CS 6140: Machine Learning
Spring 2017

Instructor: Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Email: luwang@ccs.neu.edu

Logistics
• Proposal grades and comments are available on blackboard.
  – Pick up during TA office hours
• Assignment 1 is due next week.
  – No need to print the code. Just submit it on Blackboard.
• Quiz solution will be posted.

What we learned last time
• Perceptron (and kernels)
• Support Vector Machines

Perceptron
• Weighted combination
  – The output of the neuron is a linear combination of the inputs
• Decision Function
  – At the end the results are combined into

$$f(x) = \sigma \left( \sum_{i=1}^{n} w_i x_i + b \right)$$

Perceptron Algorithm
• Nothing happens if we classify \((x_i, y_i)\) correctly
• If we see incorrectly classified observation we update \(w\) and \(b\)
• Positive reinforcement of observations

```plaintext
argument: \(X := \{x_1, \ldots, x_n\} \subset \mathcal{X}\) (data)
         \(Y := \{y_1, \ldots, y_n\} \subset \{\pm 1\}\) (labels)
function: \((w, b) = \text{Perceptron}(X, Y)\)
initialize \(w, b = 0\)
repeat
  Pick \((x, y)\) from data
  if \(y(w \cdot x + b) \leq 0\) then
    \(w' := w + y x\)
    \(b' := b + y\)
  end
until \(y(w \cdot x_i + b) > 0\) for all \(i\)
```
Perceptron Algorithm

- About the solution
  - Classification can be written in terms of dot products:
    \[ w \cdot x + b = \sum_{i \in E} y_i x_i \cdot x + b \]
  - Argument: \( X := \{x_1, \ldots, x_n\} \subseteq \mathcal{X} \) (data)
  - \( Y := \{y_1, \ldots, y_n\} \subseteq \{-1, 1\} \) (labels)
  - Function \((w, b) = \text{Perceptron}(X, Y)\)
  - Initialize \( w, b = 0 \)
  - Repeat
    - Pick \((x, y)\) from data
    - If \( y(w \cdot x_i + b) \leq 0 \) then
      \[ w' = w + y x_i \]
      \[ b' = b + y \]
    - Until \( y(w \cdot x_i + b) > 0 \) for all \( i \)

Perceptron on Features

- Argument: \( X := \{x_1, \ldots, x_n\} \subseteq \mathcal{X} \) (data)
- \( Y := \{y_1, \ldots, y_n\} \subseteq \{-1, 1\} \) (labels)
- Function \((w, b) = \text{Perceptron}(X, Y)\)
- Initialize \( w, b = 0 \)
- Repeat
  - Pick \((x, y)\) from data
  - If \( y(w \cdot \Phi(x) + b) \leq 0 \) then
    \[ w' = w + y \Phi(x) \]
    \[ b' = b + y \]
  - Until \( y(w \cdot \Phi(x) + b) > 0 \) for all \( i \)
- Important detail
  - \( w = \sum y_i \Phi(x_i) \) and hence \( f(x) = \sum y_i \Phi(x_i) \cdot \Phi(x) + b \)

Kernels

- Definition
- A kernel function \( k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \) is a symmetric function in its arguments for which the following property holds
  \[ k(x, x') = \langle \Phi(x), \Phi(x') \rangle \] for some feature map \( \Phi \)

Support Vector Machine (SVM)

- SVMs (Vapnik, 1990’s) choose the linear separator with the largest margin.

Support vector machines

- Minimize \( w, b \) \[ w \cdot x_j + b \geq 1, \forall j \]
- Example of a convex optimization problem
  - A quadratic program
  - Polynomial-time algorithms to solve
  - Hyperplane defined by support vectors
  - Could use them as a lower-dimension basis to write down the line, although we haven’t seen how yet

- Non-support vectors:
  - Everything else
  - Moving them will not change \( w \)
- Support vectors:
  - Data points on the canonical lines

- More on these later
Allowing for slack: “Soft margin SVM”

\[
\begin{align*}
\minimize_{\mathbf{w}, b} & \quad \mathbf{w} \cdot \mathbf{w} + C \sum_{j} \xi_j \\
\text{subject to} & \quad (\mathbf{w} \cdot \mathbf{x}_j + b) y_j \geq 1 - \xi_j, \quad \forall j, \xi_j \geq 0
\end{align*}
\]

Slack penalty \( C > 0 \):
- \( C \to \infty \) → have to separate the data!
- \( C \to 0 \) → ignores the data entirely!
- Select using validation data

For each data point:
- If margin \( \geq 1 \), don’t care
- If margin < 1, pay linear penalty

Dual for the non-separable case

Primal:
\[
\begin{align*}
\minimize_{\mathbf{w}, b} & \quad \frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{j} \xi_j \\
\text{subject to} & \quad (\mathbf{w} \cdot \mathbf{x}_j + b) y_j \geq 1 - \xi_j, \quad \forall j, \xi_j \geq 0, \quad \forall j
\end{align*}
\]

Solve for \( \mathbf{w}, b, \alpha \):
\[
\begin{align*}
\mathbf{w} &= \sum_{i} \alpha_i y_i \mathbf{x}_i \\
b &= y_k - \mathbf{w} \cdot \mathbf{x}_k
\end{align*}
\]

for any \( k \) where \( C > \alpha_k > 0 \)

Dual:
\[
\begin{align*}
\maximize_{\alpha} & \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\
\text{subject to} & \quad \sum_{i} \alpha_i y_i = 0 \\
& \quad C \geq \alpha_i \geq 0
\end{align*}
\]

Today’s Outline

• Feedforward neural network
• Training neural networks
• Restricted Boltzmann machine

Some slides are borrowed from Hugo Larochelle.
**Activation Function**

**Sigmoid Activation Function**
- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

\[ g(a) = \text{sign}(a) = \frac{1}{1 + e^{-a}} \]

**Hyperbolic Tangent ("Tanh") Activation Function**
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

\[ g(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \]

\[ \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \]

**Rectified Linear Activation Function**
- Bounded below by 0
  - Always non-negative
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities

\[ g(a) = \text{relu}(a) = \max(0, a) \]

**Class Neuron**

```python
class Neuron(object):
    def forward(self, inputs):
        # Assume inputs and weights are 1-D numpy arrays and bias is a number
        cell_body_plus = np.dot(inputs * self.weights + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_plus)) * self.sigmoid_activation_function
        return firing_rate
```

**Artificial Neuron**

**Capacity Decision Boundary of Neuron**
- Could do binary classification:
  - Sigmoid can interpret neuron as outputting \( p(y = 1|x) \)
  - Also known as logistic regression classifier
  - If greater than 0.5, predict class 1
  - Otherwise, predict class 0

**Capacity of Single Neuron**
- Can solve linearly separable problems

- OR \((x_1, x_2)\)
- AND \((x_1, x_2)\)
- AND \((x_1, x_2)\)
ARTIFICIAL NEURON

Topics: capacity of single neuron
- Can’t solve non-linearly separable problems...

XOR (x₁, x₂)

AND (x₁, x₂)

□ unless the input is transformed in a better representation

NEURAL NETWORK

Topics: single hidden layer neural network
- Hidden layer pre-activation:
  \[ a^{(2)} = b^{(2)} + W^{(2)} x \]
- Hidden layer activation:
  \[ h^{(2)} = g(a^{(2)}) \]
- Output layer activation:
  \[ f(x) = a^{(2)} + b^{(3)} = \sum_{i} b^{(3)} + \sum_{i} w^{(3)} h^{(2)}(x_i) \]

NEURAL NETWORK

Topics: softmax activation function
- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability \( p(y = c|x) \)
- We use the softmax activation function at the output:
  \[ o(a) = \text{softmax}(a) = \left[ \frac{\exp(a_1)}{\sum \exp(a_k)}, \ldots, \frac{\exp(a_n)}{\sum \exp(a_k)} \right] \]
  - strictly positive
  - sums to one
- Predicted class is the one with highest estimated probability

NEURAL NETWORK

Topics: single hidden layer neural network
- Hidden layer pre-activation:
  \[ a^{(2)} = b^{(2)} + W^{(2)} x \]
- Hidden layer activation:
  \[ h^{(2)} = g(a^{(2)}) \]
- Output layer activation:
  \[ f(x) = a^{(2)} + b^{(3)} = \sum_{i} b^{(3)} + \sum_{i} w^{(3)} h^{(2)}(x_i) \]

CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network
CAPACITY OF NEURAL NETWORK

**Topics:** single hidden layer neural network

![Diagram](from Neural Networks slides)

CAPACITY OF NEURAL NETWORK

**Topics:** universal approximation

- Universal approximation theorem (Cybenko, 1989)
  - A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units
- The result applies for sigmoid, tanh, and many other hidden layer activation functions
- This is a good result, but it doesn’t mean there is a learning algorithm that can find the necessary parameter values

![Graphs](3 hidden neurons, 6 hidden neurons, 20 hidden neurons)

NEURAL NETWORK

**Topics:** parallel with the visual cortex

![Diagram](from Sagi & Thurmer)
**NEURAL NETWORK**

Topics: parallel with the visual cortex

- [Image from S. Thorpe]

**NEURAL NETWORK**

Topics: parallel with the visual cortex

- [Image from S. Thorpe]

**NEURAL NETWORK**

Topics: parallel with the visual cortex

- [Image from S. Thorpe]

**BIOLOGICAL NEURONS**

Topics: synapse, axon, dendrite

- We estimate around $10^{10}$ and $10^{11}$ the number of neurons in the human brain:
  - they receive information from other neurons through their dendrites
  - the "process" the information in their cell body (soma)
  - they send information through a "cable" called an axon
  - the point of connection between the axon branches and other neurons' dendrites are called synapses

- [Image from S. Thorpe]
How to train a neural network?

The Learning Algorithm

Empirical Risk Minimization

Loss Function
The Learning Algorithm

**Topics:** stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - Initialize $\theta$ ($\theta = [W^{(1)}, b^{(1)}, \ldots, W^{(L-1)}, b^{(L-1)}]$)
  - For $N$ iterations
    - For each training example $(x^{(i)}, y^{(i)})$
      - $\Delta = -\nabla_{\theta} f(x^{(i)}; \theta, y^{(i)}) - \lambda \nabla_{\theta} g(\theta)$
      - $\theta = \theta - \alpha \Delta$
  - To apply this algorithm to neural network training, we need:
    - The loss function $f(f(x^{(i)}; \theta), y^{(i)})$
    - A procedure to compute the parameter gradients $\nabla_{\theta} f(f(x^{(i)}; \theta), y^{(i)})$
    - The regularizer $g(\theta)$ (and the gradient $\nabla_{\theta} g(\theta)$)
    - Initialization method

Gradient Computation

- **Output layer gradient** ($o$)
- **Hidden layer gradient** ($h$)
- **Activation function gradient** ($a$)
- **Parameter gradient** ($W, b$)

**Topics:**

- **Gradient Computation**
  - Partial derivative:
    - $\frac{\partial}{\partial f(x_y)} \log f(x)_y = \frac{-1}{f(x)_y}$
  - Gradient:
    - $\nabla_{f(x)_y} \log f(x)_y = \frac{-1}{f(x)_y} \begin{bmatrix} I_{(y=0)} \\ I_{(y=C-1)} \end{bmatrix}$
    - $= \frac{-e(y)}{f(x)_y}$
    - $\frac{\partial}{\partial \theta(L+1)(x)_y} \log f(x)_y$
Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)
Gradient Computation

- Output layer gradient $(o)$
- Hidden layer gradient $(h)$
- Activation function gradient $(a)$
- Parameter gradient $(W, b)$
Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)
BACKPROPAGATION

Topics: backpropagation algorithm
- This assumes a forward propagation has been made before
  - compute output gradient (before activation)
    \( \nabla_{w_{21:1}} \log f(x) \leftarrow \nabla_{w_{21:1}} \log f(x) \leftarrow \nabla_{w_{21:1}} \log f(x) \)
  - for \( k \) from 0 to 1
    - compute gradients of hidden layer parameters
      \( \nabla_{w_{k-1:1}} \log f(x) \leftarrow \nabla_{w_{k-1:1}} \log f(x) \leftarrow \nabla_{w_{k-1:1}} \log f(x) \)
    - compute gradient of hidden layer before
      \( \nabla_{w_{k-1:1}} \log f(x) \leftarrow \nabla_{w_{k-1:1}} \log f(x) \leftarrow \nabla_{w_{k-1:1}} \log f(x) \)
    - compute gradient of hidden layer before (before activation)
      \( \nabla_{w_{k-1:1}} \log f(x) \leftarrow \nabla_{w_{k-1:1}} \log f(x) \leftarrow \nabla_{w_{k-1:1}} \log f(x) \)

The Learning Algorithm

Topics: stochastic gradient descent (SGD)
- Algorithm that performs updates after each example
  - initialize \( \theta = \{ W^{(0)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)} \} \)
  - for \( N \) iterations
    - for each training example \( (x^{(i)}, y^{(i)}) \)
      \( \Delta = -\nabla_{\theta} J(x^{(i)}; \theta; y^{(i)}) - \lambda \nabla_{\theta} J(\theta) \)
    - \( \theta \leftarrow \theta + \alpha \Delta \)
- To apply this algorithm to neural network training, we need
  - the loss function \( J(f(x^{(i)}; \theta), y^{(i)}) \)
  - a procedure to compute the parameter gradients \( \nabla_{\theta} J(x^{(i)}; \theta; y^{(i)}) \)
  - a regularizer \( \Omega(\theta) \)
  - an initialization method

REGULARIZATION

Topics: L2 regularization
- \( \Omega(\theta) = \sum_k \sum_i \sum_j (w_{ij}^{(k)})^2 = \sum_k ||W^{(k)}||_2^2 \)
- \( \nabla_{W_{ij}^{(k)}} \Omega(\theta) = 2w_{ij}^{(k)} \)
- Only applied on weights, not on biases (weight decay)
- Can be interpreted as having a Gaussian prior over the weights

REGULARIZATION

Topics: L1 regularization
- \( \Omega(\theta) = \sum_k \sum_i \sum_j |w_{ij}^{(k)}| \)
- Gradient: \( \nabla_{w_{ij}^{(k)}} \Omega(\theta) = \text{sign}(w_{ij}^{(k)}) \)
- Also only applied on weights
- Unlike L2, L1 will push certain weights to be exactly 0
- Can be interpreted as having a Laplacian prior over the weights
Empirical Risk Minimization

**Topics:** empirical risk minimization, regularization

- **Empirical risk minimization**
  - framework to design learning algorithms
  
  \[
  \arg\min_{\theta} \frac{1}{n} \sum_{i} l(f(x^{(i)}; \theta), y^{(i)}) + \lambda \Omega(\theta)
  \]
  
  - \(l(f(x^{(i)}; \theta), y^{(i)})\) is a loss function
  - \(\Omega(\theta)\) is a regularizer (penalizes certain values of \(\theta\))
  - Learning is cast as optimization
    - ideally, we'd optimize classification error but it's not smooth
    - loss function is a surrogate for what we truly should optimize (e.g., upper bound)

The Learning Algorithm

**Topics:** stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initializes \(\theta\) (\(\theta \equiv \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}\))
  - for \(N\) iterations:
    - for each training example \((x^{(i)}, y^{(i)})\)
      - \(\Delta = -\nabla_{\theta} l(f(x^{(i)}; \theta), y^{(i)}) - \lambda \nabla_{\theta} \Omega(\theta)\)
      - \(\theta = \theta + \alpha \Delta\)
    - training epoch
  - To apply this algorithm to neural network training, we need:
    - the loss function \(l(f(x^{(i)}; \theta), y^{(i)})\)
    - a procedure to compute the parameter gradients \(\nabla_{\theta} l(f(x^{(i)}; \theta), y^{(i)})\)
    - the regularizer \(\Omega(\theta)\) (and the gradient \(\nabla_{\theta} \Omega(\theta)\))
    - initialization method

Toolkits

- **TensorFlow**
  - [https://www.tensorflow.org/](https://www.tensorflow.org/)
  - Python and C++

- **Theano**
  - [http://deeplearning.net/software/theano/](http://deeplearning.net/software/theano/)
  - Python

- **Torch**
  - [http://torch.ch/](http://torch.ch/)
  - LuaJIT
Unsupervised Learning with Neural Networks

• Unsupervised learning: only use the inputs for learning
  – automatically extract meaningful features for your data
  – leverage the availability of unlabeled data
  – add a data-dependent regularizer to training

\( -\log p(x^{(t)}) \)

Unsupervised Learning with Neural Networks

• Restricted Boltzmann machines

Restrict Boltzmann Machine

Topics: RBM, visible layer, hidden layer, energy function

\[
E(x, h) = -h^T W x - e^T x - h^T h + \sum_{i} x_i h_i - \sum_{i} y_i h_j
\]

\[ p(x, h) = \exp(-E(x, h))/Z \]

Example of Data Set: MNIST

Filters

Restricted Boltzmann Machine

Topics: RBM, visible layer, hidden layer, energy function

\[
E(x, h) = -h^T W x - e^T x - h^T h + \sum_{i} x_i h_i - \sum_{i} y_i h_j
\]

\[ p(x, h) = \exp(-E(x, h))/Z \]
Inference

- Conditional distributions: \( P(h|x), P(x|h) \)
- Sample distribution: \( P(x) \)

\[
p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')}
\]

\[
p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')}
= \frac{\exp(h^T W x + e^T x + b^T h)/Z}{\sum_{h' \in \{0,1\}^n} \exp(h'^T W x + e^T x + b^T h')/Z}
= \frac{\prod_{i \in \text{row of } W} \exp(h_i^T W x + e^T x + b^T h)/Z}{\sum_{h' \in \{0,1\}^n} \prod_{i \in \text{row of } W} \exp(h'^T W x + e^T x + b^T h')/Z}
\]
\[ p(h|x) = \frac{p(x, h)}{\sum_h p(x, h)} = \frac{\exp(h^T W x + e^T x + b h)}{\sum_{h' \in \{0,1\}^d} \exp(h'^T W x + e^T x + b h')} \]

\[ = \frac{\sum_{h \in \{0,1\}^d} \exp(h^T W x + e^T x + b h)}{\sum_{h' \in \{0,1\}^d} \exp(h'^T W x + e^T x + b h')} \]

\[ = \frac{\sum_{h \in \{0,1\}^d} \exp(h^T W x + b h)}{\sum_{h' \in \{0,1\}^d} \exp(h'^T W x + b h')} \]

\[ = \prod_{i \in \{0,1\}^d} \frac{\exp(h_i^T W x + b_i)}{\sum_{h' \in \{0,1\}^d} \exp(h'_i^T W x + b_i')} \]

\[ p(h|x) = \frac{p(x, h)}{\sum_h p(x, h)} = \frac{\exp(h^T W x + e^T x + b h)}{\sum_{h' \in \{0,1\}^d} \exp(h'^T W x + e^T x + b h')} \]

\[ = \frac{\sum_{h \in \{0,1\}^d} \exp(h^T W x + e^T x + b h)}{\sum_{h' \in \{0,1\}^d} \exp(h'^T W x + e^T x + b h')} \]

\[ = \frac{\sum_{h \in \{0,1\}^d} \exp(h^T W x + b h)}{\sum_{h' \in \{0,1\}^d} \exp(h'^T W x + b h')} \]

\[ = \prod_{i \in \{0,1\}^d} \frac{\exp(h_i^T W x + b_i)}{\sum_{h' \in \{0,1\}^d} \exp(h'_i^T W x + b_i')} \]

\[ p(h = 1|x) = \frac{\exp(b + W x)}{1 + \exp(b + W x)} \]
Inference

- Conditional distributions: $P(h|x)$, $P(x|h)$
- Sample distribution: $P(x)$

\[
p(h_j = 1 | x) = \frac{\exp(b_j + W_{j,x})}{1 + \exp(b_j + W_{j,x})}
\]

\[
p(h_j = 1 | x) = \frac{1}{1 + \exp(-b_j - W_{j,x})}
\]

\[
p(h_j = 1 | x) = \text{sigmoid}(b_j + W_{j,x})
\]

**RESTRICTED BOLTZMANN MACHINE**

**FREE ENERGY**

Topics: RBM, visible layer, hidden layer, energy function

Energy function: $E(x, h) = -h^T W x - c^T x - b^T h$

$p(x, h) = \exp(-E(x, h))/Z$

$p(x) = \sum_{h \in \{0,1\}^v} p(x, h) = \sum_{h \in \{0,1\}^v} \frac{1}{Z} = \exp(-F(x))/Z$

$p(x) = \sum_{h \in \{0,1\}^v} \exp(h^T W x + c^T x + b^T h)/Z$

$p(x) = \exp(c^T x) \sum_{h_j \in \{0,1\}} \cdots \sum_{h_0 \in \{0,1\}} \exp \left( \sum_j h_j W_{j,x} + b_j h_j \right)/Z$
\[ p(x) = \sum_{h \in [0,1]^n} \exp(h^T W x + e^T x + h^T b) / Z \]

\[ = \exp(e^T x) \sum_{h \in [0,1]^n} \exp \left( \sum_{j} h_j W_{j} x + h_j b_j \right) / Z \]

\[ = \exp(e^T x) \left( \sum_{h \in [0,1]^n} \exp(h_0 W_{0} x + h_0 b_0) \right) \cdots \left( \sum_{h \in [0,1]^n} \exp(h_{m} W_{m} x + h_{m} b_{m}) \right) / Z \]

\[ = \exp(e^T x) \left( 1 + \exp(h_0 W_{0} x + h_0 b_0) \right) \cdots \left( 1 + \exp(h_{m} W_{m} x + h_{m} b_{m}) \right) / Z \]

\[ = \exp(e^T x) \exp \left( \sum_{j} \log \left( 1 + \exp(h_j W_{j} x + h_j b_j) \right) \right) / Z \]

\[ = \exp(e^T x) \exp \left( \sum_{j} \text{softplus}(h_j W_{j} x + h_j b_j) \right) / Z \]

\[ = \exp(e^T x) \exp \left( \sum \text{softplus}(h_j W_{j} x) \right) / Z \]

**Restricted Boltzmann Machine**

**Topics:** free energy

\[ h_{x}(x) = \exp \left( e^T x + \sum_{j} \log \left( 1 + \exp(h_j W_{j} x) \right) \right) / Z \]

\[ x = \exp \left( e^T x + \sum \text{softplus}(h_j W_{j} x) \right) / Z \]
Useful Resources for Deep Learning

• Book for deep learning
  – Deep Learning
  – By Ian Goodfellow, Yoshua Bengio, and Aaron Courville
  – http://www.deeplearningbook.org/

Useful Resources for Deep Learning

• Tutorials
  – General
  – Natural language processing
  – Computer Vision
    • https://sites.google.com/site/deeplearningcvpr2014/

What we learned today

• Feedforward neural network
• Training neural networks
• Restricted Boltzmann machine

Homework

• Read Murphy CH 16.5 and CH 28.
• Read slides from Hugo Larochelle.
  – http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html