Day 6: Neural Networks

Summer STEM: Machine Learning

Nikola Janjušević, Akshaj Kumar Veldanda, Jacky Yuan, Tejaishwarya Gagadam

Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, New York

June 24, 2019

Learning Objectives

- What are the advantages of Neural Networks?
- What is the mathematical model for a Neural Network?
- What are the hyper-parameters associated with a Neural Network?
- Why are batch-size and learning-rate important? How are they related?
- How do we implement Neural Networks with Keras?

Outline

1. Review of Week 1
2. Neural Network Model
3. Training with Neural Networks
4. Introduction to Keras
5. Lab: Music Classification
6. (Optional) Lab: Cat vs. Non-Cat

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit
- **Problem 2:** Determining the sentiment of customer reviews for a product on Amazon.

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit
- **Problem 2:** Determining the sentiment of customer reviews for a product on Amazon.
  - Classification Problem

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit
- **Problem 2:** Determining the sentiment of customer reviews for a product on Amazon.
  - Classification Problem
  - Class Labels: Positive, Negative, Neutral

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit
- **Problem 2:** Determining the sentiment of customer reviews for a product on Amazon.
  - Classification Problem
  - Class Labels: Positive, Negative, Neutral
- **Problem 3:** Predict whether an employee's income is over 100k a year or not.

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit
- **Problem 2:** Determining the sentiment of customer reviews for a product on Amazon.
  - Classification Problem
  - Class Labels: Positive, Negative, Neutral
- **Problem 3:** Predict whether an employee's income is over 100k a year or not.
  - Classification Problem

Regression or Classification?

- **Problem 1:** Categorizing credit card applications into those who have good credit, bad credit and those who fall in the gray area.
  - Classification Problem
  - Class Labels: Good credit, Bad credit, Average credit
- **Problem 2:** Determining the sentiment of customer reviews for a product on Amazon.
  - Classification Problem
  - Class Labels: Positive, Negative, Neutral
- **Problem 3:** Predict whether an employee's income is over 100k a year or not.
  - Classification Problem
  - Class Labels: Over 100k, Under 100k

Regression or Classification?

- **Problem 4:** Estimating change in climate.

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem
  - Target Variable: Predicting future temperatures

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem
  - Target Variable: Predicting future temperatures
- **Problem 5:** Identifying hate speech in social media.

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem
  - Target Variable: Predicting future temperatures
- **Problem 5:** Identifying hate speech in social media.
  - Classification Problem

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem
  - Target Variable: Predicting future temperatures
- **Problem 5:** Identifying hate speech in social media.
  - Classification Problem
  - Class labels: Normal Speech, Hate Speech

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem
  - Target Variable: Predicting future temperatures
- **Problem 5:** Identifying hate speech in social media.
  - Classification Problem
  - Class labels: Normal Speech, Hate Speech
- **Problem 6:** Forecasting the energy demand in a region.

Regression or Classification?

- **Problem 4:** Estimating change in climate.
    - Regression Problem
    - Target Variable: Predicting future temperatures
- **Problem 5:** Identifying hate speech in social media.
    - Classification Problem
    - Class labels: Normal Speech, Hate Speech
- **Problem 6:** Forecasting the energy demand in a region.
    - Regression Problem

Regression or Classification?

- **Problem 4:** Estimating change in climate.
  - Regression Problem
  - Target Variable: Predicting future temperatures
- **Problem 5:** Identifying hate speech in social media.
  - Classification Problem
  - Class labels: Normal Speech, Hate Speech
- **Problem 6:** Forecasting the energy demand in a region.
  - Regression Problem
  - Target Variable: Predicting the amount of energy needed in the future

Machine Learning Problem Pipeline

1. Gather data
2. Visualize the data
3. Formulate ML problem
   - Regression vs Classification
   - Choose an appropriate cost function
4. Design the model and train to find the optimal parameters of the model
   - Prepare a design matrix
   - Perform feature engineering
   - Validate your choice of hyper-parameters using a cross-validation set
5. Evaluate the model on a test set
   - If the performance is not satisfactory, go back to step 4

Data

- Always save your data file as an .csv file
  - It is easy to edit in both excel and text file
  - Easy to load the data using Pandas
- Visualize the data
  - To get an rough estimate of how your machine learning model should be
  - Do you have sufficient training and testing data
- Always plot the data before pre-processing

Notation

- Numbers:

Notation

- Numbers:
  - N: total number of samples

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
    - Also used: β_0 for $K = 1$

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
    - Also used: β_0 for $K = 1$
- Matrices:

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
    - Also used: β_0 for $K = 1$
- Matrices:
  - A: (N,M) design matrix

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
    - Also used: β_0 for $K = 1$
- Matrices:
  - A: (N,M) design matrix
    - Also used: X

Notation

- Numbers:
  - N: total number of samples
  - M: model order, number of features (engineered or not)
  - K: number of outputs or classes
- Vectors:
  - $\mathbf{x}$: feature vector, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$
  - $\mathbf{y}$: target vector, $\mathbf{y} = [y_1, y_2, ..., y_K]^T$
  - $\hat{\mathbf{y}}$: predicted target vector, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, ...\hat{y}_K]^T$
  - $\mathbf{w}$: weight vector for $K = 1$ targets, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
    - Also used: $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
    - Also used: β_0 for $K = 1$
- Matrices:
  - A: (N,M) design matrix
    - Also used: X
  - W: (K,M) weight matrix

Supervised Learning

Type	Linear Regression	Logistic Regression
Use	Modeling Continuous Data	Classification
Features	Any Numerical Data, $\mathbf{x} = [x_1, x_2, ..., x_M]^T$	
Targets	Any Numerical Data, $\mathbf{y}$	Class Labels, $\mathbf{y}$
Model	$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$	$\hat{\mathbf{y}} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$
Loss Function	Error between $\mathbf{y}$ and $\hat{\mathbf{y}}$	Cross-Entropy

Optimization

- Use loss/error/cost function to find best model-parameters

Problem	Loss Function	Formula
Regression	Squared/L2 Loss	$\sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
Binary Classification	Binary Cross-Entropy	$-\sum_i (y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i))$
Multi-Class Classification	Cross-Entropy	$-\sum_i \sum_k (y_{ik} \ln(\hat{y}_{ik}))$

Optimization

- Use loss/error/cost function to find best model-parameters
- Non-linear opt. can use arbitrary Loss function

Problem	Loss Function	Formula
Regression	Squared/L2 Loss	$\sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
Binary Classification	Binary Cross-Entropy	$-\sum_i (y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i))$
Multi-Class Classification	Cross-Entropy	$-\sum_i \sum_k (y_{ik} \ln(\hat{y}_{ik}))$

Goodness of Fit

- Evaluate the accuracy of the model

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization
  - Mean Absolute Error: $\frac{1}{N}\sum_i|\mathbf{y}_i - \hat{\mathbf{y}}_i|$

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization
  - Mean Absolute Error: $\frac{1}{N}\sum_i|\mathbf{y}_i - \hat{\mathbf{y}}_i|$
    - Easily interpretable units

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization
  - Mean Absolute Error: $\frac{1}{N}\sum_i|\mathbf{y}_i - \hat{\mathbf{y}}_i|$
    - Easily interpretable units
  - Root Mean Squared Error: $\sqrt{\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}$

NYU TANDON SCHOOL OF ENGINEERING

Goodness of Fit

- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization
  - Mean Absolute Error: $\frac{1}{N}\sum_i|\mathbf{y}_i - \hat{\mathbf{y}}_i|$
    - Easily interpretable units
  - Root Mean Squared Error: $\sqrt{\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}$
    - May represent opt. & easily interpretable units

Goodness of Fit

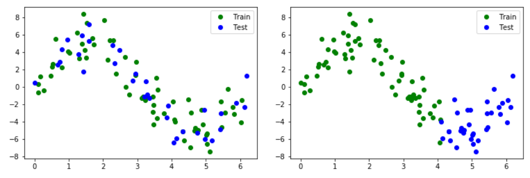
- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization
  - Mean Absolute Error: $\frac{1}{N}\sum_i|\mathbf{y}_i - \hat{\mathbf{y}}_i|$
    - Easily interpretable units
  - Root Mean Squared Error: $\sqrt{\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}$
    - May represent opt. & easily interpretable units
  - Classification Accuracy: $\frac{1}{N}\sum_i(\mathbf{y}_i == \hat{\mathbf{y}}_i)$

Train, Validation, and Test Sets

- Always split your data into train and test sets to see how well it does against new data
- Train set: set of data to be used for training
  - e.g. model.fit(x_train,y_train)
- Test set: After training is done, evaluate how well it does against unseen data using test set
- Validation set: If tuning hyper-paramters, perform one more split to get a validation set. Use validation set to tune parameters

Train and Test Sets (Dealing with Time Series)

- Train and test split is usually done by taking samples at random from the entire data set
- But when using time series to predict future, it is better to select test set to be a continuous chunk at the end of the time series
- Because we want to see how well the model does in predicting the future



Regularization

- Prevent over-fitting by adding a term to loss function
- $\textit{Loss Function} = \textit{Target loss function} + \lambda\,\textit{Regularization}$
- λ hyper-parameter determine how much to emphasize on regularizing
- Large weights usually lead to over-fitting
- Weight-based regularization is most commonly used
  - L2 (Ridge) Regularization: $\sum_{j=1}^{D} |w_j|^2$
  - L1 (Lasso) Regularization: $\sum_{j=1}^{D} |w_j|$
- First over-estimate the model order you need, then use regularization to prevent over-fitting

Outline

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...
- What if $f(\mathbf{x})$ is a linear transformation? ie. $f(\mathbf{x}) = W'\mathbf{x}$

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...
- What if $f(\mathbf{x})$ is a linear transformation? ie. $f(\mathbf{x}) = W'\mathbf{x}$
  - All linear transforms can be represented as matrices

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...
- What if $f(\mathbf{x})$ is a linear transformation? ie. $f(\mathbf{x}) = W'\mathbf{x}$
  - All linear transforms can be represented as matrices
  - So, $\hat{y} = \sigma(W''\mathbf{x} + b)$, where $W'' = WW'$

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...
- What if $f(\mathbf{x})$ is a linear transformation? ie. $f(\mathbf{x}) = W'\mathbf{x}$
  - All linear transforms can be represented as matrices
  - So, $\hat{y} = \sigma(W''\mathbf{x} + b)$, where $W'' = WW'$
- Non-linear transform gives something different...

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...
- What if $f(\mathbf{x})$ is a linear transformation? ie. $f(\mathbf{x}) = W'\mathbf{x}$
  - All linear transforms can be represented as matrices
  - So, $\hat{y} = \sigma(W''\mathbf{x} + b)$, where $W'' = WW'$
- Non-linear transform gives something different...
  - Recall polynomial transformations and exponential transformations of the data

Extension from Logistic Regression

- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(\mathbf{x})$: $\hat{y} = \sigma(Wz + b)$
  - Think of feature engineering...
- What if $f(\mathbf{x})$ is a linear transformation? ie. $f(\mathbf{x}) = W'\mathbf{x}$
  - All linear transforms can be represented as matrices
  - So, $\hat{y} = \sigma(W''\mathbf{x} + b)$, where $W'' = WW'$
- Non-linear transform gives something different...
  - Recall polynomial transformations and exponential transformations of the data
  - These cannot be expressed as matrix multiplication

Extension to Neural Network

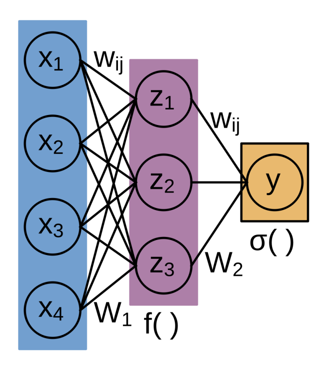
- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values

Extension to Neural Network

- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values
  - Simplest example of a **Neural Network**

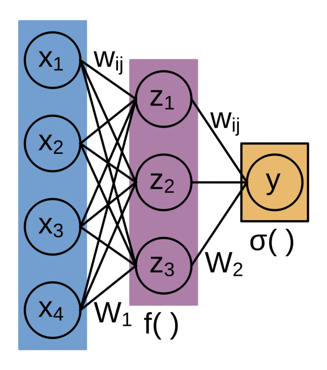
Extension to Neural Network

- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values
  - Simplest example of a **Neural Network**
- $\hat{y} = \sigma(W_2 f_1(W_1 \mathbf{x} + \mathbf{b}_1) + b_2)$



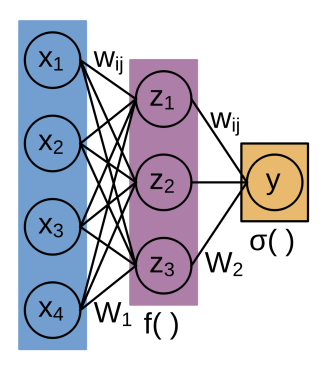
Extension to Neural Network

- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values
  - Simplest example of a **Neural Network**
- $\hat{y} = \sigma(W_2 f_1(W_1\mathbf{x} + \mathbf{b}_1) + b_2)$
- We can optimize for both $W_1, \mathbf{b}_1$ and $W_2, b_2 2$ model-parameters



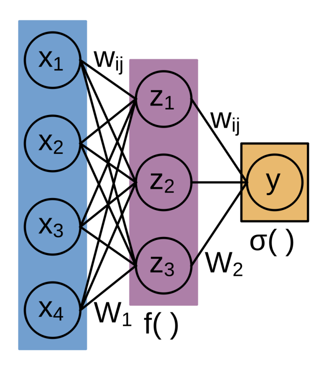
Extension to Neural Network

- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values
  - Simplest example of a **Neural Network**
- $\hat{y} = \sigma(W_2 f_1(W_1\mathbf{x} + \mathbf{b}_1) + b_2)$
- We can optimize for both $W_1, \mathbf{b}_1$ and $W_2, b_2 2$ model-parameters
  - $\nabla J = [\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_P}]^T$



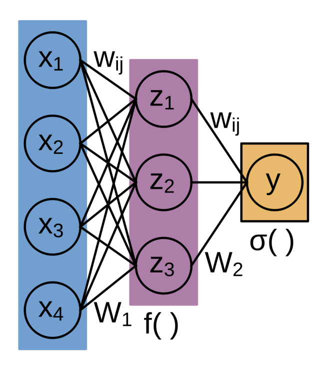
Extension to Neural Network

- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values
  - Simplest example of a **Neural Network**
- $\hat{y} = \sigma(W_2 f_1(W_1\mathbf{x} + \mathbf{b}_1) + b_2)$
- We can optimize for both $W_1, \mathbf{b}_1$ and $W_2, b_2 2$ model-parameters
  - $\nabla J = [\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_P}]^T$
  - Now we're *learning* the feature engineering



Extension to Neural Network

- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values
  - Simplest example of a **Neural Network**
- $\hat{y} = \sigma(W_2 f_1(W_1\mathbf{x} + \mathbf{b}_1) + b_2)$
- We can optimize for both $W_1, \mathbf{b}_1$ and $W_2, b_2 2$ model-parameters
  - $\nabla J = [\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_P}]^T$
  - Now we're *learning* the feature engineering
- But why stop here?...



Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L \mathbf{z}_L + b_L)) + b_{out})$$

Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L \mathbf{z}_L + b_L)) + b_{out})$$

  - Where, $z_l = f_{l-1}(W_{l-1}\mathbf{z}_{l-1} + b_{l-1})$ for $1 \leq l \leq L$, $z_0 := \mathbf{x}$, and L is the number of hidden layers

Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L \mathbf{z}_L + b_L)) + b_{out})$$

  - Where, $z_l = f_{l-1}(W_{l-1}\mathbf{z}_{l-1} + b_{l-1})$ for $1 \leq l \leq L$, $z_0 := \mathbf{x}$, and L is the number of hidden layers
  - ie. all hidden layers are non-linear activation of linear transform

Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L\mathbf{z}_L + b_L)) + b_{out})$$

  - Where, $z_l = f_{l-1}(W_{l-1}\mathbf{z}_{l-1} + b_{l-1})$ for $1 \leq l \leq L$, $z_0 := \mathbf{x}$, and L is the number of hidden layers
  - ie. all hidden layers are non-linear activation of linear transform
  - f_{out} depends on type of ML problem (regression: linear, classification: sigmoid/soft-max)

Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L \mathbf{z}_L + b_L)) + b_{out})$$

  - Where, $z_l = f_{l-1}(W_{l-1}\mathbf{z}_{l-1} + b_{l-1})$ for $1 \leq l \leq L$, $z_0 := \mathbf{x}$, and L is the number of hidden layers
  - ie. all hidden layers are non-linear activation of linear transform
  - f_{out} depends on type of ML problem (regression: linear, classification: sigmoid/soft-max)
- Layers: Input, Hidden, Output

Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L \mathbf{z}_L + b_L)) + b_{out})$$

  - Where, $z_l = f_{l-1}(W_{l-1}\mathbf{z}_{l-1} + b_{l-1})$ for $1 \leq l \leq L$, $z_0 := \mathbf{x}$, and L is the number of hidden layers
  - ie. all hidden layers are non-linear activation of linear transform
  - f_{out} depends on type of ML problem (regression: linear, classification: sigmoid/soft-max)
- Layers: Input, Hidden, Output
  - All layers before output performing feature extraction for final layer linear/logistic regression

Mathematical Model: Multi-Layer Perceptron

- Model:

$$\hat{\mathbf{y}} = f_{out}(f_L(W_L \mathbf{z}_L + b_L)) + b_{out})$$

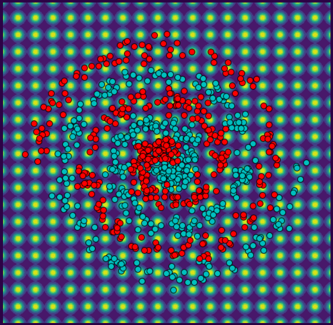
  - Where, $z_l = f_{l-1}(W_{l-1}\mathbf{z}_{l-1} + b_{l-1})$ for $1 \leq l \leq L$, $z_0 := \mathbf{x}$, and L is the number of hidden layers
  - ie. all hidden layers are non-linear activation of linear transform
  - f_{out} depends on type of ML problem (regression: linear, classification: sigmoid/soft-max)
- Layers: Input, Hidden, Output
  - All layers before output performing feature extraction for final layer linear/logistic regression
  - layer size determined by matrix multiply W_l (output dim, input dim)

Mathematical Model: Multi-Layer Perceptron

- Activation Functions: On board

Toy Example: Spiral Classification

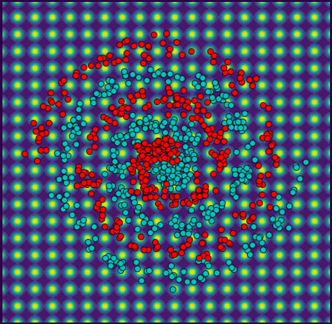
Human Engineered
Feature Transformations:



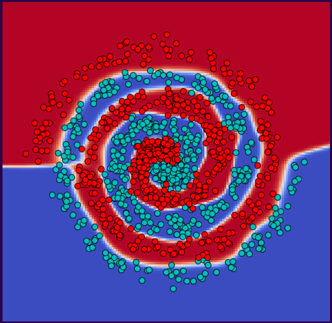
NN Engineered
Feature Transformations:

Toy Example: Spiral Classification

Human Engineered
Feature Transformations:



NN Engineered
Feature Transformations:



Advantages and Disadvantages

Advantages	Disadvantages
- Further removed need for domain knowledge - Infinitely expressive	- Less control over behavior of model - Computationally expensive

NYU TANDON SCHOOL OF ENGINEERING

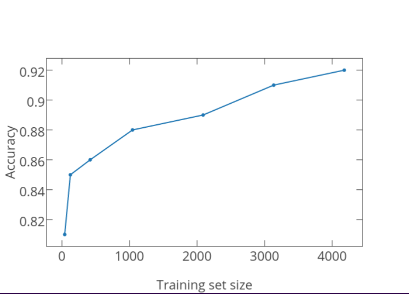
Biological Justification

- Example: Steps for Processing Vision
  1. Eyes gather light
  2. Light intensities converted to shapes
  3. shapes recognized as objects

Outline

Large Scale Machine Learning

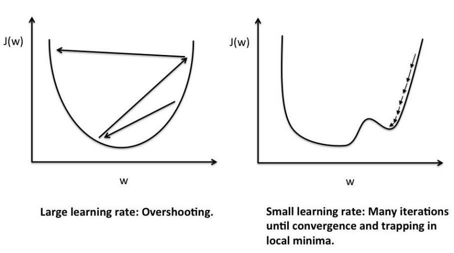
- Learning with large data sets
- Algorithms today perform so much better than five years ago due to shear amount of data availability
- "It's not who has the best algorithm that wins. It's who has the most data"
  - So we want to learn from large data sets

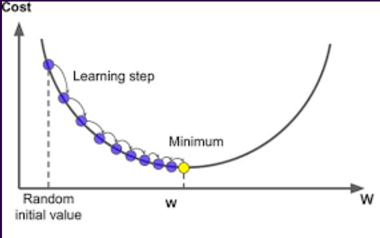


Learning with Large Data Sets

- Challenges:
  - Computationally very expensive to compute gradients
  - And each gradient computation performs only one step of update
- In large scale machine learning, we want to come up with computationally reasonable ways to deal with large data sets
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent

Digression: Revisiting Learning Rate





Correct learning rate

Batch Gradient Descent

- **Batch Gradient Descent** takes all the examples in the training data to compute one step of gradient descent update
- Algorithm: Consider linear regression (N = 100,000,000)
  - $\hat{y} = \sum_{i=0}^{N} w_i x_i$
  - $Cost, J = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2$
  - Gradient Descent Update $w_{new} = w_{old} - \alpha \frac{dJ}{dw}$

Stochastic Gradient Descent

- **SGD** takes only one example in the training example to perform one step of gradient descent
  - The algorithm modifies the parameters a little bit to fit the just first example (x_1, y_1)
  - Then again modify the parameters to fit the second training example (x_2, y_2) and so on...
- Algorithm (Let N be the total number of training examples):
  $Repeat\{$
  for $i = 1, 2...N\{$
  $Cost, J = (y_i - \hat{y}_i)^2$
  Gradient Descent Update $w_{new} = w_{old} - \alpha \frac{dJ}{dw}$
  $\}$
  $\}$

NYU TANDON SCHOOL OF ENGINEERING

Batch Gradient Descent

- **Batch Gradient Descent** uses 'b' training examples to perform one update step
  - 'b' is called batch size
  - Number of iterations = $\frac{N}{b}$
- Algorithm:

$$
\begin{aligned}
&Repeat\{\\
&\qquad j = 0\\
&\qquad \text{for } i \text{ in range(iterations)}\{\\
&\qquad\qquad Cost, J = \frac{1}{b}\sum_{j=1}^{i+b}(y_j - \hat{y}_j)^2\\
&\qquad\qquad \text{Gradient Descent Update } w_{new} = w_{old} - \alpha\frac{dJ}{dw}\\
&\qquad\qquad j = j + b\\
&\qquad \}\\
&\}
\end{aligned}
$$

NYU TANDON SCHOOL OF ENGINEERING

Outline

Outline

Outline

Learning Objectives

- What are the advantages of Neural Networks?
- What is the mathematical model for a Neural Network?
- What are the hyper-parameters associated with a Neural Network?
- Why are batch-size and learning-rate important? How are they related?
- How do we implement Neural Networks with Keras?

Thank You!

- Next Class: Convolutional Neural Networks