Artificial Neural Networks:
Tricks of the Trade and "Neural Smithing"

Alexander G. Ororbia II

Introduction to Machine Learning

CSCI-335

4/20/2026 and 4/22/2026

***So*…now we can train ANNs with backprop, so we are done, right? Machine learning is solved, class finally dismissed?**

So now we can train A… …prop,

so w… …t?

Mac… …solved, class

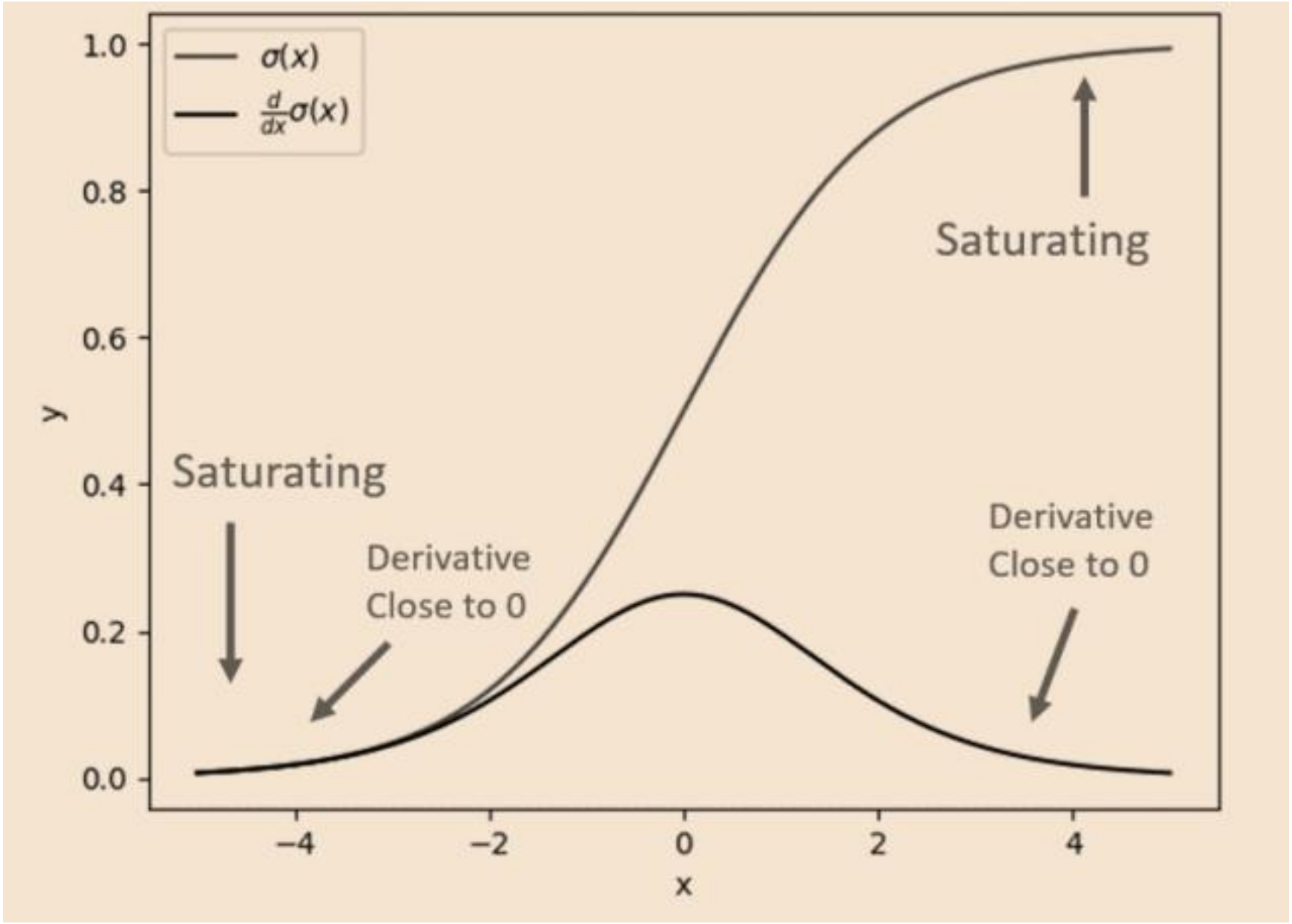
…y dismissed?

WRONG!

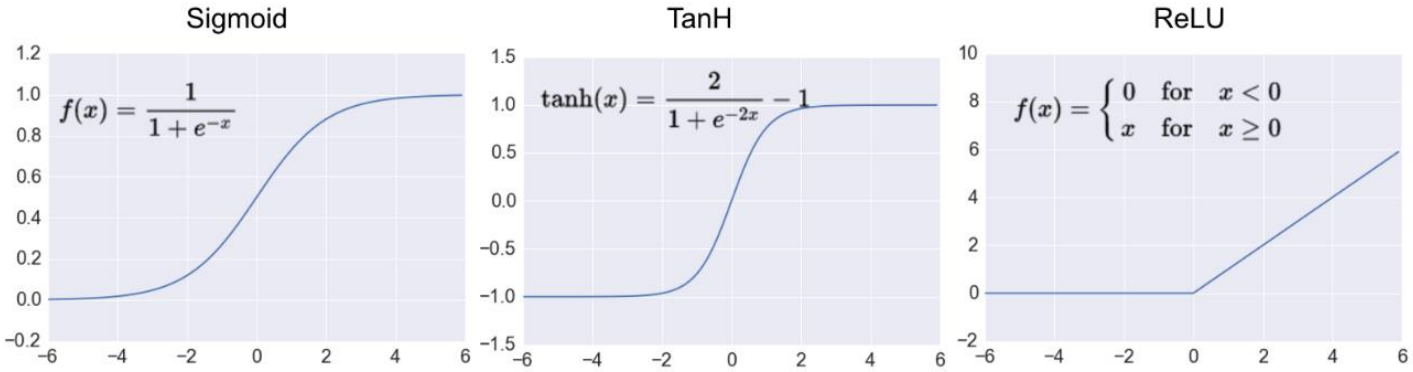
The Vanishing Gradient Problem

- Solving credit assignment problem with back-propagation (backprop) very difficult
  - Difficult to know how much importance to accord to remote inputs (Bengio et al., 1994)
  - Information passed through chain of multiplications back through network – ***vanishing gradients***
    - Any value slightly less than 1 in Hadamard product, derivative signal quickly shrinks to useless values (near zero)
  - Learning long-term dependencies in temporal sequences becomes near impossible
- Complementary problem: ***exploding gradients***
  - Any value greater than 1 in Hadamard product, derivative signal increases dramatically (numerical overflow)

The Logistic Sigmoid and Its Derivative:

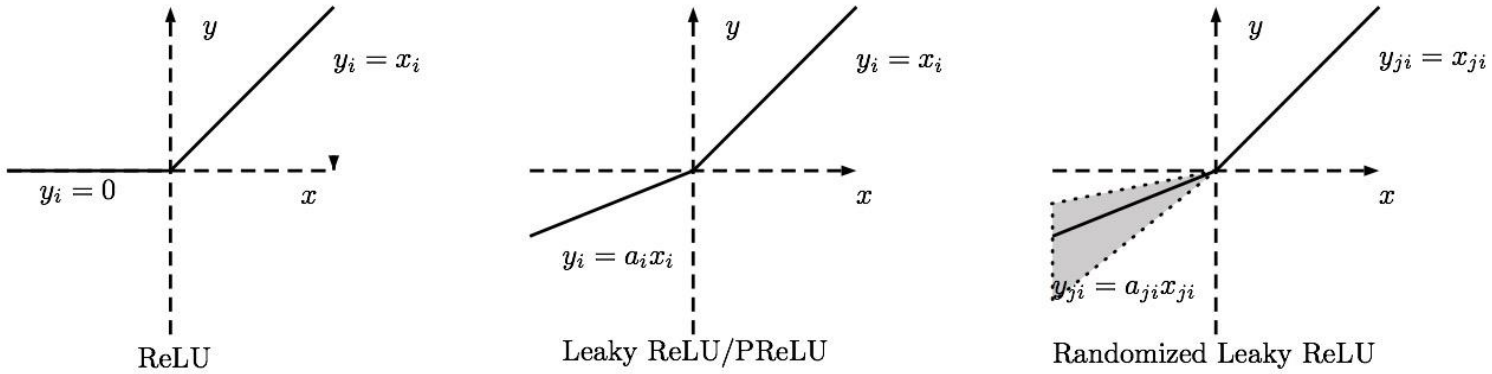


Using Other Activation Functions



Linear Rectified Unit (Relu)

Not smooth / not differentiable everywhere, ***Benefit***: Hard sparsity
Issues: Dead units, explosive weight updates
Parametric Relu (PRelu), leaky Relu: "Learn" slope of activation function

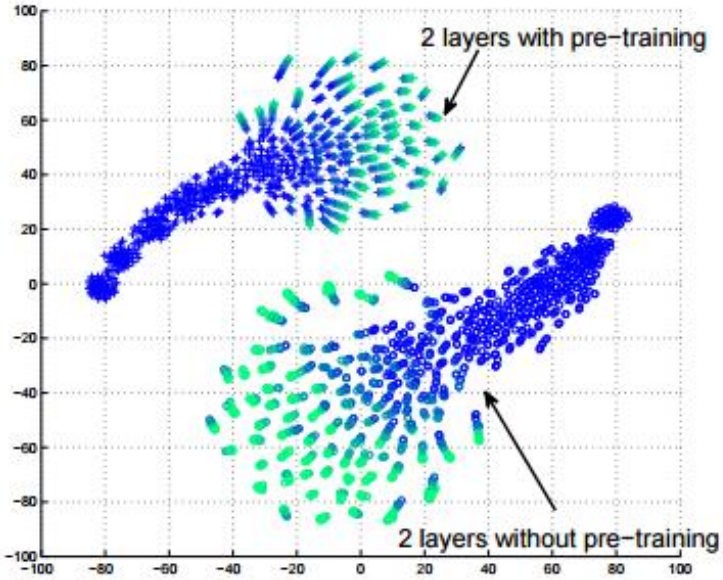


How to make those gradients work for you!

THE IMPORTANCE OF STARTING OUT RIGHT: INITIALIZATION

Why do we care how parameters are initialized?

- Initialization affects final performance
  - Will put closer to some spots in function space and farther from others



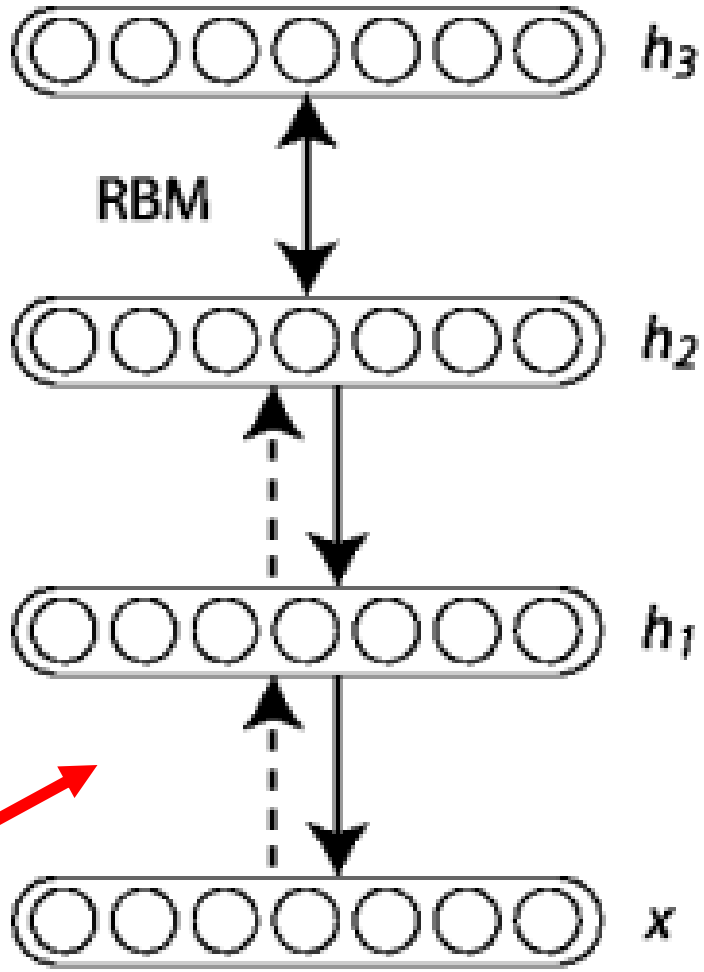
- Where we end up in function space will often correlate w/ our error performance

Figure 5: 2D visualizations with tSNE of the functions represented by 50 networks with and 50 networks without pre-training, as supervised training proceeds over MNIST. See Section 6.3 for an explanation. Color from dark blue to cyan and red indicates a progression in training iterations (training is longer without pre-training). The plot shows models with 2 hidden layers but results are similar with other depths.

"Why Does Unsupervised Pre-training Help Deep Learning?", Erhan et al. 2010 http://jmlr.org/papers/volume11/erhan10a/erhan10a.pdf

Pre-Training: Learning Your Initialization

- General idea:
  - Train another model, e.g., ***deep belief network*** →
  - Dump its parameters into the one you care about
  - Fine-tune final model



- Unsupervised generative models were largely useful for this

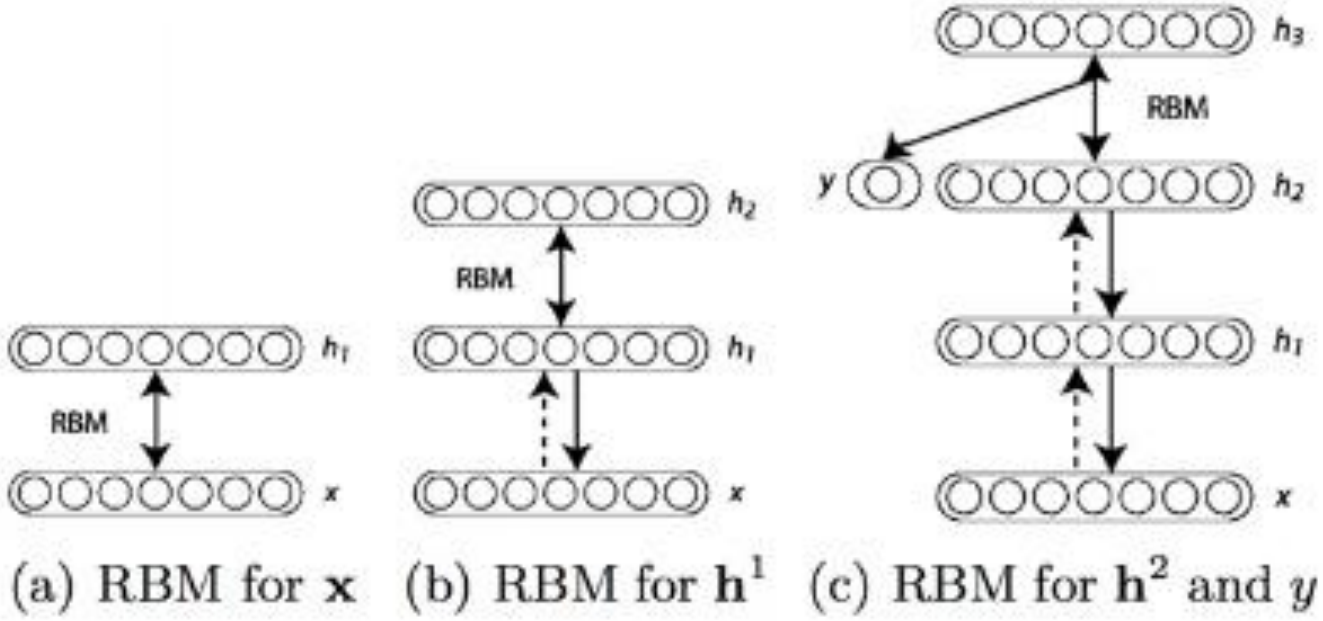
This is used something called a harmonium, a small non-backprop trained generative network!

$$P(x, h^1, \ldots, h^\ell) = \left(\prod_{k=0}^{\ell-2} P(h^k | h^{k+1}) \right) P(h^{\ell-1}, h^\ell)$$

With: $P(h^{k-1} | h^k)$, $P(h^{\ell-1}, h^\ell)$, $x = h^0$

Pretraining: Stacked (Neural) Models

- Iterative pre-training construction of **Deep Belief Network (DBN)** (Hinton et al., 2006)



(a) RBM for $\mathbf{x}$ (b) RBM for $\mathbf{h}^1$ (c) RBM for $\mathbf{h}^2$ and y

from: Larochelle et al. (2007). An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation.

White Board Time!

A Simple (Denoising) Autoencoder

Research Efforts in Pre-Training

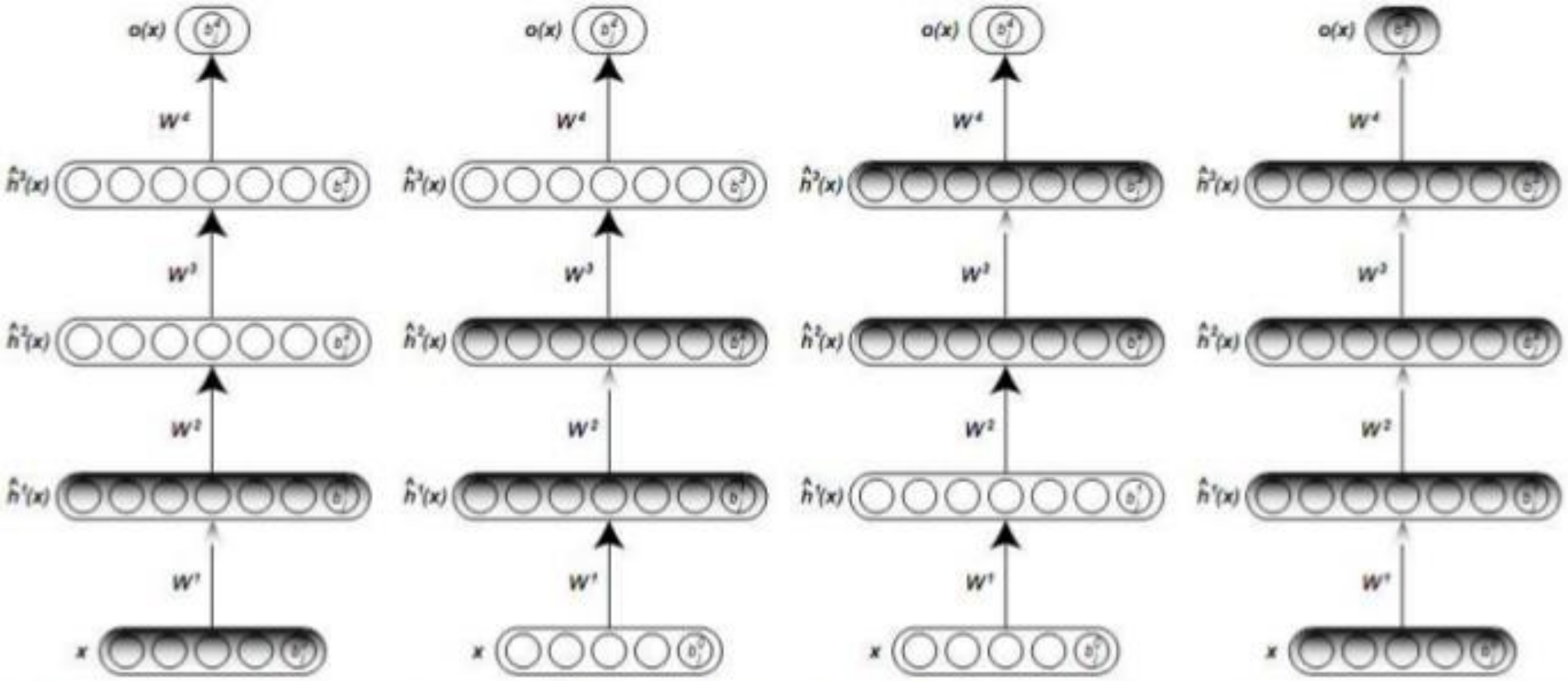
Pre-training works! (Erhan et al., 2010), but…

- 2-stage learning (Bengio et al., 2007)
  - Step 1: (Greedy) unsupervised pre-training
    - Deep Belief Networks: Contrastive Divergence (CD-k) or SML/PCD
    - ***Stacked Denoising Autoencoders***: Back-propagation w/ cross-entropy loss
  - Step 2: Supervised fine-tuning
    - 1) Toss old model, dump parameters into MLP
    - 2) (Gentle) back-propagation fine-tuning

So…

- Hybrid, single-stage training (Larochelle et al., 2012; Ororbia et al., 2015)
  - Why not learn a generative & discriminative model at same time?

Unsupervised greedy layer-wise training procedure.



(a) First hidden layer pre-training

(b) Second hidden layer pre-training

(c) Third hidden layer pre-training

(d) Fine-tuning of whole network

Could Pretrain…or…Just Use Smarter Random Initializations?

- Classical simple approaches
  - Sample from ~*U(-a, a)*, where is *a* small scalar
  - Sample from ~*N(0, a)*, where is *a* small standard deviation
- Fan-in-Fan-out (number of inputs, numberof outputs)
  - Calibrate by variances of neuronal activities
- Simple distributional schemes
  - Fan-in/Fan-out Uniform
  - Fan-in/Fan-out Gaussian (good for ReLU activations)
- Orthogonal Initialization
  - Use Singular Value Decomposition (***SVD***) to find initial weights
- Identity Initialization / Constraint (for RNNs)
  - Does not always work unless constraint is enforced

OR…Just Wait Longer?!

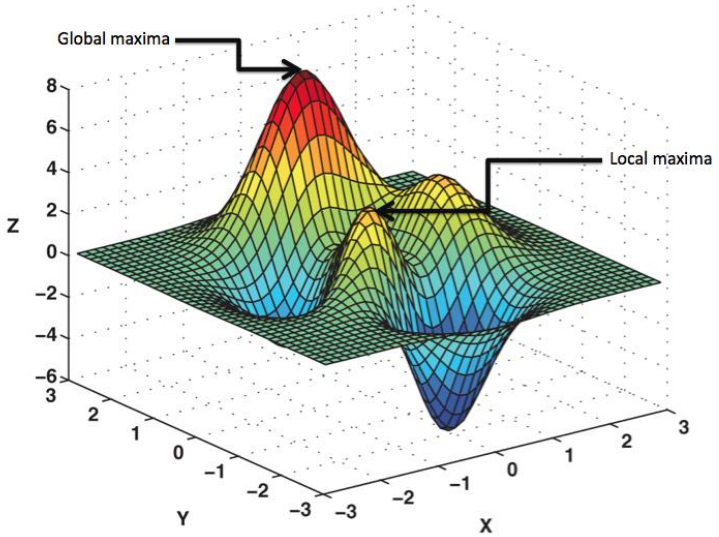
- Even with poor initialization, just wait a really long time….
- Patience + really good hardware
- So…one answer = ***more hardware***

How to make those gradients work for you!

PARAMETER OPTIMIZATION

Optimization Schemes

- Steepest (mini-batch) gradient descent
  - Use an estimator (i.e., backprop) to get gradient,
    then update parameters
  - Stochastic gradient descent (SGD)
- Alternative optimizers = shiny toys to make learning even faster



Steepest Gradient Descent

- Simplest update rule
- Combine with early stopping
  - ***Early stopping*** = tracking loss/error on validation set
  - A simple form of regularization (weights will be smaller)

```
# Vanilla update
x += - learning_rate * dx
```

Simple Momentum

- Maintains rolling average of previous gradients
  - Smooths out descent of minimization algorithm
  - Prevent “bouncing around” on loss/error surface

- *Many variants*: momentum, Nesterov’s Accelerated Gradient (NAG), etc.

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

QUESTIONS?