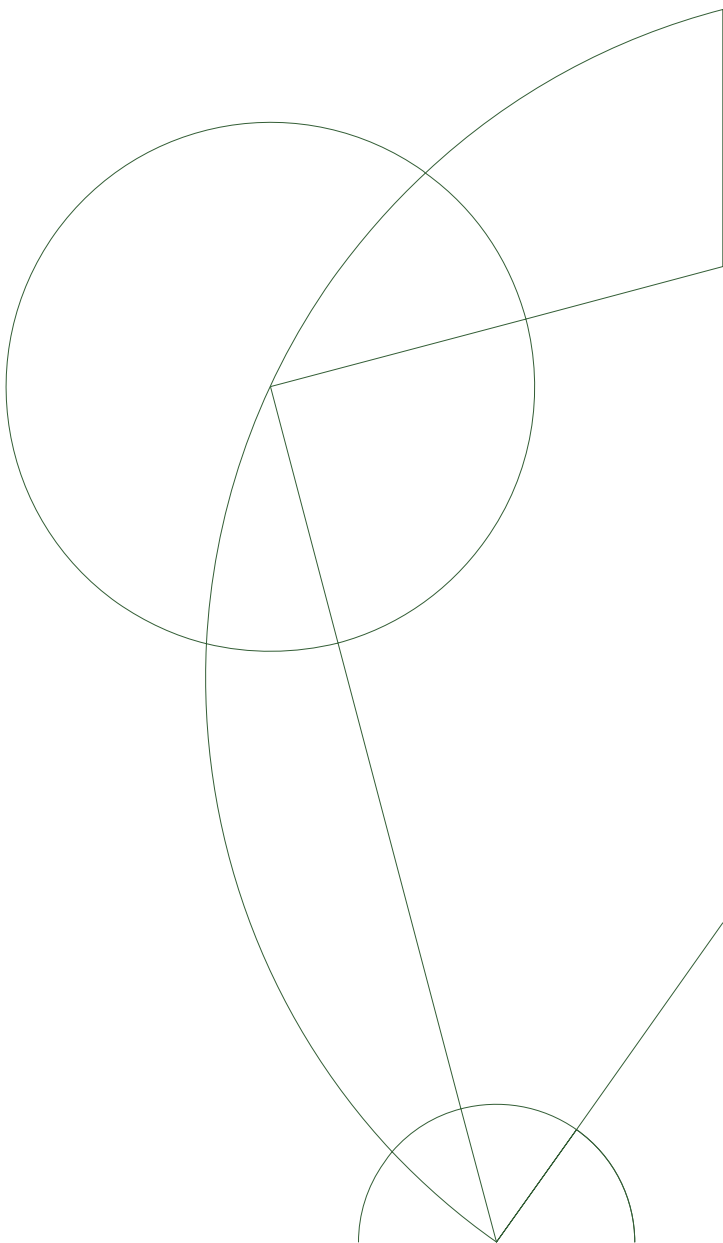
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master Thesis

Sune Andreas Dybro Debel

Deep Multi-Task Learning

For Relation Extraction



Dirk Hovy

March 5, 2017

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

Part 1

Introduction

1.1 Motivation

1.2 Problem Statement

Part 2

Background

In this part, we describe the information extraction problem and the challenges it poses. Moreover, we formally describe the supervised machine learning setting. Specifically, we discuss the challenges of noise and overfitting, and show the usefulness of Vapnik-Chervonenkis analysis.

2.1 Information Extraction

In natural language processing, information extraction is the problem of extracting structured information from unstructured text. Many practical information extraction problems fall in one of two categories: **named entity recognition**, or **relation extraction** (Jurafsky and Martin, 2009). Here, we introduce each of them, and explain why they are difficult.

2.1.1 Named Entity Recognition

A named entity is roughly anything that has a proper name. The task in named entity recognition is to label mentions of entities such as people, organisations or places occurring in natural language. As an example, consider the sentence:

Jim bought 300 shares of Acme Corp. in 2006.

A named entity recognition system designed to extract the entities *person* and *organisation* should ideally assign the labels:

$[\text{Jim}]_{person}$ bought 300 shares of $[\text{Acme Corp.}]_{organisation}$ in 2006.

This is a difficult problem because of two types of ambiguity. Firstly, two distinct entities may share the same name and category, such as *Francis Bacon* the painter and *Francis Bacon* the philosopher. Secondly, two distinct entities can have the same name, but belong to different categories such as *JFK* the former American president and *JFK* the airport near New York.

Named entity recognition can be framed as a sequence labelling problem. A common approach is to apply so called tokenisation to the text, i.e finding boundaries between words and punctuation, and associate each token with a label indicating which entity it belongs to. BIO (figure 2.1) is a widely used labelling scheme in which token labels indicate whether the token is at the **B**eginning, **I**nside, or **O**utside an entity mention.

Jim	bought	300	shares	of	Acme	Corp	.	in	2006	.
B-PER	O	O	O	O	B-ORG	I-ORG	I-ORG	O	O	O

Figure 2.1
A sentence labeled with the BIO labels for named entity recognition.

2.1.2 Relation Extraction

Relation extraction refers to the problem of identifying relationships such as *Family* or *Employment* between entities. As an example, consider the sentence:

Yesterday, New York based Foo Inc. announced their acquisition of Bar Corp.

Imagine we have designed a relation extraction system that recognises the relation *MergerBetween(organisation, organisation)* between two mentions of organisations. Ideally, we would like that system to extract the relation *MergerBetween(Foo Inc., Bar Corp.)* from the above sentence.

Because relation extraction is concerned with relationships between named entities, many systems that perform relation extraction applies named entity recognition first as a pre-processing step. This approach is sometimes called **pipelining**. An alternative to pipelining is **end-to-end** relation extraction, where relations and entities are extracted simultaneously. Pipeline approaches can suffer from the problem of **error propagation**, where the system erroneously assigns a label in the named entity recognition step, which later causes it to make an error in the relation extraction step.

2.1.3 Accuracy Measures

Information extraction systems are often evaluated empirically by applying them to collections of text, so called corpora, in which N mentions of named entities or relations are known. Accuracy measures used in such tests are usually defined in terms of:

True positives (tp) The number of true named entities or relations correctly labeled by the system.

True negatives (tn) The number of true non-entities or non-relations correctly labeled by the system.

False positives (fp) The number of true non-entities or relations incorrectly labeled by the system.

False negatives (fn) The number of true named entities or relations incorrectly labeled by the system.

The distribution of labels used in both named entity recognition and relation extraction is often highly imbalanced. Consider for example the BIO labelling scheme in figure 2.1. Most words will be outside a mention of a named entity, and will have the label `O`. Using simple accuracy $\frac{tp+tn}{N}$ as a performance metric is therefore not very informative, since a useless system which labels all tokens with `O` would achieve high performance.

Precision and **recall** are more appropriate performance metrics for this reason. Precision $\frac{tp}{tp+fp}$ is the fraction of true named entities or relations of all named entities or relations that were extracted by the system. This is equal to 0 when none of the information extracted by the system was correct and 1 when all of it was correct.

Recall $\frac{tp}{tp+fn}$ is the fraction of true named entities or relations that were extracted by the system. This is 0 when none of the extracted information was correct, and

1 when all of the extracted information was correct, and no true named entities or relations were incorrectly labeled.

To get a single number that summarises the performance, precision p and recall r are often combined into a single metric, the $F1$ measure, defined as the harmonic mean of precision and recall $\frac{2pr}{p+r}$.

2.2 Supervised Machine Learning

Most modern solutions to the information extraction problems in 2.1 are based on supervised machine learning techniques. In this setting, a system learns to recognise the named entities or relations between them from examples provided by a human annotator. In this section we formally describe this approach and introduce important theoretical tools for understanding supervised machine learning.

2.2.1 The Supervised Learning Problem

A set $\mathcal{D}_{train}$ of N training examples $(\mathbf{x}_i, \mathbf{y}_i)$ of inputs $\mathbf{x}_i$ and corresponding labels $\mathbf{y}_i$ is created by a human annotator. Each $\mathbf{x}_i$ belongs to an input space $\mathcal{X}$, for example the set of all english sentences. Each $\mathbf{y}_i$ belongs to a space $\mathcal{Y}$ of labels, for example the set of all sequences of BIO tags. As designers of the learning system, we specify the so called **hypothesis space** $\mathcal{H}$, a set of functions $h : \mathcal{X} \mapsto \mathcal{Y}$. We want to find a function $h \in \mathcal{H}$, sometimes called a **model** or **hypothesis**, that can automatically assign labels to a new set of un-labeled inputs $\mathcal{D}_{test} = \{\mathbf{x}_i \mid \mathbf{x}_i \in \mathcal{X}\}$ at some point in the future.

Supervised machine learning is the science of how to use an algorithm to find a function h using $\mathcal{D}_{train}$ that performs well on $\mathcal{D}_{test}$, as measured by some performance measure e. In classification problems such as named entity recognition or relation extraction where $\mathcal{Y}$ is discrete, we typically use binary error $e(\mathbf{y}_1, \mathbf{y_2}) = \mathbb{I}[\mathbf{y}_1 \neq \mathbf{y}_2]$. Importantly, we are not explicitly interested in the performance of h on $\mathcal{D}_{train}$ (Abu-Mostafa et al., 2012).

We can formalise the preference for functions h that perform well on examples outside of the training set with a quantity known as **generalisation error**.

Definition 2.2.1 (generalisation error)**.** Let $P(\mathbf{x}, \mathbf{y})$ be a joint probability distribution over inputs $\mathbf{x} \in \mathcal{X}$ and labels $\mathbf{y} \in \mathcal{Y}$. Let $e(\mathbf{y}_1, \mathbf{y_2}) = \mathbb{I}[\mathbf{y}_1 \neq \mathbf{y}_2]$ be the binary error function that measures agreement between labels $\mathbf{y}_1$ and $\mathbf{y}_2$. Then the generalisation error E of a function $h : \mathcal{X} \mapsto \mathcal{Y}$ is defined as:

$$E(h) = \mathbb{E}_{\mathbf{x},\mathbf{y} \sim P(\mathbf{y},\mathbf{x})}[e(h(\mathbf{x}), \mathbf{y})]$$

Now, formally, the objective of supervised machine learning is to find a function h^* in a space of functions $\mathcal{H}$ that minimises $E(h)$. We see the process generating the data as random, but with a behaviour describable by a distribution $P(\mathbf{x}, \mathbf{y})$. Unfortunately, this distribution is unknown, which makes E unknown. However, we can use sampled data $\mathcal{D} = \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \sim P(\mathbf{x}, \mathbf{y})\}$ to estimate $E(h)$ with a quantity known as **empirical error**:

Definition 2.2.2 (empirical error)**.** Let $\mathcal{D}$ be a set of N examples $\{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i, \mathbf{y}_i \sim P(\mathbf{x}, \mathbf{y})\}$. Then the empirical error $\hat{E}$ is defined as:

$$\hat{E}(h, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} e(h(\mathbf{x}_i), \mathbf{y}_i)$$

Because $\mathcal{D}$ is a random quantity, it's dangerous to use $\hat{E}$ to estimate E. We risk that the samples are not representative of $P(\mathbf{x}, \mathbf{y})$, leading us to believe that h is great,

when in fact it's terrible. We can bound the probability that $\hat{E}$ is a bad estimate of E if we make two assumptions:

Firstly, we assume that the samples in $\mathcal{D}$ are drawn independently from $P(\mathbf{x}, \mathbf{y})$, that is observing any one sample did not change the probability of observing any other sample.

Secondly, we assume that h is independent of $\mathcal{D}$, in other words, that h was not specifically chosen based on the sample. These assumptions enable us to apply **Hoeffding's inequality** to bound the probability that $\hat{E}$ is far away from E:

Theorem 2.2.1 (Hoeffding's inequality)**.** let $E(h)$ be defined as in definition 2.2.1, and let $E(h, \mathcal{D})$ be defined as in definition 2.2.2. Then:

$$\mathbb{P}\left(|E(h) - \hat{E}(h, \mathcal{D})| \geq \epsilon\right) \leq 2e^{-2N\epsilon^2}$$

The inequality tells us that the probability that E is more than ϵ away from $\hat{E}$ decreases exponentially in ϵ and N. In other words, the more samples in D, the less likely it is that E will be misleading.

Estimating E with a sample that is independent of h is a technique called **validation**. In validation, the sample provided by a human annotator is split into two datasets, $\mathcal{D}_{train}$, which we intend to use to search for h^*, and $\mathcal{D}_{validate}$, which saved until we are done searching. Since $\mathcal{D}_{validate}$ is independent of whichever h we selected, Hoeffding's inequality applies and $\mathcal{D}_{validate}$ can be used to estimate E.

Because $\mathcal{D}_{train}$ is used to select h, it cannot be used to estimate E by Hoeffding's inequality, and we need more sophisticated techniques to understand the relationship between $\mathcal{D}_{train}$ and E. The central question in supervised machine learning is *how can we best define $\mathcal{H}$ and use $\mathcal{D}_{train}$ to make E small?* Answering this question is the objective of a field of research known as **statistical learning theory**.

2.2.2 Statistical Learning Theory

We would like to know how best to define $\mathcal{H}$ and use $\mathcal{D}_{train}$ in order to make E small. $\mathcal{D}_{train}$ is the only information we have about $P(\mathbf{x}, \mathbf{y})$, and therefore also the only information we have about E. A straight-forward idea would be to find a function $g \in \mathcal{H}$ that minimises the **training error** $\hat{E}(g, \mathcal{D}_{train})$ in the hope that g will also minimise E.

As we argued in section 2.2, using $\hat{E}$ to estimate E can be misleading. Moreover, because D_{train} is used to specifically choose g that makes $\hat{E}$ small, the guarantees provided by Hoeffding's inequality no longer holds, and therefore it may be possible to select g such that $\hat{E}(g, \mathcal{D}_{train})$ is small and $E(g)$ is large, even when we have a large number of training examples.

The phenomena where training error is small but generalisation error is large is known as **overfitting**. As the name implies, it's caused by harmful idiosyncrasies of $\mathcal{D}_{train}$ that, when used to minimise $\hat{E}(h, \mathcal{D}_{train})$, leads us to a g with a larger E than other functions in $\mathcal{H}$. These idiosyncrasies of $\mathcal{D}_{train}$ are ultimately the product of **noise**.

In general, noise comes in two forms. The first form is known as **stochastic noise**. This type of noise is introduced by variation in the relationship between $\mathbf{x}$ and $\mathbf{y}$ that is inherently unpredictable. For example, human error is a common source of stochastic noise in information extraction, where an annotator incorrectly labels a piece of text. Selecting a g that repeats this error is a case of overfitting, because g will have lower training error but larger generalisation error than another h that doesn't predict the incorrect annotation, since presumably the error is the exception to the rule.

The second type of noise is called **deterministic noise**. This type of noise may be introduced when the relationship between $\mathbf{x}$ and $\mathbf{y}$ is deterministic, but $\mathcal{H}$ doesn't have the capacity to represent this relationship exactly.

To understand deterministic noise, imagine that even h^* can't represent the deterministic relationship $\mathbf{y} = f(\mathbf{x})$ exactly. Suppose that we get a $\mathcal{D}_{train}$ that contains a sample $(\mathbf{x}_i, \mathbf{y}_i)$ that falls outside the capacity of h^*, that is, $h^*(\mathbf{x}_i) \neq \mathbf{y}_i$. Now further imagine that in order to minimise $\hat{E}$, we select a g that predicts this sample, such that $h(\mathbf{x}_i) = \mathbf{y}_i$. This is a case of overfitting since we know that there is at least one function in $\mathcal{H}$ with lower generalisation error than g, namely h^*.

The risk of overfitting is linked to the diversity of $\mathcal{H}$. By diversity of $\mathcal{H}$, we roughly mean how different any function in $\mathcal{H}$ is from any other function in $\mathcal{H}$. The more diverse $\mathcal{H}$ is, the greater the risk that there exists a $h \in \mathcal{H}$ that will overfit $\mathcal{D}_{train}$.

A **dichotomy** is a central concept in measuring the diversity of $\mathcal{H}$. A dichotomy is a specific sequence of N labels. For example, if $\mathcal{Y} = \{0, 1\}$, and $N = 3$, then (0 1 0) is a dichotomy, and so is (1 0 0). We have listed all dichotomies for $N = 3$ in figure 2.2.

(0 0 0)
(1 0 0)
(0 1 0)
(0 0 1)
(1 1 0)
(0 1 1)
(1 0 1)
(1 1 1)

Figure 2.2
All dichotomies for $\mathcal{Y} = \{0, 1\}$ and $N = 3$. There are $2^3 = 8$ ways to choose a sequence of 3 labels from 2 possibilities.

Dichotomies allow us to group similar functions. In the rest of this section, let's assume that $\mathcal{Y} = \{0, 1\}$. By simple combinatorics the number of dichotomies for N must be smaller than or equal to 2^N. There may be infinitely many functions in $\mathcal{H}$, but on a specific $\mathcal{D}_{train}$, many of them will produce the same dichotomy since the number of training examples in $\mathcal{D}_{train}$ is finite. This allows us to quantify the diversity of $\mathcal{H}$ in terms of the number of dichotomies it's able to realise on a set of N points. This is achieved by a measure known as the **growth function**.

Definition 2.2.3 (growth function)**.** Let $\mathcal{H}(N) = \{(h(\mathbf{x}_1), \ldots, h(\mathbf{x}_N)) \mid h \in \mathcal{H}, \mathbf{x}_i \in \mathcal{X}\}$ be the set of all dichotomies generated by $\mathcal{H}$ on N points, and let $|\cdot|$ be the set cardinality function. Then the growth function m is:

$$m(N, \mathcal{H}) = \max |\mathcal{H}(N)|$$

In words, the growth function measures the maximum number of dichotomies that are realisable by $\mathcal{H}$ on N points. To compute $m(N, \mathcal{H})$, we consider any choice of N points from the whole input space $\mathcal{X}$, select the set that realises the most dichotomies and count them.

The growth function allows us to account for redundancy in $\mathcal{H}$. If two functions $h_i \in \mathcal{H}$ and $h_j \in \mathcal{H}$ realise the same dichotomy on $\mathcal{D}$, then any statement based only on $\mathcal{D}$ will be either true or false for for both h_i and h_j. This makes it possible to group the events $\hat{E}(h_i, \mathcal{D})$ *is far away from* $E(h_i)$ and $\hat{E}(h_j, \mathcal{D})$ *is far away from* $E(h_j)$, and thereby avoiding to overestimate the probability of the union of both events occurring.

If $\mathcal{H}$ is infinite, the number of redundant functions in $\mathcal{H}$ will also be infinite, since the number of dichotomies on N points is finite. If $m(N, \mathcal{H})$ is much smaller than 2^N, the number of redundant functions in $\mathcal{H}$ will be so large as to make the probability that $\hat{E}$ is far away from E very small.

This line of reasoning is the basis of the Vapnik-Chervonenkis bound, which bounds $E(h)$ in terms of $\hat{E}(h, \mathcal{D}_{train})$:

Theorem 2.2.2 (Vapnik-Chervonenkis bound)**.** Let $m(N, \mathcal{H})$ be defined as in definition 2.2.3, $E(h)$ as 2.2.1, and $\hat{E}(h, \mathcal{D})$ as in 2.2.2. Then, with probability $1 - \delta$:

$$E(h) \leq \hat{E}(h, \mathcal{D}_{train}) + \sqrt{\frac{8}{N} \ln \frac{4m(2N, \mathcal{H})}{\delta}}$$

The bound tells us that $E(h)$ will be close to $\hat{E}(h, \mathcal{D}_{train})$ if $m(N, \mathcal{H})$ is small and N is large. Intuitively, this tells us that a set $\mathcal{H}$ that contains "simple" functions will make it easier to choose g such that generalisation error will be close to training error, where simple means: functions that realise a small number of dichotomies.

On the other hand, having a set $\mathcal{H}$ that can realise a large number of dichotomies on N points, will make it easier to find a function that will make $\hat{E}(h, \mathcal{D}_{train})$ small. Using a $\mathcal{H}$ with functions that are too simple is called **underfitting**. It occurs when we search for a function in the set of functions $\mathcal{H}$, when there is another, more diverse set of functions $\mathcal{G}$ which contain a function with lower generalisation error.

This analysis tells us that an optimally diverse $\mathcal{H}$ balances the tradeoff between the risk of overfitting, represented in the bound by m, and the risk of underfitting, represented by $\hat{E}$. In practice, underfitting is less of a problem than overfitting, since modern supervised machine learning algorithms search in extremely diverse spaces of functions $\mathcal{H}$. In fact, most $\mathcal{H}$ are so diverse that steps must be taken to avoid minimising $\hat{E}$ as much as is actually possible. These techniques are known as **regularisation**, which we will see an instance of in section 3.2.3.

2.3 Summary

In this section we have seen that the purpose of named entity recognition is to identify mentions of entities such as people, organisations and places in natural language. The purpose of relation extraction systems is to identify relationships between them.

We have seen that simple accuracy is uninformative as an evaluation measure in information extraction, and described the alternative precision and recall.

We have described the formal setting of on supervised machine learning. We have discussed concepts such as overfitting and noise, diversity of the set of functions $\mathcal{H}$ from which to choose h, and its impact on training and generalisation error.

Part 3

Neural Networks

In this part we describe a way to define $\mathcal{H}$ using functions called **neural networks**. These functions have the advantage of being easy to adapt to multi-task learning. We begin by describing how to design a $\mathcal{H}$ with neural networks. We then turn to the issue of how to use $\mathcal{D}_{train}$ to search this hypothesis space. Lastly, we introduce specialised neural networks that are useful for natural language processing.

3.1 Feed-Forward Neural Networks

A feed-forward neural network is a function $h(\mathbf{x})$. To understand how it works, it's instructive to look at each part of its name in isolation.

h is called a **network** because it's a composition of L **layers** of other functions f_l. Each f_l receives input from f_{l-1}. For example if $L = 2$, then $h(\mathbf{x}) = f_2(f_1(\mathbf{x}))$. The number of layers L is called the **depth** of the network. f_L is called the **output layer**. The remaining functions f_1 to f_{L-1} are called **hidden layers**. Each f_l outputs a vector $\mathbf{z}_l$ of dimension d_l. The size of these vectors determine the **width** of the network. We denote the input to f_1 as $\mathbf{z}_0$, which is the input $\mathbf{x}$ to $h(\mathbf{x})$ with an added **bias** component, as described later in this section.

The functions f_1 to f_L are ordered by their index l. By ordered, we mean that the index of the innermost functions are smaller than the index of the outermost. h is called a **feed-forward** network because each f_l can receive input only from functions f_i if $l > i$. In other words, its not possible for a function f_l to feed its own output into itself, or any other function that it receives input from. Networks where this restriction is removed are called **recurrent neural networks**, and will be discussed in section 3.4.

Finally h is called a **neural** network since its design is loosely based on neurons in the brain (Goodfellow et al., 2016). Each component z of the vector $\mathbf{z}_l$ can be seen as the output of a unit similar to a neuron. Each unit in layer l receives input from units in layer $l-1$. The output $z_{l-1,v}$ of unit v in layer $l-1$ is multiplied by a weight $w_{l,v,u}$ that gives the strength of the connection between unit v in $l-1$ and unit u in l. Unit u sums all of the input it receives from units in layer $l-1$ to obtain its **activation** $a_{l,u} = \sum w_{l,v,u} z_{l-1,v}$. To compute its output $z_{l,u}$, it applies an **activation function** $\sigma(a_{l,u})$ to its activation.

Activation functions model the behaviour of biological neurons by outputting a signal only when the activation is above a certain threshold. To make it possible to learn this threshold for each unit using the same activation function, we introduce a special **bias** unit that always outputs 1. The index of the bias unit in layer l is 0 by convention. Figure 3.1. shows how a unit u computes its output $z_{l,u}$ by combining

the outputs of units in layer $l-1$.

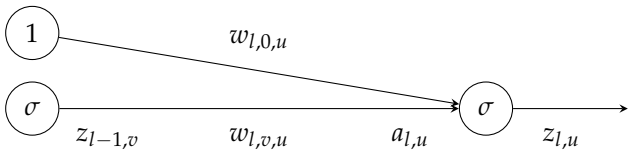


Figure 3.1
A visual representation of the connections between unit v in layer $l-1$, the bias unit in $l-1$, and unit u in layer l. The connection strength between these units is given by the weight $w_{l,v,u}$ between v and u, and $w_{l,0,u}$ between the bias unit and u. The activation $a_{l,u}$ at unit u is computed by $a_{l,u} = w_{l,v,u}z_{l-i,v} + w_{l,0,u}$. The output $z_{l,u}$ of unit u is given by $z_{l,u} = \sigma(a_{l,u})$

Keeping track of the indices l, u and v quickly becomes confusing. By collecting all of the weights of connections going into unit u in layer l in a vector $\mathbf{w}_{l,u}$, the activation at unit u can be computed as a dot product $a_{l,u} = \mathbf{w}_{l,u}{}^T\mathbf{z}_{l-1}$. Moreover, we can compute the entire vector $\mathbf{a}_l$ of activations at layer l, by organising the weight vectors $\mathbf{w}_{l,u}$ in a matrix $\mathbf{W}_l = [\mathbf{w}_{l,1} \ldots \mathbf{w}_{l,d_l}]^T$, which leads to $\mathbf{a}_l = \mathbf{W}_l\mathbf{z}_{l-1}$.

By gathering the weights in matrices $\mathbf{W}_l$, we have simplified our view of h into a composition of matrix-vector products and element-wise application of activation functions. Figure 3.2 shows the parallel views of neural networks as networks of units and matrix-vector operations.

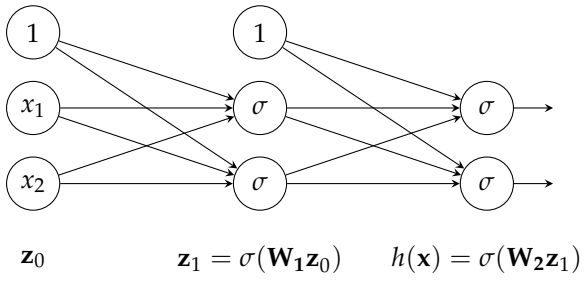


Figure 3.2
A visual representation of $h(\mathbf{x}) = f_2(f_1(\mathbf{x}))$. The activation at each layer $\mathbf{a}_l$ is computed by $\mathbf{W}_l\mathbf{z}_{l-1}$. The output at each layer is computed by element-wise application of the activation function of $\sigma(\mathbf{a}_l)$.

We now have all the components we need to specify $\mathcal{H}$ as a set of neural networks. The set is defined by the depth of the networks L, the number of units in each layer d_l , and the activation function σ. For a particular L, d_l, and σ, each $h \in \mathcal{H}$ is given by the set of all its weights $\mathcal{W} = \{\mathbf{W}_1 \ldots \mathbf{W}_L\}$. We sometimes make the dependence of h on $\mathcal{W}$ explicit by using the notation $h(\mathbf{x}; \mathcal{W})$ which means *the function h parameterised by $\mathcal{W}$*. In the next section we discuss how to choose the activation functions at the layers of the network.

3.1.1 Activation Functions

3.1.2 Objective Function

3.2 Learning Algorithm

3.2.1 Gradient Descent

3.2.2 Backpropagation

3.2.3 Regularisation

3.2.4 Adam

3.3 Convolutional Neural Networks

3.4 Recurrent and Recursive Neural Networks

3.4.1 LSTM

Part 4

Multi-Task Learning

4.1 Representation Learning

4.1.1 Auto Encoders

4.1.2 Word Embeddings

4.2 Deep Multi-Task Learning

Part 5

Experiment

5.1 Auxiliary Tasks

5.2 Network Architecture

Part 6

Results

Part 7

Discussion

Part 8

Conclusion

Bibliography

Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLbook.com, 2012.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, first edition, 2016.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Education, international edition, 2009.

Appendix A

Appendix

A.1 First