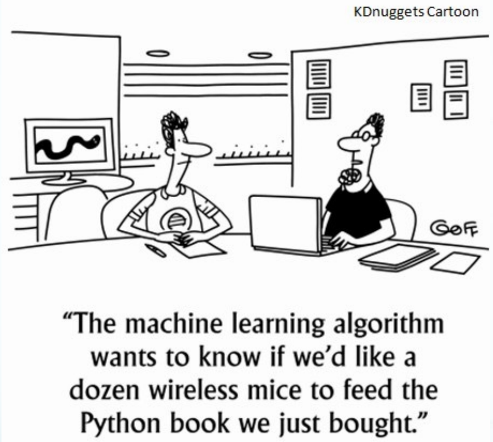


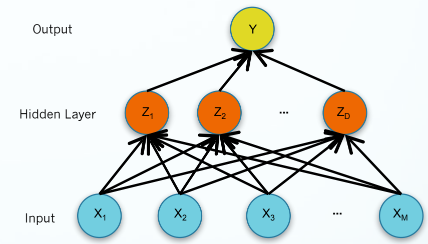
Deep Learning

Mark Dredze
Slides by Matt Gormley

Machine Learning
CS 600.475

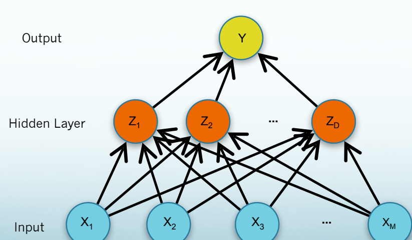
Recap of Neural Nets

- Motivation:
  - Learn features automatically
  - Capture non-linearities of data
- Two layers of binary logistic regression define a two-layer neural network
- Supervised training by SGD
  - Compute gradient with backpropagation
  - Take small steps in the direction of the gradient



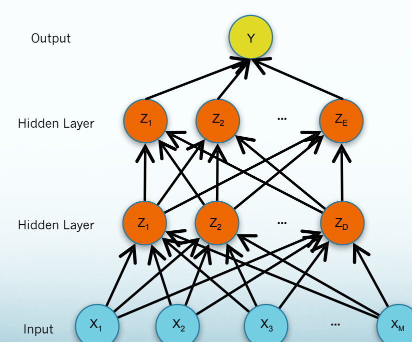
Deeper Networks

This lecture:



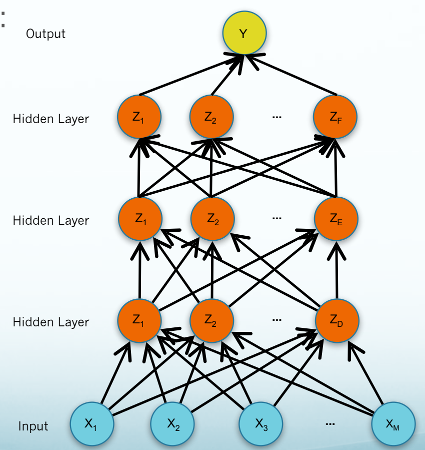
Deeper Networks

This lecture:



Deeper Networks

This lecture: Making the neural networks deeper



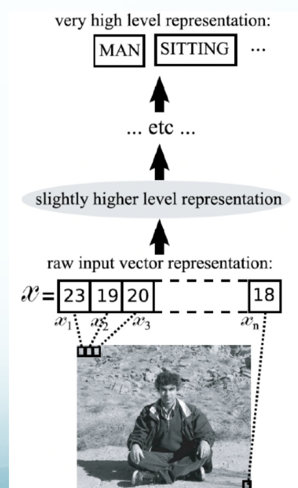
Motivation: Why go Deep?

- 2-layer Neural Nets are already universal function approximators…
  - But deep architectures can be representationally efficient
  - Fewer computational units for the same function
- 2-layer Neural Nets can represent non-linear combinations of the input features
  - But deep representations might allow for a hierarchy
  - Allows non-local generalizations
- Deep Nets: Multiple levels of latent variables allow combinatorial sharing of statistical strength
- 2-layer Neural Nets work well
  - But deep representations have been shown to work even better (vision, audio, NLP, etc.)!

Slide adapted from Honglak Lee (NIPS 2010)

The Promise of Deep Architectures

- Transform input image into higher levels of representation:
  - edges, local shapes, object parts, etc.
- We don't know the "right" levels of abstraction
- So let the model figure it out!

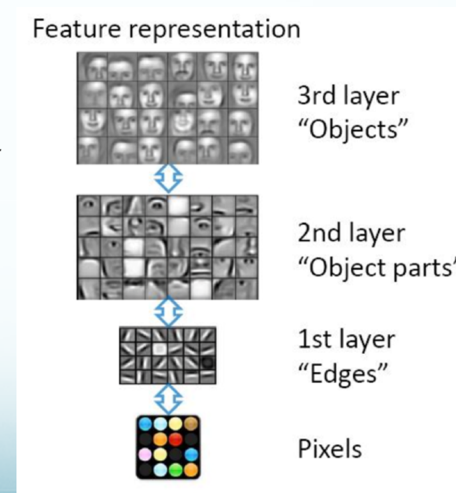


Example from Bengio (2009)

Different Levels of Abstraction

Face Recognition:

- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions



Example from Honglak Lee (NIPS 2010)

Deep Network Training

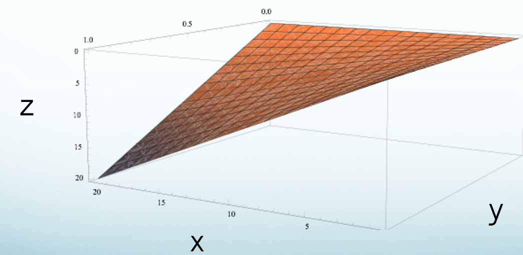
(that doesn't always work)

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- Requires labeled data
  - Usually have gobs of unlabeled data
  - But can't learn from it
- What goes wrong?
  - A. Gets stuck in local optima
    - Nonconvex objective
    - Usually start at a random (bad) point in parameter space
  - B. Gradient is progressively getting more dilute
    - "Vanishing gradients"

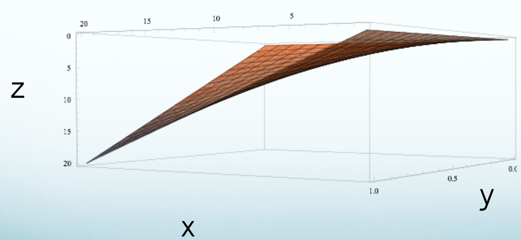
The problem: *Nonconvexity*

- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:



The problem: *Nonconvexity*

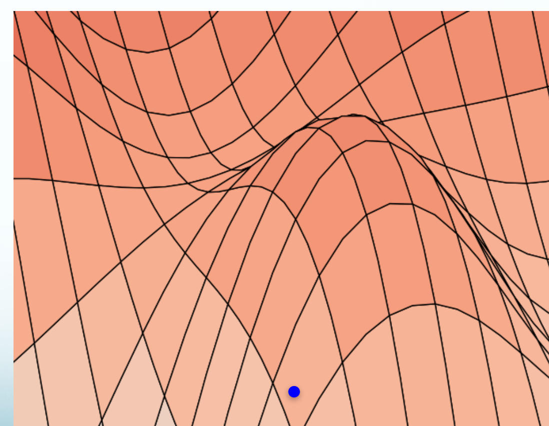
- Where does the nonconvexity come from?
- Even a simple quadratic z = xy objective is nonconvex:



The problem: *Nonconvexity*

Stochastic Gradient Descent…

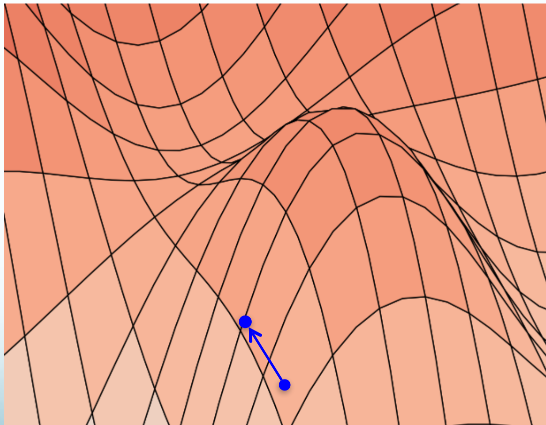
…climbs to the top of the nearest hill…



The problem: *Nonconvexity*

Stochastic Gradient Descent...

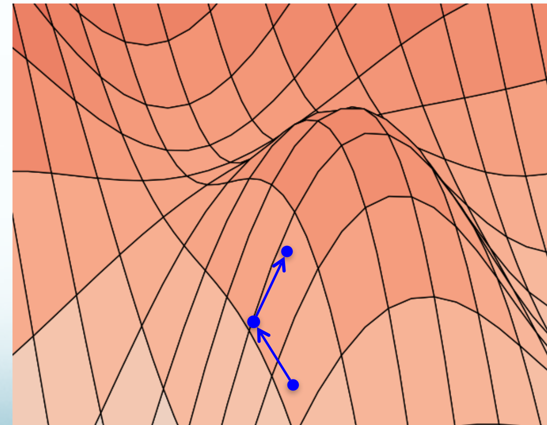
...climbs to the top of the nearest hill...



The problem: *Nonconvexity*

Stochastic Gradient Descent...

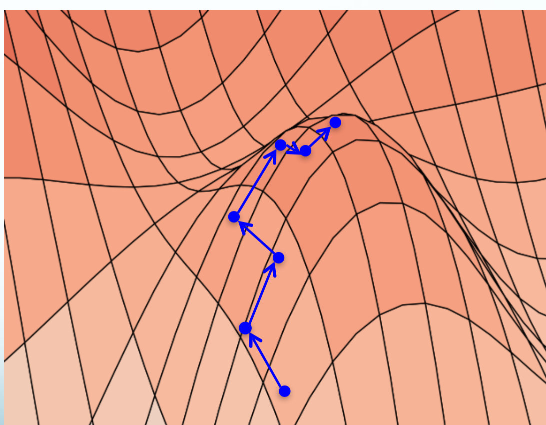
...climbs to the top of the nearest hill...



The problem: *Nonconvexity*

Stochastic Gradient Descent...

...climbs to the top of the nearest hill...

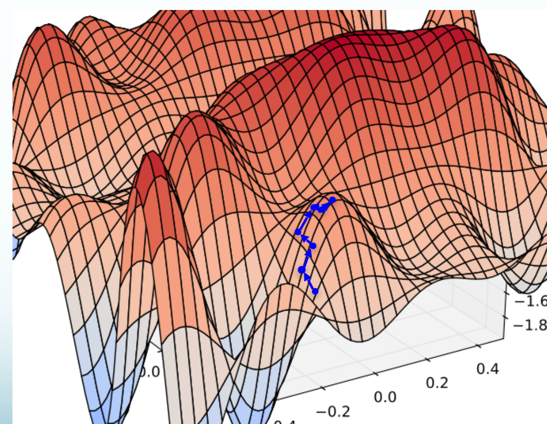


The problem: *Nonconvexity*

Stochastic Gradient Descent...

...climbs to the top of the nearest hill...

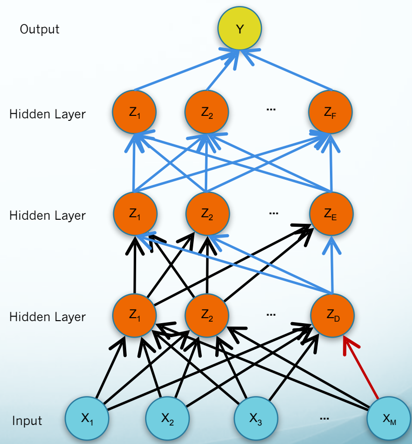
...which might not lead to the top of the mountain



The problem: *Vanishing Gradients*

The gradient for an edge at the base of the network depends on the gradients of many edges above it

The chain rule multiplies many of these gradients (adjoints) together



Deep Network Training

(that doesn't always work)

- **Idea #1: (Just like a shallow network)**
  - Compute the supervised gradient by backpropagation.
  - Take small steps in the direction of the gradient (SGD)

- Requires labeled data
  - Usually have gobs of unlabeled data
  - But can't learn from it
- What goes wrong?
  - A. Gets stuck in local optima
    - Nonconvex objective
    - Usually start at a random (bad) point in parameter space
  - B. Gradient is progressively getting more dilute
    - "Vanishing gradients"

Deep Network Training

(that still doesn't work)

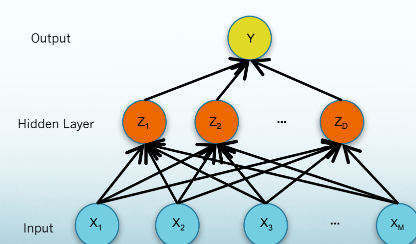
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**

1. Supervised Pre-training
   - Use **labeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following "Idea #1"
   - Refine the features by backpropagation so that they become tuned to the end-task

Deep Network Training

(that still doesn't work)

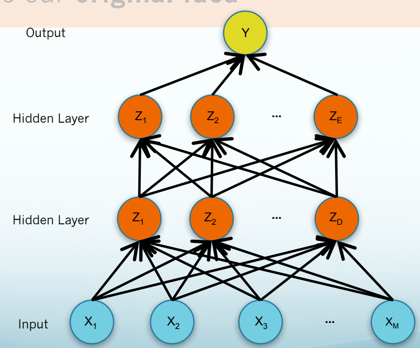
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



Deep Network Training

(that still doesn't work)

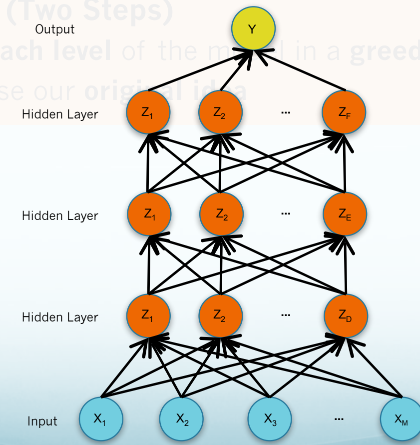
- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



Deep Network Training

(that still doesn't work)

- **Idea #2: (Two Steps)**
  - **Train each level** of the model in a **greedy** way
  - Then use our **original idea**



Deep Network Training

(that actually works!!)

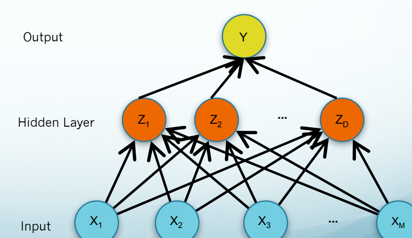
- **Idea #3: (Two Steps)**
  - Use our original idea, but **pick a better starting point**
  - **Train each level** of the model in a **greedy** way

1. Unsupervised Pre-training
   - Use **unlabeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following "Idea #1"
   - Refine the features by backpropagation so that they become tuned to the end-task

The solution:
Unsupervised pre-training

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

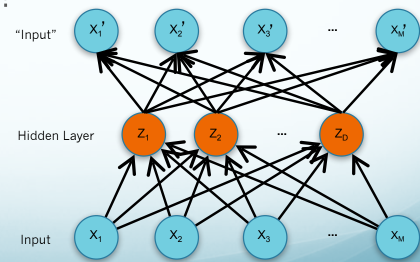


The solution:
Unsupervised pre-training

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

This topology defines an Auto-encoder.



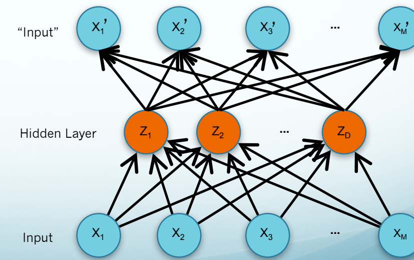
Auto-Encoders

Key idea: Encourage z to give small reconstruction error:

- x' is the *reconstruction* of x
- Loss = || x – DECODER(ENCODER(x)) ||^2
- Train with the same backpropagation algorithm for 2-layer Neural Networks with x_m as both input and output.

DECODER: x' = h(W'z)

ENCODER: z = h(Wx)

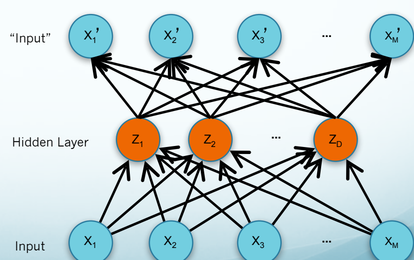


Slide adapted from Raman Arora

The solution:
Unsupervised pre-training

Unsupervised pre-training

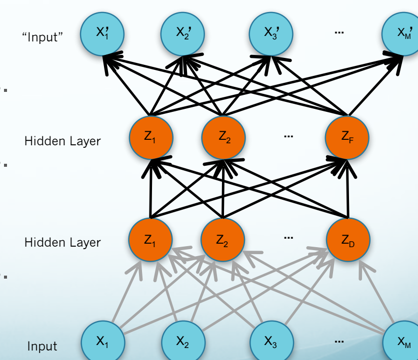
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution:
Unsupervised pre-training

Unsupervised pre-training

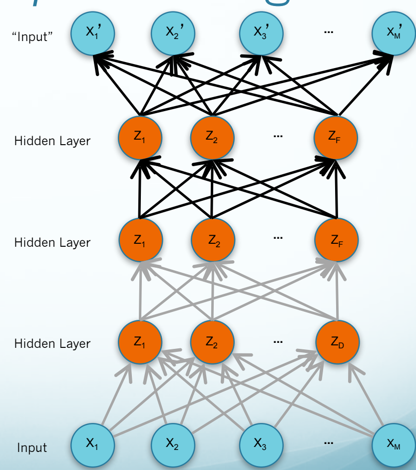
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution:
Unsupervised pre-training

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.

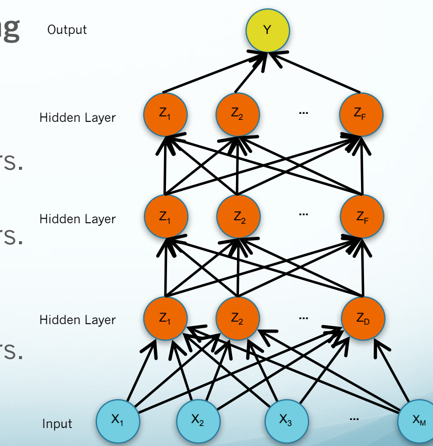


The solution:
Unsupervised pre-training

Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.

Supervised fine-tuning
Backprop and update all parameters



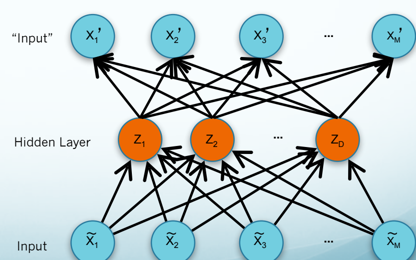
Denoising Auto-encoders

Key idea: Encourage z to give small reconstruction error

- x' is the *reconstruction* of $\tilde{x}$ = x+noise
- Loss = || x – DECODER(ENCODER(x + noise)) ||^2
- Train with the same backpropagation algorithm

DECODER: x' = h(W'z)

ENCODER: z = h(W$\tilde{x}$)
where $\tilde{x}$ = x+noise



Slide adapted from Raman Arora

Deep Network Training

- **Idea #1:**
  1. Supervised fine-tuning only
- **Idea #2:**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning
- **Idea #3:**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

Deep Network Training

Results from Bengio et al. (2006)

Percent error (lower is better) on MNIST digit classification task

		Experiment 2			Experiment 3		
		train.	valid.	test	train.	valid.	test
Idea #3:	Deep net, auto-associator pre-training	0%	1.4%	1.4%	0%	1.4%	1.6%
Idea #2:	Deep net, supervised pre-training	0%	1.7%	2.0%	0%	1.8%	1.9%
Idea #1:	Deep net, no pre-training	.004%	2.1%	2.4%	.59%	2.1%	2.2%
	Shallow net, no pre-training	.004%	1.8%	1.9%	3.6%	4.7%	5.0%

Demo of Neural Network on MNIST digit classification:
http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Is layer-wise pre-training always necessary?

Answer in 2006: Yes!
Answer in 2014: No!

1. If initialization is done well by design (e.g. sparse connections and convolutional nets), maybe won't have vanishing gradient problem
2. If you have an extremely large datasets, maybe won't overfit. (But maybe that also means you want an ever deeper net)
3. New architectures are emerging:
   - Stacked SVM's with random projections [Vinyals et al., 2012]
   - Sum-Product Networks [Poon and Domingos, 2011]

Slide adapted from Raman Arora

Deep Learning

- Goal: learn features at different levels of abstraction
- Training can be tricky due to...
  - Nonconvexity
  - Vanishing gradients
- Unsupervised layer-wise pre-training can help with both!