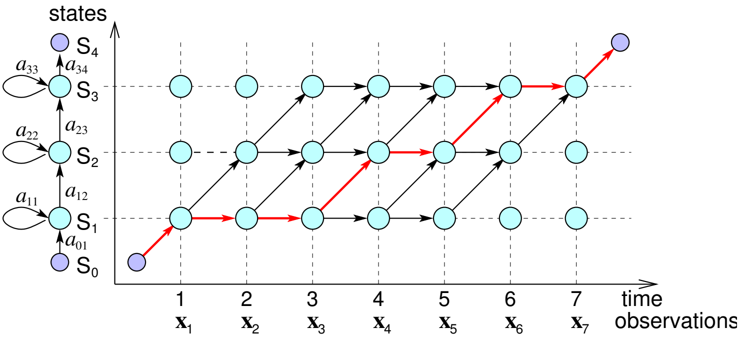
Neural Network Acoustic Models 1: Introduction

Steve Renals

Automatic Speech Recognition – ASR Lecture 7
4 February 2019

Local Phonetic Scores and Sequence Modelling

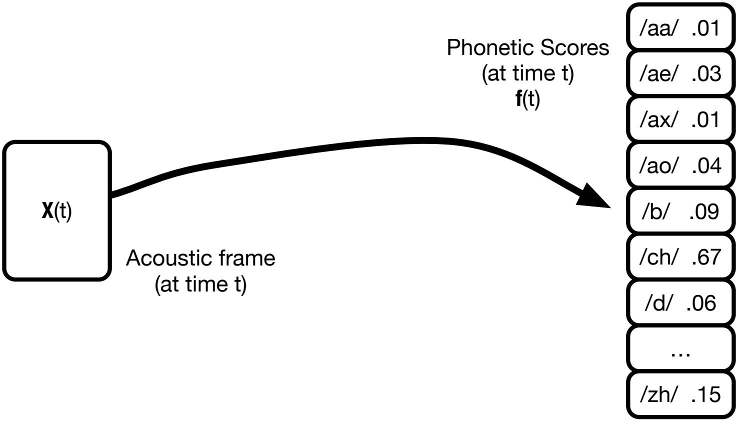
- DTW - local distances (Euclidean)
- HMM - emission probabilities (Gaussian or GMM)



- Compute the phonetic score(acoustic-frame, phone-model) – this does the detailed matching at the frame-level
- Chain phonetic scores together in a sequence - DTW, HMM

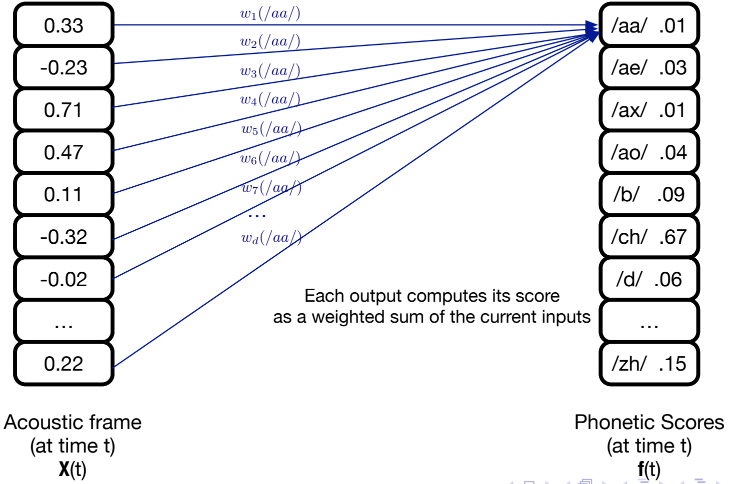
Phonetic scores

Task: given an input acoustic frame, output a score for each phone



Phonetic scores

Compute the phonetic scores using a single layer neural network (linear regression!)



Phonetic scores

Compute the phonetic scores using a single layer neural network

- Write the estimated phonetic scores as a vector
  $\boldsymbol{f} = (f_1, f_2, \ldots, f_Q)$
- Then if the acoustic frame at time t is $\boldsymbol{X} = (x_1, x_2, \ldots, x_d)$:

$$f_j = w_{j1}x_1 + w_{j2}x_2 + \ldots + w_{jd}x_d + b_j = \sum_{i=1}^{d} w_{ji}x_i + b_j$$

$$\boldsymbol{f} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

  where we call $\boldsymbol{W}$ the *weight matrix*, and $\boldsymbol{b}$ the *bias vector*.

- *Check your understanding*:
  What are the dimensions of $\boldsymbol{W}$ and $\boldsymbol{b}$?

Error function

estimated   observed

$$\mathbf{f}(t) = \mathbf{W}\mathbf{x}(t) + \mathbf{b}$$

trained

How do we learn the *parameters* $\boldsymbol{W}$ and $\boldsymbol{b}$?

- Minimise an Error Function: Define a function which is 0 when the output $\boldsymbol{f}(n)$ equals the *target output* $\boldsymbol{r}(n)$ for all n
- Target output: for TIMIT the target output corresponds to the phone label for each frame
- Mean square error: define the error function E as the mean square difference between output and the target:

$$E = \frac{1}{2} \cdot \frac{1}{N} \sum_{n=1}^{N} ||\boldsymbol{f}(n) - \boldsymbol{r}(n)||^2$$

where there are N frames of training data in total

Notes on the error function

- $\boldsymbol{f}$ is a function of the acoustic data $\boldsymbol{x}$ and the weights and biases of the network ($\boldsymbol{W}$ and $\boldsymbol{b}$)
- This means that as well as depending on the training data ($\boldsymbol{x}$ and $\boldsymbol{r}$), E is also a function of the weights and biases, since it is a function of $\boldsymbol{f}$
- We want to minimise the error function given a fixed training set: we must set $\boldsymbol{W}$ and $\boldsymbol{b}$ to minimise E
- Weight space: given the training set we can imagine a space where every possible value of $\boldsymbol{W}$ and $\boldsymbol{b}$ results in a specific value of E. We want to find the minimum of E in this weight space.
- Gradient descent: find the minimum iteratively – given a current point in weight space find the direction of steepest descent, and change $\boldsymbol{W}$ and $\boldsymbol{b}$ to move in that direction

Gradient Descent

- Iterative update – after seeing some training data, adjust the weights and biases to reduce the error. Repeat.
- To update a parameter so as to reduce the error, move downhill in the direction of steepest descent. Thus to train a network compute the gradient of the error with respect to the weights and biases:

$$\frac{\partial E}{\partial \boldsymbol{w}} = \begin{pmatrix} \frac{\partial E}{\partial w_{10}} & \cdot & \frac{\partial E}{\partial w_{1i}} & \cdot & \frac{\partial E}{\partial w_{1d}} \\ & & \cdots & & \\ \frac{\partial E}{\partial w_{j0}} & \cdot & \frac{\partial E}{\partial w_{ji}} & \cdot & \frac{\partial E}{\partial w_{jd}} \\ & & \cdots & & \\ \frac{\partial E}{\partial w_{Q0}} & \cdot & \frac{\partial E}{\partial w_{Qi}} & \cdot & \frac{\partial E}{\partial w_{Qd}} \end{pmatrix}$$

$$\frac{\partial E}{\partial \boldsymbol{b}} = \begin{pmatrix} \frac{\partial E}{\partial b_1} & \cdot & \frac{\partial E}{\partial b_j} & \cdot & \frac{\partial E}{\partial b_Q} \end{pmatrix}$$

Stochastic Gradient Descent Procedure

1. Initialise weights and biases with small random numbers
2. Randomise the order of training data examples
3. For each *epoch* (complete batch of training data)
    - Take a *minibatch* of training examples (eg 128 examples), and for all examples
        - Forward: compute the network outputs $\boldsymbol{y}$
        - Backprop: compute the gradients and accumulate $\partial E/\partial \boldsymbol{w}$ for the minibatch
        - Update the weights and biases using the accumulated gradients and the learning rate hyperparameter η: $\boldsymbol{w} = \boldsymbol{w} - \eta \partial E/\partial \boldsymbol{w}$

Terminate either after a fixed number of epochs, or when the error stops decreasing by more than a threshold.

Gradient in SLN

How do we compute the gradients $\frac{\partial E^n}{\partial w_{ki}}$ and $\frac{\partial E^n}{\partial b_k}$?

$$E^n = \frac{1}{2}\sum_{k=1}^{K}(f_k^n - r_k^n)^2 = \frac{1}{2}\sum_{k=1}^{K}\left(\sum_{i=1}^{d}(w_{ki}x_i^n + b_k) - r_k^n\right)^2$$

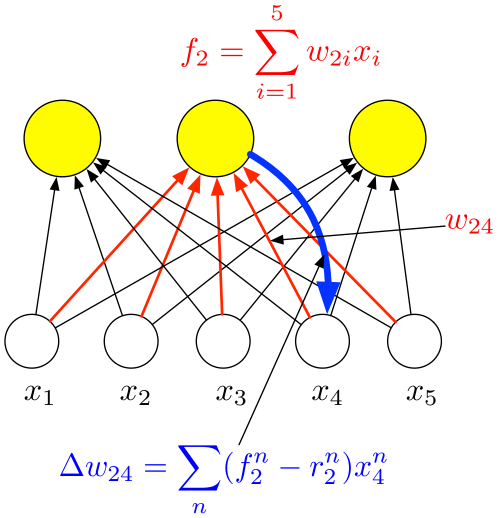
$$\boxed{\frac{\partial E^n}{\partial w_{ki}}} = (f_k^n - r_k^n)x_i^n = \boxed{g_k^n x_i^n} \qquad \boxed{g_k^n = f_k^n - r_k^n}$$

Update rule: Update a weight w_{ki} using the gradient of the error with respect to that weight: the product of the difference between the actual and target outputs for an example $(f_k^n - r_k^n)$ and the value of the unit at the input to the weight (x_i).

Check your understanding: Show that the gradient for the bias is

$$\frac{\partial E^n}{\partial b_k} = g_k^n$$

Applying gradient descent to a single-layer network



Softmax

- Our network that predicts phonetic scores is a *classifier* – at training time each frame of data has a correct label (target output of 1), other labels have a target output of 0
- At test time the the network produces real-valued outputs which we can interpret as the probability of the kth label (q_k) given the input frame $\boldsymbol{x}^t$, $P(q_k|\boldsymbol{x}^t)$
- We can design an output layer which forces the output values to act like probabilities
  - Each output will be between 0 and 1
  - The K outputs will sum to 1
- A way to do this is using the *Softmax* activation function:

$$\boxed{y_k = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}} \qquad a_k = \sum_{j=1}^{d} w_{kj} h_j + b_k$$

Cross-entropy error function

- Since we are interpreting the network outputs as probabilities, we can write an error function for the network which aims to maximise the log probability of the correct label.
- If r_k^t is the $1/0$ target of the the kth label for the tth frame, and t_k^t is the network output, then the cross-entropy error function is:

$$E^n = -\sum_{k=1}^{C} r_k^t \ln y_k^t$$

- Note that if the targets are $1/0$ then the only the term corresponding to the correct label is non-zero in this summation.

Cross entropy and softmax

- A neat thing about softmax: if we train with cross-entropy error function, we get a simple form for the gradients of the output weights:

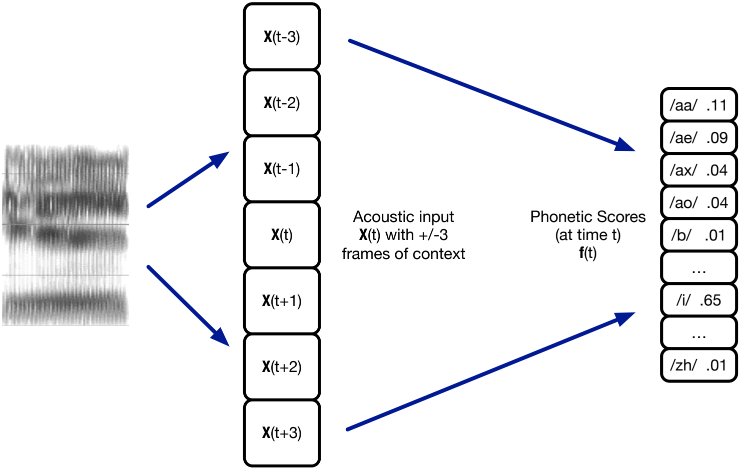
$$\boxed{\frac{\partial E^t}{\partial w_{kj}}} = \underbrace{(y_k^t - r_k^t)}\, x_j$$

- In statistics this is called *logistic regression*

- *Check your understanding:*
  - Why does the cross-entropy error function correspond to maximising the log probability of the cirrect label?
  - Why does the softmax output function ensure the set of outputs for a frame sums to 1?
  - Why are the target labels either 1 or 0? Why does only one target label per frame take the value 1?
  - Why are the network outputs real numbers and not binary (1/0)?

Extending the model: Acoustic context

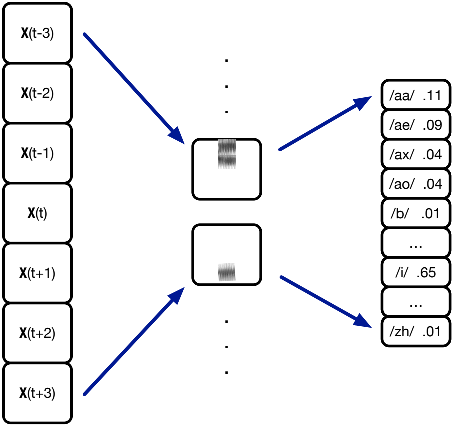
Use multiple frames of acoustic context



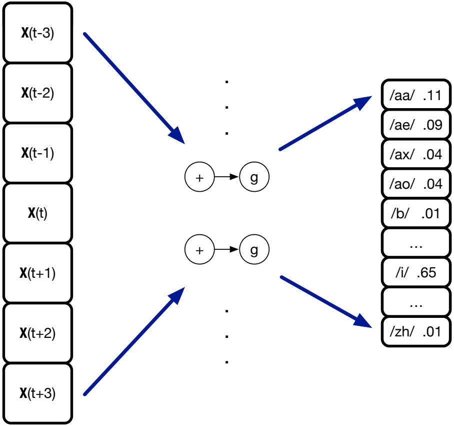
Extending the model: Hidden layers

- Single layer networks have limited computational power – each output unit is trained to match a spectrogram directly (a kind of discriminative template matching)
- But there is a lot of variation in speech (as previously discussed) – rate, coarticulation, speaker characteristics, acoustic environment
- Introduce an intermediate feature representation – layers of "hidden units" – more robust than template matching
- Can have multiple hidden layers to learn successively more abstract representations – *deep neural networks* (DNNs)

Hidden units extracting features

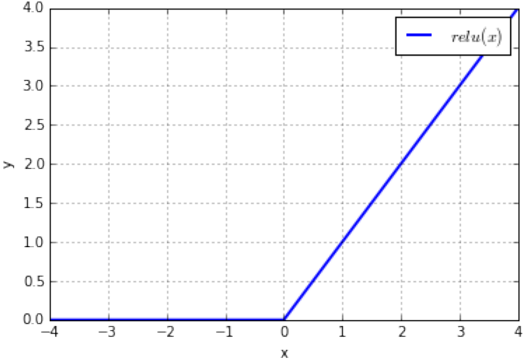


Hidden Units



$$h_j = \text{relu}\left(\sum_{i=1}^{d} v_{ji}x_i + b_j\right) \qquad f_k = \text{softmax}\left(\sum_{j=1}^{H} w_{kj}h_j + b_k\right)$$

Rectified Linear Unit – ReLU



$$\mathrm{relu}(x) = \max(0, x)$$

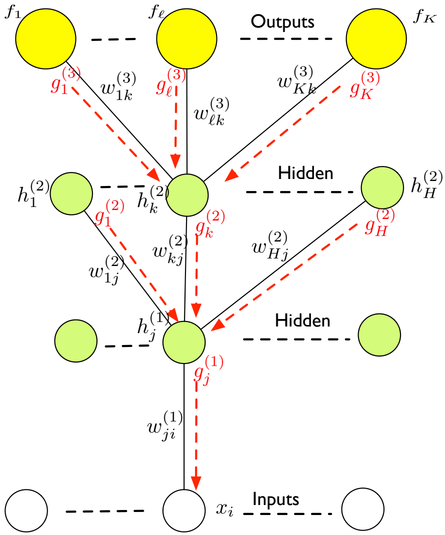
Derivative: $$\mathrm{relu}'(x) = \frac{d}{dx}\mathrm{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Training deep networks: Backprop and gradient descent

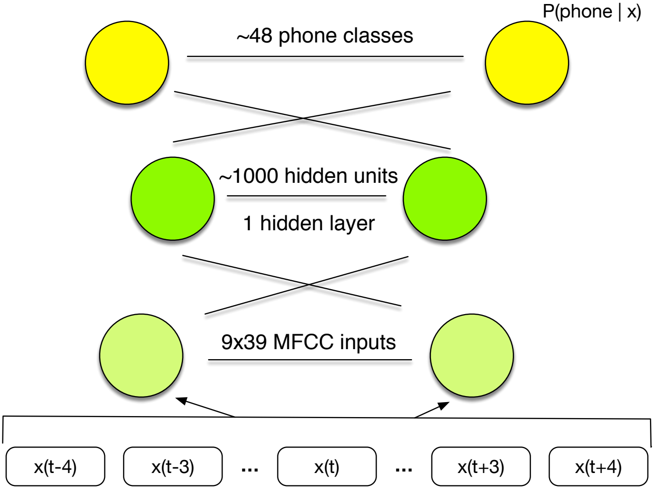
- Hidden units make training the weights more complicated, since each hidden units affects the error function indirectly via all the output units
- The credit assignment problem: what is the "error" of a hidden unit? how important is input-hidden weight w_{ji} to output unit k?
- Solution: *back-propagate* the gradients through the network – the gradient for a hidden unit output with respect to the error[1] can be computed as the weighted sum of the deltas of the connected output units. (Propagate the g values backwards through the network)
- The *back-propagation of error* (*backprop*) algorithm thus provides way to propagate the error graidents through a deep network to allow gradient descent training to be performed

[1] And this gradient can be easily used to compute the gradients of the error with respect to the weights into that hiudden unit

Training DNNs using backprop



Simple neural network for phone classification



Neural networks for phone classification

- Phone recognition task – e.g. TIMIT corpus
  - 630 speakers (462 train, 168 test) each reading 10 sentences (usually use 8 sentences per speaker, since 2 sentences are the same for all speakers)
  - Speech is labelled by hand at the phone level (time-aligned)
  - 61-phone set, usually reduced to 48/39 phones
- Phone recognition tasks
  - Frame classification – classify each frame of data
  - Phone classification – classify each segment of data (segmentation into unlabelled phones is given)
  - Phone recognition – segment the data and label each segment (the usual speech recognition task)
- Frame classification – straightforward with a neural network
  - train using labelled frames
  - test a frame at a time, assigning the label to the output with the highest score

Interim conclusions

- Neural networks using cross-entropy (CE) and softmax outputs give us a way of assigning the probability of each possible phonetic label for a given frame of data
- Hidden layers provide a way for the system to learn representations of the input data
- All the weights and biases of a network may be trained by gradient descent – back-propagation of error provides a way to compute the gradients in a deep network
- Acoustic context can be simply incorporated into such a network by providing multiples frame of acoustic input
- Introductory reading for neural networks:
  - Nielsen, *Neural Networks and Deep Learning*, (chapters 1, 2, 3) `http://neuralnetworksanddeeplearning.com`
  - Jurafsky and Martin (draft 3rd edition), chapter 7 (secs 7.1 – 7.4) `https://web.stanford.edu/~jurafsky/slp3/7.pdf`

Next Lecture

- From frames to sequences to word level transcription – hybrid HMM/DNN
- Modelling context dependence with neural network acoustic models
- Hybrid HMM/DNN systems in practice