Instructor: Joelle Pineau (jpineau@cs.mcgill.ca)

Class web page: www.cs.mcgill.ca/~jpineau/comp551
The deep learning objective

very high level representation:

\begin{align*}
\text{MAN} \quad \text{SITTING} \quad \ldots
\end{align*}

\ldots \text{etc} \ldots

slightly higher level representation

raw input vector representation:

\[ x = \begin{bmatrix}
23 & 19 & 20 \\
\vdots & \ddots & \vdots \\
\end{bmatrix} \]

\begin{align*}
x_1 & \\
x_2 & \\
x_3 & \\
x_n & \\
\end{align*}
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 $s()$ can be a sigmoid, linear, or other 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 \( s() \) can be a sigmoid, linear, or other function and \( W, W' \) are weight matrices.

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

  \[
  Err(W,W') = \sum_{i=1:n} L [ x_i, g_{w'}(f_w(x_i)) ]
  \]

  using squared-error loss (continuous inputs) or cross-entropy (binary inputs).
In the case of a linear function:

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

with squared-error loss:

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

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

**Key idea:** Apply greedy layerwise unsupervised pre-training.

http://www.dmi.usherbrooke.ca/~larocheh/projects_deep_learning.html
Regularization of autoencoders

- How can we generate *sparse autoencoders*? (And also, why?)
Regularization of autoencoders

- How can we generate **sparse autoencoders**? (And also, why?)

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

• How can we generate **sparse autoencoders**? (And also, why?)

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

• Directly **penalize the output of the hidden units** (e.g. with L1 penalty) to introduce sparsity in the weights.
Regularization of autoencoders

• How can we generate \textbf{sparse autoencoders}? (And also, why?)

• \textbf{Weight tying} of the encoder and decoder weights ($W=W'$) to explicitly constrain (regularize) the learned function.

• Directly \textbf{penalize the output of the hidden units} (e.g. with L1 penalty) to introduce sparsity in the weights.

• \textbf{Penalize the average output} (over a batch of data) to encourage it to approach a fixed target.
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.
Contractive autoencoders

- **Goal**: Learn a representation that is robust to noise and perturbations of the input data, by regularizing the latent space (represented by L2 norm of the Jacobian of the encoded input.)

- **Contractive autoencoder training criterion**:

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

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.

Many more similar ideas in the literature…
Supervised learning with deep models

**Final step:** Train the **full network** with backpropagation using error on the predicted output, $Err(W) = \sum_{i=1:n} L[y_i, o(x_i)]$

http://www.dmi.usherb.ca/~larocheh/projects_deep_learning.html
Supervised learning with deep models

Alternatively: Use the last representation layer (or concatenate all layers) as an input to a standard supervised learning predictor (e.g. SVM).

http://www.dmi.usherbrooke.ca/~larocheh/projects_deep_learning.html
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.

From: http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013
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, layerwise + supervised classifier on top:
  – Train each layer unsupervised, one after the other.
  – Train a supervised classifier on top, keeping other layers fixed.
  – Good when very few labeled examples are available.
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, layerwise + supervised classifier on top:**
  – Train each layer unsupervised, one after the other.
  – Train a supervised classifier on top, keeping other layers fixed.
  – Good when very few labeled examples are available.

• **Unsupervised, layerwise + global supervised fine-tuning.**
  – Train each layer unsupervised, one after the other.
  – 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)
Tip #1: Dropout regularization

- **Goal**: Learn model that generalizes well, robust to variability.

- **Method**: Independently set each hidden unit activity to zero with probability $p$ (usually $p=0.5$ works best).

- **Effect**: Can greatly reduce overfitting.
Tip #2: 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:
    \[ \hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{VAR[x^{(k)}]}} \]
  – 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.
Major paradigms for deep learning

• **Deep neural networks**: The model should be interpreted as a computation graph.
  – **Supervised training**: Feedforward neural networks.
  – **Unsupervised pre-training**: Stacked autoencoders.

• Special architectures for different problem domains.
  – **Computer vision** => Convolutional neural nets.
  – **Text and speech** => Recurrent neural nets.  

*Next class.*
ImageNet dataset

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.
Convolution Neural Networks

Feedforward network

Convolutional neural network (CNN)

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

From: http://cs231n.github.io/convolutional-networks/
Convolution Neural Networks

Feedforward network

Convolutional neural network (CNN)

- Convolutional neural networks leverage several ideas.
  1. Local connectivity.
  2. Parameter sharing.
  3. Pooling hidden units.

From: http://cs231n.github.io/convolutional-networks/
Convolution 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.

  depth = \# filters (a hyperparameter)
Convolution 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.
Convolution 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:

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)

Predicted Tags:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Probability</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)
Achieving super-human performance?

• Estimated 3% error in the labels.

• Differences between labeling process and human assessment:
  – Labels acquired as binary task. *Is there a dog in this picture?*
  – Human performance measured on 1K classes (>120 species of dogs in the dataset).
  – Labels acquired from experts (dog experts label the dogs, etc.).

• Machines and humans make different kinds of mistakes.
  – Both have trouble with multiple objects in an image.
  – Machines struggle with small/thin objects, image filters.
  – Humans struggle with fine-grained recognition.

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/
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.
Do we really need deep architectures?

• We can approximate any function to arbitrary levels of precision with shallow (2-level) architectures.

• Deep learning is more efficient for representing certain classes of functions, where there is certain types of structure.
  – Natural signals (images, speech) typically have such structure.

• Deep learning architectures can represent more complex functions with fewer parameters.
  – Trade-off (less) space for (more) time.

• So far, very little theoretical analysis of deep learning.
Quick recap + more resources

• A good survey paper:

• Notes and images in today’s slides taken from:
  • http://cs231n.github.io/convolutional-networks/
  • http://www.cs.toronto.edu/~hinton/csc2535
  • http://deeplearning.net/tutorial/
  • http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013
  • http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf
What you should know

- Types of deep learning architectures:
  - Stacked autoencoders
  - Convolutional neural networks

- Typical training approaches (unsupervised / supervised).

- Examples of successful applications.