Part of slides borrowed from Russ who borrowed from Hugo Larochelle
Reinforcement learning should be used to solve large problems, e.g.
- Backgammon: $10^{20}$ states
- Computer Go: $10^{170}$ states
- Helicopter, robot, …: enormous continuous state space
- 
- Tabular methods clearly cannot handle this.. why?
Value Function Approximation (VFA)

- So far we have represented value functions by a lookup table
  - Every state $s$ has an entry $V(s)$, or
  - Every state-action pair $(s,a)$ has an entry $Q(s,a)$

- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
  - You cannot generalize across states!

- Solution for large MDPs:
  - Estimate value function with function approximation
    $$\hat{v}(s, w) \approx v_\pi(s)$$
    $$\text{or } \hat{q}(s, a, w) \approx q_\pi(s, a)$$
  - Generalize from seen states to unseen states Why?
Value Function Approximation (VFA)

- **Value function approximation** (VFA) replaces the table with a general parameterized form:

Why this is a good idea?
Because those functions can be trained to map similar states to similar values. E.g., for a navigation agent the color of the carpet does not matter, or how many vases are on the table, let alone the amount of sun coming in from the window.
End-to-End RL

- **End-to-end RL methods** replace the hand-designed state representation with raw observations.

![Diagram](image)

- We get rid of manual design of state representations :-)
- We need tons of data to train the network since $O_t$ usually WAY more high dimensional than hand-designed $S_t$ :-(

Which Function Approximation?

- There are many function approximators, e.g.
  - Linear combinations of features
  - Neural networks
  - Decision tree
  - Nearest neighbour
  - ...

- In this lecture we will consider:
  - Linear combinations of features
  - Neural networks
Deep Learning

Image

Diagonal Line Node

Face Node

Cat Node
Artificial Neuron

- Neuron pre-activation (or input activation):

\[
a(x) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}
\]

- Neuron output activation:

\[
h(x) = g(a(x)) = g(b + \sum_i w_i x_i)
\]

where

- \( \mathbf{W} \) are the weights (parameters)
- \( b \) is the bias term
- \( g(\cdot) \) is called the activation function
Artificial Neuron

- Output activation of the neuron:

\[ h(x) = g(a(x)) = g(b + \sum_i w_i x_i) \]

Range is determined by \( g(\cdot) \)

Bias only changes the position of the riff

(from Pascal Vincent’s slides)
Activation Function

• Sigmoid activation function:

- Squashes the neuron’s output between 0 and 1
- Always positive
- Bounded
- Strictly Increasing

\[ g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)} \]
Activation Function

- Rectified linear (ReLU) activation function:

  ➢ Bounded below by 0 (always non-negative)
  ➢ Tends to produce units with sparse activities
  ➢ Not upper bounded
  ➢ Strictly increasing

\[ g(a) = \text{reclin}(a) = \max(0, a) \]
Single Hidden Layer Neural Net

- Hidden layer pre-activation:
  \[ a(x) = b^{(1)} + W^{(1)}x \]
  \[ (a(x)_i = b^{(1)}_i + \sum_j W^{(1)}_{i,j} x_j) \]

- Hidden layer activation:
  \[ h(x) = g(a(x)) \]

- Output layer activation:
  \[ f(x) = o \left( b^{(2)} + W^{(2)\top} h^{(1)}x \right) \]

\[ W^{(k)}_{i,j} : \text{Weight from neuron i to neuron j in layer k} \]
• Consider a network with \( L \) hidden layers.

  - layer pre-activation for \( k>0 \)
    \[
    a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)
    \]

  - hidden layer activation from 1 to \( L \):
    \[
    h^{(k)}(x) = g(a^{(k)}(x))
    \]

  - output layer activation (\( k=L+1 \)):
    \[
    h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x) \\
    (h^{(0)}(x) = x)
    \]
Capacity of Neural Nets

- Consider a single layer neural network

(from Pascal Vincent’s slides)
Capacity of Neural Nets

- Consider a single layer neural network
Universal Approximation

• Universal Approximation Theorem (Hornik, 1991):

  – “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

• This applies for sigmoid, tanh and many other activation functions.

• However, this does not mean that there is learning algorithm that can find the necessary parameter values.
Training

• Empirical Risk Minimization:

$$\arg\min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)$$

• Learning is cast as optimization.

➢ For classification problems, we would like to minimize classification error.

➢ For regression problems, we would like to minimize regression error, e.g., L1 or L2 distance from ground-truth
Stochastic Gradient Descend

- Perform updates after seeing each example:
  - Initialize: \( \theta \equiv \{ \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)} \} \)
  - For \( t=1:T \)
    - for each training example \( (\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) \)
      \[
      \Delta = - \nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta) \\
      \theta \leftarrow \theta + \alpha \Delta
      \]

- To train a neural net, we need:
  - Loss function: \( l(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)}) \)
  - A procedure to compute gradients: \( \nabla_{\theta} l(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)}) \)
  - Regularizer and its gradient: \( \Omega(\theta), \nabla_{\theta} \Omega(\theta) \)
Computational Flow Graph

- Forward propagation can be represented as an acyclic flow graph

- Forward propagation can be implemented in a modular way:
  - Each box can be an object with an `fprop` method, that computes the value of the box given its children
  - Calling the `fprop` method of each box in the right order yields forward propagation
Each object also has a \texttt{bprop} method

- it computes the gradient of the loss with respect to each child box.

By calling \texttt{bprop} in the reverse order, we obtain backpropagation
Mini-batch, Momentum

- Make updates based on a mini-batch of examples (instead of a single example):
  - the gradient is the average regularized loss for that mini-batch
  - can give a more accurate estimate of the gradient

- **Momentum**: Can use an exponential average of previous gradients:
  \[
  \overline{\nabla_\theta}(t) = \nabla_\theta l(f(x^{(t)}), y^{(t)}) + \beta \overline{\nabla_\theta}^{(t-1)}
  \]
  - can get pass plateaus more quickly, by “gaining momentum”
Model Selection

• Training Protocol:
  
  – Train your model on the Training Set \( D_{\text{train}} \)
  
  – For model selection, use Validation Set \( D_{\text{valid}} \)
    
    \( \triangleright \) Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
  
  – Estimate generalization performance using the Test Set \( D_{\text{test}} \)

• Generalization is the behavior of the model on unseen examples.
Early Stopping

• To select the number of epochs, stop training when validation set error increases (with some look ahead).
Recurrent Neural Networks

- RNNs tie the weights at each time step
- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled between inputs and outputs, as well as between output labels.
- In practice, limitations from SGD training, capacity, initialization etc.
Recurrent Neural Networks

- RNNs tie the weights at each time step
- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled between inputs and outputs, as well as between output labels.
- In practice, limitations from SGD training, capacity, initialization etc.
Recurrent Neural Network (single hidden layer)

- Given list of vectors: \( x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T \)
- At a single time step:
  \[
  h_t = \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right)
  \]
  \[
  \hat{y}_t = \text{softmax} \left( W^{(S)} h_t \right)
  \]

LSTMs and GRUs have more complex state update equations, learning when to explicitly replace or not update specific parts of the state \( h \), but it is the same idea.
Recurrent Neural Networks for camera pose estimation

DeepVO: Towards Visual Odometry with Deep Learning
Recurrent Neural Networks for video captioning
Learning Distributed Representations

• Deep learning is research on learning models with **multilayer representations**
  
  ➢ multilayer (feed-forward) neural networks
  ➢ multilayer graphical model (deep belief network, deep Boltzmann machine)

• Each layer learns “**distributed representation**”
  
  ➢ Units in a layer are not mutually exclusive
    • each unit is a separate feature of the input
    • two units can be “active” at the same time
  ➢ Units do not correspond to a partitioning (clustering) of the inputs
    • in clustering, an input can only belong to a single cluster
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators
  - Parameters for each region.
  - # of regions is linear with # of parameters.
- RBMs, Factor models, PCA, Sparse Coding, Deep models

(Bengio, 2009, Foundations and Trends in Machine Learning)
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators

- Parameters for each region.
- # of regions is linear with # of parameters.

- RBMs, Factor models, PCA, Sparse Coding, Deep models

Learned prototypes
Local vs. Distributed Representations

- Clustering, Nearest Neighbors, RBF SVM, local density estimators
  - Parameters for each region.
  - # of regions is linear with # of parameters.

- RBMs, Factor models, PCA, Sparse Coding, Deep models
  - Each parameter affects many regions, not just local.
  - # of regions grows (roughly) exponentially in # of parameters.
RNNs VS HMMs

# of parameters scales quadratically w.r.t. the number of states

But! Number of world states encoded (in ideal case of binary activations and optimization attains all those combinations) in the RNN is $2^N$ while it is $N$ for the HMM

The power of distributed representations

# of parameters scales quadratically w.r.t. the number of neurons in the hidden state
Computer Vision

- Design algorithms that can process visual data to accomplish a given task:

  ➢ For example, **object recognition**: Given an input image, identify which object it contains.

  ![Image of a sunflower]

  "sun flower"
Inspiration from Visual Cortex

[picture from Simon Thorpe]
Deep Convolutional Nets

- Convolution
- Pooling
- Normalization
- Densely connected

Very deep network

High-level feature space

Prediction
Computer Vision

- **Convolutional networks** leverage these ideas
  
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units
Local Connectivity

• Units are connected to all channels:
  ➢ 1 channel if grayscale image,
  ➢ 3 channels (R, G, B) if color image
Local Connectivity

• Example: 200x200 image, 40K hidden units, ~2B parameters!

➢ Spatial correlation is local
➢ Too many parameters, will require a lot of training data!
Local Connectivity

- **Example:** 200x200 image, 40K hidden units, filter size 10x10, 4M parameters!

> This parameterization is good when input image is registered
Computer Vision

• **Convolutional networks** leverage these ideas

  ➢ Local connectivity

  ➢ **Parameter sharing**

  ➢ Convolution

  ➢ Pooling / subsampling hidden units
Parameter Sharing

• Share matrix of parameters across some units
  ➢ Units that are organized into the ‘feature map’ share parameters
  ➢ Hidden units within a feature map cover different positions in the image

\[ W_{ij} \] is the matrix connecting the \( i^{th} \) input channel with the \( j^{th} \) feature map

same color = same matrix of connection
Parameter Sharing

• Share matrix of parameters across certain units

➢ Convolutions with certain kernels
Multiple Feature Maps

- Example: 200x200 image, 100 filters, filter size 10x10, 10K parameters
Computer Vision

- Convolutional networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Convolution
  - Pooling / subsampling hidden units
Pooling

- Pool hidden units in same neighborhood

  ➢ pooling is performed in non-overlapping neighborhoods
    (subsampling)

  \[ y_{ijk} = \max_{p,q} x_{i,j+p,k+q} \]

- \( x_i \) is the \( i \)th channel of input
- \( x_{i,j,k} \) is value of the \( i \)th feature map at position \( j,k \)
- \( p \) is vertical index in local neighborhood
- \( q \) is horizontal index in local neighborhood
- \( y_{ijk} \) is pooled / subsampled layer

Jarret et al. 2009
Example: Pooling

• Illustration of pooling/subsampling operation

Why pooling?

➢ Introduces invariance to local translations
➢ Reduces the number of hidden units in hidden layer
Example: Pooling

➢ can we make the detection robust to the exact location of the eye?
Example: Pooling

➢ By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Convolutional Network

- Convolutional neural network alternates between the convolutional and pooling layers.
Deep Convolutional Nets
Deep Convolutional Nets
Deep Convolutional Nets

Convolution

Pooling

Convolution
Deep Convolutional Nets

Convolution

Pooling

...
Convolutional Network

• For **classification**: Output layer is a regular, fully connected layer with softmax non-linearity
  
  ➢ Output provides an estimate of the conditional probability of each class
Softmax

\[
\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.
\]
ImageNet Dataset

• 1.2 million images, 1000 classes

Examples of Hammer

(Deng et al., Imagenet: a large scale hierarchical image database, CVPR 2009)
Important Breakthrough

- Deep Convolutional Nets for Vision (Supervised)

[Diagrams and images of ImageNet categories]

1.2 million training images
1000 classes
AlexNet

- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR’09]
- 18.2% top-5 error

[From Rob Fergus’ CIFAR 2016 tutorial]
AlexNet

- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!

[From Rob Fergus’ CIFAR 2016 tutorial]
AlexNet

• Let us remove upper feature extractor layers and fully connected:
  ➢ Layers 3, 4, 6 and 7
• Drop ~50 million parameters
• 33.5 drop in performance!
• Depth of the network is the key.

Later architectures avoid stacks of FC layers as the increase the # of parameter by a lot and thus, fear of overfitting

[From Rob Fergus’ CIFAR 2016 tutorial]
GoogLeNet

• 24 layer model that uses so-called inception module.

(Szegedy et al., Going Deep with Convolutions, 2014)
GoogLeNet

- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely.
- Number of parameters is reduced to 5 million.
- 6.7% top-5 validation error on Imagnet.

(Szegedy et al., Going Deep with Convolutions, 2014)
Residual Networks
[He, Zhang, Ren, Sun, CVPR 2016]

Really, really deep convnets don’t train well, E.g. CIFAR10:
Residual Networks
[He, Zhang, Ren, Sun, CVPR 2016]

Really, really deep convnets don’t train well, E.g. CIFAR10:

Key idea: introduce “pass through” into each layer

Thus only residual now needs to be learned
Residual Networks
[He, Zhang, Ren, Sun, CVPR 2016]

Really, really deep convnets don’t train well, E.g. CIFAR10:

Key idea: introduce “pass through” into each layer

Thus only residual now needs to be learned

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Err.</th>
<th>Top-5 Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>-</td>
<td>8.43³</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>-</td>
<td>7.89</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>24.4</td>
<td>7.1</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>21.99</td>
<td>5.81</td>
</tr>
<tr>
<td>ResNet-34 B</td>
<td>21.84</td>
<td>5.71</td>
</tr>
<tr>
<td>ResNet-34 C</td>
<td>21.53</td>
<td>5.60</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>20.74</td>
<td>5.25</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>19.87</td>
<td>4.60</td>
</tr>
<tr>
<td>ResNet-152</td>
<td><strong>19.38</strong></td>
<td><strong>4.49</strong></td>
</tr>
</tbody>
</table>

Table 4. Error rates (%) of single-model results on the ImageNet validation set (except ³ reported on the test set).

With ensembles, 3.57% top-5 test error on ImageNet
Value Function Approximation (VFA)

- Value function approximation (VFA) replaces the table with a general parameterized form:

Why this is a good idea?
Because those functions can be trained to map similar states to similar values. E.g., for a navigation agent the color of the carpet does not matter, or how many vases are on the table, let alone the amount of sun coming in from the window.
End-to-End RL

- **End-to-end RL methods** replace the hand-designed state representation with raw observations.

\[
O_t \xrightarrow{\theta} \hat{v}(O_t, \theta)
\]

\[
O_t \xrightarrow{\theta} \hat{q}(O_t, A_t, \theta)
\]

- We get rid of manual design of state representations :-) 
- We need tons of data to train the network since \( O_t \) usually WAY more high dimensional than hand-designed \( S_t \) :-(

End-to-End RL

- End-to-end RL methods replace the hand-designed state representation with raw observations.

Playing Atari with Deep reinforcement learning, deepmind, 2013: The first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning
End-to-End RL

- The network learns to map observations to states to actions
- All Q values for all actions are compute in one forward pass!
- Initial image is 210X160-> downsample->110X84-> crop-> 84X84-> input to network

'Playing Atari with Deep reinforcement learning, 2013: The first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning
Architecture Search

• How can we select the right architecture:

  ➢ Manual tuning of features is now replaced with the manual tuning of architectures

What does it mean to choose the right architecture?

• Depth

• Width

• Parameter count

What structural biases we need to do well for learning action policies?
What-where decomposition

• Many tasks require fine grained understanding of the scene
• CNNs are good to predict ``what” is there, not ``where” it is, due to extensive pooling and loss of resolution in the upper layers.
• How can we represent both what is in the scene and where it is?
Object Detection
Deep Object detection

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

Region-CNN, Girshick et al., 2013
Object detection

Region-CNN, Girshick et al., 2013
Object detection

Region-CNN, Girshick et al., 2013

Faster RCNN, Girshick et al., 2014
Object detection for manipulation

- Object detection works, when someone has labelled the objects we care about.
- How can we use this tools in CNNs for manipulation?
• Learn a function that maps RGB images $O_t$ into state vectors $x_t$ so that Euclidean distances in the state space reflect task progress (reward)
Architectures for deep state representations

State is a latent vector extracted convolutionally from the full frame.

State is an object graph, comprised of **cross-object distances** and **within object appearance features**.
Good reward function

Explicit attention on objects helps
But we do not have labels for our objects! What do we do?
1. Detector Training

Ground truth

Synthetic Data

Train new classes

Detector Network
Object-centric Visual Feature Learning

1. Detector Training

Ground truth

Synthetic Data

Detector Net

Dreaming data for learning on-the-fly state-of-the-art object detectors!
Object-centric Visual Feature Learning

1. Detector Training
   - Ground truth
   - Synthetic Data
   - Train new classes
   - Detector Network

2. Visual Feature Training
   - Human demonstrations
   - Object sequences
   - Train triplet loss
   - Time Contrastive Network

Bounding boxes
Object-centric Visual Feature Learning

1. Detector Training
   - Ground truth
   - Synthetic Data
   - Train new classes
   - Detector Network

2. Visual Feature Training
   - Human demonstrations
   - Object sequences
   - Train triplet loss
   - Time Contrastive Network

3. Inference
   - Bounding boxes
   - Global features (rel. arrangement)
   - Local features (visual appearance)
   - Robot trial

Finetune on failed detections
Object-centric Visual Feature Learning

We use a trajectory optimization method, attempting to minimize visual dissimilarity between execution and demonstration.

Given: \( \bar{x}_0 \)

For \( t = 0, 1, 2, \ldots, T \)

- Solve

\[
\min_{x,u} \sum_{k=t}^{T} \|x_k - x_d^k\|
\]

s.t. \( x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, \ldots, T-1\} \)

\( x_t = \bar{x}_t \)

- Execute \( u_t \)

- Observe resulting state, \( \bar{x}_{t+1} \)

- Initialize with solution from \( t - 1 \) to solve fast at time \( t \)
Learning motor conditioned Dynamics

F → Neural network →
Object-centric prediction

World-Centric Prediction

Object-Centric Prediction
Object-centric predictions

$F_1 \cdots F_t \rightarrow \text{Neural network} \rightarrow \text{Output}$
Training Time

\( \mathbf{F}_1 \cdots \mathbf{F}_t \) 

Forces applied to the centered ball

\( \mathbf{X}_{t+1} \cdots \mathbf{X}_{t+20} \)

Displacements of the centered ball in the next 20 frames
Network Architecture

\[ \mathbf{X}_t - 3 \cdots \mathbf{X}_t \]

\[ \mathbf{F}_t \]

\[ \mathbf{X}_{t+1} \cdots \mathbf{X}_{t+20} \]

long temporal memory

short temporal memory
Generalization

object-centric: solid
frame-centric: dashed
Visual Imaginations
Spatial Softmax

End-to-end learning of visuomotor policies, Levine et al. 2015
Spatial Softmax

- For each feature map, "flatten" it and compute a softmax
- Then take X and Y grid coordinates and compute the corresponding weighted averages
- Imposes a very tight bottleneck and avoids overfitting
End-to-End RL

- End-to-end RL methods replace the hand-designed state representation with raw observations.

- We get rid of manual design of state representations :-) 
- We need tons of data to train the network since $O_t$ usually WAY more high dimensional than hand-designed $S_t$ :-( 
- We can pre-train or jointly train with additional losses (auxiliary tasks) :-) For example?
Unsupervised Losses / Pretraining

• We can always fine-tune from weights trained on a supervised visual task.
• We can use auxiliary tasks, e.g., autoencoders
• We can use prediction of griper key points (we know where they are using forward kinematics and camera calibration
• We can use inverse model learning
Autoencoders are trained to reconstruct the input (e.g., L2 pixel loss) after they pass through a tight bottleneck layer (the state representation).

What can go wrong?
Train to predict the robotic action

Learning to poke by poking, Agrawal et al., 2015