

Chapter 1

Background

This thesis combines work in several fields. We provide a background for the most important and relevant fields in this chapter. We first introduce the definition and properties of the Wavelet Transforms, focussing on the DTCWT, then we describe the fundamentals of Convolutional Neural Networks and how they learn, and finally we introduce the Scattering Transform, the original inspiration for this thesis.

1.1 Notation

We define standard notation to help the reader better understand figures and equations. Many of the terms we define here relate to concepts that have not been introduced yet, so may be unclear until later.

- **Pixel coordinates**

When referencing spatial coordinates in an image, the preferred index is \mathbf{u} for a 2D vector of coordinates, or $[u_1, u_2]$ if we wish to specify the coordinates explicitly. u_1 indexes rows from top to bottom of an image, and u_2 indexes columns from left to right. We typically use $H \times W$ for the size of the image, (but this is less strict). I.e., $u_1 \in \{0, 1, \dots, H - 1\}$ and $u_2 \in \{0, 1, \dots, W - 1\}$.

- **Convolutional networks**

Image convolutional neural networks often work with 4-dimensional arrays. In particular, mini-batches of images with multiple channels. When we need to, we index over the minibatch with the variable n and over the channel dimension with c . For example, we can index an activation \mathbf{x} by $\mathbf{x}[n, c, u_1, u_2]$.

To distinguish between features, filters, weights and biases of different levels in a deep network, we may add a layer subscript, or l for the general case, i.e., $\mathbf{z}_l[n, c, \mathbf{u}]$ indexes the feature map at the l -th layer of a deep network.

- **Fourier transforms**

When referring to the Fourier transform of a function, f , we typically adopt the overbar notation: i.e., $\mathcal{F}\{f\} = \bar{f}$.

- **Wavelet Filter Banks**

Using standard notation, we define the scaling function as ϕ and the wavelet function as ψ . For a filter bank implementation of a wavelet transform, we use h for analysis and g for synthesis filters.

In a multiscale environment, j indexes scale from $\{1, 2, \dots, J\}$. For 2D complex wavelets, θ indexes the orientation at which the wavelet has the largest response, i.e., $\psi_{\theta,j}$ refers to the wavelet at orientation θ at the j -th scale.

1.2 Supervised Machine Learning

Consider a sample space over inputs and labels $\mathcal{X} \times \mathcal{Y}$ and a data generating distribution p_{data} . Given a dataset of input-label pairs $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^N$ we would like to make predictions about $p_{data}(y|x)$ that generalize well to unseen data. A common way to do this is to build a parametric model to directly estimate this conditional probability. For example, regression asserts the data are distributed according to a function of the inputs plus a noise term ϵ :

$$y = f(x, \theta) + \epsilon \quad (1.2.1)$$

This noise is often modelled as a zero mean Gaussian random variable, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, which means we can write:

$$p_{model}(y|x, \theta) = \mathcal{N}(y; f(x, \theta), \sigma^2) \quad (1.2.2)$$

with (θ, σ^2) are the parameters of the model.

We can find point estimates of the parameters by maximizing the likelihood of $p_{model}(y|x, \theta)$ (or equivalently, minimizing the KL-divergence between p_{model} and p_{data} $KL(p_{model}||p_{data})$). As the data are all i.i.d., we can multiply individual likelihoods, and solve for θ :

$$\theta_{MLE} = \arg \max_{\theta} p_{model}(y|x, \theta) \quad (1.2.3)$$

$$= \arg \max_{\theta} \prod_{n=1}^N p_{model}(y^{(n)}|x^{(n)}, \theta) \quad (1.2.4)$$

$$= \arg \max_{\theta} \sum_{n=1}^N \log p_{model}(y^{(n)}|x^{(n)}, \theta) \quad (1.2.5)$$

Using the regression model from above, this becomes:

$$\theta_{MLE} = \arg \max_{\theta} \sum_{n=1}^N \log p_{model}(y^{(n)}|x^{(n)}, \theta) \quad (1.2.6)$$

$$= \arg \max_{\theta} \left(-N \log \sigma - \frac{N}{2} \log(2\pi) - \sum_{n=1}^N \frac{(y^{(n)} - f(x^{(n)}, \theta))^2}{2\sigma^2} \right) \quad (1.2.7)$$

$$= \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N \frac{(y^{(n)} - f(x^{(n)}, \theta))^2}{2} \quad (1.2.8)$$

which gives us the well known result that we would like to find parameters that minimize the mean squared error (MSE) between observations y and predictions $\hat{y} = f(x, \theta)$.

For binary classification, ($y \in \{0, 1\}$) instead of the model in (1.2.2) we have:

$$p_{model}(y|x, \theta) = \text{Ber}(y; \sigma(f(x, \theta))) \quad (1.2.9)$$

where σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (1.2.10)$$

This expands naturally to multi-class classification ($y \in \{0, 1\}^C$) by swapping the Bernoulli distribution for the categorical and the sigmoid for a softmax function, defined by:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^C e^{z_k}} \quad (1.2.11)$$

If we let $\hat{y}_i = \sigma_i(f(x, \theta))$, this makes (1.2.9):

$$p_{model}(y|x, \theta) = \text{Cat}(y; \sigma(f(x, \theta))) \quad (1.2.12)$$

$$= \prod_{c=1}^C \prod_{n=1}^N (\hat{y}_c^{(n)})^{\mathbb{I}(y_c^{(n)}=1)} \quad (1.2.13)$$

where $\mathbb{I}(x)$ is the indicator function. As $y_c^{(n)}$ is either 0 or 1, we remove the indicator function. Maximizing this likelihood to find the ML estimate for θ :

$$\theta_{MLE} = \arg \min_{\theta} \prod_{c=1}^C \prod_{n=1}^N (\hat{y}_c^{(n)})^{y_c^{(n)}} \quad (1.2.14)$$

$$= \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N \sum_{c=1}^C y_c^{(n)} \log \hat{y}_c^{(n)} \quad (1.2.15)$$

which we recognize as the cross-entropy between y and \hat{y} .

1.2.1 Priors on Parameters and Regularization

Maximum likelihood estimates for parameters, while straightforward, can often lead to overfitting. A common practice is to regularize learnt parameters θ by putting a prior over them. If we do not have any prior information about what we expect the parameters to be, it is still useful to put an uninformative prior on the weights. For example, if our weights are in the reals, a commonly used prior is a Gaussian.

Let us extend the regression example from above by saying we would like the prior on the weights θ to be a Gaussian, i.e. $p(\theta) = \mathcal{N}(0, \tau^2)$. The corresponding maximum a posteriori (MAP) estimate is then obtained by finding:

$$\theta_{MAP} = \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N \frac{\left(y^{(n)} - f(x^{(n)}, \theta)\right)^2}{2} + \lambda \|\theta\|_2^2 \quad (1.2.16)$$

where $\lambda = \sigma^2/\tau^2$, which is equivalent to minimizing the MSE with an ℓ_2 penalty on the parameters. λ is often called **weight decay** in the neural network literature, which we will also use in this thesis.

1.2.2 Loss Functions and Minimizing the Objective

It may be useful to rewrite (1.2.16) as an objective function on the parameters $J(\theta)$:

$$J(\theta) = \frac{1}{N} \sum_{n=1}^N \frac{\left(y^{(n)} - f(x^{(n)}, \theta)\right)^2}{2} + \lambda \|\theta\|_2^2 \quad (1.2.17)$$

$$= L_{data}(y, f(x, \theta)) + L_{reg}(\theta) \quad (1.2.18)$$

where L_{data} is the data loss defined, such as MSE or cross-entropy and L_{reg} is the regularization, such as ℓ_2 or ℓ_1 penalized loss.

Now $\theta_{MAP} = \arg \min J(\theta)$. Finding the minimum of the objective function is task-dependent and is often not straightforward. One commonly used technique is called *gradient descent* (GD). This is straightforward to do as it only involves calculating the gradient at a given point and taking a small step in the direction of steepest descent. The difference equation defining this can be written as:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial J}{\partial \theta} \quad (1.2.19)$$

Unsurprisingly, such a simple technique has limitations. In particular, it has a slow convergence rate when the condition number (ratio of largest to smallest eigenvalues) of the Hessian around the optimal point is large [1]. An example of this is shown in Figure 1.1. In this

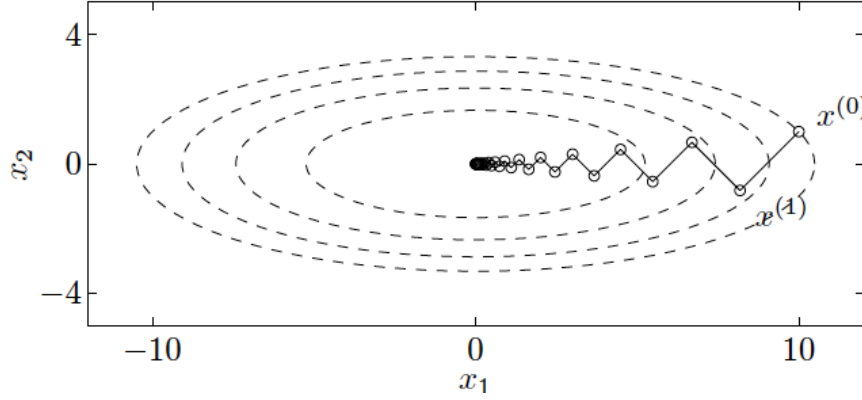


Figure 1.1: **Trajectory of gradient descent in an ellipsoidal parabola.** Some contour lines of the function $f(x) = 1/2 (x_1^2 + 10x_2^2)$ and the trajectory of GD optimization using exact line search. This space has condition number 10, and shows the slow convergence of GD in spaces with largely different eigenvalues. Image taken from [1] Figure 9.2.

figure, the step size is chosen with exact line search, i.e.

$$\eta = \arg \min_s f(x + s \frac{\partial f}{\partial x}) \quad (1.2.20)$$

To truly overcome this problem, we must know the curvature of the objective function $\frac{\partial^2 J}{\partial \theta^2}$. An example optimization technique that uses the second order information is Newton's method. Such techniques sadly do not scale with size, as computing the Hessian is proportional to the number of parameters squared, and most neural networks have hundreds of thousands, if not millions of parameters. In this thesis, we only consider *first-order optimization* algorithms.

1.2.3 Stochastic Gradient Descent

Aside from the problems associated the curvature of the function $J(\theta)$, another common issue faced with the gradient descent of (1.2.19) is the cost of computing $\frac{\partial J}{\partial \theta}$. In particular, the first term:

$$L_{data}(y, f(x, \theta)) = \mathbb{E}_{x, y \sim p_{data}} [L_{data}(y, f(x, \theta))] \quad (1.2.21)$$

$$= \frac{1}{N} \sum_{n=1}^N L_{data}(y^{(n)}, f(x^{(n)}, \theta)) \quad (1.2.22)$$

involves evaluating the entire dataset at the current values of θ . As the training set size grows into the thousands or millions of examples, this approach becomes prohibitively slow.

(1.2.21) writes the data loss as an expectation, hinting at the fact that we can remedy this problem by using fewer samples $N_b < N$ to evaluate L_{data} . This variation is called Stochastic Gradient Descent (SGD).

Choosing the batch size is a hyperparameter choice that we must think carefully about. Setting the value very low, e.g. $N_b = 1$ can be advantageous as the noisy estimates for the gradient have a regularizing effect on the network [2]. Increasing the batch size to larger values allows you to easily parallelize computation as well as increasing your accuracy for the gradient, allowing you to take larger step sizes [3]. A good initial starting point is to set the batch size to about 100 samples and increase/decrease from there [4].

1.2.4 Gradient Descent and Learning Rate

The step size parameter, η in (1.2.19) is commonly referred to as the learning rate. Choosing the right value for the learning rate is key. Unfortunately, the line search algorithm in (1.2.20) would be too expensive to compute for neural networks (as would involve evaluating the function several times at different values), each of which takes about as long as calculating the gradients themselves. Additionally, as the gradients are typically estimated over a mini-batch and are hence noisy there may be little added benefit in optimizing the step sizes in the estimated direction.

Figure 1.2 illustrates the effect the learning rate can have over a contrived convex example. Optimizing over more complex loss surfaces only exacerbates the problem. Sadly, choosing the initial learning rate is ‘more of an art than a science’ [4], but [5], [6] have some tips on what to set this at. We have found in our work that searching for a large learning rate that causes the network to diverge and reducing it hence can be a good search strategy. This agrees with Section 1.5 of [7] which states that for regions of the loss space which are roughly quadratic, $\eta_{max} = 2\eta_{opt}$ and any learning rate above $2\eta_{opt}$ causes divergence.

On top of the initial learning rate, the convergence of SGD methods require:

$$\sum_{t=1}^{\infty} \eta_t \rightarrow \infty \quad (1.2.23)$$

$$\sum_{t=1}^{\infty} \eta_t^2 = M \quad (1.2.24)$$

where M is finite. Choosing how to do this also contains a good amount of artistry, and there is no one scheme that works best. A commonly used greedy method is to keep the learning rate constant until the training loss stabilizes and then to enter the next phase of training by setting $\eta_{k+1} = \gamma\eta_k$ where γ is a decay factor. Choosing γ and the thresholds for triggering a step however must be chosen by monitoring the training loss curve and trial and error.

1.2.5 Momentum and Adam

One simple and very popular modification to SGD is to add *momentum*. Momentum accumulates past gradients with an exponentially moving average and continues to move in their direction. The name comes from the analogy of finding a minimum of a function to

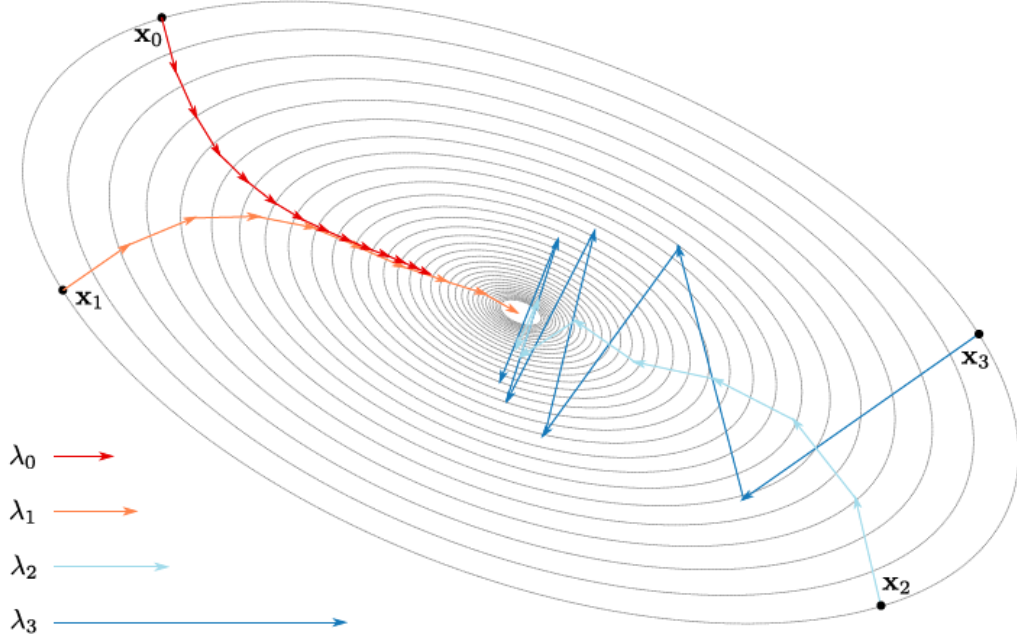


Figure 1.2: **Trajectories of SGD with different initial learning rates.** This figure illustrates the effect the step size has over the optimization process by showing the trajectory for $\eta = \lambda_i$ from equivalent starting points on a symmetric loss surface. Increasing the step size beyond λ_3 can cause the optimization procedure to diverge. Image taken from [8] Figure 2.7.

rolling a ball over a loss surface – any new force (newly computed gradients) must overcome the past motion of the ball. To do this, we create a *velocity* variable v_t and modify (1.2.19) to be:

$$v_{t+1} = \alpha v_t - \eta_k \frac{\partial J}{\partial \theta} \quad (1.2.25)$$

$$\theta_{t+1} = \theta_t + v_{t+1} \quad (1.2.26)$$

$$(1.2.27)$$

where $0 \leq \alpha < 1$ is the momentum term indicating how quickly to ‘forget’ past gradients.

Another popular modification to SGD is the adaptive learning rate technique Adam [9]. There are several other adaptive schemes such as AdaGrad [10] and AdaDelta [11], but they are all quite similar, and Adam is often considered the most robust of the three [4]. The goal of all of these adaptive schemes is to take larger update steps in directions of low variance, helping to minimize the effect of large condition numbers we saw in ???. Adam does this by

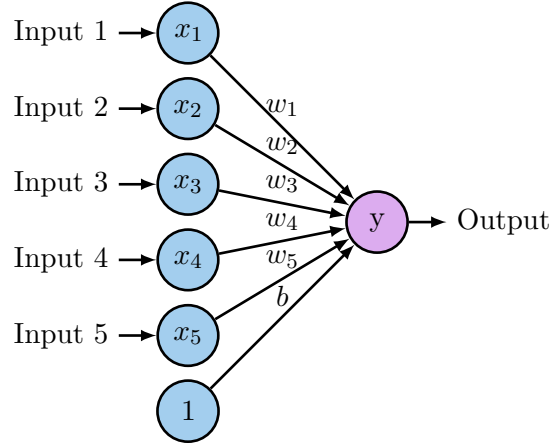


Figure 1.3: **A single neuron.** The neuron is composed of inputs x_i , weights w_i (and a bias term), as well as an activation function. Typical activation functions include the sigmoid function, tanh function and the ReLU

keeping track of the first m_t and second v_t moments of the gradients:

$$g_{t+1} = \frac{\partial J}{\partial \theta} \quad (1.2.28)$$

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) g_{t+1} \quad (1.2.29)$$

$$v_{t+1} = \beta_2 m_t + (1 - \beta_2) g_{t+1} \quad (1.2.30)$$

$$(1.2.31)$$

where $0 \leq \beta_1, \beta_2 < 1$. Note the similarity between updating the mean estimate in (1.2.29) and the velocity term in (1.2.25)¹. The parameters are then updated with:

$$\theta_{t+1} = \theta_t - \eta \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} \quad (1.2.32)$$

where ϵ is a small value to avoid dividing by zero.

1.3 Neural Networks

1.3.1 The Neuron and Single Layer Neural Networks

The neuron, shown in Figure 1.3 is the core building block of Neural Networks. It takes the dot product between an input vector $\mathbf{x} \in \mathbb{R}^D$ and a weight vector \mathbf{w} , before applying a

¹The m_{t+1} and v_{t+1} terms are then bias-corrected as they are biased towards zero at the beginning of training. We do not include this for conciseness.

chosen nonlinearity, g . I.e.

$$y = g(\langle \mathbf{x}, \mathbf{w} \rangle) = g\left(\sum_{i=0}^D x_i w_i\right) \quad (1.3.1)$$

where we have used the shorthand $b = w_0$ and $x_0 = 1$. Also note that we will use the neural network common practice of calling the *weights* w , compared to the parameters θ we have been discussing thus far.

Typical nonlinear functions g are the sigmoid function (already presented in (1.2.10)), but also common are the tanh and ReLU functions:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (1.3.2)$$

$$\text{ReLU}(x) = \max(x, 0) \quad (1.3.3)$$

See Figure 1.6 for plots of these. The original Rosenblatt perceptron [12] used the Heaviside function $H(x) = \mathbb{I}(x > 0)$.

Note that if $\langle \mathbf{w}, \mathbf{w} \rangle = 1$ then $\langle \mathbf{x}, \mathbf{w} \rangle$ is the distance from the point \mathbf{x} to the hyperplane with normal \mathbf{w} . With general \mathbf{w} this can be thought of as a scaled distance. Thus, the weight vector \mathbf{w} defines a hyperplane in \mathbb{R}^D which splits the space into two. The choice of nonlinearity then affects how points on each side of the plane are treated. For a sigmoid, points far below the plane get mapped to 0 and points far above the plane get mapped to 1 (with points near the plane having a value of 0.5). For tanh nonlinearities, these points get mapped to -1 and 1. For ReLU nonlinearities, every point below the plane ($\langle \mathbf{x}, \mathbf{w} \rangle < 0$) gets mapped to zero and every point above the plane keeps its inner product value.

Nearly all modern neural networks use the ReLU nonlinearity and it has been credited with being a key reason for the recent surge in deep learning success [13], [14]. In particular:

1. It is less sensitive to initial conditions as the gradients that backpropagate through it will be large even if x is large. A common observation of sigmoid and tanh non-linearities was that their learning would be slow for quite some time until the neurons came out of saturation, and then their accuracy would increase rapidly before levelling out again at a minimum glorot_understanding_2010. The ReLU, on the other hand, has constant gradient.
2. It promotes sparsity in outputs, by setting them to a hard 0. Studies on brain energy expenditure suggest that neurons encode information in a sparse manner. lennie_cost_2003 estimates the percentage of neurons active at the same time to be between 1 and 4%. Sigmoid and tanh functions will typically have *all* neurons firing, while the ReLU can allow neurons to fully turn off.

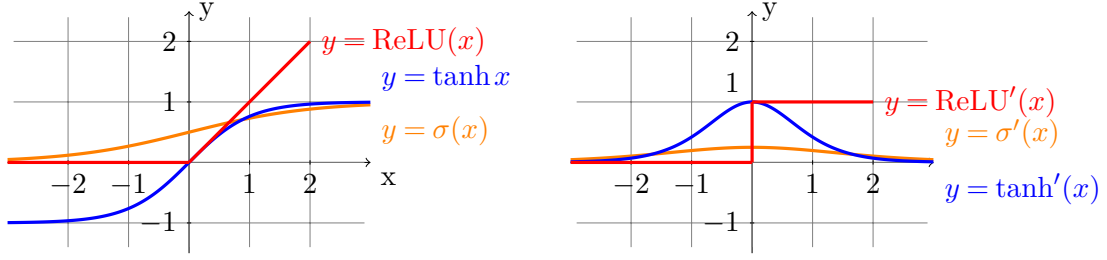


Figure 1.4: **Common Neural Network nonlinearities and their gradients.** The sigmoid, tanh and ReLU nonlinearities are commonly used activation functions for neurons. Note the different properties. In particular, the tanh and sigmoid have the nice property of being smooth but can have saturation when the input is either largely positive or largely negative, causing little gradient to flow back through it. The ReLU does not suffer from this problem, and has the additional nice property of setting values to exactly 0, making a sparser output activation.

1.3.2 Multilayer Perceptrons

As mentioned in the previous section, a single neuron can be thought of as a separating hyperplane with an activation that maps the two halves of the space to different values. Such a linear separator is limited, and famously cannot solve the XOR problem [15]. Fortunately, adding a single hidden layer like the one shown in Figure 1.5 can change this, and it is provable that with an infinitely wide hidden layer, a neural network can approximate any function [16], [17].

The forward pass of such a network with one hidden layer of H units is:

$$h_i = g\left(\sum_{j=0}^D x_j w_{ij}^{(1)}\right) \quad (1.3.4)$$

$$y = \sum_{k=0}^H h_k w_k^{(2)} \quad (1.3.5)$$

where $w^{(l)}$ denotes the weights for the l -th layer, of which Figure 1.5 has 2. Note that these individual layers are often called *fully connected* as each node in the previous layer affects every node in the next.

If we were to expand this network to have L such fully connected layers, we could rewrite the action of each layer in a recursive fashion:

$$Y^{(l+1)} = W^{(l+1)} X^{(l)} \quad (1.3.6)$$

$$X^{(l+1)} = g(Y^{(l+1)}) \quad (1.3.7)$$

where W is now a weight matrix, acting on the vector of previous layer's outputs $X^{(l)}$. As we are now considering every layer an input to the next stage, we have removed the h notation,

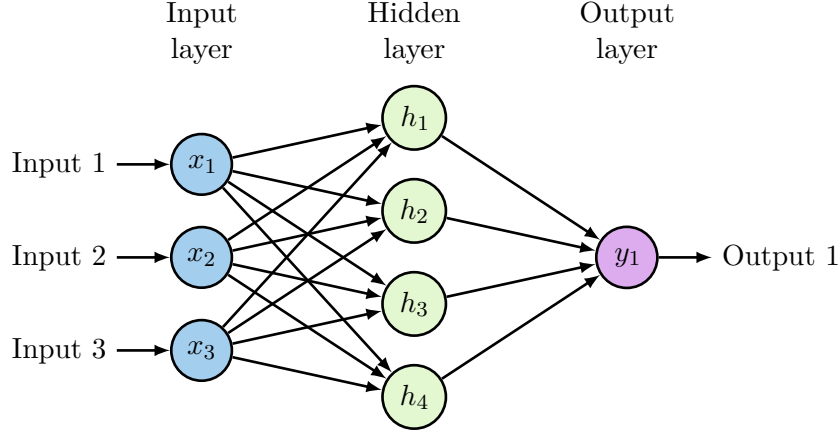


Figure 1.5: **Multi-layer perceptron.** Expanding the single neuron from Figure 1.3 to a network of neurons. The internal representation units are often referred to as the *hidden layer* as they are an intermediary between the input and output.

and added the superscript (l) to define the depth. $X^{(0)}$ is the network input and $Y^{(L)}$ is the network output. Let us say that the output has C nodes, and a hidden layer $X^{(l)}$ has C_l nodes.

1.3.3 Backpropagation

It is important to truly understand backpropagation when designing neural networks, so we describe the core concepts now for a neural network with L layers.

The delta rule, initially designed for networks with no hidden layers [18], was expanded to what we now consider *backpropagation* in [19]. While backpropagation is conceptually just the application of the chain rule, Rumelhart, Hinton, and Williams successfully updated the delta rule to networks with hidden layers, laying a key foundational step for deeper networks.

With a deep network, calculating $\frac{\partial J}{\partial w}$ may not seem particularly obvious if w is a weight in one of the earlier layers. We need to define a rule for updating the weights in all L layers of the network, $W^{(1)}, W^{(2)}, \dots, W^{(L)}$ however, only the final set $W^{(L)}$ are connected to the objective function J .

1.3.3.1 Regression Loss

Let us start with writing down the derivative of J with respect to the network output $Y^{(L)}$ using the regression objective function (1.2.8). As we now have two superscripts, one for the

sample number and one for the layer number, we combine them into a tuple of superscripts.

$$\frac{\partial J}{\partial Y^{(L)}} = \frac{\partial}{\partial Y^{(L)}} \left(\frac{1}{N} \sum_{n=1}^{N_b} \frac{1}{2} \left(y^{(n)} - Y^{(L,n)} \right)^2 \right) \quad (1.3.8)$$

$$= \frac{1}{N} \sum_{n=1}^{N_b} \left(Y^{(L,n)} - y^{(n)} \right) \quad (1.3.9)$$

$$= e \in \mathbb{R} \quad (1.3.10)$$

where we have used the fact that for the regression case, $y^{(n)}, Y^{(L,n)} \in \mathbb{R}$.

1.3.3.2 Classification Loss

For the classification case (1.2.15), let us keep the output of the network $Y^{(L,n)} \in \mathbb{R}^C$ and define an intermediate value \hat{y} the softmax applied to this vector $\hat{y}_c^{(n)} = \sigma_c \left(Y^{(L,n)} \right)$. Note that this is a vector valued function going from $\mathbb{R}^C \rightarrow \mathbb{R}^C$ so has a jacobian matrix $S_{ij} = \frac{\partial \hat{y}_i}{\partial Y_j^{(L)}}$ with values:

$$S_{ij} = \begin{cases} \sigma_i(1 - \sigma_j) & \text{if } i = j \\ -\sigma_i \sigma_j & \text{if } i \neq j \end{cases} \quad (1.3.11)$$

Now, let us return to (1.2.15) and find the derivative of the objective function to this intermediate value \hat{y} :

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left(\frac{1}{N} \sum_{n=1}^{N_b} \sum_{c=1}^C y_c^{(n)} \log \hat{y}_c^{(n)} \right) \quad (1.3.12)$$

$$= \frac{1}{N} \sum_{n=1}^{N_b} \sum_{c=1}^C \frac{y_c^{(n)}}{\hat{y}_c^{(n)}} \quad (1.3.13)$$

$$= d \in \mathbb{R}^C \quad (1.3.14)$$

Note that unlike (1.3.10), this derivative is vector valued. To find $\frac{\partial J}{\partial Y^{(L)}}$ we use the chain rule. It is easier to find the partial derivative with respect to one node in the output first, and then expand from here. I.e.:

$$\frac{\partial J}{\partial Y_j^{(L)}} = \sum_{i=1}^C \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial Y_j^{(L)}} \quad (1.3.15)$$

$$= S_i^T d \quad (1.3.16)$$

where S_i is the i th column of the jacobian matrix S . It becomes clear now that to get the entire vector derivative for all nodes in $Y^{(L)}$, we must multiply the transpose of the jacobian

matrix with the error term from (1.3.14):

$$\frac{\partial J}{\partial Y^{(L)}} = S^T d \quad (1.3.17)$$

1.3.3.3 Final Layer Weight Gradient

Let us continue by assuming $\frac{\partial J}{\partial Y^{(L)}}$ is vector valued as was the case with classification. For regression, it is easy to set $C = 1$ in the following to get the necessary results. Additionally for clarity, we will drop the layer superscript in the intermediate calculations.

We call the gradient for the final layer weights the *update* gradient. It can be computed by the chain rule again:

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial Y_i} \frac{\partial Y_i}{\partial W_{ij}} + 2\lambda W_{ij} \quad (1.3.18)$$

$$= \frac{\partial J}{\partial Y_i} X_j + 2\lambda W_{ij} \quad (1.3.19)$$

where the second term in the above two equations comes from the regularization loss that is added to the objective. The gradient of the entire weight matrix is then:

$$\frac{\partial J}{\partial W^{(L)}} = \frac{\partial J}{\partial \hat{y}} X^T + 2\lambda W \quad (1.3.20)$$

$$= S^T d \left(X^{(L-1)} \right)^T + 2\lambda W^{(L)} \in \mathbb{R}^{C \times C_{L-1}} \quad (1.3.21)$$

1.3.3.4 Final Layer Passthrough Gradient

Additionally, we want to find the *passthrough* gradients of the final layer $\frac{\partial J}{\partial X^{(L-1)}}$. In a similar fashion, we first find the gradient with respect to individual elements in $X^{(L-1)}$ before generalizing to the entire vector:

$$\frac{\partial J}{\partial X_i} = \sum_{j=1}^C \frac{\partial J}{\partial Y_j} \frac{\partial Y_j}{\partial X_i} \quad (1.3.22)$$

$$= \sum_{j=1}^C \frac{\partial J}{\partial Y_j} W_{j,i} \quad (1.3.23)$$

$$= W_i^T \frac{\partial J}{\partial Y} \quad (1.3.24)$$

$$(1.3.25)$$

where W_i is the i th column of W . Thus

$$\frac{\partial J}{\partial X^{(L-1)}} = \left(W^{(L)} \right)^T \frac{\partial J}{\partial Y^{(L)}} \quad (1.3.26)$$

$$= \left(W^{(L)} \right)^T S^T d \quad (1.3.27)$$

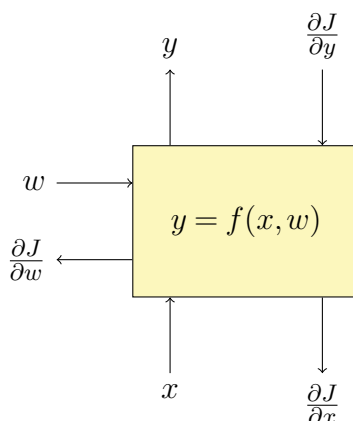


Figure 1.6: **General block form for autograd.** All neural network functions need to be able to calculate the forward pass $y = f(x, w)$ as well as the update and passthrough gradients $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial x}$. Backpropagation is then easily done by allowing data to flow backwards through these blocks from the loss.

This passthrough gradient then can be used to update the next layer’s weights by repeating ?? and ??.

1.3.3.5 General Layer Update

The easiest way to handle this flow of gradients, and the basis for most automatic differentiation packages, is the block definition shown in ??. For all neural network components (even if they do not have weights), the operation must not only be able to calculate the forward pass $y = f(x, w)$ given weights w and inputs x , but also calculate the *update* and *passthrough* gradients $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial x}$ given an input gradient $\frac{\partial J}{\partial y}$. The input gradient will have the same shape as y as will the update and passthrough gradients match the shape of w and x . This way, gradients for the entire network can be computed in an iterative fashion starting at the loss function and moving backwards.

1.3.4 Extending to multiple layers and different block types

1.3.4.1 Convolutional Layers

The image/layer of features is convolved by a set of filters. The filters are typically small, ranging from 3×3 in ResNet and VGG to 11×11 in AlexNet. We have quoted only spatial size here, as the filters in a CNN are always *fully connected in depth* — i.e., they will match the number of channels their input has.

Figure 1.7: Differences in non-linearities. Green — the *sigmoid* function, Blue — the *tanh* function, and Red — the *ReLU*. The ReLU solves the problem of small gradients outside of the activation region (around $x = 0$) as well as promoting sparsity.

Figure 1.8: ?? Tight 2×2 pooling with stride 2, vs ?? overlapping 3×3 pooling with stride 2. Overlapping pooling has the possibility of having one large activation copied to two positions in the reduced size feature map, which places more emphasis on the odd columns.

For an input $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$, and filters $\mathbf{f} \in \mathbb{R}^{H' \times W' \times D \times D''}$ (D'' is the number of filters), our output $\mathbf{z} \in \mathbb{R}^{H'' \times W'' \times D''}$ will be given by:

$$z[u_1, u_2, d''] = b[d''] + \sum_{i=-\frac{H'}{2}}^{\frac{H'}{2}-1} \sum_{j=-\frac{W'}{2}}^{\frac{W'}{2}-1} \sum_{k=0}^{D-1} f[i, j, k, d''] x[u_1 - i, u_2 - j, k] \quad (1.3.28)$$

1.3.4.2 ReLUs

Activation functions, neurons, or non-linearities, are the core of a neural networks expressibility. Historically, they were sigmoid or tanh functions, but these have been replaced recently by the Rectified Linear Unit (ReLU), which has equation $g(x) = \max(0, x)$. A ReLU

1.3.4.3 Pooling

Typically following a convolutional layer (but not strictly), activations are subsampled with max pooling. Pooling adds some invariance to shifts smaller than the pooling size at the cost of information loss. For this reason, small pooling is typically done often 2×2 or 3×3 , and the invariance to larger shifts comes after multiple pooling (and convolutional) layers.

While initial designs of max pooling would do it in non-overlapping regions, AlexNet used 3×3 pooling with stride 2 in their breakthrough design, quoting that it gave them an increase in accuracy of roughly 0.5% and helped prevent their network from ‘overfitting’. More recent networks will typically employ either this or the original 2×2 pooling with stride 2, see Figure 1.7. A review of pooling methods in mishkin_systematic_2016 found them both to perform equally well.

1.3.4.4 Batch Normalization

Batch normalization proposed only very recently in ioffe_batch_2015 is a conceptually simpler technique. Despite that, it has become quite popular and has been found to be very useful. At its core, it is doing what standard normalization is doing, but also introduces two

learnable parameters — scale (γ) and offset (β). (??) becomes:

$$\tilde{z}(u_1, u_2, d) = \gamma \frac{z - E[z]}{\sqrt{\text{Var}[z]}} + \beta \quad (1.3.29)$$

These added parameters make it a *renormalization* scheme, as instead of centring the data around zero with unit variance, it can be centred around an arbitrary value with arbitrary variance. Setting $\gamma = \sqrt{\text{Var}[z]}$ and $\beta = E[z]$, we would get the identity transform. Alternatively, setting $\gamma = 1$ and $\beta = 0$ (the initial conditions for these learnable parameters), we get standard normalization.

The parameters γ and β are learned through backpropagation. As data are usually processed in batches, the gradients for γ and β are calculated per sample, and then averaged over the whole batch.

From Equation 1.3.32, let us briefly use the hat notation to represent the standard normalized input: $\hat{z} = (z - E[z])/\sqrt{\text{Var}[z]}$, then:

$$\begin{aligned} \tilde{z}^{(i)} &= \gamma \hat{z}^{(i)} + \beta \\ \frac{\partial \mathcal{L}}{\partial \gamma} &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}^{(i)}}{\partial \tilde{z}^{(i)}} \cdot \hat{z}^{(i)} \end{aligned} \quad (1.3.30)$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}^{(i)}}{\partial \tilde{z}^{(i)}} \quad (1.3.31)$$

Batch normalization layers are typically placed *between* convolutional layers and non-linearities. I.e., if $Wu + b$ is the output of a convolutional layer, and $z = g(Wu + b)$ is the output of the non-linearity, then with the batch normalization step, we have:

$$\begin{aligned} z &= g(\text{BN}(Wu + b)) \\ &= g(\text{BN}(Wu)) \end{aligned} \quad (1.3.32)$$

Where the bias term was ignored in the convolutional layer, as it can be fully merged with the ‘offset’ parameter β .

This has particular benefit of removing the sensitivity of our network to our initial weight scale, as for scalar a ,

$$\text{BN}(Wu) = \text{BN}((aW)u) \quad (1.3.33)$$

It is also particularly useful for backpropagation, as an increase in weights leads to *smaller* gradients ioffe_batch_2015, making the network far more resilient to the problems of

Figure 1.9: A residual unit. The identity mapping is always present, and the network learns the difference from the identity mapping, $\mathcal{F}(x)$. Taken from he_deep_2015.

vanishing and exploding gradients:

$$\begin{aligned}\frac{\partial BN((aW)u)}{\partial u} &= \frac{\partial BN(Wu)}{\partial u} \\ \frac{\partial BN((aW)u)}{\partial (aW)} &= \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W}\end{aligned}\tag{1.3.34}$$

1.4 Relevant Architectures

1.4.1 LeNet

1.4.2 AlexNet

1.4.3 VGG

1.4.4 Residual Networks

The current state of the art design introduced a clever novel feature called a residual unithe_deep_2015,he_identity_2016. The inspiration for their design came from the difficulties experienced in training deeper networks. Often, adding an extra layer would *decrease* network performance. This is counter-intuitive as the deeper layers could simply learn the identity mapping, and achieve the same performance.

To promote the chance of learning the identity mapping, they define a residual unit, shown in Figure 1.8. If a desired mapping is denoted $\mathcal{H}(x)$, instead of trying to learn this, they instead learn $\mathcal{F}(x) = \mathcal{H}(x) - x$.

1.4.5 old

Convolutional Neural Networks (CNNs) were initially introduced by lecun_backpropagation_1989 in lecun_backpropagation_1989. Due to the difficulty of training and initializing them, they failed to be popular for more than two decades. This changed in 2012, when advancements in pre-training with unsupervised networks bengio_greedy_2007, the use of an improved non-linearity — the Rectified Linear Unit, or ReLU, new regularization methodshinton_improving_2012, and access to more powerful computers in graphics cards, or GPUs, allowed Krizhevsky, Sutskever and Hinton to develop AlexNetkrizhevsky_imagenet_2012. This network nearly halved the previous state of the art’s error rate. Since then, interest in them has expanded very rapidly, and they have been successfully applied to object detection ren_object_2015 and human pose estimation tompson_efficient_2015. It would take a considerable amount of effort to document the details of all the current enhancements and

Figure 1.10: Standard CNN architecture. Taken from lecun_gradient-based_1998

tricks many researches are using to squeeze extra accuracy, so for the purposes of this report we restrict ourselves to their generic design, with some time spent describing some of the more promising enhancements.

We would like to make note of some of the key architectures in the history of CNNs, which we, unfortunately, do not have space to describe:

- Yann LeCun’s LeNet-5 lecun_gradient-based_1998, the state of the art design for postal digit recognition on the MNIST dataset.
- Google’s GoogLeNet szegedy_going_2015 achieved 6.67% top-5 error on ILSVRC2014, introducing the new ‘inception’ architecture, which uses combinations of 1×1 , 3×3 and 5×5 convolutions.
- Oxford’s VGG simonyan_very_2014 — 6.8% and runner up in ILSVRC2014. The VGG design is very similar to AlexNet but was roughly twice as deep. More convolutional layers were used, but with smaller support — only 3×3 . These were often stacked directly on top of each other without a non-linearity in between, to give the effective support of a 5×5 filter.
- Microsoft Research’s ResNet he_deep_2015 achieved 4.5% top-5 error and was the winner of ILSVRC2015. This network we will talk briefly about, as it introduced a very nice novel layer — the residual layer.

Despite the many variations of CNN architectures currently being used, most follow roughly the same recipe (shown in Figure 1.9):

1.4.6 Fully Connected Layers

The convolution, pooling, and activation layers all conceptually form part of the *feature extraction* stage of a CNN. One or more fully connected layers are usually placed after these layers to form the *classifier*. One of the most elegant and indeed most powerful features of CNNs is this seamless connection between the *feature extraction* and *classifier* sub-networks, allowing the backpropagation of gradients through all layers of the entire network.

The fully connected layers in a CNN are the same as those in a classical Neural Network (NN), in that they compute a dot product between their input vector and a weight vector:

$$z_i = \sum_j W_{ij} x_j \quad (1.4.1)$$

The final output of the Fully Connected layer typically has the same number of outputs as the number of classes C in the classification problem.



Figure 1.11: **Importance of phase over magnitude for images.** The phase of the Fourier transform of the first image is combined with the magnitude of the Fourier transform of the second image and reconstructed. Note that the first image has entirely won out and nothing is left visible of the cameraman.

1.5 The Fourier and Wavelet Transforms

Computer vision is an extremely difficult task. Greyscale intensities in an image are not very helpful in understanding what is in that image. Indeed, these values are sensitive to lighting conditions and camera configurations. It would be easy to take two photos of the same scene and get two vectors x_1 and x_2 that have a very large Euclidean distance, but to a human, would represent the same objects. What is most important in an image are the local variations of image intensity. In particular, the location or phase of the waves that make up the image. A simple experiments to demonstrate this is shown in Figure 1.10.

1.5.1 The Fourier Transform

While the Fourier transform is the core of frequency analysis, it is a poor feature descriptor due to the infinite support of its basis functions. If a single pixel changes in the input, this can theoretically change all of the Fourier coefficients. As natural images are generally non-stationary, we need to be able to isolate frequency components in local regions of an image, and not have this property of global dependence.

The Fourier transform does have one nice property, however, in that the magnitude of Fourier coefficients are invariant to global translations, a nuisance variability. We explore this theme more in our review of the Scattering Transform by Mallat et. al. in section 1.7.

1.5.2 The Wavelet Transform

The Wavelet Transform, like the Fourier Transform, can be used to decompose a signal into its frequency components. Unlike the Fourier transform though, these frequency components can be localized in space. The localization being inversely proportional to the frequency of interest which you want to measure. This means that changes in one part of the image will not affect the wavelet coefficients in another part of the image, so long as the distance between the two parts is much larger than the wavelength of the wavelets you are examining.

The corollary of this property is that wavelets can also be localised in frequency.

The Continuous Wavelet Transform (CWT) gives full flexibility on what spatial/frequency resolution is wanted, but is very redundant and computationally expensive to compute, instead the common yardstick for wavelet analysis is the Discrete Wavelet Transform (DWT) which we introduce.

1.5.3 Discrete Wavelet Transform and its Shortcomings

A particularly efficient implementation of the DWT is Mallat's maximally decimated dyadic filter tree, commonly used with an orthonormal basis set. Orthonormal basis sets are non-redundant, which makes them computationally and memory efficient, but they also have several drawbacks. In particular:

- The DWT is sensitive to the zero crossings of its wavelets. We would like singularities in the input to yield large wavelet coefficients, but if they fall at a zero crossing of a wavelet, the output can be small. See Figure 1.11.
- They have poor directional selectivity. As the wavelets are purely real, they have passbands in all four quadrants of the frequency plane. While they can pick out edges aligned with the frequency axis, they do not have admissibility for other orientations. See Figure 1.12.
- They are not shift invariant. In particular, small shifts greatly perturb the wavelet coefficients. Figure 1.11 shows this for the centre-left and centre-right images. Figure 1.20 (right) also shows this.

The lack of shift invariance and the possibility of low outputs at singularities is a price to pay for the critically sampled property of the transform. Indeed, this shortcoming can be overcome with the undecimated DWT `mallat_wavelet_1998,coifman_translation-invariant_1995`, but it pays a heavy price for this both computationally, and in the number of wavelet coefficients it produces, particularly if many scales J are used.

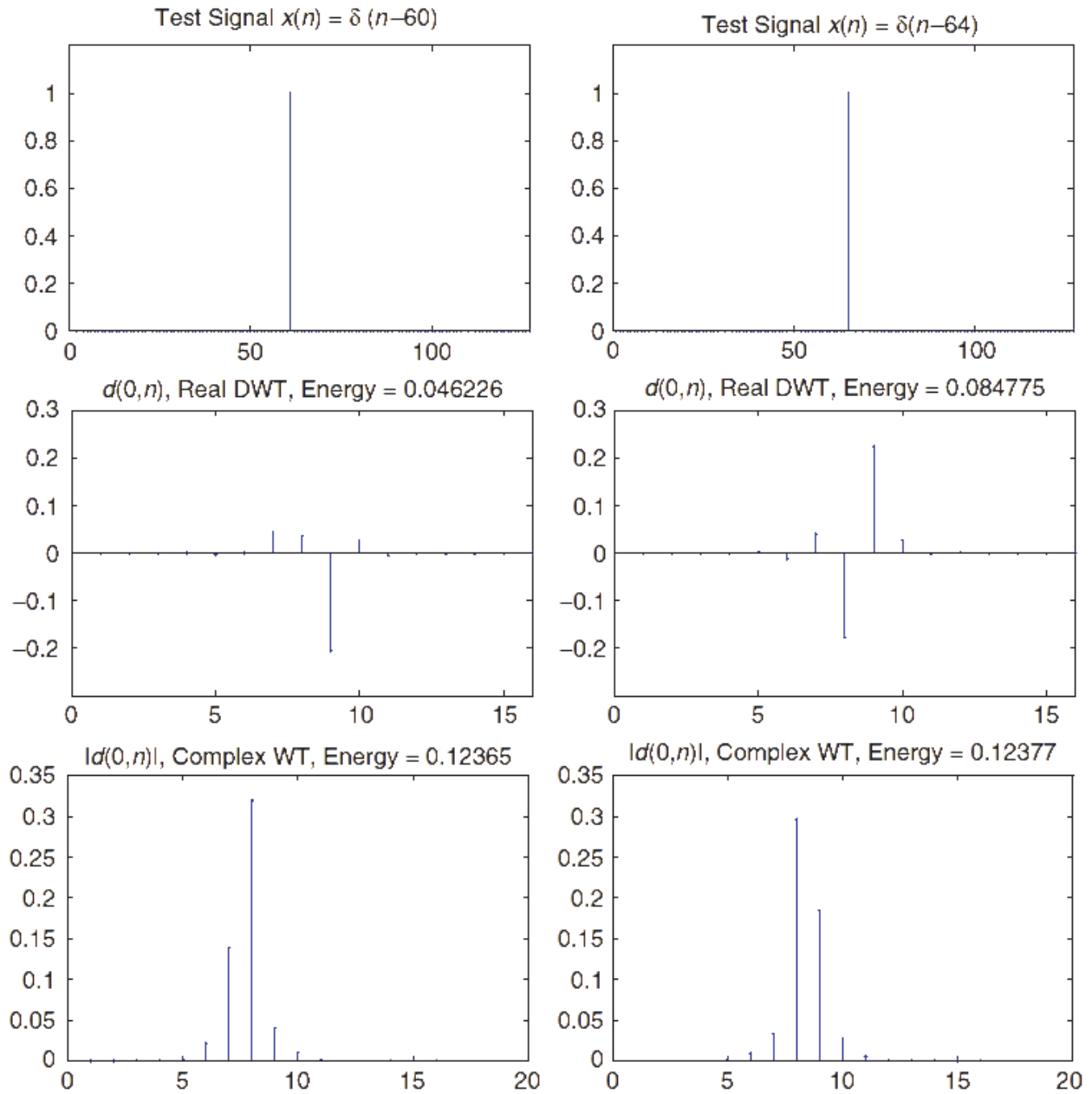


Figure 1.12: **Sensitivity of DWT coefficients to zero crossings and small shifts.** Two impulse signals $\delta(n-60)$ and $\delta(n-64)$ are shown (top), as well as the wavelet coefficients for scale $j=1$ for the DWT (middle) and for the DTCWT (bottom). In the middle row, not only are the coefficients very different from a shifted input, but the energy has almost doubled. As the DWT is an orthonormal transform, this means that this extra energy has come from other scales. In comparison, the energy of the magnitude of the DTCWT coefficients has remained far more constant, as has the shape of the envelope of the output. Image taken from selesnick_dual-tree_2005.

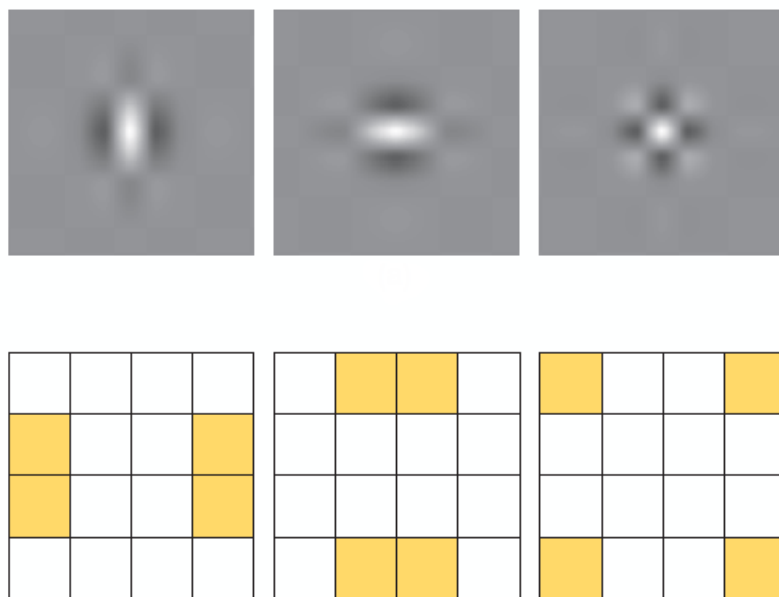


Figure 1.13: **Typical wavelets from the 2D separable DWT.** Top: Wavelet point spread functions for the low-high, high-low, and high-high wavelets. High-high wavelets are in a checkerboard pattern, with no favoured orientation. Bottom: Idealized support of the spectra of each of the wavelets. Image taken from selesnick_dual-tree_2005.

1.5.4 Complex Wavelets

So the DWT has overcome the problem of space-frequency localisation, but it has introduced new problems. Fortunately, we can improve on the DWT with complex wavelets, as they can solve these new shortcomings while maintaining the desired localization properties.

The Fourier transform does not suffer from a lack of directional selectivity and shift invariance because its basis functions are based on the complex sinusoid:

$$e^{j\omega t} = \cos(\omega t) + j \sin(\omega t) \quad (1.5.1)$$

whereas the DWT's basis functions are based on only the real sinusoid $\cos(\omega t)$ ². As t moves along the real line, the phase of the Fourier coefficients change linearly, while their magnitude remains constant. In contrast, as t moves along the real line, the sign of the real coefficient oscillates between -1 and 1, and its magnitude changes in a non-linear way.

These nice properties come from the fact that the cosine and sine functions of the Fourier transform form a Hilbert Pair, and so together they constitute an analytic signal.

We can achieve these nice properties if the mother wavelet for our wavelet transform is analytic:

$$\psi_c(t) = \psi_r(t) + j\psi_s(t) \quad (1.5.2)$$

where $\psi_r(t)$ and $\psi_s(t)$ form a Hilbert Pair (i.e., they are 90° out of phase with each other).

There are a number of possible ways to do a wavelet transform with complex wavelets. We examine two in particular, the Fourier-based method used by Mallat et. al. in their scattering transform [bruna_classification_2011](#), [bruna_invariant_2013](#), [bruna_scattering_2013](#), [oyallon_generic_2013](#), [oyallon_deep_2015](#), [sifre_rotation_2013](#), [sifre_rigid-motion_2014](#), [sifre_rigid-motion_2014-1](#), [sifre_scatnet_2013](#), and the separable, filter bank based DTCWT developed by Kingsbury [kingsbury_wavelet_1997](#), [kingsbury_dual-tree_1998](#), [kingsbury_dual-tree_1998-1](#), [kingsbury_image_1999](#), [kingsbury_shift_1999](#), [kingsbury_dual-tree_2000](#), [kingsbury_complex_2001](#), [selesnick_dual-tree_2005](#).

1.5.5 Fourier Based Wavelet Transform

The Fourier Based method used by Mallat et. al. is an efficient implementation of the Gabor Transform. Mallat tends to prefer to use a close relative of the Gabor wavelet — the Morlet wavelet — as the mother wavelet for his transform.

While the Gabor wavelets have the best theoretical trade-off between spatial and frequency localization, they have a non-zero mean. This makes the wavelet coefficients non-sparse, as they will all have a DC component to them, and makes them inadmissible as wavelets. Instead, the Morlet wavelet has the same shape, but with an extra degree of freedom chosen

²we have temporarily switched to 1D notation here as it is clearer and easier to use, but the results still hold for 2D

to set $\int \psi(\mathbf{u}) d\mathbf{u} = 0$. This wavelet has equation (in 2D):

$$\psi(\mathbf{u}) = \frac{1}{2\pi\sigma^2} (e^{i\mathbf{u}\xi} - \beta) e^{-\frac{|\mathbf{u}|^2}{2\sigma^2}} \quad (1.5.3)$$

where β is usually $\ll 1$ and is this extra degree of freedom, σ is the size of the gaussian window, and ξ is the location of the peak frequency response — i.e., for an octave based transform, $\xi = 3\pi/4$.

Mallat et. al. add an extra degree of freedom to this by allowing for a non-circular gaussian window over the complex sinusoid, which gives control over the angular resolution of the final wavelet, so this now becomes:

$$\psi(\mathbf{u}) = \frac{\gamma}{2\pi\sigma^2} (e^{i\mathbf{u}\xi} - \beta) e^{-\mathbf{u}^t \Sigma^{-1} \mathbf{u}} \quad (1.5.4)$$

Where

$$\Sigma^{-1} = \begin{bmatrix} \frac{1}{2\sigma^2} & 0 \\ 0 & \frac{\gamma^2}{2\sigma^2} \end{bmatrix}$$

The effects of modifying the eccentricity parameter γ and the window size σ are shown in Figure 1.13. To have a full two dimensional wavelet transform, we need to rotate this mother wavelet by angle θ and scale it by j/Q , where Q is the number of scales per octave (usually 1 in image processing). This can be done by doing the following substitutions in (1.5.4):

$$\begin{aligned} R_\theta &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \\ \mathbf{u}_\theta &= R_{-\theta} \mathbf{u} \\ \sigma_j &= 2^{\frac{j-1}{Q}} \sigma \\ \xi_j &= \frac{\xi}{2^{\frac{j-1}{Q}}} \end{aligned}$$

Mallat et. al. combine these two variables into a single coordinate

$$\lambda = (\theta, j/Q) \quad (1.5.5)$$

Returning to the higher level notation, we can write the Morlet wavelet as the sum of real and imaginary parts: And scaled and rotated wavelets as:

$$\psi_\lambda(\mathbf{u}) = 2^{-j/Q} \psi(2^{-j/Q} R_\theta^{-1} \mathbf{u}) \quad (1.5.6)$$



Figure 1.14: **Single Morlet filter with varying slants and window sizes.** Top left — 45° plane wave (real part only). Top right — plane wave with $\sigma = 3, \gamma = 1$. Bottom left — plane wave with $\sigma = 3, \gamma = 0.5$. Bottom right — plane wave with $\sigma = 2, \gamma = 0.5$.

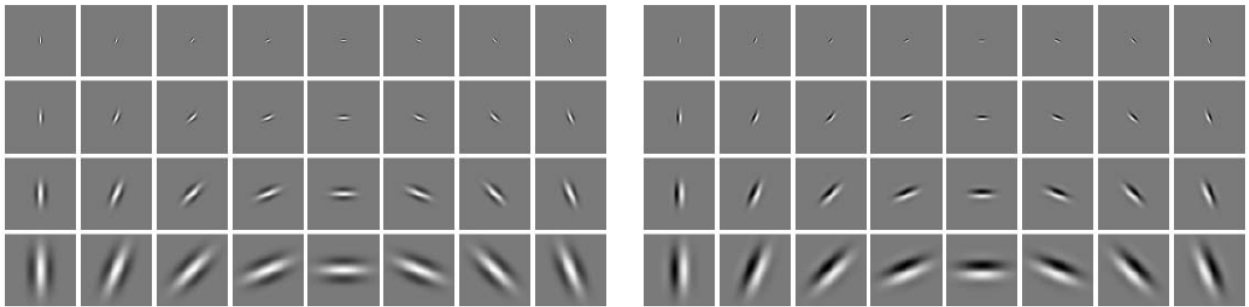


Figure 1.15: **The full dictionary of Morlet wavelets used by Mallat.** The real filters are on the left and the imaginary on the right. The first row correspond to scale $j = 1$, increasing up to $j = 4$. The first column corresponding to $\theta = 0$, rotating through $\pi/8$ up to the eighth column of $7\pi/8$, $\gamma = 1/2$.

1.5.5.1 Implementation

The Fourier Implementation of this Morlet decomposition is shown in Figure 1.15. It is based on the fact that

$$\mathcal{F}(x * \psi)(\omega) = \mathcal{F}x(\omega)\mathcal{F}\psi(\omega) \quad (1.5.7)$$

so to compute the family of outputs of $x * \psi_\lambda$, we can precompute the Fourier transform of all of the wavelets, then at run time, take the Fourier transform of the image, x , multiply with the Fourier transform of the wavelets, and then take the inverse Fourier transform of the product. The output scale can be chosen by periodizing the product of the Fourier transforms, and then compute the inverse Fourier transform at the reduced resolution.

The resulting complexity of the entire operation for an image with $N \times N$ pixels is:

- $O(N^2 \log N)$ for the forward FFT of x .
- $O(JLN^2)$ for the multiplication in the frequency domain. We can see this from Figure 1.15, there are J scales to do multiplication at, and each scale has L orientations, except for the low-low.
- $O(L \sum_j (2^{-2j} N^2) \log 2^{-2j} N)$ for the inverse FFTs. The term inside the sum is just the $O(N^2 \log N)$ term of an inverse FFT that has been downsampled by 2^j in each direction.

And altogether:

$$T(N) = O(N^2 \log N) + O(JLN^2) + O(L \sum_j N^2 \log 2^{-j} N) \quad (1.5.8)$$

1.5.5.2 Invertibility and Energy Conservation

We can write the wavelet transform of an input x as

$$\mathcal{W}x = \{x * \phi_J, x * \psi_\lambda\}_\lambda \quad (1.5.9)$$

The ℓ^2 norm of the wavelet transform is then defined by

$$\|\mathcal{W}x\|^2 = \|x * \phi_J\|^2 + \sum_\lambda \|x * \psi_\lambda\|^2 \quad (1.5.10)$$

An energy preserving transform will satisfy Plancherel's equality, so that

$$\|\mathcal{W}x\| = \|x\| \quad (1.5.11)$$

which is a nice property to have for invertibility, as well as for analysing how different signals get transformed (e.g. white noise versus standard images).

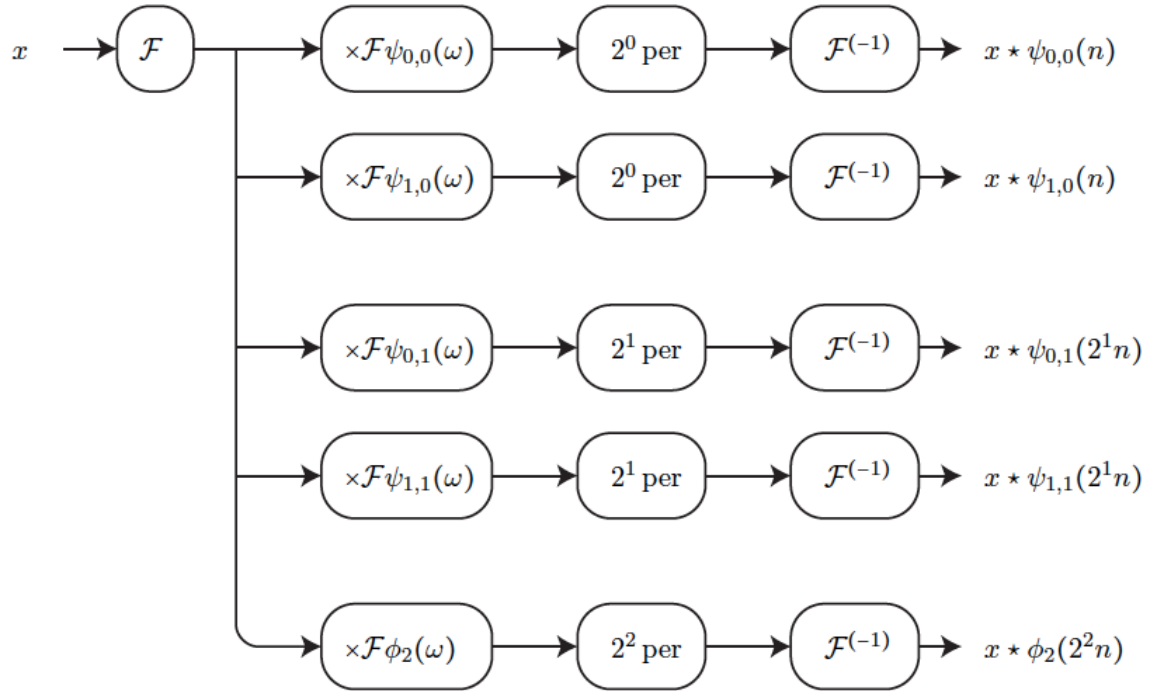


Figure 1.16: Fourier Implementation of the Morlet decomposition of an input image, with $J = 2$ scales and $L = 2$ orientations. The Fourier transform of x is calculated and multiplied with the (precomputed) Fourier transforms of a bank of Morlet filters. The results are periodized according to the target resolution, and then the inverse Fourier transform is applied. Image taken from `sifre_rigid-motion_2014-1`.

Figure 1.17: Three Morlet Wavelet Families and their tiling of the frequency plane. For each set of parameters, the point spread functions of the wavelet bases are shown, next to their Littlewood-Paley sum $A(\omega)$. None of the configurations cover the corners of the frequency plane, and they often exceed 1. Increasing J , L (Sifre uses C in these diagrams) or Q gives better frequency localization but at the cost of spatial localization and added complexity. Image taken from `sifre_rigid-motion_2014-1`.

For a transform to be invertible, we examine the measure of how tightly its basis functions tile the Fourier plane with its Littlewood-Paley function:

$$A(\omega) = |\mathcal{F}\phi_J(\omega)|^2 + \sum_{\lambda} |\mathcal{F}\psi_{\lambda}(\omega)|^2 \quad (1.5.12)$$

If the tiling is α -tight, then $\forall \omega \in \mathbb{R}^2$:

$$1 - \alpha \leq A(\omega) \leq 1 \quad (1.5.13)$$

and the wavelet operator, \mathcal{W} is an α frame. If $A(\omega)$ is ever close to 0, then there is not a good coverage of the frequency plane at that location. If it ever exceeds 1, then there is overlap between bases. Both of these conditions make invertibility difficult³. Figure 1.16 show the invertibility of a few families of wavelets used by Mallat et. al.. Invertibility is possible, but not guaranteed for all configurations. The Fourier transform of the inverse filters are defined by:

$$\mathcal{F}\phi_J^{-1}(\omega) = A(\omega)^{-1} \mathcal{F}\phi_J(\omega) \quad (1.5.14)$$

$$\mathcal{F}\psi_{\lambda}^{-1}(\omega) = A(\omega)^{-1} \mathcal{F}\psi_{\lambda}(\omega) \quad (1.5.15)$$

1.5.6 The DTCWT

The DTCWT was first proposed by Kingsbury in [20], [21] as a way to combat many of the shortcomings of the DWT, in particular, its poor directional selectivity, and its poor shift invariance. A thorough analysis of the properties and benefits of the DTCWT is done in [22], [23]. Building on these properties, it been used successfully for denoising and inverse problems [24]–[27], texture classification [28], [29], image registration [30], [31] and SIFT-style keypoint generation matching [32]–[36] amongst many other applications.

Compared to Gabor (or Morlet) image analysis, the authors of [23] sum up the dangers as:

³In practise, if $A(\omega)$ is only slightly greater 1 for only a few small areas of ω , approximate inversion can be achieved

Figure 1.18: Analysis FB for the DTCWT@. Top ‘tree’ forms the real component of the complex wavelet ψ_r , and the bottom tree forms the imaginary (Hilbert pair) component ψ_i . Image taken from selesnick_dual-tree_2005.

A typical Gabor image analysis is either expensive to compute, is noninvertible, or both.

This nicely summarises the difference between this method and the Fourier based method outlined in subsection 1.5.5. The DTCWT is a filter bank (FB) based wavelet transform. It is faster to implement than the Morlet analysis, as well as being more readily invertible.

1.5.6.1 Design Criteria for the DTCWT

It was stated in subsection 1.5.4 that if the mother (and daughter) wavelets were complex, with their real and imaginary parts forming a Hilbert pair, then the wavelet transform of a signal with these $\{\psi_{j,n}\}_{j,n}$ would give a representation that had nice shift properties⁴, was insensitive to zero crossings of the wavelet, and had good directional selectivity.

As in subsection 1.5.4, we want to have a complex mother wavelet ψ_c that satisfies (1.5.2), but now achieved with filter banks. A slight deviation from standard filter bank notation, where h_0, h_1 are the analysis and g_0, g_1 are the synthesis filters. We define:

- h_0, h_1 the low and high-pass analysis filters for ψ_r (henceforth called ψ_h)
- g_0, g_1 the low and high-pass analysis filters for ψ_i (henceforth called ψ_g)
- \tilde{h}_0, \tilde{h}_1 the low and high-pass synthesis filters for $\tilde{\psi}_h$.
- \tilde{g}_0, \tilde{g}_1 the low and high pass synthesis filters for $\tilde{\psi}_g$.

The dilation and wavelet equations for a 1D filter bank implementation are:

$$\phi_h(t) = \sqrt{2} \sum_n h_0(n) \phi_h(2t - n) \quad (1.5.16)$$

$$\psi_h(t) = \sqrt{2} \sum_n h_1(n) \phi_h(2t - n) \quad (1.5.17)$$

$$\phi_g(t) = \sqrt{2} \sum_n g_0(n) \phi_g(2t - n) \quad (1.5.18)$$

$$\psi_g(t) = \sqrt{2} \sum_n g_1(n) \phi_g(2t - n) \quad (1.5.19)$$

This implementation is shown in Figure 1.17.

Designing a filter bank implementation that results in Hilbert symmetric wavelets does not appear to be an easy task. However, it was shown by kingsbury_image_1999 (and later

⁴in particular, that a shift in input gives the same shift in magnitude of the wavelet coefficients, and a linear phase shift

proved by selesnick_hilbert_2001) that the necessary conditions are conceptually very simple. One low-pass filter must be a *half-sample shift* of the other. I.e.,

$$g_0(n) \approx h_0(n - 0.5) \rightarrow \psi_g(t) \approx \mathcal{H}\{\psi_h(t)\} \quad (1.5.20)$$

As the DTCWT is designed as an invertible filter bank implementation, this is only one of the constraints. Naturally, there are also:

- Perfect reconstruction
- Finite support
- Linear phase
- Many vanishing moments at $z = -1$ for good stopband properties

to consider when building the h 's and g 's. The derivation of the filters that meet these conditions is covered in detail in kingsbury_complex_2001, kingsbury_design_2003, and in general in selesnick_dual-tree_2005. The result is the option of three families of filters: biorthogonal filters ($h_0[n] = h_0[N - 1 - n]$ and $g_0[n] = g_0[N - n]$), q-shift filters ($g_0[n] = h_0[N - 1 - n]$), and common-factor filters.

1.5.6.2 The Resulting Wavelets and their Properties

While analytic wavelets in 1D are useful for their shift invariance, the real beauty of the DTCWT is in its ability to make a separable 2D wavelet transform with oriented wavelets.

1.18a shows the spectrum of the wavelet when the separable product uses purely real wavelets, as is the case with the DWT. 1.18b however, shows the separable product of two complex, analytic wavelets resulting in a localized and oriented 2D wavelet.

I.e., for the $+45^\circ$ wavelet⁵ (which is high in both ω_1 and ω_2), the separable product is:

$$\psi(\omega_1, \omega_2) = \psi_c(\omega_1) \overline{\psi_c(\omega_2)} \quad (1.5.21)$$

$$\begin{aligned} &= (\psi_h(\omega_1) + j\psi_g(\omega_1)) \overline{(\psi_h(\omega_2) + j\psi_g(\omega_2))} \\ &= \psi_h(\omega_1)\psi_h(\omega_2) + \psi_g(\omega_1)\psi_g(\omega_2) \\ &\quad + j(\psi_g(\omega_1)\psi_h(\omega_2) - \psi_h(\omega_1)\psi_g(\omega_2)) \end{aligned} \quad (1.5.22)$$

Similar equations can be obtained for the other five wavelets and the scaling function, by replacing ψ with ϕ for both directions, and not taking the complex conjugate in (1.5.21) to get the right hand side of the frequency plane.

Figure 1.19 shows the resulting wavelets both in the spatial domain and their idealized support in the frequency domain.

Figure 1.20 shows how the DTCWT compares with the DWT with a shifting input.

⁵note that 1.18b shows the 135° wavelet

(a)

(b)

Figure 1.19: (a) The high-high DWT wavelet having a passband in all 4 corners of the frequency plane vs (b) the high-high DTCWT wavelet frequency support only existing in one quadrant. Taken from selesnick_dual-tree_2005

Figure 1.20: Wavelets from the 2d DTCWT@. **Top:** The six oriented filters in the space domain (only the real wavelets are shown). **Bottom:** Idealized support of the Fourier spectrum of each wavelet in the 2D frequency plane. Spectra of the the real wavelets are shown — the spectra of the complex wavelets ($\psi_h + j\psi_g$) only has support in the top half of the plane. Image taken from selesnick_dual-tree_2005.

1.5.6.3 Implementation and Efficiency

Figure 1.17 showed the layout for the DTCWT for 1D signals. We saw from (1.5.22) that the 2D separable product of wavelets involved the product of ψ_g , ψ_h , ϕ_g , and ϕ_h terms, with some summing and differencing operations. Figure 1.21 shows how to efficiently implement this with FBs.

As we did for subsection 1.5.5.1, we calculate and compare the complexity of the DTCWT@. To do this, we must know the length of our h and g filters. It is also important to know that we must use different filters for the first scale to the deeper scales, as this achieves better analyticity. A typical configuration will use biorthogonal filters for the first scale, then qshift for subsequent scales. These filters have the following number of taps⁶:

| | h_0 | h_1 | g_0 | g_1 |
|--------------|-------|-------|-------|-------|
| biorthogonal | 5 | 7 | 7 | 5 |
| qshift | 10 | 10 | 10 | 10 |

The resulting complexity of the entire forward wavelet transform for an image with $N \times N$ pixels is:

- First layer (Image size = $N \times N$):
 - Column filtering requires $5N^2 + 7N^2$ multiply-adds
 - Row filtering to make the LoLo term requires $5N^2$ more multiply-adds

⁶This implementation uses the shorter near_sym_a biorthogonal filters and qshift_a filters. Smoother wavelets can have slightly more taps

Figure 1.21: The shift invariance of the DTCWT (left) vs. the real DWT (right). The DTCWT linearizes shifts in the phase change of the complex wavelet. Image taken from kingsbury_dual-tree_1998.

Figure 1.22: The filter bank implementation of the DTCWT@. Image taken from kingsbury_image_1999

- Row filtering to make 15° and 165° requires $5N^2 + 4N^2$ multiply-adds (the 4 here comes from the Σ/Δ function block in Figure 1.21).
- Row filtering to make the 45° and 135° requires $7N^2 + 4N^2$ multiply-adds
- Row filtering to make the 75° and 105° requires $7N^2 + 4N^2$ multiply-adds

The total being

$$T(N) = (5 + 7 + 5 + 5 + 4 + 7 + 4 + 7 + 4)N^2 = 48N^2$$

- Second and deeper layers (Image size = $2^{-j}N \times 2^{-j}N = M \times M$):
 - Column filtering requires $10M^2 + 10M^2$ multiply-adds
 - Row filtering to make the LoLo term requires $10M^2$ more multiply-adds
 - Row filtering to make 15° and 165° requires $10M^2 + 4M^2$ multiply-adds (the 4 here comes from the Σ/Δ function block in Figure 1.21).
 - Row filtering to make the 45° and 135° requires $10M^2 + 4M^2$ multiply-adds
 - Row filtering to make the 75° and 105° requires $10M^2 + 4M^2$ multiply-adds

The total being:

$$T(M) = (10 + 10 + 10 + 10 + 4 + 10 + 4 + 10 + 4)M^2 = 68M^2 = 68 \times 2^{-2j}N^2$$

$$T(N) = 48N^2 + \sum_{j=1}^J 68 \times 2^{-2j}N^2 \approx 100N^2 \quad (1.5.23)$$

As the term $\sum_{j=1}^J 68 \times 2^{-j}$ is a geometric series that sums to $1/3$ as $j \rightarrow \infty$. We have also rounded up the sum to be conservative.

It is difficult to compare this with the complexity of the Fourier-based implementation of the Morlet wavelet transform we have derived in subsubsection 1.5.5.1, as we cannot readily estimate the big-O order constants for the 2D FFT method. However, the central term, JLN^2 , would cost $24N^2$ multiplies for four scales and six orientations. These are complex multiplies, as they are in the Fourier domain, which requires four real multiplies. On top of this, to account for the periodic repetition of the inverse FFT implementation, Mallat et. al. symmetrically extend the image by $N/2$ in each direction. This means that the central term is already on the order of $\sim 200N^2$ multiplies, without even considering the more expensive forward and inverse FFTs.

Figure 1.23: Four scales of the DTCWT (left) and its associated frequency coverage, or $A(\omega)$ (right). Note the reduced scale compared to Figure 1.16.

Figure 1.24: Two adversarial examples generated for AlexNet. The left column shows a correctly predicted sample, the right column an incorrectly predicted example, and the centre column the difference between the two images, magnified 10 times. Image taken from [szegedy_intriguing_2013](#).

For a simple comparison experiment, we performed the two transforms on a four core Intel i7 processor (a moderately high-end personal computer). The DTCWT transform took roughly 0.15s on a black and white 32×32 image, vs. 0.5s for the Fourier-based method. For a larger, 512×512 image, the DTCWT implementation took 0.35s vs. 3.5s (times were averaged over several runs).

1.5.6.4 Invertibility and Energy Conservation

We analysed the Littlewood-Paley function for the Morlet-Fourier implementation, and saw what areas of the spectrum were better covered than others. How about for the DTCWT@?

It is important to note that in the case of the DTCWT, the wavelet transform is also approximately unitary, i.e.,

$$\|x\|^2 \approx \|\mathcal{W}x\|^2 \quad (1.5.24)$$

and the implementation is perfectly invertible as the Littlewood-Paley function is unity (or very near unity) $\forall \omega$. See Figure 1.22. This is not a surprise, as it is a design constraint in choosing the filters, but nonetheless is important to note.

A beneficial property of energy conservation is that the noise in the input will equal the noise in the wavelet coefficients. When we introduce Scatternets, we can show that we can keep the unitary property in the scattering coefficients. This is an important property, particularly in light of the recent investigations in [szegedy_intriguing_2013](#). This paper saw that it is easy to find cases in CNNs where a small amount of input perturbation results in a completely different class label (see Figure 1.23). Having a unitary transform limits the amount the features can change, which will make the entire network more stable to distortion and noise.

1.5.7 Summary of Methods

One final comparison to make between the DTCWT and the Morlet wavelets is their frequency coverage. The Morlet wavelets can be made to be tighter than the DTCWT, which gives better angular resolution — see Figure 1.24. However it is not always better to keep getting finer

(a) DT-CWT wavelets (left to right) — 15° , 45° and 75°

(c) Morlet wavelets (left to right) — 0° , 22.5° , 45° , 67.5° , and 90°

Figure 1.25: Normalized Energy spectra of the DT-CWT wavelets versus the preferred 8 orientation Morlet wavelets by Mallat for the second quadrant. Orientations listed refer to the edge orientation in the spatial domain that gives the highest response. All wavelets have been normalized to be between zero and one. The Morlet wavelets have finer angular resolution, which can give better discrimination, at the cost of decreasing stability to deformations, and requiring larger spatial support.

and finer resolutions, indeed the Fourier transform gives the ultimate in angular resolution, but as mentioned, this makes it less stable to shifts and deformations.

?? compares the advantages and disadvantages of the wavelet methods discussed in this chapter.

1.6 Shift Invariance of the DT-CWT

Firstly, let us look at what would happen if we retained only 1 of the subbands. Note that we have to keep the same band from each tree. For any pair of coefficients on the tree, this would look like Figure 1.26. E.g. if we kept x_{001a} and x_{001b} then $M = 8$ and $A(z) = H_{0a}(z)H_{00a}(z^2)H_{001a}(z^4)$ is the transfer function from x to x_{001a} . The transfer functions for $B(z)$, $C(z)$ and $D(z)$ are obtained similarly. It is well known that:

$$U(z) \downarrow M \rightarrow U(z^M) \quad (1.6.1)$$

$$U(z) \uparrow M \rightarrow \frac{1}{M} \sum_{k=0}^{M-1} U(W^k z^{1/M}) \quad (1.6.2)$$

Where $W = e^{j2\pi/M}$. So downsampling followed by upsampling becomes:

$$U(z) \downarrow M \uparrow M \rightarrow \frac{1}{M} \sum_{k=0}^{M-1} U(W^k z)$$

This means that

$$Y(z) = Y_a(z) + Y_b(z) = \frac{1}{M} \sum_{k=0}^{M-1} X(W^k z) [A(W^k z)C(z) + B(W^k z)D(z)] \quad (1.6.3)$$

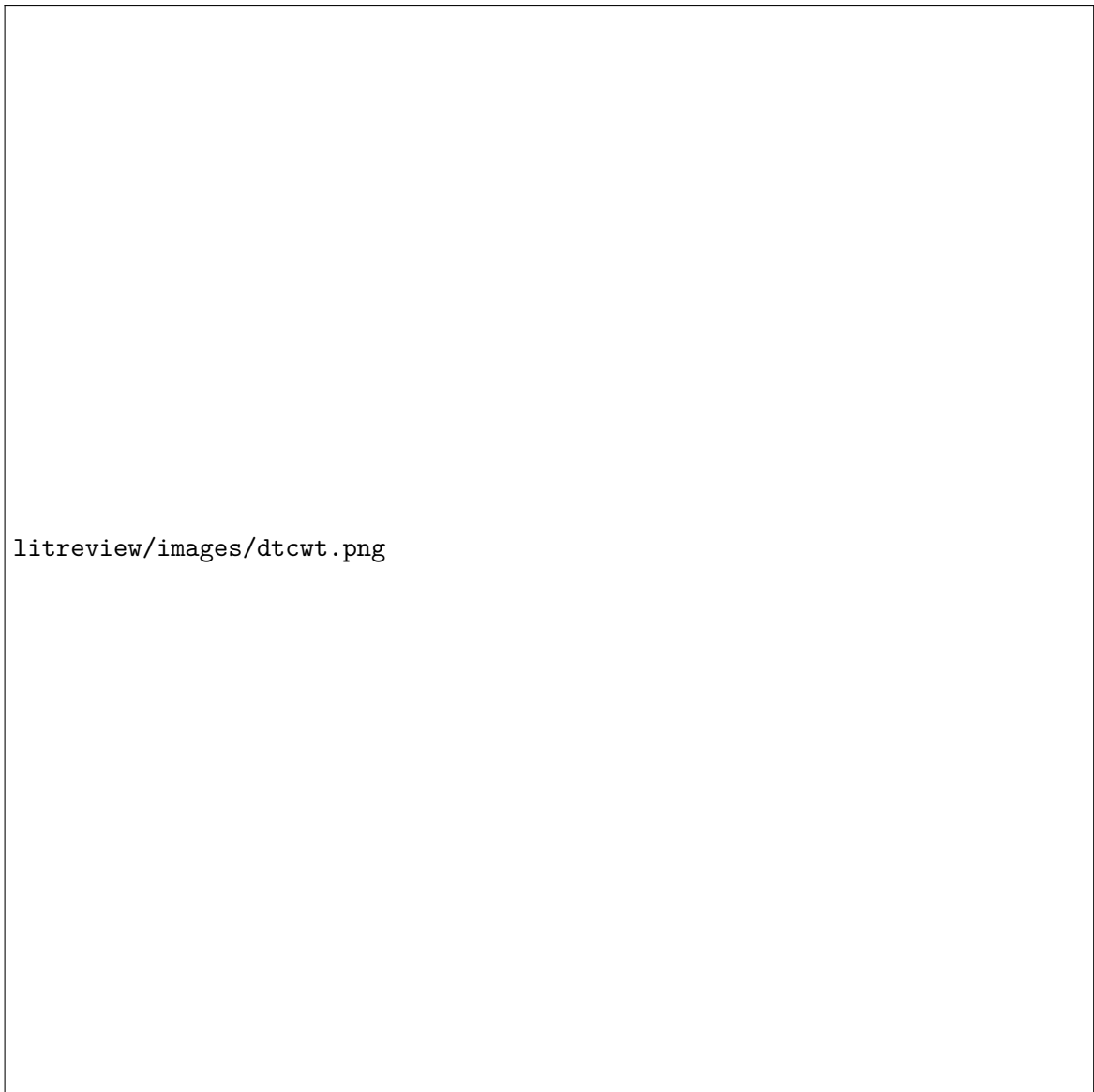


Figure 1.26: **Full 1-D DTCWT.**

Figure 1.27: **Block Diagram of 1-D DTCWT.** Note the top and bottom paths are through the wavelet or scaling functions from just level m ($M = 2^m$). Figure based on Figure 4 in [37].

litreview/images/overlaps.png

The aliasing terms for which are everywhere where $k \neq 0$ (as $X(W^k z)$ is $X(z)$ shifted by $\frac{2k\pi}{M}$). I.e. to avoid aliasing in this reduced tree, we want to make sure that $A(W^k z)C(z) + B(W^k z)D(z) = 0$ for all $k \neq 0$.

The figure below (Fig 5 from [37]) shows what $A(W^k z)$ and $C(z)$ look like for both the lowpass case (left) and the highpass case (right). Note that these responses will be similar for $B(W^k z)$ and $D(z)$. For large values of k , there is almost no overlap (i.e. $A(W^k z)C(z) \approx B(W^k z)D(z) \approx 0$, but for small values of k (particularly $k = \pm 1$), the transition bands have significant overlap with the central response. It is here that we need to use the flexibility of having 2 trees to ensure that $A(W^k z)C(z)$ and $B(W^k z)D(z)$ cancel out.

To do this, let us consider the lowpass filters first. If we let:

$$B(z) = z^{\pm M/2} A(z) \quad (1.6.4)$$

$$D(z) = z^{\mp M/2} C(z) \quad (1.6.5)$$

Then

$$A(W^k z)C(z) + B(W^k z)D(z) = A(W^k z)C(z) + (W^k z)^{\pm M/2} A(W^k z) z^{\mp M/2} C(z) \quad (1.6.6)$$

$$= A(W^k z)C(z) + e^{j \frac{k 2\pi}{M} \times (\pm \frac{M}{2})} z^{\pm M/2} z^{\mp M/2} A(W^k z)C(z) \quad (1.6.7)$$

$$= A(W^k z)C(z) + (-1)^k A(W^k z)C(z) \quad (1.6.8)$$

Which cancels when k is odd

Now consider the bandpass case. For shifts of $k = 1$ the right half of the left peak overlaps with the left half of the right peak. For a shift of $k = 2$, the left half of the left peak overlaps with the right half of the right peak. Similarly for $k = -1$ and $k = -2$. For $|k| > 2$, there is no overlap. The fact that we have overlaps at both even and odd shifts of k means that we can't use the same clever trick from before. However, what we do note is that the overlap is always caused by opposite peaks (i.e. the left with the right peak, and never the left with itself, or the right with itself). The solution then is to have B and D have upper and lower passbands of opposite polarity, while A and C should have passbands of the same polarity.

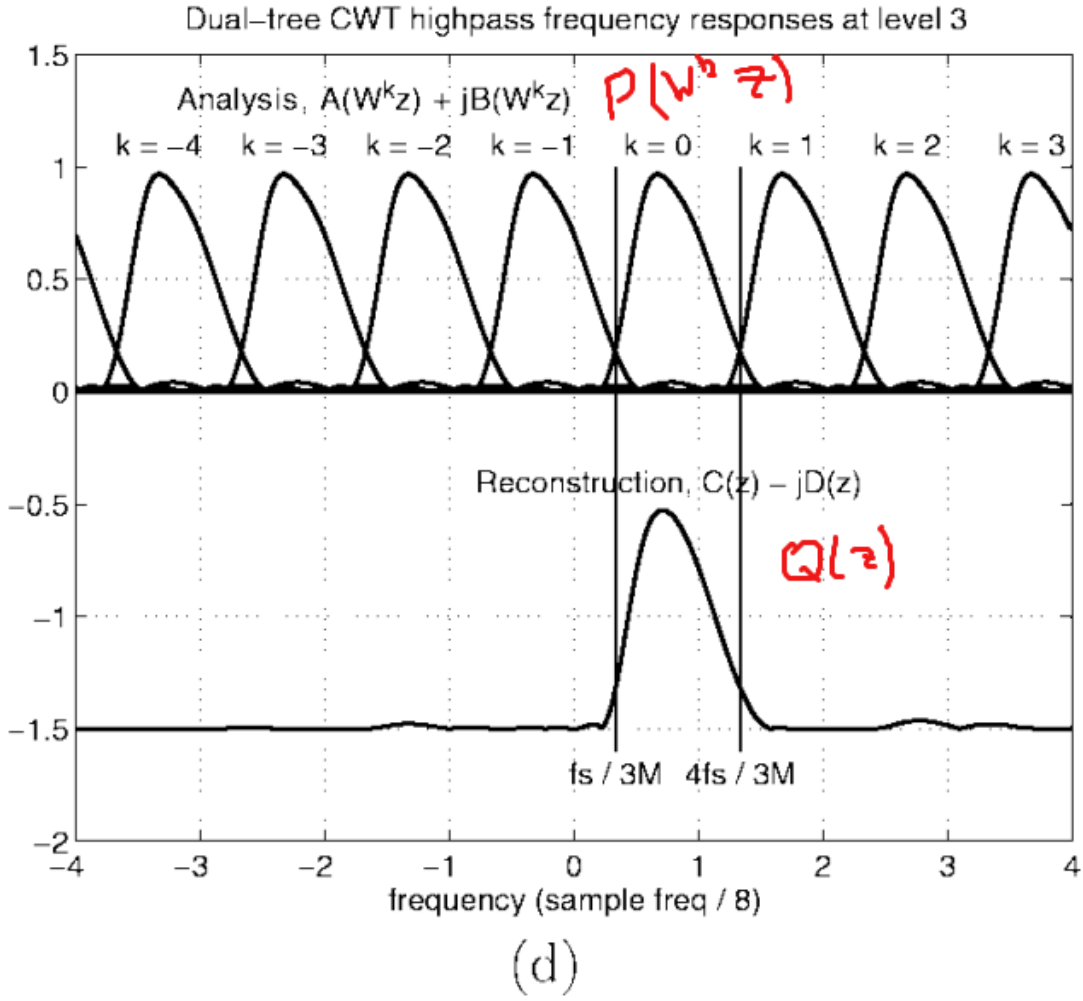
Consider two prototype complex filters $P(z)$ and $Q(z)$ each with single passbands going from $f_s/2M \rightarrow f_s/M$ (or $\frac{\pi}{M} \rightarrow \frac{2\pi}{M}$) - they must be complex to have support in only one half of the frequency plane. Now say $P^*(z) = \sum_r p_r^* z^{-r}$ is the z -transform of the conjugate of p_r , which has support only in the negative half of the frequency plane. Then we can get the required filters by:

$$A(z) = 2\Re[P(z)] = P(z) + P^*(z) \quad (1.6.9)$$

$$B(z) = 2\Im[P(z)] = -j[P(z) - P^*(z)] \quad (1.6.10)$$

$$C(z) = 2\Re[Q(z)] = Q(z) + Q^*(z) \quad (1.6.11)$$

$$D(z) = -2\Im[Q(z)] = j[Q(z) - Q^*(z)] \quad (1.6.12)$$



Then:

$$\begin{aligned}
 A(W^k z)C(z) + B(W^k z)D(z) &= [P(W^k z) + P^*(W^k z)][Q(z) + Q^*(z)] + \\
 &\quad (-j * j)[P(W^k z) - P^*(W^k z)][Q(z) - Q^*(z)] \quad (1.6.13) \\
 &= P(W^k z)Q(z)[1 + 1] + P^*(W^k z)Q(z)[1 - 1] + \\
 &\quad P(W^k z)Q^*(z)[1 - 1] + P^*(W^k z)Q^*(z)[1 + 1] \quad (1.6.14) \\
 &= 2P(W^k z)Q(z) + 2P^*(W^k z)Q^*(z) \quad (1.6.15)
 \end{aligned}$$

So now we only need to ensure that $P(W^k z)$ overlaps as little as possible with $Q(z)$. This is somewhat more manageable, the diagram below shows the problem.

1.7 Scatternets

Scatternets have been a very large influence on our work, as well as being quite distinct from the previous discussions on learned methods. They were first introduced by Bruna and Mallat in their work [bruna_classification_2011](#), and then were rigorously defined by Mallat in [mallat_group_2012](#). Several updates and newer models have since been released by Mallat’s group, which we will review in this chapter.

It is helpful to introduce this chapter with one further note. Unlike the CNNs introduced in ??, which were set up to minimize some cost function which had certain constraints to promote certain properties, the scattering operator may be thought of as an operator Φ which has some desirable properties for image understanding. These properties may ultimately help us minimize some cost function and improve our image understanding system, which we explore more in ??.

1.8 Translation Invariant Scatternets

The translation invariant Scatternets were mostly covered in [bruna_invariant_2013](#). This section summarises the method of this paper.

1.8.1 Defining the Properties

The first release of Scatternets aimed at building a translation invariant operator, which was also stable to additive noise and deformations. Translation is often defined as being uninformative for classification — an object appearing in the centre of the image should be treated the same way as an the same object appearing in the corner of an image, i.e., Φx is invariant to translations $x_c(\mathbf{u}) = x(\mathbf{u} - \mathbf{c})^7$ by $\mathbf{c} = (c_1, c_2) \in \mathbb{R}^2$ if

$$\Phi x_c = \Phi x \tag{1.8.1}$$

Stability to additive noise is another good choice to include in this operator, as it is a common feature in measured signals. Stability is defined in terms of Lipschitz continuity, which is a strong form of uniform continuity for functions, which we briefly introduce here.

Formally, a Lipschitz continuous function is limited in how fast it can change; there exists an upper bound on the gradient the function can take, although it doesn’t necessarily need to be differentiable everywhere. The modulus operator $|x|$ is a good example of a function that has a bounded derivative and so is Lipschitz continuous, but isn’t differentiable everywhere.

⁷here we adopt a slight variation on ’s notation, by using boldface letters to represent vectors, as is the custom in Signal Processing

Figure 1.28: A Lipschitz continuous function is shown. There is a cone for this function (shown in white) such that the graph always remains entirely outside the cone as it's shifted across. The minimum gradient needed for this to hold is called the 'best Lipschitz constant'.

Figure 1.29: Real, Imaginary and Modulus of complex wavelet convolved with an impulse.

Returning again to stability to additive noise, state that for a new signal $x'(\mathbf{u}) = x(\mathbf{u}) + \epsilon(\mathbf{u})$, there must exist a bounded $C > 0$ s.t.

$$\|\Phi x' - \Phi x\| \leq C \|x' - x\| \quad (1.8.2)$$

The final requirement is to be stable to small deformations. Enough so that we can ignore intra-class variations, but not so invariant that an object can morph into another (in the case of MNIST for example, we do not want to be so stable to deformations that 7s can map to 1s). Formally, for a new signal $x_\tau(\mathbf{u}) = x(\mathbf{u} - \tau(\mathbf{u}))$, where $\tau(\mathbf{u})$ is a non constant displacement field (i.e., not just a translation) that deforms the image, we require a $C > 0$ s.t.

$$\|\Phi x_\tau - \Phi x\| \leq C \|x\| \sup_{\mathbf{u}} |\nabla \tau(\mathbf{u})| \quad (1.8.3)$$

The term on the right $|\nabla \tau(\mathbf{u})|$ measures the deformation amplitude, so the supremum of it is a limit on the global deformation amplitude.

1.8.2 Finding the Right Operator

A Fourier modulus satisfies the first two of these requirements, in that it is both translation invariant and stable to additive noise, but it is unstable to deformations due to the infinite support of the sinusoid basis functions it uses. It also loses too much information — very different signals can all have the same Fourier modulus, e.g. a chirp, white noise and the Dirac delta function all have flat spectra.

Unlike the Fourier modulus, a wavelet transform is stable to deformations due to the grouping together frequencies into dyadic packets [mallat_group_2012](#), however, the wavelet transform is not invariant to shifts.

We saw in ?? that the modulus of complex, analytic wavelets commuted with shifts. The real and imaginary parts are also commutative with shifts, but these vary much quicker than the modulus (Figure 1.28). Interestingly, the modulus operator, in this case, does not lose any information [waldspurger_phase_2012](#) (due to the redundancies of the wavelet transform), which is why it may be nice to think of it as a *demodulator*.

The modulus can be made fully invariant by integrating, i.e.,:

$$\int Fx(\mathbf{u})d\mathbf{u} = \int |x * \psi_\lambda(\mathbf{u})|d\mathbf{u}$$

is translation invariant. Total invariance to shifts means integrating over the entire function, which may not be ideal as it loses a significant amount of information in doing this. Instead Bruna and Mallat define scales 2^J , over which their operator is invariant to shifts. Now instead of integrating, the output $\|x * \psi_\lambda\|$ is convolved with an averaging window, or conveniently, the scaling function for the chosen wavelet:

$$\phi_{2^J}(\mathbf{u}) = 2^{-2J}\phi(2^{-J}\mathbf{u})$$

Even still, this averaging means that a lot of information is lost from the first layer outputs ($\|x * \psi_\lambda\|$). Bruna and Mallat combat this by also convolving the output with wavelets that cover the rest of the frequency space, giving

$$U[p]x = U[\lambda_2]U[\lambda_1]x = \|\|x * \psi_{\lambda_1}|\| * \psi_{\lambda_2}\|$$

The choice of wavelet functions λ_1 and λ_2 is combined into a path variable, $p = (\lambda_1, \lambda_2, \dots \lambda_m)$.

Local invariants can be again computed by convolving this with another scaling function ϕ . The result is now a multiscale scattering transform, with coefficients:


$$S[p]x = U[p]x * \phi_{2^J}(\mathbf{u})$$

A graphical representation of this is shown in Figure 1.29.

1.9 Rotation and Translation Invariant Scatternets

Mallat's group refined their Scatternet architecture by expanding their list of invariants to also include rotation. They also experimented with adding scale invariance in `sifre_rotation_2013`, but it was limited to only averaging over scale once, and they were no longer using it in `oyallon_deep_2015`, so for brevity we omit it.

This work was done by two authors, each tackling different challenges. The first is texture analysis with Sifre in `sifre_combined_2012`, `sifre_rotation_2013`, `sifre_rigid-motion_2014`, `sifre_rigid-motion_2014-1`, and the second is image classification with Oyallon in `oyallon_generic_2013`, `oyallon_deep_2015`. In this section, we outline the properties and structure of this extended Scatternet.



images/scatternet_diagram.png

Figure 1.30: The translation invariant Scattering Transform. Scattering outputs are the leftward pointing arrows $S[p]x$, and the intermediate coefficients $U[p]x$ are the centre nodes of the tree. Taken from bruna_invariant_2013.

1.9.1 An Important note on Joint vs. Separable Invariants

When building multiple invariants, some thought must be given as to how to combine them — separably or jointly? Let us call the group of operations we want to be invariant to G , with $g \in G$ a single realization from this group — in this case, G is the group of affine transformations. We want our operator Φ to be invariant to all $g \in G$, i.e., $\Phi(gx) = \Phi(x)$. Building separable invariants would mean representing the group as $G = G_2G_1$ (an assumption of the group, not of our model), and building $\Phi = \Phi_2\Phi_1$, where Φ_1 is invariant to members of G_1 and covariant to members of G_2 , and Φ_2 is invariant to members of G_2 . I.e.,

$$\Phi_2(\Phi_1(g_1g_2x)) = \Phi_2(g_2\Phi_1(x)) = \Phi_2(\Phi_1(x)) \quad (1.9.1)$$

An example of this would be in the group G of 2D translations, building horizontal invariance first, then building vertical invariance second. warn about this approach, however, as it cannot capture the action of G_2 relative to G_1 . In the case of vertical and horizontal translations, for example, it would not be able to distinguish if the patterns had moved apart as well as being shifted, whereas a joint horizontal and vertical translation invariant would be able to distinguish these two cases.

In this vein, suggest that in the case of rotation and translation invariance, a joint invariant should be used, building on the work in [citti_cortical_2006](#), [boscaiu_anthropomorphic_2010](#), [sgallari_scale_2007](#).

1.9.2 Defining the Properties

A translation $g = (v, \theta)$ of the roto-translation group G_{rt} acting on $\mathbf{u} \in \mathbb{R}^2$ combines translation by v and rotation by R_θ as:

$$g\mathbf{u} = v + R_\theta\mathbf{u} \quad (1.9.2)$$

The product of two successive roto-translations $h = (v', \theta')$ and $g = (v, \theta)$ is:

$$gh = (v + R_\theta v', \theta + \theta') \quad (1.9.3)$$

In much the similar approach to the simple translation invariant Scatternet defined above, calculate successive layers of signal coefficients $U[p]x$ that are covariant to the actions of all $g \in G_{rt}$ — i.e.,

$$U[p](gx) = gU[p]x \quad (1.9.4)$$

Creating invariants of order $m = \text{length}(p) = \text{length}([\lambda_1, \lambda_2, \dots, \lambda_m])$ is then done by averaging $hU[p]x$ for all h in G_{rt}

$$S[p]x(g) = \sum_{h \in G_{rt}} hU[p]x\Phi_J(h^{-1}g) \quad (1.9.5)$$

This convolution averages $hU[p]x$ over all rotation angles in a spatial neighbourhood of \mathbf{u} of size proportional to 2^J .

1.9.3 The Operator

1.9.3.1 Roto-Translation Invariance

Although we want to have a joint invariant for rotations and translations, this can be done with a cascade of wavelet transforms — so long as the final averaging operation is done over both rotation and translation. To do just this, building a 3 layer scattering transform, the first layer of which is exactly identical to the previous translation scattering transform, i.e.,

$$\tilde{W}_1 x = (x * \phi_J, \{|x * \psi_{\theta,j}|\}) = (S_0 x, U_1 x) \quad (1.9.6)$$

The second and third layers are, however, new. The invariant part of U_1 is computed with an averaging over spatial and angle variables. *This averaging is implemented at fixed scales j* (see our note earlier about choosing separable scale invariance). For an action $g = (v, \theta)$, the averaging kernel is defined as:

$$\Phi_J(g) = \bar{\phi}(\theta) * \phi_J(u) \quad (1.9.7)$$

Where $\phi_J(u)$ is a kernel that averages each $U_1 x$ over scale 2^J , and $\bar{\phi}(\theta = (2\pi)^{-1})$ averages the result of that average over all angles.

To clarify, we look at an example architecture with $J = 2$ scales and $L = 4$ orientations. The output of the first layer $U_1 x$ would be a set of coefficients:

$$U_1 x = \{|x * \psi_{j,\theta}| \mid j = \{0, 1\}, \theta = k\pi/4, k = \{0, 1, 2, 3\}, \} \quad (1.9.8)$$

i.e., there would be 4 high frequency coefficients, which were created with wavelets centred at $|\omega| = 3\pi/4$, and 4 medium frequency components created with wavelets centred at $|\omega| = 3\pi/8$. Each of these 8 will be averaged across the entire image, then each pair of 4 will be averaged across all 4 rotations, leaving 2 invariants.

To recover the information lost from averaging, also convolve $U_1 x$ with corresponding rotation and scale wavelets to pass on the high frequency information. These roto-translation wavelets, while joint, can also be computed with the cascade of separable wavelets. It may be helpful to consider the spatial variable \mathbf{u} as single dimensional, and consider the rotation variable θ as a second dimension. The above equation calculated the low-low frequency component of these two variables, the remaining components are the low-high, high-low, and high-high.



Figure 1.31: Three dimensional convolution with $\Psi_{\theta_m, j_m, k_m}(u_1, u_2, \theta)$ factorised into a two dimensional convolution with $\psi_{\theta_m, j_m}(u_1, u_2)$ and a one dimensional convolution with $\psi_{k_m}(\theta)$. Colours represent the amplitude of the 3D wavelet. Image taken from sifre_rotation_2013.

We define the low frequency spatial scaling functions $\phi_J(u)$ ⁸, the spatial wavelets $\psi_{\theta, j}(u)$, the rotation scaling function $\bar{\phi}(\theta)$ (which is just the constant $(2\pi)^{-1}$, but we write out in generic form nonetheless), and the rotation wavelet $\bar{\psi}_k(\theta)$, which is a 2π periodic wavelet.

Then, the remaining low-high, high-low, and high-high information is:

$$\Psi_{0, J, k_2}(u, \theta) = \phi_J(u) * \bar{\psi}_{k_2}(\theta) \quad (1.9.9)$$

$$\Psi_{\theta_2, j_2, \cdot}(u, \theta) = \psi_{\theta_2, j_2}(u) * \bar{\phi}(\theta) \quad (1.9.10)$$

$$\Psi_{\theta_2, j_2, k_2}(u, \theta) = \psi_{\theta_2, j_2}(u) * \bar{\psi}_{k_2}(\theta) \quad (1.9.11)$$

The k parameter is newly introduced here, and it represents the number of scales the rotation wavelet has (a typical value used by was $K = 3$). We call this combined operator $\Psi_{\theta_m, j_m, k_m}$. See Figure 1.30 for what this looks like.

The wavelet-modulus operator then is:

$$\tilde{W}_m Y = (Y * \Phi_J(g), |Y * \Psi_{\theta_m, j_m, k_m}(g)|) \quad (1.9.12)$$

⁸we temporarily drop the boldface from the spatial parameter u to make it clearer it can be considered as single dimensional

for $m \geq 2$ and the final third order roto-translation Scattnet is:

$$Sx = (x * \phi_J(\mathbf{u}), U_1x * \Phi_J(p_1), U_2x * \Phi_J(p_2)) \quad (1.9.13)$$

with $p_1 = (\mathbf{u}, \theta_1, j_1)$ and $p_2 = (\mathbf{u}, \theta_1, j_1, \theta_2, j_2, k_2)$.

1.10 Gradients of Complex Activations

1.10.1 Cauchy-Riemann Equations

None of these will be solved. However, we can still find the partial derivatives w.r.t. the real and imaginary parts.

1.10.2 Complex Magnitude

Care must be taken when calculating gradients for the complex magnitude, as the gradient is undefined at the origin. We take the common approach of setting the gradient at the corner point to be 0.

Let us call the real and imaginary inputs to a magnitude block x and y , and we define the real output r as:

$$r = |x + jy| = \sqrt{x^2 + y^2}$$

Then the partial derivatives w.r.t. the real and imaginary inputs are:

$$\begin{aligned} \frac{\partial r}{\partial x} &= \frac{x}{\sqrt{x^2 + y^2}} = \frac{x}{r} \\ \frac{\partial r}{\partial y} &= \frac{y}{\sqrt{x^2 + y^2}} = \frac{y}{r} \end{aligned}$$

Except for the singularity at the origin, these partial derivatives are restricted to be in the range $[-1, 1]$. The complex magnitude is convex in x and y as:

$$\nabla^2 r(x, y) = \frac{1}{r^3} \begin{bmatrix} y^2 & -xy \\ -xy & x^2 \end{bmatrix} = \frac{1}{r^3} \begin{bmatrix} y \\ -x \end{bmatrix} \begin{bmatrix} y & -x \end{bmatrix} \geq 0$$

These partial derivatives are very variable around 0. **Show a plot of this.** We can smooth it out by adding a smoothing term:

$$r_s = \sqrt{x^2 + y^2 + b} - \sqrt{b}$$

This keeps the magnitude zero for small x, y but does slightly shrink larger values. The gain we get however is a new smoother gradient surface **Plot this.**

References

- [1] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.
- [2] D. R. Wilson and T. R. Martinez, “The general inefficiency of batch training for gradient descent learning”, eng, *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 16, no. 10, pp. 1429–1451, Dec. 2003.
- [3] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, “Don’t Decay the Learning Rate, Increase the Batch Size”, *arXiv:1711.00489 [cs, stat]*, Nov. 2017. arXiv: 1711.00489 [cs, stat].
- [4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [5] L. Bottou, “Stochastic Gradient Descent Tricks”, en-US, vol. 7700, Jan. 2012.
- [6] G. Montavon, G. Orr, and K.-R. Müller, *Neural Networks: Tricks of the Trade*, 2nd. Springer Publishing Company, Incorporated, 2012.
- [7] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient BackProp”, en, in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science 7700, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., Springer Berlin Heidelberg, 2012, pp. 9–48.
- [8] Y. A. Ioannou, “Structural Priors in Deep Neural Networks”, PhD thesis, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, United Kingdom, Sep. 2017.
- [9] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: 1412.6980 [cs].
- [10] J. Duchi, E. Hazan, and Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.
- [11] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method”, *arXiv:1212.5701 [cs]*, Dec. 2012. arXiv: 1212.5701 [cs].
- [12] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain [J]”, *Psychol. Review*, vol. 65, pp. 386–408, Dec. 1958.

- [13] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks”, in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.
- [14] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines”, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.
- [15] M. L. Minsky and S. A. Papert, *Perceptrons: Expanded Edition*. Cambridge, MA, USA: MIT Press, 1988.
- [16] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators”, *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989.
- [17] G. Cybenko, “Approximation by superpositions of a sigmoidal function”, in, *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989.
- [18] B. Widrow and M. E. Hoff, “Neurocomputing: Foundations of Research”, in, J. A. Anderson and E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, pp. 123–134.
- [19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1”, in, D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.
- [20] N. Kingsbury, “The Dual-Tree Complex Wavelet Transform: A New Technique For Shift Invariance And Directional Filters”, in *1998 8th International Conference on Digital Signal Processing (DSP)*, Utah, Aug. 1998, pp. 319–322.
- [21] —, “The dual-tree complex wavelet transform: A new efficient tool for image restoration and enhancement”, in *Signal Processing Conference (EUSIPCO 1998), 9th European*, Sep. 1998, pp. 1–4.
- [22] N. Kingsbury, “Image processing with complex wavelets”, *Philosophical Transactions of the Royal Society a-Mathematical Physical and Engineering Sciences*, vol. 357, no. 1760, pp. 2543–2560, Sep. 1999.
- [23] I. W. Selesnick, R. G. Baraniuk, and N. G. Kingsbury, “The dual-tree complex wavelet transform”, *Signal Processing Magazine, IEEE*, vol. 22, no. 6, pp. 123–151, 2005.
- [24] P. de Rivaz and N. Kingsbury, “Bayesian image deconvolution and denoising using complex wavelets”, in *2001 International Conference on Image Processing, 2001. Proceedings*, vol. 2, Oct. 2001, 273–276 vol.2.
- [25] Y. Zhang and N. Kingsbury, “A Bayesian wavelet-based multidimensional deconvolution with sub-band emphasis”, in *Engineering in Medicine and Biology Society*, 2008, pp. 3024–3027.

- [26] G. Zhang and N. Kingsbury, “Variational Bayesian image restoration with group-sparse modeling of wavelet coefficients”, *Digital Signal Processing*, Special Issue in Honour of William J. (Bill) Fitzgerald, vol. 47, pp. 157–168, Dec. 2015.
- [27] M. Miller and N. Kingsbury, “Image denoising using derotated complex wavelet coefficients”, eng, *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, vol. 17, no. 9, pp. 1500–1511, Sep. 2008.
- [28] S. Hatipoglu, S. K. Mitra, and N. Kingsbury, “Texture classification using dual-tree complex wavelet transform”, in *Seventh International Conference on Image Processing and Its Applications*, 1999, pp. 344–347.
- [29] P. de Rivaz and N. Kingsbury, “Complex wavelet features for fast texture image retrieval”, in *1999 International Conference on Image Processing, 1999. ICIP 99. Proceedings*, vol. 1, 1999, 109–113 vol.1.
- [30] P. Loo and N. G. Kingsbury, “Motion-estimation-based registration of geometrically distorted images for watermark recovery”, P. W. Wong and E. J. Delp III, Eds., Aug. 2001, pp. 606–617.
- [31] H. Chen and N. Kingsbury, “Efficient Registration of Nonrigid 3-D Bodies”, *IEEE Transactions on Image Processing*, vol. 21, no. 1, pp. 262–272, Jan. 2012.
- [32] J. Fauqueur, N. Kingsbury, and R. Anderson, “Multiscale keypoint detection using the dual-tree complex wavelet transform”, in *Image Processing, 2006 IEEE International Conference On*, IEEE, 2006, pp. 1625–1628.
- [33] R. Anderson, N. Kingsbury, and J. Fauqueur, “Determining Multiscale Image Feature Angles from Complex Wavelet Phases”, en, in *Image Analysis and Recognition*, ser. Lecture Notes in Computer Science 3656, M. Kamel and A. Campilho, Eds., Springer Berlin Heidelberg, Sep. 2005, pp. 490–498.
- [34] —, “Rotation-invariant object recognition using edge profile clusters”, in *Signal Processing Conference, 2006 14th European*, IEEE, 2006, pp. 1–5.
- [35] P. Bendale, W. Triggs, and N. Kingsbury, “Multiscale keypoint analysis based on complex wavelets”, in *BMVC 2010-British Machine Vision Conference*, BMVA Press, 2010, pp. 49–1.
- [36] E. S. Ng and N. G. Kingsbury, “Robust pairwise matching of interest points with complex wavelets”, *Image Processing, IEEE Transactions on*, vol. 21, no. 8, pp. 3429–3442, 2012.
- [37] N. Kingsbury, “Complex wavelets for shift invariant analysis and filtering of signals”, *Applied and Computational Harmonic Analysis*, vol. 10, no. 3, pp. 234–253, May 2001.

-
- [38] J. Bruna and S. Mallat, “Classification with scattering operators”, in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2011, pp. 1561–1566.
 - [39] —, “Invariant Scattering Convolution Networks”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, Aug. 2013.