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

July 15, 2019

Outline

NYU TANDON SCHOOL OF ENGINEERING

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, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
  - $\mathbf{b}$: 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, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
- 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, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
- Matrices:
  - X: (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, $\mathbf{w} = [w_1, w_2, ..., w_M]^T$
  - $\mathbf{b}$: bias vector, $\mathbf{b} = [b_1, b_2, ..., b_K]$
- Matrices:
  - X: (N,M) design matrix
  - 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	$\hat{\mathbf{y}} = \mathbf{Wx} + \mathbf{b}$	$\hat{\mathbf{y}} = \sigma(\mathbf{Wx} + \mathbf{b})$
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: $\sqrt{\frac{1}{N}\sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}$

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

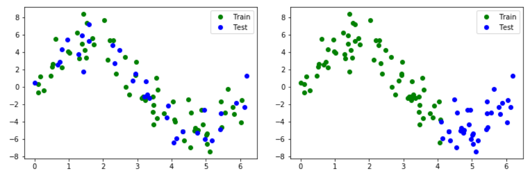
- Evaluate the accuracy of the model
- Can use criteria different than that used for optimization
- Examples:
  - Mean Squared Error: $\frac{1}{N}\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
    - May also represent result of optimization
  - Mean Absolute Error: $\frac{1}{N}\sum_i|\mathbf{y}_i - \hat{\mathbf{y}}_i|$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

Extending Logistic Regression

- Motivation: Feature engineering in the model

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(W\mathbf{x} + b)$: $\hat{y} = \sigma(Wz + b)$

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(W\mathbf{x} + b)$: $\hat{y} = \sigma(Wz + b)$
- So, $\hat{y} = \sigma(W_2 f(W_1\mathbf{x} + b_1) + b_2)$

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(W\mathbf{x} + b)$: $\hat{y} = \sigma(Wz + b)$
- So, $\hat{y} = \sigma(W_2 f(W_1\mathbf{x} + b_1) + b_2)$
- **Fact**: all linear transforms can be represented as matrix multiplication

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(W\mathbf{x} + b)$: $\hat{y} = \sigma(Wz + b)$
- So, $\hat{y} = \sigma(W_2 f(W_1\mathbf{x} + b_1) + b_2)$
- **Fact**: all linear transforms can be represented as matrix multiplication
- We use non-linear function as f to give us a more expressive model

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(W\mathbf{x} + b)$: $\hat{y} = \sigma(Wz + b)$
- So, $\hat{y} = \sigma(W_2 f(W_1\mathbf{x} + b_1) + b_2)$
- **Fact**: all linear transforms can be represented as matrix multiplication
- We use non-linear function as f to give us a more expressive model
  - Recall polynomial transformations and exponential transformations of the data

Extending Logistic Regression

- Motivation: Feature engineering in the model
  - Removes need for domain knowledge
  - Domain knowledge often doesn't exist: ex. object recognition
- Logistic Regression Model: $\hat{y} = \sigma(W\mathbf{x} + b)$
- Replace $\mathbf{x}$ with $\mathbf{z} = f(W\mathbf{x} + b)$: $\hat{y} = \sigma(Wz + b)$
- So, $\hat{y} = \sigma(W_2 f(W_1\mathbf{x} + b_1) + b_2)$
- **Fact**: all linear transforms can be represented as matrix multiplication
- We use non-linear function as f to give us a more expressive model
  - Recall polynomial transformations and exponential transformations of the data
  - These cannot be expressed as matrix multiplication

Extension to Neural Network

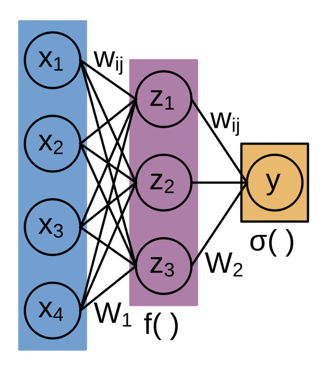
- Restrict $f(\mathbf{x})$ to non-linear function applied to all input values

Extension to Neural Network

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

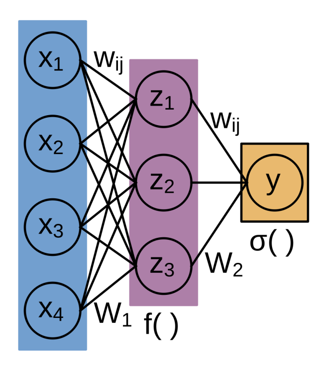
Extension to Neural Network

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



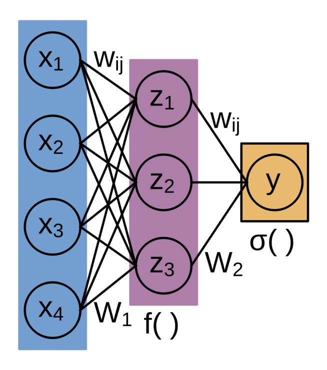
Extension to Neural Network

- 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



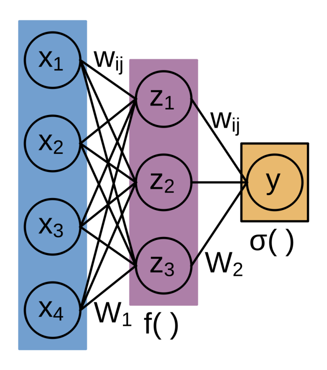
Extension to Neural Network

- 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
  - $\nabla J = [\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_P}]^T$



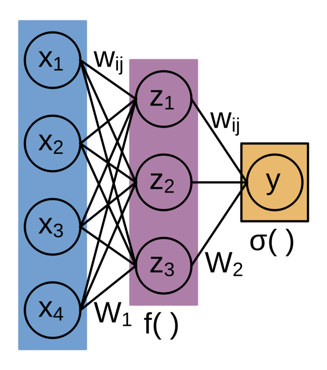
Extension to Neural Network

- 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:

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

Mathematical Model: Multi-Layer Perceptron

- Model:

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

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

Mathematical Model: Multi-Layer Perceptron

- Model:

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

  - Where, $z_l = f_l(W_l\mathbf{z}_{l-1} + b_l)$ 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:

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

  - Where, $z_l = f_l(W_l\mathbf{z}_{l-1} + b_l)$ 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:

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

  - Where, $z_l = f_l(W_l\mathbf{z}_{l-1} + b_l)$ 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)
    - **Regression**: Linear Output

Mathematical Model: Multi-Layer Perceptron

- Model:

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

  - Where, $z_l = f_l(W_l\mathbf{z}_{l-1} + b_l)$ 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)
    - **Regression**: Linear Output
    - **Binary Classification**: Sigmoid Output

Mathematical Model: Multi-Layer Perceptron

- Model:

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

  - Where, $z_l = f_l(W_l \mathbf{z}_{l-1} + b_l)$ 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)
    - **Regression**: Linear Output
    - **Binary Classification**: Sigmoid Output
    - **Multi-Class Classification**: Soft-max Output

Layers

- **Input**: feature vector, $\mathbf{x}$

Layers

- **Input**: feature vector, $\mathbf{x}$
- **Output**: target vector, $\hat{\mathbf{y}}$

Layers

- **Input**: feature vector, $\mathbf{x}$
- **Output**: target vector, $\hat{\mathbf{y}}$
    - linear/logistic regression

Layers

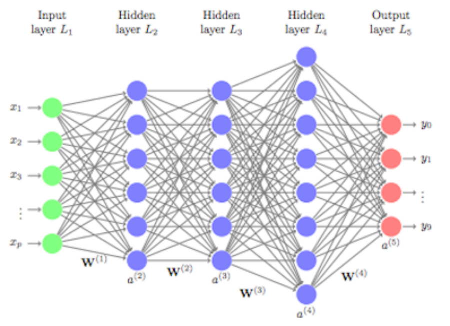
- **Input**: feature vector, $\mathbf{x}$
- **Output**: target vector, $\hat{\mathbf{y}}$
  - linear/logistic regression
- **Hidden**: intermediate vectors, $\mathbf{z}$ or $\mathbf{a}$

Layers

- **Input**: feature vector, $\mathbf{x}$
- **Output**: target vector, $\hat{\mathbf{y}}$
  - linear/logistic regression
- **Hidden**: intermediate vectors, $\mathbf{z}$ or $\mathbf{a}$
  - feature extraction

Layers

- **Input**: feature vector, $\mathbf{x}$
- **Output**: target vector, $\hat{\mathbf{y}}$
  - linear/logistic regression
- **Hidden**: intermediate vectors, $\mathbf{z}$ or $\mathbf{a}$
  - feature extraction



Common Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$

Common Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - $\sigma(z) \in (0, 1)$

Common Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - $\sigma(z) \in (0,1)$
- Tanh (hyperbolic tangent): $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Common Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - $\sigma(z) \in (0,1)$
- Tanh (hyperbolic tangent): $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
  - $tanh(z) \in [-1,1]$

Common Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - $\sigma(z) \in (0, 1)$
- Tanh (hyperbolic tangent): $tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
  - $tanh(z) \in [-1, 1]$
- ReLu (Rectified Linear Unit): $relu(z) = max(0, z)$

Common Activation Functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - $\sigma(z) \in (0, 1)$
- Tanh (hyperbolic tangent): $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
  - $tanh(z) \in [-1, 1]$
- ReLu (Rectified Linear Unit): $relu(z) = max(0, z)$
  - easy to compute, performs well in practice

Guidelines for Designing a NN

- The design space for NN is HUGE

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer
  - bs: batch-size

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer
  - bs: batch-size
  - lr: learning-rate

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer
  - bs: batch-size
  - lr: learning-rate
  - # of epochs

Guidelines for Designing a NN

- **The design space for NN is HUGE**
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer
  - bs: batch-size
  - lr: learning-rate
  - # of epochs
  - λ: weight-regularization constant

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer
  - bs: batch-size
  - lr: learning-rate
  - # of epochs
  - λ: weight-regularization constant
  - J: cost/loss function

Guidelines for Designing a NN

- The design space for NN is HUGE
- Hyper-parameters so far:
  - L: # of layers
  - N_L: # hidden units per layer
  - f: activation function for each layer
  - bs: batch-size
  - lr: learning-rate
  - # of epochs
  - λ: weight-regularization constant
  - J: cost/loss function
- This can be overwhelming...

Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers

Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units $\sim$ 128

Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units ~ 128
  - make sure code is working

Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units ~ 128
  - make sure code is working
  - increase size if val good

Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units ~ 128
  - make sure code is working
  - increase size if val good
  - classification acc $\geq$ guessing

Guidelines for Designing a NN

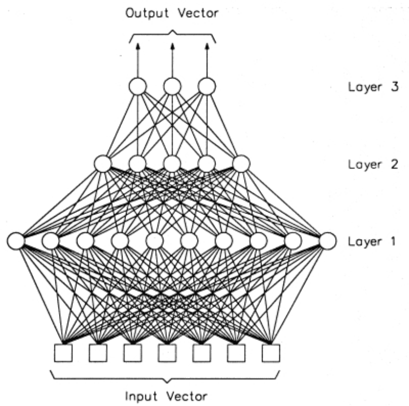
- **Start Small**: 1 or 2 layers
  - # hidden units ~ 128
  - make sure code is working
  - increase size if val good
  - classification acc $\geq$ guessing
- **One activation function**

Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - \# hidden units ~ 128
  - make sure code is working
  - increase size if val good
  - classification acc $\geq$ guessing
- **One activation function**
  - for all hidden layers

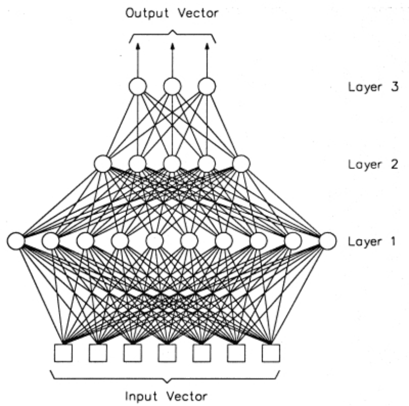
Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units ~ 128
  - make sure code is working
  - increase size if val good
  - classification acc $\geq$ guessing
- **One activation function**
  - for all hidden layers
- **Simple MLP Arch**:



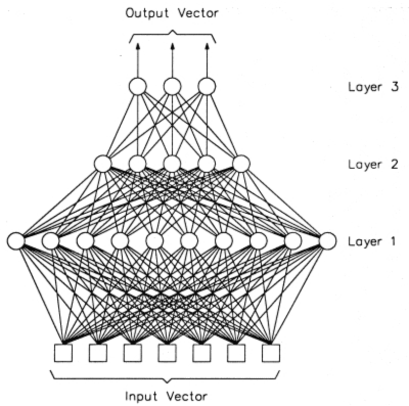
Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units ~ 128
  - make sure code is working
  - increase size if val good
  - classification acc $\geq$ guessing
- **One activation function**
  - for all hidden layers
- **Simple MLP Arch**:
  - Pyramid



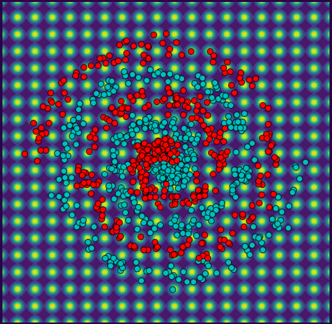
Guidelines for Designing a NN

- **Start Small**: 1 or 2 layers
  - # hidden units $\sim$ 128
  - make sure code is working
  - increase size if val good
  - classification acc $\geq$ guessing
- **One activation function**
  - for all hidden layers
- **Simple MLP Arch**:
  - Pyramid
  - Expand, combine & reduce



Toy Example: Spiral Classification

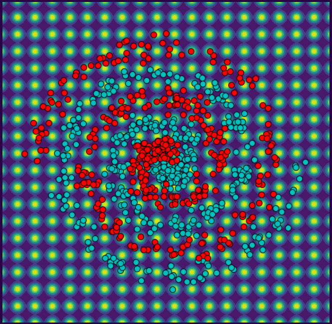
Human Engineered
Feature Transformations:



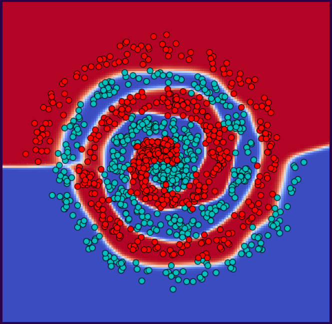
NN Engineered
Feature Transformations:

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 (kind of...)

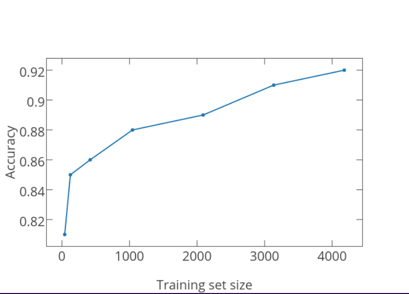
Biological Justification

- Example: Steps for Processing Vision
  1. Eyes gather light
  2. Light intensities converted to shapes
  3. Shapes recognized as objects
  4. Objects associated with ideas
  5. Idea recognized as Akshaj

Outline

Large Scale Machine Learning

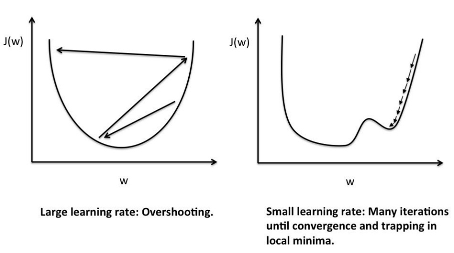
- Learning with large data sets
- Algorithms today perform so much better than five years ago due to shear amount of data availability
- "It's not who has the best algorithm that wins. It's who has the most data"
  - So we want to learn from large data sets

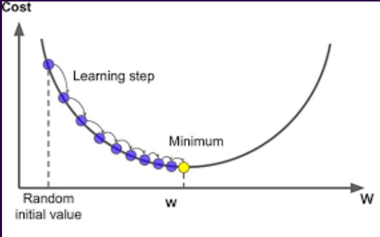


Learning with Large Data Sets

- 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 just the 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 $w_{new} = w_{old} - \alpha \frac{dJ}{dw}$

  }

  }

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:

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

Outline

Outline

Outline

Thank You!

- Next Class: Convolutional Neural Networks