DEEP LEARNING AND THE DREAM OF AI

Brandon Ballinger
@ballaballinger
Strata NY 2013

Deep Networks in the News

"Startling gains in fields as diverse as computer vision, speech recognition and the identification of promising new molecules for designing drugs."

2012 The New York Times

Deep Networks in the News

"Startling gains in fields as diverse as computer vision, speech recognition and the identification of promising new molecules for designing drugs."

2012 The New York Times

"[The deep learning] movement seeks to meld computer science with neuroscience — something that never quite happened in the world of artificial intelligence."

2013 WIRED

Deep Networks in the News

"Startling gains in fields as diverse as computer vision, speech recognition and the identification of promising new molecules for designing drugs."

2012 The New York Times

"[The deep learning] movement seeks to meld computer science with neuroscience — something that never quite happened in the world of artificial intelligence."

2013 WIRED

"This remarkable machine…is capable of what amounts to thought."

1958 THE NEW YORKER

Deep Learning: Empirical Results

Domain	Task	Prior Art	Deep Learning
Text	Paraphrase Detection	76.1%	**76.4%**
Video	Hollywood2 Classification	48%	**53%**
Video	YouTube multi-modal	71.2%	**75.8%**
Images	CIFAR Object Classification	80.5%	**82%**
Images	ImageNet 2011	9.3%	**15.8%**
Speech	Bing Voice Search	63.6%	**69.6%**
Speech	YouTube captions	47.7%	**53.4%**

Rekindling the Dream of AI

"Instead of doing AI, we ended up spending our lives doing curve fitting."

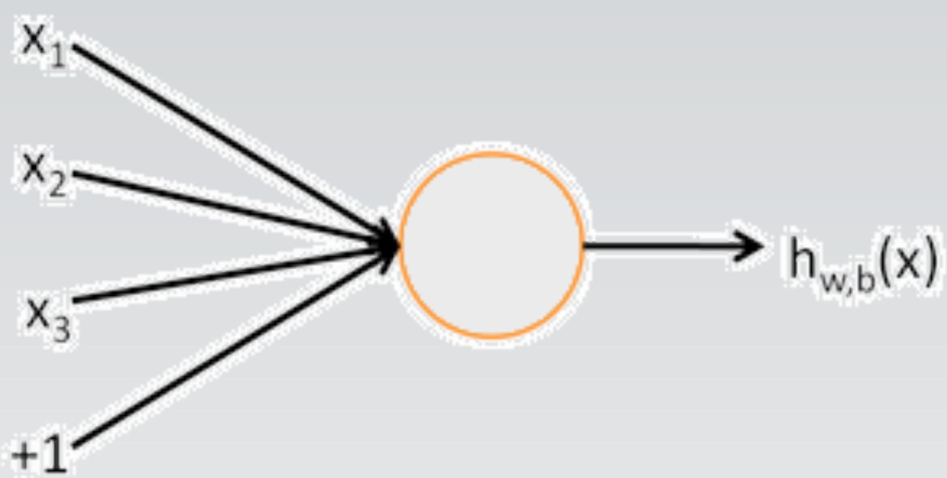
— Andrew Ng

Outline

- Why deep neural nets matter
- **A Brief History of Neural Networks**
  - Perceptron (1957)
  - Backpropagation (1974)
- Deep Neural Networks (2006-present)
  - Unsupervised pretraining: RBMs, greedy layer-wise pretraining
  - Discriminative fine-tuning
  - Dropout, Rectified Linear Units
- Case Studies
  - Speech recognition at Google
  - Molecular Activity Prediction on Kaggle

PERCEPTRON (1957)

Simple model of a single "neuron":



x_i: input vector, e.g. pixels

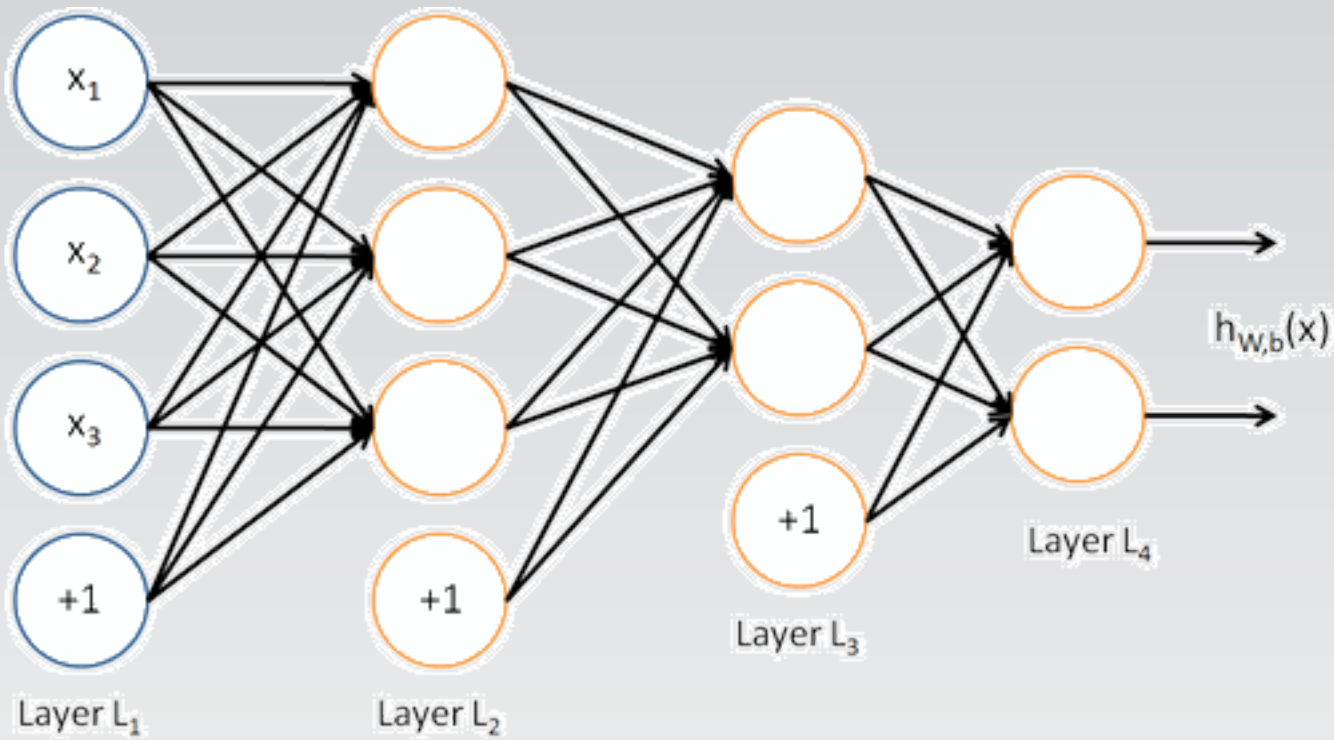
w_i: the strength of each input connection from x[i] to the neuron.

h(x): the output **activation**, using a sigmoid:

$$h(x) = 1 / (1 + \exp(-\sum x_i * w_i))$$

PERCEPTRON (1957)

Neurons are linked into a **neural network**:



x_i: input vector, e.g. pixels

w_{ijl}: connection weight between node i and node j at layer l.

$h_{il}(x)$: the output **activation** of node i and layer l, using a sigmoid:

$$h_{il}(x) = 1 / (1 + \exp(-\sum h_{jl-1} * w_{ijl}))$$

Backpropagation (1974)

But how do you find the weights $\mathbf{w_{ijl}}$?

Idea: minimize squared error **E** between the network's output at last layer (**L**) $\mathbf{h_L}$ and labels **y** from the training set **T** using gradient descent:

$$\mathbf{E(w)} = \sum\nolimits_{\mathbf{T}} (y - h_L(x))^2$$

For the last layer **L,** just take the derivative:

$$\boldsymbol{\partial E / \partial w_{ijL}} = -2 * (y - h_{iL}(x)) \bullet \text{sigmoid}'(\textstyle\sum h_{jL-1} * w_{ijL}))$$

For hidden layers, recursively apply the **chain rule** from calculus:

$$\boldsymbol{\partial E / \partial W_{ijL-1}} = \partial E / \partial h_{jL} \bullet \partial h_{jL} / \partial W_{ijL-1}$$

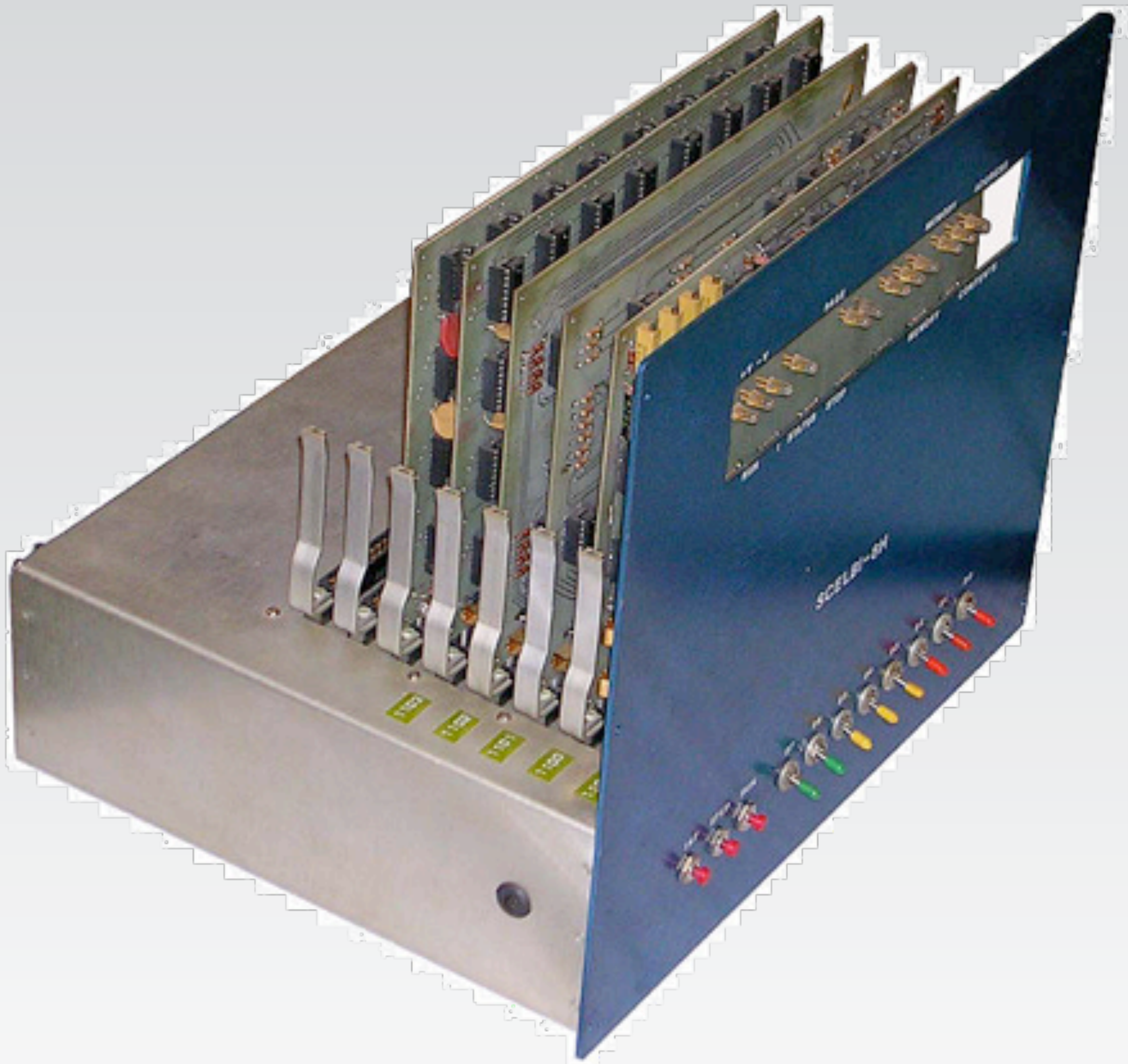
Werbos (1974) *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*
Rumelhart et al. (1986), *Learning representations by back-propagating errors*

PROBLEMS WITH BACKPROPAGATION

- Requires labeled data, and **most data is unlabeled**
- Easy to get stuck in **poor local optima**
  - Gets worse as you add more layers!
- **Slow** to train due to “diffusion of gradients”
  - Also gets worse with more layers!

So what's **different** in 2013?

WHAT'S DIFFERENT: MOORE'S LAW



A computer from 1974: SCELBI 8H

Intel 8008 @ 0.5 Mhz

1KB memory

"A vast improvement over its predecessor, the 4004, its eight-bit word afforded 256 unique arrangements of ones and zeros. For the first time, a microprocessor could handle both uppercase and lowercase letters, all 10 numerals, punctuation marks, and a host of other symbols."

In 2013, more than a **million** times more CPU power.

So what's **different** in 2013?

- Moore's law
- New Algorithms

Outline

- Why deep neural nets matter
- A Brief History of Neural Networks
  - Perceptron (1957)
  - Backpropagation (1974)
- **Deep Neural Networks (2006-present)**
  - Unsupervised pretraining: RBMs, greedy layer-wise pretraining
  - Discriminative fine-tuning
  - Dropout, Rectified Linear Units
- Case Studies
  - Speech recognition at Google
  - Molecular Activity Prediction on Kaggle

CAUTION!

- Deep learning is an active field. Results change often!

UNSUPERVISED PRETRAINING

"If you want to do computer vision, first learn computer graphics."

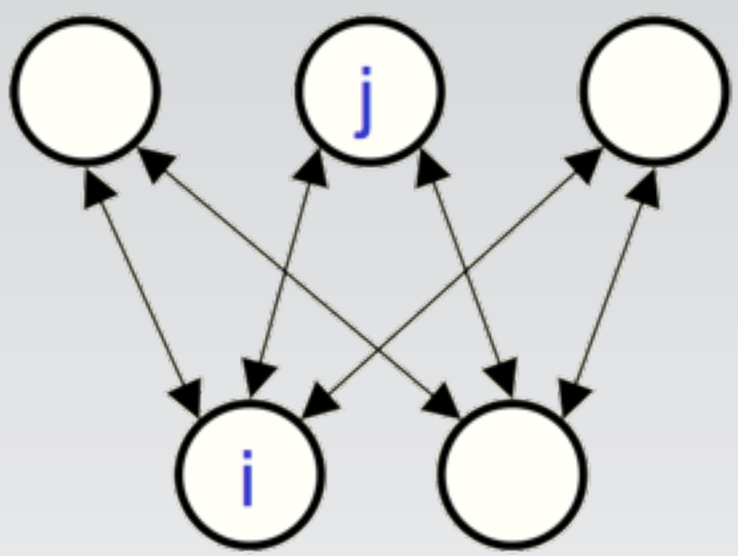
—Geoff Hinton

Add a **pretraining** phase to learn the structure of the input data:

- Requires no labeled data
- Don't get stuck in bad local optima
- A greedy **layer-wise** algorithm makes this efficient and fast
- Related terms: "unsupervised feature learning", "greedy layer-wise pretraining," "generative pretraining," "unsupervised pretraining."

Restricted Boltzman Machines

Restricted Boltzman Machine: two-layer **undirected** network.



Given **v** or **h**, can estimate the other:

$\mathbf{p(h_j=1|v)}$ = sigmoid($-\sum_i v_i w_{ij}$) **(1)**

$\mathbf{p(v_i=1|h)}$ = sigmoid($-\sum_j h_j w_{ij}$) **(2)**

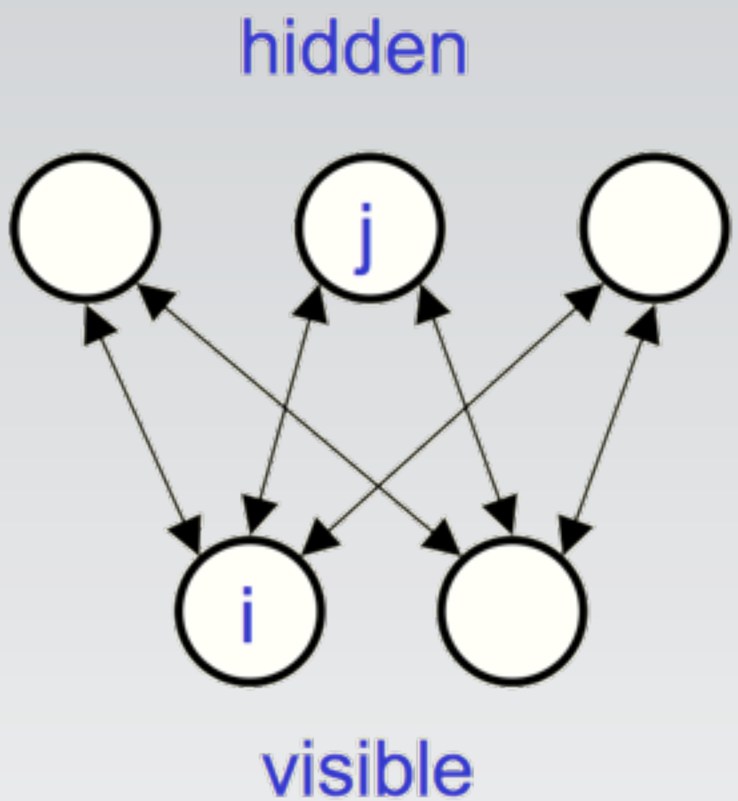
If we apply (1) and (2) iteratively, the RBM "dreams."

How to learn w_{ij}?

Hinton (2002), *Training products of experts by minimizing contrastive divergence*
Hinton (2010), *A practical guide to training Restricted Boltzmann Machines*

Restricted Boltzman Machines

Restricted Boltzman Machine: two-layer **undirected** network.



Given **v** or **h**, can estimate the other:

$\mathbf{p(h_j=1|v)}$ = sigmoid($-\sum_i v_i w_{ij}$) **(1)**

$\mathbf{p(v_i=1|h)}$ = sigmoid($-\sum_j h_j w_{ij}$) **(2)**

If we apply (1) and (2) iteratively, the RBM "dreams."

How to learn w_{ij}? Gradient descent.

Probability of the training set $\mathbf{v^{0..N-1}}$ and its derivative:

$$\mathbf{p(v^{0..N-1})} = 1/N \sum_n \left(\sum_h \exp(-v_i h_j w_{ij}) \,/\, \sum_{v'h'} \exp(-v'_i h'_j w_{ij})\right)$$

$$\Delta w_{ij} = 1/N \sum_i \partial \log(p(v^n))/w_{ij} = \mathbf{\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model}}$$

Hinton (2002), *Training products of experts by minimizing contrastive divergence*
Hinton (2010), *A practical guide to training Restricted Boltzmann Machines*

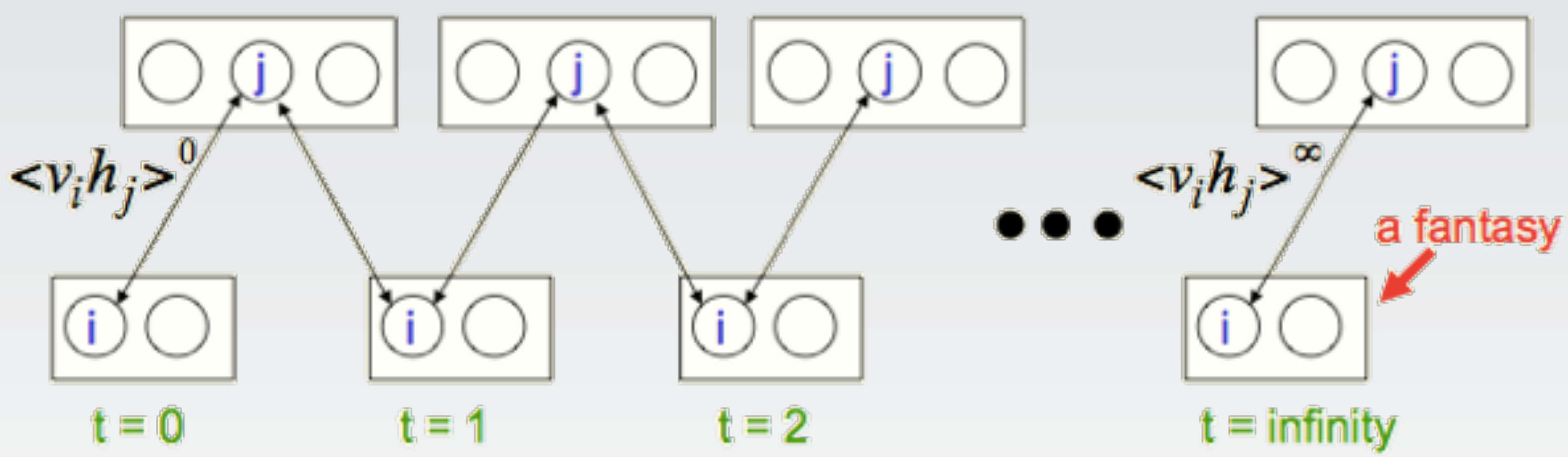
Learning RBMs: Contrastive Divergence

- $\Delta w_{ij} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$
- For $\langle v_i h_j \rangle$data, use equation 1 from previous slide:

  $\mathbf{p(h_j=1|v)} = \text{logistic}(\sum_i v_i w_{ij})$ **(1)**

  $\mathbf{p(v_i=1|h)} = \text{logistic}(\sum_j h_j w_{ij})$ **(2)**

- For $\langle v_i h_j \rangle_{model}$, in theory, use Gibbs sampling, starting with a random v and alternating (1) and (2):



Hinton (2002), *Training products of experts by minimizing contrastive divergence*
Hinton (2010), *A practical guide to training Restricted Boltzmann Machines*

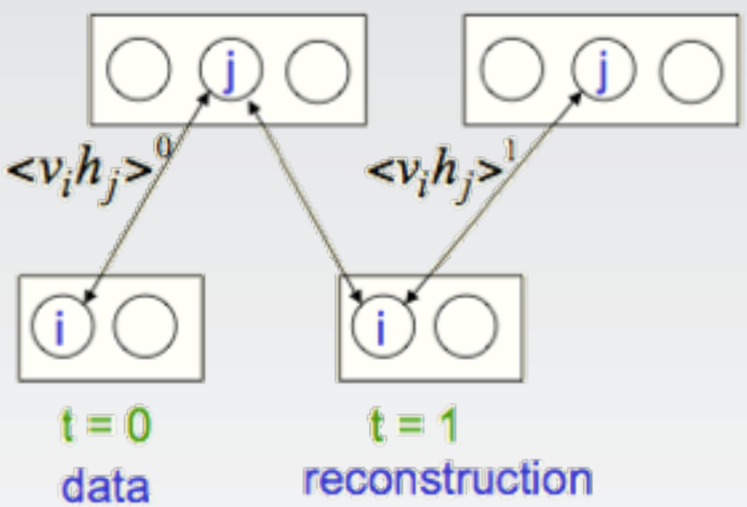
LEARNING RBMS: CONTRASTIVE DIVERGENCE

- Δw_{ij} = **$<v_ih_j>_{data}$ - $<v_ih_j>_{model}$**
- For $<v_ih_j>$data, use equation 1 from previous slide:

  $p(h_j=1|v)$ = logistic($\sum_i v_iw_{ij}$) **(1)**

  $p(v_i=1|h)$ = logistic($\sum_j h_jw_{ij}$) **(2)**

- For $<v_ih_j>_{model}$, in theory, use Gibbs sampling, starting with a random v and alternating (1) and (2):

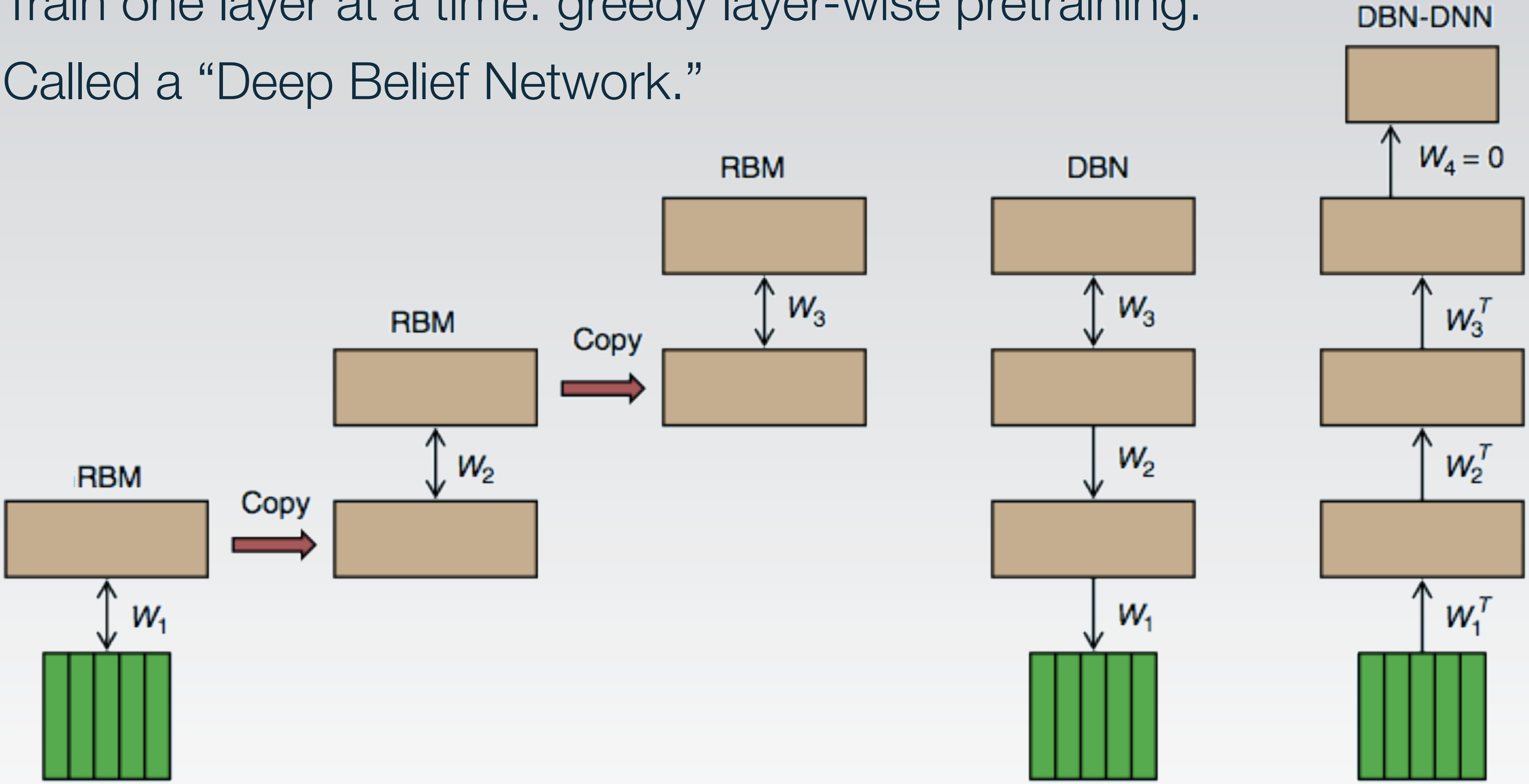


- In practice, a crude approximation called **contrastive divergence works** well: start with a training example v^n and perform just one iteration of (1) and (2)

Hinton (2002), *Training products of experts by minimizing contrastive divergence*
Hinton (2010), *A practical guide to training Restricted Boltzmann Machines*

Stacking RBMs

- To get a deep neural network, we can **stack** RBMs
- Train one layer at a time: greedy layer-wise pretraining.
- Called a "Deep Belief Network."

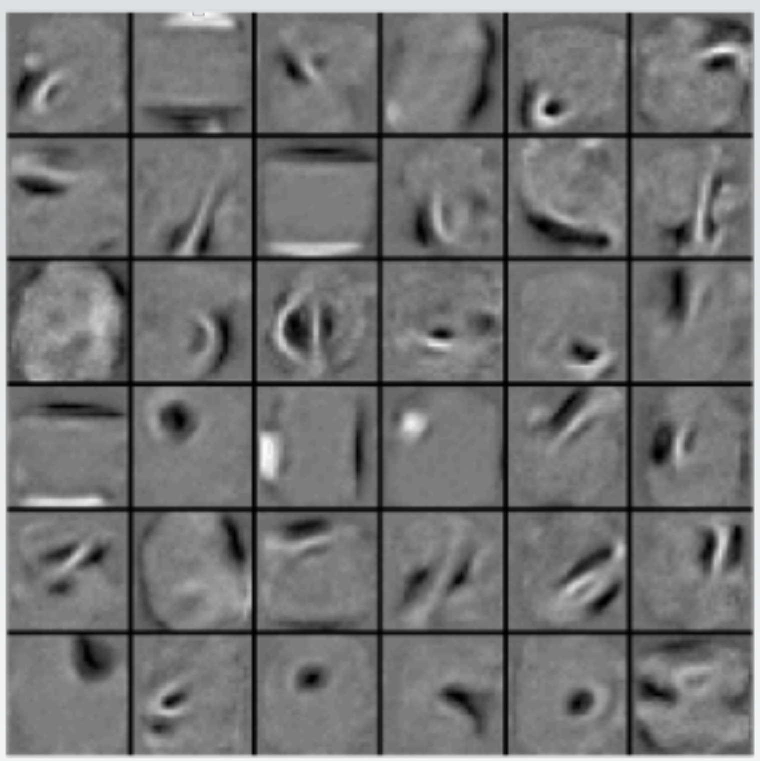


Bengio et al. (2006), *Greedy layer-wise training of deep networks*
Hinton et al. (2007), *A fast learning algorithm for deep belief nets*

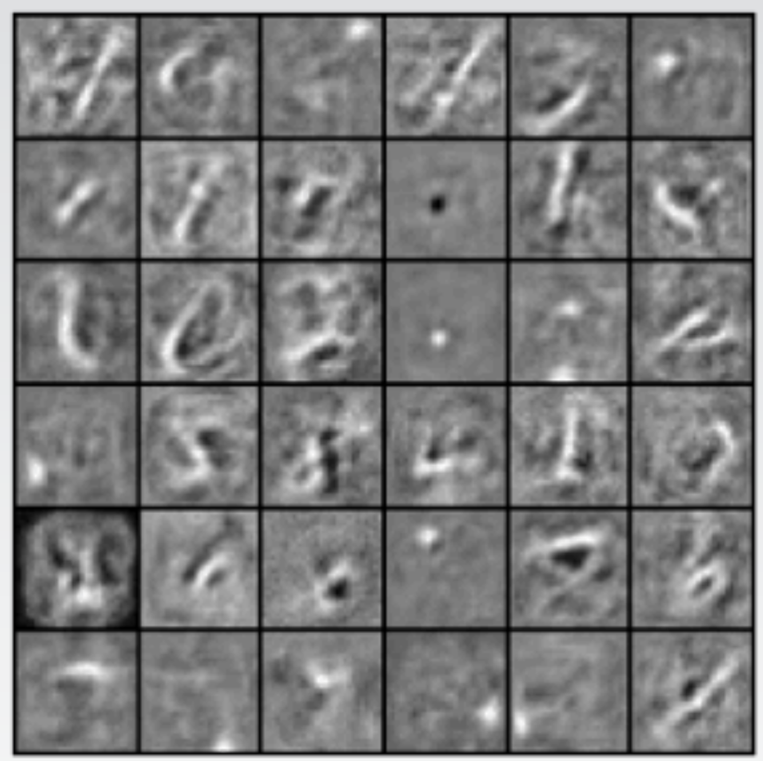
INTERPRETING THE LAYERS

- Visualization of features for digit recognition in a three-layer network
- Each panel shows a synthetic "ideal" image that maximizes the activation of a neuron in the selected layer

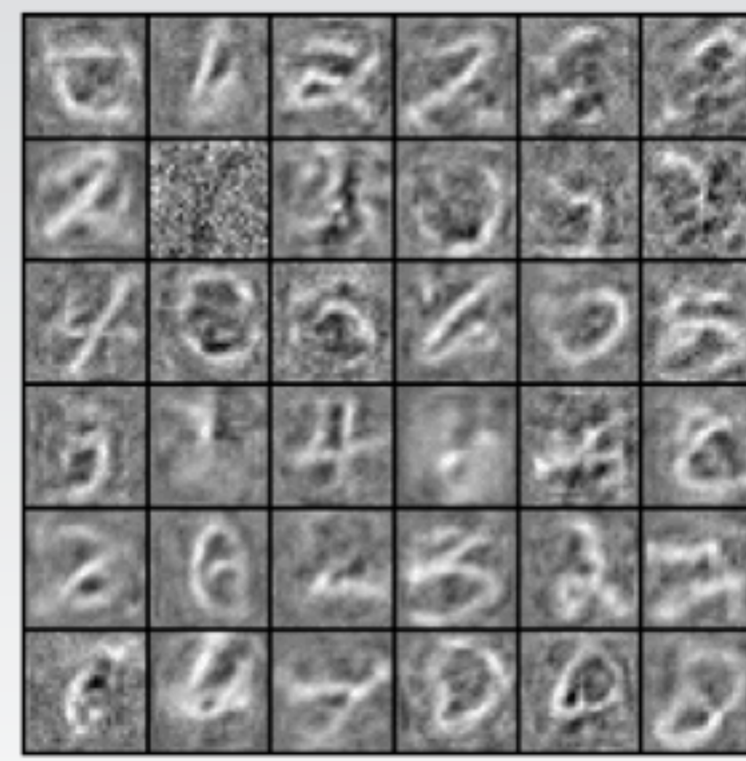
Layer 1



Layer 2



Layer 3

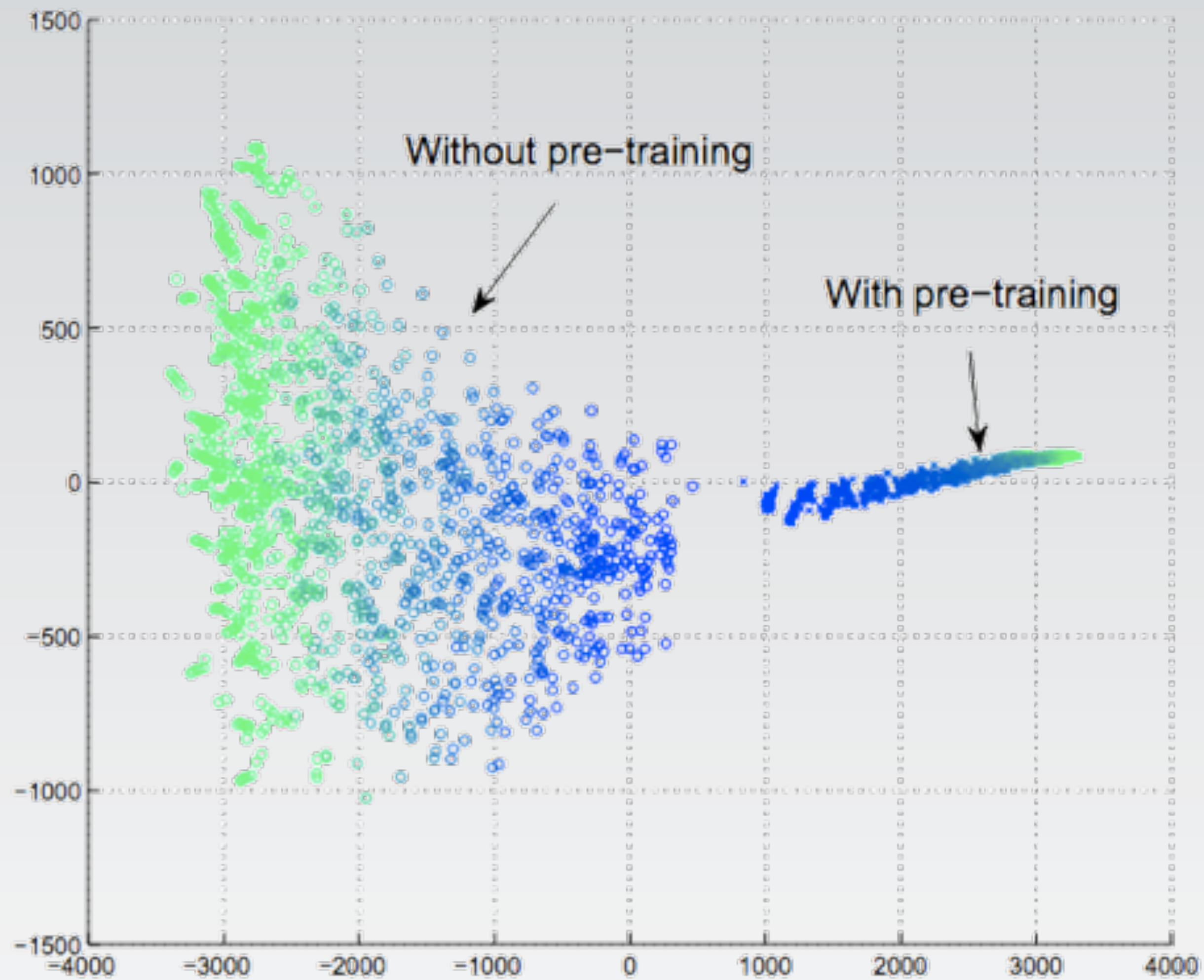


Erhan et al. (2009) *Visualizing Higher Layer Features of a Deep Network*

Strata CONFERENCE + HADOOP WORLD

Discriminative Fine-tuning

- After pre-training, apply standard back-propagation to "fine-tune"
- Different than just using back-propagation in the first place? **Yes**...

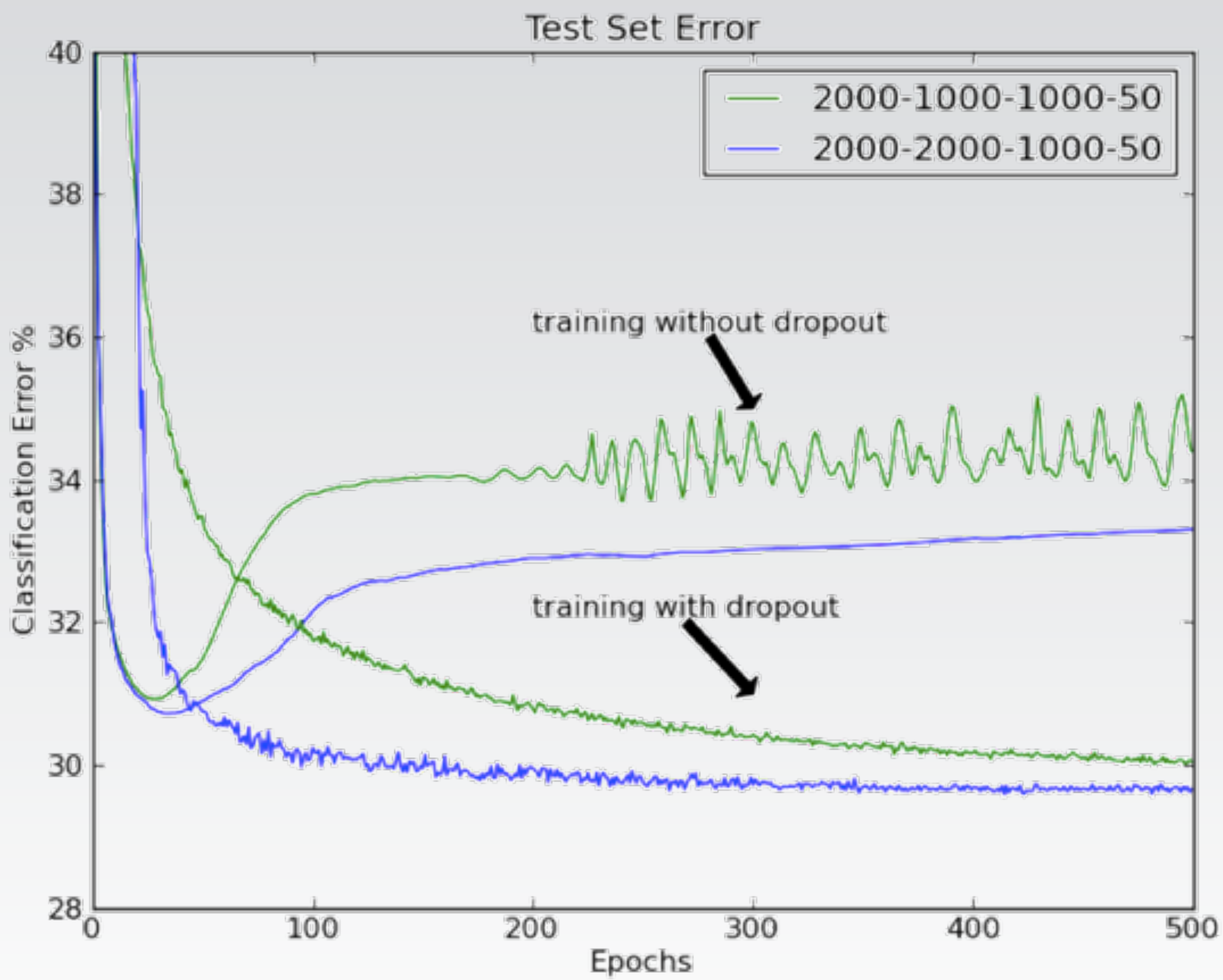


- ...and **no**. Hessian-free optimization and momentum-accelerated SGD have gotten good results *without pretraining*.

Erhan et al. (2010) *Why Does Unsupervised Pre-training Help Deep Learning?*
Martins (2010), Deep learning via Hessian-free optimization

DROPOUT

- What about overfitting?
- **Dropout**: for each training example, randomly remove half the nodes in each layer



Dahl et al (2013). *Improving deep neural networks for LVCSR using Rectified Linear Units and Dropout*

Rectified Linear Units

- Instead of a sigmoid:
  - $h_{il}(x) = 1 / (1 + \exp(-\sum h_{jl-1} * w_{ijl}))$
- ...make each neuron a rectified linear unit (ReLU):
  - $h_{il}(x) = \max(0, -\sum h_{jl-1} * w_{ijl})$

- Why do ReLUs work better than sigmoid units?
  - Lets later layers ignore irrelevant variations
  - Improves sparsity

Vinod Nair and Geoffrey E. Hinton (2010). *Rectified linear units improve restricted Boltzmann machines*

THE DEEP LEARNING RECIPE: 2006 VS. 2013

2006-2010	2013
Unsupervised pretraining via Deep Belief Nets	Unsupervised pretraining? No pretraining, or alternative algos
Fine-tuning via backpropagation	Backprop? Modified versions or use alternative algos
Sigmoid units	Rectified linear units
	Dropout

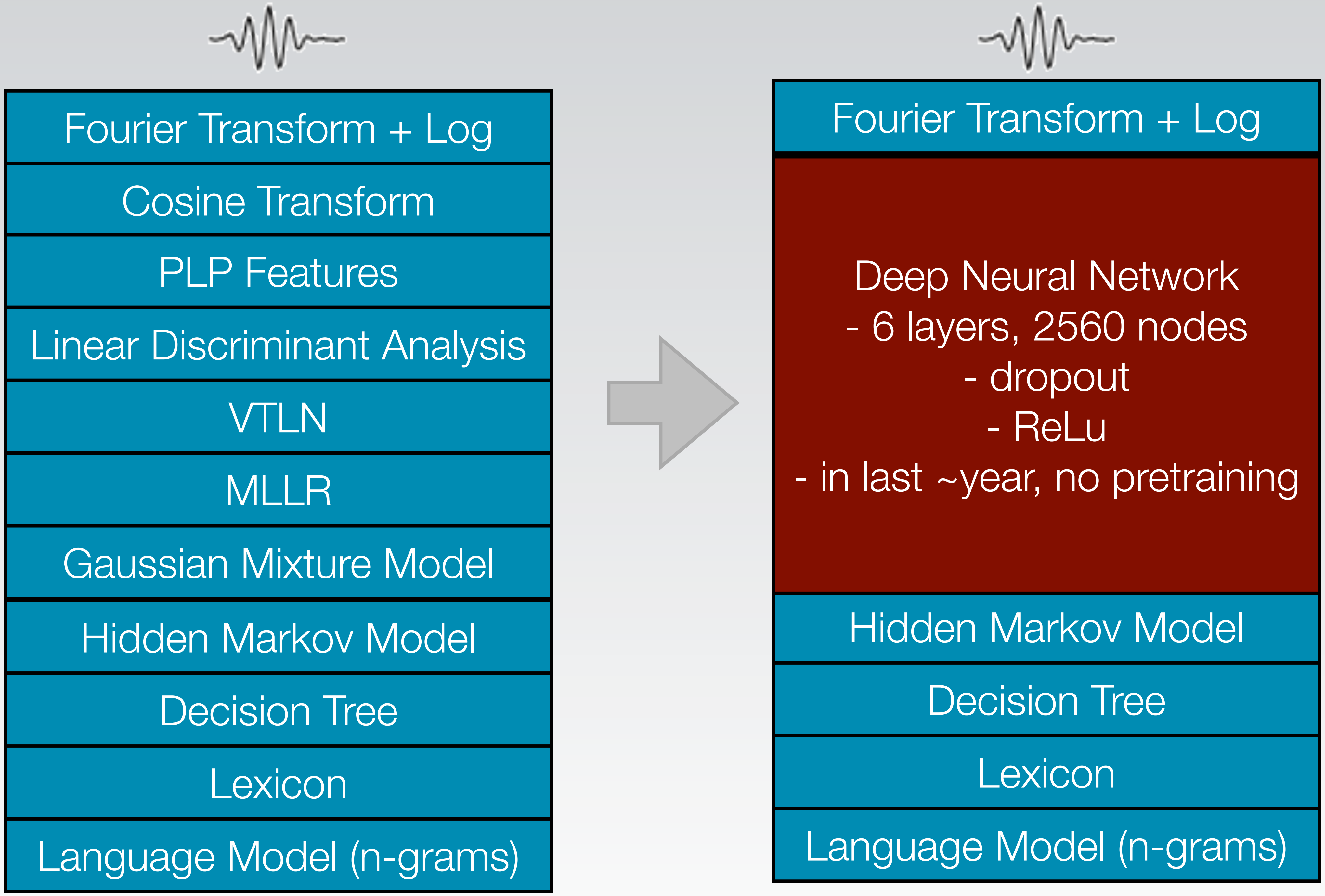
Lots of active research

Outline

- Why neural nets matter
- A Brief History of Neural Networks
  - Perceptron (1957)
  - Backpropagation (1974)
- Deep Neural Networks (2006-present)
  - Unsupervised pretraining: RBMs, greedy layer-wise training
  - Discriminative fine-tuning
  - Dropout, Rectified Linear Units
- **Case Studies**
  - Speech recognition at Google
  - Molecular Activity Prediction on Kaggle

Speech Recognition at Google

- DNNs automatically learned equivalents of hand-engineered features:



Jaitly et al (2012), *Application of pretrained deep neural networks to LVSR*

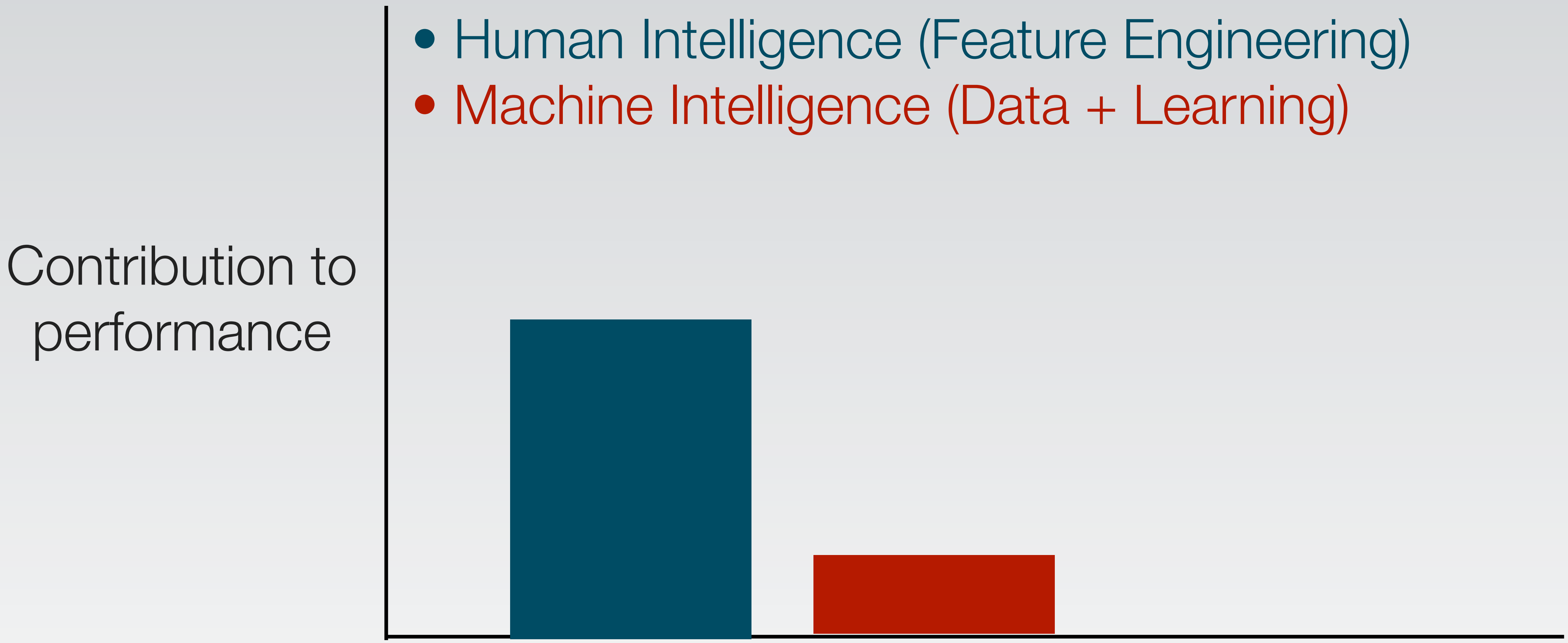
Molecular Activity Prediction

- Kaggle competition to predict interactions between different molecules
- Useful in drug discovery for identifying potential side effects
- Winning team used DNNs with no feature engineering!

> "Since our goal was to demonstrate the power of our models, we did no feature engineering and only minimal preprocessing. The only preprocessing we did was occasionally, for some models, to log-transform each individual input feature/covariate."
>
> — George Dahl, from winning team

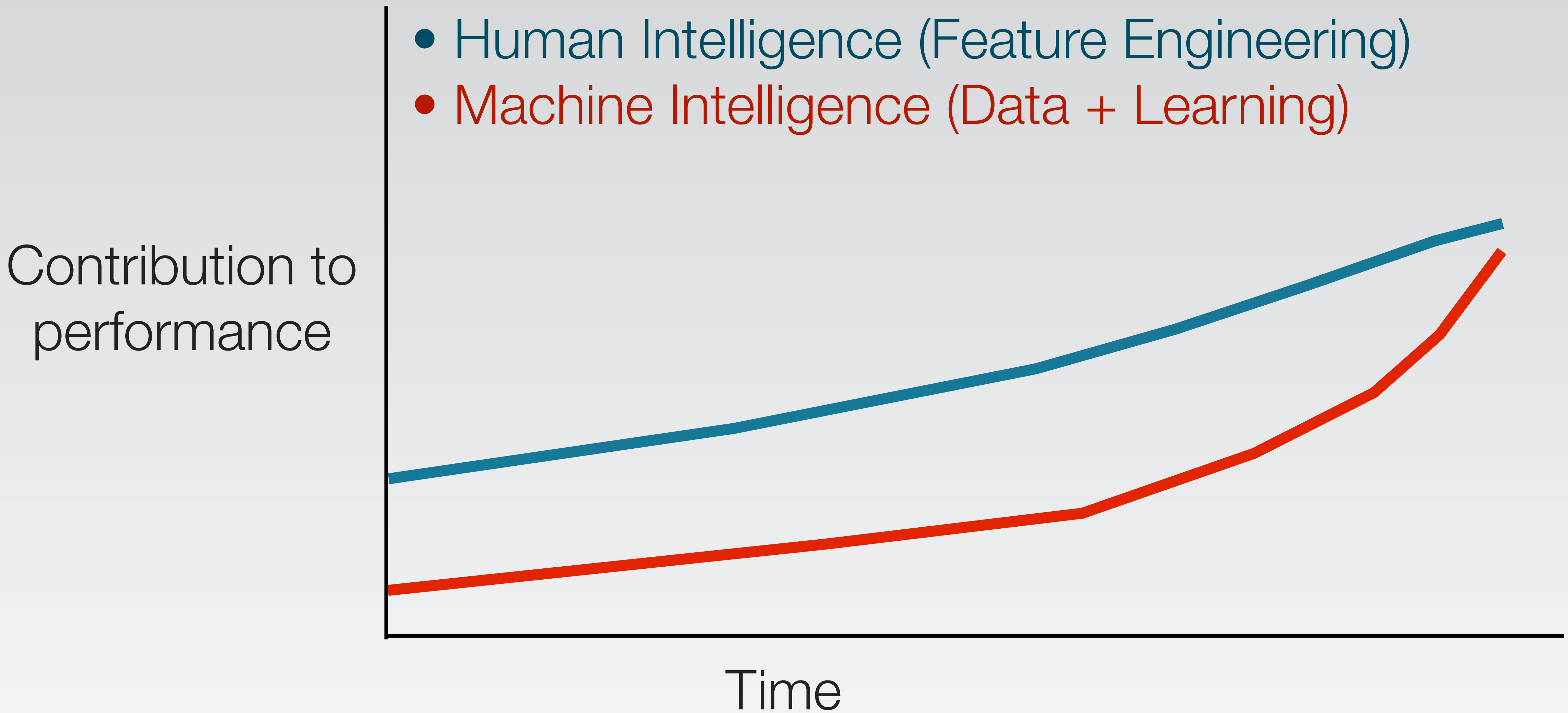
The Dream of AI

Historically, feature engineering dominated machine learning performance:



THE DREAM OF AI

Historically, feature engineering dominated machine learning performance:



In 2013, machine intelligence is finally catching up to the dream of AI. Just five decades later than expected.

DEEP LEARNING AND THE DREAM OF AI

Brandon Ballinger
@ballaballinger
Strata NY 2013

BACKUP SLIDES

THEMES

Everything old is new again. 2013's hottest idea is from 1957.

What's different in 2013 is CPU/GPU power + algorithmic changes.

The "classic" deep learning recipe adds an unsupervised pretraining step, training a stack of Restricted Boltzman Machines layer-by-layer, then using backpropagation to fine-tune.

The recipe is evolving quickly. Work on alternative optimization algorithms, unsupervised learning algorithms, new application areas.

The return of the dream of AI.

Review: The Deep Learning Recipe

```
# Unsupervised pre-training by learning a stack of RBMs
for l = 1 to L:
    while not converged:
        W_l = 0
        u = RandomTrainingExample()
        for k=1 to l-1:  # Propagate u through all layers learned so far
            u = relu(W_k u)
        RBMContrastiveDivergence(u, W_l)  # Modifies W_l
# Discriminative fine-tuning using backprop
while not converged:
    u = RandomTrainingExample()
    BackpropUpdate(u, W)  # Modifies W using gradient descent
```