This lecture

• Neural Networks
  A. Modeling
  B. Fitting
  C. Deep neural networks
  D. In practice

Some of today’s material is (adapted) from Joelle Pineau’s slides
From Linear Classification to Neural Networks
Recall Linear Classification
Recall Linear Classification

\[ y(x) = w^T x + w_0 \]

Decision

\[(w^T x + w_0) > 0 \implies \bullet \]
\[(w^T x + w_0) < 0 \implies \circ \]
Recall Linear Classification

\[ y(x) = w^T x + w_0 \]

Decision:
\[(w^T x + w_0) > 0 \implies \text{●} \]
\[(w^T x + w_0) < 0 \implies \text{○} \]
What if data is not linearly separable?

Exclusive OR (XOR)

$x_1$

$x_2$
What if data is not linearly separable?

Use the joint decision of several linear classifier?

Exclusive OR (XOR)
What if data is not linearly separable?

Use the joint decision of several linear classifier?

Exclusive OR (XOR)
Combining models

\[ f(x): \]
\[ (w^T x + w_0) > 0 \quad \Rightarrow \quad \bullet \]
\[ (w^T x + w_0) < 0 \quad \Rightarrow \quad \bullet \]

\[ f'(x): \]
\[ (w'^T x + w'_0) > 0 \quad \Rightarrow \quad \bullet \]
\[ (w'^T x + w'_0) < 0 \quad \Rightarrow \quad \bullet \]
Combining models

\[ f(x): \begin{align*} (w^T x + w_0) &> 0 \quad \implies \quad \bullet \\ (w^T x + w_0) &< 0 \quad \implies \quad \bullet \end{align*} \]

\[ f'(x): \begin{align*} (w'^T x + w'_0) &> 0 \quad \implies \quad \bullet \\ (w'^T x + w'_0) &< 0 \quad \implies \quad \bullet \end{align*} \]
Combining models

1. Evaluate each model
   - \( f(x) : (w^T x + w_0) > 0 \) \( \Rightarrow \) \( \bullet \)
   - \( f(x) : (w^T x + w_0) < 0 \) \( \Rightarrow \) \( \circ \)
   - \( f'(x) : (w'^T x + w'_0) > 0 \) \( \Rightarrow \) \( \bullet \)
   - \( f'(x) : (w'^T x + w'_0) < 0 \) \( \Rightarrow \) \( \circ \)

2. Combine the output of models
   - \( f(x) = \bullet \) and \( f'(x) = \bullet \) \( \Rightarrow \) \( \bullet \)
   - \( f(x) = \circ \) and \( f'(x) = \bullet \) \( \Rightarrow \) \( \bullet \)
   - \( f(x) = \bullet \) and \( f'(x) = \circ \) \( \Rightarrow \) \( \bullet \)
   - \( f(x) = \circ \) and \( f'(x) = \circ \) \( \Rightarrow \) \( \circ \)
Combining models

1. Evaluate each model

\[ \begin{align*}
    f(x) & : (w^T x + w_0) > 0 \implies \text{green} \\
    f(x) & : (w^T x + w_0) < 0 \implies \text{blue}
\end{align*} \]

\[ \begin{align*}
    f'(x) & : (w'^T x + w'_0) > 0 \implies \text{blue} \\
    f'(x) & : (w'^T x + w'_0) < 0 \implies \text{green}
\end{align*} \]

2. Combine the output of models

\[ f''(x) = \text{threshold}(w''^T \begin{bmatrix} f(x) \\ f'(x) \end{bmatrix} + w'_0) \]
Combining model (graphical view)
Combining model (graphical view)

\[ x_1 \]

\[ x_2 \]
Combining model (graphical view)
Combining model (graphical view)
Combining model (graphical view)
Combining model (graphical view)

Neural Network

\[ f(x) \quad f'(x) \]

\[ f''(x) \quad \{\circ, \bullet\} \]
Combining model
(graphical view)

Neural Network

Perceptron/Neuron

\[ f(x) \]

\[ f'(x) \]

\[ f''(x) \]

\{ , \}
Feed-forward neural network

- Each arrow denotes a connection
- A signal associated with a weight
- Each node is the weighted sum of its input followed by a non-linear activation
- Connections go left to right
- No connections within a layer
- No backward connections (recurrent)
Feed-forward neural network

1. An input layer
   - Its size is the number of inputs + 1
2. One or more hidden layer(s)
   - Their size is a hyper-parameter
3. An output layer
   - Its size is the number of outputs
Compute a prediction (forward pass)

Input Layer

Hidden Layer(s)

Output Layer

1. $\sigma\left(\sum_{i} w_{i} x_{i}\right)$

2. $\sigma\left(\sum_{i} w_{i2} o_{1i}\right)$
Neural Networks

• Flexible model class

• Highly-non linear models

• Good for regression/classification/density estimation

• Models behind “Deep Learning”

• Historical aspects
Learning the Parameters of a Neural Network
Fitting a neural network
Fitting a neural network

How do we estimate the model’s parameters?
Fitting a neural network

How do we estimate the model’s parameters?

- No-closed form solution
Fitting a neural network

How do we estimate the model’s parameters?

• No-closed form solution
• Gradient-based optimization
Fitting a neural network

How do we estimate the model’s parameters?

- No-closed form solution
- Gradient-based optimization
- Threshold functions are not differentiable
Fitting a neural network

How do we estimate the model’s parameters?

• No-closed form solution

• Gradient-based optimization

• Threshold functions are not differentiable

• Replace by sigmoid (inverse logit). A soft threshold.

\[
sigmoid(a) := \left( \frac{1}{1 + \exp(-a)} \right)
\]
Fit the parameters \((w)\) (backward pass)

- Derive a gradient wrt the parameters \((w)\)

\[
\frac{\partial(y - \hat{y})^2}{\partial w_j} = \frac{\partial(y - f(\sum_i w_i o_i))^2}{\partial w_j} = \frac{\partial(y - f(\sum_i w_i f(\sum_j w_j x_j)_{i_i}))^2}{\partial w_j}
\]

- The back-propagation starts from the output node(s) and heads toward the input(s)

- In practice, the order of the computation is important
Gradient descent

• No closed-form formula

• Repeat the following steps (for $t=0,1,2,...$ until convergence):
  1. Calculate a gradient $\nabla W_{ij}^t$
  2. Apply the update $W_{ij}^{t+1} = W_{ij}^t - \alpha \nabla W_{ij}^t$

• Stochastic gradient descent
  • One example at a time

• Batch gradient descent
  • All examples at a time
What can an MLP learn?

1. A single unit (neuron)
   - Linear classifier + non-linear output

2. A network with a single hidden layer
   - Any continuous function (but may require exponentially many hidden units as a function of the inputs)

3. A network with two (or more) hidden layers
   - Any function can be approximated with arbitrary accuracy.
The Importance of Representations
From Neural Networks to Deep Neural Networks

A neural Network
From Neural Networks to Deep Neural Networks

A neural Network

A deep neural Network
Modern deep learning provides a powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity.

Deep Learning — Part II, p.163
http://www.deeplearningbook.org/contents/part_practical.html
Another View of deep learning

- Representations are important
Representations
Representations

Input data (pixels) \(\rightarrow\) Feature representation (engineered)

No learning

Image

Low-level vision features (e.g. SIFT, HOG, LBP, etc.) + some operations (e.g. quantization, pooling)

[From: Honglak Lee and Graham Taylor]
Representations
Representations

No learning

Input data (pixels) → Feature representation (engineered) → Learning Algorithm (e.g., SVM)

Image → Low-level vision features (e.g., SIFT, HOG, LBP, etc.) + some operations (e.g., quantization, pooling) → Recognition or detection

[From: Honglak Lee (and Graham Taylor)]
Data → Layer 1 → Layer 2 → ... → Classifier → Output
Machine Translation

- French Encoder
- English Encoder
- Spanish Encoder

Universal Sentence Representation

- French Decoder
- English Decoder
- Spanish Decoder
Machine learning
Make machines that can learn

Deep learning
A set of machine learning techniques based on neural networks

Idea: Hugo Larochelle
Machine learning
Make machines that can learn

Deep learning
A set of machine learning techniques based on neural networks

Representation learning
Machine learning paradigm to discover data representations

Idea: Hugo Larochelle
Deep neural networks

- Several layers of hidden nodes
- Parameters at different levels of representation
Neural Network
Hyper-parameters
Hyperparameters

1. Model specific
   - Activation functions (output & hidden), Network size

2. Optimisation Objective
   - Regularization, Early-stopping, Dropout

3. Optimization procedure
   - Momentum, Adaptive learning rates
Activation Functions

• Non-linear functions that transform the weighted sum of the inputs, e.g.:

\[ f(\sum_{i} w_i x_i) \]
Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

\[ f\left(\sum \limits_i w_i x_i\right) \]
Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:

$$f\left(\sum_{i} w_i x_i\right)$$

- Non-linearities increase model representation power
Activation Functions

• Non-linear functions that transform the weighted sum of the inputs, e.g.:

\[ f\left(\sum_{i} w_i x_i\right) \]

• Non-linearities increase model representation power

• Non-linearities increase the difficult of optimization
Activation Functions

- Non-linear functions that transform the weighted sum of the inputs, e.g.:
  \[ f(\sum_i w_i x_i) \]

- Non-linearities increase model representation power
- Non-linearities increase the difficult of optimization
- Different functions for hidden units and output units
Activation functions — hidden units

- Traditional

- Logistic-like units
  - \( f(z) = \logit^{-1}(z) = \frac{1}{1 + \exp(-z)} \)
  - \( f(z) = \tanh(z) \)

- Saturate on both sides

- Derivable everywhere
Activation functions — hidden units

- Traditional

- Logistic-like units
  - \( f(z) = \logit^{-1}(z) = \frac{1}{1 + \exp(-z)} \)
  - \( f(z) = \tanh(z) \)

- Saturate on both sides

- Derivable everywhere
Activation functions — hidden units

- Traditional

- Logistic-like units
  - $f(z) = \logit^{-1}(z) = \frac{1}{1 + \exp(-z)}$
  - $f(z) = \tanh(z)$

- Saturate on both sides
- Derivable everywhere

- Rectified linear units (Relu)
  \[ f(z) = \max\{0, z\} \]

- Non-derivable at a single point

- Now Standard

- Better results / faster training
- Shuts off units

- Leaky Relu
  \[ g(z) = \max\{0, z\} + \alpha \min\{0, z\} \]
Activation functions — Output units

<table>
<thead>
<tr>
<th>Output type</th>
<th>Output Unit</th>
<th>Equivalent Statistical Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary (0,1)</td>
<td>$\text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$</td>
<td>Bernoulli</td>
</tr>
<tr>
<td>Categorical (0,1,2,3,k)</td>
<td>$\text{softmax}(z) = \frac{\exp(z_i)}{\sum_p \exp(z_p)}$</td>
<td>Multinoulli</td>
</tr>
<tr>
<td>Continuous</td>
<td>Identity(z) = z</td>
<td>Gaussian</td>
</tr>
<tr>
<td>Multi-modal</td>
<td>mean, (co-)variance, components</td>
<td>Mixture of Gaussians</td>
</tr>
</tbody>
</table>
Regularization

- Weight decay on the parameters
- L2 penalty on the parameters
- Early stopping of the optimization procedure
  - Monitor the validation loss and terminate when it stops improving
- Number of hidden layers and hidden units per layer
Momentum

\[ w^{t+1} = w^t - \alpha \nabla w^t \quad \text{(Gradient Descent)} \]

\[ v = \beta v - \alpha \nabla w^t \quad \text{(Gradient Descent w. momentum)} \]

\[ w^{t+1} = w^t + v \]

- **Pro:** Can allow you to jump over small local optima
- **Pro:** Goes faster through flat areas by using acquired speed
- **Con:** Can also jump over global optima
- **Con:** One more hyper-parameter to tune
- **More advanced adaptive steps:** adagrad, adam
Wide or Deep?
Wide or Deep?

[Figure 6.6, Deep Learning, book]
Wide or Deep?

[Figure 6.7, Deep Learning, book]
Dropout

• Standard regularization technique
• At training drop a percentage of the units
  • Used for non-output layer
  • Prevents co-adaptation / Bagging
• At test: use the full network
• Normalize the weights

[Figure 7.6, Deep Learning]
Neural Network
Takeaways
Neural Networks takeaways

- Very flexible models
  - Composed of simple units (neurons)
  - Adapt to different types of data
- (Highly) non-linear models
  - E.g., Can learn to order/rank inputs easily
- Scale to very large datasets
- May require “fiddling” with model architecture + optimization hyper-parameters
  - Standardizing data can be very important
Where do NNs shine

- Input is high-dimensional discrete or real-valued
- Output is discrete or real valued, or a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human interpretability is not important
- The computation of the output based on the input has to be fast
Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples.

Other tasks, that cannot be described as associating one vector to another, or that are difficult enough that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now.

Deep Learning — Part II, p.163
http://www.deeplearningbook.org/contents/part_practical.html
Neural Networks in Practice
In practice

- Software now derives gradients automatically
  - You specify the architecture of the network
    - Connection pattern
    - Number of hidden layers
    - Number of layers
    - Activation functions
    - Learning rate (learning rate updates)
    - Dropout
- For intuitions: https://playground.tensorflow.org
A selection of standard tools (in python)

- Scikit-learn
  - Machine learning toolbox
    - Feed-forward neural networks
- Neural network specific tools
  - PyTorch, Tensorflow
  - Keras
- More specific tools for specific tasks:
  - caffe for computer vision, pySpark for distributed computations, NLTK for natural language processing