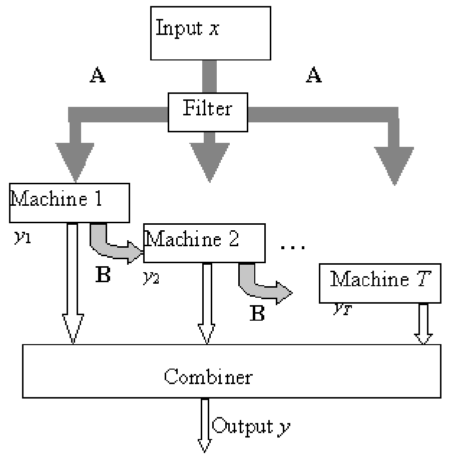
COMP9444: Neural Networks

Committee Machines, Recent Applications

©Anthony Knittel and Alan Blair, 2013

Committee Machines



©Anthony Knittel and Alan Blair, 2013

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?

©Anthony Knittel and Alan Blair, 2013

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

©Anthony Knittel and Alan Blair, 2013

Ensemble Experiment

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

Class 1:

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \sigma_1^2 = 1$$

Class 2:

$$\mu_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \qquad \sigma_2^2 = 4$$

©Anthony Knittel and Alan Blair, 2013

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"

classifier	% correct
Net 1	80.65
Net 2	76.91
Net 3	80.06
Net 4	80.47
Net 5	80.44
Net 6	76.89
Net 7	80.55
Net 8	80.47
Net 9	76.91
Net 10	80.38

©Anthony Knittel and Alan Blair, 2013

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.

©Anthony Knittel and Alan Blair, 2013

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!

©Anthony Knittel and Alan Blair, 2013

Boosting by Filtering

- the first classifier C_1 is trained on a set of N_1 examples
- repeat until N_1 items have been collected to train C_2:
  - flip a fair coin
  - if heads, keep "filtering out" items correctly classified by C_1; the first item incorrectly classified by C_1 is set aside for training C_2
  - if tails, instead filter out items incorrectly classified by C_1; the first item correctly classified by C_1 is set aside for training C_2
- once C_2 has been trained, items correctly classified by both C_1 and C_2 are filtered out, and the others set aside for training C_3 (until N_1 of them have been collected)

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 a voting mechanism, or by summing the outputs of the individual classifiers

AdaBoost

- given: N training items $(\vec{x}_1, y_1) \ldots (\vec{x}_N, y_N)$ where $x_i \in X$, $y_i \in \{-1, 1\}$
- train a series of learners $C_1, \ldots C_T$ producing hypotheses $h_1 \ldots h_T$
- training items for each learner C_t chosen using distribution $\mathcal{D}_t$
- initialize $\mathcal{D}_1(i) = \frac{1}{N}$ for $i = 1 \ldots N$

AdaBoost

- for each step $t = 1, \ldots, T$
  - train weak learner C_t using distribution $\mathcal{D}_t$
  - this produces hypothesis $h_t : X \to \{-1, 1\}$, with error ε_t
  - set

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \varepsilon_t}{\varepsilon_t})$$

  - update the distribution

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalising factor to produce a probability distribution

AdaBoost

- output the final hypothesis:

$$h'(x) = sign(\sum_t^T \alpha_t h_t(x))$$

Theorem: Assuming $\gamma_n = \frac{1}{2} - \varepsilon_n \geq 0$ for all n, then the training error of the final hypothesis is at most

$$2 \prod_{n=1}^{T} \sqrt{\varepsilon_n(1 - \varepsilon_n)} = \prod_{n=1}^{T} \sqrt{1 - 4\gamma_n{}^2} \leq \exp(-2 \sum_{n=1}^{T} \gamma_n{}^2)$$

AdaBoost Generalization

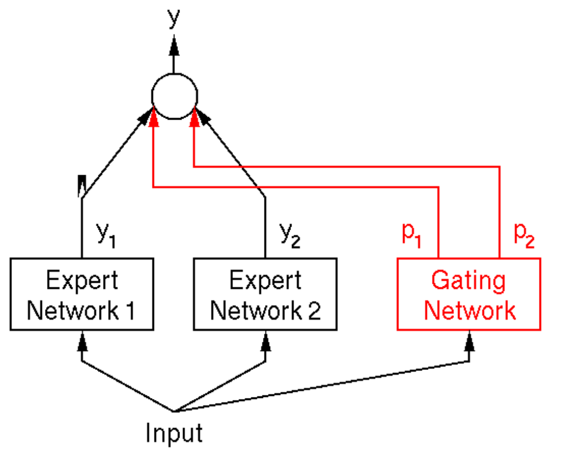
- the base learner for AdaBoost could be any kind of learner (neural networks, decision trees, stumps ...)
- with AdaBoost, as with SVM's, the test error often continues to decrease even after the training error has already reached zero
- this goes against the traditional conception of bias-variance tradeoff, Ockham's Razor and overfitting
- although the number of "free parameters" is enormous, each additional degree of freedom is highly costrained

Sensitivity to Errors

- AdaBoost, like SVM, is very sensitive to mislabled data
- AdaBoost will assign enormous weight to incorrectly labeled items, and put huge effort into learning them
- there are some alternative boosting algorithms which try to avoid this problem; the most principled approach is DOOM II – a special case of AnyBoost.

©Anthony Knittel and Alan Blair, 2013

Mixture of Experts



©Anthony Knittel and Alan Blair, 2013

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:

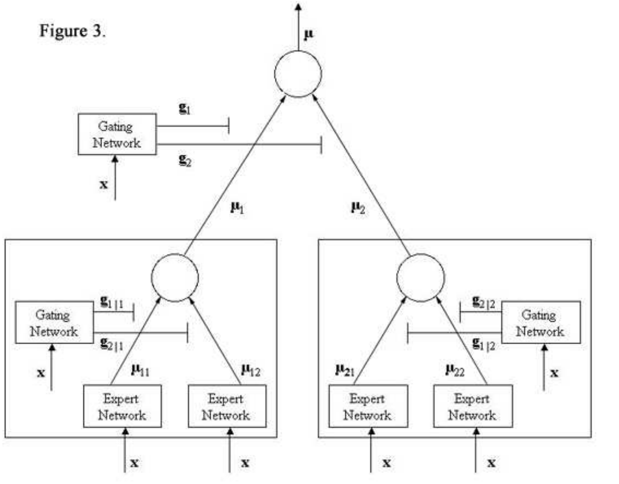
$$g_k = \frac{\exp(u_k)}{\sum_j \exp(u_j)}$$

- in stochastic training, g_k is treated as the probability of selecting expert k; otherwise it is treated as a mixing parameter for expert k.

©Anthony Knittel and Alan Blair, 2013

Hierachical Mixture of Experts

Figure 3.



©Anthony Knittel and Alan Blair, 2013

Hierachical 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

 ©Anthony Knittel and Alan Blair, 2013

Different Kinds of Modularity

- “Vertical” Modularity
  - partitioning of the input space
  - credit assignment easier, in principle
- “Horizontal” Modularity
  - output of one module becomes input of another module
  - credit assignment becomes really hard
  - still no known algorithms to do this automatically.

 ©Anthony Knittel and Alan Blair, 2013

Vertical Modularity

- Each expert tries to do full processing from input to output, but only for a limited range of inputs
- Effort can be divided arbitrarily between the “partitioner” and the individual “experts” – at one extreme, the partitioner does nothing; at the other extreme, the partitioner does everything and each expert just parrots a fixed answer
- Algorithms such as ME and HME attempt to partition the input space automatically, but with mixed success

 ©Anthony Knittel and Alan Blair, 2013

Other Modularity Issues

- “Structural” modularity
  - each module is a physically identifiable anatomical unit (spleen, liver, pancreas, etc.)
- “Functional” modularity
  - system is made up of different “functions”, which might share some of the same physical components
- What kind of modularity occurs in the brain?

 ©Anthony Knittel and Alan Blair, 2013

Summary

- Large man-made or evolved systems are always modular
- How can we get adaptive, machine learning systems to modularize automatically?
- This is a major open question, sometimes called the “scaling up” problem

©Anthony Knittel and Alan Blair, 2013

Recent Applications

- Techniques
- Examples
- Speech recognition
- Deep Networks for large scale image classification

©Anthony Knittel and Alan Blair, 2013

Training Techniques

- Scale inputs with mean 0 and standard deviation 1
- Antisymmetric activation functions have advantages (eg `tanh`)
- Alternative activation function: Rectified Linear Units (ReLUs)

$$f(x) = max(0,x)$$

©Anthony Knittel and Alan Blair, 2013

Dropout

- When neural networks are trained on small datasets, they often suffer from overfitting
- One reason for this, is that feature detectors have learned to work together based on what they have been trained on in the training set
- Complex co-adaptations can be developed between neurons, where a feature is only useful in the context of a number of other specific feature detectors
- “Dropout” is a technique to prevent the development of co-adaptations between neurons

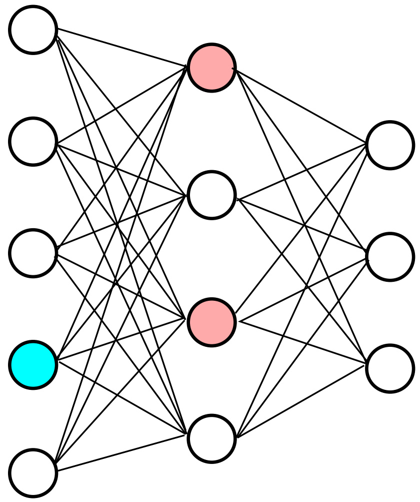
©Anthony Knittel and Alan Blair, 2013

Dropout

- During training, for each training step 50% of the neurons in the network are disabled.
- Dropped-out neurons do not participate in feed-forward activations, or in backpropagation
- Further advantages have been shown from setting 20% of the input layer neurons as inactive

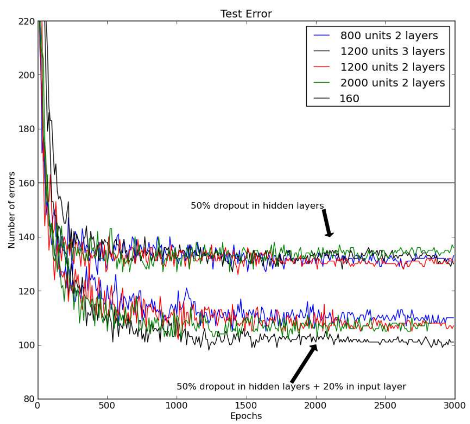
©Anthony Knittel and Alan Blair, 2013

Dropout



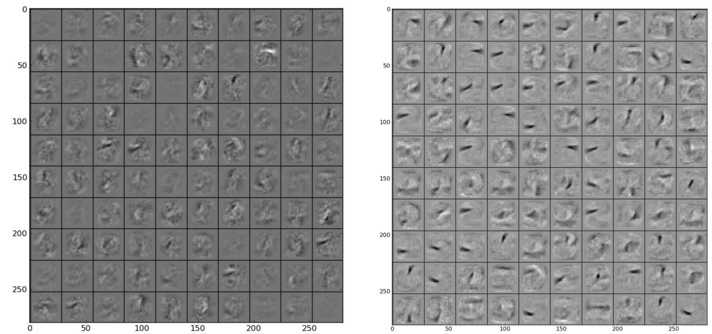
©Anthony Knittel and Alan Blair, 2013

Dropout



©Anthony Knittel and Alan Blair, 2013

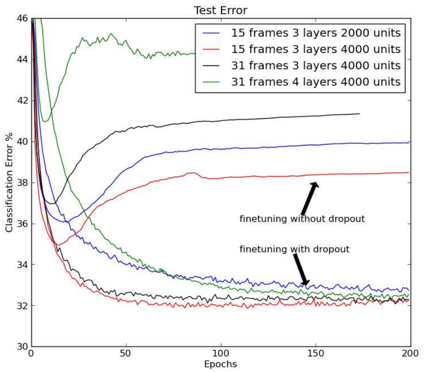
Dropout



Features produced on MNIST, using backprop, and with dropout

©Anthony Knittel and Alan Blair, 2013

Dropout

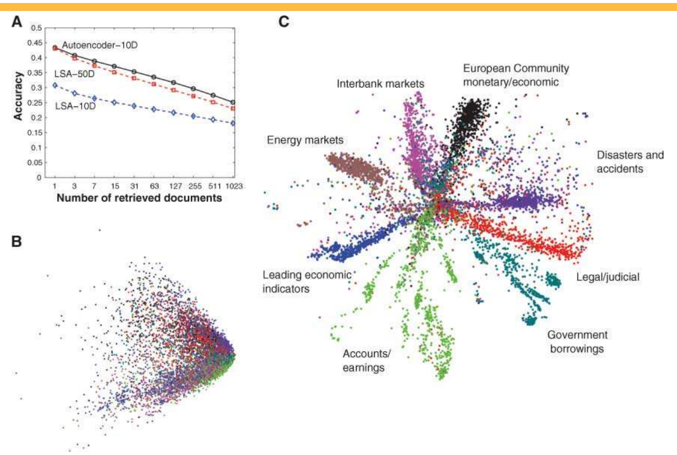


Generalisation on TIMIT speech recognition

Dropout

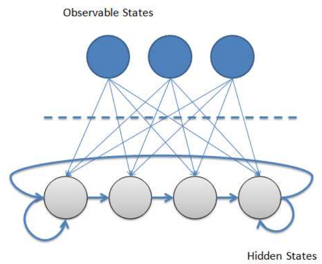
- Dropping out neurons is similar to capturing a mixture of models, using various combinations of subsets of the network
- Training time is approximately doubled
- When performing testing, the activation of each hidden unit is halved, which approximately represents an average of the child models
- This method is straightforward to implement, and can improve generalisation, useful for smaller datasets

Application- Natural Language Processing



Application- Speech Recognition

- "Context-Dependent Deep Neural Network Hidden Markov Model"
- Hidden Markov Model- learn relationships of transitions between hidden states, and between hidden states and observations

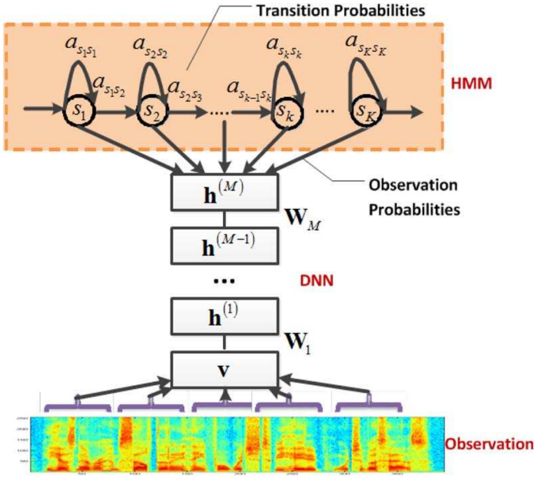


Application- Speech Recognition

- The HMM system can be used to infer the symbolic representation, based on the relationships between acoustic patterns and symbols, and probabilities of symbol sequences
- Hidden Markov Model- learn relationships of transitions between hidden states, and between hidden states and observations
- The Deep Neural Network is used to learn the probability distribution of symbolic states from audio
- Training is performed on tied triphones (Context Dependent)

©Anthony Knittel and Alan Blair, 2013

Application- Speech Recognition



©Anthony Knittel and Alan Blair, 2013

Application- Speech Recognition

- These probabilities are used to determine emission likelihoods for each state, that are used by the HMM to determine the most likely symbol sequence for a given audio sample
- 7 layers, 2048 hidden units at each layer
- Trained on 309 hours of training data
- Each layer pre-trained as a Restricted Boltzmann Machine
- Fine-tuned using 9304 triphone states (output layer)
- Improvement from 27.4% to 18.5% error (30% improvement)
- Demonstration: `http://www.youtube.com/watch?v=Nu-nlQqFCKg`

©Anthony Knittel and Alan Blair, 2013

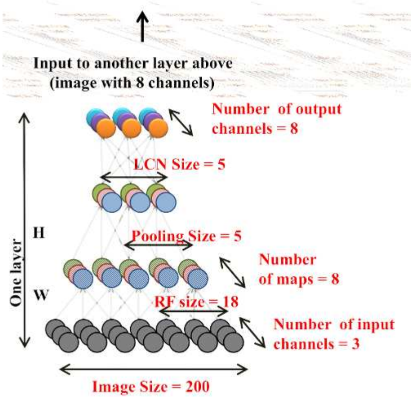
Application- Image Classification

- Convolutional Neural Networks are based on a fixed topology, using layers of specialised neurons
- “Building high-level features using large scale unsupervised learning”, study by researchers at Stanford and Google
- Large neural network, trained unsupervised. Not convolutional[a], so pooling can occur over different kinds of features.

©Anthony Knittel and Alan Blair, 2013

[a] a different form of weight sharing may be used

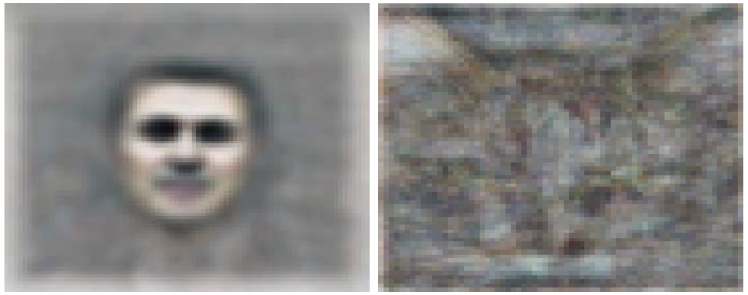
Application- Image Classification



Application- Image Classification

- 3 types of (sub) layers- a filter layer (18x18), a pooling layer (5x5) that provides a kind of averaging, and local contrast normalisation (5x5), that provides competition between neurons addressing the same region.
- 3 full layers are used (9 sub layers). Only the filter layer weights are used for training.
- Unsupervised learning is used, and the network acts as an autoencoder.
- Activations are passed through the network, from larger to smaller layers, and to perform reconstruction a reverse operation is used. (weights are not tied)

Application- Image Classification



Optimal stimulus for a face sensitive neuron and cat sensitive neuron

Application- Image Classification

- The network is trained on 10 million images, of size 200x200
- 1 billion connections
- Trained on a cluster of 1,000 machines (16,000) cores for 3 days
- After unsupervised pre-training, classification on 22,000 object categories with 15.8% accuracy
- `http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-c`

References

- “Improving neural networks by preventing co-adaptation of feature detectors”, Hinton et al 2012, `http://arxiv.org/abs/1207.0580`
- “Reducing the Dimensionality of Data with Neural Networks”, Hinton, Salakhutdinov 2007
- “Conversational Speech Transcription Using Context-Dependent Deep Neural Networks”, Seide 2011, `http://research.microsoft.com/apps/pub`
- “Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition”, Dahl 2012, `http://ieeexplore.ieee.org/stamp`

References

- “Scientists See Promise in Deep-Learning Programs”, `http://www.nytimes.com/20`
- “ ImageNet Classification with Deep Convolutional Neural Networks”, Krizhevsky et al 2012, `http://nips.cc/Conferences/2012/Program/eve`
- “Building high-level features using large scale unsupervised learning”, Le et al 2011, `http://rxiv.org/abs/1112.6209`
- “In a Big Network of Computers, Evidence of Machine Learning”, `http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-c`