms [Moriya et al., 2015]

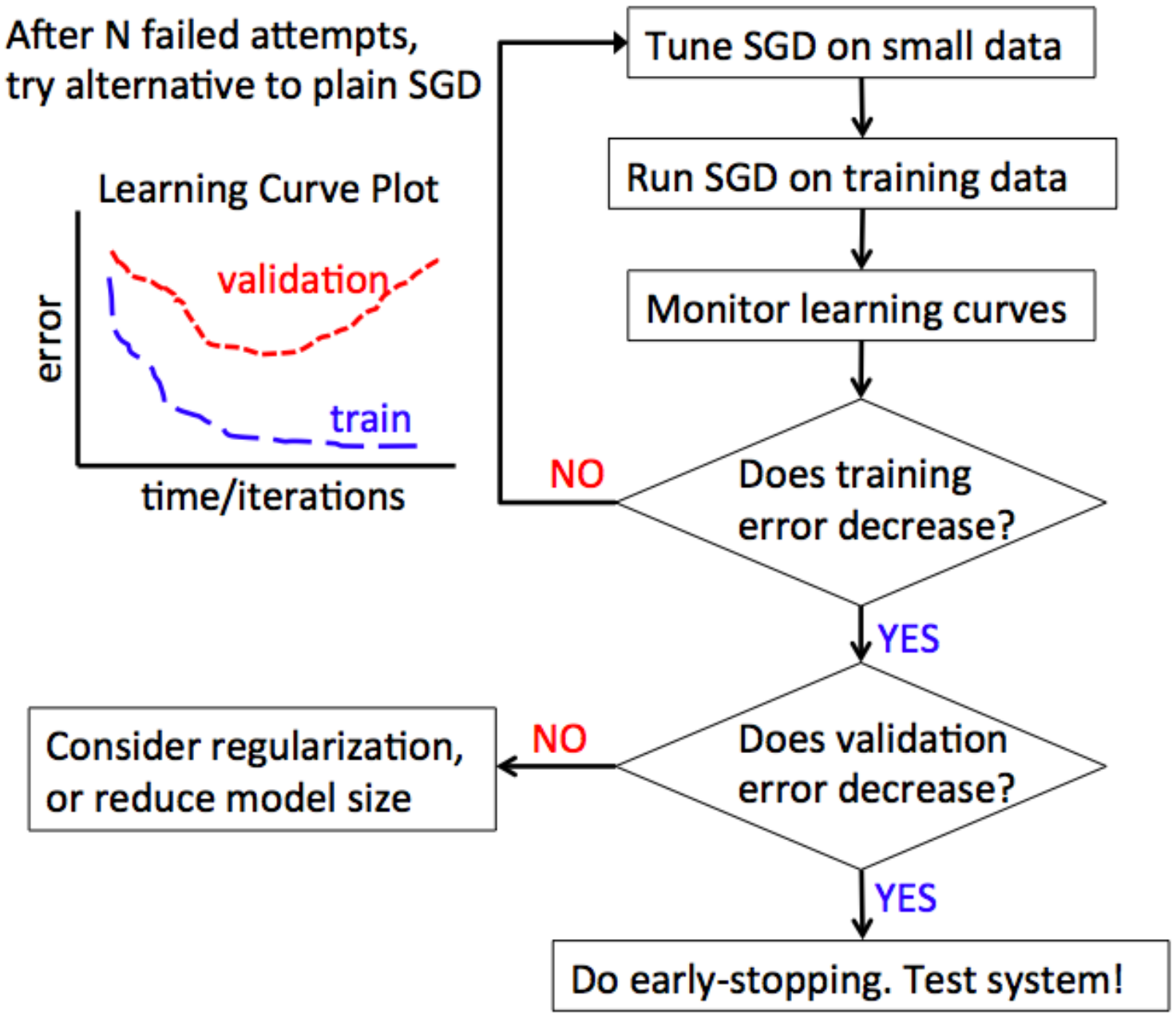
Approaches to hyper-parameter tuning

1. Exhaustive grid search
2. Intelligent manual search, a.k.a Graduate Student Descent (GSD)
3. Treat as a meta machine learning problem [Bergstra et al., 2011]
   - Input x = space of hyper-parameters
   - Output y = validation error after training with given hyper-parameters
   - Computing y is expensive, so we learn a function f(x) that can predict it based on past (x, y) pairs
     - e.g. Gaussian Process [Bergstra et al., 2011], Evolutionary Algorithms [Moriya et al., 2015]

Future research: Green Learning

- energy-efficient & environmentally-friendly training procedures
- someone, please work on this!

Section Summary



Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

So far, we've focused on classification

- Many interesting real-world problems are more complex
- e.g. Structured prediction, Generation
- Encoder-Decoders*: a general term for architectures that solve the above
  - Encodes input **x** (e.g. image) into some representation
  - Decodes representation to generate structured output **y** (e.g. sentence caption)

*Not to be confused with auto-encoders

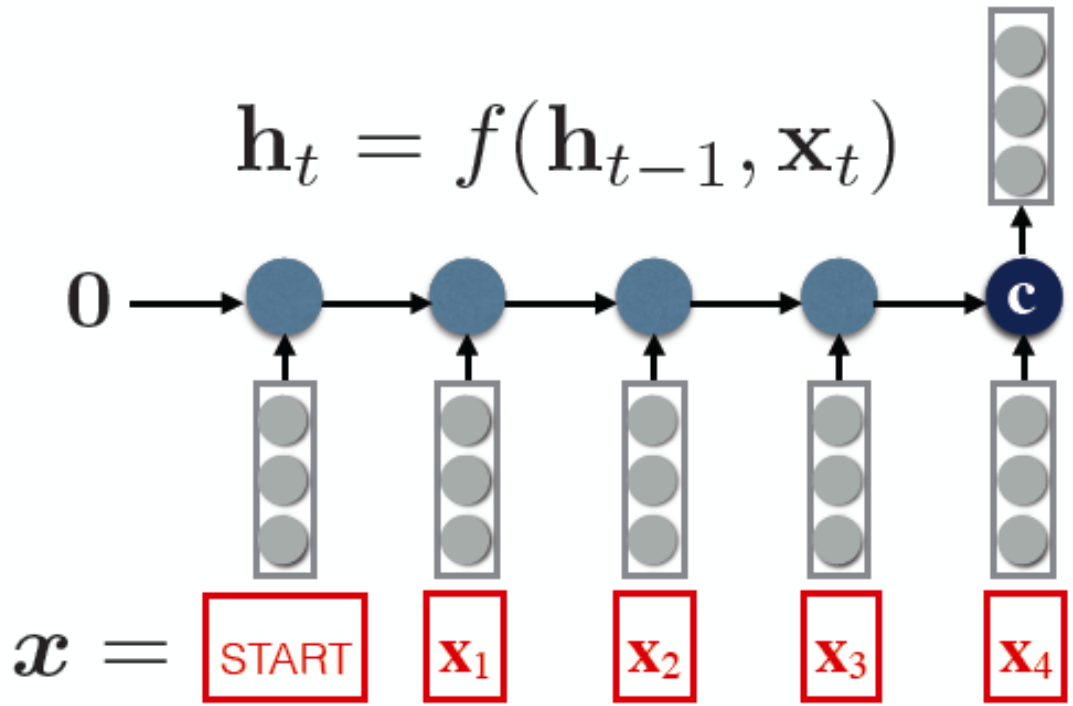
Encoder-Decoder explanation (1)

Recall the recurrent neural net (RNN)

Final hidden state $\mathbf{c}$ can be used as "encoding" of sequence

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

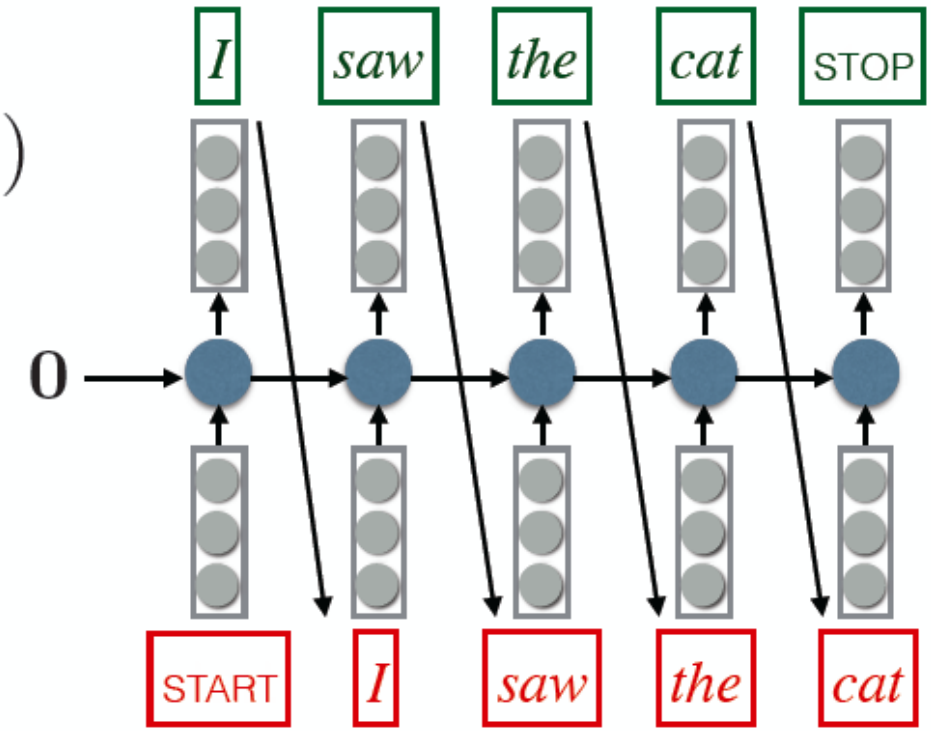


Encoder-Decoder explanation (2)

Recall the recurrent neural net (RNN)

Training objective: predict next word (classification)

$$\text{while } y_t \neq \text{STOP}$$
$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$y_t \sim g(\mathbf{h}_t)$$
$$t \leftarrow t + 1$$



Encoder-Decoder explanation (3)

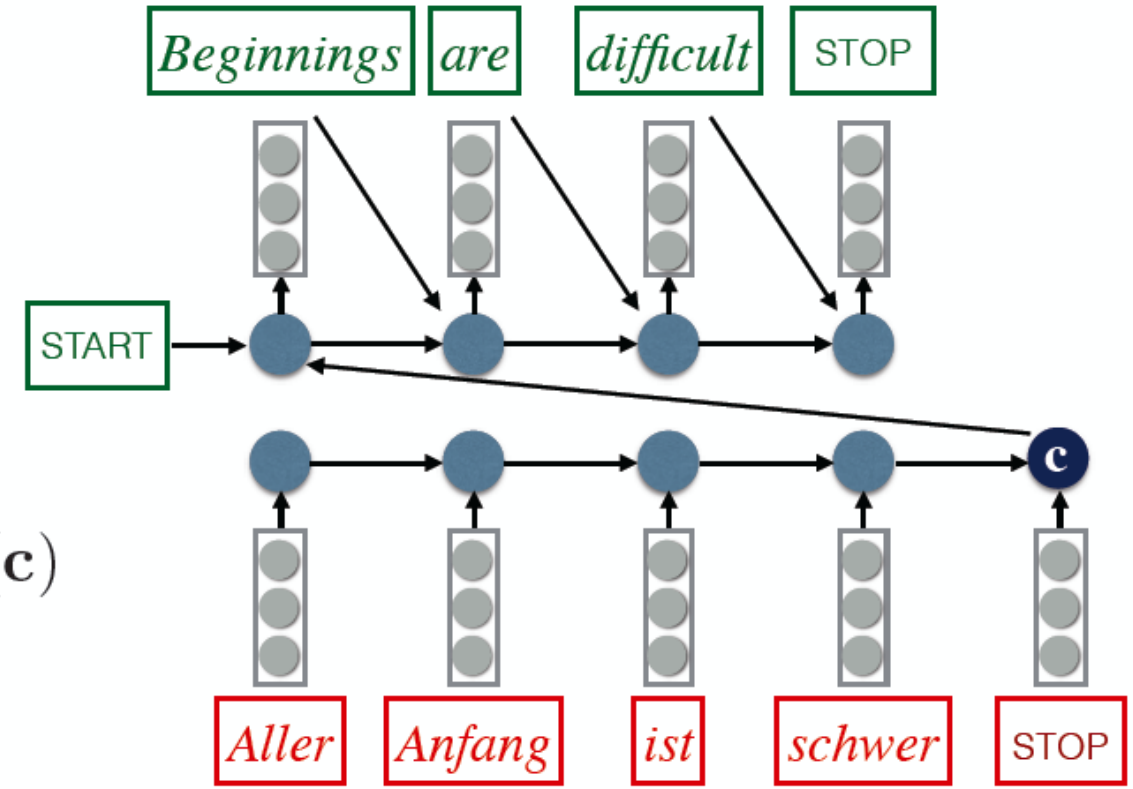
Simplest Encoder-Decoder:

One RNN for encoding, another RNN for decoding

- Decoding can be done by beam search until stop symbol
- Active research: what is the best encoder or decoder for a specific problem?

$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$

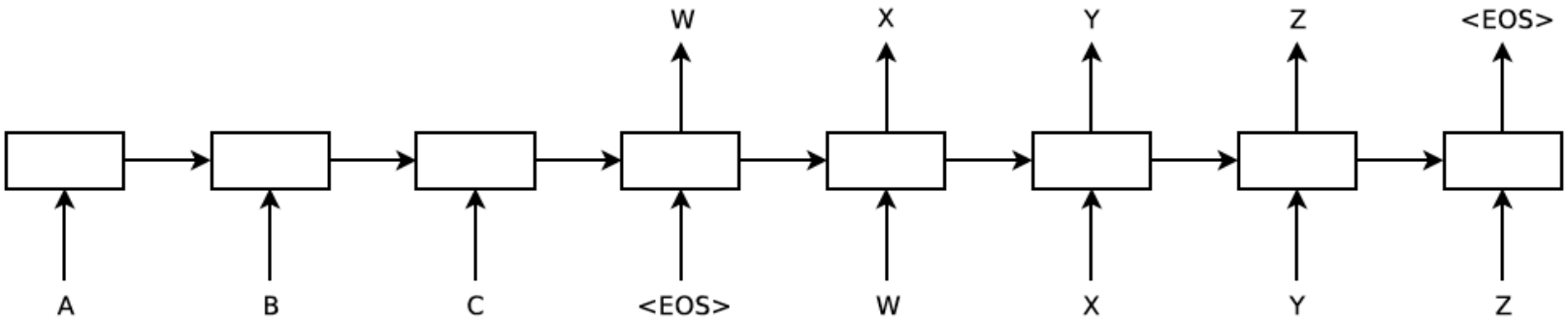
$$\boldsymbol{y} \mid \mathbf{c} \sim \mathrm{RNNLM}(\mathbf{c})$$



Encoder-Decoders for Machine Translation [Sutskever et al., 2014]

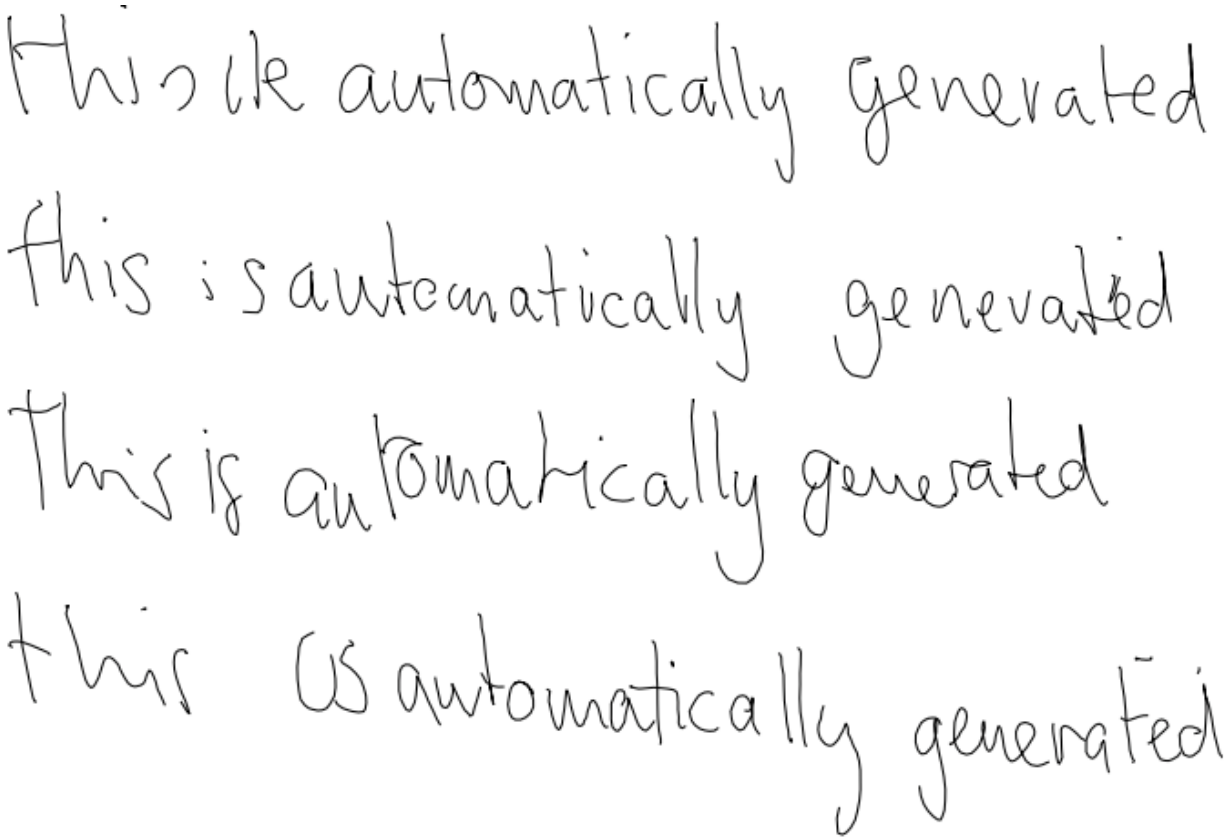
("A B C" is source sentence; "W X Y Z" is target sentence)

Each block is a (deep) LSTM



- Training procedure
  1. Encode source sentence as vector $\mathbf{c}$
  2. Condition on $\mathbf{c}$, generate target sentence
  3. Back-prop to optimize target likelihood
- See
  [Cho et al., 2014, Kalchbrenner and Blunsom, 2013] for other encoder-decoder architecture

Encoder-Decoders for Handwriting Generation



http://www.cs.toronto.edu/~graves/handwriting.html

Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Attention: one motivation [Bahdanau et al., 2014]

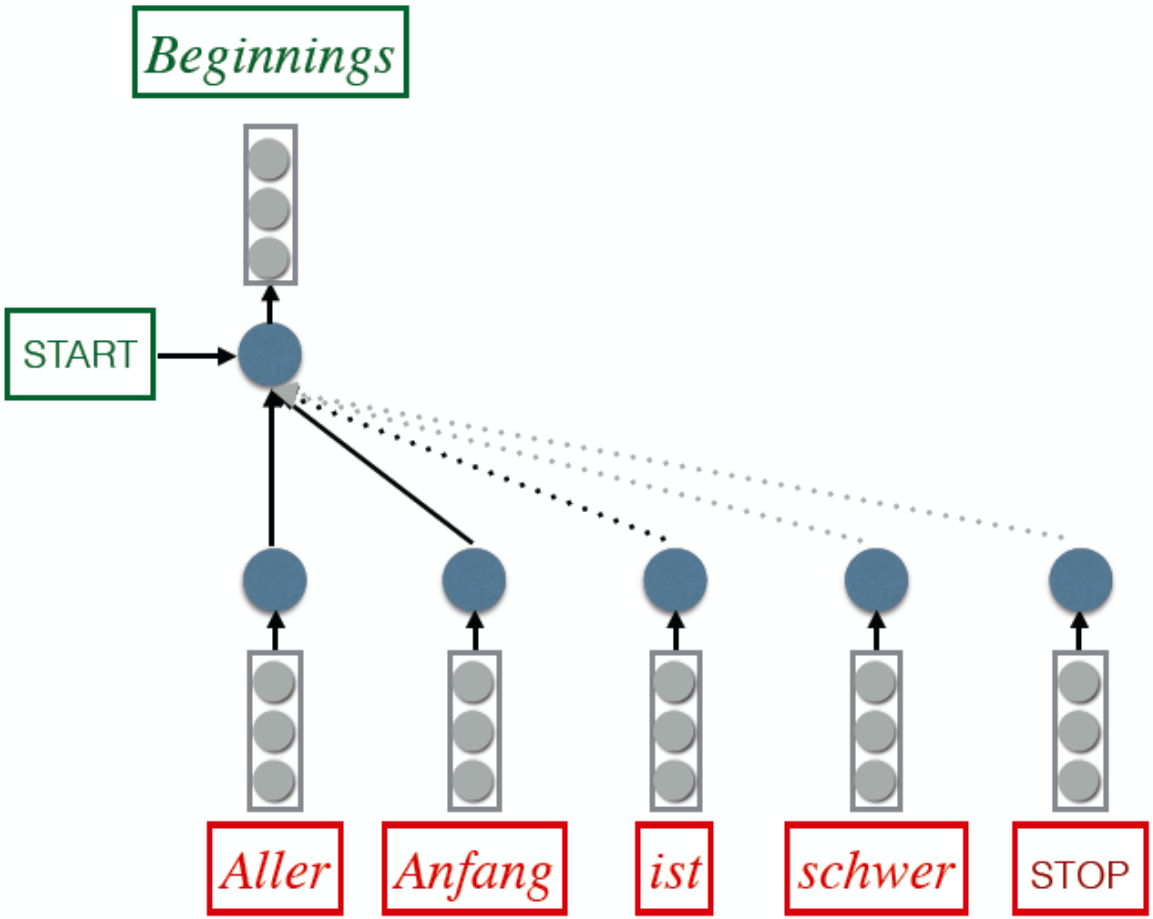
- It seems too much to expect one vector encoding of input $\mathbf{c}$ can do everything

Attention: one motivation [Bahdanau et al., 2014]

- It seems too much to expect one vector encoding of input $\mathbf{c}$ can do everything
- Idea: During decoding, dynamically pay attention to different parts of the input.

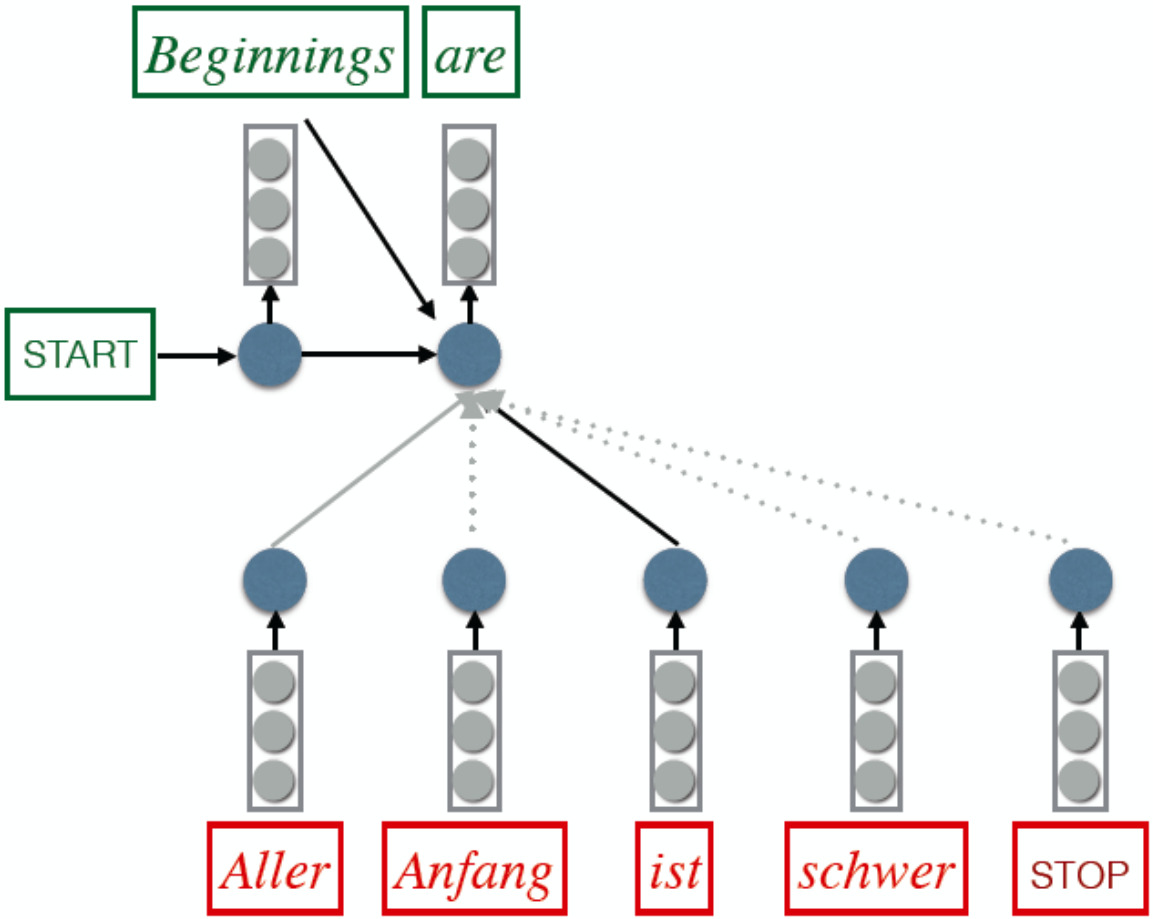
Attention: one motivation [Bahdanau et al., 2014]

- It seems too much to expect one vector encoding of input $\mathbf{c}$ can do everything
- Idea: During decoding, dynamically pay attention to different parts of the input.



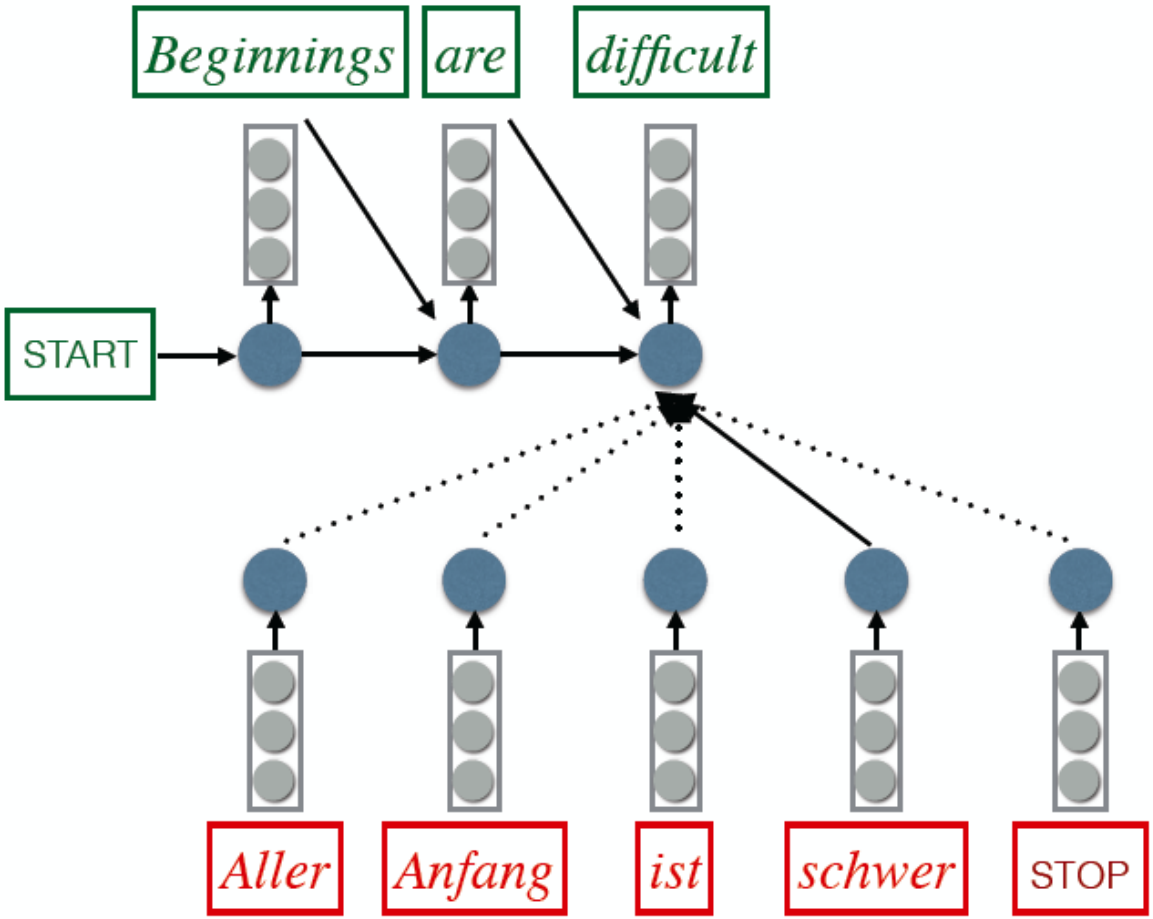
Attention: one motivation [Bahdanau et al., 2014]

- It seems too much to expect one vector encoding of input $\mathbf{c}$ can do everything
- Idea: During decoding, dynamically pay attention to different parts of the input.



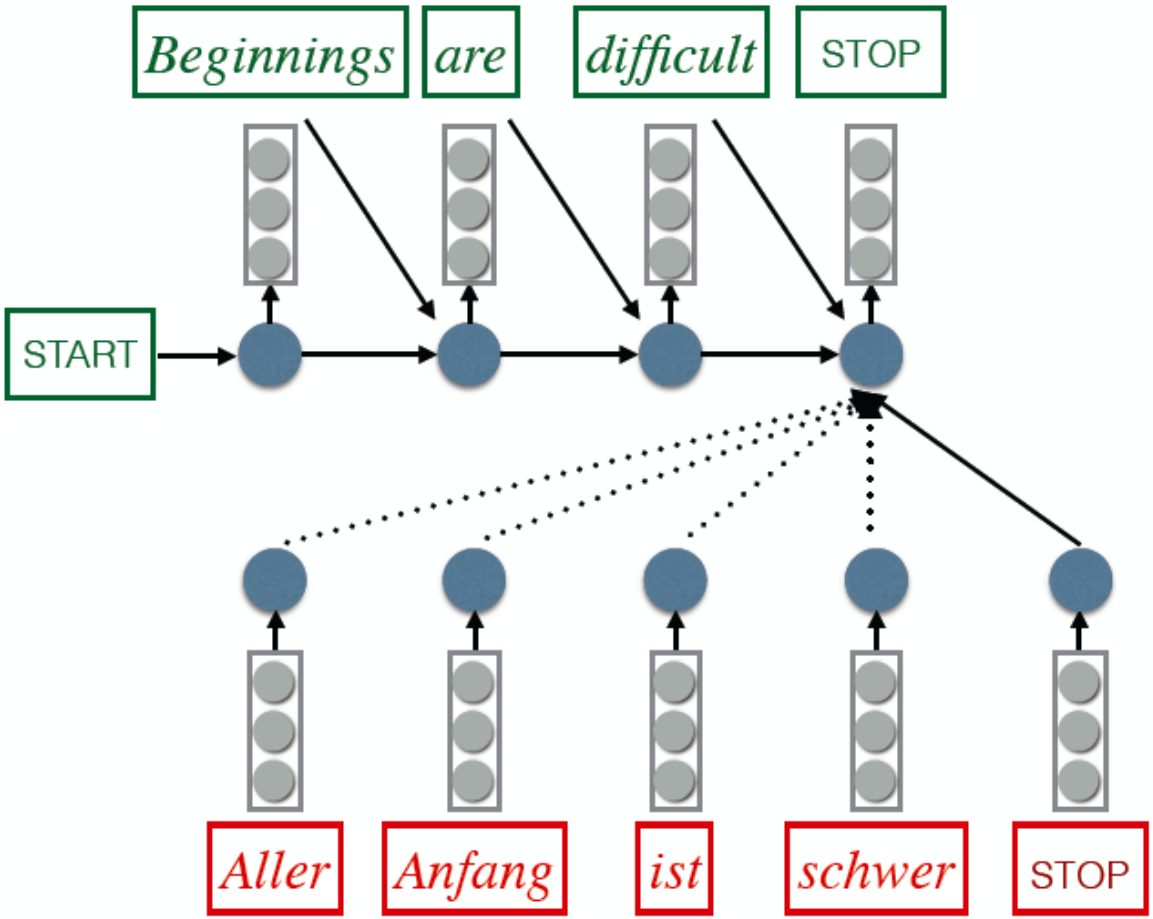
Attention: one motivation [Bahdanau et al., 2014]

- It seems too much to expect one vector encoding of input $\mathbf{c}$ can do everything
- Idea: During decoding, dynamically pay attention to different parts of the input.



Attention: one motivation [Bahdanau et al., 2014]

- It seems too much to expect one vector encoding of input $\mathbf{c}$ can do everything
- Idea: During decoding, dynamically pay attention to different parts of the input.



Encoder-Decoders with Attention

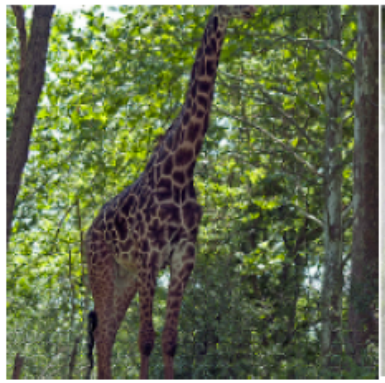
Caption Generation [Xu et al., 2015]:

- Input = image, Output = sentence caption
- Attention = image patch


A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.


A group of people sitting on a boat in the water.

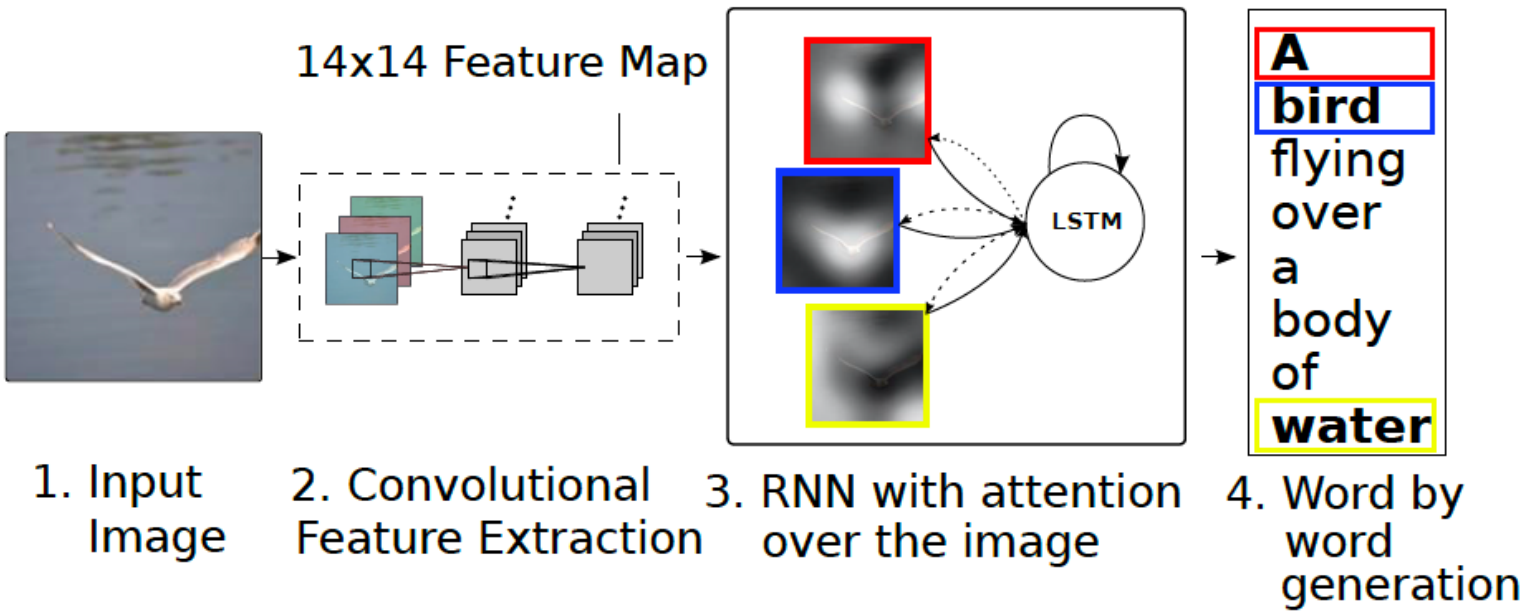



A giraffe standing in a forest with trees in the background.

Encoder-Decoders with Attention

Caption Generation [Xu et al., 2015]:

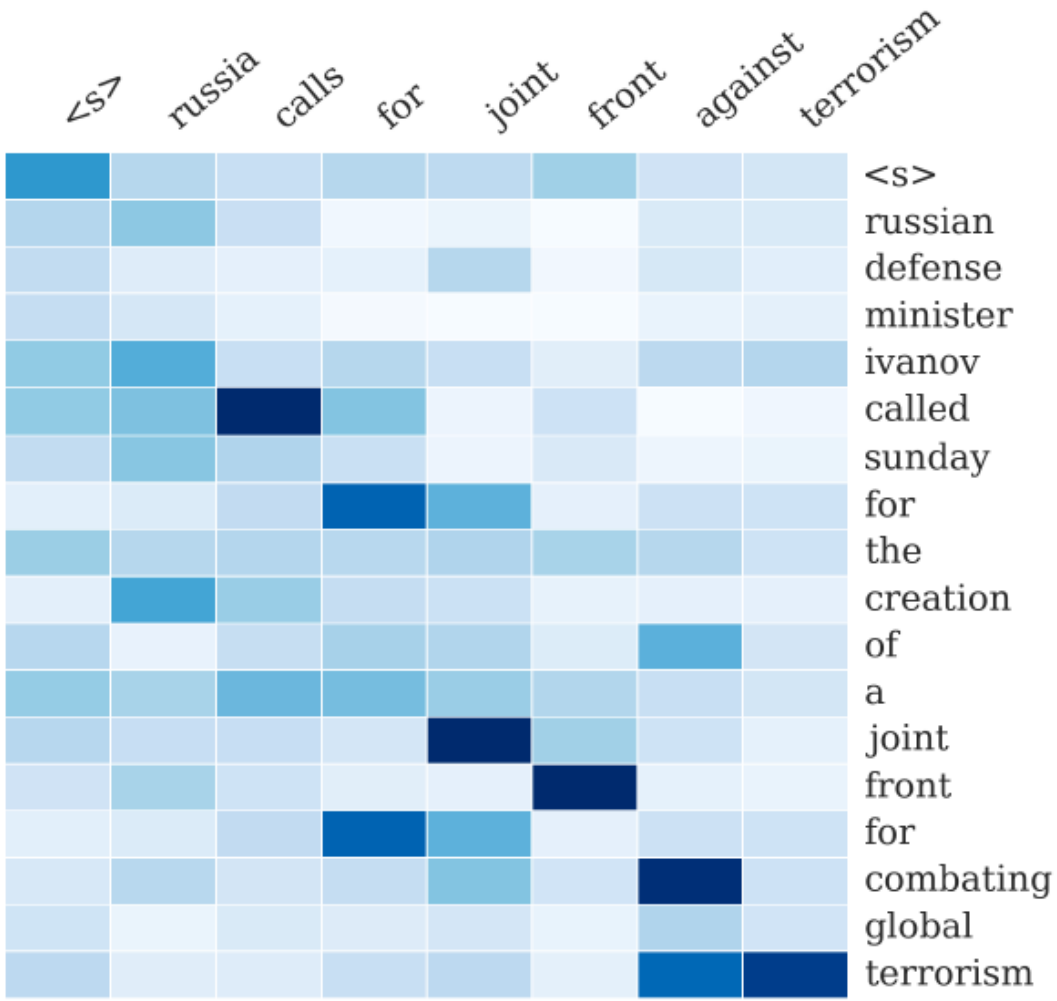
- Input = image, Output = sentence caption
- Attention = image patch



Encoder-Decoders with Attention

Abstractive Sentence Summarization [Rush et al., 2015]:

- Input = Long sentence, Output = Short sentence
- Attention = words



Attention, Memory, and Neural Turing Machines

Attention, Memory, and Neural Turing Machines

- Let's ponder about human learning:

Attention, Memory, and Neural Turing Machines

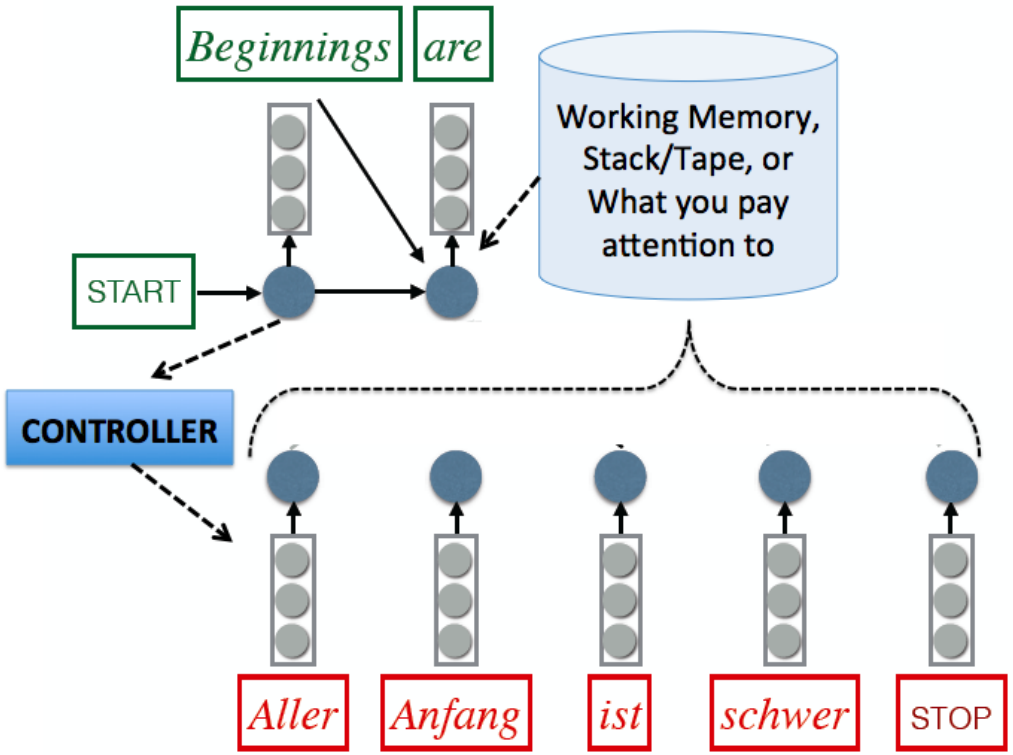
- Let's ponder about human learning:
  - You can't learn if you don't pay attention

Attention, Memory, and Neural Turing Machines

- Let's ponder about human learning:
  - You can't learn if you don't pay attention
  - Need to keep relevant concepts in working memory to build up new concepts

Attention, Memory, and Neural Turing Machines

- Let's ponder about human learning:
  - You can't learn if you don't pay attention
  - Need to keep relevant concepts in working memory to build up new concepts
- Active research: Finding new architectures (preferably differentiable) that allows this kind of learning [Graves et al., 2014, Weston et al., 2014]



Outline

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Reinforcement Learning

- Reinforcement Learning can be viewed as general-purpose tool for AI
  - We have an agent with ability to act
  - Each action affects future state
  - Plan a set of actions to maximize final reward

Reinforcement Learning

- Reinforcement Learning can be viewed as general-purpose tool for AI
  - We have an agent with ability to act
  - Each action affects future state
  - Plan a set of actions to maximize final reward
- Policy π: selects actions given states $a = \pi(s)$
- Value function
  $Q^\pi(s,a) = E_\pi[r_{t+1} + \eta r_{t+2} + \eta^2 r_{t+3} \cdots | s, a]$: expected future reward from (s,a) under π
- Optimal Value function
  $Q^\pi(s,a) = E_\pi[r_{t+1} + \eta r_{t+2} + \eta^2 r_{t+3} \cdots | s, a]$: expected future reward from (s,a) under any policy.

Deep Reinforcement Learning (Q-learning approach)

- Estimate value function by neural net:
  $Q(s, a; w) \approx Q^{\pi}(s, a)$
- Training objective: mean square error of Q-values:
  $L(w) = E[(y - Q(s, a; w))^2]$

  where $y = E[r + \eta \max_{a'} Q(s', a'; w_{previous})|s, a]$ is optimal value under previous weights
- Train by playing out the actions. See [Mnih et al., 2013] for important tricks to get it working

Deep Reinforcement Learning for Playing Atari [Mnih et al., 2013]

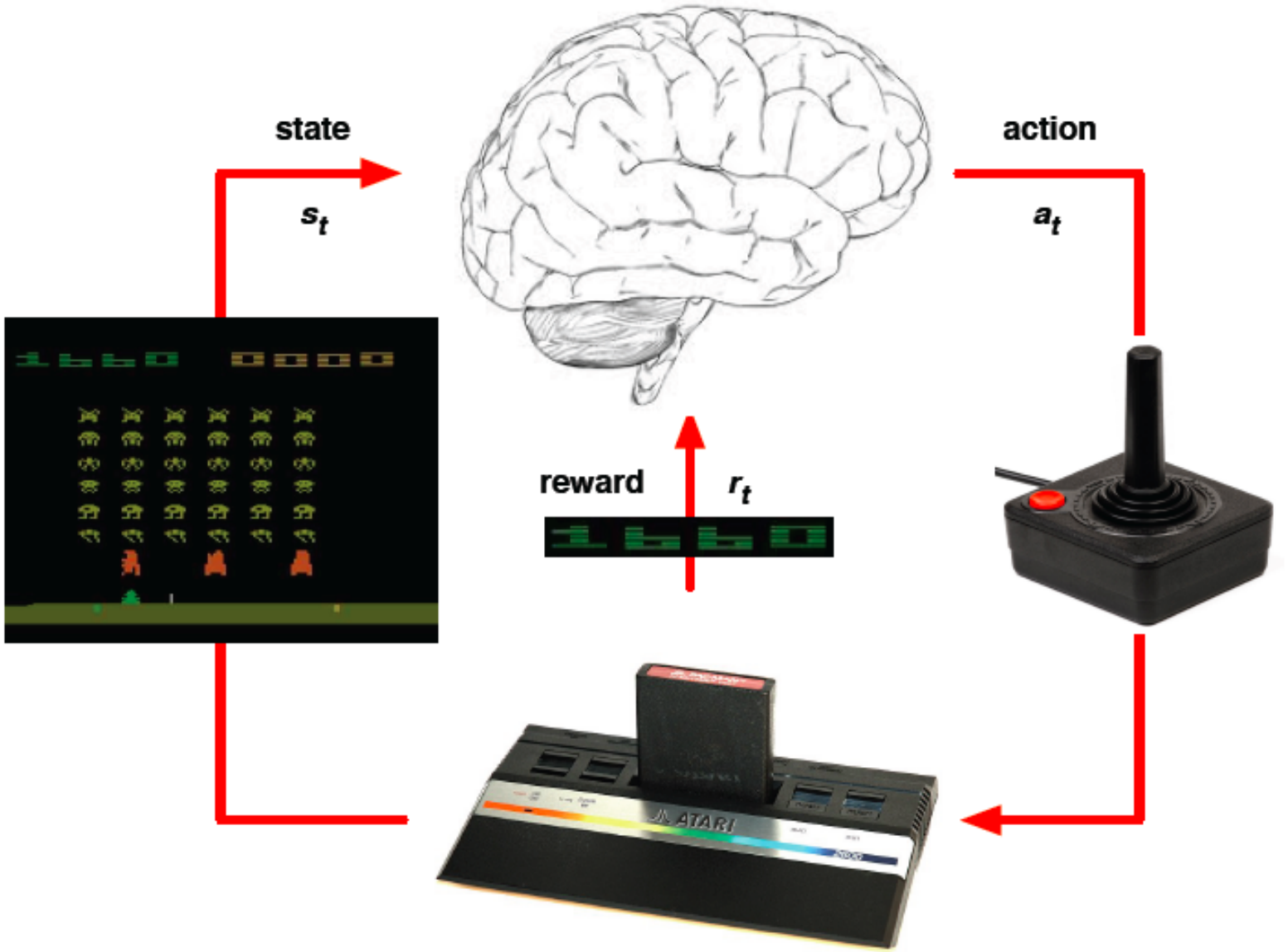


Figure from D. Silver's ICLR2015 keynote slides

Outline

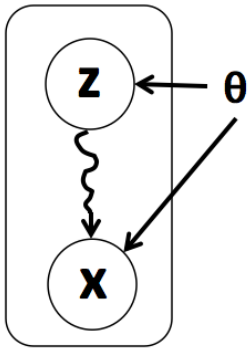
1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Motivation: Directed Probabilistic Models

- How to perform learning and inference on directed probabilistic models, in the presence of continuous latent variables with intractable posteriors?

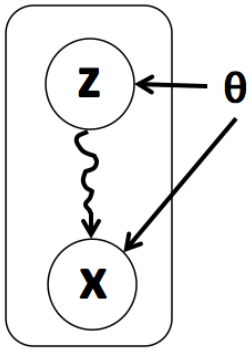
Motivation: Directed Probabilistic Models

- How to perform learning and inference on directed probabilistic models, in the presence of continuous latent variables with intractable posteriors?
- Generative process:
  1. Draw continuous latent variable $\mathbf{z} \sim$ prior $p_{\theta^*}(\mathbf{z})$
  2. Draw data $\mathbf{x} \sim p_{\theta^*}(\mathbf{x}|\mathbf{z})$
     - True $\mathbf{z}$ for each data and parameter θ^* are unknown



Motivation: Directed Probabilistic Models

- How to perform learning and inference on directed probabilistic models, in the presence of continuous latent variables with intractable posteriors?
- Generative process:
  1. Draw continuous latent variable $\mathbf{z} \sim$ prior $p_{\theta^*}(\mathbf{z})$
  2. Draw data $\mathbf{x} \sim p_{\theta^*}(\mathbf{x}|\mathbf{z})$
     - True $\mathbf{z}$ for each data and parameter θ^* are unknown



- Assume marginal likelihood $p_\theta(\mathbf{x})$ and posterior $p_\theta(\mathbf{z}|\mathbf{x})$ are intractable. Can't differentiate likelihood. Can't do EM.

Autoencoding Variational Bayes [Kingma and Welling, 2014]

- Idea: Introduce recognition model $p_\phi(\mathbf{z}|\mathbf{x})$ as approximation to posterior $p_\theta(\mathbf{z}|\mathbf{x})$.
- Unlike mean-field variational inference, it's not factorial & parameters need not be computed in closed-form.

Autoencoding Variational Bayes [Kingma and Welling, 2014]

- Idea: Introduce recognition model $p_\phi(\mathbf{z}|\mathbf{x})$ as approximation to posterior $p_\theta(\mathbf{z}|\mathbf{x})$.
- Unlike mean-field variational inference, it's not factorial & parameters need not be computed in closed-form.
- Reparameterize $\hat{\mathbf{z}} \sim p_\phi(\mathbf{z}|\mathbf{x})$ as $\hat{\mathbf{z}} = g_\phi(\mathbf{x}, \epsilon)$
  - $g_\phi()$ is non-linear transform, e.g. MLP
  - ϵ is drawn from noise distribution
  - Using this, one can take derivative of VB bound!

Autoencoding Variational Bayes [Kingma and Welling, 2014]

- Idea: Introduce recognition model $p_\phi(\mathbf{z}|\mathbf{x})$ as approximation to posterior $p_\theta(\mathbf{z}|\mathbf{x})$.
- Unlike mean-field variational inference, it's not factorial & parameters need not be computed in closed-form.
- Reparameterize $\hat{\mathbf{z}} \sim p_\phi(\mathbf{z}|\mathbf{x})$ as $\hat{\mathbf{z}} = g_\phi(\mathbf{x}, \epsilon)$
  - $g_\phi()$ is non-linear transform, e.g. MLP
  - ϵ is drawn from noise distribution
  - Using this, one can take derivative of VB bound!
- Implications:
  - Training deep generative models on large data
  - Semi-supervised learning [Kingma et al., 2014]

Section Summary

- Encoder-Decoder Architectures
- Attention and Memory Mechanism
- Deep Reinforcement Learning
- Auto-Encoding Variational Bayes

Review: what we covered

1. Deep Learning Basics
   - Neural Networks (1-layer, 2-layer)
   - Computation Graphs and Deep Learning Toolkits
   - Why Deep Architectures are Hard
   - The Breakthrough in 2006
2. Building Blocks of Deep Architectures
   - Restricted Boltzmann Machines (RBM)
   - Auto-Encoders
   - Recurrent Units
   - Convolution
3. Tricks of the Trade
   - Optimization: Making SGD work, Adaptive Learning Rate, 2nd-order methods
   - Regularization: Dropout, Multi-task learning
   - Computational Infrastructure
4. New and Exciting Stuff
   - Encoder-Decoder Architectures
   - Attention and Memory Mechanism
   - Deep Reinforcement Learning
   - Auto-Encoding Variational Bayes

Roadmap next

1. We've covered general background in deep learning
2. Dr. Tsuyoshi Okita: Deep learning / Neural Nets in NLP
3. Prof. Hermann Ney: Neural Nets for MT
4. Prof. Kyunghyun Cho: Neural Nets as MT

References:

Bahdanau, D., Cho, K., and Bengio, Y. (2014).
Neural machine translation by jointly learning to align and translate.
CoRR, abs/1409.0473.

Bengio, Y. (2009).
Learning Deep Architectures for AI, volume Foundations and Trends in Machine Learning.
NOW Publishers.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2006).
Greedy layer-wise training of deep networks.
In *NIPS'06*, pages 153–160.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégel, B. (2011).
Algorithms for hyper-parameter optimization.
In *Proc. Neural Information Processing Systems 24 (NIPS2011)*.

Bishop, C. (1995).
Neural Networks for Pattern Recognition.
Oxford University Press.

Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2012).
Joint learning of words and meaning representations for open-text semantic parsing.
In *AISTATS*.

Carreira-Perpinan, M. A. and Hinton, G. E. (2005).
On contrastive divergence learning.
In *AISTATS*.

Caruana, R. (1997).
Multitask learning.
Machine Learning, 28.

Chen, D. and Manning, C. (2014).
A fast and accurate dependency parser using neural networks.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014).
Learning phrase representations using rnn encoder-decoder for statistical machine translations.
In *Conference on Empirical Methods in Natural Language Processing (EMNLP) 2014*, number 1406.1078 in cs.CL.

Coates, A., Huval, B., Wang, T., Wu, D. J., Catanzaro, B., and Ng, A. Y. (2013).
Deep learning with COTS HPC systems.
In *Proceedings of the International Conference on Machine Learning (ICML)*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011).
Natural language processing (almost) from scratch.
Journal of Machine Learning Research, 12:2493–2537.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012).
Large scale distributed deep networks.
In *Neural Information Processing Systems (NIPS)*.

Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive subgradient methods for online learning and stochastic optimization.
JMLR, 12:2121–2159.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P., Vincent, P., and Bengio, S. (2010).
Why does unsupervised pre-training help deep learning?
Journal of Machine Learning Research, 11:625–660.

Erhan, D., Manzagol, P., Bengio, Y., Bengio, S., and Vincent, P. (2009).
The difficulty of training deep architectures and the effect of unsupervised pre-training.
In *AISTATS*.

Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002).
Learning precise timing with LSTM recurrent networks.
JMLR.

Graves, A., Wayne, G., and Danihelka, I. (2014).
Neural turing machines.
CoRR, abs/1410.5401.

Hinton, G., Osindero, S., and Teh, Y.-W. (2006).
A fast learning algorithm for deep belief nets.
Neural Computation, 18:1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012).
Improving neural networks by preventing co-adaptation of feature detectors.
CoRR, abs/1207.0580.

Kalchbrenner, N. and Blunsom, P. (2013).
Recurrent continuous translation models.
In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA. Association for Computational Linguistics.

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., and Welling, M. (2014).
Semi-supervised learning with deep generative models.
In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 27*, pages 3581–3589. Curran Associates, Inc.

Kingma, D. P. and Welling, M. (2014).
Auto-encoding variational bayes.
In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012).
Imagenet classification with deep convolutional neural networks.
In *NIPS*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).
Gradient-based learning applied to document recognition.
Proc, 86(11):2278–2324.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. (2009).
Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.
In *ICML*.

Liu, X., Gao, J., He, X., Deng, L., Duh, K., and Wang, Y.-Y. (2015).
Representation learning using multi-task deep neural networks for semantic classification and information retrieval.
In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 912–921, Denver, Colorado. Association for Computational Linguistics.

Martens, J. (2010).
Deep learning via Hessian-free optimization.
In *Proceedings of the 27th International Conference on Machine Learning (ICML)*.

Martens, J. and Sutskever, I. (2011).
Learning recurrent neural networks with hessian-free optimization.
In *Proceedings of the 28th International Conference on Machine Learning (ICML)*.

Mikolov, T., Karafiat, S., Burget, L., Černocký, J., and Khudanpur, S. (2010).
Recurrent neural network based language models.
In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*.

Mnih, V., K., K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013).
Playing atari with deep reinforcement learning.
Technical Report Technical Report arXiv:1312.5602, Deepmind Technologies.

Moriya, T., Tanaka, T., Shinozaki, T., Watanabe, S., and Duh, K. (2015).
Automation of system building for state-of-the-art large vocabulary speech recognition using evolution strategy.
In *Proceedings of ASRU*.

Rush, A. M., Chopra, S., and Weston, J. (2015).
A neural attention model for abstractive sentence summarization.
In *EMNLP*.

Salakhutdinov, R. and Hinton, G. (2009).
Deep Boltzmann machines.
In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455.

Schaul, T., Zhang, S., and LeCun, Y. (2013).
No more pesky learning rates.
In *Proc. International Conference on Machine learning (ICML'13)*.

Sutskever, I., Vinyals, O., and Le, Q. (2014).
Sequence to sequence learning with neural networks.
In *NIPS*.

Weston, J., Chopra, S., and Bordes, A. (2014).
Memory networks.
CoRR, abs/1410.3916.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015).
Show, attend and tell: Neural image caption generation with visual attention.
In *ICML*.

Yu, D., Eversole, A., Seltzer, M. L., Yao, K., Guenter, B., Kuchaiev, O., Zhang, Y., Seide, F., Chen, G., Wang, H., Droppo, J., Agarwal, A., Basoglu, C., Padmilac, M., Kamenev, A., Ivanov, V., Cyphers, S., Parthasarathi, H., Mitra, B., Huang, Z., Zweig, G., Rossbach, C., Currey, J., Gao, J., May, A., Peng, B., Stolcke, A., Slaney, M., and Huang, X. (2014).
An introduction to computational networks and the computational network toolkit.
Technical Report MSR-TR-2014-112, Microsoft.

Zeiler, M. D. (2012).
ADADELTA: an adaptive learning rate method.
CoRR, abs/1212.5701.