CS 343: Artificial Intelligence

Deep Learning II

Prof. Yuke Zhu — The University of Texas at Austin

[These slides based on those of Dan Klein, Pieter Abbeel, Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]
Neural Net Demo!

https://playground.tensorflow.org/
Neural Networks
Multi-class Logistic Regression

- special case of neural network

\[
P(y_1|x; w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}
\]

\[
P(y_2|x; w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}
\]

\[
P(y_3|x; w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}
\]
Deep Neural Network = Also learn the features!

\[ P(y_1|x; w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} \]

\[ P(y_2|x; w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} \]

\[ P(y_3|x; w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} \]
Deep Neural Network = Also learn the features!

\[ f_1(x) \]
\[ f_2(x) \]
\[ f_3(x) \]
\[ \cdots \]
\[ f_K(x) \]

\[ P(y_1|x; w) = e^{z_{OUT}^{(1)}} + e^{z_{OUT}^{(2)}} + e^{z_{OUT}^{(3)}} + \cdots \]
\[ P(y_2|x; w) = e^{z_{OUT}^{(1)}} + e^{z_{OUT}^{(2)}} + e^{z_{OUT}^{(3)}} + \cdots \]
\[ P(y_3|x; w) = e^{z_{OUT}^{(1)}} + e^{z_{OUT}^{(2)}} + e^{z_{OUT}^{(3)}} + \cdots \]

\[ g = \text{nonlinear activation function} \]

\[ z_i^{(k)} = g \left( \sum_{j} W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right) \]
Deep Neural Network = Also learn the features!

\[ P(y_1|x;w) = \text{softmax}(z^{(OUT)}_1) \]

\[ P(y_2|x;w) = \text{softmax}(z^{(OUT)}_2) \]

\[ P(y_3|x;w) = \text{softmax}(z^{(OUT)}_3) \]

\[ z^{(k)}_i = g\left(\sum_j W_{i,j}^{(k-1,k)} z^{(k-1)}_j \right) \]

\( g = \text{nonlinear activation function} \)
Common Activation Functions

**Sigmoid Function**

\[ g(z) = \frac{1}{1 + e^{-z}} \]

\[ g'(z) = g(z)(1 - g(z)) \]

**Hyperbolic Tangent**

\[ g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]

\[ g'(z) = 1 - g(z)^2 \]

**Rectified Linear Unit (ReLU)**

\[ g(z) = \max(0, z) \]

\[ g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases} \]
Training the deep neural network is just like logistic regression:

\[
\max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

just \( w \) tends to be a much, much larger vector 😊

→ just run gradient ascent
+ stop when log likelihood of hold-out data starts to decrease
Neural Networks Properties

- **Theorem (Universal Function Approximators).** A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- **Practical considerations**
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)
How about computing all the derivatives?

- Derivatives tables:

\[
\frac{d}{dx} (a) = 0
\]

\[
\frac{d}{dx} (x) = 1
\]

\[
\frac{d}{dx} (au) = a \frac{du}{dx}
\]

\[
\frac{d}{dx} (u + v - w) = \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx}
\]

\[
\frac{d}{dx} (uv) = u \frac{dv}{dx} + v \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \frac{u}{v} \right) = \frac{1}{v} \frac{du}{dx} - \frac{u}{v^2} \frac{dv}{dx}
\]

\[
\frac{d}{dx} (u^n) = n u^{n-1} \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \frac{1}{u} \right) = -\frac{1}{u^2} \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \frac{1}{u^n} \right) = -\frac{n}{u^{n+1}} \frac{du}{dx}
\]

\[
\frac{d}{dx} [f(u)] = \frac{d}{du} [f(u)] \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \ln u \right) = \frac{1}{u} \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \log_a u \right) = \frac{1}{u} \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( e^u \right) = e^u \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( a^u \right) = a^u \ln a \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( v^{-1} \right) = -v^{-2} \frac{dv}{dx}
\]

\[
\frac{d}{dx} \left( \sin u \right) = \cos u \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \cos u \right) = -\sin u \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \tan u \right) = \sec^2 u \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \cot u \right) = -\csc^2 u \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \sec u \right) = \sec u \tan u \frac{du}{dx}
\]

\[
\frac{d}{dx} \left( \csc u \right) = -\csc u \cot u \frac{du}{dx}
\]

[source: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/derfunc.html]
How about computing all the derivatives?

- But neural net $f$ is never one of those?
  - No problem: CHAIN RULE:

  \[ f(x) = g(h(x)) \]

  Then

  \[ f'(x) = g'(h(x))h'(x) \]

→ Derivatives can be computed by following well-defined procedures
Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function \( g(x,y,w) \)
  - Can automatically compute all derivatives w.r.t. all entries in \( w \)
  - This is typically done by caching info during forward computation pass of \( f \), and then doing a backward pass = “backpropagation”
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass

- Need to know this exists
- How this is done? -- outside of scope of CS343
Training a Network (setting weights)

$$z_i^{(k)} = g \left( \sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right)$$

$$g = \text{nonlinear activation function}$$
Training a Network

Key words:
• Forward
• Backwards
• Gradient
• Backprop

\[ P(y_1|x; w) = e^{z_1} \]

\[ P(y_2|x; w) = e^{z_2} \]

\[ P(y_3|x; w) = e^{z_3} \]

\[ g = \text{nonlinear activation function} \]
Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$.

Think of the function as a composition of many functions, use chain rule.
- Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.

\[ g = b + c \]
- $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
Suppose we have \( g(\mathbf{w}) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( \mathbf{w} = [2, 3] \).

Think of the function as a composition of many functions, use chain rule.

- Can use derivative chain rule to compute \( \frac{\partial g}{\partial w_1} \) and \( \frac{\partial g}{\partial w_2} \).

\[
g = b + c
\]

- \( \frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1 \)

\( b = a \times w_2 \)

- \( \frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} \)

\[
\begin{align*}
g &= b + c \\
b &= a \times w_2 \\
c &= 6
\end{align*}
\]
Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1$, $\frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a}$ = ???

Diagram:

\[ g = 30 \]
\[ \frac{\partial g}{\partial g} = 1 \]
\[ \frac{\partial g}{\partial c} = 1 \]
\[ \frac{\partial g}{\partial b} = 1 \]
\[ b = 24 \]
\[ \times \]
\[ a = 8 \]
\[ \times \]
\[ 3 \]
\[ c = 6 \]
\[ + \]
\[ w_1 \]
\[ 2 \]
\[ w_2 \]
\[ 3 \]
Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$.
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

Diagram:

- $a = 8$
- $b = 24$
- $c = 6$
- $g = 30$
- $\frac{\partial g}{\partial w_1} = 1$
- $\frac{\partial g}{\partial w_2} = 1$
- $\frac{\partial g}{\partial g} = 1$
Suppose we have \( g(w) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( w = [2, 3] \).

Think of the function as a composition of many functions, use chain rule.

- Can use derivative chain rule to compute \( \frac{\partial g}{\partial w_1} \) and \( \frac{\partial g}{\partial w_2} \).

\[
\begin{align*}
g &= b + c \\
\frac{\partial g}{\partial b} &= 1, \quad \frac{\partial g}{\partial c} = 1 \\
b &= a \times w_2 \\
\frac{\partial g}{\partial a} &= \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3 \\
a &= w_1^3 \\
\frac{\partial g}{\partial w_1} &= \text{?? ?? ?? ?}
\end{align*}
\]
Back Propagation: \( g(w) = w_1^3 w_2 + 3w_1 \)

- Suppose we have \( g(w) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( w = [2, 3] \).
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute \( \frac{\partial g}{\partial w_1} \) and \( \frac{\partial g}{\partial w_2} \).

\[
\begin{align*}
g & = b + c \\
\frac{\partial g}{\partial b} & = 1, \quad \frac{\partial g}{\partial c} = 1 \\
b & = a \times w_2 \\
\frac{\partial g}{\partial a} & = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3 \\
a & = w_1^3 \\
\frac{\partial g}{\partial w_1} & = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36
\end{align*}
\]

Interpretation: A tiny increase in \( w_1 \) will result in an approximately \( 36w_1 \) increase in \( g \) due to this cube function.
Suppose we have \( g(w) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( w = [2, 3] \).

Think of the function as a composition of many functions, use chain rule.

- Can use derivative chain rule to compute \( \partial g / \partial w_1 \) and \( \partial g / \partial w_2 \).

\[
\begin{align*}
g &= b + c \\
\frac{\partial g}{\partial b} &= 1, \quad \frac{\partial g}{\partial c} = 1 \\
b &= a \times w_2 \\
\frac{\partial g}{\partial a} &= \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3 \\
a &= w_1^3 \\
\frac{\partial g}{\partial w_1} &= \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36 \\
\frac{\partial g}{\partial w_2} &= ?? \quad \text{Hint: } b = a \times 3 \text{ may be useful.}
\end{align*}
\]
Back Propagation: $g(w) = w_1^3w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3w_2 + 3w_1$ and want the gradient at $w = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - Can use derivative chain rule to compute $\partial g/\partial w_1$ and $\partial g/\partial w_2$.

- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

\[
\begin{align*}
\frac{\partial g}{\partial w_1} &= 3 \cdot 3w_1^2 = 36 \\
\frac{\partial g}{\partial w_2} &= 1 \cdot 8 = 8 \\
\frac{\partial g}{\partial b} &= 1 \\
\frac{\partial g}{\partial c} &= 1
\end{align*}
\]
Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$

Think of the function as a composition of many functions, use chain rule.

$g = b + c$
- $\frac{\partial g}{\partial b} = 1$, $\frac{\partial g}{\partial c} = 1$

$b = a \times w_2$
- $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$

$a = w_1^3$
- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

$c = 3w_1$
- $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c} \frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$

How do we reconcile this seeming contradiction? Top partial derivative means cube function contributes $36w_1$ and bottom p.d. means product contributes $3w_1$ so add them.
Suppose we have \( g(w) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( w = [2, 3] \).

Think of the function as a composition of many functions, use chain rule.

- \( g = b + c \)
  - \( \frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1 \)

- \( b = a \times w_2 \)
  - \( \frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3 \)
  - \( \frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8 \)

- \( a = w_1^3 \)
  - \( \frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36 \)

- \( c = 3w_1 \)
  - \( \frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c} \frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3 \)

\[ \nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} & \frac{\partial g}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 39 & 8 \end{bmatrix} \]
Gradient Ascent

- **Punchline:** If we can somehow compute our gradient, we can use gradient ascent.
- **How do we compute the gradient?**
  - Purely analytically.
    - Gives exact symbolic answer. Infeasible for functions of lots of parameters or input values.
  - Finite difference approximation.
    - Gives approximation, very easy to implement.
    - Runtime for ll: $O(NM)$, where N is the number of parameters, and M is number of data points.
  - Back propagation.
    - Gives exact answer, difficult to implement.
    - Runtime for ll: $O(NM)$

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$
Summary of Key Ideas

- Optimize probability of label given input
  \[ \max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w) \]

- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = “early stopping”)

- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - \( \rightarrow \) the features are learned rather than hand-designed
  - Universal function approximation theorem
    - If neural net is large enough
    - Then neural net can represent any continuous mapping from input to output with arbitrary accuracy
    - But remember: need to avoid overfitting / memorizing the training data \( \rightarrow \) early stopping!
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 343)
Exercises: Deep Learning

Consider the following computation graph for a simple neural network for binary classification. Here \( x \) is a single real-valued input feature with an associated class \( y^* \) (0 or 1). There are two weight parameters \( w_1 \) and \( w_2 \), and non-linearity functions \( g_1 \) and \( g_2 \) (to be defined later, below). The network will output a value \( a_2 \) between 0 and 1, representing the probability of being in class 1. We will be using a loss function \( Loss \) (to be defined later, below), to compare the prediction \( a_2 \) with the true class \( y^* \).

\[
\begin{align*}
\mathbf{z}_1 &= x \cdot w_1 \\
\mathbf{a}_1 &= g_1(\mathbf{z}_1) \\
\mathbf{z}_2 &= \mathbf{a}_1 \cdot w_2 \\
\mathbf{a}_2 &= g_2(\mathbf{z}_2) \\
Loss(a_2, y^*) &= Loss(g_2(w_2 \cdot g_1(w_1 \cdot x)), y^*)
\end{align*}
\]

1. Perform the forward pass on this network, writing the output values for each node \( z_1, a_1, z_2 \) and \( a_2 \) in terms of the node’s input values:

2. Compute the loss \( Loss(a_2, y^*) \) in terms of the input \( x \), weights \( w_1 \), and activation functions \( g_i \):

3. Now we will work through parts of the backward pass, incrementally. Use the chain rule to derive \( \frac{\partial Loss}{\partial w_2} \). Write your expression as a product of partial derivatives at each node: i.e. the partial derivative of the node’s output with respect to its inputs. (Hint: the series of expressions you wrote in part 1 will be helpful; you may use any of those variables.)

4. Suppose the loss function is quadratic, \( Loss(a_2, y^*) = \frac{1}{2}(a_2 - y^*)^2 \), and \( g_1 \) and \( g_2 \) are both sigmoid functions \( g(z) = \frac{1}{1 + e^{-z}} \) (note: it’s typically better to use a different type of loss, cross-entropy, for classification problems, but we’ll use this to make the math easier).

Using the chain rule from Part 3, and the fact that \( \frac{\partial g(z)}{\partial z} = g(z)(1 - g(z)) \) for the sigmoid function, write \( \frac{\partial Loss}{\partial w_2} \) in terms of the values from the forward pass, \( y^* \), \( a_1 \), and \( a_2 \):
Computer Vision
Manual Feature Design
Features and Generalization

Image

HoG
Performance

ImageNet Error Rate 2010-2014

graph credit Matt Zeiler, Clarifai
Performance

ImageNet Error Rate 2010-2014

- Traditional CV
- Deep Learning

AlexNet

Graph credit: Matt Zeiler, Clarifai
Performance

ImageNet Error Rate 2010-2014

- Traditional CV
- Deep Learning

AlexNet

graph credit Matt Zeiler, Clarifai
Performance

ImageNet Error Rate 2010-2014

Error Rate

2010 2011 2012 2013 2014

79% 60% 40% 20% 7%

AlexNet

Traditional CV Deep Learning

graph credit Matt Zeiler, Clarifai
Speech Recognition

TIMIT Speech Recognition

- Traditional
- Deep Learning

Error Rate vs. Years

Graph credit: Matt Zeiler, Clarifai
Machine Translation

Google Neural Machine Translation (in production)

Encoder:
- $e_0$ -> $e_1$ -> $e_2$ -> $e_3$ -> $e_4$ -> $e_5$ -> $e_6$

Decoder:
- $d_0$ -> $d_1$ -> $d_2$ -> $d_3$

Input: 知识 = Knowledge
Output: is = is
Output: 力量 = power
Output: <end> = <end>