Heriot-Watt University

Doctoral Thesis

Multimodal Representation Learning

Author:
Eli Sheppard

Supervisors:
Dr. Katrin Lohan
Dr. Oliver Lemon

Doctoral Thesis submitted in fulfilment of the requirements for the degree of PhD Robotics and Autonomous Systems

in the

Edinburgh Centre for Robotics

June 2019

Declaration of Authorship

I, Eli Sheppard, declare that this theis titled, 'Multimodal Representation Learning' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: ____________________________________

Date: ____________________________________

Abstract

Acknowledgements

I would like to thank Katrin and Oliver for their continued support throughout my academic career. I would also like to thank Ingo Keller for his invaluable lessons on the art of Python programming.

Contents

Abbreviations

RNN	**R**ecurrent **N**eural **N**etwork
GRU	**G**ated **R**ecurrent **U**nit
LSTM	**L**ong **S**hort **T**erm **M**emory
CNN	**C**onvolutional **N**eural **N**etwork
YARP	**Y**et **A**nother **R**obotics **P**latform

Chapter 1

A Primer on Artifical Neural Networks

1.1 Introduction

This chapter presents an overview of the mathematical theory behind artificial neural networks and how they learn. It should serve as a quick guide to the techniques used in the experiments within this thesis.

1.2 Perceptrons

The perceptron is the parent of modern artificial neurons Rosenblatt [1958]. Perceptrons arranged in layers, referred to as multi-layer perceptrons, are therfore the predecessor of modern artificial neural networks.

1.2.1 What is a Perceptron

Perceptrons are biologically inspired computation units and are a type of artificial neuron. They take a series of binary inputs, compute a weighted sum and produce a binary output based on the value of this sum as seen in figure 1.1.

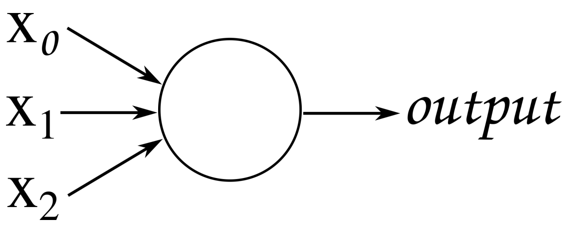

FIGURE 1.1: A perceptron with three binary inputs, x_0, x_1, x_2.

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b < 0 \\ 1 & \text{if } \sum_j w_j x_j + b \geq 0 \end{cases} \tag{1.1}$$

Where x_j is the jth input, w_j is it associated weight and b is a constant value called the bias, which affects how easy it is for the neuron to activate. The output of the perceptron is caluclated using the formulation shown in 1.1.

By adjusting each of the weights we can change the output of the perceptron. For example, if we wanted to use a perceptron to decide whether to have a picnic today we can select a set of relevant inputs, "the weather is nice" and "the pollen count is low".

If it is a sunny day, and the pollen count is high, the input to the perceptron would be $[1, 0]$.

We will set our weights depending on how important each of the inputs is. No one likes a picnic in the rain, so the weather is important, whilst the pollen count is only important if you suffer from hayfever. We will select a bias of -5 for our perceptron.

For person A, the weights might look like $[7, 0]$ (person A doesn't suffer from hayfever). Therefore the output of the perceptron would be 1 as $1 \times 7 + 0 \times 0 - 5 = 2$ is greater than 0, so person A will go for a picnic.

Person B owns a large umbrella (so the weather doesn't matter as much) but they do suffer from hayfever, their weights might look like $[4, 7]$. The output of the perceptron would be 0 as $1 \times 4 + 0 \times 7 - 5 = -1$ is less than 0. Therefore, person B would wait for a day with a lower pollen count for a picnic.

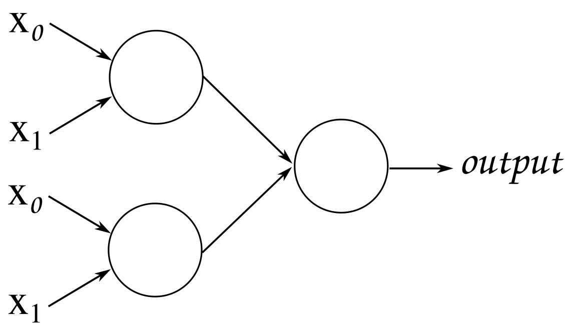

FIGURE 1.2: A multi-layer perceptron consisting of three perceptrons arranged in two layers, with three binary inputs, x_0, x_1, x_2.

1.2.2 Multi-Layer Perceptron

In the previous section I demonstrated how a single perceptron can be used to make decisions based on a set of inputs. However, things get much more interesting when we start to link multiple perceptrons together into multi-layer perceptrons (MLP).

If we take the example from the previous section about deciding to go for a picnic or not, we can use the MLP shown in 1.2 to consider what happens if person A and B want to go for a picnic together.

Given the previous conditions of a sunny day with a high pollen count, person A wants to go whilst person B does not, as demonstrated by the outputs of their individual perceptrons. Taking these outputs as inputs to the second layer of our MLP, we can see whether the picnic will go ahead.

This time we will set the weights of the second layer based on who shouts the loudest. In this case, person A really wants to go and is very vocal about it. $1 \times 9 + 0 \times 5 - 5 = 4$ so the picnic will go ahead. This resulted in person B developing a headache and sneezing a lot. Clearly we need to find a way to adjust the weights of our MLP so that it makes better decisions in the future.

1.3 Activation Functions

Only being able to handle binary values is a major drawback of the perceptron. An artificial neuron which can handle continuous values is much more useful.

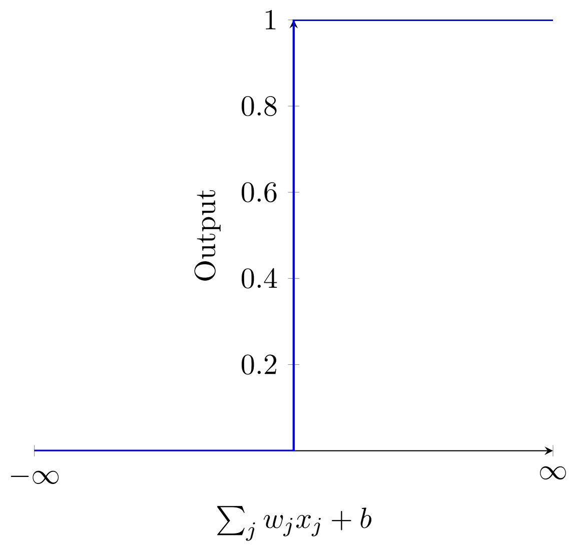

FIGURE 1.3: Visualisation of the perceptron activation function.

This limitation occurs due to the activation function of the perceptron shown in equation 1.1. Visualising the perceptron activation function highlights that changes in the output of the neuron are not proportional to changes in the weights of the neuron as seen in figure 1.3. With a few small changes we can make an artifical neuron that accepts continuous values and has a continuous output.

A continuous output is very important as it means that a small change in the weights of a neuron will cause a small change in its output. This makes it much easier to understand how changing the weights affects the output of the network as the change in output becomes proportional to the change in weights as shown in equation 1.2

$$\delta Output \propto \delta W \tag{1.2}$$

1.3.1 Sigmoid Neurons

The sigmoid neuron uses the sigmoid function, shown in equation 1.3 as its activation function.

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{1.3}$$

z is the sum of the weights multiplied by their respective inputs as seen in equation 1.4.

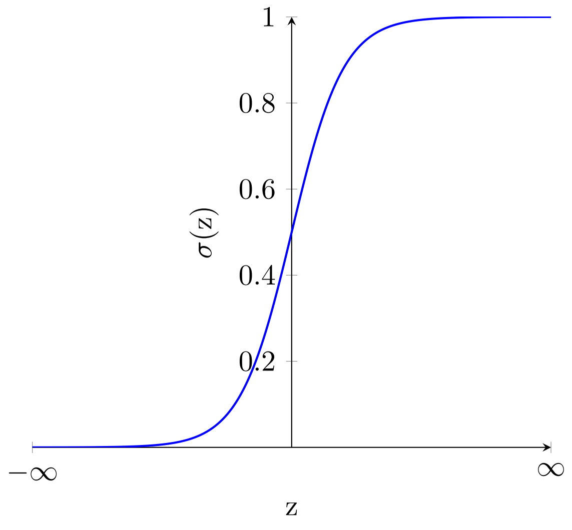

FIGURE 1.4: Visualisation of the sigmoid activation function.

$$z = \sum_j w_j x_j + b \tag{1.4}$$

As we can see in figure 1.4, as z changes, their is a proportional change in the output $\sigma(z)$, unlike in figure 1.3 where the output only changes when z crosses the y-axis.

Now that we have an activation function that can produce continuous values, we can consider a much more diverse range of data to train our neural networks with. So instead of just making yes or no decisions we can look at, for example, the colours of pixels and decide if there is a cat in the image or given todays weather, predict if it will rain tomorrow.

1.3.1.1 Other activation functions

There are two other important and commonly used activation functions, though many others exist. These are, the hyperbolic tangent (Tanh) shown in figure 1.5 and rectified linear unit (Relu) shown in figure 1.6.

The Tanh function looks similar to the sigmoid function, however it has an output between negative one and one, where the sigmoid goes from zero to one. This is useful when having negative values within the network is important.

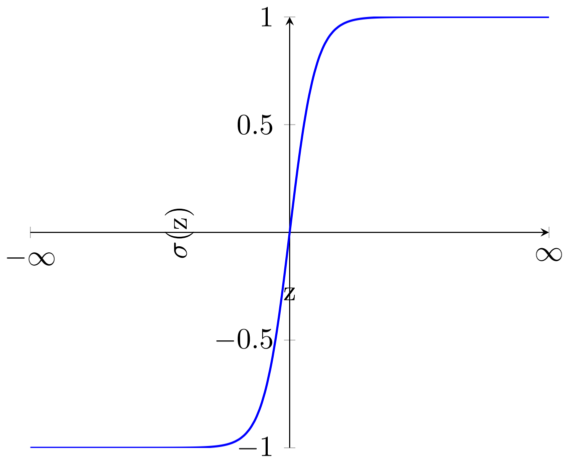

FIGURE 1.5: Visualisation of the hyperbolic tangent activation function.

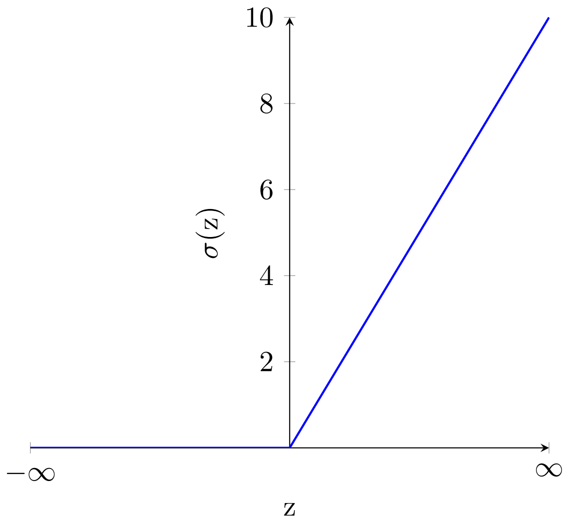

FIGURE 1.6: Visualisation of the rectified linear unit activation function.

The Relu was created to address and issue known as vanishing gradients. This is to do with how neural networks are trained, as expalined in the enxt session. Briefly, as training depends on error gradients, having a linear activation function means that deeper networks [1] can be trained as non-linear functions like the sigmoid or Tanh will have reduced gradient magnitude when they are differentiated to backpropogate the error through the network. If that doesn't make sense yet, don't worry, it will become clear shortly.

[1]Deeper networks are beneficial as they can make more complex decisions by layering more and more simple decsions as shown in the MLP example in section1.2

1.4 Learning Algorithms

In section 1.2 we saw how simple computation units can make simple decisions and how these can be chained together to make more complex decisions. However, sometimes the decisions we make are wrong and we need to learn from our mistakes.

In general, perceptrons (and their more modern decendents) will have their initial weights set randomly, unlike in our example where I chose them.

Randomly setting weights is useful in practice as we will usually have large numbers of inputs, weights, layers and artifical neurons, so setting each by hand would be intractable. Also, if we knew what weights to set, we wouldn't need a neural network or a learning algorithm to solve our problem as we would likely have a more efficient mathematical description of the problem [2]

With random weights our networks will make a lot of wrong decisions! However if we have a smart way of adjusting our weights we can make our neural networks get better at making decisions over time. This improvement is what I am referring to when I say "machine learning".

1.4.1 Types of Training

There are many ways to train neural networks, supervised, unsupervised, reinforcement or adversarial learning (to name a few). The experiments in this thesis will focus on supervised and unsupervised learning.

Supervised learning - an external system is used to provide ground truth values for the desired output of the network. An example would be classification using standardised datasets like MNIST handwritten digits LeCun [1998].

Unsupervised learning - no external system provides ground truth values, instead these are inferred directly from the data. Autoencoders (discussed in great deal in later chapters) are a good example of this.

[2] there is a large body of work concerning the best way to initialise neural networks, I will briefly comment on this in section **??**.

1.4.2 Cost functions: How wrong am I?

In order for our neural networks to learn, they need feedback as to how wrong they are. This is done using a cost function, a formula which gives a measure of how good or bad our network is at a particular task.

Depending on the particular method we use to train our networks, the cost function may be given a different name. For example in reinforcement learning, the cost function is typically called a value function. However, they all serve the same purpose, which is to provide feedback as to how well our network is doing at the task it has been assigned. Therefore, to avoid jargon I will refer to this as a cost or loss function throughout this thesis.

The cost fuction can take on many forms, in supervised and unsupervised training alike. Here are a few of the more commonly used ones and an explanation of each.

1.4.2.1 Cross Entropy

1.4.2.2 Mean Squared Error

$$C = \frac{1}{n}\sum_{i=0}^{n}(Y_i - \hat{Y})^2 \tag{1.5}$$

1.4.2.3 Kullback-Leibler Divergence

1.4.3 Gradient Decent

now that we have a method for determining how well our neural network performs a task, we can observe how this changes as the weights of the network are changed.

To make things simple, lets assume our network only has two weights, though the following derivation generalises to any number of weights. This could give us a cost landscape like that shown in figure 1.7. Our aim is to minimise the cost as the cost is a measure of how wrong our network is.

In more definate terms we wish to minimise equation 1.6, where Y_i is the desired output value for input i and z is as defined in equation 1.4.

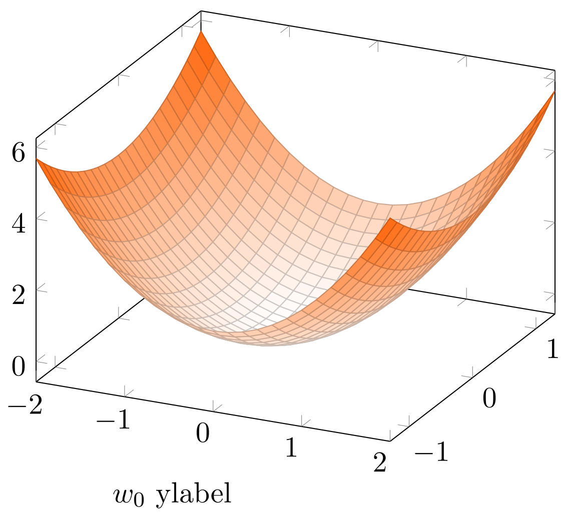

FIGURE 1.7: The cost landscape of a bivariate function

$$C(w_0, w_1) = \frac{1}{n}\sum_{i=0}^{n}(Y_i - \sigma(z_i))^2 \tag{1.6}$$

This is the mean squared error rewritten slightly to highlight the dependence on the weights of the network.

If we make a small change to either of the weights w_0 and w_1 we will get a small change in the cost, C as seen in equation 1.7.

$$\Delta C \approx \frac{\delta C}{\delta w_0}\Delta w_0 + \frac{\delta C}{\delta w_1}\Delta w_1 \tag{1.7}$$

We want to minimise C so we want ΔC to be negative. To make ΔC negative we will need to first denote that the gradient of C, ∇C.

$$\nabla C \equiv (\frac{\delta C}{\delta w_0}, \frac{\delta C}{\delta w_1})^T \tag{1.8}$$

In equation 1.8 we can see that the gradient of C is equivalent to the partial derivatives of C with repect to the weights of the network. For simplicity we will collect the changes in weights into a single vector $\Delta w \equiv (\Delta w_0, \Delta w_1)^T$.

By substituing this into equation 1.7 we get equation 1.9.

$$\Delta C \approx \nabla C \cdot \Delta w \tag{1.9}$$

Looking at equation 1.9, we can see that to make ΔC negative we can set Δw as in equation 1.10, where η is a small positive constant called the learning rate.

$$\Delta w = -\eta \nabla C \tag{1.10}$$

By substituting equation 1.10 into equation 1.9 we can see that $\Delta C = -\eta |\nabla C|^2$ and because $|\nabla C|^2 \geq 0$ this gaurantees that C will always decrease if the weights are changed in accordance to equation 1.10.

By iteratively making changes to the weights according to 1.10 we will eventually reach the minima of C so long as we select an appropriate learning rate [3].

[3]Learning rate is discussed in more detail in section 1.4.5.1

1.4.4 Backpropagation

1.4.5 Extensions and Improvements

1.4.5.1 Learning Rate Schedule

1.4.5.2 Momentum

1.5 Convolutional Neural Networks

1.5.1 What is Convolution?

1.5.1.1 Kernels

1.5.1.2 Strides

1.5.1.3 Dilations

1.5.2 Transposed Convolutions

1.6 Recurrent Neural Networks

1.6.1 Vanilla RNN

1.6.2 Gated RNN

1.7 Summary

Chapter 2

Sensory Redundancy: Are two heads better than one?

2.1 Why have one sensor when two are better?

A key part of the human experience is that it is multisensory.

2.1.1 Aims

2.2 Flikr25K

Flikr25k classification experiment - more robustness in multimodal case.

Model	Test Condition	MSE	Acc
Image	Im	2.2417	0.2969
TAG	Tag	2.6069	0.2196
Multimodal	Bi	2.2221	0.3182
	Im	2.3801	0.2530
	Tag	3.2315	0.2304

TABLE 2.1: Results from the combined MNIST Handwritten Digits and UCU Arabic Spoken Digits for the bimodal only testing condition.

2.2.1 Dataset Description

2.2.2 Problem Description

2.2.3 Network Description

2.2.4 Results

2.2.5 Discussion

2.3 There and back again: Generating images from natural language

Combining multiple sources of information can be beneficial for classification, as shown in the previous section 2.2. However, it also presents an opportunity to explore the generative capabilities of neural networks.

2.4 UCU Arabic Spoken Digits and MNIST

2.4.1 Dataset Description

2.4.1.1 UCU Arabic Spoken Digits

UCU Arabic Spoken Digits (UCU) is a dataset containing the 13 Mel Frequency Cepstrum Coefficients representing the audio of the digits 0 to 9 being said by 88 different speakersHammami and Bedda [2010], Hammami and Sellam [2009]. It contains 8800 utterances (10 digits x 10 repetitions x 88 speakers). Audio was sampled at 11025Hz.

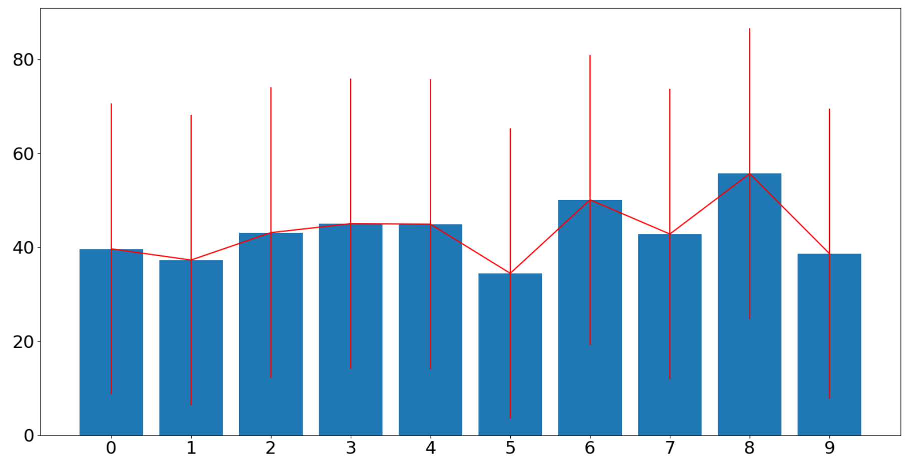

FIGURE 2.1: Mean number of samples for each digit in the UCU Arabic Spoken Digits Dataset. Red bars show the standard devaition in length for each each digit.

As utterances are not of a fixed length, it is necessary to pad the utterances to be equal length. The longest utterance in the dataset contains 93 samples. For ease of downsampling within the neural network, we pad all uttereances to be 100 samples long by appending zeros. We also pad each sample to contain 16 features, the 13 MFCCs and a 3 zeros to facilitate down sampling [1]Downsampling is easier with even numbers of features and samples as we can half the size of the data using strided convolutions, s=(2,2), without worrying about integer division errors e.g. $5/2 = 2$).

As we will be classifying the digits, it is necessary to check that padding the digits does not provide a cheat for this. As such, we analise the mean number of samples for each digit and their standard deviations as depicted in 2.1. We then perform a linear regression on the number of samples to demonstrate that it is not possible to accurately predict a digits class given only the length of the utterance.

Figure 2.1 shows the mean number of samples for each digit in the UCU dataset. Whilst there is a difference in the mean length of each digit, the standard deviation for each digit is large as can be seen in 2.2. Therefore, trying to classifiy the digits just by the number of samples in an example, would be ineffective.

Performing a linear regression on the number of samples for each digit versus its class, we get an accuracy of 6.18% when classifying the test set.

[1](

Digit	Mean Length (samples)	Standard Deviation
0	39.66	±29.25
1	37.27	±25.58
2	43.11	±31.43
3	45.02	±34.05
4	44.92	±31.63
5	34.44	±23.63
6	50.10	±35.90
7	42.81	±30.41
8	55.67	±41.01
9	38.63	±30.94

TABLE 2.2: Mean length and standard deviation of digits in the UCU dataset.

2.4.1.2 MNIST Handwritten Digits

MNIST Handwritten Digits (MNIST) is a well known and ofte used dataset in the machine learning community. It contains 60000 training samples and 10000 test samples, evenly split between the digits 0 to 9. Each digit is presented as a grey-scale, 28x28 pixel image LeCun [1998].

2.4.2 Problem Description

The experiments with these datasets will explore two problems, classification and bidirectional symbol grounding.

2.4.2.1 Classification

Each of the utterances or images from the UCU or MNIST datasets are classified as being a numeric digit, 0 to 9. By combining both datasets, we can explore whether the addition of a second modality enhances overall classification accuracy.

I start off exploring the classification abilities of neural networks by performing a linear regression with a single, dense neural network layer with a softmax activation on the embedding of an autoencoder. To generate baseline classifications accuracies for each dataset, I use autoencoders for each dataset separately.

Once a classification baseline has been established for each dataset, I make use of a multimodal autoencoder, using paired examples from each dataset to train it.

2.4.2.2 Bidirectional Symbol Grounding

I demonstrate that a neural network can perform bidirectional symbol grounding Barsalou [2008] when presented with aligned images and utterances. To explore this, I show how a multimodal autoencoder (MAE) can learn to generate the correct image of a digit given an utterance and MFCCs which have a small mean squared error from the correct utterance.

2.5 Experiment Details

2.5.1 Dataplumbing

Images from the MNIST dataset are kept at their original scale of 28x28 pixels and are normalised such that each pixel has a value from 0 to 1 based on it's intensity, with zero equating to black and 1 being white.

The UCU dataset

blahblah

.

2.5.1.1 Combining Datasets

Due the difference in size between the datasets, 8,800 samples for UCU and 80,000 samples for MNIST, I randomly sample from the UCU dataset and pair this with a sample from MNIST. This means that each sample from UCU is repeated approxiamtely 9.1 times ($80000/8800 = 9.0909$).

2.5.2 Merging Modalities

In order to combine the two different modalities, I explore different merging techniques: Concatenation (*Concat*) and Addition (*Add*). To do this, the embeddings of both modalities are ensured to have the same shape and then are merged using one of the methods described in Equations 2.1 and 2.2.

$$merged = x1_i^* || x2_i^* \tag{2.1}$$

$$merged = x1_i^* + x2_i^* \tag{2.2}$$

Where $x1_i^*$ and $x2_i^*$ represent the output of the image and MFCC encoders respectively for given image and MFCC inputs $x1_i$ and $x2_i$.

2.5.3 Training Procedures

The MAEs are trained in two different ways for both types of merging, these are referred to as bimodal (Bi) and randomly degraded (RD).

2.5.3.1 Bimodal

Under the bimodal training procedure, the MAE is presented with bimodal input for all training instances. That is, both image and MFCC inputs are present for all inputs and the MAE is trained to reproduce this input as its output.

2.5.3.2 Randomly Degraded

Under the randomly degraded training procedure, one third of image inputs are removed at random and a third of MFCC inputs are removed at random. It is ensured that each training instance always has at least one input modality present. That is, a training instance never has both its image and MFCC input removed.

The removed modality is replaced with an array of zeros of the same shape as the original input (28x28 for the image and 100x16 for MFCCs).

The MAE is then trained to reproduce both the image and MFCCs regardless of whether one of these has been ommited from the input.

2.5.4 Testing Conditions

Each MAE (Concatenate and Addition) was tested in three different ways, Bimodal, Image only and MFCC only.

2.5.4.1 Bimodal

In the bimodal testing condition both image and MFCC data are used as input for each testing instance and we are interested in observing the classification accuracy as well as the total regeneration loss.

2.5.4.2 Image Only

In the image only testing condition, only images are provided as input data. We are interested in the classification accuracy as well as the MFCC regeneration loss. We are not interested in the image regeneration loss, as it is expected that, as the image is provided as input this will be low (which it is for all models and training procedures as seen in table 2.6).

2.5.4.3 MFCC Only

Similarly to the image only condition, the MFCC only condition provides only MFCCs as input and we are interested in the classification accuracy as well as the image regeneration loss. We are not interested in the MFCC regeneration loss as MFCCs are given as input it will be low (which it is for all models and training procedures as seen in table 2.6).

2.5.5 Network Description

To generate the baseline classification accuracies for both the UCU and MNIST data, I make use of unimodal autoencoders. The descrition of the autoencoder for the MNIST dataset can be found in table 2.3 and the autoencoder for the UCU dataset in table 2.4.

Block	Layer	Type	Neurons	Kernel	Strides	Activation
Encoder	1	2D Conv	32	(3,3)	(1,1)	Relu
	2	2D Conv	64	(3,3)	(2,2)	Relu
	3	2D Conv	64	(3,3)	(2,2)	Relu
	4	Dropout p=0.25				
Embedding	5	2D Conv	32	(3,3)	(1,1)	Relu
Classifier	6c	Dense	10			Softmax
Decoder	6	Dropout p=0.25				
	7	2D Trans Convn	64	(3,3)	(2,2)	TanH
	8	2D Transp Conv	64	(3,3)	(2,2)	TanH
	9	2D Trans Conv	32	(3,3)	(1,1)	TanH
	10	2D Trans Conv	1	(3,3)	(1,1)	Sigmoid

Table 2.3: Image autoencoder and classifier. Layer 6c performs classification, whilst the branch starting at layer 6 regenerates the image.

Block	Layer	Type	Neurons	Kernel	Strides	Activation
Encoder	1	2D Conv	32	(3,3)	(1,1)	Relu
	2	2D Conv	64	(3,3)	(2,2)	Relu
	3	Dropout p=0.25				
	4	Dense	3136			
	5	Reshape (7,7,64)				
Embedding	6	2D Conv	32	(3,3)	(1,1)	Relu
Classifier	7c	Dense	10			Softmax
Decoder	7	Dropout p=0.25				
	9	Dense	3200			
	10	Reshape (25,4,32)				
	11	2D Trans Conv	64	(3,3)	(2,2)	TanH
	12	2D Trans Conv	64	(3,3)	(2,2)	TanH
	13	2D Trans Conv	32	(3,3)	(1,1)	TanH
	14	2D Trans Conv	1	(3,3)	(1,1)	Sigmoid

Table 2.4: MFCC autoencoder and classifier. Layer 7c performs classification, whilst the branch starting at layer 7 regenerates the MFCCs. The addition of reshape layers is to ensure the final shape of the regenerated MFCCs matches the target shape whilst the embedding shape matches that of the embedding of the image autoencoder.

By combining the two autoencoders described in tables 2.3, 2.4, a multimodal autoencoder is created. The embeddings from each of the unimodal autoencoders are merged by either, concatenation, addition or multiplication.

2.5.6 Results

2.5.7 Classification Results

Table 2.6 shows the complete results for all of the experiments run on the combined UCU and MNIST dataset. For convenience, this table is broken down into three other

Block	Layer	Type	Neurons	Kernel	Strides	Activation
Image Encoder	1i	2D Conv	32	(3,3)	(1,1)	Relu
	2i	2D Conv	64	(3,3)	(2,2)	Relu
	3i	2D Conv	64	(3,3)	(2,2)	Relu
	4i	Dropout p=0.25				
MFCC Encoder	1m	2D Conv	32	(3,3)	(1,1)	Relu
	2m	2D Conv	64	(3,3)	(2,2)	Relu
	3m	Dropout p=0.25				
	4m	Dense	3136			
	5m	Reshape (7,7,64)				
Merge	6im	Merge				
Embedding	7im	2D Conv	32	(3,3)	(1,1)	Relu
Classifier	8c	Dense	10			Softmax
Image Decoder	8i	Dropout p=0.25				
	9i	2D Trans Conv	64	(3,3)	(2,2)	TanH
	10i	2D Trans Conv	64	(3,3)	(2,2)	TanH
	11i	2D Trans Conv	32	(3,3)	(1,1)	TanH
	12i	2D Trans Conv	3	(3,3)	(1,1)	Sigmoid
MFCC Decoder	8m	Dropout p=0.25				
	9m	Dense	3200			
	10m	Reshape (25,4,32)				
	11m	2D Trans Conv	64	(3,3)	(2,2)	TanH
	12m	2D Trans Conv	64	(3,3)	(2,2)	TanH
	13m	2D Trans Conv	32	(3,3)	(1,1)	TanH
	14m	2D Trans Conv	1	(3,3)	(1,1)	Sigmoid

TABLE 2.5: Image and MFCC multimodal autoencoder. Layers marked i, m, im and c are image, MFCC, image and MFCC and classification respectively.

tables, 2.7 for the bimodal testing condition, 2.8 for the image only condition and 2.9 for the MFCC only testing condition.

All results reported here are the mean of a four-fold cross validation. That is, each training and testing condition was run four times and the average results are shown.

In table 2.7 it can be seen that both Concatenate and Addition merging produce models with better prediction accuracy than the baseline models, regardless of training procedure.

In comparison to the results shown in table 2.7, the results in table 2.8 show that the MAEs trained using the bimodal training procedure do not generalise well when only a single input modality is available. Where as the MAEs trained using the randomly degraded training procedure perform almost as well as the baseline model.

Model	Training	Testing	Im MSE	MFCC MSE	Lb MSE	Total MSE	Acc
Image AE	Im	Im	0.0027		0.0019	0.0046	0.9883
MFCC AE	MFCC	MFCC		0.0113	0.0026	0.0139	0.9832
Add MAE	Bi	Bi	0.0030	0.0401	0.0006	0.0437	0.9967
		Im	0.0076	0.1641	0.0763	0.2479	0.4501
		MFCC	0.1138	0.0399	0.0057	0.1594	0.9624
	RD	Bi	0.0033	0.0176	0.0001	0.0210	0.9993
		Im	0.0027	0.0455	0.0025	0.0508	0.9834
		MFCC	0.0556	0.0166	0.0031	0.0752	0.9789
Concat MAE	Bi	Bi	0.0031	0.0423	0.0009	0.0462	0.9945
		Im	0.0055	0.2029	0.0843	0.2927	0.3496
		MFCC	0.1138	0.0420	0.0079	0.1637	0.9455
	RD	Bi	0.0030	0.0426	0.0002	0.0458	0.9986
		Im	0.0026	0.0737	0.0026	0.0788	0.9834
		MFCC	0.0554	0.0416	0.0026	0.0996	0.9827

Table 2.6: Results from the combined MNIST Handwritten Digits and UCU Arabic Spoken Digits

Model	Training	Im MSE	MFCC MSE	Acc
Image AE	Im	**0.0027**		0.9883
MFCC AE	MFCC		**0.0113**	0.9832
Add MAE	Bi	0.0030	0.0401	0.9967
	RD	0.0033	0.0176	**0.9993**
Concat MAE	Bi	0.0031	0.0423	0.9945
	RD	0.0030	0.0426	0.9986

Table 2.7: Results from the combined MNIST Handwritten Digits and UCU Arabic Spoken Digits for the bimodal only testing condition.

Model	Training	MFCC MSE	Acc
Image AE	Im		**0.9883**
Add MAE	Bi	0.1641	0.4501
	RD	**0.0455**	0.9834
Concat MAE	Bi	0.2029	0.3496
	RD	0.0737	0.9834

Table 2.8: Results from the combined MNIST Handwritten Digits and UCU Arabic Spoken Digits for the image only testing condition.

Model	Training	Im MSE	Acc
MFCC AE	MFCC		**0.9832**
Add MAE	Bi	0.1138	0.9624
	RD	0.0556	0.9789
Concat MAE	Bi	0.1138	0.9455
	RD	**0.0554**	0.9827

Table 2.9: Results from the combined MNIST Handwritten Digits and UCU Arabic Spoken Digits for the MFCC only testing condition.

Similarly, the reuslts in 2.9 show that the behaviour of the MAEs is very similar under the randomly degraded training procedure regardless of where it is MFCCs or Images being used as input. However, we see that classification accuracy remains reasonably good under the bimodal training procedure when only MFCCs are present.

2.5.8 Generative Results

In figures 2.2 and 2.3 we see examples of digits generated by both the Addition and Concatenation MAEs.

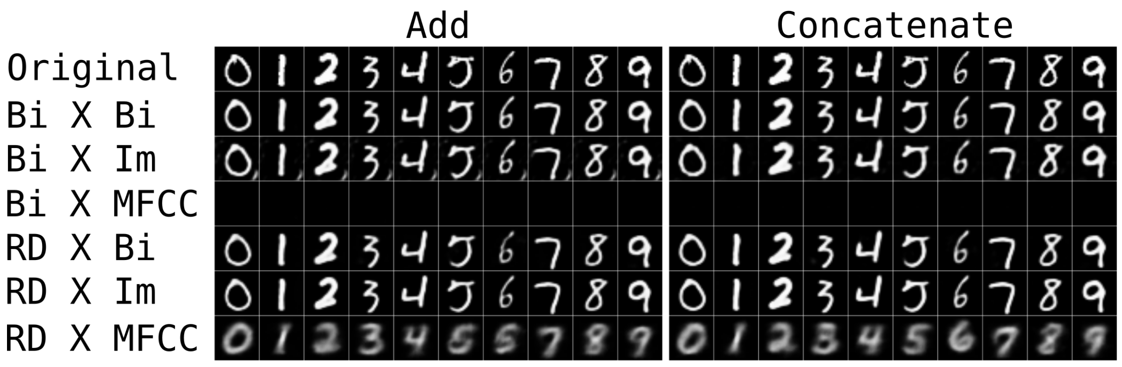

FIGURE 2.2: A selection of randomly sampled digits generated in different training and testing conditions for both *Add* and *Concat* merging methods.

Figure 2.3 shows a comparison of two different examples of the digit "5". One is (subjectively) poorly written and the other is more prototypical and well formed. We see that under different testing conditions, the MAEs regenerate different "fives".

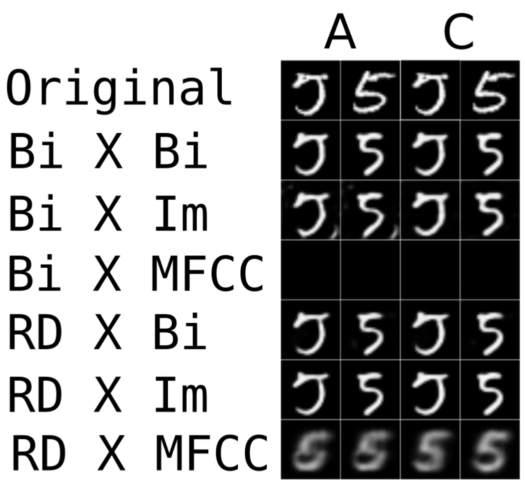

FIGURE 2.3: Two examples of the digit 5 being generated in different training and testing conditions for both *Add* and *Concat* merging methoids.

2.5.9 Discussion

2.5.9.1 Discussion of Classification Results

Looking at table 2.7 we see that both the Add and Concatenate MAEs have better classification accuracy than the baseline, unimodal autoencoder models. However, it is not clear that there is a difference in between the two training procedures, Bimodal and Randomly Degraded.

The difference between the two training procedures becomes clear when we look at table 2.8 where we see that the MAEs trained using the Bimodal training procedure have much worse classification rates when presented with only images as input, performing less than half as well as the baseline model, 0.4501 and 0.3496 for the Addition and Concatenation MAEs respectively versus 0.9832 for the baseline unimodal image autoencoder.

We also start to see a trade off between unimodal accuracy and mulitmodal accuracy when we look at the Randomly Degraded training procedure MAEs. Whilst both the Addition and Concatenation MAEs outperform the baseline models in the bimodal test condition, they have slightly worse performance when tested in either a unimodal image or MFCC manner. For example, the best performance is given by the Addition MAE under the Randomly Degraded training procedure when both images and MFCCs are present as inputs, with an accuracy of 0.9993. However, if one of the modalities is removed, the accuracy drops to 0.9834 and 0.9789 for image only and MFCC only testing respectively. Comparing this to the baseline unimodal autoencoders, we see classification accuracies of 0.9883 and 0.9832 for image and MFCC autoencoders respectively.

This means there is a drop of 0.0049 and 0.0043 for image only and MFCC only testing using the Addition MAE. However, this reduction in performance is conterbalanced by the improvement in performnace when both modalities are present, 0.9993 versus the 0.9883 and 0.9832 of the baseline models. This improvement of 0.0110 to 0.0161 over the baseline image and MFCC autoencoders is more than twice the decrease in performance seen when only one modality is available.

Specifically, the improvement of bimodal classification is 2.2449 times larger than the decrease in performance when only images are available ($0.011/0.0049 = 2.2449$). Comparing to the MFCC baseline, the bimodal performance improvement is 3.7442 times larger than the decrease in performance when only MFCCs are available.

2.5.9.2 Discussion of Generation Results

Observing tables 2.7, 2.8 and 2.9 we can see that different models had different reconstruction losses, Mean Squared Errors. Whilst all MAE models had higher reconstruction losses than the baseline models, this is to be expected. The MAE models are trading off between representing each modality well, whilst the unimodal autoencoders that give the baseline numbers can use their full capacity to minimise the reconstruction loss of their modaility as they do not have to reconstruct the second modality.

That being siad, the image reconstruction loss, is comparable across all MAE models, approximately 0.0031, (see table 2.7 and is very close to the baseline of 0.0027 under the bimodal testing condition (i.e. when the images to be reconstructed are available as input). Further to this, the image reconstruction loss for the MFCC only testing condition is still quite low, though an order of magnitude worse, for the MAEs trained using the Randomly Degraded procedure.

Similarly for the reconstruction of MFCCs, the reconstruction loss is best for the baseline model but is only 4.0265 times worse for the best performing model on this task. Considering how small the baseline construction error is, being four times worse, is still good performance, especially considering the difficulty of the task.

The MFCC reconstruction error is an order of magnitude worse than the image reconstruction error. In part this is due to the nature of the data, as seen by the difference in baseline reconstruction errors. I.e. it appears to be more difficult to reconstruct MFCCs than images, even when MFCCs are given as input. Another contributing factor to the worse MFCC reconstruction error for the MAEs, is the inbalance between sample numbers for MNIST and UCU. Whilst both datasets are balanced within themselves (there are an equal number fo training smaples for each class) there is a big difference between the sizes of MNIST and UCU as discussed previously in section **??**.

The difference in sizes between the datasets means that reconstruction MFCCs from image inputs will map a large number of inputs to a smaller number of outputs and vice-versa for generating images from MFCC inputs. Images in the test set which look similar to images in the training set, will be mapped to MFCCs similar to those which the training images map too. As there are a smaller number of MFCC examples to map to, our output is less Gaussian. Whilst idealy we would like to map similar looking digits to similar sounding utterances, there is no gaurantee that this is true, either in our dataset or in general. I.e. People with similar handwritting don't necessarily have similar voices. This is simulated by the random assignment of images to MFCCs but this results in discontinuities in the latent space of the network, particularly with respect to generating MFCCs, of which we have fewer examples to capture the true distribution of the data.

Conversely, as there are more examples of images of digits, the distribution of the latent space with respect to the images can get closer to modelling the real distribution of the images. Therefore, given a test MFCC, regardless of whether it is similar to a training MFCC example, its target image, is more likely to be similar to images in the training data.

Figure 2.2 we can clearly see that randomly degrading the training data has a significant effect, beyond just the improvements to classification accuracy seen in tables 2.7, 2.8 and 2.9. Both the Addition and Concatenation MAEs fail to learn to generate images from MFCC data when only bimodal data is presented as training data.

Comparing the images generated by the Add and Concatenate MAEs for the MFCC test condition when the training data has been randomly degraded, we see that the concatenate MAE outperforms the Addition MAE. This is particularly clear when you loom at the 5 and 6 generated by the Addition MAE for RD X MFCC, where the MAE has failed to correctly generate a six and has instead generated a five, which is similar to the prototypical five generated when MFCCs for a five are presented. Unlike the Addition MAE, the Concatenation MAE, correctly generates all digits, including six.

In figure 2.3 we see that different examples of the same digit produce different generation results. Most interestingly however, is that the generated fives shown in the RD X MFCC row, you can see that a more prototypical five is generated from MFCCs than if an image of a non-prototypical five is given.

2.6 Conclusion

Whilst multimodal systems can provide better classification accuracy, this comes at the cost of higher computational costs and lower classification accuracy when only one modality is available as demonstrated by the results presented in this chapter.

Bibliography

Barsalou, L. W. (2008). Grounded cognition. *Annu. Rev. Psychol.*, 59:617–645.

Hammami, N. and Bedda, M. (2010). Improved tree model for arabic speech recognition. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 5, pages 521–526. IEEE.

Hammami, N. and Sellam, M. (2009). Tree distribution classifier for automatic spoken arabic digit recognition. In *2009 International Conference for Internet Technology and Secured Transactions,(ICITST)*, pages 1–4. IEEE.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Sheppard, E., Lehmann, H., Rajendran, G., McKenna, P. E., Lemon, O., and Lohan, K. S. (2018). Towards life long learning: Multimodal learning of mnist handwritten digits. *IEEE ICDL EPIROB 2018 Workshop on Life Long Learning.*