For the input weight matrices, we get

$$w_{xh}^{new}(i,j) = w_{xh}(i,j) - \gamma \sum_{t=1}^{T} \frac{\partial E}{\partial u_t(i)} \frac{\partial u_t(i)}{\partial w_{xh}(i,j)} = w_{xh}(i,j) - \gamma \sum_{t=1}^{T} \delta_t^h(i) x_t(j) \quad (3.25)$$

or

$$\mathbf{W}_{xh}^{new} = \mathbf{W}_{xh} + \gamma \sum_{t=1}^{T} \delta_h^t \mathbf{x}_t^\top \quad (3.26)$$

For the recurrent weight matrices we have

$$\begin{aligned} w_{hh}^{new}(i,j) &= w_{hh}(i,j) - \gamma \sum_{t=1}^{T} \frac{\partial E}{\partial u_t(i)} \frac{\partial u_t(i)}{\partial w_{hh}(i,j)} \\ &= w_{hh}(i,j) - \gamma \sum_{t=1}^{T} \delta_h^t(i) h_{t-1}(j) \\ \text{or } &= \mathbf{W}_{hh}^{new} = \mathbf{W}_{hh} + \gamma \sum_{t=1}^{T} \delta_h^t \mathbf{h}_{t-1}^\top \end{aligned} \quad (3.27)$$

In the BPTT algorithm the sub gradients are summed over all time frames. The algorithm is summarised below:

**Result:** Optimal weights

initialise weights randomly;

**for** *error is significant or epochs less than maximum* **do**

  forward computation ;

  determine layer-wise error for weights and biases $\Delta_{\mathbf{W}} E$ and $\Delta_{\mathbf{b}} E$ ;

  update weights and biases according to gradient descent;

**end**

**Algorithm 2:** RNN training algorithm

### 3.2.4 LSTMs and GRUs

A special implementation of the RNN called the Long Short Term Memory (LSTM) has been designed to capture patterns over particularly long sequences of data and thus is an ideal candidate for generating character sequences while preserving syntactic language rules learned from the training data.

The internal structure and working of the LSTM cell is documented by its creators in Sak et al. (2014). The ability to recall information over extended sequences results from the internal gated structure which performs a series of element wise

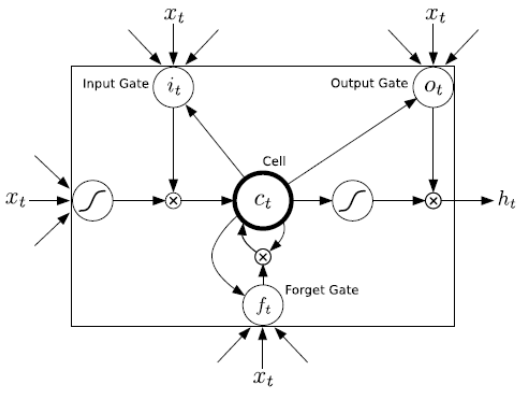


Figure 3.3: An LSTM Cell Graves et al. (2013)

multiplications on the inputs and internal state of the LSTM cell at each time step. In addition to the output neurons which in this text we refer to as the write gate and denote as the current cell state, $\mathbf{c}_t$, three additional gates (comprising a neural network sub-layer) located within the LSTM cell are the input gate, the forget gate and the output gate. Together with the initial current state cell, these gates along with the current-state cell itself enable the LSTM cell architecture to store information, forward information, delete information and receive information. Generally however, the LSTM cell looks like a regular feed-forward network having a set of neurons capped with a nonlinear function. The recurrent nature of the network arises, however due to the fact that the internal state of the RNN cell is rerouted back as an input to the RNN cell or input to the next cell in the time-series giving rise to sequence memory within the LSTM architecture. Mathematically, these gates are formulated as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}^{(xi)}\mathbf{x}_t + \mathbf{W}^{(hi)}\mathbf{h}_{t-1} + \mathbf{W}^{(ci)}\mathbf{c}_{t-1} + \mathbf{b}^{(i)}) \tag{3.28}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^{(xf)}\mathbf{x}_t + \mathbf{W}^{(hf)}\mathbf{h}_{t-1} + \mathbf{W}^{(cf)}\mathbf{c}_{t-1} + \mathbf{b}^{(f)}) \tag{3.29}$$

$$\mathbf{c}_t = \mathbf{f}_t \bullet \mathbf{c}_{t-1} + \mathbf{i}_t \bullet \tanh(\mathbf{W}^{(xc)}\mathbf{x}_t + \mathbf{W}^{(hc)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}) \tag{3.30}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^{(xo)}\mathbf{x}_t + \mathbf{W}^{(ho)}\mathbf{h}_{t-1} + \mathbf{W}^{(co)}\mathbf{c}_{t-1} + \mathbf{b}^{(o)}) \tag{3.31}$$

$$\mathbf{h}_t = \mathbf{o}_t \bullet \tanh(\mathbf{c}_t) \tag{3.32}$$

The gates in the above formula are illustrated in Figure 3.3. $\mathbf{i}_t$ represents the input gate, $\mathbf{f}_t$ is the forget gate and $\mathbf{o}_t$ represents the output gate. At each of these

gates therefore, the inputs consisting of hidden states in addition to the regular inputs are multiplied by a set of weights and passed through a soft-max function. These weights during training learn whether the gate will, during inference, open or not. In summary, the input gate tells the LSTM whether or not to receive new information, the forget gate determines whether the current information it already has from the previous step should be kept or dropped and the output gate determines what should be forwarded to the next LSTM cell. Note also that the LSTM has two sigmoid (*tanh*) activation functions utilised at the input and output of the current cell $\mathbf{c}_t$.

One particular variant of the original LSTM model is the GRU cell. Though simpler than an LSTM cell the GRU cell performs equally efficiently. The GRU cell is a subset implementation of the LSTM cell. Rather than using the output gate of the LSTM, this gate is omitted in the GRU and the output result of the other internal gates are always forwarded. The second simplification is a merge of the internal gate state vectors into a single vector $\mathbf{h}_{(t)}$. This merged gate here referred to as $\mathbf{z}(t)$, controls both the forget gate an the input gate and acts as follows. Whenever a value is retained by the cell the previous value is erased first. That is, if the gate controller outputs a 1, in the LSTM this corresponds to the input gate is open and the forget gate is closed. Therefore if $\mathbf{z}(t)$ it outputs a 0, the reverse happens for the input gate and the forget gate in the LSTM. There is, however, a new gate controller, $\mathbf{r}(t)$, which determines which portion of the previous state will be shown at the output (Cho et al., 2014).

The architecture of a GRU is formulated as follows:

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}^T \cdot \mathbf{x}_{(t-1)}) \tag{3.33}$$

$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}^T \cdot \mathbf{x}_{(t-1)}) \tag{3.34}$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)})) \tag{3.35}$$

$$\mathbf{h}_{(t)} = (1 - \mathbf{z}_{(t)}) \otimes (\mathbf{h}_{(t-1)}) + \mathbf{z}_{(t)} \otimes \mathbf{g}_t \tag{3.36}$$

Due to the light-weight nature of the GRU cell, it is common practice to use GRU cells in place of LSTM cells. This precedence achieves the much desired lighter

computation load on the actual hardware performing the RNN training. As each of the gates required in an LSTM cell are high density matrix multiplication operations in themselves, the condensation of two gates into one and the omission of the output gate within GRU cells pushes towards halving the architectural complexity and coupled with the equally efficient performance of the GRU when compared to the LSTM cell ultimately serves as an overall improvement on the LSTM architecture. For this reasons, GRUs are more attractive to researchers than LSTMs and was naturally the RNN cell of choice used in this report.

## 3.3 Deep speech architecture

This work makes use of an enhanced RNN architecture called the Bi-directional Recurrent Neural Network (BiRNN). While Hannun et al. (2014b) assert that forward recurrent connections does reflect the sequential relationships of an audio waveform, perhaps the BiRNN model poses a more powerful sequence model.

The BiRNN is a preferred end to end mechanism due to the length of sequence over which temporal relationships can be captured. This implies that BiRNNs will be suited for capturing temporal relationships over much longer sequences than a forward only RNN, because hidden state information is preserved in both forwards and backwards direction.

In addition, such a model has a notion of complete sentence or utterance integration, having information over the entire temporal extent of the input features when making each prediction.

The formulation of the BiRNN is derived by starting off with the basic RNN architecture which is referred to as the forward architecture. From the forward architecture we derive the backward architecture. If we choose a temporally recurrent layer $j$, the BiRNN forward and backward intermediate hidden representation $h_t^{(f)}$ and $h_t^{(b)}$ is given as.

$$h_t^{(f)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(f)T} h_{t-1}^{(j)} + b^{(j)})) \tag{3.37}$$

$$h_t^{(b)} = \sigma(\mathbf{W}^{(j)T} h_t^{(i-1)} + \mathbf{W}_k^{(b)T} h_{t+1}^{(b)} + b^{(j)})) \tag{3.38}$$

Temporal weight matrices $W^{(f)}$ and $W^{(b)}$ propagate $h_t^{(f)}$ and $h_t^{(b)}$ forward and backward in time respectively.

Hannun et al. (2014b) points out that the recurrent forward and backward components are evaluated entirely independent of each other and for optimal training, a modified non linearity function $\sigma(z) = min(max(z, 0), 20)$ is recommended.

The final BiRNN representation $h_t^{(j)}$ for the layer is now the superposition of the two RNN components,

$$h_t^{(j)} = h_t^{(f)} + h_t^{(b)} \tag{3.39}$$

Also note that back propagation through time (BPTT) sub gradient evaluations are computed from the combined BiRNN structure directly during training.

### 3.3.1 Connectionist Temporal Classification (CTC)

The term CTC stands for Connectionist Temporal classification. This algorithm was designed to solve the problem of fuzzy alignment between the source input data and the output classification desired from the machine learning system. This type of fuzzy alignment is observed in speech recognition systems as the same speech in either the same individual or different individuals will have different signal forms. This is a many to one relationship between the input signal and the output classification that is dependent on the speaker style of speech when the utterance is spoken. Unlike hybrid DNN-HMM networks the CTC algorithm deploys an end-to-end framework that models all aspects of the input sequence in a single neural network, therefore discarding the need for an HMM interpretation of the input sequence. In addition, the CTC method does not require pre-segmented training data at the same time output classification is made independent of post-processing.

CTC works by making predictions at any point in the input sequence. For the case of speech modelling, CTC makes a character prediction for every time step of the raw audio input speech signal. Although this initially seems counter intuitive, this method models the many to one relationship seen in the fuzzy audio speech to text alignment.

For hybrid DNN-HMM systems, speech or more accurately, acoustic models, require separate training of targets for every time-slice in the input sequence. Secondly,

and as a consequence of this, it becomes necessary to segment the audio sequence, in order to provide targets for every time-slice. A third consequence is the limitation of DNNs previously discussed. As the DNN network only outputs local classifications, global aspects such as the likelihood of two consecutive labels appearing together cannot be directly modelled. Without an external model, usually in the form of a language model, the hybrid speech model will significantly degrade performance.

In the CTC case, so long as the overall sequence of labels is correct the network can be optimised to correct the temporal or fuzzy alignments. Since this many to one fuzzy alignment is simultaneously modelled in CTC, then there is no need for pre-segmented data. At the same time, CTC models probabilities of complete label sequences, hence external post-processing required by hybrid models is eliminated.

Similar to the HMM sequence model, the CTC algorithm is a sequence model that predicts the next label in a sequence as a cumulative of previous sequences. This section develops the CTC loss function borrowing concepts used in HMM models such as the forward backward algorithm as outlined in (Graves et al., 2006). In the following paragraph we introduce terminology associated with the CTC loss function.

Given two symbols $A$ and $\mathcal{B}$ such that $A$ has a many to one relationship with $\mathcal{B}$, signifying the temporal nature of the classification. The symbol $A$ represents an alphabet from which a sequence of the output classifications are drawn from. This CTC output consists of a soft-max layer in a BiRNN (bidirectional recurrent neural network).

This output models the probability distribution of a complete sequence of arbitrary length $|A|$ over all possible labels in $A$ from activations within $|A|$. An extra activation is given to represent the probability of outputting a *blank*, or no label. At each time-step leading up to the final step, the probability distribution estimated as distribution over all possible label sequences of length leading up to that of the input sequence.

It is now possible to define the extended alphabet $A' = A \cup \{blank\}$, also, $y_{t,p}$ as the the activation of network output $p$ at time $t$. Therefore $y_{t,p}$ is the probability that the network will output element $p \in A'$ at time $t$ given that $x$ is the input sequence of length $T$. The distribution sought after $Pr(\pi|x)$, is the conditionally

independent distribution over the subset $A'^T$ where $A'^T$ denotes the set of length $T$ sequences in $A'$.

$$\Pr(\pi \mid x) = \prod_{t=1}^{T} y_{t,\pi_t} \tag{3.40}$$

From the above, it is now possible to define the many-to-one mapping $\mathcal{B} : A'^T \rightarrow A^{\leq T}$, from the set of paths onto the set $A^{\leq T}$ of possible labellings of $x$ (i.e. the set of sequences of length less than or equal to $T$ over $A$). We do this by removing first the repeated labels and then the blanks from the paths. For example,

$$\begin{aligned} \mathcal{B}(a - ab-) &= aab \\ \mathcal{B}(-aa - -abb) &= aab. \end{aligned} \tag{3.41}$$

Intuitively, this corresponds to outputting a new label when the network either switches from predicting no label to predicting a label, or from predicting one label to another. As $\mathcal{B}$ is many-to-one, the probability of some labelling $l \in A^{\leq T}$ can be calculated by summing the probabilities of all the paths mapped onto it by $\mathcal{B}$:

$$\Pr(l \mid x) = \sum_{\pi \in \mathcal{B}^{-1}(l)} \Pr(\pi \mid x) \tag{3.42}$$

This 'collapsing together' of different paths onto the same labelling is what makes it possible for CTC to use unsegmented data, because it allows the network to predict the labels without knowing in advance where they occur. In theory, it also makes CTC networks unsuitable for tasks where the location of the labels must be determined. However in practice CTC tends to output labels close to where they occur in the input sequence.

### 3.3.2 Forward-backward algorithm

The forward-backward algorithm is used to estimate the probability of a point in the sequence as the product of all point leading up to that point from the initial state, the forward variable ($\alpha$), multiplied by the probability of all the points from that state to the end of the sequence, the backward variable ($\beta$).

The difference between this estimation and that determined from equation (3.42) is the fact that the forward-backward algorithm converts equation (3.42) into a

form that is both recursive as well as reduces the computational complexity from an otherwise intractable computation to one that is readily computable.

With CTC, consider a modified "label sequence" $l'$, that caters for blank characters in between regular ones $l$, as defined in $A$. Thus, if $U$ is defined as the length of $l$. Then $U'$ is of length $2U + 1$. CTC therefore integrates probability distributions of transitions between blank and non-blank labels at the same time CTC calculates those transition occurring between pairs of distinct non-blank labels. The forward variable, $\alpha(t, u)$ now becomes the summed probability of all length $t$ paths that are mapped by $\mathcal{B}$ onto the length $\lfloor u/2 \rfloor$ prefix of $l$. (Note, $\lfloor u/2 \rfloor$ is the floor of $u/2$, the greatest integer less than or equal to $u/2$.) For some sequence $s$, let $s_{p:q}$ denote the sub-sequence $s_p, s_{p+1}, \ldots, s_{q-1}, s_q$, and define the set $V(t, u) \equiv \{\pi \in A'^t : \mathcal{B}(\pi) = l_{1:\lfloor u/2 \rfloor} \text{ and } \pi_t = l'_u\}$. $\alpha(t, u)$ then becomes

$$\alpha(t, u) \equiv \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y_{i,\pi_i} \tag{3.43}$$

The forward variables at time $t$ can be calculated recursively from those at time $t - 1$ and expressed as the sum of the forward variables with and without the final blank at time $T$.

$$\Pr(l \,|\, x) = \alpha(T, U') + \alpha(T, U' - 1) \tag{3.44}$$

All correct paths must start with either a blank ($b$) or the first symbol in $l$ ($l_1$), yielding the following initial conditions:

$$\begin{aligned} \alpha(1, 1) &= y_{1,b} \\ \alpha(1, 2) &= y_{1,l_1} \\ \alpha(1, u) &= 0, \, \forall u > 2 \end{aligned} \tag{3.45}$$

Thereafter the variables can be calculated recursively:

$$\alpha(t, u) = y_{t,l'_u} \sum_{i=f(u)}^{u} \alpha(t - 1, i) \tag{3.46}$$

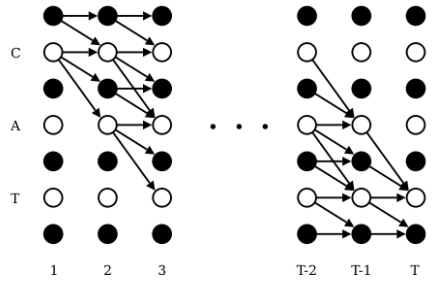


Figure 3.4: Beam Search Lattice Structure (Graves et al., 2006)

where

$$f(u) = \begin{cases} u-1, & \text{if } l'_u = blank \text{ or } l'_{u-2} = l'_u \\ u-2, & \text{otherwise} \end{cases} \tag{3.47}$$

Graphically we can express the recurrence relation for $\alpha(t,u)$ as follows.

where $t$ runs along the $x$ axis and $u$ runs along the $y$ axis. The black circles of the diagram represent *blank* elements of $l'$ while the white circles represent non-*blank* elements of $l'$. The arrows represent computational dependencies derived from our recursion relation for $\alpha(t,u)$. So, for example, the value of $\alpha(2,3)$, corresponding to the *blank* at $t=2$ and $u=3$, is derived from $\alpha(1,2)$. Similarly, the value of $\alpha(2,2)$, corresponding to the letter $c$ at $t=2$ and $u=2$, is derived from $\alpha(1,2)$ and $\alpha(1,1)$.

$$\alpha(t,u) = 0 \quad \forall u < U' - 2(T-t) - 1 \tag{3.48}$$

because these variables correspond to states for which there are not enough time-steps left to complete the sequence. We also impose the boundary condition

$$\alpha(t,0) = 0 \quad \forall t \tag{3.49}$$

The backward variables $\beta(t,u)$ are defined as the summed probabilities of all paths starting at $t+1$ that "complete" $l$ when appended to any path $\hat{\pi}$ contributing to $\alpha(t,u)$. Define $W(t,u) \equiv \{\pi \in A'^{T-t} : \mathcal{B}(\hat{\pi}+\pi) = l \ \forall \hat{\pi} \in V(t,u)\}$. Then

$$\beta(t,u) \equiv \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y_{t+i,\pi_i} \tag{3.50}$$

The rules for initialisation of the backward variables are as follows

$$\begin{aligned} \beta(T, U') &= 1 \\ \beta(T, U' - 1) &= 1 \\ \beta(T, u) &= 0, \, \forall u < U' - 1 \end{aligned} \tag{3.51}$$

The rules for recursion are as follows:

$$\beta(t, u) = \sum_{i=u}^{g(u)} \beta(t+1, i) y_{t+1, l'_i} \tag{3.52}$$

where

$$g(u) = \begin{cases} u+1, & \text{if } l'_u = blank \text{ or } l'_{u+2} = l'_u \\ u+2, & \text{otherwise} \end{cases} \tag{3.53}$$

### 3.3.3 CTC Loss function

The cross entropy error is a loss function used to measure accuracy of probabilistic measures. It is calculated as the negative log probability of a likelihood measure. The CTC loss function $\mathcal{L}(S)$ uses the cross entropy loss function of and is defined as the cross entropy error of correctly labelling all the training samples in some training set S:

$$\mathcal{L}(S) = -\ln \prod_{(x,z) \in S} \Pr(z \,|\, x) = -\sum_{(x,z) \in S} \ln \Pr(z \,|\, x) \tag{3.54}$$

where $z$ is the output label and $x$ is the input sequence. Since $\mathcal{L}(S)$ in equation 3.54 is differentiable, this loss function can be back propagated to the softmax layer in the BiRNN configuration discussed in section 3.3.

$$\mathcal{L}(x, z) \equiv -\ln \Pr(z \,|\, x) \tag{3.55}$$

and therefore

$$\mathcal{L}(S) = \sum_{(x,z) \in S} \mathcal{L}(x, z) \tag{3.56}$$

From the definition of the forward and backward variables ($\alpha(t, u)$ and $\beta(t, u)$),

we also establish that $X(t,u) \equiv \{\pi \in A'^T : \mathcal{B}(\pi) = z,\ \pi_t = z'_u\}$, such that

$$\alpha(t,u)\beta(t,u) = \sum_{\pi \in X(t,u)} \prod_{t=1}^{T} y_{t,\pi_t} \tag{3.57}$$

then substituting $\Pr(\pi \,|\, x)$ from the expression in equation 3.40, we have

$$\alpha(t,u)\beta(t,u) = \sum_{\pi \in X(t,u)} \Pr(\pi \,|\, x) \tag{3.58}$$

Also observe that $\Pr(l \,|\, x)$ is equivalent to the total probability $\Pr(z \,|\, x)$. Paths going through $z'_u$ at time $t$ can be obtained as summed over all $u$ to get

$$\Pr(z \,|\, x) = \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{3.59}$$

Thus a sample loss is determined by

$$\mathcal{L}(x,z) = -\ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{3.60}$$

and therefore the overall loss is given by

$$\mathcal{L}(S) = -\sum_{(x,z) \in S} \ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{3.61}$$

In the model described in this work, the gradient $\mathcal{L}(x,z)$is computed using TensorFlow's automatic differentiation capabilities. In practice, computations soon lead to underflow however the log scale, being used in the above loss function calculations avoids this situation and another useful equation in this context is

$$\ln(a+b) = \ln(a) + \ln(1 + e^{\ln b - \ln a}) \tag{3.62}$$

# Chapter 4

# Deep Scattering network

Curve fitting is a very common theme in pattern recognition. The concept of invariant functions convey mapping functions that approximate a discriminating function when a parent function is reduced from a high dimensional space to a low dimensional space Mallat (2016). In this chapter an invariance function called a scattering transform enables invariance of groups of deformations that could apply to speech signals thereby preserving higher level characterisations useful for classifying speech sounds. Works done by (Andén and Mallat, 2011, Peddinti et al., 2014, Sainath et al., 2014, Zeghidour et al., 2016) have shown that when the scattering spectrum are applied to speech signals and used as input to speech systems have state of the art performance. In particular Sainath et al. (2014) shows 4-7% relative improvement in word error rates (WER) over Mel frequencies cepstral coefficients (MFCCs) for 50 and 430 hours of English Broadcast News speech corpus. While experiments have been performed with hybrid HMM-DNN systems in the past, this thesis focuses on the use of scatter transforms in end-to-end RNN speech models.

This chapter iterates the use of the Fourier transform as the starting analysis function for building invariant functions and then discusses the Mel filter bank solution and then establishes why the scattering transform through the wavelet modulus operator provides better invariance features over the Mel filters.

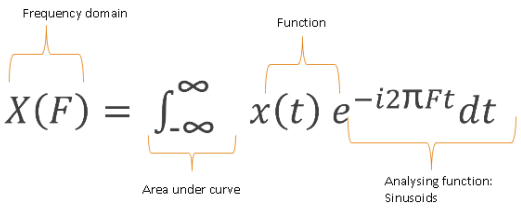


Figure 4.1: Fourier Equation

## 4.1 Fourier transform

The Fourier transform often referred to as the power spectrum, allows us to discover frequencies contained within a signal. The Fourier transform is a convolution between a signal and a complex sinusoid from $-\infty$ to $+\infty$ (Figure 4.1).

From the orthogonal property of complex exponential function, two functions are orthogonal if $\int f(x)g(x) = 0$ where f(x) and g(x) are complimentary functions, one being referred to as the analysis equation and the other referred to as the synthesis function.

If the discrete form of the Fourier transform analysis equation is given by

$$a_k = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{\left(-j\frac{2\pi kt}{T}\right)} \tag{4.1}$$

Then, the corresponding synthesis equation is given by

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{\left(j\frac{2\pi kt}{T}\right)} \tag{4.2}$$

Recall that $x(t)$ is the original signal while $a_k$ is the Fourier Series coefficient. This coefficient indicates the amplitude and phase of the original signal's higher order harmonics indexed by $k$ such that higher values of $k$ correspond to higher frequency components. In a typical spectrogram (figure 4.2), it can be seen that the energy of the signal is concentrated about a central region and then harmonic spikes of energy content exponentially decrease and taper off. Therefore in figure 4.2, the energies are concentrated at frequencies of about 100, 150 and 400 hertz.

The Fourier transform discussed in the previous section constitutes a valuable tool for the analysis of the frequency component of a signal. However is not able to determine when in time a frequency occurs hence is not able to analyse time related

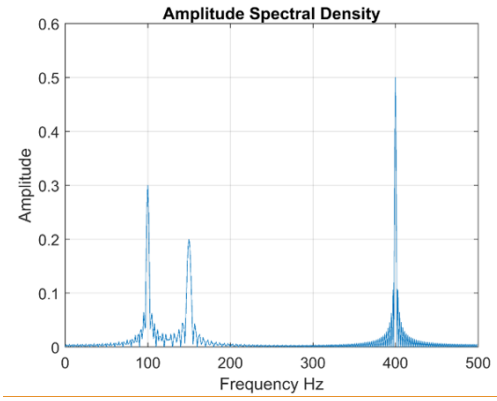


Figure 4.2: Sample Spectrogram
**?**

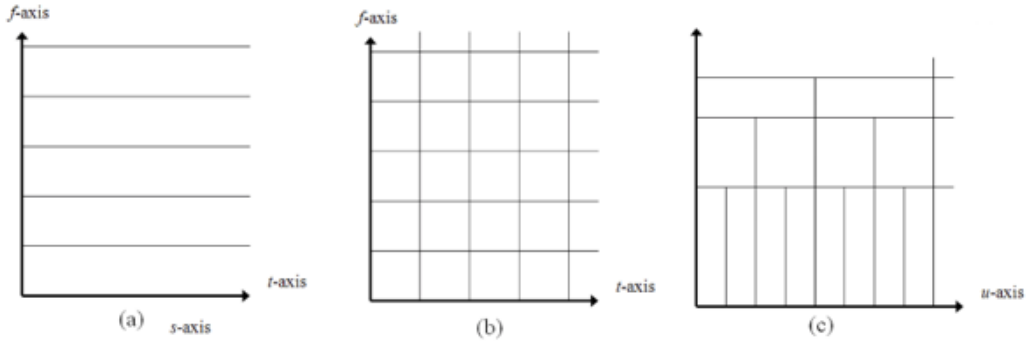


Figure 4.3: Time frequency tiling for (a) Fourier Transform (b) Short-time Fourier Transform (STFT) (c) Wavelet transform

signal deformations. The Short-time Fourier Transform (STFT) attempts to salvage this by windowing the signal in time signal and performing Fourier transforms over sliding windows sections of the original signal rather than the whole signal. There is however, a resolution trade off that ensues from this operation such that, the higher the resolution in time accuracy, the lower the frequency accuracy and vice versa. In the next section on the continuous wavelet transform, how the wavelet transform improves on the weaknesses of the Fourier Transform and the STFT is reviewed.

## 4.2 Wavelet transform

The continuous wavelet transform can be defined as a signal multiplied by scaled and shifted version of a wavelet function $\psi(t)$ referred to as the mother wavelet. The time-frequency tile-allocation of the three basic transforms examined in the first part of this chapter is illustrated in figure **??**

It can be seen here that for the Fourier transform there is no time information obtained. In the STFT, as there is no way of telling where in time the frequencies are contained, the STFT makes a blanket range of the resolution of the window and is therefore equally tiled potentially losing information based on this setup. For the case of the wavelet, because it is a scaled and shifted convolution, it takes care of the this problem providing a good resolution in both time and frequency. The fundamental representation of the continuous wavelet function is:

$$C(a,b) = \int f(t) \frac{1}{\sqrt{a}} \psi \left( \frac{t-b}{a} \right) dt \tag{4.3}$$

In this equation, $a$ and $b$ respectively represent the scaling and shifting resolution variables of the wavelet function. This is referred to as a mother wavelet. A few other mother wavelet functions discussed later in this chapter. Generally a mother wavelet is identified as being energy spikes in an infinite signal whose accumulative energy sums to zero.

## 4.3 Discrete and Fast wavelet transform

Synthesis and analysis equations (4.2 and 4.1) can be formulated as a linear combination of the basis $\phi_k(t)$ such that the basis, $\phi_k(t) = e^{j2\pi kt}$, and it's conjugate or orthonormal basis, $\tilde{\phi}_k(t) = e^{-j2\pi kt}$, equations (4.2 and 4.1) now become

$$x(t) = \sum_k a_k \phi_k \tag{4.4}$$

$$a_k = \int x(t) \tilde{\phi}_k(t) \tag{4.5}$$

With respect to scaling and shifting variables of continuous wavelet transforms in equation (4.3), a similar linear combination transformation can be applied by constructing orthonormal bases parameters, referred to as scaling ($\phi$) and translating ($\psi$) functions. For example, a simple Haar mother wavelet transform associated with a delta function, it is seen that:

$$\phi_{j,k}(t) = 2^{j/2} \phi(2^j t - k) \tag{4.6}$$

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k) \tag{4.7}$$

where j is associated with the dilation (scaling) parameter and k is associated with the position (shifting) parameter. If the Haar coefficients $h_{(.)}[n] = \{1/\sqrt{2}, 1/\sqrt{2}\}$ are extracted we have the following dilation and position parameters.

$$\phi(t) = h_\phi[n]\sqrt{2}\phi(2t - n) \tag{4.8}$$

$$\psi(t) = h_\phi[n]\sqrt{2}\psi(2t - n) \tag{4.9}$$

For any signal, a discrete wavelet transform in $l^2(Z)^1$ can be approximated by

$$f[n] = \frac{1}{\sqrt{M}}\sum_k W_\phi[j_0, k]\phi_{j_0,k}[n] + \frac{1}{\sqrt{M}}\sum_{j=j_0}^{\infty}\sum_k W_\psi[j, k]\psi_{j,k}[n] \tag{4.10}$$

Here $f[n], \phi_{j_0,k}[n]$ and $\psi_{j,k}[n]$ are discrete functions defined in [0,M - 1], having a total of M points. Because the sets $\{\phi_{j_0,k}[n]\}_{k\in\mathbf{Z}}$ and $\{\psi_{(j,k)\in\mathbf{Z}^2, j\geq j_0}\}$ are orthogonal to each other. We can simply take the inner product to obtain the wavelet coefficients.

$$W_\phi[j_0, k] = \frac{1}{\sqrt{M}}\sum_n f[n]\phi_{j_0,k}[n] \tag{4.11}$$

$$W_\psi[j, k] = \frac{1}{\sqrt{M}}\sum_n f[n]\psi_{j,k}[n] \quad j \geq j_0 \tag{4.12}$$

Equation (4.11) is called approximation coefficient while (4.12) is called detailed coefficients.

These two components show that the approximation coefficient, $W_\phi[j_0, k]$, models a low pass filter and the detailed coefficient,$W_\psi[j_0, k]$, models a high pass filter. It is possible to determine the approximation and detailed coefficients without the scaling and dilating parameters. The resulting coefficients, called the fast wavelet transform, are a convolution between the wavelet coefficients and a down-sampled version of the next order coefficients. The fast wavelet transform was first postulated in (Mallat, 1989).

$$W_\phi[j, k] = h_\phi[-n] * W_\phi[j + 1, n]|_{n=2k,k\geq 0} \tag{4.13}$$

$$W_\psi[j_0, k] = h_\psi[-n] * W_\phi[j+1, n]|_{n=2k, k\geq 0} \tag{4.14}$$

For analysis of the Haar wavelet and the derivation of equations (4.13 and 4.14) see appendix **??**.

## 4.4 Mel filter banks

Mel Frequency Cepstral Coefficients (MFCCs) are a feature widely used in automatic speech and speaker recognition. They were introduced by Davis and Mermelstein in the 1980's, and have been state-of-the-art ever since. Prior to the introduction of MFCCs, Linear Prediction Coefficients (LPCs) and Linear Prediction Cepstral Coefficients (LPCCs) and were the main feature type for automatic speech recognition (ASR), especially with HMM classifiers.

An audio signal is constantly changing, so to simplify things we assume that on short time scales the audio signal doesn't change much (when we say it doesn't change, we mean statistically i.e. statistically stationary, obviously the samples are constantly changing on even short time scales). This is why we frame the signal into 20-40ms frames. If the frame is much shorter we don't have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame.

The next step is to calculate the power spectrum of each frame. This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds. Depending on the location in the cochlea that vibrates (which wobbles small hairs), different nerves fire informing the brain that certain frequencies are present. Our periodogram estimate performs a similar job for us, identifying which frequencies are present in the frame.

The periodogram spectral estimate still contains a lot of information not required for Automatic Speech Recognition (ASR). In particular the cochlea can not discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason we take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by our Mel filter bank: the first filter is very narrow and gives an indication of how much energy exists near 0 Hertz. As the frequencies get

higher our filters get wider as we become less concerned about variations. We are only interested in roughly how much energy occurs at each spot. The Mel scale tells us exactly how to space our filter banks and how wide to make them.

Once we have the filter bank energies, we take the logarithm of them. This is also motivated by human hearing: we don't hear loudness on a linear scale. Generally to double the perceived volume of a sound we need to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression operation makes our features match more closely what humans actually hear. Why the logarithm and not a cube root? The logarithm allows us to use cepstral mean subtraction, which is a channel normalisation technique.

The final step is to compute the DCT of the log filterbank energies. There are 2 main reasons this is performed. Because our filterbanks are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal co-variance matrices can be used to model the features in e.g. a HMM classifier. But notice that only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade ASR performance, so we get a small improvement by dropping them.

## 4.5 Deep scattering spectrum

In this section reference is made to (Andén and Mallat, 2011, 2014, Zeghidour et al., 2016). For a signal $x$ we define the following transform $W_x$ as a convolution with a low-pass filter $\phi$ and higher frequency complex analytic wavelets $\psi_{\lambda_1}$:

$$Wx = (x \star \phi(t), x \star \psi_{\lambda_1}(t))_{t \in R, \lambda_1 \in \Lambda_1} \tag{4.15}$$

We apply a modulus operator to the wavelet coefficients to remove complex phase and extract envelopes at different resolutions

$$|W|x = (x \star \phi(t), |x \star \psi_{\lambda_1}(t)|)_{t \in R, \lambda_1 \in \Lambda_1} \tag{4.16}$$

$S_0x = x \star \phi(t)$ is locally invariant to translation thanks to the time averaging $\phi$. This time-averaging loses the high frequency information, which is retrieved in the wavelet modulus coefficients $|x \star \psi_{\lambda_1}|$. However, these wavelet modulus coefficients are not invariant to translation, and as for $S_0$, a local translation invariance is obtained by a time averaging which defines the first layer of scattering coefficients

$$S_1x(t, \psi_{\lambda_1}) = |x \star \psi_{\lambda_1}| \star \phi(t)) \tag{4.17}$$

It is shown in Andén and Mallat (2014) that if the wavelets $\psi_{\lambda_1}$ have the same frequency resolution as the standard Mel-filters, then the $S_1x$ coefficients approximate the Mel-filter coefficients. Unlike the Mel-filter banks however, there is a strategy to recover the lost information, by passing the wavelet modulus coefficients $|x \star \phi_{\lambda_1}|$ through a bank of higher frequency wavelets $\psi_{\lambda_2}$:

$$|W_2||x \star \phi_{\lambda_1}| = (|x \star \psi_{\lambda_1}| \star \phi, ||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}|)_{\lambda_2 \in \Lambda_2}) \tag{4.18}$$

This second layer of wavelet modulus coefficients is still not invariant to translation, hence we average these coefficients with a low-pass filter $\phi$ to derive a second layer of of scattering coefficients.

$$|W_2||x \star \phi_{\lambda_1}| = (|x \star \psi_{\lambda_1}| \star \phi, ||x \star \psi_{\lambda_1}| \star \psi_{\lambda_2}|)_{\lambda_2 \in \Lambda_2}) \tag{4.19}$$

Repeating these successive steps of computing invariant features and retrieving lost information leads to the scattering spectrum, as seen in Fig. 1, however speech signals are almost entirely characterized by the first two layers of the spectrum, that is why a two layers spectrum is typically used for speech representation. It is shown in [6] that this representation is invariant to translations and stable to deformations, while keeping more information than the Mel-filter banks coefficients

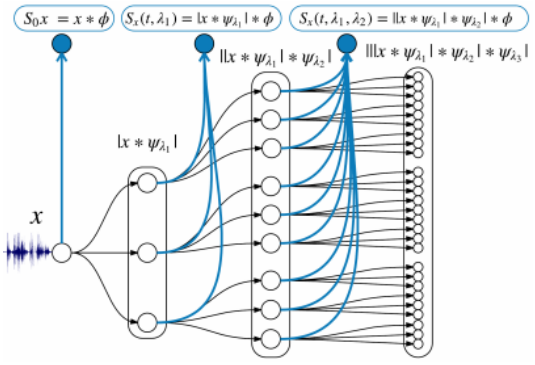


Figure 4.4: Scattering network - 2 layers deep
Zeghidour et al. (2016)