Dodowhile

Department of Computer Science
UNIVERSITY OF COLORADO **BOULDER**

Machine Learning: Yoshinari Fujinuma

University of Colorado Boulder

LECTURE 14

Slides adapted from Chenhao Tan, Jordan Boyd-Graber

Logistics

- HW3 will be available on Github next Monday, due on 3/15
- Project team formulation due on 3/1
  - Create a Piazza post with team members names

Learning Objective

- Introducing neural networks

Outline

From Logistic Regression & Perceptron to Neural Networks

Feed Forward Neural Networks

Logistic Regression as a Single-layer Neural Network

- Multinomial regression ouputs the probability of $\boldsymbol{x}$ being class k i.e., $P(Y = k \mid \boldsymbol{x})$

Logistic Regression as a Single-layer Neural Network

- Multinomial regression ouputs the probability of $\boldsymbol{x}$ being class k i.e., $P(Y = k \mid \boldsymbol{x})$
- Let $P(Y = k \mid \boldsymbol{x}) = o_k$

Logistic Regression as a Single-layer Neural Network

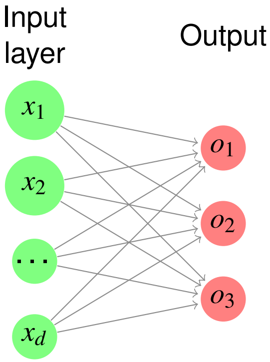
- Multinomial regression ouputs the probability of $\boldsymbol{x}$ being class k i.e., $P(Y = k \mid \boldsymbol{x})$
- Let $P(Y = k \mid \boldsymbol{x}) = o_k$
- Logistic regression for k classes ($k = 1, 2, 3$) can be separated out into two components:

$$o_k = f(\beta_k^T \boldsymbol{x}) \quad \text{output layer}$$

where f is

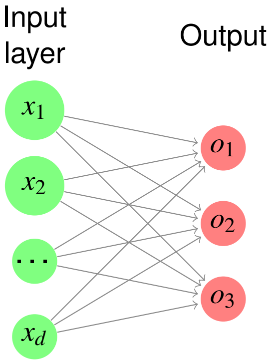
$$f(\beta_k^T \boldsymbol{x}) = \frac{\exp(\beta_k^T \boldsymbol{x})}{\sum_{c \in \{1,2,3\}} \exp(\beta_c^T \boldsymbol{x})} \quad \text{softmax activation function}$$

Logistic Regression as a Single-layer Neural Network



$$
\begin{aligned}
o_1 &= \text{softmax}(\beta_1^T \boldsymbol{x}) \\
o_2 &= \text{softmax}(\beta_2^T \boldsymbol{x}) \\
o_3 &= \text{softmax}(\beta_3^T \boldsymbol{x})
\end{aligned}
$$

Logistic Regression as a Single-layer Neural Network



$$
\begin{aligned}
o_1 &= \text{softmax}(\beta_1^T \boldsymbol{x}) \\
o_2 &= \text{softmax}(\beta_2^T \boldsymbol{x}) \\
o_3 &= \text{softmax}(\beta_3^T \boldsymbol{x})
\end{aligned}
$$

If we vertically stack β_k and define a parameter matrix W_{logistic} i.e.,

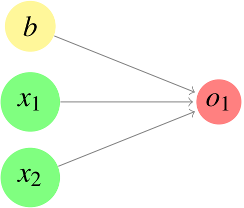
$$
W_{\text{logistic}} = \begin{pmatrix} \beta_1^T \\ \beta_2^T \\ \beta_3^T \end{pmatrix}
$$

then we can write the output as

$$
O = W_{\text{logistic}} \boldsymbol{x}
$$

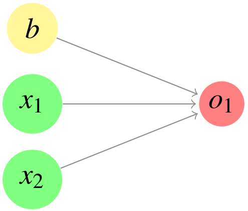
A Perceptron Example

$$\boldsymbol{x} = (x_1, x_2), y = f(\mathbf{w}\boldsymbol{x} + b)$$



A Perceptron Example

$\boldsymbol{x} = (x_1, x_2), y = f(\mathbf{w}\boldsymbol{x} + b)$



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`?

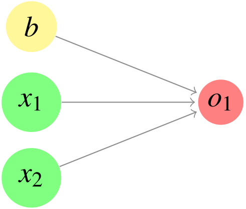
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

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`?

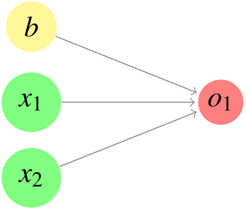
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \wedge x_2$	0	0	0	1

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`?

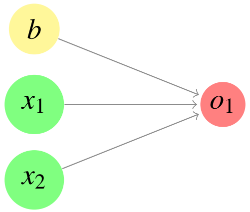
x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	1	1	0

A Perceptron Example

Simple Example: Can we learn `NAND`?

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

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



A Perceptron Example

Simple Example: Can we learn XOR? (Question 2 from Quiz 1)

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? (Question 2 from Quiz 1)

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? (Question 2 from Quiz 1)

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? (Question 2 from Quiz 1)

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? (Question 2 from Quiz 1)

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.

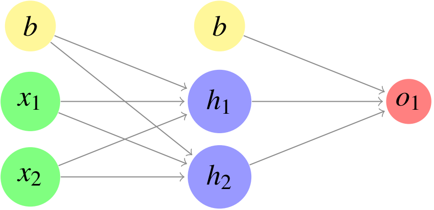
x_1	0	1	0	1
x_2	0	0	1	1
x_1 XOR x_2	0	1	1	0

$$\texttt{XOR} = \texttt{AND}\left(\texttt{OR}, \texttt{NAND}\right)$$

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



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

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

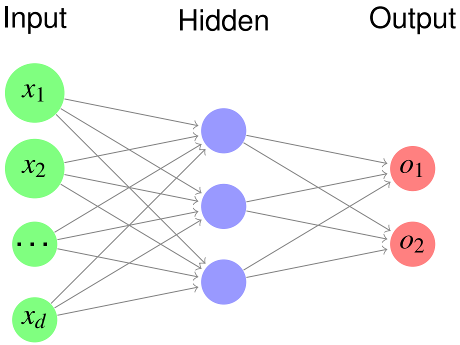
Outline

From Logistic Regression & Perceptron to Neural Networks

Feed Forward Neural Networks

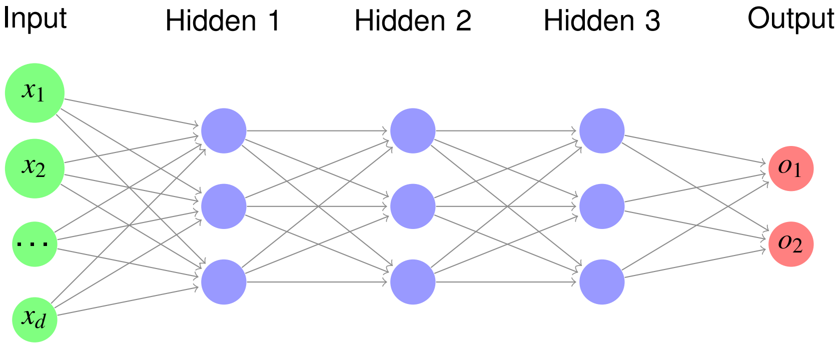
Multi-layer Perceptrons

A two-layer example (one hidden layer)



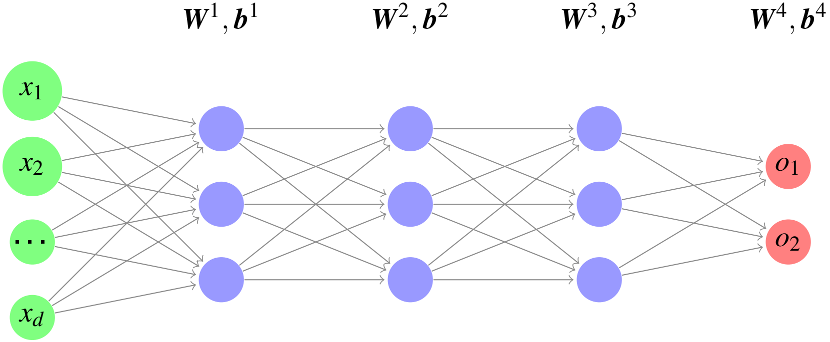
Multi-layer Perceptrons

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 $\boldsymbol{x}$

```
Initialize a^0 = x

for l = 1 to L do
    z^l = W^l a^{l-1} + b^l
    a^l = g(z^l)
end for
The prediction ŷ is simply 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.

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)

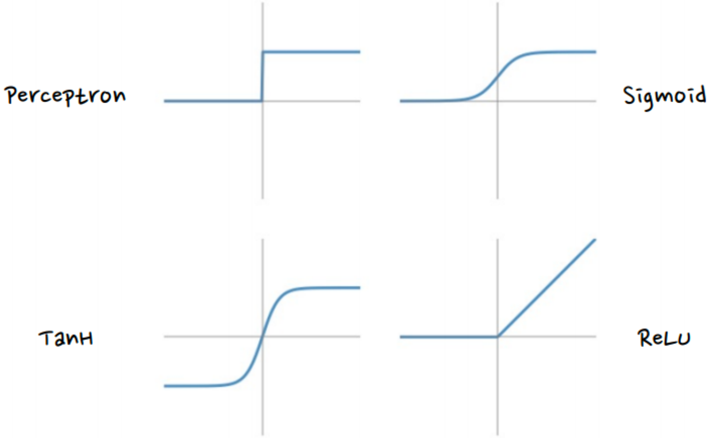
$$f(x) = \max(0, x)$$

- softmax

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

`https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity`

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://pytorch.org/docs/stable/nn.html#loss-functions`

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)

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

- Training (next week)

Summary

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

References