MPhil in Machine Learning and Machine Intelligence

Module MLMI2: Speech Recognition

L8: Neural Networks for ASR Acoustic Models

Phil Woodland
pcw@eng.cam.ac.uk

Michaelmas 2021

Cambridge University Engineering Department

Introduction

This lecture concentrates on the use of **artificial neural networks (ANNs)** for speech recognition acoustic modelling. It concentrates on feed-forward **deep neural networks (DNNs)** and also introduces some standard **recurrent neural network (RNN)** models.

The use use of DNNs and RNNs are also covered in module 4F10.

We will first briefly review multi-layer perceptrons (MLPs) and feed-forward DNNs and include

- Overall architecture and activation functions
- Training Criterion for classification and SGD training
- Some practical issues
- Deep Neural Networks

Then we will briefly examine

- DNNs for speech recognition and DNN-HMM hybrids
- Standard feed-forward DNN configuration
- Typical DNN build procedure
- Performance of feed-forward DNNs for ASR
- Recurrent networks

Note that a future lecture will cover more advanced forms of ANNs for acoustic modelling.

Multi-layer Perceptron (MLP): feed-forward neural network

A **layered** feed-forward structure: performs **universal function approximation**.

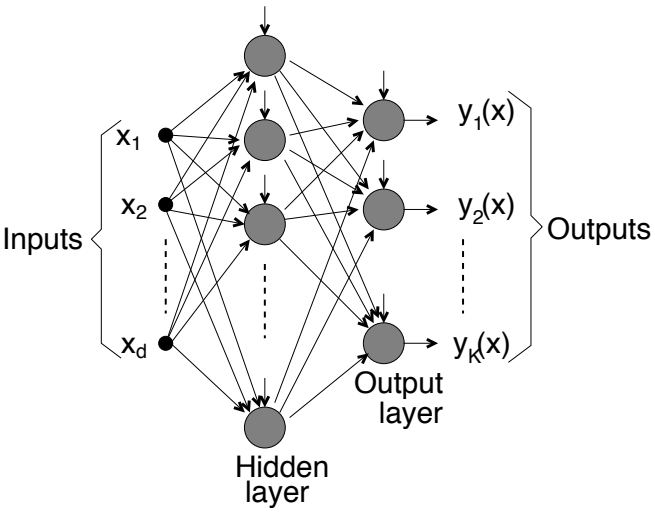
Each node, k forms the a weighted sum, a_k from outputs of previous layer (with a bias).

The trainable parameters are the network weights/biases.

The result is passed through a (typically) **non-linear activation function**.

- sigmoid
- ReLU

For classification, typically a 1-of-K ("1-hot") encoding used at output, and posterior probability of each class estimated.



For regression a linear output activation function can be used.

Arbitrary decision boundaries can be learned with a single sigmoid hidden layer with enough hidden nodes (and DNNs can perform **universal function approximation**).

Structures are more efficient/better performing if narrower models but more hidden layers are used (**deep networks**).

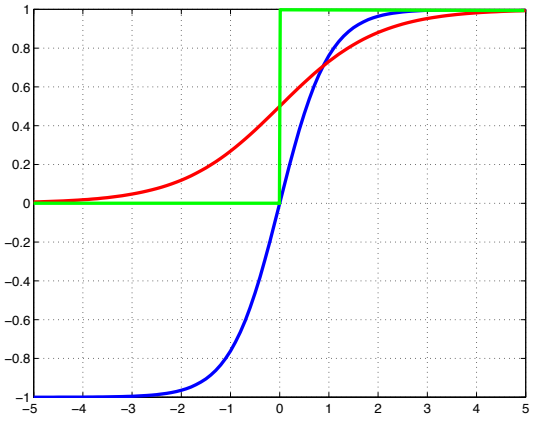
The output is computed using a **forward-pass** through the network.

Activation Functions

A non-linear activation function is normally used (linear output activation fns for regression)

- **Sigmoid** (or logistic regression) function:

$$y_i(\boldsymbol{x}) = \frac{1}{1 + \exp(-a_i)}$$



- **Softmax** (+ve, sums to 1: probabilities!)

$$y_i(\boldsymbol{x}) = \frac{\exp(a_i)}{\sum_{j=1}^{n} \exp(a_j)}$$

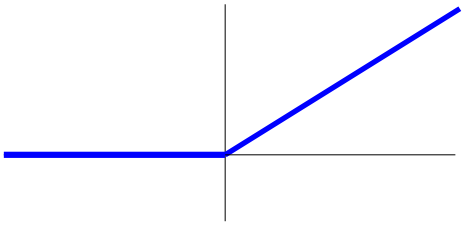
- **Hyperbolic tan** (or tanh) function:

$$y_i(\boldsymbol{x}) = \frac{\exp(a_i) - \exp(-a_i)}{\exp(a_i) + \exp(-a_i)}$$

- **Rectified Linear Unit (ReLU)**

$$y_i = \max(0, a_i)$$

Doesn't saturate (can lead to faster training)



Objective Functions and Output Activations

When classification probabilities are estimated

- output layer typically uses a **softmax** activation function $y_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}}$
- the **cross-entropy** training objective function for targets $t_k^{(n)}$ is used with N patterns

$$\mathcal{E}_{\text{CE}} = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_k^{(n)} \log y_k^{(n)}$$

- For 1-hot targets, equivalent to maximising the log likelihood of outputs given input

Will require derivative of objective function w.r.t. to a_k for a particular input/output pair:

$$\frac{\partial \mathcal{E}_{\text{CE}}}{\partial a_k^{(n)}} = \sum_{k'} \frac{\partial \mathcal{E}_{\text{CE}}}{\partial y_{k'}^{(n)}} \frac{\partial y_{k'}^{(n)}}{\partial a_k^{(n)}} = -\sum_{k'} \frac{t_{k'}^{(n)}}{y_{k'}^{(n)}} \left[\delta(k,k') y_{k'}^{(n)} - y_k^{(n)} y_{k'}^{(n)}\right] = y_k^{(n)} - t_k^{(n)}$$

where $\delta(k,k') = 1$ if $k = k'$ and zero otherwise, and using $\sum_{k'} t_{k'}^{(n)} = 1$.

Objective function and activation function are **matched** to give this simple form: the **error**.

Use of a sum-squared error & linear activation function (for regression) gives same error form.

MLP Training

The objective function is normally optimised by simply using **gradient descent**.

The **partial derivatives ("gradients")** of the training objective function with respect to all of the network weights is computed for each training input/output pair by applying a **backward pass**.

This is known as **Error Back Propagation** (which is really just the chain rule) starting from the **errors** computed at the output e.g. $\frac{\partial \mathcal{E}_{\text{CE}}}{\partial a_k} = \sum_{n=1}^{N} y_k^{(n)} - t_k^{(n)}$

Use this to then back-propagate derivatives through the network (after passing each pattern forward through network) and find for all weights ψ, $\frac{\partial \mathcal{E}_{\text{CE}}}{\partial \psi}$

Note that activation function must be continuous & differentiable (ReLU defined at each point).

Gradient descent in batch mode can be used to update the MLP parameters.

This changes each MLP weight ψ such to decrease the objective function moving in the direction of steepest descent i.e.

$$\psi[\tau+1] = \psi[\tau] - \eta[\tau+1] \frac{\partial \mathcal{E}_{\text{CE}}}{\partial \psi}\bigg|_{\boldsymbol{\Psi}[\tau]}$$

where $\boldsymbol{\Psi}$ is the parameter set of the MLP, $\psi \in \boldsymbol{\Psi}$, η is the learning rate, τ is the batch number.

Could update network weights after all training patterns seen: however there are often too many (stable but slow).
Could update weights after every training sample with new estimate of derivative, but very noisy (slow with GPUs).

In practice update weights after a **mini-batch** of inputs (e.g, hundreds or thousands of examples) seen. Known as mini-batch **Stochastic Gradient Descent (SGD)**.

Mini-batch SGD still has a **noisy estimate** of the gradient but has advantages:

1. More frequent updates than full batch, and scales to large data-sets (perhaps many millions of input/output training pairs).
2. Calculation of a whole mini batch can be computed in parallel (matrix-matrix multiplies rather a set of matrix-vector). This is faster using CPU libraries and on General Purpose Graphics Processor Units (**GPUs**).

Often basic SGD is altered to smooth successive weight updates by adding a **momentum term** which adds on a proportion of the previous update.

The value of the learning rate is often adaptive and typically reduces as training proceeds. It is often updated once after all the data has been seen (once per **epoch**) and performance checked on a held-out set of data (cross-validation or CV set).

Regularisation term is often added to objective function. **"Weight decay"**, adds an *L2* regulariser on weights. If all MLP weights denoted by $\mathbf{v}$ then modify objective function to be

$$\hat{\mathcal{E}}_{\text{CE}}(\mathbf{v}) = \mathcal{E}_{\text{CE}}(\mathbf{v}) + \frac{\nu}{2}\mathbf{v}^{\mathbf{T}}\mathbf{v}.$$

Some Practical Issues

1. **Input Data Normalisation** The input data to MLPs is **normalised** so that it has zero mean and unity variance in each dimension.
   - subtract the global mean of the data for each element of the input vector
   - divide the result by the global standard deviation of that element
2. **Learning Rate Schedule** A suitable learning rate schedule needs to be set.
   - too large a learning rate and convergence is poor or learning is unstable
   - too small a value and the convergence is very slow

   A wide range of different learning rate schedules are used. The cross-validation performance is often measured at the end of each epoch. If the CV performance has degraded, then the learning rate should be decreased and the epoch re-run.
3. **Early Stopping** Typically learning isn't continued to convergence of the training data objective function and the performance (e.g. classification accuracy) on cross-validation data is used. This can be viewed as a simple form of regularisation.
4. **Random Weight Range** Initial range of weights depends typically on the inverse square root of the number of nodes in the previous layer. They should be small enough not to initially saturate activation functions, but large enough so that training isn't too slow!

Deep Neural Network

Use of MLPs originally concentrated on using a single large hidden layer.

However it can be shown that it is often more efficient to use multiple hidden layers .

- A Deep Neural network (DNN) is simply a network with many hidden layers

Major problem with straight-forward SGD with sigmoid activation functions:

- random initialisation of many layers often causes models that don't converge
- gradients can decay to small values if back-propagate through many layers (vanishing gradients)

Initialisation or "**pre-training**" schemes have been proposed:

- Use a generative model of stacked Restrictive Boltzmann Machines (RBMs)
- Discriminative pre-training

Discriminative pre-training simply starts with a trained (single epoch?) single hidden layer network and then

1. removes output layer and adds another randomly initialised hidden layer and output layer
2. trains the network (or just the most recently added layers) for one epoch using EBP

This process is repeated until enough hidden layers have been added.

Once pre-training complete the entire network is trained using EBP.

These final epochs of EBP training is often called "**fine-tuning**".

MLPs for Speech Recognition Acoustic Models

For isolated word recognition could feed in complete words for classification

- This can work well, but need to convert each word to a fixed length pattern!

More interestingly, for LVCSR, try and classify frames of input as sub-word units

- Input consists of standard acoustic features (possibly with derivatives)
- **Acoustic Context Window** used when classifying the current frame
  - typically 3-5 frames before and after the current frame
- Mel filter bank log energies can be used directly: no need to de-correlated inputs.
- very flexible in terms of input features

Train to predict probability of sub-word unit e.g. phone, phone state or context dependent phone state

- Frame-based training, using cross-entropy objective (and softmax output layer)
- Uses a **frame-level alignment** from a pre-existing system ...

During SGD training, need to **randomise frame/utterance presentation in mini-batches**

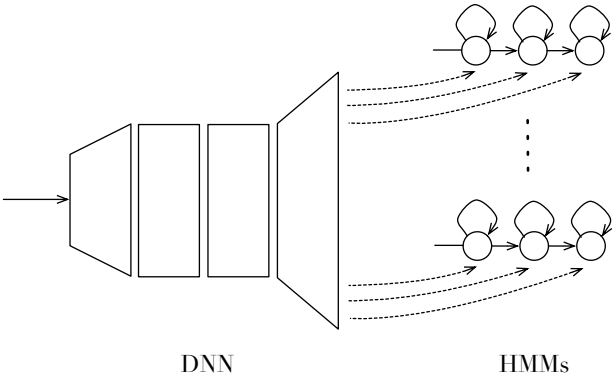
- ensure that each mini-batch contains frames from multiple utterances / speakers etc.
- can use a frame-level **cache** that preserves suitable context window
- randomisation important in order to get good performance

DNN-HMM hybrid

Output of network is posterior $P(y_j|\boldsymbol{x})$, but using Bayes' law $p(\boldsymbol{x}|y_j) \propto \frac{P(y_j|\boldsymbol{x})}{P(y_j)}$

Need to divide posterior by the **training data prior** in order to find a value **proportional to the class-conditional probability density**

- calculated from the number of frames aligned to a particular output target
- normally stored in log domain



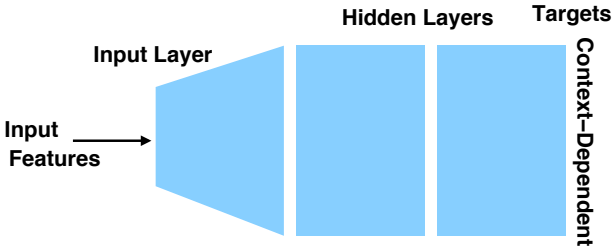
Hence in a **ANN-HMM hybrid**, in recognition, the **posterior probabilities**, after converting to a **pseudo likelihood** are used **in place of the GMM-based likelihoods**.

Note: the standard HMM transition structure is retained, and the Viterbi algorithm used for recognition (still need language model scale factor!)

Standard feed-forward DNN-HMM Acoustic Models

Standard feed-forward DNN uses a number of fully-connected layers.

Softmax activation function is used at output.



Key aspects of feed-forward DNN systems with fully-connected layers:

- Use of many hidden layers (typically 5 layers of 500-2000 hidden nodes per layer).
- Appropriate initialisation of network ("pre-training") via RBMs or discriminative pre-training for sigmoid activation functions
- Random initialisation used for ReLU-based systems
- Train directly to predict the context-dependent phone states
- Estimate the CD states using decision-tree state tying as usual (with GMM models)
- Alignment is from a well-trained GMM-HMM system
- Frame based cross-entropy training "fine-tuning", based on mini-batch gradient descent
- Possible re-alignment using DNN system
- Fast computation using GPUs for large scale application
- Use mel filter bank log energies directly
- Integrate into a DNN-HMM hybrid

Note: state-of-the art augments frame-based training with **discriminative sequence training**

Typical Training Procedure (HTK)

Train a context dependent GMM-HMM system

- context-independent phone models (3 emitting state)
- unclustered context dependent phone models
- decision tree state clustering
- GMM up-mixing
- optionally discriminative training of GMM-HMM system
- Vitberbi force-alignment at the (clustered) state-frame level

DNN Initialisation for sigmoids (ReLUs normally used random initialisation)

- layer-by-layer discriminative pre-training from random initialisation(weight range depends on number of weights)
- during pre-training use context-independent targets (from GMM-HMM alignment) & at end swap to new output layer with context-dependent targets

DNN Fine tuning

- Fine tuning for multiple epochs (e.g. 12)
- Often use L2 regularisation and gradient/update clipping (max absolute value of gradients)
- Optionally re-align training data at state-frame level and re-train (fine tuning only or from scratch).

From training statistics, compute the training data prior for each state in training

DNN-HMM Performance

DNN-HMM hybrid proposed for TIMIT phone recognition (by Geoff Hinton's group)

- Used context independent phone targets, RBM initialisation, and a deep structure.
- Often TIMIT a poor predictor of word-level speech recognition on larger corpora ...

Main ASR community took notice after Microsoft published results on the Switchboard corpus in September 2010

- context dependent phone targets, GPU-based training.
- 30% WER reduction over non-adapted discriminatively trained GMM-HMM system (but poorer than best systems published)

Since then all major groups worldwide (research and commercial) have moved to ANN based systems (used direct ANN-HMM hybrids, DNN-generated features for GMM systems, more recently end-to-end trainable systems).

WERs for MGB Challenge (`http://www.mgb-challenge.org/`) (MGB-1, 2015) to transcribe etc, general BBC programme output

- DNNs use 5 layers of 1000 nodes and 6000 output targets.
- 700h training set from distributed data, manual segmentation, 64k vocab, 4-gram LM.

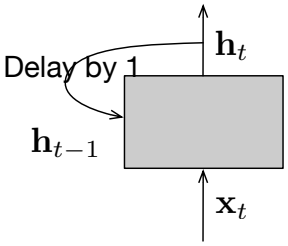
AM	%WER
GMM-HMM ML HLDA	42.7
GMM-HMM MPE	40.7
DNN-HMM Sigmoid CE	28.4

We will return to this example for both using DNNs for feature generation and using discriminative sequence training in place of cross-entropy.

Recurrent Networks

Hidden layer(s) can have **recurrent** connections to model dynamics of speech waveforms

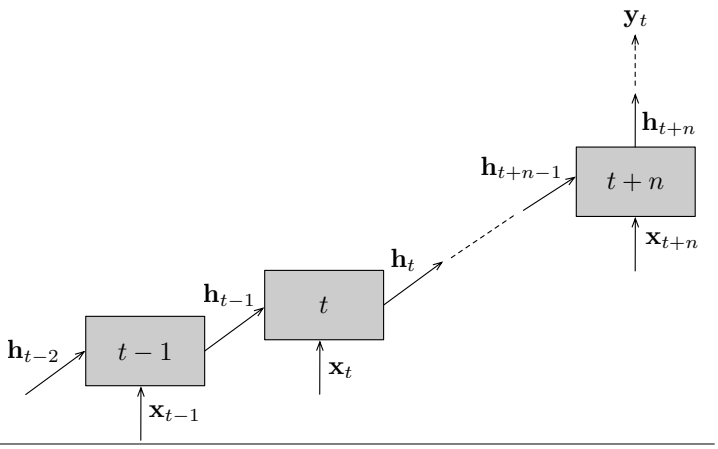
- feed in one input frame at a time
- hidden layer input is current input frame vector and hidden layer output from the previous time step
- learn the features required to classify "current" frame



Have (again) become a focus of speech research (note: for many years from early 1990s, best performing system for TIMIT phone recognition used an RNN developed by Tony Robinson at CUED).

To train, **Unfold** network into a deep feed-forward structure, with recurrent network weights shared across time instances.

- Unfolded RNN structure shown
- Can use some future frames while predicting current output (typically 15 past, 5 future)
- Possible to additionally have a recurrent hidden layer operating backwards in time: **bidirectional RNNs**



Training Recurrent Networks

Training RNNs uses the unfolded view of an RNN

- Deep feed-forward model with shared weights and inputs at each layer (i.e. time)
- Training by SGD involves back-propagating errors through the unfolded network
- This is known as **Back Propagation Through Time (BPPT)**.
- Need to accumulate (& normalise) gradients over the shared weights in unfolded network

It is much more challenging to train RNNs than feed-forward DNNs, since remembering information from many frames ago is difficult (vanishing gradients)

- Use of ReLUs in RNNs means that gradients do not necessarily decay in magnitude during BPTT
- Another way to tackle vanishing gradient problem is to use Long Short Term Memory (LSTM) models which include gated memory cells (covered in a later lecture)

In order to still allow frame randomisation during training (important for speech!)

- constrain the unfolding period: **Truncated BPTT**
- Typically unfold for 20 frames (e.g. -15 . . . +5).
- Used during TIMIT practical

ANN-HMM Part of TIMIT Practical

Use HTK to investigate hybrid ANN-HMMs for TIMIT phone recognition.

- Use HTK 3.5.1 pre-release (HTK 3.5 HTKBook) + PyHTK
- Defining and training ANN models with python script (calls PyHTK)
- Default models have ReLU activation functions: random initialisation
- Initially training feed-forward DNNs
- Use GMMs to define target classes (decision-tree clustered or phones)
- Use GMMs to define initial alignments
- Investigate DNN structures (input features, input context window, number of layers, context independent and context dependent output targets).
- Train using accelerated Intel MKL or via Nvidia K40 GPUs (donated by Nvidia)
- Recurrent Models
  - Time steps for unfolding
  - Comparison of context-independent and context-dependent targets
- Extensions
  - Bigram language model (for both GMMs and ANNs)
  - Extended RNNs: e.g. extra fully connected layers; multiple recurrent layers; use of LSTM models (covered later in course) etc.
  - Use of Time Delay Neural Network (TDNN) feed-forward model (also covered later)

Summary

Deep Neural Networks for ASR acoustic models

- trained to predict context dependent phone states from decision tree state tying
- rely on GMM-HMM systems for alignment and state-clustering
  - ... but possible to do these GMM free!
- mini-batch (randomised frames) stochastic gradient descent
- integrate into ANN-HMM hybrid

Initialisation

- Originally from RBMs or Discriminative pre-training
- Full random initialisation can work for ReLU models without pre-training

Performance of DNN-HMM hybrids:

- Typically 25% lower WER than GMM-HMM speaker independent system (using discriminative training, standard features, no adaptation)

Recurrent neural networks can also be trained

- Uses truncated back-propagation through time

Active research area e.g.

- Alternative network structures
- Improved training algorithms (multi-GPU, joint training, pseudo second order methods etc)
- Adaptation for speaker/environment

Practical allows you to explore some issues with ANNs.

DNN References


H. Bourlard and N. Morgan.
Connectionist Speech Recognition: A Hybrid Approach.
Kluwer Academic Publishers, 1993.


G.E. Hinton, L. Deng, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, and B. Kingsbury.
Deep Neural Networks for Acoustic Modeling in Speech Recognition.
IEEE Signal Processing Magazine, pp. 2–17, November 2012.


T. Robinson and F. Fallside.
A Recurrent Error Propagation Network Speech Recognition System.
Computer Speech & Language, Vol. 5, No. 3, pp. 259–274, 1991.


G. Saon, H. Soltau, A. Emami, and M. Picheny.
Unfolded Recurrent Neural Networks for Speech Recognition.
Proc. Interspeech, 2014.


F. Seide, G Li, X. Chen, and D. Yu.
Feature Engineering in Context-Dependent Deep Neural Networks for Conversational Speech Transcription.
Proc. ASRU, 2011.


C. Zhang and P.C. Woodland.
Standalone Training of Context-Dependent Deep Neural Network Acoustic Models
Proc. ICASSP, 2014