Introduction to Neural Networks

Steve Renals

Automatic Speech Recognition – ASR Lecture 7
5 February 2018
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

\[ X(t) \]

Acoustic frame (at time t)

Phonetic Scores (at time t)

\( f(t) \)

\[ /aa/ .01 \]
\[ /ae/ .03 \]
\[ /ax/ .01 \]
\[ /ao/ .04 \]
\[ /b/ .09 \]
\[ /ch/ .67 \]
\[ /d/ .06 \]
\[ /zh/ .15 \]
Compute the phonetic scores using a single layer neural network (linear regression!)

Acoustic frame (at time t)

\[ X(t) \]

Phonetic Scores (at time t)

\[ f(t) \]

Each output computes its score as a weighted sum of the current inputs.

\[ w_1(\text{/aa/}) \]
\[ w_2(\text{/aa/}) \]
\[ w_3(\text{/aa/}) \]
\[ w_4(\text{/aa/}) \]
\[ w_5(\text{/aa/}) \]
\[ w_6(\text{/aa/}) \]
\[ w_7(\text{/aa/}) \]
\[ \ldots \]
\[ w_d(\text{/aa/}) \]

\[ /aa/ .01 \]
\[ /ae/ .03 \]
\[ /ax/ .01 \]
\[ /ao/ .04 \]
\[ /b/ .09 \]
\[ /ch/ .67 \]
\[ /d/ .06 \]
\[ /zh/ .15 \]
Phonetic scores

Compute the phonetic scores using a single layer neural network

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

  \[
  f_j = w_{j1}x_1 + w_{j2}x_2 + \ldots + w_{jd}x_d + b_j
  \]

  or, write it using summation notation:

  \[
  f_j = \sum_{i=1}^{d} w_{ji}x_i + b_j
  \]

  or, write it as vectors:

  \[
  \mathbf{f} = \mathbf{Wx} + \mathbf{b}
  \]

  where we call \( \mathbf{W} \) the weight matrix, and \( \mathbf{b} \) the bias vector.

- Check your understanding:
  What are the dimensions of \( \mathbf{W} \) and \( \mathbf{b} \)?
Error function

\[ f(t) = Wx(t) + b \]

How do we learn the parameters \( W \) and \( b \)?

- **Minimise an Error Function**: Define a function which is 0 when the output \( f(n) \) equals the target output \( r(n) \) 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:

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

where there are \( N \) frames of training data in total
Notes on the error function

- \( f \) is a function of the acoustic data \( x \) and the weights and biases of the network (\( W \) and \( b \)).

- This means that as well as depending on the training data (\( x \) and \( r \)), \( E \) is also a function of the weights and biases, since it is a function of \( f \).

- We want to minimise the error function given a fixed training set: we must set \( W \) and \( b \) to minimise \( E \).

- **Weight space**: given the training set we can imagine a space where every possible value of \( W \) and \( b \) 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 \( W \) and \( b \) to move in that direction.
Gradient Descent

- Iterative update – after seeing some training data, we adjust the weights and biases to reduce the error. Repeat until convergence.

- To update a parameter so as to reduce the error, we move downhill in the direction of steepest descent. Thus to train a network we must compute the gradient of the error with respect to the weights and biases:

\[
\frac{\partial E}{\partial w_{10}} \cdot \frac{\partial E}{\partial w_{1i}} \cdot \frac{\partial E}{\partial w_{1d}} \\
\frac{\partial E}{\partial w_{j0}} \cdot \ldots \cdot \frac{\partial E}{\partial w_{jd}} \\
\frac{\partial E}{\partial w_{Q0}} \cdot \ldots \cdot \frac{\partial E}{\partial w_{Qd}}
\]

\[
\left( \frac{\partial E}{\partial b_1} \cdot \frac{\partial E}{\partial b_j} \cdot \frac{\partial E}{\partial b_Q} \right)
\]
Gradient Descent Procedure

1. Initialise weights and biases with small random numbers
2. For each batch of training data
   1. Initialise total gradients: $\Delta w_{ki} = 0, \Delta b_k = 0$
   2. For each training example $n$ in the batch:
      • Compute the error $E^n$
      • For all $k, i$: Compute the gradients $\partial E^n / \partial w_{ki}, \partial E^n / \partial b_k$
      • Update the total gradients by accumulating the gradients for example $n$
        $$\Delta w_{ki} \leftarrow \Delta w_{ki} + \frac{\partial E^n}{\partial w_{ki}} \quad \forall k, i$$
        $$\Delta b_k \leftarrow \Delta b_k + \frac{\partial E^n}{\partial b_k} \quad \forall k$$
3. Update weights:
   $$w_{ki} \leftarrow w_{ki} - \eta \Delta w_{ki} \quad \forall k, i$$
   $$b_k \leftarrow b_k - \eta \Delta b_k \quad \forall k$$

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 \( \frac{\partial E^n}{\partial b_k} \)?

\[
E^n = \frac{1}{2} \sum_{k=1}^{K} (f^n_k - r^n_k)^2 = \frac{1}{2} \sum_{k=1}^{K} \left( \sum_{i=1}^{d} (w_{ki} x^n_i + b_k) - r^n_k \right)^2
\]

\[
\frac{\partial E^n}{\partial w_{ki}} = (f^n_k - r^n_k) x^n_i = g^n_k x^n_i \quad g^n_k = f^n_k - r^n_k
\]

**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 \((f^n_k - r^n_k)\) 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

\[
\frac{\partial E^n}{\partial b_k} = g^n_k
\]
Applying gradient descent to a single-layer network

\[ f_2 = \sum_{i=1}^{5} w_{2i} x_i \]

\[ \Delta w_{24} = \sum_{n} (f^n_2 - r^n_2) x^n_4 \]
Acoustic context

Use multiple frames of acoustic context

Acoustic input $X(t)$ with $+/-3$ frames of context

Phonetic Scores (at time $t$) $f(t)$

- /aa/ .11
- /ae/ .09
- /ax/ .04
- /ao/ .04
- /b/ .01
- /i/ .65
- /zh/ .01
Hidden units

- 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 – “hidden units” – more robust than template matching.
- Intermediate features represented by hidden units.
Hidden units extracting features

\[ X(t-3) \]
\[ X(t-2) \]
\[ X(t-1) \]
\[ X(t) \]
\[ X(t+1) \]
\[ X(t+2) \]
\[ X(t+3) \]

\[ /aa/ .11 \]
\[ /ae/ .09 \]
\[ /ax/ .04 \]
\[ /ao/ .04 \]
\[ /b/ .01 \]
\[ /i/ .65 \]
\[ /zh/ .01 \]
Hidden Units

\[ h_j = \text{relu} \left( \sum_{i=1}^{d} w_{ji} x_i + b_j \right) \]

\[ f_k = \text{softmax} \left( \sum_{j=1}^{H} v_{kj} h_j + b_k \right) \]
Rectified Linear Unit – ReLU

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

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

\[ y_k = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)} \]

\[ a_k = \sum_{j=1}^{H} v_{kj} h_j + b_k \]

- This form of activation has the following properties
  - Each output will be between 0 and 1
  - The denominator ensures that the \( K \) outputs will sum to 1
- Using softmax we can interpret the network output \( y_k^n \) as an estimate of \( P(k|x^n) \)
Cross-entropy error function

- Cross-entropy error function:

\[ E^n = - \sum_{k=1}^{C} r^n_k \ln y^n_k \]

Optimise the weights \( W \) to maximise the log probability – or to minimise the negative log probability.

- 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:

\[ \frac{\partial E^n}{\partial v_{kj}} = \underbrace{(y_k - r_k)}_{g_k} h_j \]
Training multilayered networks – output layer

\[ f_1 \quad f_e \quad f_K \]

\[ v_{1j} \quad v_{ej} \quad v_{Kj} \]

\[ h_j \]

\[ w_{ji} \]

\[ x_i \]
Training multilayered networks – output layer

\[
\begin{align*}
\text{Outputs} & : f_1, \ldots, f_K \\
\text{Hidden units} & : v_{1j}, \ldots, v_{Kj} \\
\delta_1, \delta_\ell, \delta_K & : \text{error terms} \\
\end{align*}
\]

\[
\begin{align*}
\frac{\partial E}{\partial v_{kj}} &= \delta_k h_j \\
\end{align*}
\]
Backprop

- Hidden units make training the weights more complicated, since the hidden units affect the error function indirectly via all the outputs.
- 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 deltas through the network.
- $g_j$ for a hidden unit is the weighted sum of the deltas of the connected output units. (Propagate the $g$ values backwards through the network.)
- Backprop provides way of estimating the error of each hidden unit.
Backprop

\[
\begin{align*}
&\text{Outputs} \\
&f_1, f_\ell, f_K \\
&\text{Hidden units} \\
&g_j = \left( \sum_\ell g_\ell v_{\ell j} \right) \text{relu}'_j \\
&\frac{\partial E}{\partial v_{kj}} = g_k h_j \\
&\frac{\partial E}{\partial w_{ji}} = g_j x_i
\end{align*}
\]
The back-propagation of error algorithm is summarised as follows:

1. Apply an input vectors from the training set, \( \mathbf{x} \), to the network and forward propagate to obtain the output vector \( \mathbf{f} \).
2. Using the target vector \( \mathbf{r} \) compute the error \( E^n \).
3. Evaluate the error signals \( g_k \) for each output unit.
4. Evaluate the error signals \( g_j \) for each hidden unit using back-propagation of error.
5. Evaluate the derivatives for each training pattern.

Back-propagation can be extended to multiple hidden layers, in each case computing the \( g_s \) for the current layer as a weighted sum of the \( g_s \) of the next layer.
Summary and Reading

- Single-layer and multi-layer neural networks
- Error functions, weight space and gradient descent training
- Multilayer networks and back-propagation
- Transfer functions – sigmoid and softmax
- Acoustic context
- M Nielsen, *Neural Networks and Deep Learning*, http://neuralnetworksanddeeplearning.com (chapters 1, 2, and 3)

**Next lecture:** Neural network acoustic models