28: Intro to Deep Learning

Jerry Cain
June 1, 2022
Deep Learning
Innovations in deep learning

Deep learning (neural networks) is the core idea behind the current revolution in AI.

Errata:
- Checkers is the last solved game (from game theory, where perfect player outcomes can be fully predicted from any gameboard).
  https://en.wikipedia.org/wiki/Solved_game
- The first machine learning algorithm defeated a world champion in Chess in 1996.
Computers making art

The Next Rembrandt

A Neural Algorithm of Artistic Style
https://arxiv.org/abs/1508.06576
https://github.com/jcjohnson/neural-style

Google Deep Dream
https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html
Detecting skin cancer

Deep learning

**def Deep learning is**
maximum likelihood estimation with neural networks.

**def A neural network is**
(at its core) many logistic regression pieces stacked on top of each other.

[1, 0, ..., 1] 
\(x\), input  

\(\hat{y}\), output  
\(P(Y = 1|X = x) > 0.5?\)  
Yes. Predict 1  

Lots of Logistic (regressions)  

LOL
Let's focus on the model up to $\hat{y}$.
Let's focus on the model up to $\hat{y}$. 

$X \xrightarrow{\theta_0 + \sum_{j=1}^{m} \theta_j X_j} \hat{y} = P(Y = 1|X) \xrightarrow{\hat{y} = \text{arg max}_Y P(Y | X)}$ 

$y=\{0,1\}$
One neuron = One logistic regression
Biological basis for neural networks

A neuron

One neuron = one logistic regression

Your brain

Neural network = many logistic regressions

(Actually, probably someone else’s brain)
## Digit recognition example

<table>
<thead>
<tr>
<th>Input image</th>
<th>Input feature vector</th>
<th>Output label</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image0.png" alt="Input image 0" /></td>
<td>$x^{(i)} = [0,0,0,0, ..., 1,0,0,1, ..., 0,0,1,0]$</td>
<td>$y^{(i)} = 0$</td>
</tr>
<tr>
<td><img src="image1.png" alt="Input image 1" /></td>
<td>$x^{(i)} = [0,0,1,1, ..., 0,1,1,0, ..., 0,1,0,0]$</td>
<td>$y^{(i)} = 1$</td>
</tr>
</tbody>
</table>

We make feature vectors from (digitized) pictures of numbers.
Logistic Regression

\[ P(Y = 1 | X = x) \]

\( x \), input features
(pixels, on/off)
Logistic Regression

\( \hat{y}, \text{output} \)

\[ P(Y = 1|X = x) \]

\( x, \text{input features} \)
(pixels, on/off)

\( > 0.5? \)

No. Predict 0

indicates logistic regression connection
Logistic Regression

$x, \text{ input features}$

$\hat{y}, \text{ output}$

indicates logistic regression connection

$> 0.5? \Rightarrow \text{Yes. Predict 1}$
Logistic Regression

\[ \hat{y}, \text{output} \]

\[ \text{indicates logistic regression connection} \]

\[ x, \text{input features} \]

\[ \text{Yes. Predict 1} \]

\[ > 0.5? \]

What can we do to increase complexity of our model?
Take two big ideas from Logistic Regression

Big idea #1 \( P(Y|X = x) \)
Model conditional probability \( \hat{y} \) of class label given input

\( \sigma(\theta^T x) \)
Non-linear transform of multiple values into one value, using parameter \( \theta \)

\( x \), input features

\( \hat{y} \), output
Introducing: The Neural network

\[ x, \text{ input features} \rightarrow h, \text{ hidden layer} \rightarrow \hat{y}, \text{ output} \]

\( h > 0.5? \)

No. Predict 0

\[ \text{✅} \]
Neural network

$x$, input features

$h$, hidden layer

$\hat{y}$, output

Big idea #1  \[ P(Y|X = x) \]
Model conditional probability \( \hat{y} \) of class label given input

> 0.5?  
No.  
Predict 0
Feed neurons into other neurons

\[ h, \text{ hidden layer} \]

<table>
<thead>
<tr>
<th>Neuron = logistic regression [ \sigma(\theta^T x) ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-linear transform of multiple values into one value, using parameter ( \theta )</td>
</tr>
</tbody>
</table>

Big idea #2

\( \sigma(\theta^T x) \)

Non-linear transform of multiple values into one value, using parameter \( \theta \)

\( \hat{y}, \text{ output} \)

\( > 0.5? \) No. Predict 0

\( x, \text{ input features} \)
Feed neurons into other neurons

- Neuron = logistic regression
- Different parameters for every connection

$x$, input features

$h$, hidden layer

Another hidden neuron

$\sigma$, hidden layer

$\hat{y}$, output

$x_1, x_2, \ldots, x_n$
Feed neurons into other neurons

$x$, input features

$h$, hidden layer

$|h|$, logistic regression connections

$|x| \cdot |h|$, parameters

$\hat{y}$, output

- Neuron = logistic regression
- Different parameters for every connection

$x \cdot |h| > 0.5$?

No. Predict 0
Feed neurons into other neurons

$\mathbf{x}$, input features

$h$, hidden layer

$|h|$ logistic regression connections

$|x| \cdot |h|$ parameters

$\hat{y}$, output

output neuron

$\sigma$ (logistic regression)

$x \cdot |h|$ parameters

$h > 0.5$?

No. Predict 0

- Neuron = logistic regression
- Different parameters for every connection
Feed neurons into other neurons

\( \mathbf{x} \), input features

| \( \mathbf{h} \) | logistic regression connections \\
| \( |\mathbf{x}| \cdot |\mathbf{h}| \) parameters \\

\( \mathbf{h} \), hidden layer

| \( \mathbf{h} \) | parameters \\

1 logistic regression connection

\( \hat{y} \), output

\( > 0.5? \)

No. Predict 0

- Neuron = logistic regression
- Different parameters for every connection
Why doesn’t a linear model introduce “complexity”?

Neural network:

1. for \( j = 1, \ldots, |h| \):
   \[
   h_j = \sigma \left( \theta_j^{(h)^T} x \right)
   \]
2. \( \hat{y} = \sigma \left( \theta^{(\hat{\theta})^T} h \right) = P(Y = 1 | X = x) \)

 Linear network:

1. for \( j = 1, \ldots, |h| \):
   \[
   h_j = \theta_j^{(h)^T} x
   \]
2. \( \hat{y} = \sigma \left( \theta^{(\hat{\theta})^T} h \right) = P(Y = 1 | X = x) \)
Why doesn’t a linear model introduce “complexity”?

Neural network:
1. for $j = 1, \ldots, |h|$: 
   \[ h_j = \sigma \left( \theta_j^{(h)^T} x \right) \]
2. \[ \hat{y} = \sigma \left( \theta^{(\hat{\phi})^T} h \right) = P(Y = 1|X = x) \]

Linear network:
1. for $j = 1, \ldots, |h|$: 
   \[ h_j = \theta_j^{(h)^T} x \]
2. \[ \hat{y} = \sigma \left( \theta^{(\hat{\phi})^T} h \right) = P(Y = 1|X = x) \]

The linear model is effectively a single logistic regression with $|x|$ parameters.
Demonstration

https://www.cs.cmu.edu/~aharley/vis/conv/flat.html
Neural networks

A neural network (like logistic regression) gets intelligence from its parameters $\theta$.

**Training**
- Learn parameters $\theta$
- Find $\theta_{MLE}$ that maximizes likelihood of training data (MLE)

**Testing/Prediction**
For input feature vector $X = x$:
- Use parameters to compute $\hat{y} = P(Y = 1|X = x)$
- Classify instance as: $\begin{cases} 1 & \hat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases}$
Neural networks

A neural network (like logistic regression) gets intelligence from its parameters $\theta$.

- Learn parameters $\theta$
- Find $\theta_{MLE}$ that maximizes likelihood of training data (MLE)

How do we learn the $|x| \cdot |h| + |h|$ parameters?

Gradient ascent + chain rule!
Training: Logistic Regression

1. Optimization problem:

\[ \theta_{MLE} = \arg\max_{\theta} \prod_{i=1}^{n} f(y^{(i)}|x^{(i)}, \theta) = \arg\max_{\theta} LL(\theta) \]

\[ LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \]

\[ \hat{y} = \sigma(\theta^T x^{(i)}) = P(Y = 1|X = x) \]

2. Compute gradient

Find |x| parameters

3. Optimize

initialize params
repeat many times:
  compute gradient
  params += \eta * gradient
1. Optimization problem:

\[ \theta_{MLE} = \arg \max_\theta \prod_{i=1}^n f(y^{(i)} | x^{(i)}, \theta) = \arg \max_\theta LL(\theta) \]

2. Compute gradient

3. Optimize
1. Same output $\hat{y}$, same log conditional likelihood

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^{n} f(y^{(i)}|x^{(i)}, \theta) = \arg \max_{\theta} \mathcal{L}(\theta)$$

$$L(\theta) = \prod_{i=1}^{n} P(Y = y^{(i)}|X = x^{(i)}, \theta)$$

$$= \prod_{i=1}^{n} (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}}$$

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

for $j = 1, ..., |h|$: $h_{j} = \sigma \left( \theta^{(h)^{T}} x \right)$

$\hat{y} = \sigma \left( \theta^{(\gamma)^{T}} h \right) = P(Y = 1|X = x)$

Binary class labels: $Y \in \{0, 1\}$
(model is a little more complicated)

\[
\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^{n} f(y^{(i)} | x^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)
\]

\[
LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})
\]

To optimize for log conditional likelihood, we now need to find:

\[|h| \cdot |x| + |h| \quad \text{parameters}\]
2. **Compute gradient**

1. **Optimization problem:**

   \[ \theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^{n} f(y^{(i)} | x^{(i)}, \theta) = \arg \max_{\theta} LL(\theta) \]

   \[ LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \]

   \[ h_j = \sigma \left( \theta_j^{(h)} x \right) \quad \text{for } j = 1, ..., |h| \quad \hat{y} = \sigma \left( \theta^{(y)} h \right) \]

2. **Compute gradient**

   Take gradient with respect to all \( \theta \) parameters.

3. **Optimize**

   - **Calculus refresher #1:** Derivative(sum) = sum(derivative)
   - **Calculus refresher #2:** Chain rule 🌟🌟🌟
3. Optimize

1. Optimization problem:
   \[ \theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^{n} f(y^{(i)} | x^{(i)}, \theta) = \arg \max_{\theta} LL(\theta) \]
   \[ LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \]
   \[ h_j = \sigma \left( \theta_j^{(h)}^T x \right) \text{ for } j = 1, ..., |h| \]
   \[ \hat{y} = \sigma \left( \theta^{(\hat{y})}_T h \right) \]

2. Compute gradient
   Take gradient with respect to all \( \theta \) parameters

3. Optimize
   initialize params
   repeat many times:
   compute gradient
   params += \( \eta \) * gradient
Training a neural net

1. Optimization problem:
   \[ \theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^{n} f(y^{(i)} | x^{(i)}, \theta) = \arg \max_{\theta} LL(\theta) \]
   \[ LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \]

2. Compute gradient with respect to all \( \theta \) parameters

3. Optimize
   \[
   \text{initialize params} \\
   \text{repeat many times:} \\
   \text{compute gradient} \\
   \text{params} += \eta \times \text{gradient}
   \]

Wait, did we just skip something difficult?
2. **Compute gradient via backpropagation**

1. **Optimization problem:**

   \[
   \theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^{n} f(y^{(i)}|x^{(i)}, \theta) = \arg \max_{\theta} LL(\theta)
   \]

   \[
   LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})
   \]

   \[
   h_j = \sigma \left( \theta_j^T \right) \text{ for } j = 1, \ldots, |h| \\
   \hat{y} = \sigma \left( \hat{y}^T \right) T h)
   \]

2. **Compute gradient**

   Take gradient with respect to all \( \theta \) parameters

3. **Optimize**

   Learn the tricks behind **backpropagation** in CS229, CS231N, CS224N, etc.
Beyond the basics
Shared weights?

It turns out if you want to force some of your weights to be shared over different neurons, the math isn’t much harder.

**Convolution** is an example of such weight-sharing and is used a lot for vision (Convolutional Neural Networks, CNN).
Neural networks with multiple layers
Neurons learn features of the dataset

Neurons in later layers will respond strongly to high-level features of your **training data**.
If your training data is faces, you will get lots of face neurons.

If your training data is all of YouTube...
...you get a cat neuron.

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012
Hire the smartest people in the world

Invent cat detector
**Multiple outputs?**

**Softmax** is a generalization of the sigmoid function.

- **sigmoid**($z$): value in range $[0, 1]$
  - $z \in \mathbb{R}$:
    - $P(Y = 1|X = x) = \sigma(z)$
    - (equivalent: Bernoulli $p$)

- **softmax**($z$): $k$-dimensional values in range $[0,1]$ that add up to 1
  - $z \in \mathbb{R}^k$:
    - $P(Y = i|X = x) = \text{softmax}(z)_i$
    - (equivalent: Multinomial $p_1, ..., p_k$)
### Softmax test metric: Top-5 error

| $Y = y$ | $P(Y = y|X = x)$ |
|---------|-----------------|
| 5       | 0.14            |
| 8       | 0.13            |
| 7       | 0.12            |
| 2       | 0.10            |
| 9       | 0.10            |
| 4       | 0.09            |
| 1       | 0.09            |
| 0       | 0.09            |
| 6       | 0.08            |
| 3       | 0.05            |

**Top-5 classification error**

What % of datapoints did *not* have the correct class label in the top-5 predictions?
ImageNet classification

22,000 categories

14,000,000 images

Hand-engineered features (SIFT, HOG, LBP),
Spatial pyramid,
SparseCoding/Compression

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012
ImageNet classification challenge

22,000 categories
14,000,000 images

Hand-engineered features
(SIFT, HOG, LBP),
Spatial pyramid,
SparseCoding/Compression

1000 categories
1,200,000 images in train set
200,000 images in test set

smooth hammerhead, Sphyrna zygaena
smoothhound shark, Mustelus mustelus
smalleye hammerhead, Sphyrna tudes
shovelnose shark, bonnethead, bonnet shark, Sphyra tiburo
angel shark, angelfish, Squatina squatina, monkfish
electric ray, crampfish, numbfish, torpedo
cow-nosed sawfish, Pristis pectinatus
guitarfish
roughtail stingray, Dasyatis centoura
butterfly ray
eagle ray
spotted eagle ray, spotted ray, Aetobatus narinari
cownose ray, cow-nosed ray, Rhinoptera bonasus
manta, manta ray, devilfish
Atlantic manta, Manta birostris
devil ray, Mobula hypostoma
grey skate, gray skate, Raja batis
little skate, Raja erinacea
...

Le, et al., Building high-level features using large-scale unsupervised learning, ICML 2012
ImageNet challenge: Top-5 classification error

99.5%
Random guess

\[ P(\text{true class label not in 5 guesses}) = \binom{999}{5} / \binom{1000}{5} = \frac{995}{1000} \]
ImageNet challenge: Top-5 classification error

<table>
<thead>
<tr>
<th></th>
<th>99.5%</th>
<th>25.8%</th>
<th>5.1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random guess</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-Neural Networks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Humans (2014)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GoogLeNet (2015)</td>
<td>16.4%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(lower is better)

Russakovsky et al., ImageNet Large Scale Visual Recognition Challenge. IJCV 2015
Szegedy et al., Going Deeper With Convolutions. CVPR 2015
### ImageNet challenge: Top-5 classification error

<table>
<thead>
<tr>
<th>Method</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random guess</td>
<td>99.5%</td>
</tr>
<tr>
<td>Pre-Neural Networks</td>
<td>25.8%</td>
</tr>
<tr>
<td>Humans (2014)</td>
<td>5.1%</td>
</tr>
<tr>
<td>GoogLe Net (2015)</td>
<td>16.4%</td>
</tr>
<tr>
<td>SENet (2017)</td>
<td>2.25%</td>
</tr>
</tbody>
</table>

(lower is better)

---

Russakovsky et al., ImageNet Large Scale Visual Recognition Challenge. IJCV 2015
Szegedy et al., Going Deeper With Convolutions. CVPR 2015
GoogLeNet (2015)

1 Trillion Artificial Neurons
(btw human brains have 1 billion neurons)

Multiple, Multi class output
22 layers deep!

Szegedy et al., Going Deeper With Convolutions. CVPR 2015
### Speeding up gradient descent

** Initialize \( \theta_j = 0 \) for \( 0 \leq j \leq m \)

** Repeat many times: **

  ** Gradient update:**
  \[
  \text{gradient}[j] = 0 \text{ for } 0 \leq j \leq m
  \]

  ** For each training example \((x, y)\): **
  \[
  \text{compute gradient}
  \]

  ** Update rule:**
  \[
  \theta_j \leftarrow \eta \times \text{gradient}[j] \text{ for all } 0 \leq j \leq m
  \]

1. **What if we have 1,200,000 images in our training set?**
2. **How can we speed up the update?**

Our **batch gradient descent** (over the entire training set) will be slow + expensive.

1. **Use stochastic gradient descent** (randomly select training examples with replacement).
2. **Momentum update** (Incorporate “acceleration” or “deceleration” of gradient updates so far)
Good ML = Generalization

**Overfitting**

Fitting the training data too well, such that we lose generality of model for predicting new data.

- Perfect fit, but bad predictor for new data
- More general fit + better predictor for new data

**Dropout**

During training, randomly leave out some neurons each training step.

It will make your network more robust.
Making decisions?

Not everything is classification.

Deep Reinforcement Learning

Instead of having the output of a model be a probability, you make output an expectation.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html
Deep Reinforcement Learning

http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

Deep Mind Atari Games
Score compared to best human
The End