Deep Learning and Neural Nets

Will Stanton
Data Science and Business Analytics Meetup, May 28, 2014
Return Path

- Worldwide leader in email intelligence
- Collect and aggregate enormous amounts of email data, including *raw text data*
- Help receivers improve spam filtering with whitelists, blacklist, reputation scoring
- Help senders improve their email sending program
- Great place to work!
Supervised/Unsupervised Learning

- **Supervised learning (classification/regression):** start with a *training set* of *labeled* observations \((x_1, y_1), \ldots, (x_N, y_N)\), where \(x\)'s are *inputs*, \(y\)'s are *outputs*
  - Create algorithm to “learn” pattern from observations to make predictions on new inputs (like generalized curve-fitting)
    - Example: given a *labeled* training set of pictures of dogs and other animals, create an algorithm to recognize pictures of dogs
- **Unsupervised learning (clustering, feature extraction, dimensionality reduction):** automatically find patterns, groupings, useful variables in an *unlabeled* dataset
  - Example: given an *unlabeled* set of pictures of animals, create an algorithm to automatically distinguish between different types of animals
- Supervised learning is typically easier
  - **BUT** you need a labeled training set, often a BIG one
How to train a learning algorithm

1. Initialize *parameters* of model
2. Run data through algorithm
3. Compute *cost function* based on model run-through
   a. Supervised learning: cost function computed based on how close model output is to training set labels
      i. Example: RMSE = \( \sqrt{\text{mean}((\text{outputs} - \text{labels})^2)} \)
   b. Unsupervised learning: usually problem-specific
      i. Example: how well-separated are the clusters?
4. Adjust the parameters to reduce the value of the cost function
   a. Usual method: gradient descent
5. Repeat the process until the cost function reaches a *threshold* (usually defined by the *gradient* of the cost function being *small*)
Gradient descent

- **Gradient descent**: adjust parameters of cost function according to direction of gradient
- **Gradient**: If $f$ is a function of $N$ parameters $w_1, w_2, \ldots, w_N$, then the gradient of $f$ is the vector $\text{grad } f = [ \partial f/\partial w_1, \ldots, \partial f/\partial w_N ]$, where $\partial f/\partial w_i$ is the rate of change of $f$ with respect to a change in the variable $w_i$ – called the partial derivative of $f$ with respect to $w_i$

Cost function at a minimum

$\text{grad } f = 0$ (does *not* work the other way!!)
What is a Neural Network?

- Artificial Neural Networks are machine-learning algorithms (loosely) based on the human brain
- Network of nodes ("neurons") that perform computations
- Can be used for supervised or unsupervised learning
A simple neural net

1. Inputs enter first hidden layer “neurons”, and are transformed by activation function, then
2. passed to second hidden layer, and transformed again by activation function, then passed to output layer
3. Cost function calculated based on outputs
4. Parameters of activation function adjusted to reduce value of cost function (gradient descent)
5. Process repeated until gradient reaches small enough tolerance

Inputs $x = (x_1, x_2, x_3)$

Output $y$
Activation and Cost Functions

- **Activation Function** (one at each node of each hidden layer):
  - **logistic function**: \( f(x) = \frac{1}{1 + \exp(-x)} \)
    - \( x = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \) (weighted sum of inputs from previous layer)
    - parameters: \( w_1, \ldots, w_n \)
  - **ReLU** (rectified linear units): approx. linear combo of logistic functions
    - prevents *local minimum problem* in gradient descent (more later)

- **Cost Function**:
  - Supervised learning (classification, regression): often \( \text{RMSE} = \sqrt{\text{mean}((\text{outputs} - \text{labels})^2)} \)
  - Unsupervised learning (clustering, feature extraction): problem specific
What is Deep Learning?

- *Deep learning* is a collection of methods based on training neural nets with *many* hidden layers
- Advantages:
  - State of the art for machine translation, image recognition, speech recognition tasks
  - Accurate at classification and regression (supervised learning) with much smaller *labeled* training sets than typical ML algorithms
  - Automatically *learns* useful features of dataset
- Disadvantages:
  - Slow to train
  - Prone to *overfitting*
  - Prone to *local minimum problem*
Feature learning

- A feature of a dataset is a carefully-selected combination of the original variables
  - Ex: edges or colors in a collection of nature pictures
  - Ex: noses or eyes in a collection of pictures of faces
  - Ex: meaningful, common phrases in a collection of documents
  - Ex: chords or repeated rhythms in a collection of songs

- ML algorithms work better when you feed in the right features to the training algorithm
- The problem: typically, humans have to engineer useful features by hand
- The solution: deep learning algorithms (just like our brain) learn useful features automatically
Layered feature representations

- In the human brain, images are represented as a *distributed, multi-layered, feature* representation
  - Humans see *features*, not just *pixels*

*Each layer is made up of combinations of the previous layer, using multiple nodes (distributed, multi-layered)*
One learning algorithm hypothesis

- Sight, hearing, touch all seem to use the same distributed multi-layered feature representation learning algorithm
- How do we know?
  - Experiments “rewiring” the vision and sound centers of animal brains
  - In humans: “seeing” with your tongue, feeling the direction North
- Deep learning neural nets perform well on image recognition, speech recognition, text processing, etc.
  - Same architecture, same algorithm, same results -- on different tasks!
How does deep learning work?

- Modeled after multi-layer, distributed feature representation in brain
- Multiple hidden layers in neural nets
  - Each layer learns a new feature representation of previous layer (ie. pixels -> edges -> shapes -> typical face types)
    - uses an autoencoder or Restricted Boltzmann Machine (more on this later)
  - Feature representation is learned one layer at a time, starting with the simplest representation (pixels -> edges)
    - called layerwise pre-training
Autoencoders

- An autoencoder is a neural network that attempts to learn an efficient, distributed, feature representation of its inputs
  - tries to learn a new encoding $c(x)$ of the input $x$
  - Cool aside: an autoencoder with a linear activation function does the same transformation as PCA
Layerwise pre-training

- Each hidden layer of a deep neural net is itself an autoencoder
- At each layer, a new representation of the inputs from previous layer is learned
- This automatically *learns* useful features from dataset
Why does deep learning work?

- The features deep-learning networks automatically learn are often *much* more useful than human-engineered features (and take *much* less work to create)
- Useful features means
  - Faster training time
  - Fewer examples needed for training (smaller *training set*)
  - Easier to recognize similar examples, distinguish different examples
- Ex: a child only needs to see a few trucks before learning the typical features of a truck
  - Can generalize the feature representation of “truck” to bulldozers and army tanks, and even see the relationship to planes or boats
  - Deep learning neural nets attempt to do essentially the same thing
The local minimum problem

- **grad** \( f = 0 \) does not imply \( f \) is at a *global* minimum!
- Why: a cost function could have multiple local minima
- Training can get *stuck* at a *local* minimum that is not a *global* minimum if the gradient gets really *small*, because:
  - parameters are adjusted less when the gradient is small
  - algorithm is stopped when the gradient reaches small enough *tolerance*
- Why is this bad?
  - Algorithm is not as accurate as it *could* be if the cost function is not as *low* as it could be
Deep learning is prone to *local minimum problem*:
- can get “stuck” at a low point of the gradient of the cost function

Problem: *logistic* activation function has very small gradient (rate of change) for small values of $x$

Solution: instead of using *tanh* function, use activation function called *ReLU*: “rectified linear units”

$$f(x) = \max(0, x)$$

Gradient of ReLU function does not get too small for small $x$
Overfitting

- **Overfitting**: *overfitting* occurs when a machine-learning algorithm fits “too well” to the training set, and does not generalize well to new data.
- **Example:**

\[
y = x + \varepsilon \\
\varepsilon \sim \mathcal{N}(0,1)
\]

Line plus random Gaussian noise.

The *real* pattern is just a straight line. An *overfit* ML algorithm learns a pattern that is simply *not there*!
Preventing overfitting

- Cause: too many variables for the dataset
  - Ex: Fitting a 100-variable linear model to a dataset with only 3 relevant variables
- Solution: choose a number of variables appropriate to modeling task
- Cause: modeling algorithm too complicated for the dataset
  - Ex: Fitting a complex deep neural net to a linear dataset
- Solution: Test multiple types of modeling algorithms. Select model hyperparameters (like number of nodes of a neural network or number of variables in a tree-based model) with a tuning grid
- Specific to neural nets: dropout methods
Dropout

- Deep learning neural nets prone to *overfitting*
  - May learn features that are *not* important
    - Ex: may learn the logo “Ford” if looking at lots of trucks

- Solution: *dropout*
  - Randomly leave out neurons on each training example during training

- Works by not adjusting parameters *too much* on any given training example
The state of the art

- For supervised learning problems, “traditional” deep learning (from 2006 up until a few years ago) used layerwise unsupervised pre-training
  - Using ReLU and Dropout, it is possible to train deep learning models faster and more accurately, *without* using unsupervised pre-training
    - Caveat: you typically need a *lot* of training data for this to work
- Deep learning at (very) large scale: lots of top experts have moved to industry to implement deep learning for *huge* data business problems
  - Geoffrey Hinton works at Google
  - Yann LeCun works at Facebook
  - Andrew Ng works at Baidu
- Deep learning is getting easier to implement
  - Better documentation, better software libraries, Amazon GPU clusters, etc.
When to use deep learning

- **When to use deep learning:**
  - Complex, cognitive tasks with latent *deep structure*
    - Machine translation, image recognition, speech recognition, feature selection from a complex dataset
- **When not to use deep learning:**
  - Typical machine-learning tasks *without* deep structure: risk of overfitting
    - Deep learning is still difficult to implement compared to simpler methods
Implementing Deep Learning

- Python libraries: *theano* and *pylearn2*
  - *theano*: a high-performance computing and computer-algebra system library
    - includes GPU computing functionality
  - *pylearn2*: contains methods for training deep neural nets (uses *theano* for computation)
- Check out [http://deeplearning.net](http://deeplearning.net)
Questions?

- Thanks for listening, and thanks for inviting me to speak!
- Find me on my personal website: [http://williamgstanton.com](http://williamgstanton.com)
- Connect with me on LinkedIn