Deep Learning for Graphics

Machine Learning Introduction

Niloy Mitra
UCL

Iasonas Kokkinos
UCL/Facebook

Paul Guerrero
UCL

Vladimir Kim
Adobe Research

Kostas Rematas
U Washington

Tobias Ritschel
UCL
Course Overview

• Part I: Introduction and ML Basics

• Part II: Supervised Neural Networks: Theory and Applications

• Part III: Unsupervised Neural Networks: Theory and Applications

• Part IV: Beyond Image Data
Machine Learning
Machine Learning

**Machine learning** is a field of **computer science** that uses statistical techniques to give computer systems the ability to **learn** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

Machine Learning

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to learn (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.

Machine Learning

**Machine learning** is a field of **computer science** that uses statistical techniques to give computer systems the ability to **learn** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

Machine Learning Variants

• **Supervised**
  • Classification
  • Regression
  • Data consolidation

• **Unsupervised**
  • Clustering
  • Dimensionality Reduction

• **Weakly supervised/semi-supervised**
  Some data supervised, some unsupervised

• **Reinforcement learning**
  Supervision: sparse reward for a sequence of decisions
Machine Learning Variants

- **Supervised**
  - Classification
  - Regression
  - Data consolidation

- **Unsupervised**
  - Clustering
  - Dimensionality Reduction

- **Weakly supervised/semi-supervised**
  - Some data supervised, some unsupervised

- **Reinforcement learning**
  - Supervision: sparse reward for a sequence of decisions
Classification Examples

• Digit Recognition
Classification Examples

• Digit Recognition

• Spam Detection
Classification Examples

- Digit Recognition
- Spam Detection
- Face detection
Segmentation + Classification in Real Images
Segmentation + Classification in Real Images
Segmentation + Classification in Real Images

Evaluation measures: Confusion matrix, ROC curve, precision, recall, etc.
`Faceness’ Function: Classifier

background

face

Course: “Deep Learning for Graphics”
`Faceness’ Function: Classifier

background

decision boundary

face
Face Detection

Course: “Deep Learning for Graphics”
Face Detection

Course: “Deep Learning for Graphics”
Machine Learning Variants

- **Supervised**
  - Classification
  - **Regression**
  - Data consolidation

- **Unsupervised**
  - Clustering
  - Dimensionality Reduction

- **Weakly supervised/semi-supervised**
  - Some data supervised, some unsupervised

- **Reinforcement learning**
  - Supervision: sparse reward for a sequence of decisions
Human Face/Pose Estimation

[Blanz and Vetter, Siggraph, 1999]
Human Face/Pose Estimation

[Blanz and Vetter, Siggraph, 1999]
Regression: Model Estimation

[Mitra et al. SoCG, 2003]

[Guennebaud et al., Siggraph, 2007]

[Zwicker et al., EGSR, 2005]
Machine Learning Variants

- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**
  Some data supervised, some unsupervised
- **Reinforcement learning**
  Supervision: sparse reward for a sequence of decisions
Clustering: Color Points According to X
Clustering: Color Points According to X
Clustering: Color Points According to X
Clustering: Color Points According to X
Clustering: Color Points According to $X$

$L = 5$
Clustering: Color Points According to X

$L = 20$
Clustering Examples: Image Segmentation using NCuts
Clustering Examples

[Chu et al., TVCG, 2009]  [Zheng et al., Eurographics, 2014]
Machine Learning Variants

• **Supervised**
  • Classification
  • Regression
  • Data consolidation

• **Unsupervised**
  • Clustering
    • **Dimensionality Reduction**

• **Weakly supervised/semi-supervised**
  Some data supervised, some unsupervised

• **Reinforcement learning**
  Supervision: sparse reward for a sequence of decisions
Dimensionality Reduction (Manifold Learning)

Isomap

[Tenenbaum et al., Science, 2000]

Face Manifold

[Yang et al., TOG, 2011]

[Averkiou et al., Eurographics, 2014]
Example of Nonlinear Manifold: Faces

\[ X_1 \]

\[ X_2 \]
Example of Nonlinear Manifold: Faces

\[ x_1 \]

\[ \frac{1}{2}(x_1 + x_2) \]

\[ x_2 \]
Example of Nonlinear Manifold: Faces

\[ x_1 \]

\[ \frac{1}{2}(x_1 + x_2) \]

\[ x_2 \]
Morphing (Interpolation in Shape Space)

[Kilian et al., Siggraph, 2007]
Morphing (Interpolation in Shape Space)

[Kilian et al., Siggraph, 2007]
Moving Along Learned Face Manifold

Trajectory along the “male” dimension

[Lample et. al. Fader Networks, NIPS 2017]
Moving Along Learned Face Manifold

Trajectory along the “male” dimension

Trajectory along the “young” dimension

[Lample et. al. Fader Networks, NIPS 2017]
Notations: Vectors and Matrices
Notations: Vectors and Matrices

\[ \text{vector } x \]
Notations: Vectors and Matrices

- **vector**: $\mathbf{x}$
- **matrix**: $A_{m \times n} = [a_1 \ldots a_n]$
Notations: Vectors and Matrices

vector \( \mathbf{x} \)

matrix \( \mathbf{A}_{m \times n} = [\mathbf{a}_1 \ldots \mathbf{a}_n] \)

linear equation \( \mathbf{A}\mathbf{x} = \mathbf{b} \)
Notations: Vectors and Matrices

vector \( x \)

matrix \( A_{m \times n} = [a_1 \ldots a_n] \)

linear equation \( Ax = b \)

inner prod. \( \langle x, y \rangle = x^T y \)
Notations: Vectors and Matrices

vector \( \mathbf{x} \)

matrix \( A_{m \times n} = [a_1 \ldots a_n] \)

linear equation \( A\mathbf{x} = \mathbf{b} \)

inner prod. \( \langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} \quad \| \mathbf{x} \| = \sqrt{\mathbf{x}^T \mathbf{x}} \quad \mathbf{x}^T \mathbf{y} = \| \mathbf{x} \| \| \mathbf{y} \| \cos(\theta) \)
Notations: Vectors and Matrices

- **linear independence**; **rank** of a matrix
- **span** of a matrix

Vector: $x$

Matrix: $A_{m \times n} = [a_1 \ldots a_n]$

Linear equation: $Ax = b$

Inner product:

$$< x, y > = x^T y$$

$$\|x\| = \sqrt{x^T x}$$

$$x^T y = \|x\| \|y\| \cos(\theta)$$
Notations: Vectors and Matrices (cont.)
Notations: Vectors and Matrices (cont.)

\[ \| \mathbf{x} \|_p = (|x_1|^p + |x_2|^p + \ldots)^{1/p} \]
Notations: Vectors and Matrices (cont.)

\[ \| \mathbf{x} \|_p = \left( |x_1|^p + |x_2|^p + \ldots \right)^{1/p} \]

\[ \| \mathbf{x} \|_p = \max\{|x_1|, |x_2|, \ldots\} \quad p = \infty \]
Notations: Vectors and Matrices (cont.)

\[ \|x\|_p = (|x_1|^p + |x_2|^p + \ldots)^{1/p} \]

\[ \|x\|_p = \max\{|x_1|, |x_2|, \ldots\} \quad p = \infty \]

\[ L_1, L_2, L_p, L_\infty \]
Notations: Vectors and Matrices (cont.)

\[ \|x\|_p = (|x_1|^p + |x_2|^p + \ldots)^{1/p} \]

\[ \|x\|_p = \max\{|x_1|, |x_2|, \ldots\} \quad p = \infty \]

range \quad \mathcal{R}(A) = \{Ax : x \in \mathbb{R}^n\}
Notations: Vectors and Matrices (cont.)

\[ \|x\|_p = (|x_1|^p + |x_2|^p + \ldots)^{1/p} \]

\[ \|x\|_p = \max\{|x_1|, |x_2|, \ldots\} \quad p = \infty \]

\[ L_1, L_2, L_p, L_{\infty} \]

range \quad \mathcal{R}(A) = \{Ax : x \in \mathbb{R}^n\}

null space \quad \mathcal{N}(A) = \{x \in \mathbb{R}^n : Ax = 0\}
Eigenvectors and Eigenvalues

\[ y = Ax \]
Eigenvectors and Eigenvalues

\[ y = Ax \]

\[ Ae_i = \lambda_i e_i \]
Eigenvectors and Eigenvalues

\[ y = Ax \]

\[ Ae_i = \lambda_i e_i \]

\[ T = [v_1 \ v_2 \ldots] \]

\[ T^{-1} A T = \text{diag}(\lambda_1, \lambda_2, \ldots) \]
Eigenvectors and Eigenvalues

\[ y = Ax \]

\[ A e_i = \lambda_i e_i \]

\[ T = [v_1 \ v_2 \ldots] \]

\[ T^{-1} A T = \text{diag}(\lambda_1, \lambda_2, \ldots) \]

• All eigenvalues of symmetric matrices are real.
• Any real symmetric nxn matrix has a set of \( n \) mutually orthogonal eigenvectors.
Code Example

![Plot](image.png)
Code Example
Code Example

```python
import numpy as np

rng = np.random.RandomState(10)
X = np.dot(rng.randn(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0] - 1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```
Code Example

```python
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T
mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```
Code Example

```python
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T
mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```
Morphable Faces
Morphable Faces
Singular Value Decomposition (SVD)

• Very useful for matrix manipulation.
• Used for robust numerical computation.
Singular Value Decomposition (SVD)

• Very useful for matrix manipulation.
• Used for robust numerical computation.

\[ A = U \Sigma V^T \]
Singular Value Decomposition (SVD)

• Very useful for matrix manipulation.
• Used for robust numerical computation.

\[
A = U \Sigma V^T
\]
Singular Value Decomposition (SVD)

- Very useful for matrix manipulation.
- Used for robust numerical computation.

\[ A = U \Sigma V^T \]

scaling

rotation
Singular Value Decomposition (SVD)

• Very useful for matrix manipulation.
• Used for robust numerical computation.

\[ A = U \Sigma V^T \]

scaling rotation

\[ A = A^T = U \Sigma U^T \]
Code Example

Course: “Deep Learning for Graphics”
Differentiation (chain rule recap)
Differentiation (chain rule recap)
Differentiation (chain rule recap)
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]

\[ z = f(y) \]

\[ y = g(x) \]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]
\[ \frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \]

\[ z = f(y) \]
\[ y = g(x) \]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]
\[ z = f(y) \]
\[ y = g(x) \]
\[
\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x)
\]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]

\[ \frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x) \]

\[ z = f(y) \]

\[ y = g(x) \]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]

\[ z = f(y) \]

\[ y = g(x) \]

\[ z = \sin(5x) \]

\[
\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)
\]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]
\[ z = f(y) \]
\[ y = g(x) \]

\[ z = \sin(5x) \]
\[ = \frac{d \sin(5x)}{d(5x)} \cdot \frac{d(5x)}{dx} \]

\[ \frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x) \]
Differentiation (chain rule recap)

\[ z = f \circ g(x) = f(g(x)) \]
\[ z = f(y) \]
\[ y = g(x) \]

\[ z = \sin(5x) \]
\[ = \frac{d \sin(5x)}{d(5x)} \cdot \frac{d(5x)}{dx} \]
\[ = 5 \cos(5x) \]

\[ \frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x) \]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ f(x) = \begin{bmatrix}
  f_1(x) \\
  \vdots \\
  f_m(x)
\end{bmatrix} \]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \]

\[ \frac{\partial f(x)}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_j} \\ \vdots \\ \frac{\partial f_m(x)}{\partial x_j} \end{bmatrix} \]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix}, \quad \frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_j} \\ \vdots \\ \frac{\partial f_m(x)}{\partial x_j} \end{bmatrix} \]

\[ L = Df = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \cdots & \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[
f(x) = \begin{bmatrix}
f_1(x) \\
\vdots \\
f_m(x)
\end{bmatrix}
\]

\[
\frac{\partial f(x)}{\partial x_j} = \begin{bmatrix}
\frac{\partial f_1(x)}{\partial x_j} \\
\vdots \\
\frac{\partial f_m(x)}{\partial x_j}
\end{bmatrix}_{m \times 1}
\]

\[
L = Df = \begin{bmatrix}
\frac{\partial f(x)}{\partial x_1} & \ldots & \frac{\partial f(x)}{\partial x_n}
\end{bmatrix}
\]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \quad \frac{\partial f(x)}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_j} \\ \vdots \\ \frac{\partial f_m(x)}{\partial x_j} \end{bmatrix}_{m \times 1} \]

\[ L = Df = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \ldots & \frac{\partial f(x)}{\partial x_n} \end{bmatrix}_{m \times n} \]
Derivative Matrix

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \quad \frac{\partial f(x)}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_j} \\ \vdots \\ \frac{\partial f_m(x)}{\partial x_j} \end{bmatrix}_{m \times 1} \]

Jacobian matrix

\[ L = Df = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \cdots & \frac{\partial f(x)}{\partial x_n} \end{bmatrix}_{m \times n} \]
Regression: Continuous Output
Regression: Continuous Output
Learning a Function

\[ y = f_w(x) \]
Learning a Function

\[ y = f_w(x) \]

Calculus
\[ x \in \mathbb{R} \]

Vector calculus
\[ x \in \mathbb{R}^d \]
Learning a Function

\[ y = f_w(x) \]

- **Prediction**
- **Method**
- **Parameters**
- **Input**

Calculus

\[ x \in \mathbb{R} \]

Vector calculus

\[ \mathbf{x} \in \mathbb{R}^d \]

*Machine learning: can work also for discrete inputs, strings, images, meshes, animations, ...*
Learning a Function

\[ y = f_w(x) \]

- **Prediction**: \( y \) is the predicted output.
- **Method**: Function \( f_w \) with parameters \( w \).
- **Input**: \( x \) is the input data.
- **Parameters**: \( w \) are the learnable parameters.

**Calculus**

- \( x \in \mathbb{R} \)

**Vector calculus**

- \( \mathbf{x} \in \mathbb{R}^d \)

**Classification**

- \( y \in \{0, 1\} \)

**Machine learning**: can work also for discrete inputs, strings, images, meshes, animations, ...
Learning a Function

\[ y = f_w(x) \]

Calculus

\[ x \in \mathbb{R} \]

Vector calculus

\[ x \in \mathbb{R}^d \]

Classification:

\[ y \in \{0, 1\} \]

Regression:

\[ y \in \mathbb{R} \]

*Machine learning: can work also for discrete inputs, strings, images, meshes, animations, ...*
Learning a Simple Separator/Classifier

separating hyperplane
Learning a Simple Separator/Classifier

\[ x_1 \quad \rightarrow \quad x_2 \quad \rightarrow \quad \text{separating hyperplane} \]
Learning a Simple Separator/Classifier

![Diagram showing a simple separator/classifier](image)

- $y$
- $x_1$
- $x_2$

separating hyperplane
Learning a Simple Separator/Classifier

\[ y = w_1 x_1 + w_2 x_2 \]

separating hyperplane
Learning a Simple Separator/Classifier

\[ y = f(w_1 x_1 + w_2 x_2) \]
Learning a Simple Separator/Classifier

\[ y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2) \]
Learning a Simple Separator/Classifier

\[ y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2) \]
Learning a Simple Separator/Classifier

\[ y = f(w_1 x_1 + w_2 x_2) = H(w_1 x_1 + w_2 x_2) \]
Combining Simple Functions/Classifiers

2 layers of trainable weights

convex region
Combining Simple Functions/Classifiers

3 layers of trainable weights

complex polygons

Course: “Deep Learning for Graphics”
Learning a Function: Modeling

\[ y = f_w(x) = f(x; w) \]

\[ w \in \mathbb{R} \]

\[ \mathbf{w} \in \mathbb{R}^K \]
Regression

1. Least Squares fitting

2. Nonlinear error function and gradient descent

3. Perceptron training
Regression

1. Least Squares fitting

2. Nonlinear error function and gradient descent

3. Perceptron training
Assumption: Linear Function

\[ y = f_w(x) = f(x, w) = w^T x \]
Assumption: Linear Function

\[ y = f_w(x) = f(x, w) = w^T x \]

\[ w^T x = \langle w, x \rangle = \sum_{d=1}^{D} w_d x_d \]

\[ x \in \mathbb{R}^D, \ w \in \mathbb{R}^D \]
Reminder: Linear Classifier

feature coordinate

feature coordinate
Reminder: Linear Classifier

Course: “Deep Learning for Graphics”
Reminder: Linear Classifier

\[ x_i \text{ positive: } x_i \cdot w \geq 0 \]
\[ x_i \text{ negative: } x_i \cdot w < 0 \]
Reminder: Linear Classifier

\[ x_i \text{ positive: } x_i \cdot w \geq 0 \]
\[ x_i \text{ negative: } x_i \cdot w < 0 \]

supervised setting

labelled input

\[ y_t = \begin{cases} 
+1 & \text{if } x_i \cdot w \geq 0 \\
-1 & \text{if } x_i \cdot w < 0 
\end{cases} \]
Which Line to Pick?

\[ x_i \text{ positive: } x_i \cdot w \geq 0 \]
\[ x_i \text{ negative: } x_i \cdot w < 0 \]

**supervised setting**

labelled input

\[ y_t = \begin{cases} 
  +1 & \text{red} \\
  -1 & \text{blue} 
\end{cases} \]
Linear Regression in 1D

Training set: input–output pairs

\[ S = \{(x^i, y^i)\}, \quad i = 1 \ldots, N \]

\[ x^i \in \mathbb{R}, \quad y^i \in \mathbb{R} \]
Linear regression in 1D

\[ y^i = w_0 + w_1 x_1^i + \epsilon^i \]
Linear regression in 1D

\[ y^i = w_0 + w_1 x_1^i + \epsilon^i \]

\[ = w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i \]
Linear regression in 1D

\[ y^i = w_0 + w_1 x^i_1 + \epsilon^i \]

\[ = w_0 x^i_0 + w_1 x^i_1 + \epsilon^i, \quad x^i_0 = 1, \quad \forall i \]
Linear regression in 1D

\[
y^i = w_0 + w_1 x_1^i + \epsilon^i
\]

\[
= w_0 x_0^i + w_1 x_1^i + \epsilon^i, \quad x_0^i = 1, \quad \forall i
\]

\[
= w^T x^i + \epsilon^i
\]
Linear regression in 1D

\[ y^i = w_0 + w_1 x^i_1 + \epsilon^i \]

\[ = w_0 x^i_0 + w_1 x^i_1 + \epsilon^i, \quad x^i_0 = 1, \quad \forall i \]

\[ = w^T x^i + \epsilon^i \]

\( w_0 \) bias

noise
Sum of Square Errors \textit{(MSE without the mean)}

\[ y^i = w^T x^i + \epsilon^i \]
Sum of Square Errors (MSE without the mean)

\[ y^i = w^T x^i + \epsilon^i \]

Loss function: sum of squared errors

\[ L(w) = \sum_{i=1}^{N} (\epsilon^i)^2 \]
Sum of Square Errors \((MSE \ without \ the \ mean)\)

\[ y^i = w^T x^i + \epsilon^i \]

Loss function: sum of squared errors

\[ L(w) = \sum_{i=1}^{N} (\epsilon^i)^2 \]

In two variables:

\[ L(w_0, w_1) = \sum_{i=1}^{N} [y^i - (w_0 x_0^i + w_1 x_1^i)]^2 \]
**Sum of Square Errors** (*MSE without the mean*)

\[ y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i \]

Loss function: sum of squared errors

\[ L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2 \]

In two variables:

\[ L(w_0, w_1) = \sum_{i=1}^{N} [y^i - (w_0 x_0^i + w_1 x_1^i)]^2 \]

**Question:** what is the best (or least bad) value of \( \mathbf{w} \)?
Calculus 101

$f(x)$

Course: “Deep Learning for Graphics”
Calculus 101

\[ f(x) \]

\[ x^* = \arg\max_x f(x) \]
Local Extrema Condition

\[ f(x) \]

\[ x^* = \arg\max_x f(x) \]
Local Extrema Condition

\[ f(x) \]

\[ x^* = \text{argmax}_x f(x) \rightarrow f'(x^*) = 0 \]
Local Extrema Condition

\[ x^* = \arg\max_x f(x) \quad \rightarrow \quad f'(x^*) = 0 \]
Vector Calculus 101

\[ f(x) \]

2D function graph
Vector Calculus 101

\[ f(x) \]

2D function graph

\[ f(x) = c \]

isocontours
Vector Calculus 101

$f(x)$

$2D$ function graph

$f(x) = c$

isocontours

$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$

gradient field
Vector Calculus 101

$f(x)$

$f(x) = c$

$\nabla f(x) = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right]$

2D function graph

isocontours

gradient field

● at minimum of function: $\nabla f(x) = 0$
Vector Calculus 101

\[ f(\mathbf{x}) \]

\[ f(\mathbf{x}) = c \]

\[ \nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]

2D function graph  
isocontours  
gradiant field

\[ \bullet \text{ at minimum of function: } \nabla f(\mathbf{x}) = 0 \]
Line Fitting

\[ L(w_0, w_1) = \sum_{i=1}^{N} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2 \]
Line Fitting

\[ L(w_0, w_1) = \sum_{i=1}^{N} [y^i - (w_0 x_0^i + w_1 x_1^i)]^2 \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [y^i - (w_0 x_0^i + w_1 x_1^i)]^2}{\partial w_0} \]
Line Fitting

\[ L(w_0, w_1) = \sum_{i=1}^{N} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2 \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial}{\partial w_0} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2 \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} \]
Line Fitting

\[ L(w_0, w_1) = \sum_{i=1}^{N} [y^i - (w_0 x_0^i + w_1 x_1^i)]^2 \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [y^i - (w_0 x_0^i + w_1 x_1^i)]^2}{\partial w_0} \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z_i]^2}{\partial z_i} \frac{\partial z_i}{\partial w_0} \]
Line Fitting

\[ L(w_0, w_1) = \sum_{i=1}^{N} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2 \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial}{\partial w_0} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2 \]

\[ \frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0} \]

\[ z^i = y^i - (w_0 x_0^i + w_1 x_1^i) \]
Line Fitting

\[
L(w_0, w_1) = \sum_{i=1}^{N} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2
\]

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [y^i - (w_0 x_0^i + w_1 x_1^i)]^2}{\partial w_0}
\]

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0} = \sum_{i=1}^{N} (2z^i)(-x_0^i)
\]

\[
z^i = y^i - (w_0 x_0^i + w_1 x_1^i)
\]
### Line Fitting

Let's consider the problem of fitting a line to a set of data points. The goal is to find the parameters $w_0$ and $w_1$ such that the line $y = w_0x_0 + w_1x_1$ minimizes the sum of squared errors.

The cost function $L(w_0, w_1)$ is defined as:

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[ y^i - (w_0x_0^i + w_1x_1^i) \right]^2$$

The gradient of the cost function with respect to $w_0$ is:

$$\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [z^i]^2}{\partial z^i} \frac{\partial z^i}{\partial w_0} = \sum_{i=1}^{N} (2z^i)(-x_0^i) = -2 \sum_{i=1}^{N} (y^i - (w_0x_0^i + w_1x_1^i))x_0^i$$

where $z^i = y^i - (w_0x_0^i + w_1x_1^i)$.

The gradient with respect to $w_1$ follows a similar pattern.
Line Fitting (continued)
Line Fitting (continued)

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial}{\partial w_0} \left[ y^i - \left( w_0 x_0^i + w_1 x_1^i \right) \right]^2
\]
Line Fitting (continued)

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial}{\partial w_0} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2
\]

\[
= -2 \sum_{i=1}^{N} (y^i x_0^i - w_0 x_0^i x_0^i - w_1 x_1^i x_0^i)
\]
Line Fitting (continued)

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial [y^i - (w_0 x_0^i + w_1 x_1^i)]^2}{\partial w_0} \\
= -2 \sum_{i=1}^{N} (y^i x_0^i - w_0 x_0^i x_0^i - w_1 x_1^i x_0^i)
\]

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = 0
\]
Line Fitting (continued)

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = \sum_{i=1}^{N} \frac{\partial}{\partial w_0} \left[ y^i - (w_0 x_0^i + w_1 x_1^i) \right]^2
\]

\[
= -2 \sum_{i=1}^{N} \left( y^i x_0^i - w_0 x_0^i x_0^i - w_1 x_1^i x_0^i \right)
\]

\[
\frac{\partial L(w_0, w_1)}{\partial w_0} = 0
\]

\[
\sum_{i=1}^{N} y^i x_0^i = w_0 \sum_{i=1}^{N} x_0^i x_0^i + w_1 \sum_{i=1}^{N} x_1^i x_0^i
\]
Line Fitting (continued)

\[
\sum_{i=1}^{N} y^i x_0^i = w_0 \sum_{i=1}^{N} x_0^i x_0^i + w_1 \sum_{i=1}^{N} x_1^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i = w_0 \sum_{i=1}^{N} x_0^i x_1^i + w_1 \sum_{i=1}^{N} x_1^i x_1^i
\]
Line Fitting (continued)

\[
\begin{align*}
\sum_{i=1}^{N} y^i x_0^i &= w_0 \sum_{i=1}^{N} x_0^i x_0^i + w_1 \sum_{i=1}^{N} x_1^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i &= w_0 \sum_{i=1}^{N} x_0^i x_1^i + w_1 \sum_{i=1}^{N} x_1^i x_1^i
\end{align*}
\]

2x2 system of equations

\[
\begin{bmatrix}
\sum_{i=1}^{N} y^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i
\end{bmatrix} = \begin{bmatrix}
\sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\
\sum_{i=1}^{N} x_1^i x_0^i & \sum_{i=1}^{N} x_1^i x_1^i
\end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}
\]
Line Fitting (continued)

$$\begin{bmatrix}
\sum_{i=1}^{N} y^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i
\end{bmatrix}
= \begin{bmatrix}
\sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\
\sum_{i=1}^{N} x_1^i x_0^i & \sum_{i=1}^{N} x_1^i x_1^i
\end{bmatrix}
\begin{bmatrix}
w_0 \\
w_1
\end{bmatrix}$$
Line Fitting (continued)

\[
\begin{bmatrix}
\sum_{i=1}^{N} y^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i
\end{bmatrix}
= \begin{bmatrix}
\sum_{i=1}^{N} x_0^i x_0^i \\
\sum_{i=1}^{N} x_0^i x_1^i \\
\sum_{i=1}^{N} x_1^i x_0^i \\
\sum_{i=1}^{N} x_1^i x_1^i
\end{bmatrix}
\begin{bmatrix}
w_0 \\
w_1
\end{bmatrix}
\]

\[X^T y = X^T X w\]

\[
y = \begin{bmatrix}
y^1 \\
\vdots \\
y^N
\end{bmatrix}
\quad X = \begin{bmatrix}
x_0^1 & x_1^1 \\
x_0^N & x_2^N \\
x_0^N & x_2^N
\end{bmatrix}
\]
Line Fitting (continued)

\[
\begin{bmatrix}
\sum_{i=1}^{N} y^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i \\
\end{bmatrix}
= \begin{bmatrix}
\sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\
\sum_{i=1}^{N} x_1^i x_0^i & \sum_{i=1}^{N} x_1^i x_1^i \\
\end{bmatrix}
\begin{bmatrix}
w_0 \\
w_1 \\
\end{bmatrix}
\]

\[X^T y = X^T X w\]

\[y = \begin{bmatrix}
y^1 \\
\vdots \\
y^N \\
\end{bmatrix}
\quad X = \begin{bmatrix}
x_0^1 & x_1^1 \\
\vdots & \vdots \\
x_0^N & x_2^N \\
\end{bmatrix}\]

\[w = (X^T X)^{-1} X^T y\]
Line Fitting (continued)

\[
\begin{bmatrix}
\sum_{i=1}^{N} y^i x_0^i \\
\sum_{i=1}^{N} y^i x_1^i
\end{bmatrix}
= \begin{bmatrix}
\sum_{i=1}^{N} x_0^i x_0^i & \sum_{i=1}^{N} x_0^i x_1^i \\
\sum_{i=1}^{N} x_1^i x_0^i & \sum_{i=1}^{N} x_1^i x_1^i
\end{bmatrix}
\begin{bmatrix}
w_0 \\
w_1
\end{bmatrix}
\]

\[X^T y = X^T X w\]

\[y = \begin{bmatrix}
y^1 \\
\vdots \\
y^N
\end{bmatrix}, \quad X = \begin{bmatrix}
x^1_0 & x^1_1 \\
\vdots & \vdots \\
x^N_0 & x^N_2
\end{bmatrix}\]

\[w = (X^T X)^{-1} X^T y\]
Code Example
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin = 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x = linspace(-5,5,28)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)

# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin = 2
w = rand(2, 1)  # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x = np.linspace(-5, 5, 20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack([np.ones((len(x), 1)), x.reshape(-1, 1)])

# These are the normal equations in matrix form: w = (X'X)^{-1}X'y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
weight = 0.01
w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)

# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()}
Code Example

```python
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# Generate data on a line perturbed with some noise
noise_margin = 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x = np.linspace(-5,5,20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# Create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack([np.ones([len(x), 1]), x.reshape(-1, 1)])

# These are the normal equations in matrix form: w = (X'X)^{-1}X'y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
weight = 0.01
w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)

# Evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# Visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

\[ w = (X^T X)^{-1} X^T y \]
Linear Regression (Line/Plane Fitting)
Linear Regression (Line/Plane Fitting)
LS Solution for Regression

\[ L(w) = \sum_{i=1}^{N} (y^i - w^T x^i)^2 = \sum_{i=1}^{N} (\epsilon^i)^2 \]

\[ L(w) = \begin{bmatrix} \epsilon^1 & \epsilon^2 & \ldots & \epsilon^N \end{bmatrix} \begin{bmatrix} \epsilon^1 \\ \epsilon^2 \\ \vdots \\ \epsilon^N \end{bmatrix} \]

\[ y = Xw + \epsilon \]
LS Solution for Regression

\[ L(w) = \sum_{i=1}^{N} (y^i - w^T x^i)^2 = \sum_{i=1}^{N} (\epsilon^i)^2 \]

\[ L(w) = \begin{bmatrix} \epsilon^1 & \epsilon^2 & \ldots & \epsilon^N \end{bmatrix} \begin{bmatrix} \epsilon^1 \\ \epsilon^2 \\ \vdots \\ \epsilon^N \end{bmatrix} \]

\[ L(w) = \epsilon^T \epsilon \quad y = Xw + \epsilon \]
Generalized Linear Regression

\[ x \rightarrow \phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_M(x) \end{bmatrix} \]
Generalized Linear Regression

\[ x \rightarrow \phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_M(x) \end{bmatrix} \]

known nonlinearity
1D Example: $k$-th Degree Polynomial Fitting

$$
\phi(x) = \begin{bmatrix}
1 \\
x \\
\vdots \\
(x)^K
\end{bmatrix}
$$

$$
\langle w, \phi(x) \rangle = w_0 + w_1 x + \ldots + w_k (x)^K
$$
Generalized Linear Regression

\[ L(w) = \sum_{i=1}^{N} (y^i - w^T \phi(x^i))^T = \sum_{i=1}^{N} (\epsilon^i)^2 \]
Generalized Linear Regression

\[ L(w) = \sum_{i=1}^{N} (y^i - w^T \phi(x^i))^T = \sum_{i=1}^{N} (\epsilon^i)^2 \]

\[
\begin{bmatrix}
  y^1 \\
  y^2 \\
  \vdots \\
  y^N \\
\end{bmatrix}_{N \times 1} =
\begin{bmatrix}
  \phi(x^1)^T \\
  \phi(x^2)^T \\
  \vdots \\
  \phi(x^N)^T \\
\end{bmatrix}_{N \times M} \begin{bmatrix}
  w_1 \\
  w_2 \\
  \vdots \\
  w_M \\
\end{bmatrix}_{M \times 1} +
\begin{bmatrix}
  \epsilon^1 \\
  \epsilon^2 \\
  \vdots \\
  \epsilon^N \\
\end{bmatrix}_{N \times 1}
\]

\[ \phi(x) : \mathbb{R}^D \rightarrow \mathbb{R}^M \]
LS Solution for Linear Regression

\[ y = Xw + \epsilon \]

\[ L(w) = \epsilon^T \epsilon \]

\[ X = \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^N)^T \end{bmatrix} \]

\[ w^* = (X^T X)^{-1} X^T y \]
LS Solution for Generalized Linear Regression

\[ y = \Phi w + \epsilon \]

\[ \Phi = \left[ \begin{array}{c} \phi(x^1)^T \\
\phi(x^2)^T \\
\vdots \\
\phi(x^N)^T \end{array} \right] \]
LS Solution for Generalized Linear Regression

\[ y = \Phi w + \epsilon \]

\[ L(w) = \epsilon^T \epsilon \]

\[ \Phi = \begin{bmatrix} 
\phi(x^1)^T \\
\phi(x^2)^T \\
\vdots \\
\phi(x^N)^T 
\end{bmatrix} \]

\[ w^* = (\Phi^T \Phi)^{-1} \Phi y \]
import numpy as np
from numpy import array
from numpy import matmul
from numpy import linalg
import matplotlib
import matplotlib.pyplot
import random
import numpy
import matplotlib
import pyplot

# generate data on a line perturbed with some noise
noise_margin= 3
w = 2*rand(3,1) # w[0] is random constant term (offset from origin), w[1] is random linear term, w[2] is random quadratic term
x = np.linspace(-5,5,20)
y = w[0] + w[1]*x + w[2]*x**2 + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack([np.ones([len(x), 1]), x, x.reshape(-1, 1), (x**2).reshape(-1, 1)])

# These are the normal equations in matrix form: w = (X’ X)^-1 X’ y
w_est = matmul(inv(matmul(X.transpose(), X)),X.transpose()).dot(y)

# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x + w_est[2]*x**2

# visualize the fitted model
pyplot.scatter(x, y)
pyplot.plot(x, y_est, color='red')
pyplot.show()
\[ \mathbf{w}^* = \left( \Phi^T \Phi \right)^{-1} \Phi \mathbf{y} \]
Underfitting vs. Overfitting

Underfitting

classification

regression

Course: “Deep Learning for Graphics”
Underfitting vs. Overfitting

- **Underfitting**
  - Classification
  - Regression

- **Overfitting**
  - Classification
  - Regression

Course: “Deep Learning for Graphics”
Underfitting vs. Overfitting

Underfitting

just right

Overfitting

Classification

Regression

Course: “Deep Learning for Graphics”
Tuning Model’s Complexity
Tuning Model’s Complexity

A **flexible model** approximates the target function well in the training set but can “**overtrain**” and have poor performance on the test set (“variance”).
Tuning Model’s Complexity

A *flexible model* approximates the target function well in the training set but can “overtrain” and have poor performance on the test set (“variance”).

A *rigid model*’s performance is more predictable in the test set but the model may not be good even on the training set (“bias”).
Tuning Model’s Complexity

A flexible model approximates the target function well in the training set but can “overtrain” and have poor performance on the test set (“variance”).

A rigid model’s performance is more predictable in the test set but the model may not be good even on the training set (“bias”).
Regularized Linear Regression

\[ \epsilon = y - \Phi w \quad \text{residual vector} \]

\[ L(w) = \epsilon^T \epsilon \quad \text{linear regression: minimize model error} \]
Regularized Linear Regression

\[ \epsilon = y - \Phi w \quad \text{residual vector} \]

\[ L(w) = \epsilon^T \epsilon \quad \text{linear regression: minimize model error} \]

Complexity term: (regularizer)

\[ R(w) = \|w\|_2^2 = w^T w \]
Regularized Linear Regression

\[ \epsilon = y - \Phi w \]  
residual vector

\[ L(w) = \epsilon^T \epsilon \]  
linear regression: minimize model error

Complexity term:  
(regularizer)  
\[ R(w) = \|w\|^2_2 = w^T w \]

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \]
Regularized Linear Regression

\[ \epsilon = y - \Phi w \quad \text{residual vector} \]

\[ L(w) = \epsilon^T \epsilon \quad \text{linear regression: minimize model error} \]

Complexity term: (regularizer)

\[ R(w) = \| w \|_2^2 = w^T w \]

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \quad \text{“data fidelity”} \]
Regularized Linear Regression

\[ \epsilon = y - \Phi w \quad \text{residual vector} \]

\[ L(w) = \epsilon^T \epsilon \quad \text{linear regression: minimize model error} \]

Complexity term: (regularizer)

\[ R(w) = \|w\|^2_2 = w^T w \]

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \quad \text{“data fidelity” complexity} \]
Regularized Linear Regression

\[ \epsilon = y - \Phi w \]  
residual vector

\[ L(w) = \epsilon^T \epsilon \]  
linear regression: minimize model error

Complexity term: (regularizer)
\[ R(w) = \|w\|_2^2 = w^T w \]

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \]  
“data fidelity” complexity

minimum remains to be determined
Regularized Linear Regression

\[ \epsilon = y - \Phi w \]  
residual vector

\[ L(w) = \epsilon^T \epsilon \]  
linear regression: minimize model error

Complexity term:  
(regularizer)  
\[ R(w) = \|w\|_2^2 = w^T w \]

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \]  
“data fidelity”  complexity

minimum remains to be determined  
scalar, remains to be determined
Least Squares Solution
Least Squares Solution

\[ L(w) = \epsilon^T \epsilon \]
Least Squares Solution

\[ L(w) = \epsilon^T \epsilon \]

\[ = (y - Xw)^T (y - Xw) \]
Least Squares Solution

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T (y - Xw) \]
\[ = y^T y - 2y^T Xw + w^T X^T X w \]
Least Squares Solution

\[ L(w) = \epsilon^T \epsilon \]

\[ = (y - Xw)^T (y - Xw) \]

\[ = y^T y - 2y^T Xw + w^T X^T X w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]
Least Squares Solution

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T(y - Xw) \]
\[ = y^T y - 2y^T Xw + w^T X^T X w \]

Condition for minimum:
\[ \nabla L(w^*) = 0 \]
\[ -2X^T y + 2X^T X w^* = 0 \]
Least Squares Solution

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T (y - Xw) \]
\[ = y^T y - 2y^T Xw + w^T X^T X w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]
\[ -2X^T y + 2X^T X w^* = 0 \]
\[ w^* = (X^T X)^{-1} X^T y \]
Ridge regression: L2-regularized Linear Regression

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \]

\[ = y^T y - 2y^T X w + w^T X^T X w + \lambda w^T I w \]

as before, for linear regression

\[ = y^T y - 2y^T X w + w^T (X^T X + \lambda I) w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]

\[ -2X^T y + 2(X^T X + \lambda I) w^* = 0 \]

\[ w^* = (X^T X + \lambda I)^{-1} X^T y \]
Ridge regression: L2-regularized Linear Regression

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \]

\[ = y^T y - 2y^T Xw + w^T X^T Xw + \lambda w^T I w \]

as before, for linear regression

\[ \text{identity matrix} \]

\[ = y^T y - 2y^T Xw + w^T (X^T X + \lambda I) w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]

\[ -2X^T y + 2(X^T X + \lambda I) w^* = 0 \]

\[ w^* = (X^T X + \lambda I)^{-1} X^T y \]
Ridge regression: L2-regularized Linear Regression

\[ L(w) = \epsilon^T \epsilon + \lambda w^T w \]
\[ = y^T y - 2y^T Xw + w^T X^T Xw + \lambda w^T Iw \]

as before, for linear regression

identity matrix

\[ = y^T y - 2y^T Xw + w^T (X^T X + \lambda I) w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]
\[ -2X^T y + 2(X^T X + \lambda I)w^* = 0 \]
\[ w^* = (X^T X + \lambda I)^{-1} X^T y \]
Bias-Variance Tradeoff (function of $\lambda$)
Bias-Variance Tradeoff (function of $\lambda$)

**Diagram:**
- **Prediction Error**
- **Model Complexity**
- **Training Sample**
- **Test Sample**
- **sweet spot!**

- High Bias, Low Variance
- Low Bias, High Variance

Course: “Deep Learning for Graphics”
Selecting $\lambda$ with Cross-validation
Selecting $\lambda$ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
Selecting $\lambda$ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error

- K-fold cross validation:
  - K splits, average K errors
Selecting $\lambda$ with Cross-validation

• Cross validation technique
  • Exclude part of the training data from parameter estimation
  • Use them only to predict the test error

• K-fold cross validation:
  • K splits, average K errors
Selecting \( \lambda \) with Cross-validation

- **Cross validation technique**
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error

- **K-fold cross validation:**
  - K splits, average K errors

- Use cross-validation for different values of \( \lambda \)
  - pick value that minimizes cross-validation error
Selecting $\lambda$ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error

- K-fold cross validation:
  - K splits, average K errors

- Use cross-validation for different values of $\lambda$
  - pick value that minimizes cross-validation error

Least glorious, most effective of all methods
Form of posterior distribution

Bernoulli-type conditional distribution

\[
\begin{align*}
P(Y = 1 | X = x; w) &= f(x, w) \\
P(Y = 0 | X = x; w) &= 1 - f(x, w)
\end{align*}
\]

Particular choice of form of f:
Form of posterior distribution

Bernoulli-type conditional distribution

\[
\begin{align*}
P(Y = 1 | X = x; w) &= f(x, w) \\
P(Y = 0 | X = x; w) &= 1 - f(x, w)
\end{align*}
\]

Particular choice of form of f:

\[
P(Y = 1 | X = x; w) = g(w^T x)
\]
Form of posterior distribution

Bernoulli-type conditional distribution

\[
P(Y = 1 | X = x; w) = f(x, w) \\
P(Y = 0 | X = x; w) = 1 - f(x, w)
\]

Particular choice of form of \( f \):

\[
P(Y = 1 | X = x; w) = g(w^T x)
\]

Sigmoidal:

\[
g(\alpha) = \frac{1}{1 + \exp(-\alpha)}
\]
Form of posterior distribution

Bernoulli-type conditional distribution

\[ P(Y = 1|X = x; w) = f(x, w) \]
\[ P(Y = 0|X = x; w) = 1 - f(x, w) \]

Particular choice of form of \( f \):

\[ P(Y = 1|X = x; w) = g(w^T x) \]

Sigmoidal: \( g(\alpha) = \frac{1}{1 + \exp(-\alpha)} \)

“squashing function”: \( -\infty \rightarrow 0 \)
\( +\infty \rightarrow 1 \)
Form of posterior distribution

Bernoulli-type conditional distribution

\[
P(Y = 1 | X = x; w) = f(x, w) \\
P(Y = 0 | X = x; w) = 1 - f(x, w) \\
P(Y = y | X = x; w) = f(x, w)^y (1 - f(x, w))^{1-y}
\]

Particular choice of form of \( f \):

\[
P(Y = 1 | X = x; w) = g(w^T x)
\]

Sigmoidal:

\[
g(\alpha) = \frac{1}{1 + \exp(-\alpha)}
\]

“squashing function”:

\[
-\infty \rightarrow 0 \\
+\infty \rightarrow 1
\]
Logistic vs Linear Regression
Logistic vs Linear Regression

Logistic Regression
Linear Regression
From Two to Many

• How about multi-class classification?
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:
\[ y^i = (0, 0, 1, 0) \]
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)  
4 classes, i-th sample is in 3\textsuperscript{rd} class: 
\[ y^i = (0, 0, 1, 0) \]

Matrix notation:
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)  

4 classes, i-th sample is in 3\textsuperscript{rd} class: 
\[ y^i = (0, 0, 1, 0) \]

Matrix notation:
\[ Y = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} \]
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)  
4 classes, i-th sample is in 3\textsuperscript{rd} class:  
\[ y^i = (0, 0, 1, 0) \]

Matrix notation:
\[
Y = \begin{bmatrix}
  y^1 \\
  \vdots \\
  y^N
\end{bmatrix} = \begin{bmatrix}
  y_1 & \cdots & y_C
\end{bmatrix}
\]

where  
\[
y_c = \begin{bmatrix}
  y^1_c \\
  \vdots \\
  y_N^c
\end{bmatrix}
\]
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)  

4 classes, i-th sample is in 3\textsuperscript{rd} class:  
\( y^i = (0, 0, 1, 0) \)

Matrix notation:

\[
Y = \begin{bmatrix}
  y^1  \\
  \vdots  \\
  y^N
\end{bmatrix} = \begin{bmatrix}
  y_1 & \cdots & y_C
\end{bmatrix} \\
 \text{where } y_c = \begin{bmatrix}
  y^1_c  \\
  \vdots  \\
  y^N_c
\end{bmatrix} \\
W = \begin{bmatrix}
  w_1 & \cdots & w_C
\end{bmatrix}
\]
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3\textsuperscript{rd} class:
\[ y^i = (0, 0, 1, 0) \]

Matrix notation:
\[
Y = \begin{bmatrix}
y^1 \\
\vdots \\
y^N \\
\end{bmatrix} = \begin{bmatrix}
y_1 & \cdots & y_C \\
\end{bmatrix}
\]
\[
\text{where } y_c = \begin{bmatrix}
y^1_c \\
\vdots \\
y^N_c \\
\end{bmatrix}
\]

\[
W = \begin{bmatrix}
w_1 & \cdots & w_C \\
\end{bmatrix}
\]

Loss function:
\[
L(W) = \sum_{c=1}^{C} (y_c - Xw_c)^T (y_c - Xw_c)
\]
Multiple Classes & Linear Regression

C classes: one-of-c coding (or one-hot encoding)

4 classes, i-th sample is in 3rd class:
\[ y^i = (0, 0, 1, 0) \]

Matrix notation:
\[
Y = \begin{bmatrix} y^1 \\ \vdots \\ y^N \end{bmatrix} = \begin{bmatrix} y_1 & \cdots & y_C \end{bmatrix}
\]

where
\[
y_c = \begin{bmatrix} y^1_c \\ \vdots \\ y^N_c \end{bmatrix}
\]

\[
W = \begin{bmatrix} w_1 & \cdots & w_C \end{bmatrix}
\]

Loss function:
\[
L(W) = \sum_{c=1}^{C} (y_c - Xw_c)^T (y_c - Xw_c)
\]

Least squares fit (decouples per class):
\[
w_c^* = (X^T X)^{-1} X^T y_c
\]
Linear Regression Masking Problem

One linear discriminant per class:

\[ S_C(x) = W_C^T x \]
Linear Regression Masking Problem

Class 1

$y^1 = xw^1$

Class 2

Class 3

One linear discriminant per class:

$S_C(x) = w_C^T x$

$x$
Linear Regression Masking Problem

Class 1

\[ y^1 = xw^1 \]

Class 2

\[ y^2 = xw^2 \]

Class 3

One linear discriminant per class:

\[ S_C(x) = W^T_C x \]
Linear Regression Masking Problem

Class 1
$y^1 = xw^1$

Class 2
$y^2 = xw^2$

Class 3
$y^3 = xw^3$

One linear discriminant per class:

$S_C(x) = w_C^T x$
Linear Regression Masking Problem

Class 1: $y^1 = xw^1$

Class 2: $y^2 = xw^2$

Class 3: $y^3 = xw^3$

One linear discriminant per class:

$$S_C(x) = w_T^C x$$

Nothing ever gets assigned to class 2!
Linear Regression Masking Problem

Class 1

\[ y^1 = x w^1 \]

Class 2

\[ y^2 = x w^2 \]

Class 3

\[ y^3 = x w^3 \]

One linear discriminant per class:

\[ S_C(x) = w^T_C x \]

Nothing ever gets assigned to class 2!

2D version:
Soft maximum (softmax) of competing classes:

\[
P(y = c | x; W) = \frac{\exp(w_c^T x)}{\sum_{c'=1}^{C} \exp(w_{c'}^T x)} \equiv g_c(x, W)
\]
Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

\[ P(y = c | x; W) = \frac{\exp(w_c^T x)}{\sum_{c'}^{C} \exp(w_{c'}^T x)} = g_c(x, W) \]
Multiple classes & Logistic regression

Soft maximum (softmax) of competing classes:

\[
P(y = c | x; W) = \frac{\exp(w_c^T x)}{\sum_{c'=1}^{C} \exp(w_{c'}^T x)} = g_c(x, W)
\]
Logistic vs Linear Regression, n>2 classes
Logistic vs Linear Regression, n>2 classes
Logistic vs Linear Regression, n>2 classes

Linear regression

Logistic regression
Logistic vs Linear Regression, n>2 classes

Logistic regression does not exhibit the masking problem
LS Solution (in vector form)
LS Solution (in vector form)

\[ L(\mathbf{w}) = \epsilon^T \epsilon \]
LS Solution (in vector form)

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T (y - Xw) \]
LS Solution (in vector form)

\[ L(w) = \epsilon^T \epsilon \]

\[ = (y - Xw)^T (y - Xw) \]

\[ = y^T y - 2y^T Xw + w^T X^T Xw \]
**LS Solution (in vector form)**

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T (y - Xw) \]
\[ = y^T y - 2y^T Xw + w^T X^T X w \]

**Condition for minimum:**

\[ \nabla L(w^*) = 0 \]
LS Solution (in vector form)

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T (y - Xw) \]
\[ = y^T y - 2y^T Xw + w^T X^T X w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]
\[ -2X^T y + 2X^T X w^* = 0 \]
LS Solution (in vector form)

\[ L(w) = \epsilon^T \epsilon \]
\[ = (y - Xw)^T (y - Xw) \]
\[ = y^T y - 2y^T Xw + w^T X^T X w \]

Condition for minimum:

\[ \nabla L(w^*) = 0 \]
\[ -2X^T y + 2X^T X w^* = 0 \]
\[ w^* = (X^T X)^{-1} X^T y \]
Gradient of Cross-entropy Loss

\[ L(w) = - \sum_{i=1}^{N} y^i \log g(w^T x^i) + (1 - y^i) \log(1 - g(w^T x^i)) \]
Gradient of Cross-entropy Loss

\[ L(w) = -\sum_{i=1}^{N} y^i \log g(w^T x^i) + (1 - y^i) \log(1 - g(w^T x^i)) \]

\[ \frac{\partial L(w)}{\partial w_k} = -\sum_{i=1}^{N} \left[ y^i \frac{1}{g(w^T x^i)} \frac{\partial g(w^T x^i)}{\partial w_k} + (1 - y^i) \frac{1}{1 - g(w^T x^i)} \left( -\frac{\partial g(w^T x^i)}{\partial w_k} \right) \right] \]

using \[ g(x) = \frac{1}{1 + \exp(-x)} \rightarrow \frac{dg}{dx} = g(x)(1-g(x)) \]

\[ = -\sum_{i=1}^{N} \left[ y^i \frac{1}{g(w^T x^i)} - (1 - y^i) \frac{1}{1 - g(w^T x^i)} \right] g(w^T x^i)(1 - g(w^T x^i)) \frac{\partial w^T x^i}{\partial w_k} \]

\[ = -\sum_{i \neq 1}^{N} \left[ y^i (1 - g(w^T x^i)) - (1 - y^i) g(w^T x^i) \right] x^i_k \]

\[ = -\sum_{i=1}^{N} \left[ y^i - g(w^T x^i) \right] x^i_k \]
Gradient of Cross-entropy Loss

\[
L(\mathbf{w}) = - \sum_{i=1}^{N} y^i \log g(\mathbf{w}^T \mathbf{x}^i) + (1 - y^i) \log(1 - g(\mathbf{w}^T \mathbf{x}^i))
\]

\[
\frac{\partial L(\mathbf{w})}{\partial w_k} = - \sum_{i=1}^{N} \left[ y^i \frac{1}{g(\mathbf{w}^T \mathbf{x}^i)} \frac{\partial g(\mathbf{w}^T \mathbf{x}^i)}{\partial w_k} + (1 - y^i) \frac{1}{1 - g(\mathbf{w}^T \mathbf{x}^i)} \left(- \frac{\partial g(\mathbf{w}^T \mathbf{x}^i)}{\partial w_k}\right) \right]
\]

Using
\[
g(x) = \frac{1}{1 + \exp(-x)} \quad \Rightarrow \quad \frac{dg}{dx} = g(x)(1 - g(x))
\]

\[
= - \sum_{i=1}^{N} \left[ y^i \frac{1}{g(\mathbf{w}^T \mathbf{x}^i)} - (1 - y^i) \frac{1}{1 - g(\mathbf{w}^T \mathbf{x}^i)} \right] g(\mathbf{w}^T \mathbf{x}^i)(1 - g(\mathbf{w}^T \mathbf{x}^i)) \frac{\partial \mathbf{w}^T \mathbf{x}^i}{\partial w_k}
\]

\[
= - \sum_{i \neq 1} \left[ y^i (1 - g(\mathbf{w}^T \mathbf{x}^i)) - (1 - y^i)g(\mathbf{w}^T \mathbf{x}^i) \right] x_k^i
\]

\[
= - \sum_{i=1} y^i - g(\mathbf{w}^T \mathbf{x}^i) \quad x_k^i
\]

\[
\nabla L(\mathbf{w}^*) = 0
\]

nonlinear system of equations!!
Gradient Descent Minimization

\[ \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]

Fact: gradient at any point gives direction of fastest increase
Gradient Descent Minimization

\[ \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]

Fact: gradient at any point gives direction of fastest increase
Idea: start at a point and move in the direction opposite to the gradient
Gradient Descent Minimization

Fact: gradient at any point gives direction of fastest increase
Idea: start at a point and move in the direction opposite to the gradient

\[ \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]
Gradient Descent Minimization

Fact: gradient at any point gives direction of fastest increase
Idea: start at a point and move in the direction opposite to the gradient
Gradient Descent Minimization

Fact: gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ for $i=0$
Gradient Descent Minimization

Update:

\[ x_{i+1} = x_i - \alpha \nabla f(x_i) \]

\( i = 1 \)
Gradient Descent Minimization

\[ \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]

Update:

\[ x_{i+1} = x_i - \alpha \nabla f(x_i) \quad i=1 \]
Gradient Descent Minimization

\[ \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]

Update:

\[ x_{i+1} = x_i - \alpha \nabla f(x_i) \]

\( i = 2 \)
Gradient Descent Minimization

\[ \nabla f(x) = \left[ \begin{array}{c} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{array} \right] \]

Update:

\[ x_{i+1} = x_i - \alpha \nabla f(x_i) \quad i=2 \]
Gradient Descent Minimization

\[ \nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \]

Update:

\[ x_{i+1} = x_i - \alpha \nabla f(x_i) \]  

\[ i=3 \]
Gradient Descent Minimization

Initialize: \( x_0 \)

Update: \( x_{i+1} = x_i - \alpha \nabla f(x_i) \)
Gradient Descent Minimization

Initialize: \( x_0 \)

Update: \( x_{i+1} = x_i - \alpha \nabla f(x_i) \)

We can always make it converge for a convex function.
Gradient Descent Minimization

Initialize: \( x_0 \)

Update: \( x_{i+1} = x_i - \alpha \nabla f(x_i) \)

We can always make it converge for a convex function.
Gradient Descent Minimization

Initialize: \[ x_0 \]

Update: \[ x_{i+1} = x_i - \alpha \nabla f(x_i) \]

We can always make it converge for a convex function.
XOR Problem
XOR Problem

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
XOR Problem

\[ y = f(x_1, x_2) \]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**XOR Problem**

\[ y = f(x_1, x_2) \]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**XOR Problem**

\[ y = f(x_1, x_2) \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2) \]
XOR Problem

\[ y = f(x_1, x_2) \]

\[ y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2) \]
XOR Problem

\[ y = f(x_1, x_2) \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2) \]
XOR Problem
XOR Problem

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$z_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
XOR Problem

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$z_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Course: “Deep Learning for Graphics”
## XOR Problem

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$z_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$z_1$</th>
<th>$z_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Course: “Deep Learning for Graphics”
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( z_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( z_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( z_1 )</th>
<th>( z_2 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) = f(\mathcal{H}(w^1, x_1, x_2), \mathcal{H}(w^2, x_1, x_2)) \]
XOR Problem

\[
y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
= f(\mathcal{H}(w^1, x_1, x_2), \mathcal{H}(w^2, x_1, x_2)) \\
= \mathcal{H}(w^3, \mathcal{H}(g_1(w^1, x_1, x_2)), \mathcal{H}(g_2(w^2, x_1, x_2)))
\]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) = f(H(w^1, x_1, x_2), H(w^2, x_1, x_2)) = H(w^3, H(g_1(w^1, x_1, x_2)), H(g_2(w^2, x_1, x_2))) \]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) = f(H(w^1, x_1, x_2), H(w^2, x_1, x_2)) = H(w^3, H(g_1(w^1, x_1, x_2)), H(g_2(w^2, x_1, x_2))) \]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \\
= f(\mathcal{H}(w^1, x_1, x_2), \mathcal{H}(w^2, x_1, x_2)) \\
= \mathcal{H}(w^3, \mathcal{H}(g_1(w^1, x_1, x_2)), \mathcal{H}(g_2(w^2, x_1, x_2))) \]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \]

\[ = f(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)) \]

\[ = \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2))) \]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \]
\[ = f(H(w^1, x_1, x_2), H(w^2, x_1, x_2)) \]
\[ = H(w^3, H(g_1(w^1, x_1, x_2)), H(g_2(w^2, x_1, x_2))) \]
**XOR Problem**

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) \]
\[ = f(\mathcal{H}(\mathbf{w}^1, x_1, x_2), \mathcal{H}(\mathbf{w}^2, x_1, x_2)) \]
\[ = \mathcal{H}(\mathbf{w}^3, \mathcal{H}(g_1(\mathbf{w}^1, x_1, x_2)), \mathcal{H}(g_2(\mathbf{w}^2, x_1, x_2))) \]
XOR Problem

\[ y = f(z_1, z_2) = f(g_1(x_1, x_2), g_2(x_1, x_2)) = f(H(w^1, x_1, x_2), H(w^2, x_1, x_2)) = H(w^3, H(g_1(w^1, x_1, x_2)), H(g_2(w^2, x_1, x_2))) \]
Course Information (slides/code/comments)

http://geometry.cs.ucl.ac.uk/dl4g/