Machine Learning: Chenhao Tan

University of Colorado Boulder

LECTURE 14

Slides adapted from Jordan Boyd-Graber

Logistics

- HW3 available on Github, due on October 16
- Prelim 1 grades have been released
- Project team match

Outline

Revisiting Logistic Regression

Feed Forward Networks

Revisiting Logistic Regression

$$P(Y = 0 \mid \boldsymbol{x}, \beta) = \frac{1}{1 + \exp\left[\beta_0 + \sum_j \beta_j \boldsymbol{x}_j\right]}$$

$$P(Y = 1 \mid \boldsymbol{x}, \beta) = \frac{\exp\left[\beta_0 + \sum_j \beta_j \boldsymbol{x}_j\right]}{1 + \exp\left[\beta_0 + \sum_j \beta_j \boldsymbol{x}_j\right]}$$

$$\mathscr{L} = -\sum_i \log P(y^{(i)} \mid \boldsymbol{x}^{(i)}, \beta)$$

Revisiting Logistic Regression

- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0, 1\}$ to $\{1, 2\}$):

$$l_k = \beta_k^T \boldsymbol{x}, k = 1, 2$$

$$o_k = \frac{\exp l_k}{\sum_{c \in \{1,2\}} \exp l_c}, k = 1, 2$$

Revisiting Logistic Regression

- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0, 1\}$ to $\{1, 2\}$):

$$l_k = \beta_k^T \boldsymbol{x}, k = 1, 2 \quad \text{linear layer}$$

$$o_k = \frac{\exp l_k}{\sum_{c \in \{1,2\}} \exp l_c}, k = 1, 2 \quad \text{softmax layer}$$

Revisiting Logistic Regression

- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0, 1\}$ to $\{1, 2\}$):

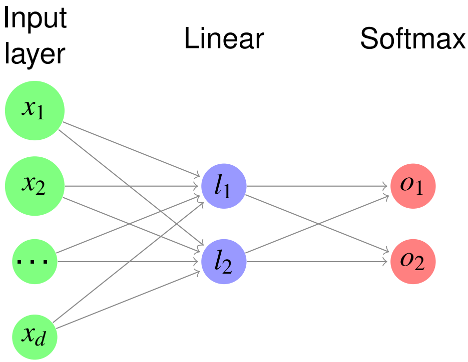
$$l_k = \beta_k^T \boldsymbol{x}, k = 1, 2 \quad \text{linear layer}$$

$$o_k = \frac{\exp l_k}{\sum_{c \in \{1,2\}} \exp l_c}, k = 1, 2 \quad \text{softmax layer}$$

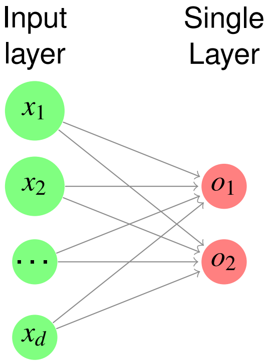
- Objective function (using cross entropy $-\sum_i p_i \log q_i$):

$$\mathscr{L}(Y, \hat{Y}) = -\sum_i \left[P(y^{(i)} = 1) \log P(\hat{y}_i = 1 \mid \boldsymbol{x}^{(i)}, \beta) + P(y^{(i)} = 0) \log \hat{P}(y_i = 0 \mid \boldsymbol{x}^{(i)}, \beta) \right]$$

Logistic Regression as a Single-layer Neural Network



Logistic Regression as a Single-layer Neural Network



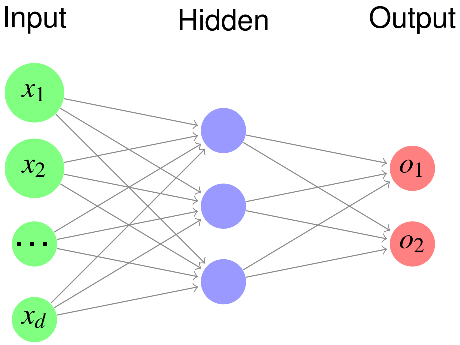
Outline

Revisiting Logistic Regression

Feed Forward Networks

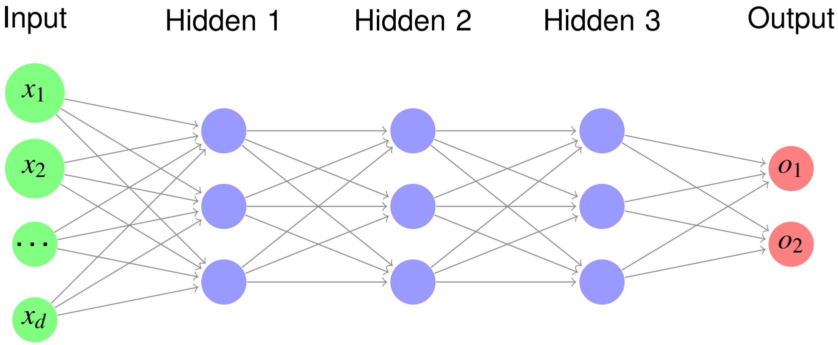
Deep Neural networks

A two-layer example (one hidden layer)



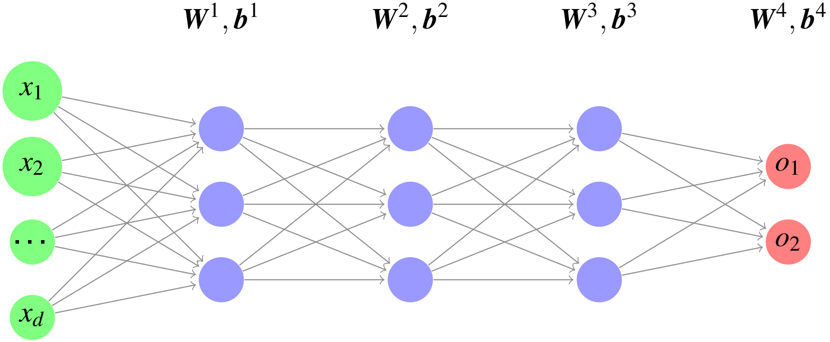
Deep Neural networks

More layers:



Forward propagation algorithm

How do we make predictions based on a multi-layer neural network?
Store the biases for layer l in $\boldsymbol{b}^l$, weight matrix in $\boldsymbol{W}^l$



Forward propagation algorithm

Suppose your network has L layers

Make prediction for an instance $\boldsymbol{x}$

1: Initialize $\boldsymbol{a}^0 = \boldsymbol{x}$
2: **for** $l = 1$ to L **do**
3: $\quad \boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$
4: $\quad \boldsymbol{a}^l = g(\boldsymbol{z}^l)$
5: **end for**
6: The prediction $\hat{y}$ is simply $\boldsymbol{a}^L$

Nonlinearity

What happens if there is no nonlinearity?

Nonlinearity

What happens if there is no nonlinearity?
Linear combinations of linear combinations are still linear combinations.

Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_w(\boldsymbol{x})$$

- Loss function (objective function)

$$\mathscr{L}(y, \hat{y})$$

- Learning (next week)

Nonlinearity Options

- Sigmoid

$$f(x) = \frac{1}{1 + \exp(x)}$$

- tanh

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- ReLU (rectified linear unit)

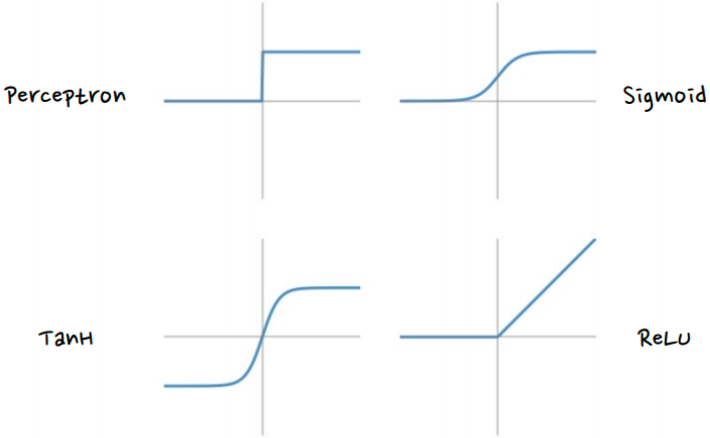
$$f(x) = \max(0, x)$$

- softmax

$$\boldsymbol{x} = \frac{\exp(\boldsymbol{x})}{\sum_{x_i} \exp(x_i)}$$

`https://keras.io/activations/`

Nonlinearity Options



Loss Function Options

- ℓ_2 loss

$$\sum_i (y_i - \hat{y}_i)^2$$

- ℓ_1 loss

$$\sum_i |y_i - \hat{y}_i|$$

- Cross entropy (logistic regression)

$$-\sum_i y_i \log \hat{y}_i$$

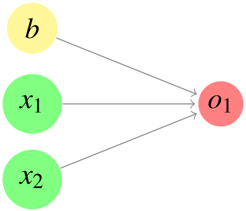
- Hinge loss (more on this during SVM)

$$\max(0, 1 - y\hat{y})$$

`https://keras.io/losses/`

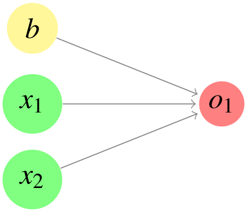
A Perceptron Example

$$\boldsymbol{x} = (x_1, x_2), y = f(x_1, x_2)$$



A Perceptron Example

$$\boldsymbol{x} = (x_1, x_2), y = f(x_1, x_2)$$



We consider a simple activation function

$$f(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

A Perceptron Example

Simple Example: Can we learn `OR`?

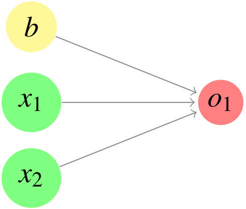
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

A Perceptron Example

Simple Example: Can we learn OR?

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

$$\boldsymbol{w} = (1, 1), b = -0.5$$



A Perceptron Example

Simple Example: Can we learn `AND`?

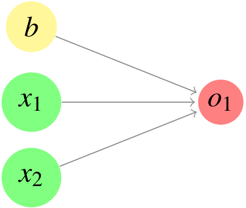
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \land x_2$	0	0	0	1

A Perceptron Example

Simple Example: Can we learn `AND`?

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \wedge x_2$	0	0	0	1

$$\boldsymbol{w} = (1, 1), b = -1.5$$



A Perceptron Example

Simple Example: Can we learn `NAND`?

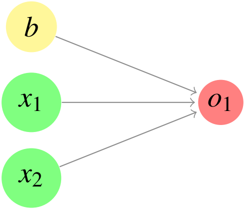
x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	1	1	0

A Perceptron Example

Simple Example: Can we learn `NAND`?

x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	1	1	0

$$\boldsymbol{w} = (-1, -1), b = 1.5$$



A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!

A Perceptron Example

Simple Example: Can we learn `XOR`?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!
But why?

A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!
But why?
The single-layer perceptron is just a linear classifier, and can only learn things that are linearly separable.

A Perceptron Example

Simple Example: Can we learn XOR?

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

NOPE!
But why?
The single-layer perceptron is just a linear classifier, and can only learn things that are linearly separable.
How can we fix this?

A Perceptron Example

Increase the number of layers.

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

A Perceptron Example

Increase the number of layers.

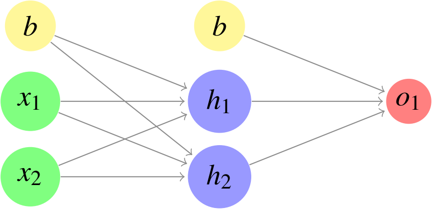
x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

$$\mathrm{XOR} = \mathrm{AND}(\mathrm{OR}, \mathrm{NAND})$$

A Perceptron Example

Increase the number of layers.

x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0



$$\boldsymbol{W}^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \boldsymbol{b}^1 = \begin{bmatrix} -0.5 \\ 1.5 \end{bmatrix}$$

$$\boldsymbol{W}^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{b}^2 = -1.5$$

General Expressiveness of Neural Networks

Neural networks with a single hidden layer can approximate any measurable functions [Hornik et al., 1989, Cybenko, 1989].

Summary

- Logistic regression and perceptron can be seen as special cases of neural networks
- Feed-forward algorithm (forward propagation)

References

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.