COMP9444: Neural Networks
Committee Machines
Committee Machines

![Diagram of Committee Machines]

- Input $x$
- Filter
- Machine 1: $y_1$
- Machine 2: $y_2$
- ... (up to Machine $T$: $y_T$)
- Combiner
- Output $y$
Motivation

If several classifiers are trained on (subsets of) the same training items, can their outputs be combined to produce a composite machine with better accuracy than the individual classifiers?
Outline

- Static structures (Combiner does not make direct use of the Input)
  - Ensemble Averaging
  - Bagging
  - Boosting

- Dynamic structures (Combiner does make direct use of the Input)
  - Mixture of Experts
  - Hierarchical Mixture of Experts
Ensemble Experiment

Distinguish between two classes, each generated according to a Gaussian distribution:

Class 1:

\[ \mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \sigma_1^2 = 1 \]

Class 2:

\[ \mu_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \sigma_2^2 = 4 \]
Ensemble Experiment

- Ten neural networks
- MLPs with 2 hidden nodes
- trained on same 500 patterns
- each with different initial weights
- same learning rate and momentum
- tested on the same 500 (new) patterns
- individual networks deliberately “overtrained”

<table>
<thead>
<tr>
<th>classifier</th>
<th>% correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net 1</td>
<td>80.65</td>
</tr>
<tr>
<td>Net 2</td>
<td>76.91</td>
</tr>
<tr>
<td>Net 3</td>
<td>80.06</td>
</tr>
<tr>
<td>Net 4</td>
<td>80.47</td>
</tr>
<tr>
<td>Net 5</td>
<td>80.44</td>
</tr>
<tr>
<td>Net 6</td>
<td>76.89</td>
</tr>
<tr>
<td>Net 7</td>
<td>80.55</td>
</tr>
<tr>
<td>Net 8</td>
<td>80.47</td>
</tr>
<tr>
<td>Net 9</td>
<td>76.91</td>
</tr>
<tr>
<td>Net 10</td>
<td>80.38</td>
</tr>
</tbody>
</table>
Ensemble Experiment

- The average probability of correct classification for the individual networks is 79.37%.

- If we instead base our classification on the sum of the outputs of the individual networks, the probability of correct classification rises, but only marginally, to 80.27%

Question:

Can we do better?

Answer:

Yes, by feeding a different distribution of inputs to each classifier.
Weak and Strong Learners

- A weak learner is one that is only guaranteed to achieve an error rate slightly less than what would be achieved by random guessing.
- A strong learner is one which can achieve an error rate arbitrarily close to zero, in the PAC learning sense.

Question:

Can a weak learner be “boosted” into a strong learner, by applying it repeatedly to different subsets of the training data?

Answer:

Yes!
Boosting by Filtering

Assume you have access to an unlimited stream of training examples:

- The first classifier $C_1$ is generated by applying the weak learner to $n$ training examples.

- $C_1$ is used as a filter to collect $n$ new training examples: A fair coin is flipped: If head turns up, the next example from the stream is collected that is incorrectly classified by $C_1$. If tail turns up, the next example is collected that is correctly classified by $C_1$. Generate a new classifier $C_2$ using the weak learner and the collected training examples.

- Generate a third classifier by using the weak learner and a training sample of $n$ examples created by just retaining those examples which are differently classified by $C_1$ and $C_2$. 
Boosting by Filtering

- of the total number of items seen, only a subset are used for the actual training of the classifiers; the procedure filters out items that are easy to learn and focuses on those that are hard to learn.

- in the original work (Schapire, 1990) a voting mechanism was used to combine the classifiers, but it has later been shown that summing the outputs of the individual classifiers gives better performance.

- it can be proved that if the error rate for the individual classifiers is \( \varepsilon < 1/2 \), then the error rate for the committee machine is less than

\[
g(\varepsilon) = 3\varepsilon^2 - 2\varepsilon^3
\]

therefore, by applying the boosting algorithm recursively, the error rate can be made arbitrarily close to zero.
Discussion

- Boosting by Filtering has the drawback that it requires a huge number of training items
- there are alternative algorithms which use fewer items, by judiciously re-using data:
  - Bagging
  - AdaBoost
Bagging

- start with a training set of $N$ items
- for each classifier, choose a set of $N$ items from the original set with replacement; this means that some items can be chosen more than once, while others are left out
- train each classifier on the chosen items
- once all classifiers have been trained, new (test set) items are classified by majority vote, or by averaging the outputs of the individual classifiers for numerical outputs
AdaBoost

- given: $N$ training items $(\tilde{x}_1, d_1) \ldots (\tilde{x}_N, d_N)$
- train a series of learners $C_1 \ldots C_T$ producing hypotheses $f_1 \ldots f_T$
- training items for $C_n$ chosen using distribution $D_n$
- initialize $D_1(i) = \frac{1}{N}$ for $i = 1 \ldots N$
- set
  \[ \beta_n = \frac{\varepsilon_n}{1 - \varepsilon_n}, \quad \text{where} \quad \varepsilon_n = \text{training error of } f_n \]
- update
  \[ D_{n+1}(i) = \frac{D_n(i)}{Z_n} \times \begin{cases} \beta_n, & \text{if } f_n(\tilde{x}_i) = d_i \\ 1, & \text{otherwise} \end{cases} \]

where $Z_n$ is a normalizing constant.
AdaBoost

- output the final hypothesis:

\[
f(\vec{x}) = \text{sign} \left( \sum_{n=1}^{N} f_n(\vec{x}) \log \frac{1}{\beta_n} \right).
\]
AdaBoost Generalization

- The **base learner** for AdaBoost could be any kind of learner (neural networks, decision trees, stumps ... )
- With AdaBoost, as with SVM’s, the test error often continues to decrease even after the training error has already reached zero.
- This goes against the traditional conception of bias-variance trade-off, Ockham’s Razor and overfitting.
- Although the number of “free parameters” is enormous, each additional degree of freedom is highly constrained.
Sensitivity to Errors

- AdaBoost, like SVM, is very sensitive to mislabeled data
- AdaBoost will assign enormous weight to incorrectly labeled items, and put huge effort into learning them
Mixture of Experts

\[ y = \sum_{i=1}^{2} p_i y_i \]

- \( y_1 \) from Expert Network 1
- \( y_2 \) from Expert Network 2
- \( p_1 \) and \( p_2 \) from Gating Network

\[ \text{Input} \]
Mixture of Experts

- Each individual “expert” tries to approximate the target function on some subset of the input space
- The gating network tries to learn which expert(s) are best suited to the current input
- For each expert $k$, the gating network produces a linear function $u_k$ of the inputs.
- The outputs $g_1 \ldots g_K$ of the gating network are computed using the “softmax” principle:
  \[ g_k = \frac{\exp(u_k)}{\sum_j \exp(u_j)} \]
- In stochastic training, $g_k$ is treated as the probability of selecting expert $k$; for soft training, it is treated as a mixing parameter for expert $k$. 
Hierarchical Mixture of Experts
Hierarchical Mixture of Experts

- HME can be trained either by maximum likelihood estimation, or by the expectation maximization (EM) algorithm
- HME model is often seen as a “soft” version of decision trees