Neural Networks and Ensemble Learning
What if you want your neural network to predict **continuous** outputs rather than +1/-1 (i.e., perform regression)?

E.g., Teaching a network to drive
**Continuous Outputs with Sigmoid Networks**

Output: \[ v = g(w^T u) = g\left(\sum_i w_i u_i\right) \]

\[ u = (u_1 \quad u_2 \quad u_3)^T \]

**Sigmoid output function:**

\[ g(a) = \frac{1}{1 + e^{-\beta a}} \]

Parameter \( \beta \) controls the slope
Learning the weights

**Given:** Training data (input $u$, desired output $d$)

**Problem:** How do we learn the weights $w$?

**Idea:** *Minimize squared error* between network’s output and desired output:

$$E(w) = (d - v)^2$$

where $v = g(w \cdot u)$

Starting from random values for $w$, want to change $w$ so that $E(w)$ is minimized – How?
Learning by Gradient-Descent

(opposite of “Hill-Climbing”)

Change \( w \) so that \( E(w) \) is minimized

- Use Gradient Descent: Change \( w \) in proportion to
  \(-dE/dw\) (why?)

\[
E(w) = (d - v)^2 \quad v = g(w \cdot u)
\]

\[
w \rightarrow w - \varepsilon \frac{dE}{dw}
\]

\[
\frac{dE}{dw} = -2(d - v) \frac{dv}{dw} = -2(d - v)g'(w \cdot u)u
\]

Derivative of sigmoid

\( \text{delta} = \text{error} \)

Also known as the “delta rule” or “LMS (least mean square) rule”
But wait!

This rule is for a one layer network

• One layer networks are not that interesting!! (remember XOR?)

What if we have multiple layers?
Learning Multilayer Networks

\[ v_i = g \left( \sum_j W_{ji} g \left( \sum_k w_{kj} u_k \right) \right) \]

Start with random weights \( W, w \)

Given input vector \( u \), network produces output vector \( v \)

Use gradient descent to find \( W \) and \( w \) that minimize total error over all output units (labeled \( i \)):

\[ E(W, w) = \frac{1}{2} \sum_i (d_i - v_i)^2 \]

This leads to the famous “Backpropagation” learning rule
Backpropagation: Output Weights

\[ E(W, w) = \frac{1}{2} \sum_i (d_i - v_i)^2 \]

\[ v_i = g(\sum_j W_{ji} x_j) \]

Learning rule for hidden-output weights \( W \):

\[ W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}} \left\{ \text{gradient descent} \right\} \]

\[ \frac{dE}{dW_{ji}} = -(d_i - v_i) g'(\sum_j W_{ji} x_j) x_j \left\{ \text{delta rule} \right\} \]
Backpropagation: Hidden Weights

\[ E(W, w) = \frac{1}{2} \sum_i (d_i - v_i)^2 \]

\[ v_i^m = g(\sum_j W_{ji} x_j) \]

\[ x_j = g(\sum_k w_{kj} u_k) \]

Learning rule for input-hidden weights \( w \): 

\[ w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \quad \text{But:} \quad \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \{ \text{chain rule} \} \]

\[ \frac{dE}{dw_{kj}} = \left[ - \sum_i (d_i - v_i) g'(\sum_j W_{ji} x_j) W_{ji} \right] \cdot \left[ g'(\sum_k w_{kj} u_k) u_k \right] \]
Examples: Pole Balancing and Backing up a Truck
(courtesy of Keith Grochow)

• Neural network learns to balance a pole on a cart
  • Input: $x_{\text{cart}}, v_{\text{cart}}, \theta_{\text{pole}}, v_{\text{pole}}$
  • Output: New force on cart

• Network learns to back a truck into a loading dock
  • Input: $x, y, \theta$ of truck
  • Output: Steering angle
Ensemble Learning

Sometimes each learning technique yields a different “hypothesis” (function)

But no perfect hypothesis...

Could we combine several imperfect hypotheses to get a better hypothesis?
Why Ensemble Learning?

Wisdom of the Crowds...
Example

Combine 3 linear classifiers
⇒ More complex classifier

This line is one simple classifier saying that everything to the left is + and everything to the right is -
Ensemble Learning: Motivation

Analogies:

• Elections combine voters' choices to pick a good candidate (hopefully)
• Committees combine experts' opinions to make better decisions
• Students working together on a capstone project

Intuitions:

Individuals make mistakes but the "majority" may be less likely to

Individuals often have partial knowledge; a committee can pool expertise to make better decisions
Ensemble Technique 1: Bagging

Combine hypotheses (classifiers) via majority voting

instance

\[ X \]

classification

\[ \text{Majority}(h_1(x), h_2(x), h_3(x), h_4(x), h_5(x)) \]

Ensemble of hypotheses

For the classification to be wrong, at least 3 out of 5 hypotheses have to be wrong
Bagging: Details

1. Generate $m$ new training datasets by sampling with replacement from the given dataset

2. Train $m$ classifiers $h_1, \ldots, h_m$ (e.g., decision trees), one from each newly generated dataset

3. Classify a new input by running it through the $m$ classifiers and choosing the class that receives the most “votes”

Example: *Random forest* = Bagging with $m$ decision tree classifiers, each tree constructed from random subset of attributes
Bagging: Analysis

• Assumptions:
  - Each $h_i$ makes error with probability $p$
  - The hypotheses are independent

• Majority voting of $n$ hypotheses:
  - $k$ hypotheses make an error: $\binom{n}{k} p^k (1-p)^{n-k}$
  - Majority makes an error: $\sum_{k>n/2} \binom{n}{k} p^k (1-p)^{n-k}$
  - With $n=5$, $p=0.1 \Rightarrow \text{err(majority)} < 0.01$

Error probability went down from 0.1 to 0.01!
Weighted Majority Voting

In practice, hypotheses rarely independent

Some hypotheses have less errors than others ⇒ all votes are not equal!

Idea: Let’s take a weighted majority

How do we compute the weights?
Next Time

• Weighted Majority Ensemble Classification
  • Boosting
• Survey of AI Applications
• To Do:
  • Project 4 due tonight!
  • Finish Chapter 18