Locally Competing Neural Networks

Anna Hjertvik Aasen, Nanna Bryne, Carl Martin Fevang, Johan Mylius Kroken, and Håkon Kvernmoen

December 16, 2022

Abstract

We integrated two LWTA algorithms, maxout and channel-out, with `tensorflow` neural networks and applied them to the MNIST and CIFAR-10 image recognition datasets. We visualised how these algorithms train specific pathways in its architecture, providing an opportunity to discern how information is encoded in the neural networks. A maxout neural network including dropout achieved a test accuracy of 51.28% on the CIFAR-10 dataset, and a test accuracy of 50.69% with a channel-out network with L2 weight penalisation.

Furthermore, we created and analysed a dataset comprised of statistics from English Premier League football matches, and applied LWTA NNs in classifying results of the final matches 124 matches of the 2019/2020 season. With a maxout NN with dropout and L2 weight penalisation we got an accuracy of 58.06%, and with a plain channel-out NN we got 54.84%. Neither of these beat an ordinary dense NN with ReLU activation achieving an accuracy of 58.87%. We performed PCA on the EPL data, which did not yield a satisfactory result, only halving the number of features in order to obtain 99 % explained variance. Further analysis of this was hence discarded.

Contents

1 Introduction

The root inspiration of neural networks was trying to mimic the way biological systems encode and process sensory information. As the field of neuroscience has advanced, it has again become source of inspiration for improving upon the existing neural network (NN) frameworks. In this project, we want to explore the biological phenomenon of local competition among neurons to encode information by looking at *local winner-takes-all* (LWTA) algorithms for neural networks. We go on to present two algorithms for LWTA networks: *Maxout* networks and channel-out networks. The former implements the idea of local competition between neurons, and the latter build on the idea of encoding information on neural pathways between the layers.

Winner-takes-all behaviour is well documented in visual perception, and as such it would be natural to try our algorithms against image recognition datasets. Thus, we apply our networks to the benchmark MNIST and CIFAR-10 datasets. In doing so, we aim to explore the architecture of LWTA NN and how information is encoded.

Additionally, we want to build our own dataset, both for analysis and for applying LWTA NNs. Inspired by current events, we build a dataset based on statistics from the 2019/2020 Premier League football season, seeing whether neural networks can use data from previous games and seasons to predict the outcome of individual matches.

As well as for many other sports, data science is becoming more and more influential in the football industry [Herbinet, 2018], both from the viewpoint of the club, e.g. in terms of match strategy or player acquisition, and as seen by the odds market or the solely economically interested parties, be it club owners or football associations, for making sure they benefit from the industry by possessing valuable data and powerful prediction algorithms. Hence, building a dataset and investigating this is of great interest, albeit on an introductory level. We aim to perform principal component analysis on this dataset, in order to investigate its features. The final aim is to predict the outcome of games, and validate the model's accuracy of doing so.

Complementary material to this report can be found in our GitHub repository at `https://github.com/hkve/FYS-STK4155/tree/main/Project3`.

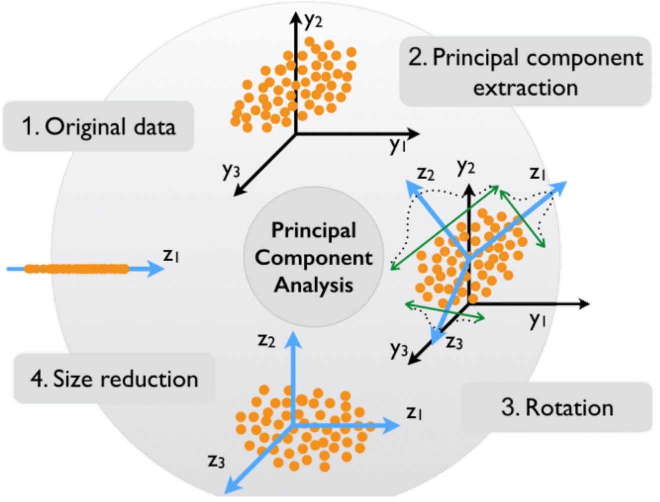

1. Original data
2. Principal component extraction
3. Rotation
4. Size reduction
Principal Component Analysis
y1
y2
y3
z1
z2
z3

Figure 1: Schematic representation of PCA. New axes, represented as z in the figure are generated, forming a new basis and a new hyperplane onto which we project the data in order to maximise variance and minimise covariance.[Bellemans et al., 2018]

2 Theory

2.1 Principal Component Analysis (PCA)

2.1.1 Motivation and assumptions

Consider the collection of data X which is a $(n \times m)$ matrix, with n data points, and m features. We might find ourselves in a situation where some features are highly correlated. As a result, some features might be considered obsolete since the trend they describe can easily be described by other features, or combinations of other features. If this is the case, then X contains more features than is strictly necessary in order to explain the trends in the data. Principal component analysis is a technique where we make linear combinations of the original features in such a way that we can describe the data with as few as possible of these new features, effectively reducing the dimensionality of the problem, without loosing too much information about the data itself. By doing this we can perform the computations with a dataset of fewer dimensions, increasing the computational efficiency. We assume that we can describe the data by its variance only. This implies that the mean must be zero, i.e. the data must be in mean-deviation form. We also assume the data to be normally distributed, as a mean-centred normal distribution is the only distribution described by its variance alone.

From a geometrical point of view, this can be thought of as expressing the data in a different coordinate system, and PCA involves determining these new axes. Figure 1 shows a visual interpretation of this, where the black axes are the original ones, and we find the new blue axes with PCA. These are determined in such a way that the variance of the data is maximised in the new coordinate system, while minimising the covariance between features. We must impose this criterion since we want as few new features as possible, hence we do not want them to covary, but rather vary as much as possible.

2.1.2 Mathematical consideration

Our goal is to reduce the dimensionality of the data, thus we want to represent the data as a matrix Y with shape $(n \times d)$ where $d \leq m$. This indicates that we are after a linear transformation $T : \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times d}$ which we can define by $T(X) = XP = Y$, where P is an orthonormal matrix of shape $(m \times d)$, which we intend to find. P must fulfil our criterion on Y, which is high variance and low covariance. This can be done by diagonalising the covariance matrix of Y given by:

$$\begin{aligned} S_Y &= n_c Y^T Y \\ &= n_c (XP)^T (XP) \\ &= n_c P^T (X^T X) P, \end{aligned} \tag{1}$$

where $n_c = 1/(n-1)$ is the normalisation due to n data points.

With [Lay, 2016, ch. 5.3] we can show that any square symmetric matrix, like $X^T X$, can be expressed as EDE^T where the columns of E are the orthonormal eigenvectors of $X^T X$, and D is a diagonal matrix. Inserting this into equation 1 we obtain that $S_Y = n_c P^T EDE^T P$. We see that $P = E \implies S_Y = n_c D$,[1] since P is then orthonormal and thus $P^T = P^{-1}$.

Hence, choosing the columns of P to be the orthonormal eigenvectors of $X^T X$ yield a diagonal covariance matrix S_Y, exactly what we want. This link can also be made through the singular value decomposition (SVD). We can express any matrix X as $X = U\Sigma V^T$ [Lay, 2016, ch. 7.4], where V is a matrix containing the orthonormal eigenvector of $X^T X$. Without delving further into SVD, we see that P may also be found from the column vectors in V.

We conclude that the principal components of X can be expressed as $P = [\boldsymbol{p}_1, \dots \boldsymbol{p}_d]$, where $\boldsymbol{p}_1$ is the eigenvector of $X^T X$ that corresponds to the largest eigenvalue, $\boldsymbol{p}_2$ to the second largest and so on. This continues until we have found d principal components. Each $\boldsymbol{p}_i$ is by construction a linear combination of the original features and has shape $(m \times 1)$. The elements of PCA may then be summarised as follows:

[1] This is not necessarily trivial since we want P to have shape $(m \times d)$. E contains all the eigenvectors of $X^T X$ and has thus shape $(m \times m)$. This is easily surpassed by setting $d = m$ while showing this, and then reduce the dimension of P when we need it later. We also note that this works mathematically if we let E contain the d eigenvectors of $X^T X$ with the largest eigenvalues. Then D must be a $(d \times d)$ diagonal matrix, which is of similar shape as S_Y.

1. Make sure X has shape $(n \times m)$ and is in mean-deviation form.
2. Find P either from V or the eigenvectors of $X^T X$ directly.
3. Calculate $Y = XP$.
4. Find the diagonal covariance $S_Y = n_c Y^T Y$.

2.1.3 Interpretation

Now we have a recipe of finding the principal component P, but what do they really mean? We perform PCA in order to reduce the redundancy of features, to express the data with as few features/dimensions as possible without loosing too much information. We therefore make new features that are linear combinations of the original features, but with zero covariance. This means that when we "choose" the first principal component $\boldsymbol{p}_1$, we find the direction in feature space, along which the variance of the data is maximised. In other words, if we were to project all data points down on a line, $\boldsymbol{p}_1$ would be the line that gives the largest variance, i.e. yield as much information about the data as possible for a 1D data set. When finding the next principal component we repeat this process, but restrict ourselves to only "choosing" directions in feature space orthogonal to the previous.

Having d principal components means that we have found d directions in feature space, all with a maximised variance. These can be though of as a new basis for representing our data X. Then the matrix P can be interpreted as the change of basis matrix between X and Y. Y are the same data points, but expressed in the basis of the principal components. They are the data points projected onto a hyperplane set up by the vectors $\boldsymbol{p}_i$. This can also be inferred from figure 1. We also notice a rather neat consequence of the SVD, that finding the principal components of X equates to finding an orthonormal basis that span $\text{Row}(X)$. The i-th diagonal element of S_Y is the variance, σ_i^2 of the data along $\boldsymbol{p}_i$. The total variance is the sum over these diagonal elements, or the trace, $\text{tr}(S_Y)$.

2.2 Neural Networks

First, it will be wise to take a minute to recap some of the details of how a *feed-forward nerual network* (FFNN) is built and trained from Aasen et al. [2022b]. They are made up of sequential *layers* taking in the input from the previous layer and passing it on to the next one. There will always be an input layer, taking in the features x producing some observation y, and an output layer transforming the input from the second to last layer into an appropriate output. In classification problems, the outputs will usually be probabilities p_c between 0 and 1 for the input to produce an output in a given category

c. In regression, the output layer usually transforms the output to be a single number $\tilde{y}$ for the predicted value.

When passing the output from one layer as input to the next, a linear transformation is done. Denoting the output vector from layer ℓ as $\boldsymbol{a}^\ell$, the transformation produces a *response* $\boldsymbol{z}^\ell = W^\ell \boldsymbol{a}^{\ell-1} + \boldsymbol{b}^\ell$, where W^ℓ is a weight matrix with elements w^ℓ_{ij}, and $\boldsymbol{b}^\ell$ is called the bias.[2] The response vector is then passed through an activation function (which is generally non-linear) to produce the *activation* $\boldsymbol{a}^\ell = f(\boldsymbol{z}^\ell)$.

Now introducing some new terms, when referring to a node with activation a^ℓ, the corresponding weights W^ℓ will be referred to as the *incoming weights*, and $W^{\ell+1}$ as *outgoing weights*.

2.3 Neurological Background: Competing Neurons

The first artificial neural networks were inspired by biological neurons (cells mainly located in the brain); artificial nodes represent the neurons and their activation, the weights represent the synapses (connections between neurons), and biases address the threshold for activation of each neuron/node [Aasen et al., 2022b]. Since then, the development of neurobiologically inspired artificial intelligence has only increased and many other algorithms mimicking mechanisms of the brain have been employed to enhance machine learning performance. An up-and-coming and interesting field of research is the implementation of biological competative mechanisms.

First some important terminology. Biological neurons are often considered to be either activated or not activated; a binary state of either on or off. Neurons are activated by receiving sufficient signals from sensory receptors (analogous to input in ANNs) or other activated neurons. When an activated neuron passes on its signal to another neuron, that signal can be either *excitatory*, nudging the second neuron towards activation, or *inhibitory*, reducing the chances of activation. A series of activated neurons can be considered a pathway created in the brain, and there exists a consensus that information in the brain is represented, not by the signal within a cell (which is very similar for all cells and therefore convey little information), but rather through the signal's pathway between cells [Kandel et al., 2000]. It is hypothesised that some of these pathways arise through competition between adjacent neurons [Kohonen, 1982].

2.3.1 Lateral Inhibition

An example of competitive mechanisms that are present in neurobiological systems is a phenomenon referred to as *lateral inhibition*. It exhibits the local competitive selectivity between neurons and neural pathways. This is assumed to enhance the sense of contrast and is present especially in sensory procedures. The mechanism behind lateral inhibition is that the neurons receiving the most sensory input inhibit the activity of the neurons in close proximity (radius from 200 to 500 μm), making themselves the "winners" and the nearby neurons the "losers". This results in a winner-takes-all behaviour. For example, when neurons receive input from receptors of some external stimuli, the neurons connected to the most stimulated receptors will inhibit the nearby neurons connected to less stimulated receptors, making them less likely to be activated. This will lead to the outer part of the stimulus being toned down and the central stimulus appearing stronger in contrast. There exists both anatomical a physiological evidence supporting the theory of lateral inhibition [Kohonen, 1982] [Cohen, 2011].

2.3.2 Functionality of Local Competition

In addition to enhancing contrast there are other neurological functions, and even higher-level cognitive functions which can be modelled by implementing the concept of competition within the brain.

Competition between neurons is hypothesised to cause biological neural pathways to arise [Kohonen, 1982]. As previously mentioned, neural pathways are series of activated neurons. These pathways are important as they are considered a manner of conveying information in the brain; assuming information is in some way encoded in brain activity, it follows that simultaneous and/or constant activation of all neurons will convey no information, and that pathways are necessary [Chen, 2017]. When neurons compete, it will prevent neurons from all firing simultaneously and rather create sparse pathways.

Furthermore, the pathways are hypothesised to be essential in the brain for making useful distinctions between input, and develop so-called *feature selectivity* [Chen, 2017]. Assuming that a certain pathway encodes part of the input, called a feature,[3] the input will be decomposed and processed by various pathways. In that case neurons can be tuned to e.g., different orientations or spatial frequencies when processing a visual input.[4]

The immense amount of input the brain receives at any given moment cannot be processed; somehow the brain prioritises the relevant the input. One hypothesis is that the competition between neural pathway leads to only some of the information being processed; only the winning neural pathways are activated [Chen, 2017]. This hypothesis coincides with an intuitive understanding of the concept of prioritising, as competition based on immediate stimuli and established neural structures seems plausible; only the input which is most pressing and co-

[2] Here i enumerates the component nodes of the current layer, and j enumerates the inputs to the layer.

[3] Not to be confused with the input features in machine learning, even though there can be an overlap

[4] *Spatial frequency* is defined as the periodic distribution of light and dark in an image; it is high around edges and elsewhere the gradient of the pixel value is large [American Psychological Association , 2015]

herent with previously important input is processed.

Another way of looking at this prioritising is as a model of *bottom-up attention*. Bottom-up refers to a process starting at the bottom of the cognitive processing hierarchy and working its way up. The bottom is often associated with input reception, and the top with higher-level cognitive functions. This has been used especially to model visual attention [Itti and Koch, 2001].

However, a problem then arises: How can attention deviate from permanently focusing on the information of the winner-pathway. Another neurobiological phenomenon observed in humans to counteract the winners taking over is *inhibition of return (IOR)*. This refers a discrimination of the locations previously activated, making these neurons less likely to activate again, and allowing the focus of attention to alternate [Itti and Koch, 2001].

2.3.3 From Neurobiology to Machine Learning

When implementing machine learning models mimicking these neurobiological phenomena, one can assume some of their functionalities will follow. For one, implementing winner-takes-all behaviour in an artificial neural network will lead to neural pathways. With such artificial neural pathways one can expect decomposition of input which are encoded in various paths. Creating the pathways can be considered creating several ANNs where each is specialised on a certain feature. Each pathway will have the same input as the entire network but only a subset of the nodes and weights.

The creation of pathways might lead to higher performance as it aims to enhance feature selectivity. As existing research on many of the presented neurobiological phenomena are executed on visual perception, e.g. [Itti and Koch, 2001], it is reasonable to assume the winner-takes-all implementation will have the largest impact on data similar to external visual stimuli. A way to check whether a path is specialised on a feature is to send similar datapoints through a trained network and examine whether a pathway arises. However, it should be mentioned that the features might not be what a human expect; a model might pick up on completely different features to a human.

It is reasonable to assume that inhibition of return observed in the neurobiological systems can be expressed as a form of regularisation when implemented in the ANNs.

2.4 Local Winner-Takes-All (LWTA)

The idea behind LWTA algorithms is to implement *local learning* in the neural network, meaning that the entire network does not learn how to transform a given input, but the learning is rather outsourced to local parts of the network. In LWTA algorithms, the nodes in each layer are grouped into a number of groups G, and Winner-Takes-All is applied on the activation of each of these groups, only passing on the maximal output in the group. This creates various neural pathways which learn during training, and are selected at inference time. The hypothesis is that different pathways get specialised in recognising different features of the training data and that a pathway will be activated given a datapoint with similar features as it is trained on, making the network more flexible and better at separating input [Wang and Jaja, 2013].

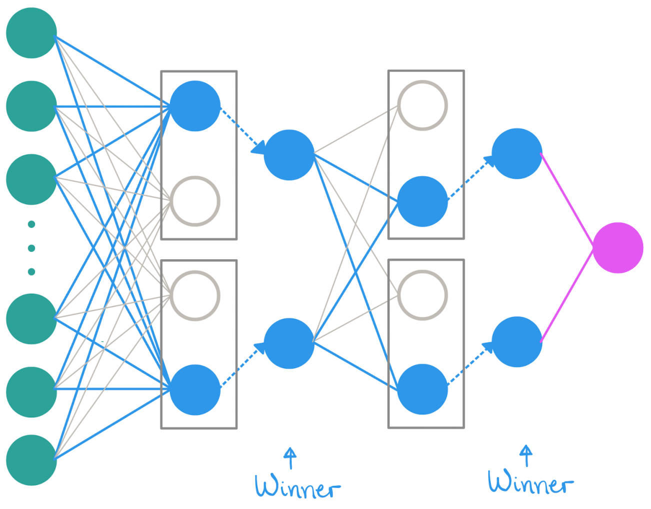


Figure 2: Illustration of a neural network with maxout.

Two different LWTA algorithms will be employed in this report: maxout, where pathways are only formed posteriori; and channel-out, where they are also formed anteriori.

2.4.1 Maxout and Channel-Out

The *maxout* activation passes the input from the previous layer through the weight kernel and adds the bias as normal, but then only passes on the activation from the most active node in each group of the layer. Each group then has a common set of output weights connected to the next layer in the network.See Fig. 2 for illustration of a neural network with maxout activation. During backpropagation in training, only the incoming weights connected to the most activated nodes will be trained. This can be seen from the gradient of the cost function in the direction of one of the weights of an inactive node: the learning is proportional to $\frac{\partial C}{\partial w_{ij}^{\ell}} \propto \frac{\partial C}{\partial a_i^{\ell}} = 0$, because any infinitesimal change in the activation would still result in no activation if a_i was not already the maximum of the group. Given a datapoint, the neural pathways created through this algorithm consist of weights and nodes (see Fig. 2 for an example of pathways). The relevant nodes of a pathway are chosen based on which activation is largest within a group, and the input weights corresponding to those nodes are consequently chosen [Wang and Jaja, 2013].

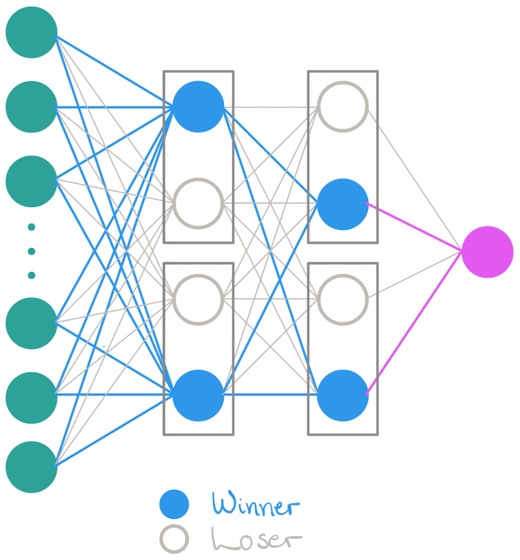


Figure 3: Illustration of a neural network with channel-out.

Channel-out makes a subtle, but meaningful, change to the maxout algorithm. Instead of every group having a common set of output weights, every node has its own set, making for a larger total number of parameters. This results in not only local learning of the weights during backpropagation, but also leads to the activation winner dictating the set of weights that will be used in the next layer at inference time (outgoing weights). Effectively, a channel-out layer works the same way as an ordinary dense layer with a linear activation function, but sets all the activations of the nodes that do not win their group to zero before passing them on to the next layer. Same as with maxout, only the weights of the active nodes are trained, but this also applies to the next layer; it not only affects the incoming weights, but also the outgoing weights. So the local learning goes both forwards and backwards from a channel-out layer, in contrast to just backwards in the case of maxout [Wang and Jaja, 2013].

A way to look at the function of the LWTA groups is to study the effective activation output of the group. Simplifying by looking at a group of nodes whose input is only one scalar quantity x. The response of the nodes in the groups are linear functions $z_i = w_i x + b_i$ of the input, and the activation output is $\max(z_i)$. Fig. 4 illustrates how the maximum of a group of such functions can mimic and parameterise different convex function. In fact, it is shown that a group of an arbitrary amount of nodes can approximate any convex function [Goodfellow et al., 2013].

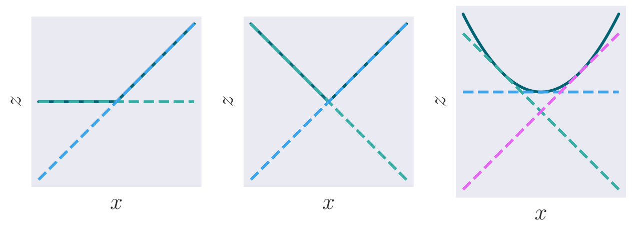


Figure 4: An example of how different convex activations functions ReLU, $|x|$ and x^2 can be approximated or exactly reproduced with the max-function on the responses of a group of units. The responses are here depicted with dashed lines.

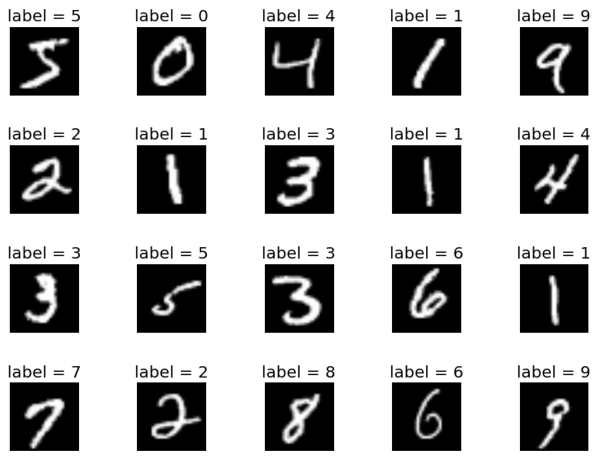


Figure 5: 20 example images from the MNIST dataset with their appropriate labels.

3 Method

3.1 Datasets

First, it will be useful to introduce the datasets we will use for training and testing our networks in this project.

3.1.1 MNIST

A simpler test for our neural networks is the MNIST benchmark dataset. It consists of 60 000 training images and 10 000 testing images. Each image consists 28 × 28 pixels depicting handwritten number between 0 and 9. As such, the images are to be classified to one of the 10 categories of digits. Some examples of the images in the dataset are depicted in Fig. 5.

3.1.2 CIFAR-10

Initially, we applied our the LWTA NNs on the CIFAR-10 benchmark dataset for image recognition. This dataset consists of 60 000 images with 32 × 32 coloured pixels with RGB values, amounting to 3072 features per image. The images are divided into 10 categories, depicting one of either aeroplanes, birds, cars, cats, deer, dogs, frogs, horses, ships or trucks (see Fig. 6).

Figure 6: Examples of the images in the CIFAR10 dataset.

3.1.3 Premier League 2019 Season

Analyst groups working in the larger football leagues have access to technology that gives them more complex data than the main match events such as number of goals scored or corners given. For instance, a group on football analysts will most likely measure a team's or player's performance through metrics like expected goals (xG) or the number of passes allowed per defensive action (PPDA). The former is a very much used measure in football, but the algorithm behind it is not universal. In any case, an xG model estimates the probability of scoring a goal given some action (e.g. shot from a good position) and should sum up to a number close to the actual goals scored during a season, ideally also a match [Sumpter, 2016, Ch. 14]. In the English Premier League (EPL), various sites provide live updates of the xG's for two opposing teams during their clash, however, the exact measurement will be dependent on the xG model the site uses.

A vital part of football analytics is not only to evaluate a team's performance, but predict future match outcomes based on said statistics [Herbinet, 2018]. The basic idea of the following analysis was to use match events and underlying statistics from previous games and seasons to predict if a match will end in a home or away win or in a draw.

The EPL is the primary division for men in the English football league system. 20 teams participate each season, three of which newly promoted from the English Championship. For the biggest football league in the world, the bookmakers tend to keep the most interesting data to themselves, as do the clubs' data science teams [Sumpter, 2016, Ch. 15]. However, being the most-watched sports league in the world, the public datasets from EPL are reliable and usually uncorrupted, and one can retrieve quite detailed match data from decades ago.

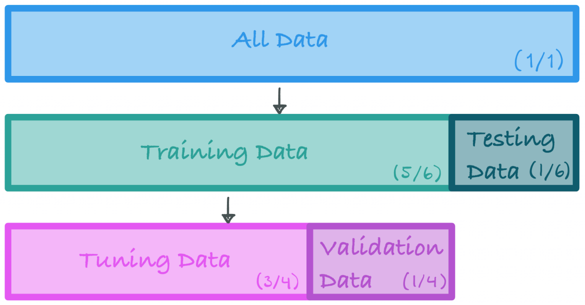


Figure 7: Illustration of data splitting.

The challenges in collecting data and the time scope of this project compelled us to keep the purpose simple and ambitions somewhat low. In particular, we aim to foresee the outcome of a league match in EPL season 19/20 with the self-composed dataset described in App. A. After preprocessing data from various sources, our dataset contains information about the ten league matches in each game week following the first one, but from the perspective of each team, giving $10 \times 37 \times 2 = 740$ data points. We used the games from the first $5/6$ part of the season, not counting the 10 opening matches, for training, corresponding to 308 games. The remaining 62 final games (last $1/6$ part of the season) was reserved for testing. All in all, we had 616 training data points and 124 testing data points.

The choice to look back only one league game for one team was due to a great ease in the data acquisition process. Trying to predict the full time result is a much more comprehensive task than to classify the outcome for a team as win (W), draw (D) or loss (L).

Another important simplification we made concerns the promoted teams. As these teams did not participate in EPL by the beginning of 2018, we (quite naively) assumed that their previous season team profile corresponds to that of the average lower quartile of that season's table.

3.1.4 Data Preparation

To train and evaluate our models we implemented the relevant data and as in our previous projects we split it in training data ($5/6$ of the data) and testing data ($1/6$ of the data) [Aasen et al., 2022a,b]. When tuning the architecture, we further split the training data into a tuning set and a validation set used for monitoring overfitting, implementing early stopping with $p = 5$. We used a split of $3/4$. See Fig. 7 for illustration of the data splitting.

All data was preprocessed by scaling the training dataset features with the Standard Scaling algorithm, such that the features all have mean zero and with standard deviation one. This is explained in detail in Aasen et al. [2022a].

3.1.5 PCA

Before being subject to the principal component method, the Premier League data was z-score normalised, centring the mean around zero, with unit variance. This ensured that no single feature with a relatively large variance would dominate any of the principal components. The principal components found created a new basis for the data, and had similar shape as the original features (they were linear combinations of them). Extracting the first few of these yielded the most important components. For all principal components we found the explained variance, which is the "amount" of variance accounted for by the principal component in question. In order to make an attempt at reducing the dimension of the problem, we found the number of principal components necessary in order to explain 99 % of the variance of the original features. The Premier League dataset was the only dataset on which we performed PCA.

3.2 Neural Network Training

3.2.1 Initialisation of the Neural Network Weights

It is important to take care when initialising the weights and biases of a neural network to ensure fast convergence during training. We initialised the weights according to the He algorithm [He et al., 2015]. In our case it meant initialising the weights of node i in layer ℓ by the distribution

$$w_{ij}^{\ell} \overset{d}{\sim} \mathcal{N}\left(0, \sqrt{2/\hat{n}}\right), \tag{2}$$

where $\hat{n}$ is the number of inputs to the layer. The biases were all initialised to zero.

3.2.2 Optimisation Algorithm

When optimising the cost function of our problems we built on what we found in [Aasen et al., 2022b], using stochastic gradient descent with a batch size of 32. We used the Adam algorithm with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a learning rate $\eta = 0.001$. These hyperparameters were not tuned in any of our searches. The number of training epochs was decided with early stopping termination criterion. This means monitoring the cost of the network on a validation dataset after each epoch of training, and stopping the training if there is no improvement after a number of epochs p, called the *patience*.

All the datasets used pose classification problems with 3 or 10 categories. The cost function we used for optimisation was the standard categorical cross-entropy. Predicting probabilities $\boldsymbol{p}_c$ for the outcome being in category c for the datapoints $\boldsymbol{y}_c$, with n datapoints, the categorical cross-entropy is given by

$$C(\boldsymbol{p}) = \sum_{i=1}^{n} \sum_{c \in \text{categories}} y_{i,c} \log\left(p_{i,c}\right). \tag{3}$$

3.2.3 Regularisation

The first method we applied to combat overfitting was to add a L2 penalisation term to the kernel weights of the layers, as was done in Aasen et al. [2022b]. This adds a term to the cost function penalising large weights, and is parametrised with a hyperparameter λ, whose size is proportional to the penalisation.

A problem that can occur in LWTA layers is when the weight of a certain input to one of the nodes of a group becomes much larger than those of the other nodes in the group, resulting in the output of the group being dominated by a single node and a single input. This can result in overly local training, perhaps even of a single node and a single weight. A common technique to combat this is to add a dropout layer before and/or after the LWTA layer. The dropout layer randomly sets elements of its input vector to zero, according to a dropout rate, before passing it on to the next layer. This means that the nodes in a group cannot solely lean on one specific node or input, and forces pathways in the network to not learn specific trends 'too much'. As such, it prevents overfitting, and can be seen as a regularisation method. The dropout layers are only active during training, and do not drop any activations at inference time. The effect of dropout on a LWTA network is illustrated in Fig. 8.

When adding dropout to our networks, we did so with between every layer with the same dropout rate. This includes dropout on the input layer.

3.3 Visualising Pathways

A good place to start with LWTA networks is to take a look at what actually happens during inference. We fitted a channel-out network on the full training MNIST dataset with three layers, 8 nodes in each and 4, 2 and 4 groups respectively in each network. This was done with a dropout rate of 0.125, ridge penalisation added with $\lambda = 0.01$, and early stopping with $p = 5$. We then plotted the pathways that were activated when inputting 200 test images depicting zeros, ones, fours and fives.

3.4 Architecture Tuning

To see what kind of network architectures were interesting and useful when applying to the datasets at hand, we tuned the architecture using the *hyperband* search algorithm [Li et al., 2018].

We searched with a variable number of hidden layers between 2–5, a number of nodes in the layers between 8–64 in powers of 2, and a number of groups in the layers

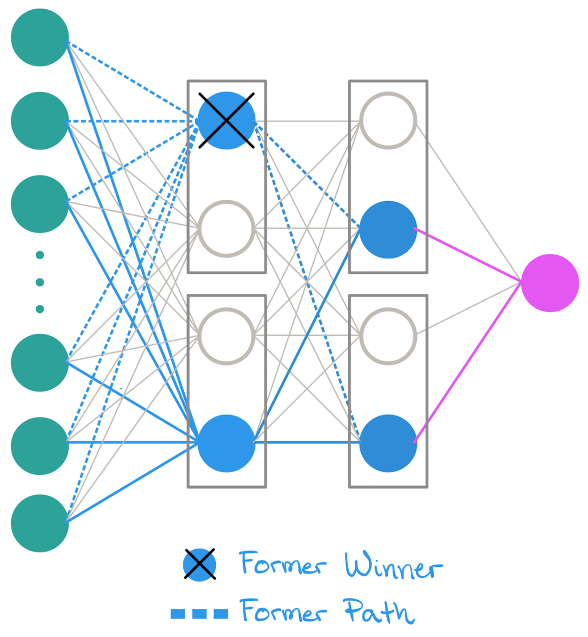


Figure 8: An illustration channel-out neural network with a dropout added to the first channel-out layer. Only the lower group in the first hidden layer is trained.

between 4–32, again in powers of 2. When hyperparameters were chosen such that the number of groups was equal to or greater than the number of units in a layer, we set the layer to be a dense layer with the specified number of units, but added a ReLU activation to the output.

3.4.1 Hyperband Search Algorithm

The hyperband search algorithm serves as a way to tune the hyperparameters of a model in a way such that more computing resources are dedicated to more promising areas of the hyperparameter space. It is based on the *successive halving* algorithm, in which a finite budget B (i.e. number of training epochs) is divided evenly between n randomly sampled points from the hyperparameter space. These hyperparameters are then used to build n models that are trained according to $r = B/n$ allocated resources on a tuning dataset, and their loss evaluated on a validation dataset, passing on the k top performing models.[5] These k models are then trained further, with a new $n = k$ and consequently more resources r. This halving is then repeated until there is one model remaining. As hyperparameter points are discarded (n decreases), the amount of allocated resources r increases; meaning the more promising hyperparameter points are allotted exponentially more resources.

This introduces a trade-off in the division of the resources B; sampling a larger number of hyperparameter points n leaves less resources r for exploring each point.

[5]Usually the top half of the is passed on, as the algorithm name would indicate.

The hyperband algorithm aims remedy this by doing a grid search of different values of n and smaller resource allotments r. For each value of n and r, the successive halving is performed, and the best preforming model is found. In the end, models that perform better deeper into the training will be found, but many hyperparameter points will have been explored.

Hyperband takes two hyper(hyper)parameters, R indicating the maximum amount of epochs of training for any single hyperparameter point, and a factor η which is the reciprocal of the fraction of points that are passed on during each halving.

3.4.2 CIFAR-10

On the CIFAR-10 dataset, we tuned using 1 hyperband iteration with a maximum number of epochs $R = 30$ and $\eta = 3$. We used early stopping with $p = 5$ to terminate the training. After finding best network architectures, we fitted them on the full training dataset, evaluating their accuracy on the test dataset. We used early stopping with $p = 10$ during this training.

When adding dropout we used a dropout rate of 0.1, and with Ridge penalisation we used $\lambda = 10^{-4}$.

3.4.3 Premier League Dataset

For comparison, we made an algorithm that guesses the outcome of a match given the ground (home/away) with the knowledge that overall, almost half EPL games result in home win, about a third in away win and the rest in draw.[6] In addition, we attempted to simulate our very own "Paul the Octopus [Parkinson, 2022] by telling the algorithm to assume that the three outcomes are equally frequent. We were interested in the accuracy scores from these guesses. Finally, we calculated the accuracy score for the rather boring case of only betting home wins, necessarily corresponding to the portion of EPL 19/20 matches in the set ending this way.

With the smaller Premier League dataset, we could afford 3 full iterations of the hyperband search with a maximum number of epochs $R = 150$ and again $\eta = 3$. Again, we used early stopping during the tuning, with $p = 15$. After finding the top-performing architectures, we trained on the full training set with early stopping with $p = 30$.

When adding dropout we used a dropout rate of 0.25, and with Ridge penalisation we used $\lambda = 10^{-4}$.

4 Results

To have a compact manner of referring to a specific implemented network structure, we introduce a new notation. A network with a specific architecture will hereby

[6]According to [SoccerSTATS.com], in EPL 18/19, 48% ended in home wins, 34% in away wins and 19% in draws.

be referred to as $\mathbf{N}^{nodes\ by\ layer}_{groups\ by\ layer}$, where the superscript indicates the number of nodes in each hidden layer, chronologically, and the subscript indicates number of groups in each these layers. Whenever the number of groups equals the number of nodes, and we get an ordinary dense layer, we add a ReLU activation on the node output, and indicate it with an asterisk $*$. As an example, if a network has three layers: 32 nodes and two nodes per group; 64 nodes and four nodes per group; and 16 nodes and one node per group with ReLU, it will be denoted as $\mathbf{N}^{32,64,16}_{16,16,*}$.

4.1 Visualising Sparse Pathways

After training the channel-out network $\mathbf{N}^{8,8,8}_{4,2,4}$ on the MNIST dataset, we got a final test accuracy of 84.09%. The activation pathways are visualised in Fig. 9.

4.2 Tuning LWTA Architecture

Tuning the architecture of our LWTA networks, we found the maxout network architecture $\mathbf{N}^{64,64}_{32,16}$ with dropout to do the best on the CIFAR-10 dataset, achieving a test accuracy of 51.28%. Likewise, the best performing channel-out network architecture was $\mathbf{N}^{64,16,64}_{32,*,32}$ with L2 penalisation added, achieving a test accuracy of 50.69%, slightly underperforming maxout. The ordinary ReLU dense network achieved an accuracy of 49.19% on the test dataset with two layers of 32 nodes. The results are tabulated in Tab. 1. The history of the test accuracy of the LWTA algorithms during training is found in Fig. 10.

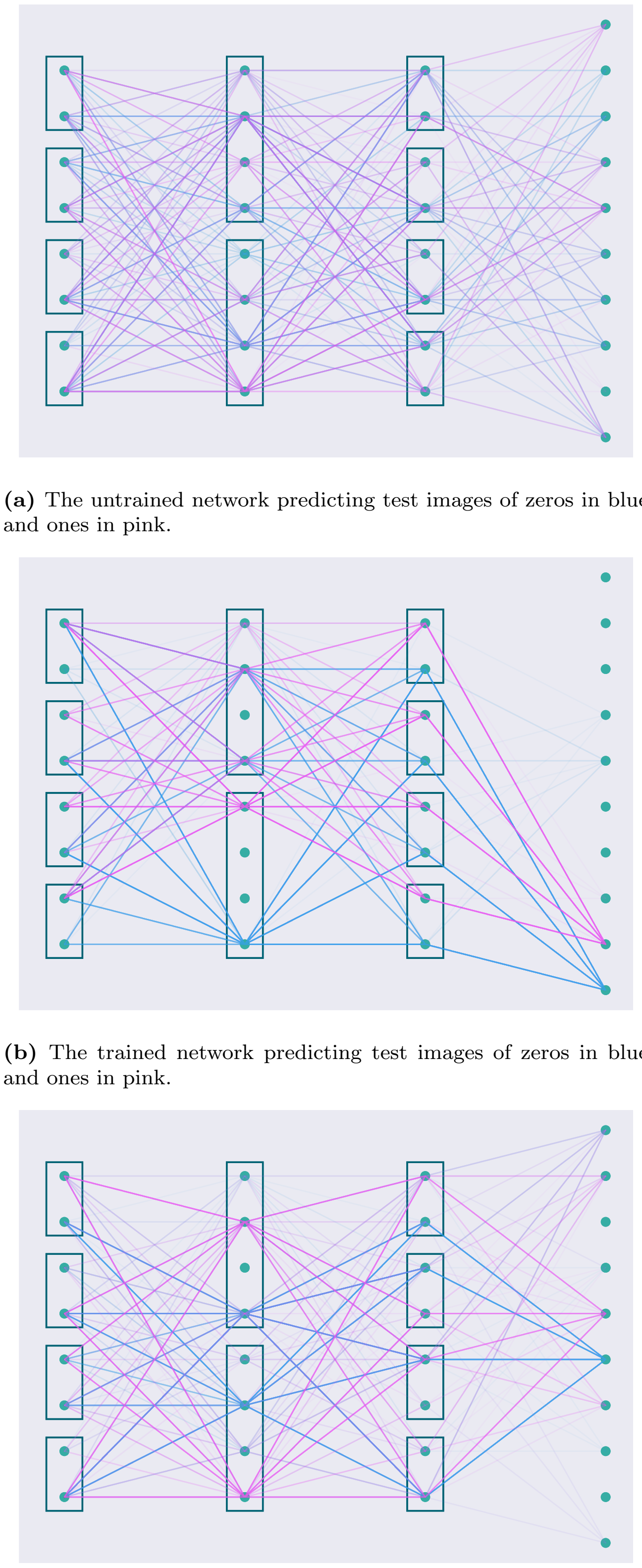
4.2.1 Tuning on the Premier League Dataset

On the Premier League data we found the maxout network $\mathbf{N}^{16,64}_{8,32}$ with dropout and L2 penalisation to score best, with a test accuracy of 58.06%. The best channel-out network was $\mathbf{N}^{8,64,16,64}_{*,32,*,32}$ with no additional regularisation, scoring an accuracy of 54.84% on the test data. An ordinary dense ReLU network $\mathbf{N}^{16,16}_{*,*}$ outdid both LWTA algorithms however, with a test accuracy of 58.87%. The results are tabulated in Tab. 2. The training history of the LWTA algorithms is found in Fig. 11.

In comparison, the random guesses we made all resulted in accuracy scores below 50%. The wild guesser (the octopus) performed worst with an accuracy score of 35.48 %, whereas the craven guesser (only home wins) got an accuracy score of 48.39% on the test set. The ‘learned’ random guesser (50/20/30 for home win/draw/loss) was correct in 39.52% of the cases.

4.3 PCA on Premier League data

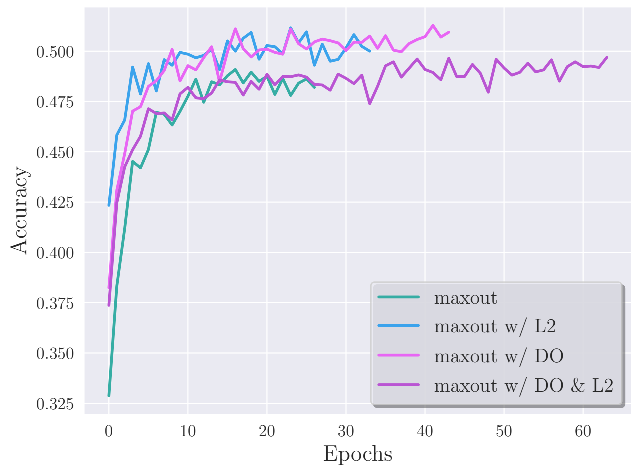
By performing PCA on the Premier League data we found its principal components. The explained variance as a function of principal components is plotted in Fig. 12 as both a cumulative sum and histogram bars. As seen


(a) The untrained network predicting test images of zeros in blue and ones in pink.

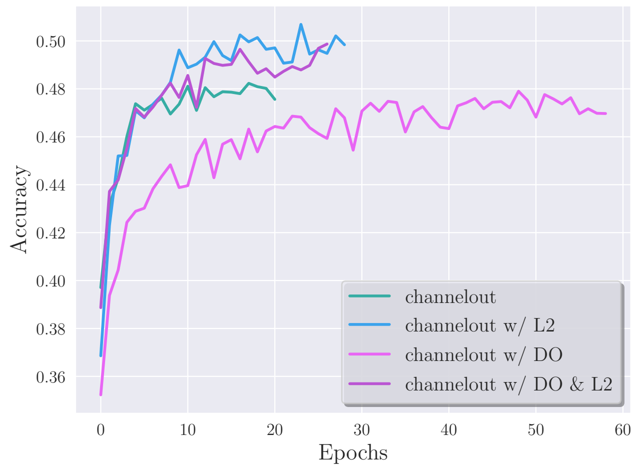
(b) The trained network predicting test images of zeros in blue and ones in pink.

(c) The trained network predicting test images of fours in blue and fives in pink.

Figure 9: Illustrations of the active pathways as 100 input images from the MNIST test dataset belonging to two different classes are passed through a channel-out network. The network was trained on the full training dataset for 26 epochs before stopping early with $p = 5$.



(a) The test accuracy history of the best performing maxout networks.



(b) The test accuracy history of the best performing channel-out networks.

Figure 10: The best performing LWTA models after tuning on the CIFAR-10 dataset were trained on the full training dataset and their test accuracy history is plotted here. For the network architecture of the various models see Tab. 1.

	Regularisation	**Architecture**	**Accuracy**
Maxout	None	$\mathbf{N}^{64,32,64}_{32,16,8}$	0.4909
	Ridge	$\mathbf{N}^{64,32}_{32,*}$	0.5116
	Dropout	$\mathbf{N}^{64,64}_{32,16}$	0.5128
	Both	$\mathbf{N}^{32,64}_{16,16}$	0.4969
Channel-Out	None	$\mathbf{N}^{64,16,32}_{16,8,*}$	0.4823
	Ridge	$\mathbf{N}^{64,16,64}_{32,*,32}$	0.5069
	Dropout	$\mathbf{N}^{32,16,64}_{16,*,32}$	0.4790
	Both	$\mathbf{N}^{32,64}_{16,32}$	0.4987
Dense	None	$\mathbf{N}^{32,32}_{*,*}$	0.4843
	Ridge	$\mathbf{N}^{32,32}_{*,*}$	0.4919

Table 1: Table of the best performing network architectures on the CIFAR-10 dataset. We used maxout, channel-out or dense ReLU layers and various combinations of dropout and ridge penalisation added.

	Regularisation	**Architecture**	**Accuracy**
MO	None	$\mathbf{N}^{8,16}_{*,8}$	0.5403
	Both	$\mathbf{N}^{16,64}_{8,32}$	0.5806
CO	None	$\mathbf{N}^{8,64,16,64}_{*,32,*,32}$	0.5484
	Both	$\mathbf{N}^{8,64}_{*,32}$	0.4839
Dense	None	$\mathbf{N}^{16,16}_{*,*}$	0.5161
	Ridge	$\mathbf{N}^{16,16}_{*,*}$	0.5887

Table 2: Table of the best performing network architectures on the EPL dataset, with maxout, channel-out or dense ReLU layers and various combinations of dropout and ridge penalisation added.

from the plot, 42 principal components explains 99 % of the variance of the original features. Since this would reduce the dimensionality by approximately one half only, and we considered relatively few features we decided not to use the principal components for further analysis.

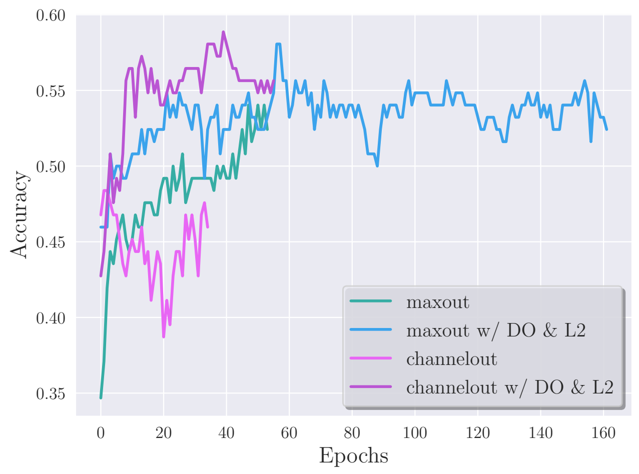


Figure 11: The history accuracy on the test portion of the EPL dataset of the four best LWTA architectures, with and without regularisation methods added.

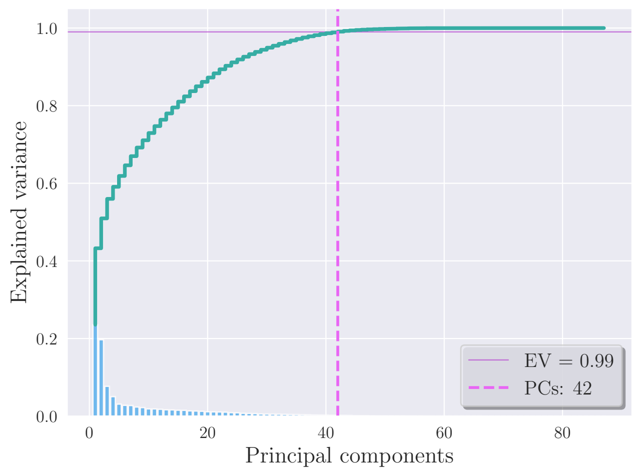


Figure 12: Explained variance as function of the number of principal components, shown both as a cumulative step function and histogram bars showing the step increase for each component.

5 Discussion

5.1 Interpretation of LWTA Networks

From Fig. 9b and Fig. 9c it is clear that separate pathways are formed that are specialised on input that are to be categorised differently. The untrained network Fig. 9a displays no such patterns. In Fig. 9b we can see that the lowest node in the middle layer strongly correlates with results that are classified as 0, with clear pathways being formed with the same nodes in the groups of the layers before and after. This coincides with our expectations based on the neurobiological pathways and feature selectivity. A further analysis of different inputs could help give insight into what these pathways have learned, be it roundness of the digits, amount of dark vs. light, digit placement in the picture, or perhaps some completely different and unintuitive feature. This provides a way to start extracting information about what the network is actually learning, making the model more explainable. It might however learn trends that are not obvious for a human to discern.

5.2 Comparison to State-of-the-Art LWTA Performance

The results we got on the CIFAR-10 dataset, with the best test accuracy of 51.28%, was not particularly impressive. It is however known to be a difficult dataset to score well on, with the state-of-the-art convolutional maxout network achieving a test accuracy 88.32% [Goodfellow et al., 2013], which was the best result of any NN at the time. This was done with a significantly larger network, with convolutional layers, and with more thorough image pre-processing. Channel-out networks have not achieved the same level of accuracy, with a top score of 86.80% test accuracy [Wang and Jaja, 2013]. This was also our conclusion, with the channel-out network slightly underperforming maxout. However, both channel-out and maxout networks bettered the ordinary ReLU network result by 1.50 pp and 2.09 pp respectively.

5.3 Overfitting

Throughout this project, we implemented many methods to reduce the chance of overfitting the NNs to the training data. Primary of these have been early stopping, which uses a validation dataset which the network does not train on to monitor whether the cost is improving during the training. As long as the validation cost is improving, we can assume that the network has not yet overfitted the training data, but when the training cost keeps improving and the validation cost goes up, it is a clear sign that the model is starting to overfit.

L2 penalisation on the kernel weights also helps by penalising situations where the networks learn to delicately balance large weights against each other. This can create results that are stable on the training dataset, but when exposed to inputs not yet seen, the large weights that were balanced on the training inputs can create large fluctuations in the outputs resulting in instability.

As expected based on the neurobiological phenomenon inhibition of return, dropout works a third regularisation method that ensures that the network does not rely overly much on the activation of certain nodes or on certain input features. In LWTA networks where information is encoded in pathways, it is especially regularising, as it prevents pathways form becoming overly specific to certain trends. If a certain pathway learns how to handle a specific trend at the cost of other pathways learning it, if it is dropped during training, the network will not do well on that trend, and the other pathways are forced to learn it too.

Finally, LWTA methods can prevent overfitting themselves, as the information is learned locally on subnetworks that are not as complex or capable of learning much information. This means that the subnetworks chosen at inference time are not as prone to being overfit.

5.3.1 Hyperparameter Tuning

An obvious flaw in the tuning of our networks is the lack of tuning of the optimisation algorithm, like the learning rate of the Adam optimiser, or any tuning of dropout and L2 penalisation. We used the common default parameters for the Adam algorithm, and chose dropout rate and L2 penalisation more or less by random with some trial and error. To get better results, these hyperparameters could definitely have been included in the tuning, but this was outside the scope of this project. The main motivation was to see how our neural networks performed with and without regularisation methods added. We did find regularisation to improve results on the CIFAR-10 dataset, but there is no clear difference in what kind of network architectures were preferred with or without regularisation. This was somewhat of a surprise, as we did believe regularisation methods would make larger, deeper networks more viable. It did however provide some stability, and allowed for longer training periods, as can be seen from Fig. 10 and Fig. 11.

The lack of clear results is probably due to the fact that even after tuning, only a rather small part of the parameter space was ever explored. Doing one hyperband iteration with $R = 30$, $\eta = 3$, as we did with the CIFAR-10 dataset, means only 49 hyperparameter points were explored, whereas there are roughly 1 million available architecture configurations. Without the use of model averaging or any resampling techniques, we can also not be sure that the tuning did not find network architectures that were particularly 'lucky' with the initial parameters and thus found a good optimum within the tuning time. On the EPL dataset we could afford a bit more thorough exploration of the architecture hyperparameter space, with three iterations $R = 150$, $\eta = 3$ we explored a total of 429 hyperparameter points; still not great, but better.

5.4 Generation of Dataset

The MNIST and CIFAR-10 datasets was used "out of the box" and are thus not relevant for this analysis. On the other hand, the EPL dataset was generated for this purpose, with individual parts of the dataset obtained from a variety of sources.

When concatenating different parts into one big dataset like this, some features are likely to correlate to a large extend. This is not unusual for large dataset, but it is fair to assume that it has affected both the analysis of the data and the PCA. There is also a question as to how the data was sampled and processed by the different sources. Ultimately this lead to a possible incoherence in the dataset which is hard to measure. On the other hand, the features all originated from the same games so, assuming valid sources, they should more or less tell the same story, yielding a reliable dataset.

5.4.1 PCA

The decision to not use PCA for our further analysis of the Premier League data was first and foremost made with respect to Fig. 12 where we would need 42 principal components in order to explain 99 % of the variance. Considering the relatively few features in the original dataset, 87, we could as well perform the analysis directly on the original features. Perhaps if we put the threshold lower than 99 %, say 80 %, the principal components would greatly reduce,[7] but at the caveat of reduced explained variance. It is fair to assume that PCA would be of great benefit for a larger number of features, as PCA is normally a popular and effective approach when dealing with overfitted models.

One drawback with regard to the dataset could also be the fact that we composed it ourselves, from a variety of sources, hence yielding some highly correlated features.[8] These should probably have been analysed further with a correlation matrix, but this work was not prioritised here, as we quickly discarded the analysis of principal components. Performing correlation analysis would be of greater importance if the dataset contained more features and/or if the nature of the dataset was completely unknown to us in advance.

For instance, performing PCA on the MNIST dataset of $28 \times 28 = 784$ features would arguably be a good idea. Likewise for the CIFAR-10 dataset of $32 \times 32 = 1024$ features. However, this was not done due to the time limit of this project. The analysis of the models trained

[7]By-eye inspection of the figure yield about 15 principal components for 80 % explained variance.

[8]This is because different sources may calculate similar features in different ways. Hence, when they are all included some are bound to correlate

using these datasets could be performed without PCA while not loosing too much computational efficiency, but a PCA analysis would have been informative in its own rights.

6 Concluding remarks

Looking at how information is handled in biological processes provides an interesting angle in guiding machine learning algorithms. The concept of winner-takes-all behaviour and local learning does have an obvious parallel that can be exploited in neural networks. We found effective implementations of LWTA behaviour with two types of networks; maxout and channel-out. These networks encode information in local pathways, creating subnetworks that are chosen at inference time. We found this to help with the explainability of notoriously opaque neural networks.

Applying the network on the benchmark image recognition dataset CIFAR-10, we found a test accuracy of 51.28% with maxout layers with dropout, and 50.69% with channel-out layers with ridge penalisation. This was an improvement on an ordinary Dense neural network with ReLU activation, achieving 49.19% test accuracy. Applying to our Premier League dataset from the 2019/2020 season, we found a test accuracy in predicting match outcomes of 58.06% with maxout layers with dropout and ridge penalisation and 54.84% with purely channel-out layers. This did not improve on the results from an ordinary dense ReLU network achieving 58.87% test accuracy.

These results were achieved after tuning the network architecture with the hyperband search algorithm, and we believe it could be improved with tuning of more hyperparameters, and if with more computational resources applied to the tuning search. A natural next step to take would be to include convolutional LWTA layers on the image recognition data, and reduce the features using max pooling layers. On the EPL data, it would be natural to expand the dataset to include more statistics from more than one prior match, and use recurrent neural networks to allow them to get a better sense of the form trends in the team performances.

From our results, it is not clear whether LWTA networks are useful in and of themselves, other than providing more transparency in their inner workings. A natural application could perhaps be to model neurobiological systems to study how these systems encode and infer information. It would also be interesting to look at more complete neurological network models, like those of spiking neural networks or recurrent neural networks that implement a time component to the network [Chen, 2017].

The principal component analysis did not prove fruitful enough to be included in the final analysis of the EPL data, due to the relatively few original features, and limited time. With more time however, this would have been improved and implemented, both on the EPL dataset, and also on the MNIST and CIFAR-10 datasets. Thus, future work might include a more thorough PCA, comparing the accuracy of trained models using different numbers of principal components. An interesting analysis could also be to investigate the principal components' dependence on the original features.

A Self-Composed Premier League Dataset

In Fig. 13 we present the idea behind our homemade dataset. There are in total 87 features that we use in our analysis, in addition to the actual outcome for each match.

For a clash between a team and its opponent, we have some general facts such as what weekday it happens on and whether the team is playing on familiar ground (home) or not (away).

Both the team and the team it is facing, each have their own *team profile* (lower left panel of Fig. 13): information about how much the club spends on player wages[9] from both the season we are looking at and the previous one, and a set of underlying statistics from the season before. In addition to table position, the underlying statistics include metrics like the total xG or none-penalty xG (npxG), PPDA in the opposition half and opponent PPDA (OPPDA) in the opposition half. A number of the features describe the deviation from an estimator and the actual value, such as the difference between expected points (xpts) and actual points (pts) gained.

For the team we have included its form for each match, only looking back one match (again, in order to not use too much time on the data acquisition). This is why we do not include the first game week. The form (right panel of Fig. 13) includes basic match data and events, as well as a set of underlying statistics somewhat larger than that of the team profile.

B Bias-variance Trade off

We will in the following take a little detour from LWTA networks to investigate an important dissection of model error. If you are reading this in one sitting, this is the time to grab yourself a new cup of coffee. The total error of a statistical model can be decomposed into three main contributions; the *bias*, *variance* and *irreducible* error.

Every learning algorithm makes some assumptions about the relationship between input features and target outputs. Using a model with high bias indicates that there are hard restrictions on the predictions and might result in missing key relations between features and targets, frequently called underfitting. Relaxing model restrictions can yield a lower bias, but at the cost of potentially increasing the variance, often called overfitting. The variance describes how sensitive a model is to fluctuations in training data. High variance might result from a model taking training data noise into consideration when fitting its parameters. Giving the model more constraints can help to decrease the variance, at the cost of introducing more bias. This is the basis for the *bias-variance tradeoff*, where we aim to find the set of parameters giving both a small bias and a small variance. The irreducible error comes from deviation between the assumed underlying function and the targets in the dataset (noise). As the name suggests, this contribution will always be there (assuming the dataset targets are noisy) irrespective of an optimal model.

[9] Potential to add more attributes here, such as average age and match attendance, or scope of fanbase. Extracting such data turned out to be circumstantial, which is why we have not gone further here.

C Setup and Resampling

We will in the following only consider regression problems [10]. In [Aasen et al., 2022a] the bias-variance decomposition for the MSE was derived, giving:

$$\mathrm{MSE}(\hat{f}) = \left(\mathrm{Bias}(\hat{f})\right)^2 + \mathrm{Var}(\hat{f}) + \sigma^2 \tag{C.1}$$

With σ^2 as the irreducible error. To estimate model bias and variance, the bootstrap resampling technique will be performed, using a total of 200 bootstrap rounds for each set of model parameters. Taking different sets of models, we will make a comparison and try to find the optimal model based minimising both bias and variance. If not specified otherwise, MSE values (and of course bias and variance) is calculated using the bootstrapped test data.

D Dataset and Preparation

The popular video game series FIFA has new releases yearly, with a large fan base for both recreational and competitive play. Consequently, interest of analyzing ingame data with the goal of getting a small edge is becoming more popular. For instance [Al-Asadi and Tasdemır, 2022] tries to estimate player market value based on player characteristics. Here, we will use a similar dataset [Leone, 2020], with the goal of calculating the *overall score*, a single number between 1 and 99 representing the players skill and worth. There are some discrepancies on how scoring is performed across different games, thus we restrict ourselves to the FIFA21 version. Overall score will be estimated using 38 different minor attributes, such as player dribbling skill and mental composure. All of these are real numbers in the range 1 to 99. In total there are $n = 18944$ players, where we pick a subset of 10000 players, weighted such that we get players from different football leagues and skill levels. These $p = 38$ features are presumably not all FIFA uses to calculate the overall score, indicating that a perfect model can not be found through this approach. This could be viewed as a regression problem with 99 possible classes, however due to the ordinality and large number possible

[10] For classification problems, a similar decomposition can be performed, for instance using the *Zero-One loss*, see [Kohavi and Wolpert, 1997]

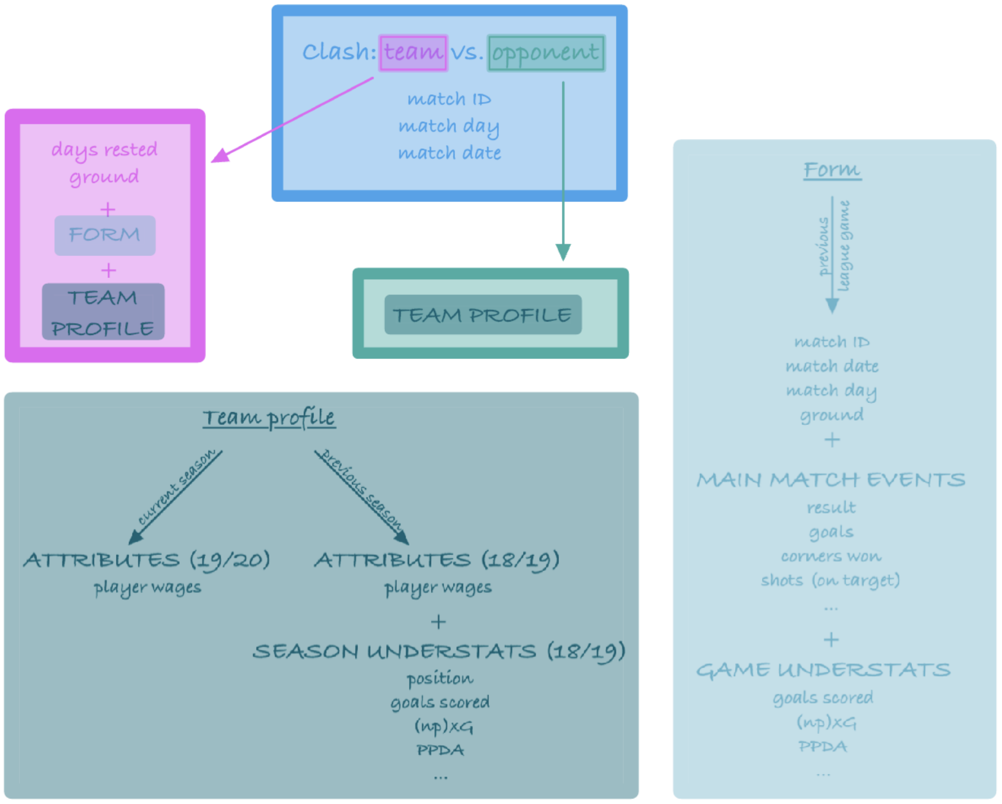


Figure 13: Schematic of how the features describing a match are connected. See text for further details.

scores, approaching it as a regression problem sensible. To incorporate this into our models (where predictions are made in $\mathbb{R}$), rounding to the nearest whole number in the range $[1, 99]$ will be done. The errors (that is MSE, bias and variance) will therefor be presented unscaled.

E Models

We will present the key theory required for three different sets of estimation methods; *Linear Regression*, *Tree based methods* and *Super Vector Machines.*

E.1 Linear Regression

For linear regression models, we will consider ordinary least squares (OLS), Ridge regression and Lasso regression. Linear regression, as the name implies, assumes a linear relationship between the features and target. A specific data point is simply the sum over features, multiplied by a coefficient for each feature fitted by minimising the MSE. Ridge and Lasso adds a constraint to coefficient sizes, using an L_2 and L_1 norm respectively, presumably introducing bias. This will be represented through the hyperparameter α, proportional to the regularisation strength. See [Aasen et al., 2022a] for a more in depth explanation.

In addition, all features are assumed to give a positive contribution to the overall score (setting all to 99 should yield an overall score of 99), the coefficients are constrained to be positive.

E.2 Tree Based Methods

A decision tree uses a flow-chart like structure to generate predictions. Based on a single features value, a *node* performs some test on the feature (i.e. threshold). Using the outcome of the test, the tree follows one of two paths, called *branches*. Each of these branches can then have new nodes, testing based on new features, creating more branches. This continues until no more nodes are reached, where in the regression case, a specific value is predicted. These final outcomes of branches are called *leaves*. The maximum number of nodes following a single branch gives the *depth* of the tree. There is a plethora of ways to construct a regression tree. The most common approach is *recursive binary splitting*, where separation of the targets into regions $R_1, R_2, \ldots R_J$ is performed based on the input features. When a simple feature inequality is used, these regions are high-dimensional boxes. Considering the set of features $\{x_a\}$ $a = 1, \ldots, p$. We propose a pair of half-planes R_1 and R_2.

$$R_1(a, s) = \{X|x_a < s\} \text{ and } R_2(a, s) = \{X|x_a \geq s\}$$

We then aim to find the values of j and s which minimises:

$$\underset{a,s}{\text{argmin}}\left[\sum_{y_i\in R_1}(y_i-\bar{y}_{R_1})^2+\sum_{y_i\in R_2}(y_i-\bar{y}_{R_2})^2\right] \tag{E.1}$$

Where $\bar{y}_{R_j}$ is the calculated mean of the training data within R_j. This is done in a top-down approach, performing the optimasation from Eq. (E.1) down each branch, continuing until a stopping criterion is reached. This gives us our regression tree $\hat{f}(x)$, following a specific branch when evaluated for a data point x_i. There is no single good measure of model complexity for decision trees, but the depth gives an indication of high finely tuned predictions can become.

E.2.1 Bagged Trees

Based on the idea of bootstrap, an ensemble of trees can be constructed. Decision trees are notorious high-variance models which are easily overfitted. By performing bootstrap resampling, a single tree can be fitted for each bag b, with the final prediction being the average prediction of each tree.

$$\hat{F}(x)=\frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x) \tag{E.2}$$

Where $\hat{f}^b$ is a single decision tree, fitted using the data from bag b. B represents the total number of trees, which we will call ensemble size. We will call this method bagged trees (BT).

E.2.2 Random Forest

Random forest (RF) uses the same methodology as bagged trees. Bootstrap is performed in the same manner as Eq. (E.2), but only a random subset of features is considered at each node split. We represent the number of features considered by m, with $m = 1$ reducing to bagged trees. From [Hastie et al., 2001], a ration of $m = \lfloor p/3 \rfloor$ usually performs well on regression problems. This is expedient since trees might be highly correlated, especially if a single feature is very important for reducing the training MSE. If all features are considered for each split, the random forest reduced to the bagged trees.

E.2.3 Boosted trees

The idea behind boosted trees is combining several weak learners which iteratively learn from previous mistakes to enhance performance. In the case of decision trees, weak learners are simple trees only performing slightly better than random guesses.

When employing gradient boost for regression the process starts by creating a single leaf, with a value $f_0(x)$ often set to the average of all output values in the training set. Implementing a differentiable cost function, e.g. the squared error for regression, the cost of the initial prediction and its gradient with respect to the predicted value is calculated. A simple tree is fitted to minimise the cost. The prediction of that tree is then scaled by a coefficient and is added to the initial prediction, $f_0(x)$, creating the next: $f_1(x) = f_0(x) + \beta b(x; \gamma_1)$. Here β is the scaling coefficient and γ_1 the prediction parameters. The process of calculating the gradient of the cost function, fitting a new tree using the gradient, and adding this tree's prediction to the existing one, is repeated, iteratively creating a better model:

$$f_M(x)=f_0(x)+\sum_{m=1}^{M}\beta b(x;\gamma_m) \tag{E.3}$$

β can be dynamic for various m. In that case the gradient of the cost function must also be calculated with respect to β_m [Hastie et al., 2001].

E.3 Support Vector Machines

The Support Vector Machine (SVM) is most commonly used in classification problems, with the idea of constructing some hyperplane separating target classes in feature space. By slightly changing this approach, the method can also be applied to regression problems proposed in [Drucker et al., 1996] aptly named *Support Vector Regression* (SVR).

E.3.1 Regression Approach

The predictions are performed using a simple linear function

$$f(x_i)=\boldsymbol{w}^T x_i + b \text{ with } \boldsymbol{w}\in\mathbb{R}^p, b\in\mathbb{R} \tag{E.4}$$

We want to find the optimal values of $\{\boldsymbol{w}, b\}$ such that the predictions of $f(x_i)$ lies inside some region around the y_i training data. This region is defined by the hyperparameter ϵ (the so-called ϵ intensive tube [11]). This can be formulated as an optimasation of $\boldsymbol{w}$, subject to the ϵ constraint [Smola and Schölkopf, 2004].

$$\underset{\boldsymbol{w}}{\text{argmin}}\ ||\boldsymbol{w}||^2 \quad \text{subject to} \quad |y_i - f(x_i)| \leq \epsilon \tag{E.5}$$

A larger ϵ gives larger errors admitted to the solution. This is a so-called *hard margin* approach, where $f(x_i)$ must make a prediction in $[y_i - \epsilon, y_i + \epsilon]$. Relaxations of this constraint can be done, allowing small deviations around the ϵ tube. This is characterized by the hyperparameter $C \geq 0$, often called a *soft margin*. C is proportional to the margin relaxation (inversely proportional to the regularisation strength), with $C = 0$ reducing to the hard margin approach.

[11]Not to be confused with the famouse ϵ-sausage used for proofs of continuity, introduced by the math professor *Arne Hole*.

E.3.2 Kernels

One of the benefits of SVMs is the magic of the *kernel trick*. Introducing some non-linearity can improve predictions (for instance, see [Aasen et al., 2022a] for an example using linear regression). This is done by applying some transformation ϕ giving new features in another (often higher dimensional) space $\mathcal{F}$, $\phi : \mathbb{R}^p \to \mathcal{F}$. For large data sets, especially if $\dim(\mathcal{F}) \gg p$, the transformation complexity can result in long computation times. Luckily, both our model Eq. E.4 and optimisation Eq. E.5 only requires the computation of the dot product in $\mathcal{F}$. The kernel trick gives us a method to calculate the transformed dot product $\phi(x_i)^T\phi(x_i)$, without explicitly calculating $\phi(x_i)$ [Kroon and Omlin, 2004]. We express this a function $K(x_i, x_j)$, witch is usually a function of some simpler dot product such as $x_i^T x_j$ or $||x_i - x_j||^2$.

$$\phi(x_i)^T\phi(x_j) = K(x_i, x_j)$$

In the following we will, in addition to the linear (not transformed) product $K(x_i, x_j) = x_i^T x_j$, consider the *Gaussian radial basis function* (RBF) kernel

$$\text{RBF:} \qquad K(x_i, x_j) = \exp(-\gamma||x_i - x_j||) \tag{E.6}$$

Where $\gamma > 0$ is a hyperparameter. Here we will only consider $\gamma = 1/(p\sigma_X^2)$, sometimes simply called the Gaussian kernel, where σ_X^2 is the variance in X across all features.

F Quantitative Exploration

F.1 Linear Regression

The bias-variance decomposition of OLS, Ridge and Lasso can be seen in figure Fig. 14. For OLS there is no penalisation to vary, and all bias contributions comes from the assumption of first order polynomial response. Increasing the feature space, be it introducing higher order polynomials or exponential terms, would allow the fit to be more flexible, presumably decreasing bias. When increasing the penalisation parameter α slightly, both Ridge and Lasso stays quite close to the simple OLS case as expected. At around $\alpha = 10^{-2}$ there is a slight decrease in bias and variance for both models, with MSE minima at 8.46, 8.27 using $\alpha = 1.38, 0.12$ for Ridge and Lasso respectively. In these regions, the variance decreases as expect since the solution space for the optimal parameters shrink. There is also a dip in bias, which might be due to finding a specific set of parameters that works better for our subset of features. We do not use all of FIFAs features, as well as probably including some unimportant ones. There is probably some exclusion and inclusion bias where a penalised model better handles these by shrinking unimportant features. Going beyond the dip both Ridge and Lasso bias increases, indicating that the model is underfitting.

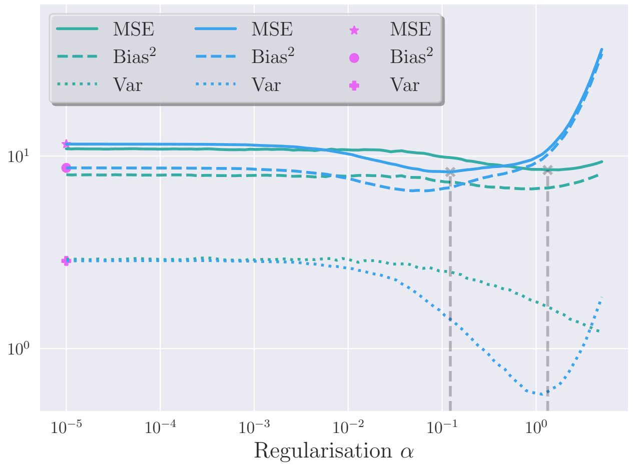


Figure 14: MSE, bias and variance estimates for OLS (pink), Ridge (green) and Lasso (blue) as a function of regularisation parameter α.

F.2 Tree Based Methods

We begin by fitting a single tree, gradually allowing for more complexity by increasing the allowed tree depth, shown in figure Fig. 15. In contrast to the linear regression models, the variance is significantly larger, overcoming the bias contribution for a tree of depth 2. The MSE values are generally quite large, with a minimum of 22.3 using a depth of 2 with 4 leafs. By considering the train MSE, we see that we quickly move into the realm of extreme overfitting. Allowing the tree to grow to a depth of 5 results in close to zero train MSE. A single tree, through the parameters searched here, does not result in a good model for this problem.

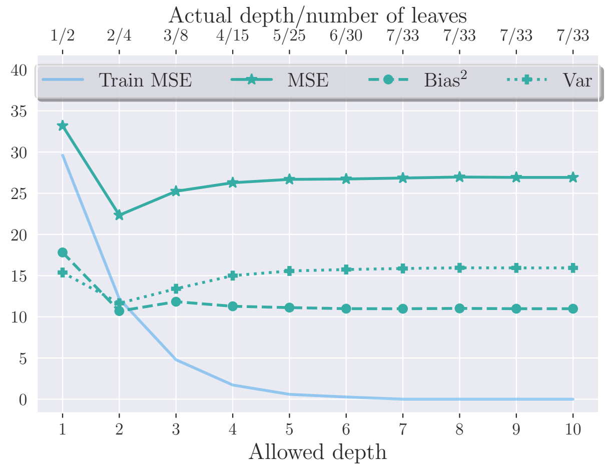


Figure 15: MSE, bias and variance estimates for a single tree against increasing relaxation of tree depth. Train MSE is also included to show the drastic overfitting.

Even though a single tree did not perform well, an ensemble of trees can often give better results. Varying the number of trees in our ensemble, both the BT and RF can be seen in Fig. 16. Comparing with the single tree from Fig. 15, we see a large improvement concerning both MSE, bias and variance. When adding more trees, the variance decreases. This agrees with the idea that many high variance trees can produce a smaller variance when their predictions are averaged. Similarly, the bias decreases, but not as drastically. The rectangular distribution of targets in feature space is still assumed in this model, therefor still adding quite a lot of bias. For comparison, the bias of simple OLS from Fig. 14 is smaller. When reaching an ensemble size of approximately 30 trees, only slight improvements of the MSE is seen. Comparing between BT and RF, we see that considering a subset of features does decrease variance, presumably due to less correlated trees. This however does not yield significant improvements in the MSE, with BT having a more stable test MSE across different ensemble sizes than RF.

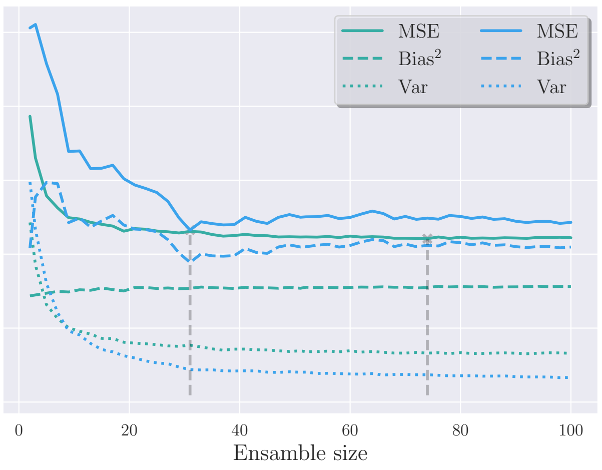


Figure 16: MSE, bias and variance estimates for bagged trees (green) and random forest (blue) considering $\lfloor p/3 \rfloor$ features as each split, against increasing bag size/number of trees.

For the last of our tree based methods, we investigate the boosted trees. The MSE, bias and variance calculated while varying the number of predictors is shown in Fig. 17. For a small ensemble size at around 10 predictors, the trees are not trained enough to give accurate score predictions, signifying that the model is underfitting. Increasing the number of predictors, the bias is decreased while the variance increases, indicating that the weak learners together has enough flexibility to mimic the underlying function. The smallest test MSE was found to be 10.89, using 17 weak learners.

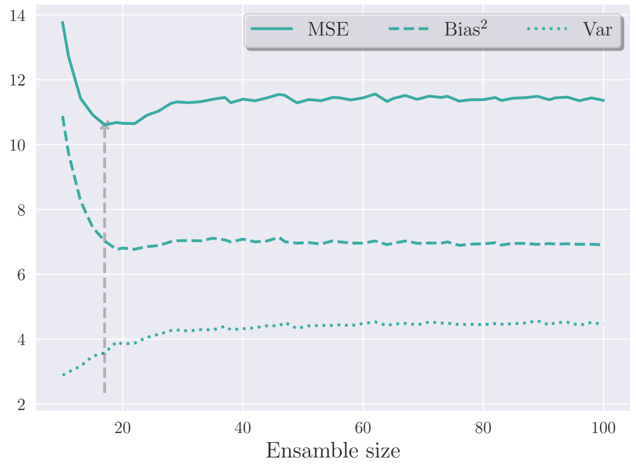


Figure 17: MSE, bias and variance estimates for boosted trees against the increasing ensemble size.

F.3 Support Vector Machines

Lastly, we investigate how our SVM performs when increasing the margin (ϵ). To assure convergence, the margin is slightly soft, setting $C = 1$. The results from both the linear and RBF kernel can be seen in Fig. 18. For small ϵ, the RBF kernel outperforms the linear kernel substantially. Though having approximately equal bias', the variance of RBF is much lower than the linear kernel. Increasing ϵ actually yields worse results for the RBF kernel, in contrast to the linear kernel where we find a low variance region with a bias dip similar to what we found for linear regression (Fig. 14). The increasing bias for high ϵ values might be somewhat counterintuitive, since the constraint in Eq. E.5 is loosened. However, the inclusion of many points might actually increase the bias since more points are used to center the line draw out by Eq. E.4. Assuming the bagged training sample is large enough to accurately represent the population, only very small variations of the coefficients $\boldsymbol{w}$ should be allowed. This is in agreement with the variance decreasing even more for very large ϵ in Fig. 18. These results yielded optimal MSE values of 8.58 and 11.76 at $\epsilon = 0.35, 0.005$ for the linear and RBF kernel respectively.

Can we characterize bias and variance in SVMs with respect to the kernel and its parameters?

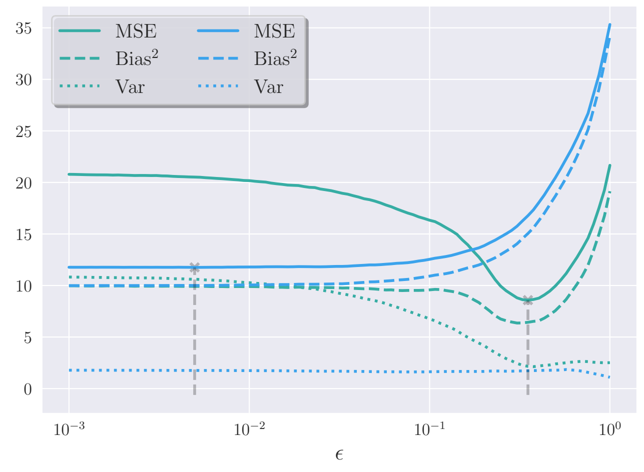


Figure 18: MSE, bias and variance estimates for SVR, against increasing the margin size ϵ. In green, we have the linear kernel while in blue the RBF kernel. A soft margin of $C = 1$ has been used across all ϵ.

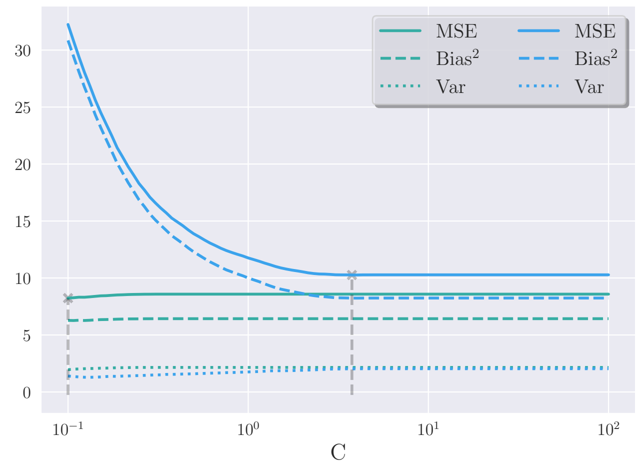


Figure 19: MSE, bias and variance estimates for SVR, against increasing the softness of the margin C. In green, we have the linear kernel while in blue the RBF kernel. The respective optimal margin size ϵ found for $C = 1$ has been.

Losning the margin parameterised by C, is shown in Fig. 19. Effects on the bias while losing the soft margin has clearer interpretation. With C being small, very few or even none of the prediction by f are allowed to lay outside the margin calculated by the train data. This is very clear when considering the RBF kernel, having a steeply increasing bias when C approaches zero. The linear kernel however, does not seem to be very dependent on margin softness. Why this is the case is not clear, but it might give an indication of a linear ϵ tube being a good representation of the true function. However, probing the bias of different kernels is a difficult task and no obvious comparison between the two methods can be drawn. Kernel specific parameters like γ for RBF, can also influence the bias, see for instance [Valentini and Dietterich, 2004]. The lowest test MSE was found to be 8.25 and 10.26 at $C = 0.10, 3.77$ for the linear and RBF kernel respectively.

F.4 Summarising Results

Through a study of the bootstrapped estimates for test MSE, bias and variance applied to player scores from FIFA21, we found that penalised linear regression and the linear kernel SVR performed the best. In general, the bias was larger than variance, except for the single regression tree and ensemble tree methods composed of a few trees. This is in agreement with decision trees in general being easily overfitted. Increasing the ensemble size did decrease the test MSE, with increasing variance for boosting and reducing it for BT and RF. We hypothesize that the scores calculated in FIFA21 are calculated using some linear relationship between features. Since we might have included irrelevant and excluding relevant features, the penalised linear models and linear kernel SVMs are natural choices. Moving forward, it would be interesting to make some polynomial features for the linear regression models, introducing more parameters to fit and presumably reducing bias. For SVR, more extensively searches tuning kernel parameters in addition to trying more kernels could potentially improve results.

References

Anna Hjertvik Aasen, Carl Martin Fevang, and Håkon Kvernmoen. Bias-variance tradeoff in simple linear models, 2022a. URL https://github.com/hkve/FYS-STK4155/blob/main/Project1/tex/main.pdf.

Anna Hjertvik Aasen, Carl Martin Fevang, and Håkon Kvernmoen. Optimising simple neural networks as regressors and classifiers, 2022b. URL https://github.com/hkve/FYS-STK4155/blob/main/Project2/tex/main.pdf.

Mustafa A. Al-Asadi and Sakir Tasdemır. Predict the value of football players using fifa video game data and machine learning techniques. *IEEE Access*, 10:22631–22645, 2022. doi: 10.1109/ACCESS.2022.3154767.

American Psychological Association . Perception of high and low spatial frequency information in pigeons and people, 2015. URL https://www.apa.org/pubs/highlights/peeps/issue-59. Accessed: 2022-12-15.

Aurelie Bellemans, Gianmarco Aversano, Axel Coussement, and Alessandro Parente. Title: Feature extraction from principal component analysis based reduced-order models using orthogonal rotation, 03 2018.

Yanqing Chen. Mechanisms of winner-take-all and group selection in neuronal spiking networks, 2017. URL https://www.frontiersin.org/articles/10.3389/fncom.2017.00020/full.

R.A. Cohen. Lateral inhibition. in: Kreutzer, j.s., deluca, j., caplan, b. (eds) encyclopedia of clinical neuropsychology., 2011.

Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In M.C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996. URL https://proceedings.neurips.cc/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. 2013. doi: 10.48550/ARXIV.1302.4389. URL https://arxiv.org/abs/1302.4389.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/abs/1502.01852.

Corentin Herbinet. Predicting football results using machine learning techniques, 2018. URL shorturl.at/joK67.

L. Itti and C. Koch. Computational modelling of visual attention., 2001.

E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth. Principles of neural science, 2000.

Ron Kohavi and David Wolpert. Bias plus variance decomposition for zero-one loss functions. 09 1997.

T. Kohonen. Self-organized formation of topologically correct feature maps, 1982. URL https://link.springer.com/article/10.1007/BF00337288#citeas.

Steve Kroon and Christian Omlin. Getting to grips with support vector machines: Theory. *South African Statistical Journal*, 38, 01 2004.

David C. Lay. *Linear Algebra and its Applications*. Pearson Education Limited, fifth edition, 2016.

Stefano Leone. Fifa 21 complete player dataset, 09 2020. URL https://www.kaggle.com/datasets/stefanoleone992/fifa-21-complete-player-dataset.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL http://jmlr.org/papers/v18/16-558.html.

Hannah Jane Parkinson. Paul the octopus, taiyo the otter and the world cup's other psychic animals. *The Guardian*, 2022. URL https://www.theguardian.com/sport/2022/dec/12/paul-the-octopus-taiyo-the-otter-world-cup-psychic-anim

Alex Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 08 2004. doi: 10.1023/B%3ASTCO.0000035301.49549.88.

SoccerSTATS.com. Premier league: League statistics 19/20. URL https://www.soccerstats.com/latest.asp?league=england_2019. Accessed: 2022-12-16.

David Sumpter. *Soccermatics: Mathematical Adventures in the Beautiful Game*. Bloomsbury Publishing, 2016.

Giorgio Valentini and Thomas Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. volume 5, pages 725–775, 07 2004. ISBN 978-3-540-43818-2. doi: 10.1007/3-540-45428-4_22.

Q. Wang and J. Jaja. From maxout to channel-out: Encoding information on sparse pathways, 2013. URL https://arxiv.org/abs/1312.1909.