Machine Learning for Large-Scale Data Analysis and Decision Making

80-629-17A

Neural Networks
— Week #6
Today

• Neural Networks
• Modeling
• Fitting
• Deep neural networks

Today’s material is (adapted) from Joelle Pineau’s slides
Neural Networks

- Models behind deep learning
- Flexible model class
  - Highly-non linear models
- Good for regression/classification/density estimation
- Historical aspects
Recall Linear Classification
Recall Linear Classification

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

\[
\begin{align*}
(w^\top x + w_0) > 0 & \implies \text{Green point} \\
(w^\top x + w_0) < 0 & \implies \text{Blue point}
\end{align*}
\]
Recall Linear Classification

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

\[ (w^\top x + w_0) > 0 \implies \bullet \]

\[ (w^\top x + w_0) < 0 \implies \circ \]
What if data is not linearly separable?

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

Could we use the joint decision of several linear classifiers?

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

Could we use the joint decision of several linear classifier?

Exclusive OR (XOR)
Combining models

$$w^T x + w_0 > 0 \implies \bullet$$

$$w^T x + w_0 < 0 \implies \circ$$

$$w'^T x + w'_0 > 0 \implies \circ$$

$$w'^T x + w'_0 < 0 \implies \bullet$$
Combining models

Combining models

\[ (w^T x + w_0) > 0 \implies \quad f(x) = \bullet \text{ and } f'(x) = \bullet \implies \bullet \]

\[ (w^T x + w_0) < 0 \implies \quad f(x) = \bullet \text{ and } f'(x) = \bullet \implies \bullet \]

\[ (w'^T x + w'_0) > 0 \implies \quad f'(x) = \bullet \text{ and } f'(x) = \bullet \implies \bullet \]

\[ (w'^T x + w'_0) < 0 \implies \quad f(x) = \bullet \text{ and } f'(x) = \bullet \implies \bullet \]
Combining models

1. Evaluate each model
   - \( w^T x + w_0 > 0 \) \( \implies \) green
   - \( w^T x + w_0 < 0 \) \( \implies \) blue

2. Combine the output of models
   - \( f(x) = \text{green} \) and \( f'(x) = \text{blue} \) \( \implies \) green
Combining models

1. Evaluate each model

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

\[ (w'^T x + w'_0) > 0 \implies \bullet \]
\[ (w'^T x + w'_0) < 0 \implies \circ \]

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)

\[ f(x) \rightarrow f''(x) \]
\[ f'(x) \rightarrow f''(x) \]
Combining model (graphical view)

\[ f''(x) = f'(x) \cdot f(x) \]
Combining model (graphical view)

\[ f''(x) \]
Combining model (graphical view)

Neural Network

Perceptron/Neuron

\[ f(x) \]

\[ f'(x) \]

\[ f''(x) \]

\{ blue, green \}
Feed-forward neural network

- An input layer
  - Its size is the number of inputs + 1
- A set of hidden layer(s)
  - Its size is a hyper-parameter
- An output layer
  - Its size is the number of outputs
  - A scalar for simple classification problems
  - Can be a vector
Feed-forward neural network

- An input layer
  - Its size is the number of inputs + 1
- A set of hidden layer(s)
  - Its size is a hyper-parameter
- An output layer
  - Its size is the number of outputs
  - A scalar for simple classification problems
  - Can be a vector

Input Layer  Hidden Layer(s)  Output Layer

\[ X_1 \rightarrow \begin{array}{c} \text{Circle} \\ \text{Circle} \end{array} \rightarrow y \]

\[ X_2 \rightarrow \begin{array}{c} \text{Circle} \\ \text{Circle} \end{array} \rightarrow y \]

\[ 1 \rightarrow \begin{array}{c} \text{Circle} \\ \text{Circle} \end{array} \rightarrow y \]
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

- 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)
Compute a prediction (forward pass)

\[ x_1 \]
\[ x_2 \]
\[ 1 \]
\[ \sigma \left( \sum_i w_{i1} x_i \right) \]
\[ 2. \sigma \left( \sum_i w_{i2} o_1 i \right) \]

Input Layer  Hidden Layer(s)  Output Layer
Compute a prediction (forward pass)

\[ \sigma \left( \sum_{i} w_{i1} x_i \right) \]

\[ \sigma \left( \sum_{i} w_{i2} o_{1i} \right) \]
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.
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.

\[
\text{sigmoid}(a) := \frac{1}{1 - \exp(-a)}
\]

\[
\text{logit}(a) := \log \left( \frac{a}{1 - a} \right)
\]
Fit the parameters (w) (backward pass)

\[
\frac{\partial (y - \hat{y})^2}{\partial w_j} = \frac{\partial (y - f(\sum_i w_i o_i))^2}{\partial w_j} = \frac{\partial (y - f(\sum_{i'} w_{i'} f(\sum_j w_j x_j)_{i'}))^2}{\partial w_j}
\]
Fit the parameters \( (w) \)  
(backward pass)

\[ \frac{\partial (y - \hat{y})^2}{\partial w_j} = \frac{\partial (y - f(\sum_i w_i o_i))^2}{\partial w_j} = \frac{\partial (y - f(\sum_i w_i f(\sum_j w_j x_j)))^2}{\partial w_j} \]
Fit the parameters \( (w) \) (backward pass)

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

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

- In practice, the order of the computation is important.
Fit the parameters \( (w) \) (backward pass)

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

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

- In practice the order of the computation is important.

- For MLP you back-propagate the gradient starting from the output node(s) and heading toward the input.
Gradient descent

- No closed-form formula

- Repeat the following steps (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 unit

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.
Rectified Linear units (Relu)
- Can shut off units
- Efficient gradients
- Faster training (empirical)
Output units

- Easy to model different types/distributions of output variables

- Binary (Bernoulli)
  - Sigmoid unit
    \[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]

- Categorical (Multinoulli/Categorical)
  - Softmax units
    \[ \text{softmax}(a)_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \]

- Continuous (Gaussian)
  - Identity
    \[ \text{identity}(a) = a \]
Regularization

- Weight decay on the parameters
- L2 penalty on the parameters
- Early stopping of the optimization procedure
- Monitor the validation loss and terminate when no more improvements
- Number of hidden layers and hidden units per layer
Momentum

\[ w_{ij}^{t+1} = w_{ij}^t - (\alpha \nabla w_{ij}^t + \beta \nabla w_{ij}^{t-1}) \]

- A: Can allow you to jump over small local optima
- A: Goes faster through flat areas by using acquired speed
- D: Can also jump over global optima
- D: One more hyper-parameter to tune
- More advanced: adagrad, adam
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
- Training time is not important
- Form of target function is unknown
- Human readability of result is not important
- The computation of the output based on the input has to be fast
Neural Networks takeaways

- (Highly) non-linear models
  - Can learn to order/rank inputs easily
- Scales to very large datasets
- Very flexible model
  - Composed of units
  - Adapts to different types of data
- May require “fiddling” with model architecture + optimization hyper-parameters
  - Standardizing data can be very important
How do we solve more complex tasks?

- Complex task: “Rich input that maps to rich outputs”
- Perception (e.g., visual recognition, speech understanding)
- Related tasks: Language understanding, question/answering, dialogue
Neural networks

• Good properties for such tasks

• Flexible

• Can be combined

• Specific modules for specific subtasks that connect to others

• Can be fitted to large datasets

• Community bias
Deep neural networks

- Several layers of hidden nodes
- Parameters at different levels of representation
- Very young field with little theoretical understanding

Figure 1: We would like the raw input image to be transformed into gradually higher levels of representation, representing more and more abstract functions of the raw input, e.g., edges, local shapes, object parts, etc. In practice, we do not know in advance what the “right” representation should be for all these levels of abstractions, although linguistic concepts might help guessing what the higher levels should implicitly represent.
Different architectures

- General neural networks with many layers
- Specific architectures for specific problems
  - Images: Convolutional neural networks
  - Text (varying input sizes): Recurrent neural networks
One example: recurrent neural networks (RNNs)
One example: recurrent neural networks (RNNs)
One example: recurrent neural networks (RNNs)
One example: recurrent neural networks (RNNs)
One example:
recurrent neural networks (RNNs)
One example: recurrent neural networks (RNNs)

The Brown Dog

\[ y_{t+1} \]
One example: recurrent neural networks (RNNs)

\( y_{t-1} \quad y_t \quad y_{t+1} \)

The Brown Dog
RNNs for machine translation

[https://smerity.com/articles/2016/google_nmt_arch.html]
RNNs takeaways (I)

• Can be used to learn from varying-length input

• Typically for discrete data (e.g., words)

• Can be used both for predictions and for representations
RNNs takeaways (II)

• Can easily be used as modules inside more complex systems

• Current applications: Natural language understanding (Q&A, Dialogs), machine translation

• Very active research field

• Still difficult to learn very long sequences (reading a book)

• Not available in scikit-learn