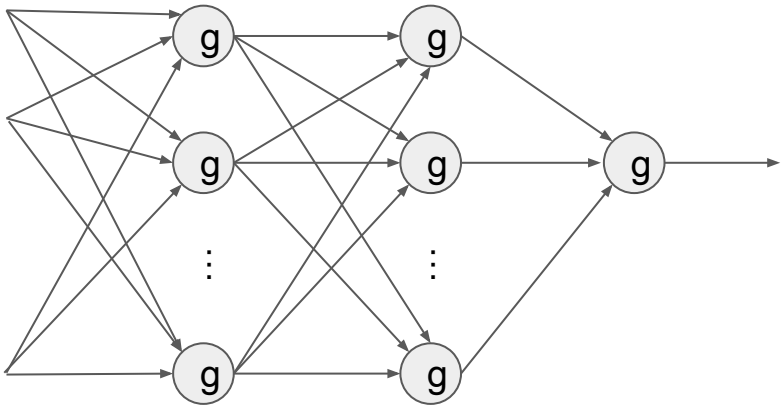
Deep Learning

CS 480
Intro to Artificial Intelligence

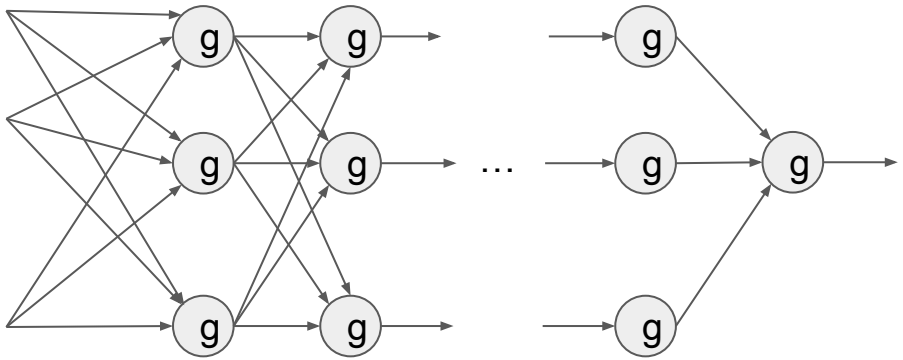
What is Deep Learning?

1. Neural Networks with depth > 2
2. A collection of tricks for handling problems unique to high-depth networks
3. A theory about why adding layers should improve performance

Width



Depth

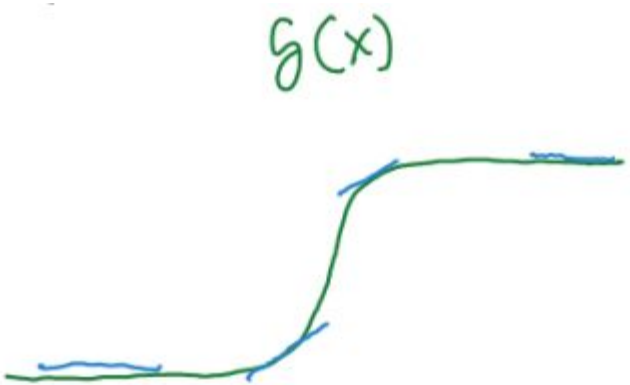


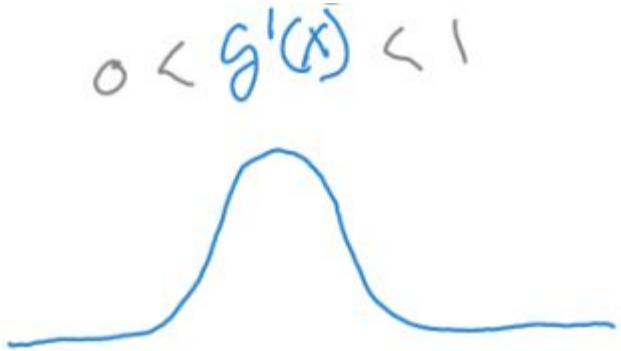
Vanishing gradient

$$w_{j,k} \leftarrow w_{j,k} + \alpha \cdot a_j \cdot \Delta_k$$

$$\Delta_j = g'(in_j) \cdot \sum_k (w_{j,k} \cdot \Delta_k)$$

The derivative of the logistic function is small in the “tails”

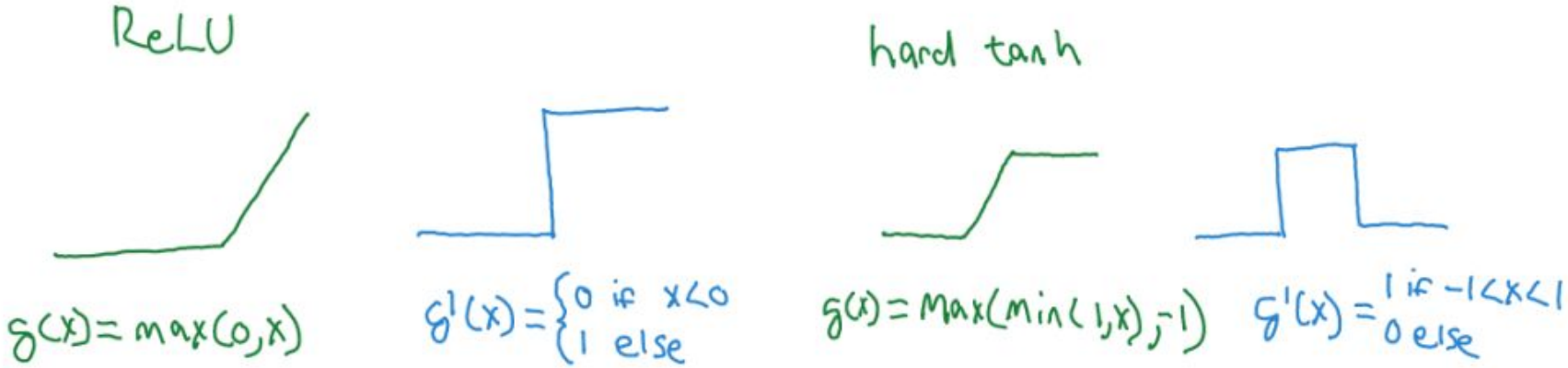




Multiplying the error by g’ shrinks the signal at each layer, so weight updates become very small in deep networks

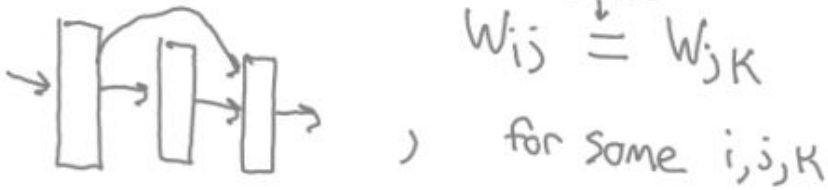
Preserving the gradient

Newer threshold functions avoid the vanishing gradient problem

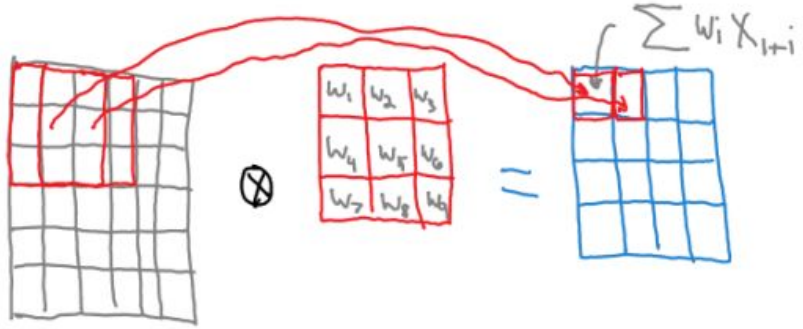


New problem: Exploding Gradient! Other architecture solutions

- skip connections
- shared weights



Convolutional NNs



Input is an image

- Fully Connected architecture: WxH weights per neuron in the **first layer**!
- Convolutional architecture:
  - Only combine **nearby** pixels (k-by-k filter, or “kernel”)
  - Sweep filter across image to create outputs for next layer
- Has the effect of forcing most of the weights to be the same
- Spatial relationships preserved
- Other special types of layers: pooling (reduce dimensionality), fully connected (standard NN)

Not the same “kernel” as in SVMs

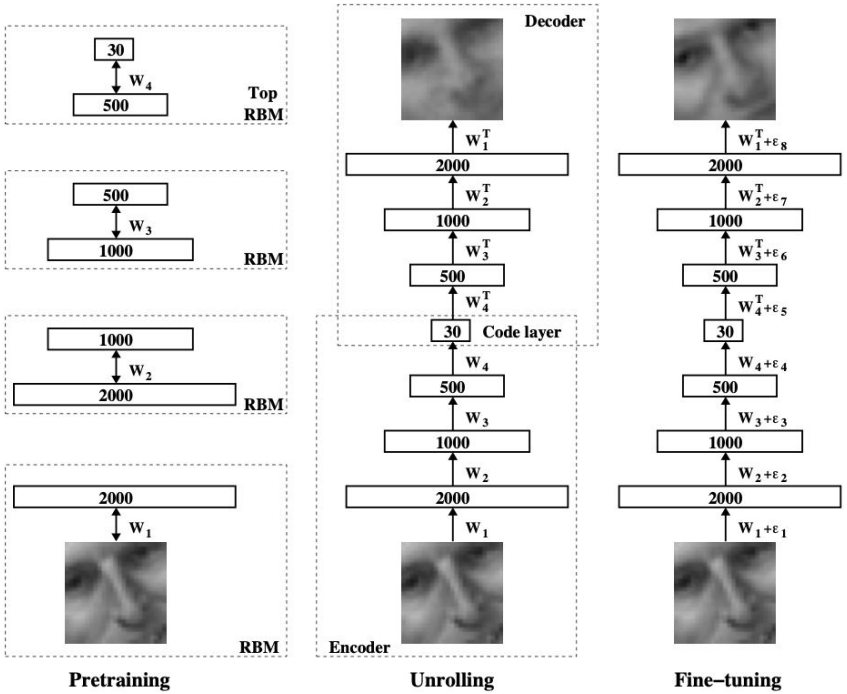
Auto-encoders

An unsupervised method

- Input passes through network, reconstructed at the last layer
- Force the network to learn a low dimensional representation by having a narrow “waist”

After unsupervised training,

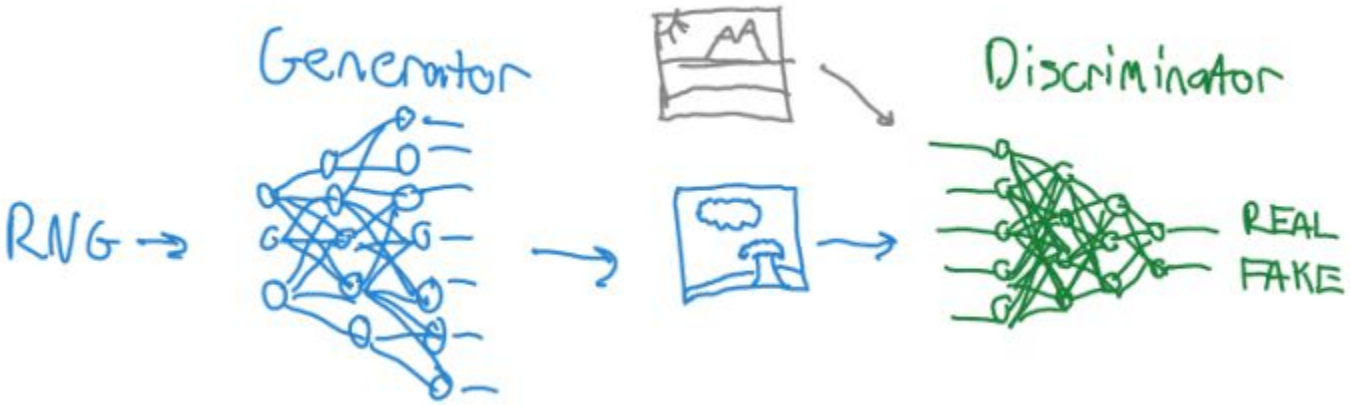
- Use as a pre-training
- Use as a dimensionality reduction



Generative Adversarial Networks

DNNs as generative models?

1. Train one network to generate images
2. Train a separate network to classify images as “real” or “fake”
3. Feed the output of (1) into (2)



GAN challenges

1. Keeping discriminator (D) and generator (G) at roughly the same level of performance

It's typically easier to get good performance for (D) than (G). If (D) outpaces (G), there's little information in the loss signal (needle-in-a-haystack objective function)

2. Keeping (G) from outputting a single image (**mode collapse**)

The criteria for "diversity" in outputs from (G) is hard to quantify

3. Keeping (G) from outputting "blurry" images

Different architectures handle all three of these things in different ways

Recurrent Networks

What if we allow networks to have cycles?

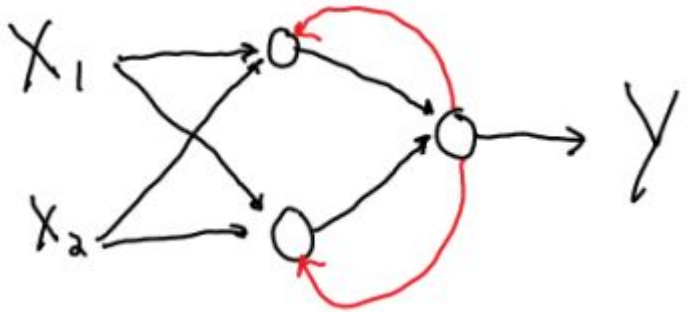
- Neurons now have state!

Can be useful for modeling time-series data

- Text/speech (NLP)
- Activity generation/detection

Popular models:

- Long short-term memory (LSTM)
- Hopfield networks
- Echo state networks



Training RNNs?

Having state makes this a **dynamical system**: much more complex to train/analyze!

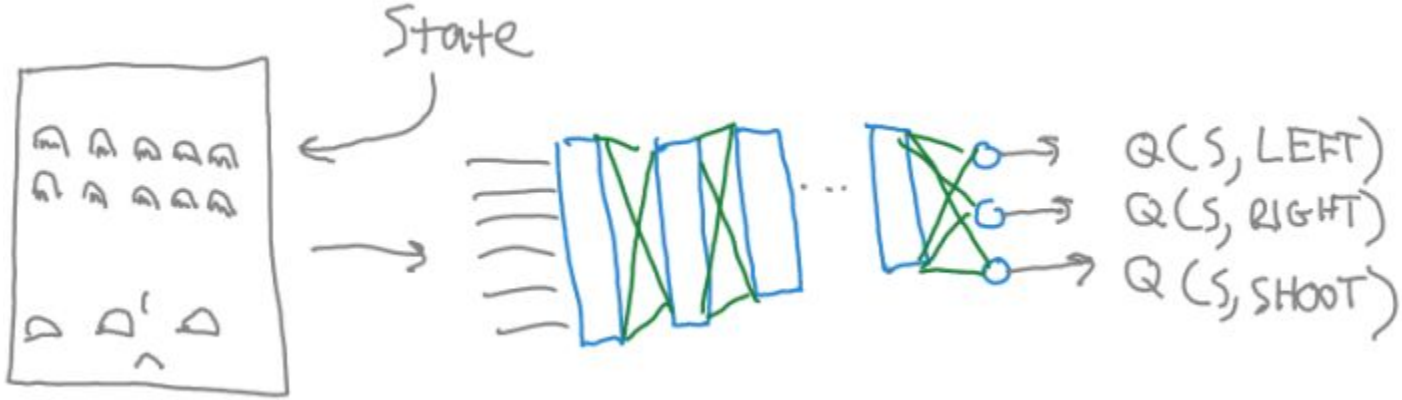
One option: “unroll” the recurrent connections, then run backprop

Backpropagation **t**hrough **t**ime (**BPTT**)

Deep Reinforcement Learning (DQN and AlphaGo)

Recall from MDPs/RL: if we knew the utility function (U(s) or Q(s,a)) we can compute policy

Deep Q-Learning: Train a network to estimate Q(s,a)



Deep RL notes

How do we train this? What's the loss? **Bellman**

$$\mathcal{L} = (r + \alpha \cdot \max_{a'} Q(s', a') - Q(s, a))^2$$

Practical considerations

- Stable updates
- Local minima

Similar approach for 2-player games (Go, DotA)

- Train by self-play (TD-gammon, '92!)

Why are deep networks so successful? (1)

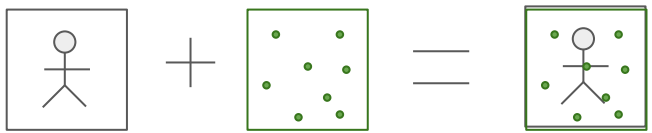
Deep networks should have much more difficulty with overfitting than they do in practice

- Thousands/millions of parameters, a lot of research on good regularization practices
  - VC-dimension in particular predicts deep networks should require much more data to achieve the results we regularly see in practice. Active research area!
- Stochastic gradient descent may be good at avoiding local optima
  - Since SGD uses “noisy” gradient estimates and a learning decay schedule, it may “bounce” out of local optima early in the run. Alternatively, SGD may not suffer from “plateaus”. Active research area!
- Deep network error landscapes are not well understood
  - Extremely high dimensional space, but with some pretty strong restrictions. Active research area!

Why are deep networks so successful? (2)

A common argument: “Hidden layers learn **semantically useful** representations”

- Early empirical work in Computer Vision showed that hidden layers were learning what looked like well known low level features (Gabor filters)
  - No reason this should happen! Nothing in the math “makes” this occur.
- Unfortunately, this frequently **doesn’t** happen in other domains
  - Even some CV problems don’t always have easily interpretable hidden layers
- There are strong biological arguments that the **human brain** doesn’t work this way
  - Grandmother neuron
  - Adversarial noise



Network output “pedestrian” “noise” “Green light”

Why are deep networks so successful? (3)

Why does depth improve performance?

- Argument 1: Increasing depth allows a network to fit a function with exponentially fewer neurons
  - Proof shows the **existence** of **some** functions with this property, no reason to believe these are “realistic” problems
- Argument 2: hidden layers learn to “compress” the input into only features that matter for generalization (Information Bottleneck)
  - Backprop doesn’t explicitly do **anything** to improve generalization performance, or require the data be “compressed”
  - Specific counter examples recently found
- New results coming out every day. Active area of research!

Summary and preview

Wrapping up

- Deep Learning: networks with depth > 2
- Threshold functions matter: Vanishing gradient, exploding gradient
- Convolutional NNs: networks for images as input
- GANs: networks for generating output
- Auto-encoders: unsupervised training, learn low-dim representation
- Recurrent NNs: “unroll” the network and train with BPTT
- DQN: using a deep network to approximate the Q-function
- Theory: Many open questions!

Next time: Decision Trees and Ensemble methods