Artificial Neural Networks and Support Vector Machines

CS 486/686: Introduction to Artificial Intelligence
Outline

• What is a Neural Network?
  - Perceptron learners
  - Multi-layer networks

• What is a Support Vector Machine?
  - Maximum Margin Classification
  - The kernel trick
  - Regularization
Introduction

• Machine learning algorithms can be viewed as approximations of functions that describe the data

• In practice, the relationships between input and output can be extremely complex.

• We want to:
  • Design methods for learning arbitrary relationships
  • Ensure that our methods are efficient and do not overfit the data

• Today we'll discuss two modern techniques for learning arbitrary complex functions
Artificial Neural Nets

- Idea: The humans can often learn complex relationships very well.
- Maybe we can simulate human learning?
Human Brains

- A brain is a set of densely connected neurons.
- A neuron has several parts:
  - Dendrites: Receive inputs from other cells
  - Soma: Controls activity of the neuron
  - Axon: Sends output to other cells
  - Synapse: Links between neurons
Human Brains

- Neurons have two states
  - Firing, not firing
- All firings are the same
- Rate of firing communicates information (FM)
- Activation passed via chemical signals at the synapse between firing neuron's axon and receiving neuron's dendrite
- **Learning** causes changes in how efficiently signals transfer across specific synaptic junctions.
Artificial Brains?

- Artificial Neural Networks are based on very early models of the neuron.

- Better models exist today, but are usually used theoretical neuroscience, not machine learning
Artificial Brains?

- An artificial Neuron (McCulloch and Pitts 1943)

  Link ~ Synapse
  Weight ~ Efficiency
  Input Fun. ~ Dendrite
  Activation Fun. ~ Soma
  Output = Fire or not

\[ a_0 = -1 \]
\[ W_{0,i} \]
\[ W_{j,i} \]
\[ a_j \]
\[ \sum_{i} \]
\[ g(\text{in}) \]
• **Collection** of simple artificial neurons.
  • Weights $W_{i,j}$ denote strength of connection from $i$ to $j$
  • Input function: $in_i = \sum_j W_{i,j} \times a_j$
  • Activation Function: $a_i = g(in_i)$
Activation Function

• Activation Function: \( a_i = g(in_i) \)
• Should be non-linear (otherwise, we just have a linear equation)
• Should mimic firing in real neurons
  - Active (\( a_i \sim 1 \)) when the "right" neighbors fire the right amounts
  - Inactive (\( a_i \sim 0 \)) when fed "wrong" inputs
Common Activation Functions

Threshold Function

\[ g(in_i) \]

Weights determine threshold

Sigmoid Function

\[ g(in_i) = \frac{1}{1 + e^{-in_i}} \]
Logic Gates

- It is possible to construct a universal set of logic gates using the neurons described (McCulloch and Pitts 1943)
Logic Gates

• It is possible to construct a universal set of logic gates using the neurons described (McCulloch and Pitts 1943)
Network Structure

- **Feed-forward ANN**
  - Direct *acyclic* graph
  - No internal state: maps inputs to outputs.

- **Recurrant ANN**
  - Directed *cyclic* graph
  - Dynamical system with an internal state
  - Can *remember* information for future use
Example

\[ a_5 = g(W_{3,5} \cdot a_5 + W_{4,5} \cdot a_4) \]
Example

\[ a_5 = g(W_{3,5} \cdot a_5 + W_{4,5} \cdot a_4) \]

\[ a_5 = g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \]
Outline

• What is a Neural Network?
  - Perceptron learners
  - Multi-layer networks

• What is a Support Vector Machine?
  - Maximum Margin Classification
  - The kernel trick
  - Regularization
Perceptrons

Single layer feed-forward network
Perceptrons

Can learn only linear separators
Training Perceptrons

- Learning means adjusting the weights
  - Goal: minimize loss of fidelity in our approximation of a function

- How do we measure loss of fidelity?
  - Often: Half the sum of squared errors of each data point

\[ E = \sum_{i} 0.5(y_i - h_W(x_i))^2 \]
Gradient Descent

\[
\frac{\partial E}{\partial W_k} = \frac{\partial}{\partial W_k} \sum_i 0.5(y_i - h_W(x_i))^2
\]
Gradient Descent

\[
\frac{\partial E}{\partial W_k} = \frac{\partial}{\partial W_k} \sum_i 0.5(y_i - h_W(x_i))^2 \\
\frac{\partial E}{\partial W_k} = \sum_i \frac{\partial}{\partial W_k} 0.5(y_i - h_W(x_i))^2
\]
Gradient Descent

\[
\frac{\partial E}{\partial W_k} = \frac{\partial}{\partial W_k} \sum_i 0.5(y_i - h_W(x_i))^2
\]

\[
\frac{\partial E}{\partial W_k} = \sum_i \frac{\partial}{\partial W_k} 0.5(y_i - h_W(x_i))^2
\]

\[
\frac{\partial E}{\partial W_k} = \sum_i 0.5 \cdot 2 \cdot (y_i - h_W(x_i)) \frac{\partial}{\partial W_k} (y - g(\sum_j W_j x_{i,j}))
\]
Gradient Descent

\[
\frac{\partial E}{\partial W_k} = \frac{\partial}{\partial W_k} \sum_i 0.5(y_i - h_W(x_i))^2
\]

\[
\frac{\partial E}{\partial W_k} = \sum_i \frac{\partial}{\partial W_k} 0.5(y_i - h_W(x_i))^2
\]

\[
\frac{\partial E}{\partial W_k} = \sum_i 0.5 \cdot 2 \cdot (y_i - h_W(x_i)) \frac{\partial}{\partial W_k}(y - g(\sum_j W_j x_{i,j}))
\]

\[
\frac{\partial E}{\partial W_k} = \sum_i (y_i - h_W(x_i))(-g'(w \cdot x_i)) \frac{\partial}{\partial W_k} w \cdot x_i
\]
Gradient Descent

\[
\frac{\partial E}{\partial W_k} = \frac{\partial}{\partial W_k} \sum_i 0.5(y_i - h_W(x_i))^2 \\
\frac{\partial E}{\partial W_k} = \sum_i \frac{\partial}{\partial W_k} 0.5(y_i - h_W(x_i))^2 \\
\frac{\partial E}{\partial W_k} = \sum_i 0.5 \cdot 2 \cdot (y_i - h_W(x_i)) \frac{\partial}{\partial W_k} (y - g(\sum_j W_j x_{i,j})) \\
\frac{\partial E}{\partial W_k} = \sum_i (y_i - h_W(x_i)) (-g'(w \cdot x_i) \frac{\partial}{\partial W_k} w \cdot x_i) \\
\frac{\partial E}{\partial W_k} = \sum_i (y_i - h_W(x_i)) (-g'(w \cdot x_i) \cdot x_{i,k})
\]
Learning Algorithm

- Repeat for "some time"

- For each example i:

\[
I \leftarrow w \cdot x_i \\
E \leftarrow y_i - g(I) \\
W_j \leftarrow W_j + \alpha(E \cdot g'(I) \cdot x_{i,j}) \forall j
\]
Outline

• What is a Neural Network?
  - Perceptron learners
  - Multi-layer networks

• What is a Support Vector Machine?
  - Maximum Margin Classification
  - The kernel trick
  - Regularization
Multilayer Networks

- Minsky’s 1969 book *Perceptrons* showed perceptrons could not learn XOR.
- At the time, no one knew how to train deeper networks.
- Most ANN research abandoned.
Multilayer Networks

- Any continuous function can be learned by an ANN with just one hidden layer (if the layer is large enough).
Training Multilayer Nets

• For weights from hidden to output layer, just use Gradient Descent, as before.

\[ \Delta_i = E \cdot g'(I) \]
\[ W_{j,i} = W_{j,i} + \alpha \Delta_i a_j \]

• For weights from input to hidden layer, we have a problem: What is \( y \)?

\[ E = \sum_i 0.5(y_i - h_W(x_i))^2 \]
Back Propigation

• Idea: Each hidden layer caused some of the error in the output layer.

• Amount of error caused should be proportionate to the connection strength.

\[ \Delta_i = E \cdot g'(I) \]
\[ W_{j,i} = W_{j,i} + \alpha \Delta_i a_j \]
\[ \Delta_j = g'(I) \cdot \sum_i W_{j,i} \Delta_i \]
\[ W_{k,j} = W_{k,j} + \alpha \Delta_j x_k \]
Back Propigation

• Repeat for "some time":

• Repeat for each example:
  - Compute Deltas and weight change for output layer, and update the weights.
  - Repeat until all hidden layers updated:
    - Compute Deltas and weight change for the deepest hidden layer not yet updated, and update it.
When to use ANNs

- When we have high dimensional or real-valued inputs, and/or noisy (e.g. sensor data)
- Vector outputs needed
- Form of target function is unknown (no model)
- Not import for humans to be able to understand the mapping
Drawbacks of ANNs

• Unclear how to interpret weights, especially in many-layered networks.

• How deep should the network be? How many neurons are needed?

• Tendency to overfit in practice (very poor predictions outside of the range of values it was trained on)
Outline

• What is a Neural Network?
  - Perceptron learners
  - Multi-layer networks

• What is a Support Vector Machine?
  - Maximum Margin Classification
  - The kernel trick
  - Regularization
• We want an algorithm that can learn \textit{arbitrary} functions (like multilayered ANNs)
• But, it shouldn't require picking a lot of \textit{parameters}, and should \textit{extrapolate} in a reasonable, predictable way.
SVMs

• Support Vector Machines (SVMs) can learn arbitrary functions.
• SVMs also extrapolate in a predictable, controllable way, and have just two parameters to set.
• Often the first choice of modern practitioners.
Maximum Margin

• Idea: We do not know where the exact decision boundary is, so pick "safest" one.

• Best separating hyperplane is furthest from any point, but still partitions cleanly
Support Vectors

• A maximum margin hyperplane is defined by its "Support Vectors"

• These are the points that "support" the plane on either side

• Find the optimal SVs using Quadratic Programming
Finding the SVs

• Find a weight $\alpha_j$ for each point.

$$\arg \max_\alpha \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_j (x_j \cdot x_k)$$

s.t.

$$\alpha_j \geq 0 \forall j$$

$$\sum_j \alpha_j y_j = 0$$

• $y_i =$ label of example i (-1 for negative, +1 for positive)
• $x_i =$ input vector for example i
Using the SVs

• To classify a new point, figure out whether it's above or below the plane

$$H(x) = sign \left( \sum_{j} \alpha_j y_j (x \cdot x_j) - b \right)$$

• Only need to store $x_j$ vectors for points with non-zero weight, so model can be compact.
Non-Linear Learning

• If we're just finding the best line, how can we learn arbitrary functions?

• Insight: A high-dimensional line can be projected into any shape in lower dimensions.
Non-Linear Learning
Outline

- What is a Neural Network?
  - Perceptron learners
  - Multi-layer networks

- What is a Support Vector Machine?
  - Maximum Margin Classification
  - The kernel trick
  - Regularization
The Kernel Trick

- **Solution:** Define a function $F(x)$ that maps $x$ to a new feature space. Learn an SVM using $F(x)$ instead of $x$.

- **Problems:**
  - What should $F(x)$ be?
  - If $F(x)$ has many more dimensions than $x$, then the dot product will become expensive to compute.
The Kernel Trick

• We call any function of the inner product of two points a "kernel".

• Examples:
  - "Linear kernel": \(K(x_i,x_j) = (x_i \cdot x_j)\)
  - Polynomial Kernel: \(K(x_k, x_j) = (1 + x_j \cdot x_k)^d\)
The Kernel Trick

• Mercer's Theorem: Define a matrix $K$ over every $x_i, x_j$ in your data, such that

$$K_{i,j} = K(x_i, x_j)$$

• If $K$ is positive definite, then there exists some feature space $F(x)$ such that:

$$K(x_k, x_j) = F(x_j) \cdot F(x_k)$$
Example

\[ K(x_i, x_j) = (x_i \cdot x_j)^2 \]
Example

\[ K(x_i, x_j) = (x_i \cdot x_j)^2 \]

\[ K(x_i, x_j) = ((x_{i,1} x_{j,1}) + (x_{i,2} x_{j,2}))^2 \]
Example

\[ K(x_i, x_j) = (x_i \cdot x_j)^2 \]

\[ K(x_i, x_j) = \left( (x_{i,1} x_{j,1}) + (x_{i,2} x_{j,2}) \right)^2 \]

\[ K(x_i, x_j) = (x_{i,1} x_{j,1})^2 + (x_{i,2} x_{j,2})^2 + 2x_{i,1} x_{i,2} x_{j,1} x_{j,2} \]
Example

\[ K(x_i, x_j) = (x_i \cdot x_j)^2 \]

\[ K(x_i, x_j) = ((x_{i,1}x_{j,1}) + (x_{i,2}x_{j,2}))^2 \]

\[ K(x_i, x_j) = (x_{i,1}x_{j,1})^2 + (x_{i,2}x_{j,2})^2 + 2x_{i,1}x_{i,2}x_{j,1}x_{j,2} \]

\[ F([x_1, x_2]) = [x_1^2, x_2^2, \sqrt{2}x_1x_2] \]
Example

\[ K(x_i, x_j) = (x_i \cdot x_j)^2 \]

\[ K(x_i, x_j) = ((x_{i,1} x_{j,1}) + (x_{i,2} x_{j,2}))^2 \]

\[ K(x_i, x_j) = (x_{i,1} x_{j,1})^2 + (x_{i,2} x_{j,2})^2 + 2x_{i,1} x_{i,2} x_{j,1} x_{j,2} \]

\[ F([x_1, x_2]) = [x_1^2, x_2^2, \sqrt{2} x_1 x_2] \]

\[ K(x_i, x_j) = F(x_i) \cdot F(x_j) \]
The Kernel Trick

• Insight: data points themselves are never used in optimization, only \((x_i \cdot x_j)\)

• Mercer's Theorem means that, if we replace every \((x_i \cdot x_j)\) by \(K_{i,j} = K(x_i, x_j)\) then we are finding the optimal support vectors in some other feature space \(F(x)\)!
The Kernel Trick

• Since we never need to compute $F(x)$ explicitly, the algorithm runs equally fast whether we use a kernel or not

• Not true of other methods. If a decision tree wants to work in a larger space, it needs to split on more attributes

• This is one of the main advantages of SVMs
Some Useful Kernels

- Polynomial kernel, $F()$ exponentially larger as $d$ increases:
  \[
  K(x_k, x_j) = (1 + x_j \cdot x_k)^d
  \]

- RBF kernel, $F()$ has *infinite* dimensionality:
  \[
  K(x_k, x_j) = \exp(\gamma \sum_l (x_{k,l} - x_{j,l})^2)
  \]
Picking Kernels

• More dimensions is not always better
• RBF often good for image processing and related dense domains with smooth, circular or elliptical decision boundaries
• Linear kernel ($d = 1$) often better for sparse domains like text
• Depends on your problem
Outline

• What is a Neural Network?
  - Perceptron learners
  - Multi-layer networks

• What is a Support Vector Machine?
  - Maximum Margin Classification
  - The kernel trick
  - Regularization
Regularization

• An SVM can usually find an exact decision boundary (if the feature space is big enough)

• But the exact boundary may over-fit the data
Example
Regularization

• To avoid this, we add regularization parameter $C$

• Cost for misclassifying an example is $C$

• Cost for increasing total SV weight by 1 is $1/C$.

• Right choice of $C$ depends on your problem
Parameter Selection

• Pick a kernel

• Pick a regularization cost $C$, directly manipulates overfitting vs. underfitting
When to use an SVM

• Often a good first choice for classification
• Outcome is binary, and there are not too many features or data points
• Have some notion of the proper kernel
When to avoid an SVM

- "Big Data" problems (tons of data, speed is very important, QP too slow)
- Human interpretation is important
- Problem has many outputs
You should be able to:

• Describe what a Neural Network is, and how to train one from data using back-propagation.

• Describe what maximum margin hyperplanes are and support vectors are.

• Explain how an SVM can learn non-linear functions efficiently, using the kernel trick.
Announcements

• A5 Due Today
• A6 Out Today (Fun!)
• Thursday: Computational Learning Theory
  - R&N Ch 18.5
  - P & M Ch 7.7.2
Perceptrons

- Minsky's 1969 book *Perceptrons* showed perceptrons could not learn XOR.

- Led to collapse of neural approaches to AI, and the rise of symbolic approaches during the AI winter. Note that we already knew the neural model was Turing-complete though!

- Now thought to have been a *philosophically* motivated shift, rather than one required by the science.