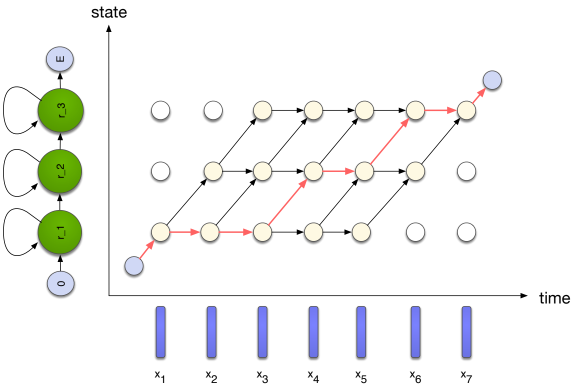
Neural Network Acoustic Models 1: Introduction

Peter Bell

Automatic Speech Recognition – ASR Lecture 10
16 February 2023

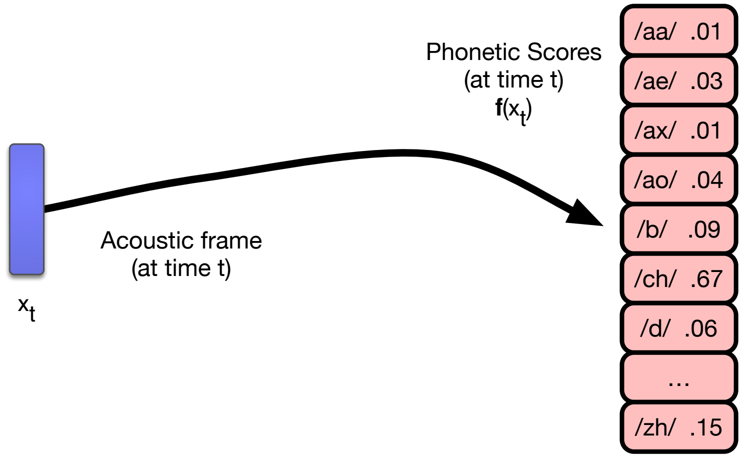
Local phonetic scores and sequence modelling



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

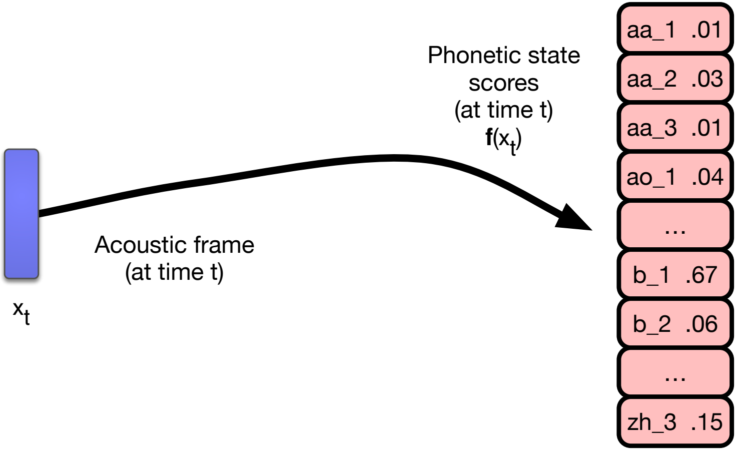
Phonetic scores

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



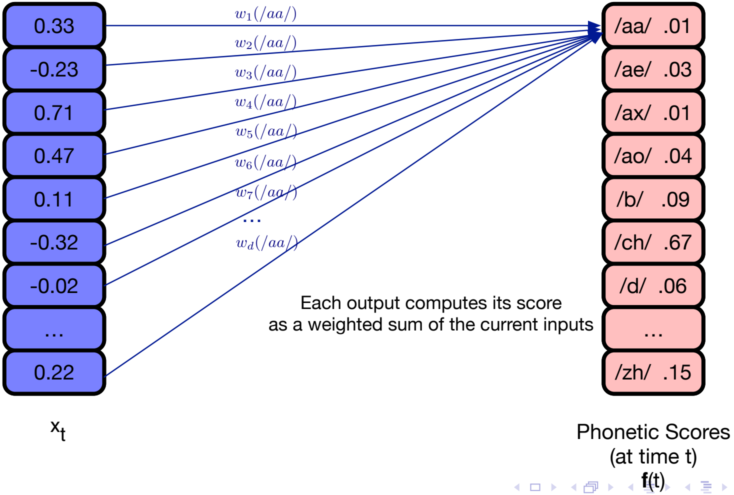
Phone state scores

Output a score for each phone state



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 $\boldsymbol{f} = (f_1, f_2, \ldots, f_J)$
- Then if the acoustic frame at time t is $\boldsymbol{x}_t = (x_1, x_2, \ldots, x_D)$:

$$f_j = w_{j1}x_1 + w_{j2}x_2 + \ldots + w_{jD}x_D + b_j = \sum_{d=1}^{D} w_{jd}x_d + b_j$$

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

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

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

- Minimise an Error Function: Define a function which is 0 when the output $\boldsymbol{f}(\boldsymbol{x}_t)$ equals the *target output* $\boldsymbol{r}(t)$ for all t
- Target output: for phone classification 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:

$$E = \frac{1}{2} \cdot \frac{1}{T} \sum_{t=1}^{T} ||\boldsymbol{f}(\boldsymbol{x}_t) - \boldsymbol{r}(t)||^2$$

where there are T 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:

$$\frac{\partial E}{\partial \boldsymbol{w}} = \begin{pmatrix} \frac{\partial E}{\partial w_{10}} & \cdot & \frac{\partial E}{\partial w_{1d}} & \cdot & \frac{\partial E}{\partial w_{1D}} \\ & & \cdots & & \\ \frac{\partial E}{\partial w_{j0}} & \cdot & \frac{\partial E}{\partial w_{jd}} & \cdot & \frac{\partial E}{\partial w_{jD}} \\ & & \cdots & & \\ \frac{\partial E}{\partial w_{J0}} & \cdot & \frac{\partial E}{\partial w_{Jd}} & \cdot & \frac{\partial E}{\partial w_{JD}} \end{pmatrix}$$

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

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 $\boldsymbol{f}$
        - 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 $\frac{\partial E^t}{\partial w_{jd}}$ and $\frac{\partial E^t}{\partial b_j}$?

$$E^t = \frac{1}{2}\sum_{j=1}^{K}(f_j^t - r_j^t)^2 = \frac{1}{2}\sum_{j=1}^{J}\left(\sum_{d=1}^{D}(w_{jd}x_d^t + b_j) - r_j^t\right)^2$$

$$\boxed{\frac{\partial E^t}{\partial w_{ji}}} = (f_j^t - r_j^t)x_i^t = \boxed{g_j^t x_i^t} \qquad \boxed{g_j^t = f_j^t - r_j^t}$$

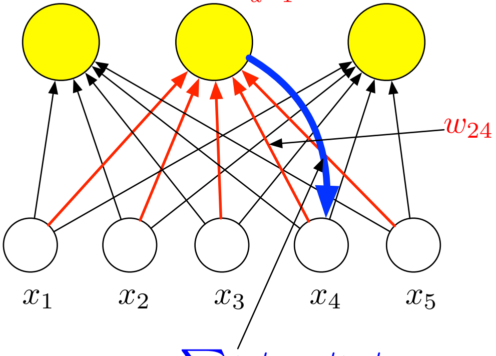
Update rule: Update a weight w_{jd} using the gradient of the error with respect to that weight: the product of the difference between the actual and target outputs for an example $(f_j^t - r_j^t)$ and the value of the unit at the input to the weight (x_d).

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

$$\frac{\partial E^t}{\partial b_j} = g_j^t$$

Applying gradient descent to a single-layer network

$$f_2 = \sum_{d=1}^{5} w_{2d} x_d$$



$$\Delta w_{24} = \sum_t (f_2^t - r_2^t) x_4^t$$

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 jth label given the input frame $\boldsymbol{x}_t$, $P(q_t = j | \boldsymbol{x}_t)$
- 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:

$$\boxed{y_j = \frac{\exp(a_j)}{\sum_{k=1}^{K} \exp(a_k)}} \qquad a_j = \sum_{d=1}^{D} w_{jd} x_d + b_j$$

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_j^t is the 1/0 target of the the jth label for the tth frame, and y_j^t is the network output, then the cross-entropy (CE) error function is:

$$E^t = -\sum_{j=1}^{J} r_j^t \ln y_j^t$$

- 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

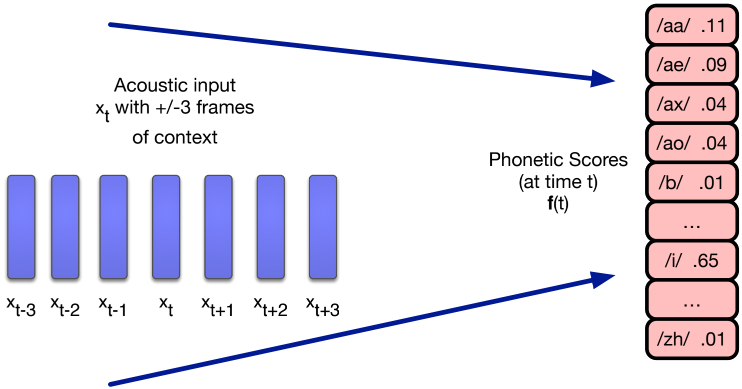
- A neat thing about softmax: if we train with cross-entropy error function, we get a simple form for the gradients of the output weights:

$$\boxed{\frac{\partial E^t}{\partial w_{jd}}} = \underbrace{(y_j^t - r_j^t)}\, x_d^t$$

- 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 correct 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

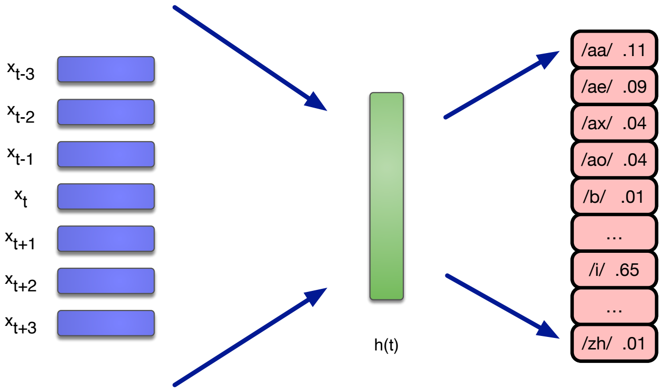
Use multiple frames of acoustic context



Extending the model: Hidden layers

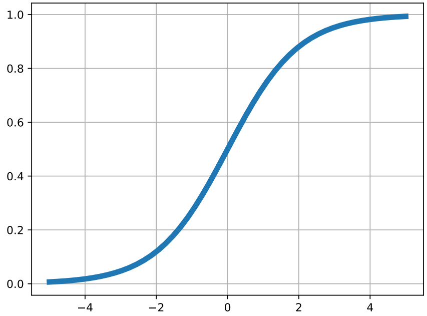
- Single layer networks have limited computational power – each output unit is trained to match a spectrogram directly (a kind of discriminative template matching)
- But there is a lot of variation in speech (as previously discussed) – rate, coarticulation, speaker characteristics, acoustic environment
- 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



$$h_k = \sigma\left(\sum_{d=1}^{D} v_{kd} x_d + b_k\right) \qquad y_j = \mathrm{softmax}\left(\sum_{k=1}^{K} w_{jk} h_k + b_j\right)$$

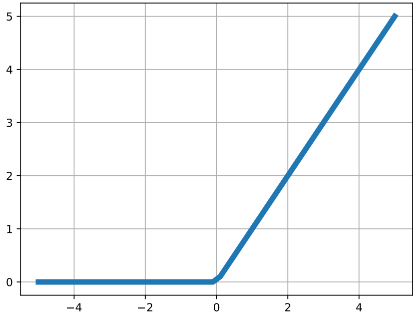
Sigmoid function



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Derivative: $$\sigma'(x) = \frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x))$$

Rectified Linear Unit – ReLU



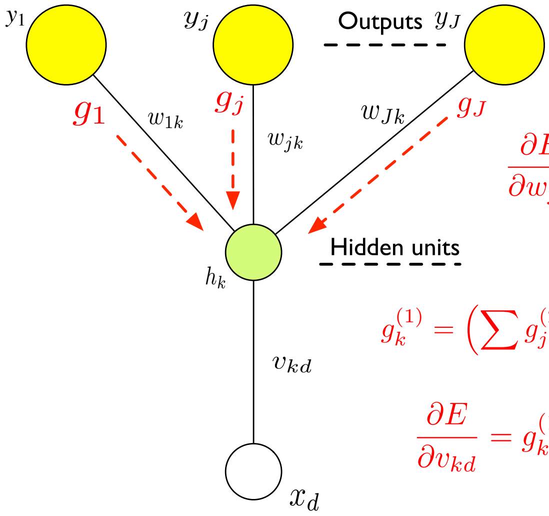
$$\text{relu}(x) = \max(0, x)$$

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

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 v_{kd} to output unit j?
- Solution: *back-propagate* the gradients through the network – the gradient for a hidden unit output with respect to the error 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

Training DNNs using backprop

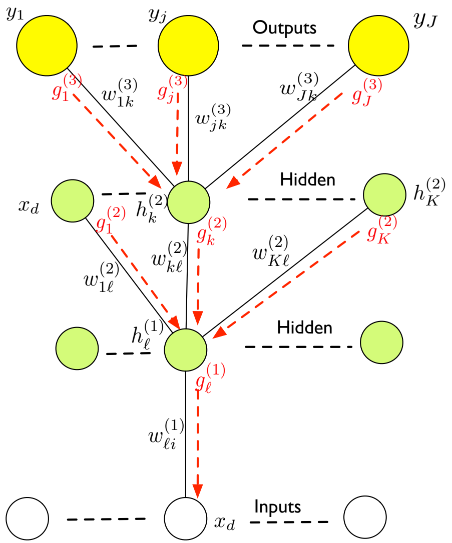


$$\frac{\partial E}{\partial w_{jk}} = g_j^{(2)} h_k$$

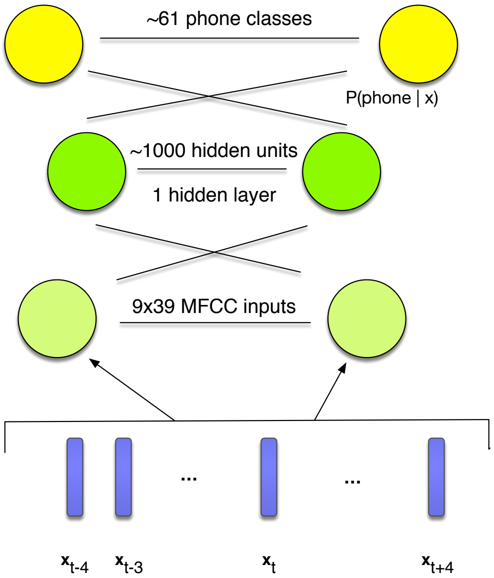
$$g_k^{(1)} = \left(\sum g_j^{(2)} w_{jk} \right) relu'_k$$

$$\frac{\partial E}{\partial v_{kd}} = g_k^{(1)} x_d$$

Multiple hidden layers



Simple neural network for phone classification



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