Announcements

- Project 4 released! Due on **March 21st**

- 3rd tutorial on Friday, 5-6pm, on PyTorch
What is deep learning?

• Processing of data through multiple layers of non-linear functions to produce an output

• Not just neural networks!
  – Includes neural networks, but also Boltzmann machines, deep belief networks, CNNs, RNNs, etc.

• Main goal is to learn a representation of the data that is useful for many different tasks
  – Representation of data: function or transformation of the data into a (usually smaller) form that is easier to use (to solve various tasks)
The deep learning objective

very high level representation:

\[ \begin{array}{c}
\text{MAN} \\
\text{SITTING} \\
\cdots
\end{array} \]

\[ \cdots \text{etc} \cdots \]

slightly higher level representation

raw input vector representation:

\[
\mathbf{\mathcal{X}} = \begin{bmatrix}
23 & 19 & 20 \\
\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_n
\end{bmatrix}
\]
Learning an autoencoder function

- **Goal:** Learn a compressed representation of the input data.

- We have two functions:
  - **Encoder:** \( h = f_W(x) = s_f(Wx) \)
  - **Decoder:** \( x' = g_W(h) = s_g(W'h) \)

where \( s() \) is the activation function and \( W, W' \) are weight matrices.
Learning an autoencoder function

- **Goal**: Learn a compressed representation of the input data.

- **We have two functions**:
  - **Encoder**: \( h = f_W(x) = s_f(Wx) \)
  - **Decoder**: \( x' = g_W(h) = s_g(W'h) \)

where \( s() \) is the activation function and \( W, W' \) are weight matrices.

- **To train**, minimize reconstruction error:

\[
Err(W, W') = \sum_{i=1:n} L \left[ x_i, g_W(f_W(x_i)) \right]
\]

using squared-error loss (continuous inputs) or cross-entropy (binary inputs).
PCA vs autoencoders

In the case of a linear function:

\[
f_W(x) = Wx \quad g_{\hat{W}}(h) = W'h
\]

with squared-error loss:

\[
Err(W, W') = \sum_{i=1:n} \| x_i - g_{\hat{W}}( f_W(x_i) ) \|^2
\]

we can show that the minimum error solution \( W \) yields the same subspace as PCA.
Regularization of autoencoders

- **Weight tying** of the encoder and decoder weights \( W = W' \) to explicitly constrain (regularize) the learned function.

- How can we generate **sparse autoencoders**? (And also, why?)
  - Directly penalize the output of the hidden units (e.g. with L1 penalty) to introduce sparsity in the weights.
  - Helps ‘disentangle’ some of the factors of variation in the data.
Denoising autoencoders

Noisy input $\rightarrow$ Encoder $\rightarrow$ Decoder $\rightarrow$ Denoised image

The feature we want to extract from the image

*Image source: blog.sicara.com*
Denoising autoencoders

- **Idea**: To force the hidden layer to discover more robust features, train the autoencoder with a corrupted version of the input.
Denoising autoencoders

- **Idea**: To force the hidden layer to discover more robust features, train the autoencoder with a corrupted version of the input.

- **Corruption processes**:
  - Additive Gaussian noise
  - Randomly set some input features to zero.
  - More noise models in the literature.
Denoising autoencoders

- **Idea**: To force the hidden layer to discover more robust features, train the autoencoder with a corrupted version of the input.

- **Corruption processes**:
  - Additive Gaussian noise
  - Randomly set some input features to zero.
  - More noise models in the literature.

- **Training criterion**:

  \[
  \text{Err}(W, W') = \sum_{i=1:n} E_{q(x'_i|x_i)} L \left[ x_i, g_{W'}(f_W(x'_i)) \right]
  \]

  where \( x \) is the original input, \( x' \) is the corrupted input, and \( q() \) is the corruption process.
Denoising autoencoders

prior: examples concentrate near a lower dimensional “manifold”

original input

Corrupted input

Reconstruction function

Corrupted input
Contractive autoencoders

- **Goal**: Learn a representation that is robust to noise and perturbations of the input data, by regularizing the latent space

- **Contractive autoencoder training criterion**:

  \[
  \text{Err}(W, W') = \sum_{i=1}^{n} L \left[ x_i, g_{W'}(f_W(x_i)) \right] + \lambda \| J(x_i) \|_F^2
  \]

  where \( J(x_i) = \frac{\partial f_W(x_i)}{\partial x_i} \) is a Jacobian matrix of the encoder evaluated at \( x_i \), \( F \) is the Frobenius norm, and \( \lambda \) controls the strength of regularization.

- **Idea**: Penalize the model if a small change in input will result in a big change in representation (output of encoder)

  *Many more similar ideas in the literature…*
Unsupervised pretraining

- Autoencoders are a kind of ‘unsupervised learning’
- When do we want to use autoencoders?

1. Want to learn representations (features) of the data, but not sure for what task
2. Useful as a kind of ‘extra data’ for supervised tasks (e.g. \textit{pretraining})
3. Can be used for clustering or visualization
Variety of training protocols

• **Purely supervised:**
  – Initialize parameters randomly.
  – Train in supervised mode (gradient descent w/backprop.)
  – Used in most practical systems for speech and language.

• **Unsupervised pretraining + supervised classifier on top:**
  – Train an autoencoder to learn features of the data.
  – Train a supervised classifier on top, keeping other layers fixed.
  – Good when very few labeled examples are available.

• **Unsupervised pretraining + global supervised fine-tuning.**
  – Train an autoencoder to learn features of the data.
  – Add a classifier layer, and retrain the whole thing supervised.
  – Good when label set is poor.

• **Unsupervised pretraining often uses regularized autoencoders.**

From: [http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013](http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013)
Problem #1: feature co-adaptation

- In neural networks, derivative received by each parameter tells it what to do, *given what the other parameters are doing*
- This could lead to some neurons ‘fixing’ the problems caused by other neurons -> *co-adaptation*
- While this is okay on the training set, these fixes often don’t generalize to the test set

- “*Dropout: a simple way to prevent neural networks from overfitting,*” Srivastava et al., 2014
Dropout

- Independently set each hidden unit activity to **zero** with probability $p$ (usually $p=0.5$ works best).
- Neurons are forced to work with random subset of neighbours.
Problem #2: internal covariate shift

• During training, each layer of a neural network gets ‘used to’ the distribution of its inputs from the lower layer

• But the *distribution of outputs at each layer changes over time* as the network trains!
  – Each layer has to keep re-adapting to the new distribution
  – This problem is called *internal covariate shift*

• This can slow down and destabilize learning

Batch normalization

• Idea: Feature scaling makes gradient descent easier.
  • We already apply this at the input layer; extend to other layers.
  • Use empirical batch statistics to choose re-scaling parameters.

• For each mini-batch of data, at each layer $k$ of the network:
  – Compute empirical mean and var independently for each dimension
  – Normalize each input:
    \[ \hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{VAR}[x^{(k)}]}} \]
  – Output has tunable parameters $(\gamma, \beta)$ for each layer:
    \[ y^k = \gamma^k \cdot \hat{x}^{(k)} + \beta^k \]

• Effect: More stable gradient estimates, especially for deep networks.
Batch normalization

- Many other kinds of normalization: *e.g.* *weight normalization*, *layer normalization*, *batch re-normalization*, etc.
- Dropout and batch normalization empirically act as *regularizers*
  - Usually don’t need to use an L2 penalty on the weights
- Can use both, but *batch normalization alone works extremely well*
Do we really need deep architectures?

• We can approximate any function with a one-hidden-layer neural network. Why go deeper?

• **Deep networks are more efficient for representing certain classes of functions, with certain types of structure.**
  – Natural signals (images, speech) typically have such structure.

• Deep architectures can represent more complex functions with **fewer parameters**.

• So far, very little theoretical analysis of deep learning.
Do we really need deep architectures?

Deep neural networks learn hierarchical feature representations.
Major paradigms for deep learning

• **Deep neural networks**
  – Unsupervised pre-training: Autoencoders.

• Special architectures for different problem domains.
  – Computer vision => Convolutional neural nets.
  – Text and speech => Recurrent neural nets. (*Next class.*)
ImageNet dataset

ImageNet 2011 Fall Release (32326)
- plant, flora, plant life (4486)
- geological formation, formation (1)
- natural object (1112)
- sport, athletics (176)
- artifact, artefact (10504)
- fungus (308)
- person, individual, someone, somel
- animal, animate being, beast, brute
- Misc (20400)
  - julienne, julienne vegetable (0)
  - raw vegetable, rabbit food (0)
  - pulse (0)
  - goa bean (0)
  - kidney bean (0)
  - navy bean, pea bean, white bean
  - pinto bean (0)
  - frijole (0)
  - black bean, turtle bean (0)
  - snap bean, snap (0)
  - string bean (0)
  - Kentucky wonder, Kentucky woe
  - scarlet runner, scarlet runner b
  - haricot vert, haricots verts, Fre
  - green bean (5)
  - wax bean, yellow bean (0)
  - Fordhook (0)
  - lima bean (1)
  - sieva bean, butter bean, butterl
  - fava bean, broad bean (0)
  - green sovbean (0)

http://www.image-net.org
Neural networks for computer vision

• Design neural networks that are specifically adapted to:
  
  – Deal with very high-dimensional inputs
    • E.g. 150x150 pixels = 22,500 inputs, or 3x22,500 if RGB

  – Exploit 2D topology of pixels (or 3D for video)

  – Built-in invariance to certain variations we can expect
    • Translations, illumination, etc.
Why not feed-forward networks?

- Don’t take into account the structure of the data!
- Since input neurons have no ordering, an **image looks the same as a shuffled image**

These look the same to a feed-forward network!

*(so long as the same shuffling is applied to all of the data)*
Convolutional Neural Networks

Feedforward network

Convolutional neural network (CNN)

- CNN characteristics:
  - Input is a 3D tensor: 2D image x 3 colours (or 2D if grayscale)
  - Each layer transforms an input 3D tensor to an output 3D tensor using a differentiable function.

From: http://cs231n.github.io/convolutional-networks/
• **Convolutional neural networks** leverage several ideas.
  1. Local connectivity.
  2. Parameter sharing.
  3. Pooling hidden units.

Convolutional Neural Networks

• A few key ideas:

  1. Features have **local receptive fields**.
     - Each hidden unit is connected to a patch of the input image.
     - Units are connected to all 3 colour channels.

\[ \text{depth} = \# \text{ filters} \quad (a \text{ hyperparameter}) \]
Convolutional Neural Networks

• A few key ideas:

1. Features have **local receptive fields**.

2. **Share matrix of parameters** across units.
   - Constrain units within a depth slice (at all positions) to have **same** weights.
   - Feature map can be computed via discrete convolution with a kernel matrix.
Convolutional Neural Networks

- A few key ideas:
  1. Features have **local receptive fields**.
  2. **Share matrix of parameters** across units.
  3. **Pooling/subsampling** of hidden units in same neighbourhood.

Example:

```
Single depth slice
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
```

max pool with 2x2 filters and stride 2

Convolutional Neural Networks

• Local receptive fields
  – **Intuition:** there are some data features (e.g. edges, corners) that
    *only depend on a small region of the image*

• Parameter sharing
  – **Intuition:** processing these local features should be done *the*
    **same way** regardless of where the feature is in the image
  – *Much more efficient to train*

• Pooling/ subsampling
  – **Intuition:** usually doesn’t matter where *exactly* a feature occurs,
    only that it occurs somewhere
  – As we go deeper in the network, want to consider features that
    cover more area (i.e. *more global features*)
Convolutions

- What is a convolution?
- Formula: \((x * w)(t) = \sum_a x(a)w(t - a)\)
- \(x\) is the input data, \(w\) is the kernel
- The kernel is a function of learned parameters repeatedly applied to various parts of the input
Convolution

- $w, x, y, z$ are **learned** parameters
- Can have multiple kernels in a layer
Convolution

Input

Kernel

Output (feature map)
Convolution

- Averaging in a 3x3 box blurs the image

- Can be used for edge detection
Convolutional neural nets (CNNs)

- Alternate between **convolutional**, **pooling**, and **fully connected** layers.
  - Fully connected layer typically only at the end.
- Train full network using **backpropagation**.

(image from Yann Lecun)
Convolutional neural nets (CNNs)

From: http://cs231n.github.io/convolutional-networks/
Example: ImageNet

- SuperVision (a.k.a. AlexNet, 2012):
  - **Deep**: 7 hidden “weight” layers
  - **Learned**: all feature extractors initialized at white Gaussian noise and learned from the data
  - Entirely supervised
  - **More data = good**

Example: ImageNet

- SuperVision (a.k.a. AlexNet, 2012):

  - Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
  - 650,000 neurons
  - 60,000,000 parameters
  - 630,000,000 connections
  - Final feature layer: 4096-dimensional

Training results: ImageNet

- 96 learned low-level filters

Image classification

- 95% accuracy (on top 5 predictions) among 1,000 categories. Better than average human.
Empirical results (2012)

ImageNet 1K competition, fall 2012

Empirical results for image retrieval

- **Query items in leftmost column:**

Empirical results (2015)

ILSVRC top-5 error on ImageNet

CNNs vs traditional computer vision

From: Razavian et al. CVPR workshop paper. 2014.
### Picture tagging (From clarifai.com)

<table>
<thead>
<tr>
<th>Predicted Tags</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>food</td>
<td>(16.00%)</td>
</tr>
<tr>
<td>dinner</td>
<td>(3.10%)</td>
</tr>
<tr>
<td>bbq</td>
<td>(2.90%)</td>
</tr>
<tr>
<td>market</td>
<td>(2.50%)</td>
</tr>
<tr>
<td>meal</td>
<td>(1.40%)</td>
</tr>
<tr>
<td>turkey</td>
<td>(1.40%)</td>
</tr>
<tr>
<td>grill</td>
<td>(1.30%)</td>
</tr>
<tr>
<td>pizza</td>
<td>(1.30%)</td>
</tr>
<tr>
<td>eat</td>
<td>(1.10%)</td>
</tr>
<tr>
<td>holiday</td>
<td>(1.00%)</td>
</tr>
</tbody>
</table>

### Stats:
- Size: 247.24 KB
- Time: 110 ms
Scene parsing

(Farabet et al., 2013)
YOLO: Real-time object detection
Practical tips for CNNs

• Many hyper-parameters to choose!

• Architecture: filters (start small, e.g. 3x3, 5x5), pooling, number of layers (start small, add more).

• Training: learning rate, regularization, dropout rate (=0.5), initial weight size, batch size, batch norm.

• Read papers, copy their method, then do local search.
What you should know

• Types of deep learning architectures:
  – Autoencoders
  – Convolutional neural networks
  – Tricks to get neural networks to work

• Typical training approaches (unsupervised / supervised).

• Examples of successful applications.
More resources

• Deep learning textbook
  – In-depth treatment of all deep learning fundamentals
  – Available online for free: http://www.deeplearningbook.org/

• All articles on colah.github.io (highly recommended)
  – Well-explained articles on various neural network topics

• Convolution arithmetic
More resources

- Notes and images in today’s slides taken from:
  - http://www.cs.toronto.edu/~hinton/csc2535
  - http://deeplearning.net/tutorial/
  - http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013