Intro to Deep Learning

Nick Locascio
2016: year of deep learning

2016: The Year That Deep Learning Took Over the Internet
WIRED - Dec 25, 2016
The project is still in the early stages, but it hints at the widespread impact of deep learning over past year. In 2016, this very old but newly ...

Deep learning takes on physics

This Is Why A Computer Winning At Go Is Such A Big Deal
People didn’t think this would happen for at least 10 years; it’s a sign of how far artificial intelligence has come.

MIT 6.S191 | Intro to Deep Learning | IAP 2017
Kristen Stewart co-authored a paper on style transfer and the AI community lost its mind

Posted Jan 19, 2017 by John Monnes (@JohnMonnes)
Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Speech Synthesis
- Game Playing

... and many, many more
Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Speech Synthesis
- Game Playing

... and many, many more

ILSVRC top-5 error on ImageNet

AlexNet
Krizhevsky, Sutskever, Hinton 2012
Better than humans
Deep Learning Success

Image Classification
- **Machine Translation**
Speech Recognition
Speech Synthesis
Game Playing

... and many, many more
Deep Learning Success

Image Classification
Machine Translation
- Speech Recognition
Speech Synthesis
Game Playing

... and many, many more
Deep Learning Success

Image Classification
Machine Translation
Speech Recognition
*Speech Synthesis*
Game Playing

... and many, many more
Deep Learning Success

Image Classification
Machine Translation
Speech Recognition
Speech Synthesis
- Game Playing

... and many, many more
Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Speech Synthesis
- Game Playing
- ... and many, many more
6.S191 Goals

1. Fundamentals
2. Practical skills
3. Up to speed on current state of the field
4. Foster an open and collaborative deep learning community within MIT

Knowledge, intuition, know-how, and community to do deep learning research and development.
Class Information

- 1 week, 5 sessions
- P/F, 3 credits
- 2 TensorFlow Tutorials
  - In-class Monday + Tuesday
- 1 Assignment: (more info in a few slides)
Typical Schedule

- 10:30am-11:15am  Lecture #1
- 11:15am-12:00pm  Lecture #2
- 12:00pm-12:30pm  Coffee Break
- 12:30pm-1:30pm   Tutorial / Proposal Time
Assignment Information

1 Assignment, 2 options:
- Present a novel deep learning research idea or application
- OR
- Write a 1-page review of a deep learning paper
Option 1: Novel Proposal

- Proposal Presentation
  - Groups of 3 or 4
  - Present a novel deep learning research idea or application
  - 1 slide, 1 minute
  - List of example proposals on website: introtddeeplearning.com
  - Presentations on Friday
  - Submit groups by Wednesday 5pm to be eligible
  - Submit slide by Thursday 9pm to be eligible
Option 2: Paper Review

- Write a 1-page review of a deep learning paper
  - Suggested papers listed on website [introtodeeplearning.com](http://introtodeeplearning.com)
  - We will read + grade based on clarity of writing and technical communication of main ideas.
Class Support

- **Piazza**: [https://piazza.com/class/iwmlwep2fnd5uu](https://piazza.com/class/iwmlwep2fnd5uu)
- **Course Website**: [introtodeeplearning.com](http://introtodeeplearning.com)
- **Lecture slides**: [introtodeeplearning.com/schedule](http://introtodeeplearning.com/schedule)
- **Email us**: [introtodeeplearning-staff@mit.edu](mailto:introtodeeplearning-staff@mit.edu)
- **OH by request**
Our Fantastic Sponsors!

Google

NVIDIA

IBM Watson

amazon alexa
Why Deep Learning and why now?
Why Deep Learning?

- Hand-Engineered Features vs. Learned features
Why Now?

1. Large Datasets
2. GPU Hardware Advances + Price Decreases
3. Improved Techniques
Fundamentals of Deep Learning
The Perceptron

1. Invented in 1954 by Frank Rosenblatt
2. Inspired by neurobiology
The Perceptron

Inputs  Weights  Sum  Non-linearity

\[ \sum \text{inputs} \times \text{weights} \]

bias
Perceptron Forward Pass

\[ output = \]
Perceptron Forward Pass

\[
\text{output} = \sum_{i=0}^{N} x_i \cdot w_i
\]
Perceptron Forward Pass

\[
output = \left( \sum_{i=0}^{N} x_i \cdot w_i \right) + b
\]
Perceptron Forward Pass

\[
output = g\left(\sum_{i=0}^{N} x_i \cdot w_i + b\right)
\]
Perceptron Forward Pass

\[ \text{output} = g(XW + b) \]

\[ X = x_0, x_1, \ldots x_n \]

\[ W = w_0, w_1, \ldots w_n \]
Perceptron Forward Pass

\[ \text{output} = g(X W + b) \]

\[ X = x_0, x_1, \ldots x_n \]

\[ W = w_0, w_1, \ldots w_n \]
Sigmoid Activation

\[ output = g(XW + b) \]

\[ g(a) = \sigma(a) = \frac{1}{1 + e^{-a}} \]
Common Activation Functions

Sigmoid:
\[ f(x) = \frac{1}{1 + e^{-x}} \]

Tanh:
\[ \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \]

ReLU:
\[ f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \]
Importance of Activation Functions

- Activation functions add non-linearity to our network’s function
- Most real-world problems + data are non-linear
Perceptron Forward Pass

\[ output = g(XW + b) \]
Perceptron Forward Pass

\[ output = g(\text{inputs} \times \text{weights}) \]

\[
(2 \times 0.1) + \\
(3 \times 0.5) + \\
(-1 \times 2.5) + \\
(5 \times 0.2) + \\
(1 \times 3.0)
\]
Perceptron Forward Pass

\[ \text{output} = g(3.2) = \sigma(3.2) \]

\[ = \frac{1}{1 + e^{-3.2}} = 0.96 \]
How do we build neural networks with perceptrons?
Perceptron Diagram Simplified
Perceptron Diagram Simplified

- Inputs: $x_0, x_1, x_2, \ldots, x_n$
- Output: $o_0$
Multi-Output Perceptron

Input layer

\[ x_0, x_1, x_2, \ldots, x_n \]

Output layer

\[ o_0, o_1 \]
Multi-Layer Perceptron (MLP)
Multi-Layer Perceptron (MLP)
Deep Neural Network
Applying Neural Networks
Example Problem: Will my Flight be Delayed?
Example Problem: Will my Flight be Delayed?

Temperature: -20 F

Wind Speed: 45 mph
Example Problem: Will my Flight be Delayed?

[-20, 45]
Example Problem: Will my Flight be Delayed?
Example Problem: Will my Flight be Delayed?

[-20, 45]
Example Problem: Will my Flight be Delayed?

Predicted: 0.05
Actual: 1
Quantifying Loss

\[
\text{loss}(f(x^{(i)}; \theta), y^{(i)}))
\]

Predicted: 0.05
Actual: 1
Total Loss

\[
\text{total loss} := J(\theta) = \frac{1}{N} \sum_{i}^{N} \text{loss}(f(x^{(i)}; \theta), y^{(i)})
\]

**Input**

\[
[ \\
[-20, 45], \\
[80, 0], \\
[4, 15], \\
[45, 60], \\
]
\]

**Predicted**

\[
[ \\
0.05 \\
0.02 \\
0.96 \\
0.35 \\
]
\]

**Actual**

\[
[ \\
1 \\
0 \\
1 \\
1 \\
]
\]
Total Loss

\[
\text{total loss} := J(\theta) = \frac{1}{N} \sum_{i}^{N} \text{loss}(f(x^{(i)}; \theta), y^{(i)})
\]
Binary Cross Entropy Loss

\[
cross_{-}entropy(\theta) = \frac{1}{N} \sum_{i=1}^{N} y^{(i)} \log(f(x^{(i)}; \theta)) + (1 - y^{(i)}) \log(1 - f(x^{(i)}; \theta))
\]

**Input**

\[
[[-20, 45],
[80, 0],
[4, 15],
[45, 60]]
\]

**Predicted**

\[
[0.05, 0.02, 0.96, 0.35]
\]

**Actual**

\[
[1, 0, 1, 1]
\]

**Input**

\[
[
\begin{array}{c}
-20 \\
45 \\
80 \\
0 \\
4 \\
15 \\
45 \\
60
\end{array}
\]

**Predicted**

\[
\begin{array}{c}
0.05 \\
0.02 \\
0.96 \\
0.35
\end{array}
\]

**Actual**

\[
\begin{array}{c}
1 \\
0 \\
1 \\
1
\end{array}
\]
Mean Squared Error (MSE) Loss

\[
\text{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (f(x^{(i)}; \theta) - y^{(i)})^2
\]

Input

\[
\begin{bmatrix}
-20, 45, \\
80, 0, \\
4, 15, \\
45, 60,
\end{bmatrix}
\]

Predicted

Actual

\[
\begin{bmatrix}
10, \\
45, \\
100, \\
15
\end{bmatrix}
\]

\[
\begin{bmatrix}
40, \\
42, \\
110, \\
55
\end{bmatrix}
\]

\[
\begin{bmatrix}
10, \\
45, \\
100, \\
15
\end{bmatrix}
\]

\[
\begin{bmatrix}
40, \\
42, \\
110, \\
55
\end{bmatrix}
\]
Training Neural Networks
Training Neural Networks: Objective

\[ \arg g_\theta \min \frac{1}{N} \sum_{i}^{N} \text{loss}(f(x^{(i)}; \theta), y^{(i)}) \]
Training Neural Networks: Objective

\[
\arg g_\theta \min \frac{1}{N} \sum_{i}^{N} \text{loss}(f(x^{(i)}; \theta), y^{(i)})
\]

\[J(\theta)\]

loss function
Training Neural Networks: Objective

\[
\arg_{\theta} \min \frac{1}{N} \sum_{i}^{N} \text{loss}(f(x^{(i)}; \theta), y^{(i)})
\]

\[ J(\theta) \]

\[ \theta = W_1, W_2 \ldots W_n \]
Loss is a **function** of the model’s parameters
How to minimize loss?

Start at random point

\[ J(\theta) \]

\[ \theta_0 \quad \theta_1 \]
How to minimize loss?

Compute: \[ \frac{\partial J(\theta)}{\partial \theta} \]
How to minimize loss?

Move in direction opposite of gradient to new point
How to minimize loss?

Move in direction opposite of gradient to new point

\[ J(\theta) \]

\[ \theta_0 \]

\[ \theta_1 \]
How to minimize loss?

Repeat!
This is called Stochastic Gradient Descent (SGD)

Repeat!
Stochastic Gradient Descent (SGD)

- Initialize $\theta$ randomly
- For N Epochs
  - For each training example $(x, y)$:
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update $\theta$ with update rule:
      $$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$
Stochastic Gradient Descent (SGD)

- Initialize $\theta$ randomly
- For $N$ Epochs
  - For each training example $(x, y)$:
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update $\theta$ with update rule:
      \[
      \theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}
      \]
Stochastic Gradient Descent (SGD)

- Initialize $\theta$ randomly
- For N Epochs
  - For each training example $(x, y)$:
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update $\theta$ with update rule:
      $$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

- How to Compute Gradient?
Calculating the Gradient: Backpropagation
Calculating the Gradient: Backpropagation

\[
\frac{\partial J(\theta)}{\partial W_2} =
\]
Calculating the Gradient: Backpropagation

\[ \frac{\partial J(\theta)}{\partial W_2} = \]

Apply the chain rule.
Calculating the Gradient: Backpropagation

Apply the chain rule

\[
\frac{\partial J(\theta)}{\partial W_2} = \frac{\partial J(\theta)}{\partial o_0}
\]
Calculating the Gradient: Backpropagation

Apply the chain rule

\[
\frac{\partial J(\theta)}{\partial W_2} = \frac{\partial J(\theta)}{\partial o_0} \ast \frac{\partial o_0}{\partial W_2}
\]
Calculating the Gradient: Backpropagation

\[
\frac{\partial J(\theta)}{\partial W_1} =
\]
Calculating the Gradient: Backpropagation

Apply the chain rule

\[ \frac{\partial J(\theta)}{\partial W_1} = \]
Calculating the Gradient: Backpropagation

Apply the chain rule

\[
\frac{\partial J(\theta)}{\partial W_1} = \frac{\partial J(\theta)}{\partial o_0} \cdot \frac{\partial o_0}{\partial h_0}
\]
Calculating the Gradient: Backpropagation

\[ \frac{\partial J(\theta)}{\partial W_1} = \frac{\partial J(\theta)}{\partial o_0} \times \frac{\partial o_0}{\partial h_0} \]
Calculating the Gradient: Backpropagation

\[
\frac{\partial J(\theta)}{\partial W_1} = \frac{\partial J(\theta)}{\partial o_0} \times \frac{\partial o_0}{\partial h_0} \times \frac{\partial h_0}{\partial W_1}
\]

Apply the chain rule

\[
x_0 \quad W_1 \quad h_0 \quad W_2 \quad o_0 \quad J(\Theta)
\]
Training Neural Networks In Practice
Loss function can be difficult to optimize
Loss function can be difficult to optimize

Update Rule: \[ \theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta} \]
Loss function can be difficult to optimize

Update Rule:

\[ \theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta} \]
Learning Rate & Optimization

- Small Learning Rate

Small learning rate: Many iterations until convergence and trapping in local minima.
Learning Rate & Optimization

- Large learning rate

Large learning rate: Overshooting.
How to deal with this?

1. Try lots of different learning rates to see what is ‘just right’
How to deal with this?

1. Try lots of different learning rates to see what is ‘just right’
2. Do something smarter
How to deal with this?

1. Try lots of different learning rates to see what is ‘just right’
2. Do something smarter: Adaptive Learning Rate
Adaptive Learning Rate

- Learning rate is no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc
Adaptive Learning Rate Algorithms

- ADAM
- Momentum
- NAG
- Adagrad
- Adadelta
- RMSProp

Escaping Saddle Points
Escaping Saddle Points
Training Neural Networks In Practice 2: MiniBatches
Why is it **Stochastic** Gradient Descent?

- Initialize $\theta$ randomly
- For N Epochs
  - For each training example $(x, y)$:
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update $\theta$ with update rule:
      $$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

Only an estimate of true gradient!
Minibatches Reduce Gradient Variance

- Initialize $\theta$ randomly
- For N Epochs
  - For each training batch \{(x_0, y_0), ..., (x_B, y_B)\}:
    - Compute Loss Gradient: 
      \[
      \frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B} \sum_{i}^{B} \frac{\partial J_i(\theta)}{\partial \theta}
      \]
    - Update $\theta$ with update rule:
      \[
      \theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}
      \]

More accurate estimate!
Advantages of Minibatches

● More accurate estimation of gradient
  ○ Smoother convergence
  ○ Allows for larger learning rates

● Minibatches lead to fast training!
  ○ Can parallelize computation + achieve significant speed increases on GPU’s
Training Neural Networks In Practice 3: Fighting Overfitting
The Problem of Overfitting
Regularization Techniques

1. Dropout
2. Early Stopping
3. Weight Regularization
4. ...many more
Regularization I: Dropout

- During training, randomly set some activations to 0
Regularization I: Dropout

- During training, randomly set some activations to 0
Regularization I: Dropout

During training, randomly set some activations to 0
  Typically ‘drop’ 50% of activations in layer
  Forces network to not rely on any 1 node
Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node

MIT 6.S191 | Intro to Deep Learning | IAP 2017
Regularization II: Early Stopping

- Don’t give the network time to overfit
- ...
- **Epoch 15**: Train: 85% Validation: 80%
- **Epoch 16**: Train: 87% Validation: 82%
- **Epoch 17**: Train: 90% Validation: 85%
- **Epoch 18**: Train: 95% Validation: 83%
- **Epoch 19**: Train: 97% Validation: 78%
- **Epoch 20**: Train: 98% Validation: 75%
Regularization II: Early Stopping

- Don’t give the network time to overfit
- ...
- **Epoch 15**: Train: 85% Validation: 80%
- **Epoch 16**: Train: 87% Validation: 82%
- **Epoch 17**: Train: 90% Validation: 85%
- **Epoch 18**: Train: 95% Validation: 83%
- **Epoch 19**: Train: 97% Validation: 78%
- **Epoch 20**: Train: 98% Validation: 75%

Stop here!
Regularization II: Early Stopping

![Graph showing training and test set accuracy over epochs, with annotations indicating early stopping and overfitting.]

Stop here!
Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small
Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

\[
\arg_{\theta} \min \frac{1}{T} \sum_{t} \text{loss}(f(x^{(t)}; \theta), y^{(t)})
\]
Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

\[
\arg_{\theta} \min \frac{1}{T} \sum_t \text{loss}(f(x^{(t)}; \theta), y^{(t)}) + \lambda \sum_i (\theta_i)^2
\]
Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

\[
\arg_\theta \min \frac{1}{T} \sum_t \text{loss}(f(x^{(t)}; \theta), y^{(t)}) + \lambda \sum_i (\theta_i)^2
\]

\[
J(\theta)
\]

MIT 6.S191 | Intro to Deep Learning | IAP 2017
Core Fundamentals Review

- Perceptron Classifier
- Stacking Perceptrons to form neural networks
- How to formulate problems with neural networks
- Train neural networks with backpropagation
- Techniques for improving training of deep neural networks
Questions?