EE482/682: DSP APPLICATIONS
OVERVIEW OF ML AND NEURAL NETWORKS

Géron Chapter 1 + 10
Géron Chapter 1 – Machine Learning (ML) Landscape
  - What is ML
  - Why use ML
  - Types of ML systems
  - Challenges

Géron Chapter 10 – Intro Artificial Neural Networks (ANNs)
  - Biological inspiration
  - Perceptron
  - Multilayer perceptron (MLP)
WHAT IS ML?

- [Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed. – Arthur Samuel, 1959

- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. – Tom Mitchell, 1997

- Machine Learning is the science (and art) of programming computers so they can learn from data
  - Not enough to have lots of data, must be able to use data to solve a task
**WHY USE ML?**

- Traditional approach has complex rules and difficult to maintain
- ML learns from data making code shorter, easier to maintain, and more accurate
- Can inform humans of what was learned for new insight into a problem
- Can automatically be updated to changes
EXAMPLE APPLICATIONS

- Analyzing production line images to classify or detecting tumors in brain scans
  - Chapter 14 – convolutional neural networks (CNNs)
- Visual representation of complex, high-dimensional data
  - Chapter 8 – data visualization and data reduction
- Intelligent bot for a game
  - Chapter 18 – reinforcement learning (RL)
- Forecasting future company revenue
  - Chapter 4, 5, 7, 10, 15, 16 – regression using classical (linear/polynomial regression, SVM, Random Forest or deep learning methods)
- News article classification, flagging offensive comments, long document summarization, chatbot creation
  - Chapter 16 – natural language processing (NLP)
- Making an app react to voice commands
  - Chapter 15/16 – recurrent neural networks (RNNs), CNNs, Transformers
- Detecting credit card fraud, segmenting customers for marketing strategy
  - Chapter 9 – anomaly detection, clustering
- Product recommendations based on past purchases
  - Chapter 10 – ANN
TYPES OF ML SYSTEMS

- Broad categories:
  - Training with human supervision (supervised, unsupervised, semi-supervised, and reinforcement learning)
  - Learning incrementally or on the fly (online vs batch learning)
  - Comparison with known data points or by finding patterns in training data to build predictive models (instance-based vs model-based learning)
- Criteria are not exclusive and can be combined
SUPERVISED LEARNING

- Training with data and labels (desired solutions)
- Typical tasks
  - Classification: determine data class
    - Spam filter: data=emails, label={spam, not-spam}
  - Regression: predict target numeric value
    - Price of car: data=features (mileage, age, brand, etc.) label=price
- Important algorithms
  - k-NN, linear and logistic regression, SVM, decision trees and random forests, neural networks
UNSUPERVISED LEARNING

- Training with unlabeled data (no teacher)
- Important algorithms
  - Clustering – discovering groups
    - K-means, DBSCAN
  - Visualization and dimensionality reduction – reduce feature dimension and maintain structure
    - (Kernel) principle component analysis (PCA), LLE, t-SNE
  - Anomaly/novelty detection – find unusual test data
    - One-class SVM, isolation forest
  - Association rule learning – find relations between features
    - Apriori, Eclat
Training with partially labeled data
- Lots of unlabeled and few labeled instances

Most are combination of unsupervised and supervised algorithms
- Deep belief networks (DBNs) are based on stacked unsupervised restricted Boltzmann machines (RBMs) that are fine-tuned using supervised learning techniques

Example: Google Photos
- Given large personal photo library
- Automatically cluster photos into groups of people
- Supervision when specify name of group
  - Need to merge and split groups to fine-tune
REINFORCEMENT LEARNING

- Agent-based learning paradigm
  - Agent – learning system that can observe environment, select and perform actions, and get rewards
  - Rewards/penalties – “value” associated with actions
  - Policy – strategy, action to choose which is learned to maximize reward over time
- Popular for robotics and game playing
  - E.g. DeepMind’s Q-Learning Atari Breakout or AlphaGo
BATCH LEARNING

- Learning uses all available data
- Offline learning – train then launched into production with no more training
  - Often because of heavy time and resource requirements
- Can update model fairly easily by incorporating new data (say every 24 hours)
- Does not work in many situations
  - Rapidly changing data – need to adapt more quickly
  - Big data and computational restrictions – too costly to train, too large to batch, or not enough resources (mobile phone or Mars rover)
- Incremental training by feeding data instances sequentially
  - Use of mini-batch – small groupings of data
- Well-suited for streaming data or limited computing resources
  - Can react/adapt quickly to changes autonomously
  - Can discard samples after incorporating into model
  - Out-of-core learning for large datasets that do not fit in memory
- Learning-rate – how fast to adapt to changing data
  - High: quickly adapt, but forget old
  - Low: less sensitive to noise/outliers but slower to update (inertia)
- Major challenge: graceful degradation over time
  - How to handle bad data that comes in?
INSTANCE-BASED LEARNING

- System learns examples by-heart, then generalizes to new cases using a similarity measure
  - Simple learning method (e.g. k-NN)
  - Needs to store instances (database)
  - Define meaningful similarity measure
MODEL-BASED LEARNING

- Build model of examples and use model to make predictions
- Need to choose a “model”
  - Tune parameters for good fit
    - Define utility/fitness function for goodness or cost function for badness of fit
- Data
- Linear model
MAIN CHALLENGES OF ML

“Bad Data”
- Insufficient quantity of data – not enough
- Non-representative data – biased data
- Poor quality data – errors, noise, outliers
- Irrelevant features – not measuring the right things

“Bad Algorithms”
- Overfitting – overreliance on limited training data
- Underfitting – not enough model capacity
ML still requires a lot of data to work properly
- 1000s or more (millions for image/speech)

The Unreasonable Effectiveness of Data
- Given enough data, very different ML algorithms (including fairly simple) all perform similarly
- “Reconsider trade-off between spending time and money on algorithm development versus spending it on corpus development”
  - Has led to much of modern ML and computer vision → massive datasets
  - Do we now have enough (too much) data?

**INSUFFICIENT QUANTITY OF TRAINING DATA**
Training data must be representative of test cases to generalize well

- Dashed blue old model using blue dots
- Solid line trained using also red squares
- Poor performance with old model
  - Especially with poor and rich countries

Sampling noise – nonrepresentative sample data due to chance

Sampling bias – training samples have systematic issue in collection which produces non-uniformity (or mismatching of underlying distribution)

- E.g. facial recognition systems performing poorly on darker skin tones
POOR-QUALITY DATA

- Data full of errors, outliers, and noise (e.g., due to poor-quality measurements)
  - Will make it harder to detect underlying patterns and less likely to perform well
- Data scientist spend significant time to cleaning up data
  - Clear outliers – discard or manually fix errors
  - Missing a few features – decide to ignore attribute, instances with “holes”, fill in missing value, or train multiple models (with/without missing features)
IRRELEVANT FEATURES

- Garbage in, garbage out
  - Can only learn if features are relevant, not too much irrelevant info
- Feature engineering – process of determining a good set of features to train on
  - Feature selection – select most useful features among all available/existing features
  - Feature extraction – combining existing features to produce more useful ones
  - Creating new features by gathering new data
- Classical ML uses “hand-crafted” features while deep learning has data-driven features
OVERFITTING THE TRAINING DATA

- Overgeneralizing based on limited data
  - Model is too complex relative to the amount of noisiness in the training data → modeling noise
  - Good performance on training but poor generalization (bad performance on test)

- Options to address problem
  - Simplify model by selecting one with fewer parameters, reducing the number of features, or constraining model
  - Gather more training data
  - Reduce noise in training data (e.g., fix data errors and remove outliers)

High degree polynomial with overfitting
Constraining a model to make it simpler and reduce the risk of overfitting
- E.g. constrain parameters to limit search space

Hyperparameter – parameter of a learning algorithm (not model) to control regularization
- Constant set prior to training
- Not affected by the learning parameter itself

Will have to tune (train) hyperparameters for best performance
UNDERFITTING THE TRAINING DATA

- Occurs when your model is too simple to learn the underlying structure of the data
  - Data is more complex than your selected model
  - Predictions will be poor, even on training data
- Options to address the problem
  - Select a more powerful model, with more parameters
  - Use better features (feature engineering)
  - Reduce the constraints on the model (e.g., reduce regularization hyperparameter)
ML – making machines get better at a task by learning from data rather than explicitly coding rules

ML comes in many flavors: un/supervised, batch/online, instance/model-based

ML steps
- Select modeling approach
- Feed data to learning algorithm
- Tune parameters to fit model to training data

ML systems do not perform well if:
- Training data is too small
- Data is too noisy or polluted with irrelevant features
- Model is too simple or too complex
Most important goal for ML is to generalize well
- Model should behave as expected to new unseen cases
- Evaluate and fine-tune models to be sure it works well

Split training into training and test sets
- Training data – used to train model
- Test data – test model on unseen data and measure the error rate (generalization or out-of-sample error) to estimate how well model performs

Low training and test error is desired
- Low training error but high test error means the model is overfitting
HYPERPARAMETER TUNING AND MODEL SELECTION

- Must select a model with various # parameters (e.g. linear and polynomial) and add regularization to avoid overfitting
- Can use test set for model generalization but not for regularization parameter tuning (test set tuning)

- Use a validation (val or development or dev set) for holdout validation
  - Subset of training data used specifically for model and hyper parameter tuning
  - Train full model on train+val and get generalization error on test set

- Cross-validation (multiple train/val data splits) can be used for better characterization with smaller datasets by averaging performance across splits
  - Val too small $\Rightarrow$ imprecise model evaluations
  - Val too large $\Rightarrow$ not enough training data
DATA MISMATCH

- Data must be representative
  - Don’t want to train on magazine/professional (web) images if the use case are coming from user cell phones
- Makes sure val/test sets match use case
- Train-dev set is split of training data used to determine if model is overfitting or if there is data mismatch
  - Poor val performance → data mismatch
  - Poor train-dev performance → overfit and need to simplify model, add regularization, get more data, or clean data
If you make absolutely no assumptions about the data, then there is no reason to prefer one model over any other – David Wolpert 1996

A priori, there is no model guaranteed to work better

Cannot test all possible models

Must make reasonable assumptions about the data and evaluate only a few reasonable models

Simple tasks – linear models with regularization

Complex tasks – neural networks
OUTLINE

- Géron Chapter 1 – Machine Learning (ML) Landscape
  - What is ML
  - Why use ML
  - Types of ML systems
  - Challenges

- Géron Chapter 10 – Intro Artificial Neural Networks (ANNs)
  - Biological inspiration
  - Perceptron
  - Multilayer perceptron (MLP)
Artificial neural networks (ANNs) first introduced in 1943
Excitement with ANNs waned in the 1960s
1980s had renewed interest but was overtaken in the 1990s with ML techniques such as SVM
Since 2010s major renewed interest
- Huge quantities of data are available to train networks
- Major computing power increases for reduced training times (GPU and cloud)
- Improved training algorithms
- Local optima issue rare
- Lots of funding in ANNs (Artificial Intelligence/Deep Learning)
BIOLOGICAL NEURONS

- Cell mostly found in animal brains
- Produce short electrical impulses (action potentials, APs, or signals) to make synapses release chemical signals (neurotransmitters)
- When a neuron receives enough neurotransmitters it fires its own electrical pulses
- Individual neurons are simple but arranged into vast networks of billions
  - Each neuron connected to thousands of other neurons
  - Neurons seem to be organized in consecutive layers
Artificial neuron proposed by McCulloch and Pitts
- Simple binary inputs and one binary output
- Activates output when certain number of inputs on/active

Even with the simple model, any logical proposition can be computed

Basic building block networks can be combined for more complex logical expressions

Building block networks
- Implement basic logic functions

\[ C = A \]
\[ C = A \land B \]
\[ C = A \lor B \]
\[ C = A \land \neg B \]
THE PERCEPTRON I (TLU)

- Invented by Frank Rosenblatt in 1957
- Inputs/outputs are numbers (instead of binary)
- Based on threshold logic unit (TLU) or linear threshold unit
- Inputs associated with a weight
- TLU computes weighted sum of input
  - \( z = w_1x_1 + w_2x_2 + w_3x_3 \)
- Output after a step (threshold) function
  - Heavyside of sign function
- TLU can be used as a simple linear binary classifier

Output: \( h_w(x) = \text{step}(x^T w) \)

Step function: \( \text{step}(z) \)

Weighted sum: \( z = x^T w \)
THE PERCEPTRON II

- Perceptron is a layer for TLU
  - Fully connected (dense) layer – all inputs connected to all neurons
- Input neuron – pass value through unchanged
- Bias neuron – always outputs 1

Example: Multilabel classifier
- 2 inputs 3 outputs
- Can classify into three binary classes based on two input values
THE PERCEPTRON III

- Output of fully connected layer
  \[ h_{W,b}(X) = \phi(XW + b) \]
  - \( X \) – matrix of input features
  - \( W \) – weight matrix (all weights between input and neurons)
    - One row per input neuron
    - One column per neuron layer
  - \( b \) – bias (weights) vector
  - \( \phi \) – activation function (e.g. step)

- Produces linear (non-complex) decision boundary

- Perceptron training – reinforce connections that reduce prediction error
  \[ w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \]
  - \( w_{i,j} \) - connection weight between ith input and jth output neuron
  - \( x_i \) - ith input value
  - \( \hat{y}_j \) - perceptron output of jth neuron
  - \( y_j \) - target (ground truth) output of jth neuron
  - \( \eta \) – learning rate
MULTILAYER PERCEPTRON (MLP)

- Stack TLU layers for more complicated functions
  - Input layer - passthrough
  - Hidden layer – intermediate TLU layer
  - Output layer – final fully connected TLU layer
- Lower layers – closer to input
- Upper layers – closer to output
- Deep neural network (DNN) has many hidden layers
Effective method to train a MLP developed in 1986

- Gradient Descent method with efficient gradient computation technique
- Single forward-backward pass through network to compute gradient of network error for all model parameters
  - Can update all connection weights and bias terms

Backpropagation uses reverse-mode autodiff to automatically compute gradients (Appendix D)
Process full dataset each epoch
  - Use mini-batch at each iteration – larger more efficient and more stable gradient but requires more memory

Mini-batch of input is sent through the MLP in a forward pass (from input to output prediction)
  - All intermediate results (from hidden layers) are saved for backward pass

Measure current network prediction error
  - Use of loss function to define error metric

Compute contribution of each connection to the total error
  - Performed backward from output through hidden layers back to input using the chain rule

Perform Gradient Descent step to adjust all connection weights
  - Using the error gradients from the backward pass
ACTIVATION FUNCTIONS

- Cannot use step for activation since it has no gradient information
- Sigmoid (logistic) function
  - \( \sigma(z) = 1/(1 + \exp(-z)) \)
  - S-shaped between [0, 1]
- Hyperbolic tangent function
  - \( \tanh(z) = 2\sigma(2z) - 1 \)
  - Output between [-1,1] helps speed convergence
- Rectified Linear Unit function
  - \( \text{ReLU}(z) = \max(0, z) \)
  - Not differentiable, but works well and fast so popular

Activation functions add non-linearity!
Single output neuron

- Multivariate regression requires an output neuron for each output dimension
  - 2: (x,y) for center of object
  - 4: (x,y,h,w) for a bounding box around object

Output activation

- No activation – no limits on output range of value
- ReLU or softplus (smooth ReLU) – positive output only
- Scaled sigmoid/tanh – fixed output range

Loss function

- Mean squared error (L2 norm)
- Mean absolute error (L1 norm) when there are a lot of outliers
- Huber loss is a combination

Regression MLP summary

Table 10.1. Typical regression MLP architecture

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td># input neurons</td>
<td>One per input feature (e.g., 28 x 28 = 784 for MNIST)</td>
</tr>
<tr>
<td># hidden layers</td>
<td>Depends on the problem, but typically 1 to 5</td>
</tr>
<tr>
<td># neurons per hidden layer</td>
<td>Depends on the problem, but typically 10 to 100</td>
</tr>
<tr>
<td># output neurons</td>
<td>1 per prediction dimension</td>
</tr>
<tr>
<td>Hidden activation</td>
<td>ReLU (or SELU, see Chapter 11)</td>
</tr>
<tr>
<td>Output activation</td>
<td>None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)</td>
</tr>
<tr>
<td>Loss function</td>
<td>MSE or MAE/Huber (if outliers)</td>
</tr>
</tbody>
</table>
CLASSIFICATION MLPs I

- Single class (binary) – single output neuron
  - Output between [0,1] using sigmoid
  - Estimate probability of positive class (confidence)

- Multilabel binary – output neuron for every binary classification
  - Output between [0,1] using sigmoid
  - Output probabilities do not sum to one
  - Combinational output space
Multiclass classification – multiple possible classes (e.g. number 0-9)
- Each input instance can only belong to a single class (>2)
- One output neuron per class
- Softmax activation on the full output layer (Chapter 4 pg 148)
  \[
  \hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_j \exp(s_j(x))}
  \]
  \[
  s_k(x) = (\theta^{(k)})^T x
  \]
  - Estimated probabilities between [0,1] and sum to 1
- Cross entropy loss
  \[
  J(\theta) = -\frac{1}{m} \sum_i \sum_k y_k^{(i)} \log(\hat{p}_k^{(i)})
  \]
  - Penalizes models with low probability estimate for the ground truth class

Classification summary
FINE-TUNING HYPERPARAMETERS

- Many hyperparameters must be tweaked for good model performance
- Grid search can evaluate different hyperparameter combinations → slow
  - Book gives other libraries for hyperparam optimization
  - These typically explore more in good hyperparameter space
- Number of hidden layers → deeper is better
  - Transfer learning – reuse lower layers from network trained on large dataset (good initialization and avoid cost of learning from scratch)
- Number of neurons per hidden layers → use fixed size
- Activation function → ReLU works well
- Learning rate – very important parameter, need learning schedule
- Optimizer – more than just mini-batch gradient descent (e.g. Adam)
- Batch size – significant impact on model performance and training time
  - Large batch – efficiently process for reduced training time → maximize for GPU with learning rate warm-up (schedule)
  - Small batch – more stable early in learning and good generalization