LU TP 14-nn
January 2017

Applying Dropout to Prevent Shallow Neural Networks from Overtraining

Denhanh Huynh

Department of Astronomy and Theoretical Physics, Lund University

Bachelor thesis supervised by Mattias Ohlsson

Abstract

Artificial neural networks are machine learning systems based on the neural networks of the human brain. A problem that has to be overcome for neural networks is overtraining, which means that the network performs well on data that has been used for training the network, but does not make good predictions on new data. One branch of artificial neural networks, called deep neural networks, uses a lot of hidden layers of neurons to produce state-of-the-art results on a wide variety of problems. Because of the size of these networks, training requires a lot of computation, and some methods for dealing with overtraining that are available for shallow neural networks, with only a few hidden layers, become impractical. Dropout is a recently developed method to reduce overtraining without being too computationally demanding for deep neural networks. In this project, however, dropout is applied to shallow neural networks, and in this thesis it is shown that dropout is a good way to reduce overtraining in shallow neural networks on a variety of classification problems.

Populärvetenskaplig sammanfattning

Dagens samhälle blir mer och mer datoriserad och tekniken blir allt mer avancerad. I teknikens framkant finner man artificiell intelligens. Idag finns det artificiell intelligens som kan vinna över världsmästare i en mängd olika brädspel som schack och go, kan analysera och förstå bild och tal, och hjälpa läkare med diagnosering av patienter. Artificiell intelligens tillämpas också i självkörande bilar. Detta är möjligt på grund av utvecklingen av artificiella neurala nätverk, som är baserade på neuronerna och och de komplexa neurala nätverken i hjärnan. En relativt ny metod kallad dropout har visats kunna förbättra förmågan hos djupa neurala nätverk att kunna lösa nya uppgifter baserat på tidigare exempel. Detta projektet handlar om att studera dropout genom att tillämpa metoden på små nätverk. Små nätverk har fördelarna att de är lättare att implementera och går fortare att träna, vilket gör det enklare att testa metoden på en mängd olika problem.

Contents

1 Introduction

The goal of this project is to study the effect of the dropout method, when applying it to shallow networks.

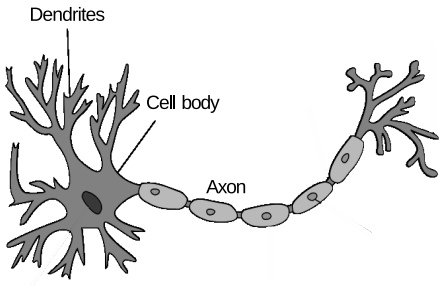
This introduction aims to give an understanding of neural networks, and how they are implemented.

1.1 Simple Perceptron

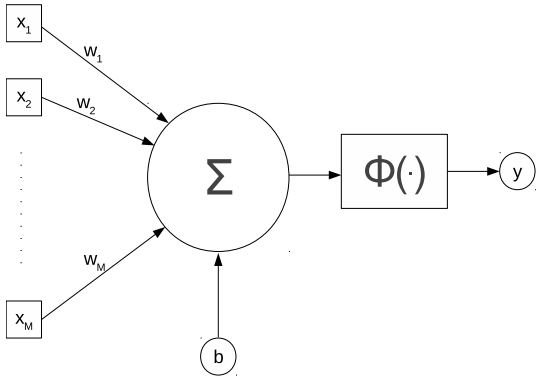
1.1.1 The biological neuron

The algorithms used in artificial neural networks are inspired by the nerve cells of the human brain, called neurons. In 1943, Warren Sturgis McCulloch, an American neuropsysiologist, and Walter Harry Pitts, a logician working in the field of computational neuroscience, published a paper called "A Logical Calculus of Ideas Immanent in Nervous Activity", where they proposed a simple model of how a biological neuron might work [1].

The cell body receives electrical signals from other neurons or sensory cells via the dendrites (fig.1a). The input signals are processed in the cell body and the generated output signals are transmitted through the axon which is connected to the dendrites of other neurons. These connections are called synapses and can have varying strengths, and they can either excite or inhibit the receiving neuron, making it more likely or less likely to fire a signal.



(a) A biological neuron [2]. The input signals go into the neuron via the dendrites. The signals are processed in the cell body, and new signals are transmitted via the axon.



(b) A simple perceptron. The inputs x_m are multiplied by their respective weights w_m, and summed up along with the bias b. The sum is then passed through the activation function $\phi(x)$ to finally give the output y.

Figure 1: A comparison between the biological neuron and the simple perceptron with one output node.

1.1.2 The artificial neuron

The simplest model of an artificial neural network is the artificial neuron, also referred to as the simple perceptron, which was introduced by Frank Rosenblatt in 1958 [3, p.10]. The simple perceptron has an input layer and an output layer. Each input node is connected to each output node with weights deciding the strength of the connection. These weights can be either positive or negative, leading to constructive interference if two weights have the same sign or destructive interference if the weights have different signs.

With only one output node the similarities between the neuron and the simple perceptron become apparent (fig.1b). The simple perceptron sums up all the M inputs x_m, multiplied by their assigned weights w_m and adds a bias b. The sum is then passed through an activation function $\phi(x)$, and the output y from the simple perceptron is given by the output from this activation function. Mathematically this can be written as:

$$h = \sum_{m=1}^{M} w_m x_m + b$$

$$y = \phi(h). \tag{1.1}$$

By defining a bias weight $w_0 = b$ and a bias input $x_0 = 1$ the equation can be simplified to:

$$y = \phi(\sum_{m=0}^{M} w_m x_m) = \phi(\mathbf{w}^T \mathbf{x}) \tag{1.2}$$

$$\mathbf{w} \equiv \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}; \qquad \mathbf{x} \equiv \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix} \tag{1.3}$$

1.1.3 Decision plane

The McCullock-Pitts model uses a sharp activation function given by:

$$\phi(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases} \tag{1.4}$$

The plane given by $h = 0$ is the decision plane, and divides the M-dimensional input space into two regions where $h > 0 \Rightarrow y = 1$ and $h \leq 0 \Rightarrow y = -1$. This is easily illustrated with the AND-problem, where the inputs are 2-dimensional and the decision plane becomes a line (table 1 and fig.2). During a process called learning, the decision plane is adjusted so that the outputs y matches the given target outputs d. This is done by tuning the weights w_m (including the bias weight w_0). The learning process is also called training

the network [3, p.12-14].

x_1	x_2	d
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Table 1: The AND-problem. d is the target output for the network given the inputs x_1 and x_2.

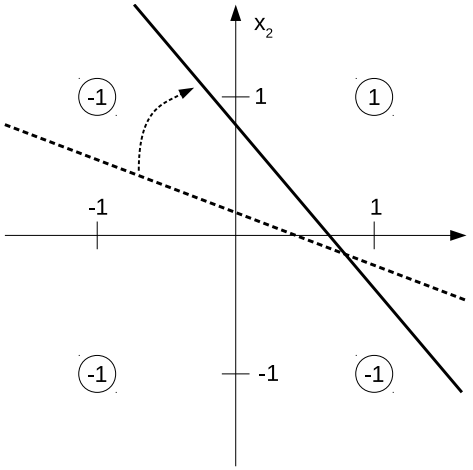


Figure 2: The AND-problem. This shows how the decision plane might move during the learning process to separate the data points into two groups, one with $d = 1$ and one with $d = -1$.

x_1	x_2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Table 2: The XOR-problem.

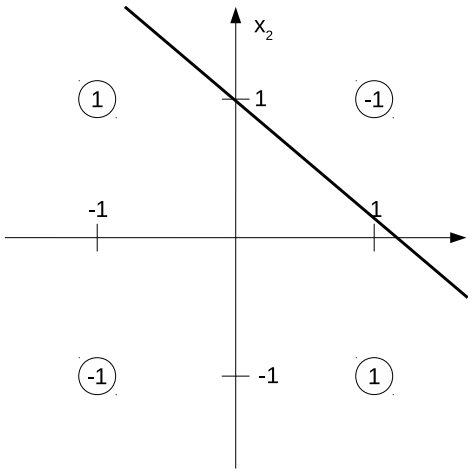


Figure 3: The XOR-problem. It is impossible to separate the data points into one group with targets $d = 1$ and another group with targets $d = -1$ using only one straight line. There will always be at least one data point misplaced.

1.2 Multilayer Perceptron

Some problems are unsolvable with the simple perceptron. A simple example is the XOR-problem (table 2 and fig.3). To solve these problems, one or more hidden layers of nodes are introduced. With one hidden layer, the hidden nodes can be seen as simple perceptrons with the same inputs, where each node creates a decision plane. This allows the network to work

with several decision planes and combine them to create complex decision boundaries. The resulting structure is called a multilayer perceptron, commonly referred to by its acronym, MLP. One of the simplest MLP-models is the fully connected feed-forward MLP, where all nodes from one layer are connected to all nodes in the next layer (fig.4), which means that the outputs from one layer are the inputs for the nodes in the next layer.

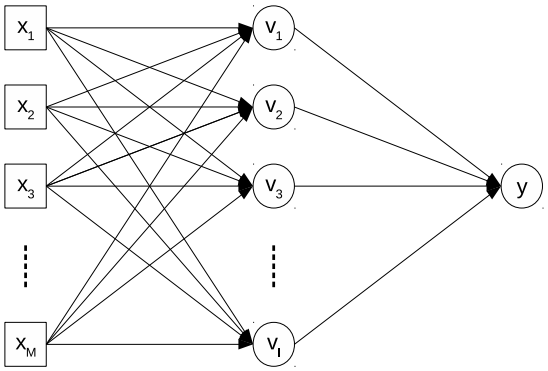


Figure 4: A multilayer perceptron with M inputs, one hidden layer of I hidden nodes with outputs v_i, and an output layer with one output node y. For simplicity, the summations and the activation functions for each node are often not drawn out. Note that a general MLP might have more than one output.

With the nodes, weights, and activation function denoted v_i, $w_{im}^{(1)}$ and $\phi^{(1)}(x)$ for the hidden layer, and y, $w_i^{(2)}$ and $\phi^{(2)}(x)$ for the output layer, the output of an MLP with one hidden layer and one output node is given by (compare with eq.1.1):

$$v_i = \phi^{(1)}\left(\sum_{m=0}^{M} w_{im}^{(1)} x_{im}\right) \tag{1.5}$$

$$y = \phi^{(2)}\left(\sum_{i=0}^{I} w_i^{(2)} v_i\right) \tag{1.6}$$

Usually the activation function for the output layer $\phi^{(2)}(x)$ is chosen to be a sigmoidal function if the network is supposed to solve a classificasion problem, or a linear function if the network is supposed to solve a regression problem. However, the activation function for the hidden layer needs to be a nonlinear function (usually sigmoidal). It can be proven that a linear activation function for the hidden layer makes the MLP equivalent to a simple perceptron $[3, p.15, 21-23]$.

1.3 Gradient Descent Learning

A problem can be given by a data set of N input-output patterns. The AND-problem and the XOR-problem, for example, both have 4 input-output patterns (see table 1 and 2). Given a data set of N input-output patterns an error function can be defined as:

$$E = \frac{1}{2N}\sum_{n=1}^{N}(d(n) - y(n))^2 \tag{1.7}$$

where $y(n)$ is the output of the network and $d(n)$ is the target output given an input signal $(x_1(n), x_2(n), ..., x_M(n))$.

The function given by equation 1.7 is the mean square error function and is a measurement of how well the network performs on a given problem. Since all terms in the sum are positive, an error of $E = 0$ would mean that all data points are classified correctly. The learning process aims to minimize E as a function of the weights $(w_0, w_1, w_2, ..., w_M)$ (which affects the output of the network y). This can be done with gradient descent learning, which has the following update rule:

$$w_{im}^{(1)} \to w_{im}^{(1)} + \Delta w_{im}^{(1)}; w_i^{(2)} \to w_i^{(2)} + \Delta w_i^{(2)} \tag{1.8}$$

$$\Delta w_{im}^{(1)} = -\eta\frac{\partial E}{\partial w_{im}^{(1)}}; \Delta w_i^{(2)} = -\eta\frac{\partial E}{\partial w_i^{(2)}} \tag{1.9}$$

where η is a parameter called the learning rate.

Updating the weights using all input-output patterns for each update is called batch updating. In practice, this is often too inefficient as the gradients for each data point in the whole data set have to be calculated for just one update. Therefore, block updating is often preferred. Block updating uses a few patterns for each update (typically 10-50 patterns). Gradient descent with block updating is called mini-batch gradient descent [4].

The amount of training on a network is measured in epochs. When the updating has cycled through all patterns once, one epoch has passed. In the case of batch updating, each update is an epoch, whereas for block updating, several updates are required to cycle through all patterns.

2 Dropout

2.1 Deep neural networks

In theory, a multilayer perceptron can approximate any continuous function to an arbitrary degree with only one hidden layer, given that it has enough hidden nodes. The problem however, is to find the correct amount of hidden nodes and to find a way to train the weights to their optimal value. In practice more hidden layers are usually added to solve more complex problems. Artificial neural networks that uses a large number of hidden layers and parameters, called deep neural networks, are very powerful models that are used to solve a wide variety of problems, including image recognition and speech recognition [5].

2.2 Overtraining

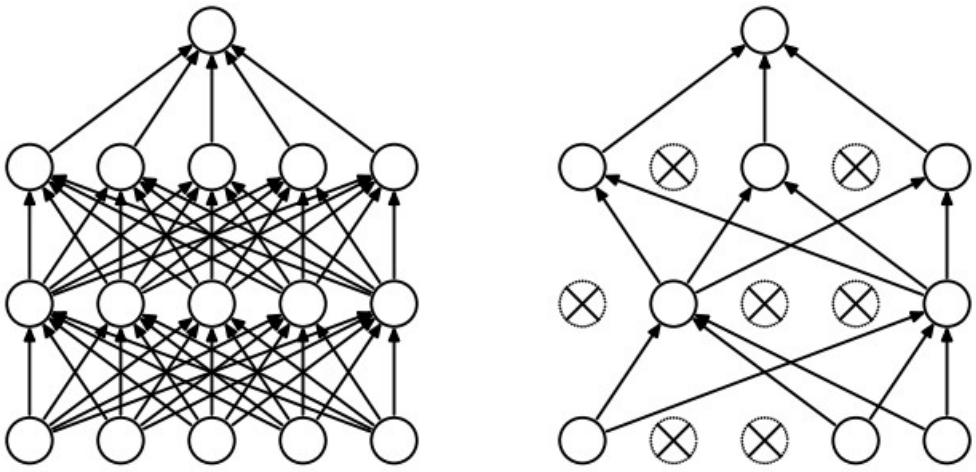
A problem with these networks, however, is overtraining. As you increase the size of the networks (by adding more hidden layers and more hidden nodes), the decision boundaries created by the networks can become increasingly complex. Overtraining occurs when the decision boundaries conforms to the training set too much in detail, giving a very good performance on the training set, but a poor performance on new data that has not been used for training [6]. One common solution when working with shallow networks (with few layers and nodes) is to train several neural networks and create an ensemble output. The ensemble output can be the average output of the networks or some kind of weighted output. However, because of the size of deep neural networks, training requires a lot of computation, and creating an ensemble of many networks may not be a practical solution. Other ways of dealing with overtraining include early stopping, which is a method where you stop the training as soon as the validation performance get worse, and using weight penalties, which limits the complexity of the network.

2.3 The Dropout Method

Dropout is a recently developed technique that deals with the problem of overtraining without being as computationally demanding as an ensemble of networks [7]. For each data point during training, a thinned network is sampled and trained. The thinned networks are sampled by randomly dropping out nodes with a predefined probability given by $1 - p$. A dropped out node is temporarily removed from the network, along with its connections (fig.5).

When simulating the network, for instance when validating the network, no nodes are dropped out. Instead, the output of each node is multiplied by their probability of being retained p. This gives the same value as the expected output of the nodes during training.

Dropout can be seen as a way of approximating an ensemble of 2^n possible thinned networks, where n is the number of nodes in the network, excluding output nodes (since they are not dropped out).



(a) Standard Neural Net (b) After applying dropout.

Figure 5: (b) shows a possible thinned network during training after dropout has been applied to the network shown in (a). The crossed out nodes has been dropped out. [7]

2.3.1 Network Size

With n hidden nodes and the probability of retaining a hidden node during training given by p_h, there will on average be $p_h n$ hidden nodes present after dropout. A network with fewer nodes will have a lower capacity, meaning that it will not be able to learn as complex features as a network with more nodes. Hence, to retain the complexity of the network, a standard MLP with n hidden nodes will be compared to a dropout network with n/p_h hidden nodes, rounded to the nearest integer [7].

3 Methods

3.1 Validation

The error function given by equation 1.7 can also be used to evaluate the performance of a network.

The training performance, that is, the error when evaluating the network on data points that were included in the training, can become very close to zero by using a lot of nodes and by training the network for a very long time. However, this is most often not a good measurement of how the network performs on new data sets.

Since the idea is to use neural networks to get solutions that you do not already know the answer to, the generalization performance, that is, the performance on data that were not part of the training is most often of greater value. One way to estimate the generalization performance is to split the given data set into two parts, creating a training set and a validation set. When the neural network has been trained using the training set, the validation set can be used to estimate the generalization performance, since it has not been used during the training.

However, there are some drawbacks to this method. Since a part of the data set has been set aside for validating the network, not all information available is used for improving the network. Also, if the validation set is too small, the estimation of the generalization performance will not be accurate.

3.1.1 K-fold cross validation

To overcome these drawbacks, another method called K-fold cross validation, which uses the given data in a more efficient way, may be used [3, p.52-53]. The data set is split into K parts of approximately equal size. K models are then trained on K slightly different training data sets created by removing one part from the data set (fig.6). The part that has been removed from the training data set is the corresponding validation data set and is used to give a generalization performance. The average of the K validation results is used as the final estimate of the generalization performance. This way, all data is used for both training and validation without validating a network using the same data it has been trained with.

3.2 Problem description

In this project, the effect of the dropout method applied to shallow networks, and how well it reduces overtraining, was studied. The data sets used are described in the next subsection.

3.2.1 Network Structure and Training

The networks used were multilayer perceptrons with only one hidden layer. All the data sets used in this project were classification problems with one dimensional outputs, and

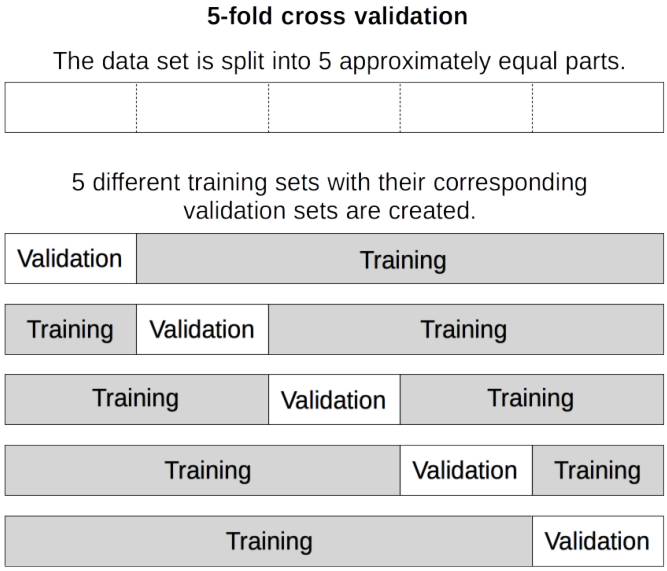


Figure 6: K-fold cross validation with $K = 5$. Each of the 5 parts is used as a validation set for a model trained on the remaining parts.

hence, the networks only had one output node. The activation function used for the hidden nodes was:

$$\phi^{(1)}(x) = tanh(x), \tag{3.1}$$

whereas, since the classes (i.e. the outputs) were given by either 0 or 1, the output activation function was set to a logistic function given by:

$$\phi^{(2)}(x) = \frac{1}{1 + exp(-x)} \tag{3.2}$$

This function is bounded between 0 and 1, in contrast to the hidden activation function $tanh(x)$ which is bounded between -1 and 1 (fig. 7).

The networks were trained using mini-batch gradient descent with 10 data points in each block update.

3.2.2 Learning rate

A dynamical learning rate was used. A few different starting values for the learning rate η was tested. However, the impact of changing the starting value was not big, and a good value to start at was $\eta = 0.01$ for all problems. The learning rate was modified during training according to:

$$\eta_{t+1} = \begin{cases} \eta_t \cdot \gamma & \text{if } E_{t+1} \geq E_t \\ \eta_t \cdot \left(1 + \frac{1-\gamma}{10}\right) & \text{if } E_{t+1} < E_t \end{cases} \tag{3.3}$$

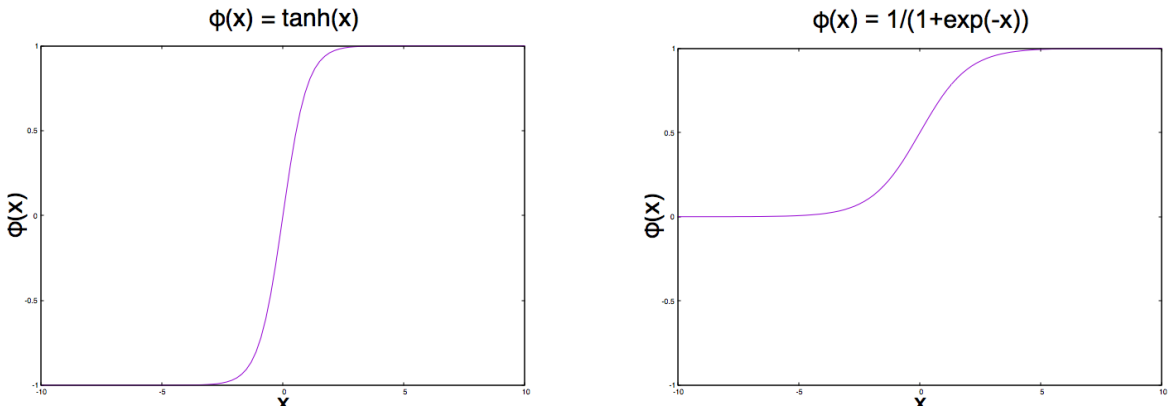


Figure 7: The hyperbolic function $tanh(x)$ (left) and the logistic function $\phi(x) = \frac{1}{1+exp(-x)}$ (right). The outputs of the hyperbolic function are bounded between -1 and 1 and the outputs of the logistic function are bounded between 0 and 1.

where E_{t+1} and E_t are the error rates after $t+1$ and t epochs. The scale factor γ was set to $\gamma = 0.999$.

3.2.3 Normalization of inputs

Before any training, the inputs of the data sets were normalized according to:

$$x_m = \frac{x_m - < x_m >}{\sigma_{x_m}}. \tag{3.4}$$

where $< x_m >$ is the mean value of all inputs in the data set with index m and σ_{x_m} is the standard deviation.

3.2.4 Validation

The generalization performance was estimated using 5-fold cross validation (fig. 6).

3.2.5 Implementation

All software used in this project was implemented from scratch in Java using Eclipse (IDE).

3.3 Data Sets

3.3.1 Pima

This data set is from a study where a population was tested for diabetes [8]. The inputs contain information about the individuals and the output is either a one, if they tested positive, or a zero, if they tested negative. The data set contains 768 data points where about a third of the data points has the output one. Each data point has eight input values.

3.3.2 NL1

This is an artificial non-linear problem with 1000 data points where each data point contains a total of eight randomly generated inputs. The first four inputs x_1, x_2, x_3, x_4 are normally distributed variables whereas the other four inputs b_1, b_2, b_3, b_4 are binary variables. Using these random variables a value y was calculated using the following formula:

$$y = x_1x_2 + x_3x_4 + 2x_1^2 + 2x_2^2 + 2sin(2\pi x_3) + b_1b_2 + 3b_3b_4. \tag{3.5}$$

The data points were separated into two classes. The 500 data points with the highest value of y belonged to class 1 and the others belonged to class 0.

3.3.3 NL2

This is an altered version of the previous data set where each data point gets an additional four normally distributed random variables. These variables do not affect the class of the data point and only serve to add noise, making the network more susceptible to overtraining.

3.3.4 NL2 - halved data set

500 data points from the NL2 data set was chosen at random. This data set was created to further increase the likelihood of overtraining.

3.3.5 Pancreat

This data is from a study trying to distinguish patients with pancreatic cancer from healthy controls using biomarkers (in this case protein content in the blood). This data set contains 229 datapoints where about half of them has output 1 and the other half has output 0. The data points have 253 input values.

4 Results

The figures in this section show the mean square error (eq. 1.7) as a function of the amount of training (measured in number of epochs) for several different network sizes. In the figures showing the results for the dropout networks, p_h denotes the retaining rate of the hidden nodes, and p_i denotes the retaining rate of the input nodes.

4.1 Pima

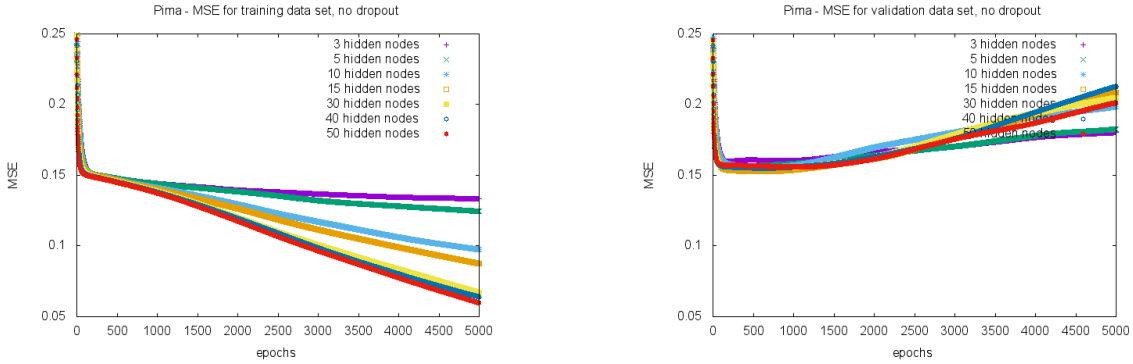


Figure 8: The training errors (left) and validation errors (right) when training on the Pima data set without applying dropout.

The Pima data set was tested with several different networks with varying amounts of hidden nodes. This was first done without dropout to see how susceptible the networks were to overtraining. A clear case of overtraining can be seen in fig. 8. The training errors decrease as the number of hidden nodes and the number of epochs increases. The validation errors, however, increase with the number of epochs after reaching their optimal values at 0.16 ± 0.01 (these are listed in table 3 as best validation errors without dropout), and the lowest validation errors after 5000 epochs were achieved with the fewest number of hidden nodes (3 and 5).

To find a good drop rate for the hidden layer, several different networks with varying value of the drop rate were tested with $p_h n$ held constant at $p_h n = 50$ (fig. 9). The reason for choosing a high value of $p_h n$ is that a network with more capacity is more susceptible to overtraining, and that the effect of applying dropout would be clearer. Even with low dropout, the validation error of the network is improved. However, the optimal value of p_h seems to be around $p_h = 0.7$ (30% dropout), as the error does not decrease appreciably even when applying 90% dropout. Note that there will always be a slight variation because of the randomness of the starting values of the weights as well as the randomness of dropout.

With 30% dropout on the hidden layer, barely any overtraining is observed (fig. 10), even for the largest network with 71 hidden nodes (note that the corresponding network without dropout has $0.7 \cdot 71 \approx 50$ hidden nodes). Since the network size does not affect the validation performance, the smallest network, in this case with 4 hidden nodes would

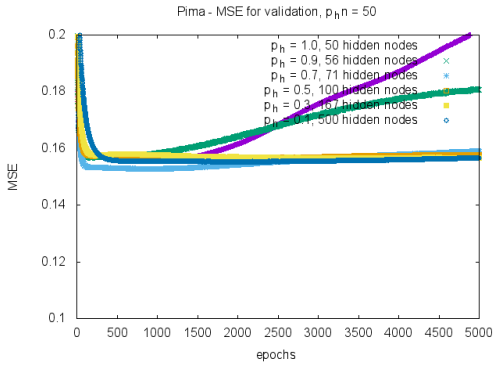


Figure 9: The validation errors for networks with varying dropout for the hidden layer and no drop out on the input layer, where $p_h n$ was held constant at $p_h n = 50$. The validation error of a network with no dropout is included for comparison.

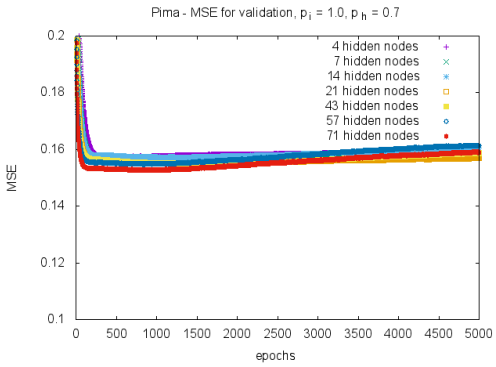


Figure 10: Validation errors when training on the Pima data set with 30% dropout on the hidden layer. The corresponding networks in this figure and the ones in figure 8 have the same color.

be preferred because less computing is needed to update the network, ultimately leading to a reduction of time needed to train the network.

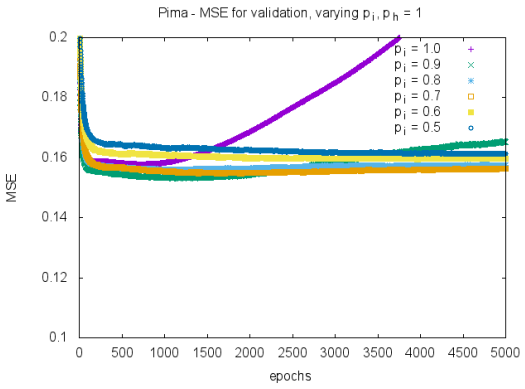


Figure 11: The validation errors for varying drop rates on the input layer. All networks used in this test had 50 hidden nodes.

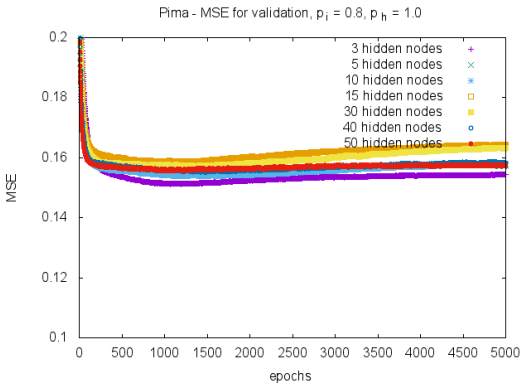


Figure 12: Validation errors when training on the Pima data set with 20% dropout on the input layer.

Retaining rates for the input nodes ranging from 0.5 to 0.9 were tested on a network with 50 hidden nodes (fig. 11).

Almost the same result as 30% drop rate on the hidden layer was achieved by applying 20% dropout on the input layer (fig. 12). Since all hidden nodes are retained during training, the complexity of the decision boundaries are kept. Thus the size of the networks do not need to be changed when applying dropout to the input layer, keeping the networks smaller. As mentioned before, this is preferred because of reduction in training time.

No improvements could be observed when dropout was applied on both the input layer and the hidden layer at the same time.

Table 3: The second column lists the best validation errors reached for each network size with no dropout applied to the networks. The columns 3, 4, and 5 lists the validation errors after 5000 epochs.

$p_h n$	best validation errors without dropout	validation errors without dropout	validation errors with $p_h = 0.7$	validation errors with $p_i = 0.8$
3	0.16083	0.17047	0.15952	0.15435
5	0.15729	0.17868	0.15769	0.15737
10	0.15650	0.19028	0.16041	0.15803
15	0.15497	0.20516	0.15684	0.16437
30	0.15901	0.20724	0.15868	0.16323
40	0.15904	0.21105	0.16131	0.15835
50	0.15781	0.22333	0.15889	0.15741

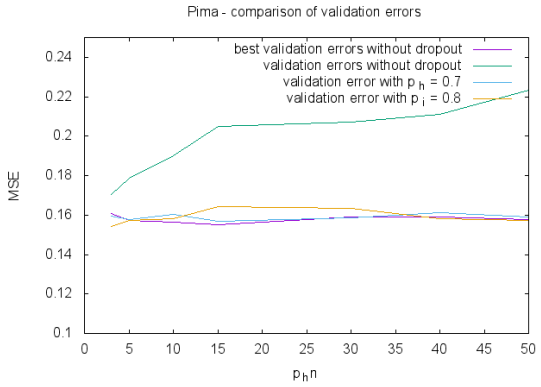


Figure 13: The purple line shows the best validation errors reached with no dropout applied to the networks. The other lines show the validation errors after 5000 epochs. The exact values are listed in table 3.

The validation errors when applying dropout reach their optimal value and do not increase with further training. The validation errors after 5000 epochs with and without dropout were compared to the optimal validation errors reached without any dropout (table 3 and fig. 13). Fig. 13 shows us that dropout reduces the overtraining without limiting the validation performance of the network, and that no additional regularization of the networks is needed.

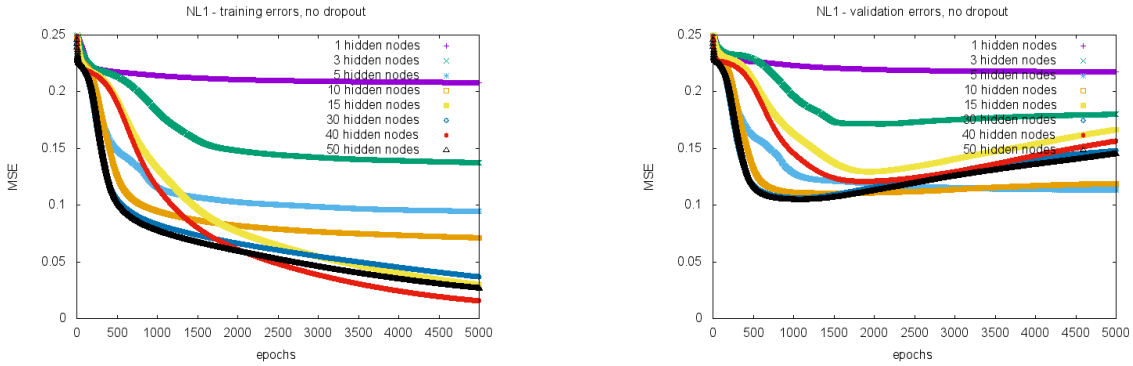


Figure 14: The training errors (left) and validation errors (right) when training on the NL1 data set without dropout.

4.2 NL1

The training and validation errors when training on the NL1 data set is shown in fig. 14. More hidden nodes, and thus more complex decision boundaries are required to get good validation performance, which indicates the non linearity of the problem. For the networks with more than 15 hidden nodes, overtraining can be seen after a certain amount of epochs where the validation errors increases as the training errors decreases. However, despite the absence of overtraining for the smallest network, it is still outperformed by the larger networks. The networks with 3, 5, and 10 hidden nodes only have slight overtraining.

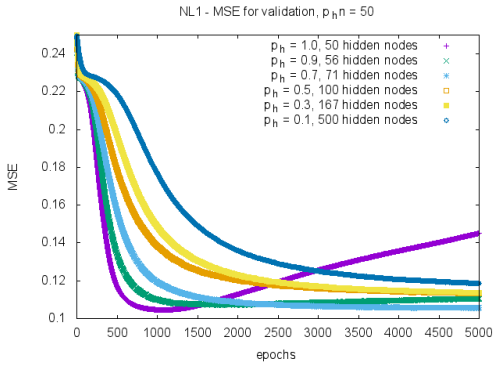


Figure 15: The validation errors for networks, trained on the NL1 data set, with varying dropout for the hidden layer, and no drop out on the input layer. $p_h n$ was held constant at $p_h n = 50$.

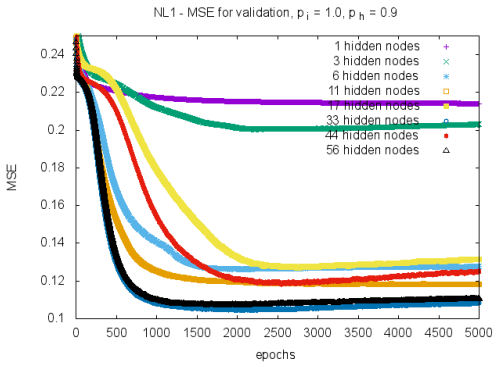


Figure 16: Validation errors when training on the NL1 data set with 10% dropout on the hidden layer.

For the NL1 data set a 10% drop rate on the hidden layer was enough to remove most of the overtraining (fig. 15). One thing to note here, is that when the drop rate is 90%, that is, when $p_h = 0.1$, the error reduces at a slower rate compared to the other

networks with dropout. This is due to a lower percentage of weights being trained at each update, meaning that on average, more epochs will be needed for a given weight to be updated, and therefore, more epochs will be needed for a given weight to reach a good value. Additionally, since the size of the network is greater, each individual epoch will require more calculations and hence more time.

With 10% dropout on the hidden layer, a clear reduction of overtraining on the larger networks can be seen in fig. 16.

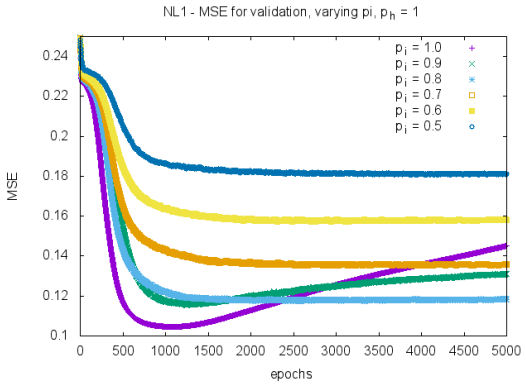


Figure 17: The validation errors for varying drop rates on the input layer. All networks used in this test had 50 hidden nodes.

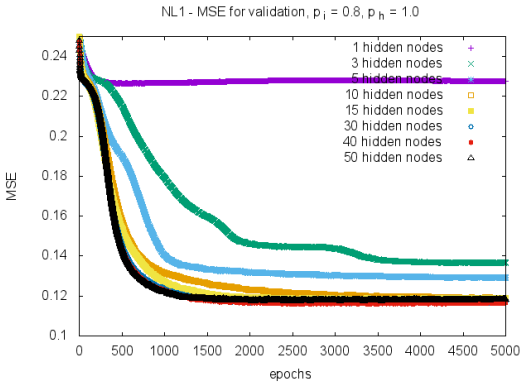


Figure 18: Validation errors when training on the NL1 data set with 20% dropout on the input layer.

The best input drop rate for the NL1 data set was 20% (fig. 17). With a higher drop rate than 20% for the input layer the error increased with increasing drop rate. Compared to applying a 10% drop rate on the hidden nodes (fig. 16), all networks except for the network with 3 hidden nodes had a similar result when applying 20% dropout on the inputs (fig. 18). However, the error of the network with 3 hidden nodes was decreased by 32%.

Table 4: The second column lists the best validation errors reached for each network size with no dropout applied to the networks. The columns 3, 4, and 5 lists the validation errors after 5000 epochs.

$p_h n$	best validation errors without dropout	validation errors without dropout	validation errors with $p_h = 0.9$	validation errors with $p_i = 0.8$
1	0.21762	0.21764	0.21413	0.22775
3	0.17160	0.18008	0.20300	0.13665
5	0.11348	0.11360	0.12757	0.12921
10	0.11010	0.11905	0.11821	0.11900
15	0.12945	0.16661	0.13155	0.11905
30	0.10632	0.14816	0.10822	0.11700
40	0.12095	0.15658	0.12512	0.11675
50	0.10449	0.14504	0.11056	0.11820

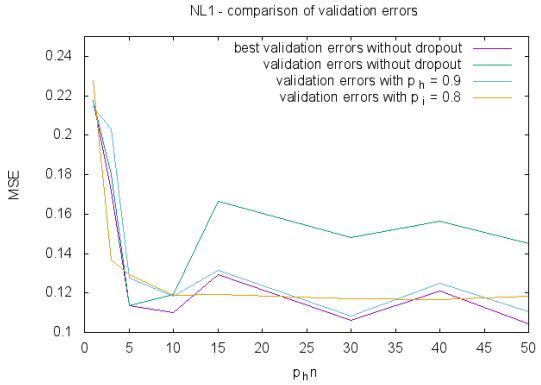


Figure 19: The purple line shows the best validation errors reached with no dropout applied to the networks. The other lines show the validation errors after 5000 epochs. The exact values are listed in table 4.

The validation errors after 5000 epochs with and without dropout were compared to the optimal validation errors reached without any dropout (table 4 and fig. 19). No sign of loss in validation performance can be seen when dropout is applied.

4.3 NL2

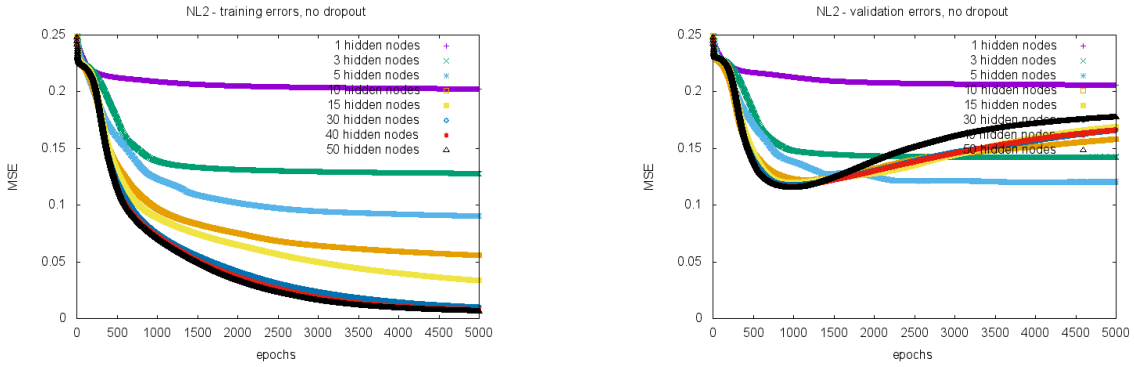


Figure 20: The training errors (left) and validation errors (right) when training on the NL2 data set without dropout.

Training neural networks on the NL2 data set, which is an alternated version of NL1, showed similar results as NL1. The networks with 15 hidden nodes or more had a slight increase of the validation errors and a slight decrease in training errors, signifying a slight increase in overtraining. This was to be expected since noise was added into the data set. The network with 10 hidden nodes had the most increase in overtraining with the validation error increasing from 0.12 to 0.15 and the training error decreasing from 0.071

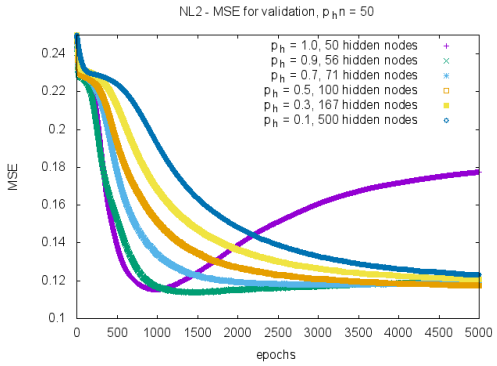


Figure 21: The validation errors for networks, trained on the NL2 data set, with varying dropout for the hidden layer, and no drop out on the input layer. $p_h n$ was held constant at $p_h n = 50$.

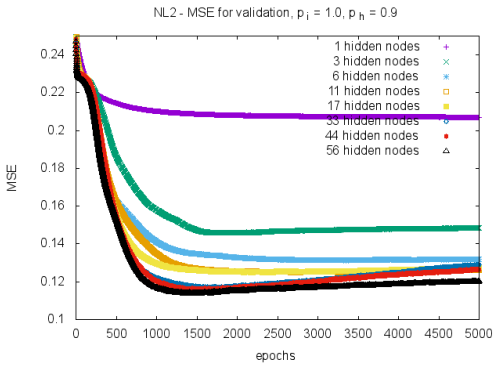


Figure 22: Validation errors when training on the NL2 data set with 10% dropout on the hidden layer and no drop out on the inputs.

to 0.056. The networks with 1 and 3 hidden nodes however, showed a decrease in both training and validation errors.

As for the NL1 data set, 10% dropout on the hidden layer was enough to reduce most of the overtraining when training networks with $p_h n = 50$ on the NL2 data set (fig. 21). This was also true for the other network sizes (fig. 22).

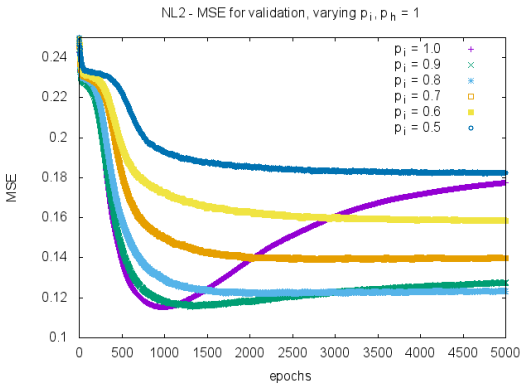


Figure 23: The validation errors for varying drop rates on the input layer. All networks used in this test had 50 hidden nodes.

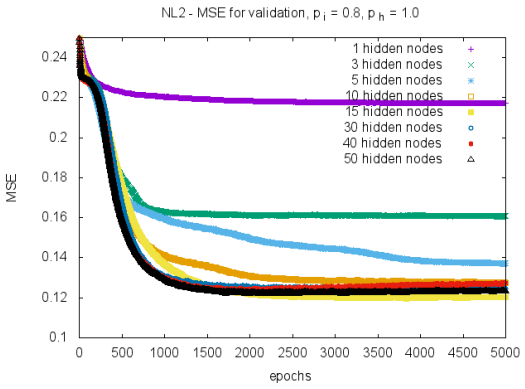


Figure 24: Validation errors when training on the NL2 data set with 20% dropout on the input layer.

A 20% drop rate was the optimal value for the input layer (fig. 23). Again, the results are very similar to those of the NL1 data set (fig. 24).

Table 5: The second column lists the best validation errors reached for each network size with no dropout applied to the networks. The columns 3, 4, and 5 lists the validation errors after 5000 epochs.

$p_h n$	best validation errors without dropout	validation errors without dropout	validation errors with $p_h = 0.9$	validation errors with $p_i = 0.8$
1	0.20598	0.20599	0.20707	0.21730
3	0.14237	0.14257	0.14846	0.16080
5	0.11990	0.12057	0.13188	0.13721
10	0.12227	0.15819	0.12603	0.12755
15	0.11980	0.16900	0.12761	0.12032
30	0.11784	0.16614	0.12885	0.12461
40	0.11649	0.16628	0.12655	0.12729
50	0.11526	0.17746	0.12015	0.12311

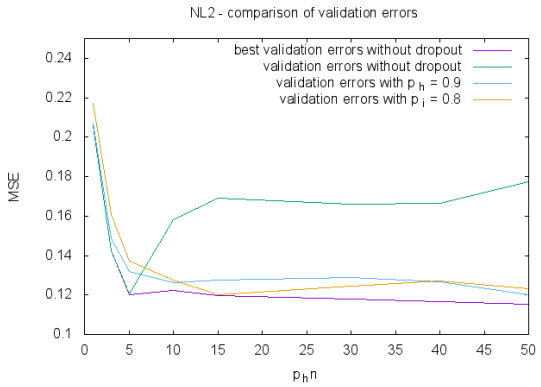
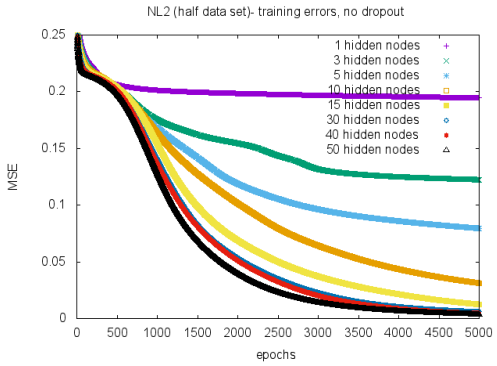


Figure 25: The purple line shows the best validation errors reached with no dropout applied to the networks. The other lines show the validation errors after 5000 epochs. The exact values are listed in table 5.

The validation errors after 5000 epochs with and without dropout were compared to the optimal validation errors reached without any dropout (table 5 and fig. 25). Again, no loss in validation performance is observed.

4.4 NL2 - half data set

Since the NL1 and NL2 data sets had so similar results, the added noise did not make as big a difference as expected. A likely reason for this is that the size of the data set was big enough for the networks to learn that the noise variables did not affect the output by lowering the absolute value of the weights corresponding to those inputs. Therefore, to increase the impact of the noise, and to increase the networks susceptibility to overtrain, the



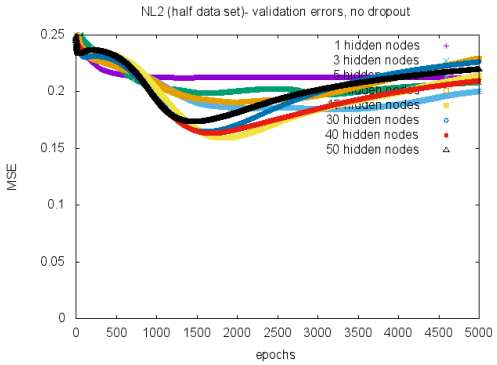


Figure 26: The training errors (left) and validation errors (right) when training on the NL2 data set without dropout.

NL2 data set was halved. The training and validation errors when training without dropout on the halved data set are shown in fig. 26, where a significant increase of overtraining can be seen.

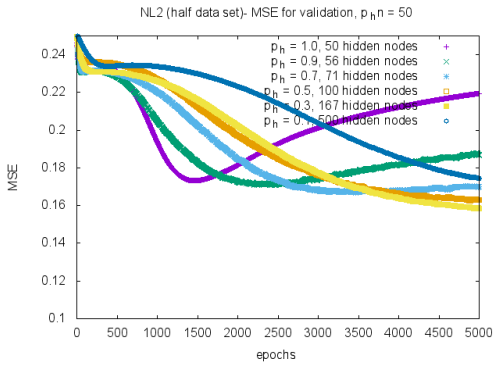


Figure 27: The validation errors for networks, trained on the halved NL2 data set, with varying dropout for the hidden layer, and no drop out on the input layer. $p_h n$ was held constant at $p_h n = 50$.

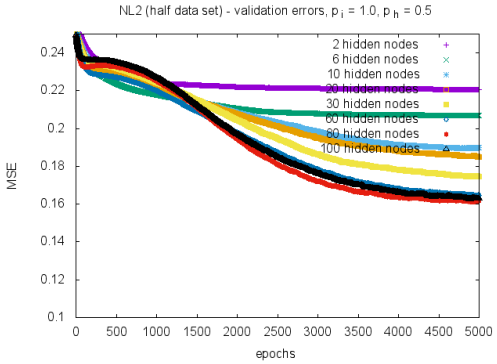


Figure 28: Validation errors when training on the halved NL2 data set with 50% dropout on the hidden layer.

A much higher drop rate was required for reducing the overtraining (fig. 27). Since less information is used for training the networks, the networks produce higher validation errors compared to when training on the whole data set. However, the dropout method still removes almost all of the overtraining with a 50% drop rate on the hidden layer (fig. fig. 28).

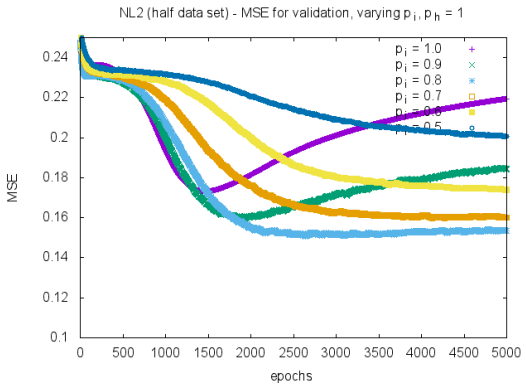


Figure 29: The validation errors for varying drop rates on the input layer. All networks used in this test had 50 hidden nodes.

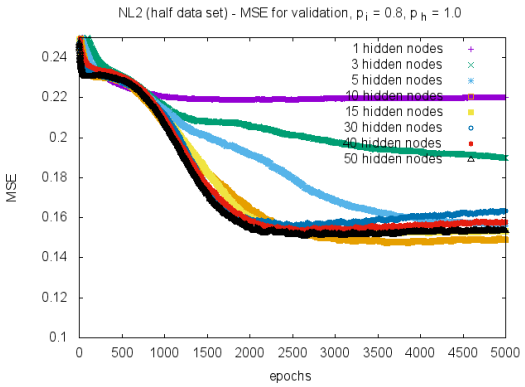


Figure 30: Validation errors when training on the halved NL2 data set with 20% dropout on the input layer.

As for the NL1 and NL2 data sets, 20% was the optimal value for the drop rate on the inputs (fig. 29). Some overtraining can still be observed. However, the validation errors are slightly lower compared to applying dropout on the hidden layer (fig. 30; compare to fig. 28).

Table 6: The second column lists the best validation errors reached for each network size with no dropout applied to the networks. The columns 3, 4, and 5 lists the validation errors after 5000 epochs.

$p_h n$	best validation errors without dropout	validation errors without dropout	validation errors with $p_h = 0.5$	validation errors with $p_i = 0.8$
1	0.21188	0.21188	0.22069	0.21999
3	0.19745	0.20835	0.20699	0.19009
5	0.18419	0.20040	0.18974	0.15707
10	0.19076	0.22945	0.18508	0.14905
15	0.22945	0.21416	0.17477	0.15330
30	0.16471	0.22634	0.16484	0.16334
40	0.16335	0.20964	0.16158	0.15780
50	0.17320	0.21948	0.16291	0.15337

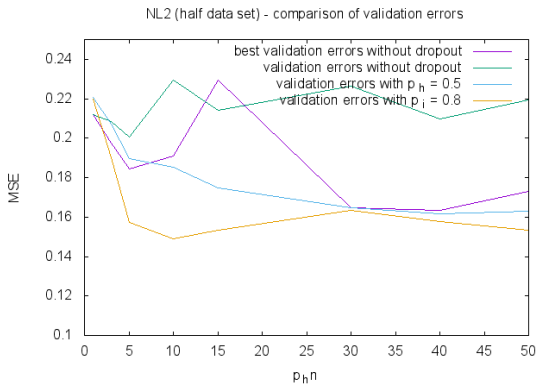


Figure 31: The purple line shows the best validation errors reached with no dropout applied to the networks. The other lines show the validation errors after 5000 epochs. The exact values are listed in table 6.

Table 6 and fig. 31 shows that even for this halved data set, the validation performance was not limited due to applying dropout. On the contrary, the validation performances of the smaller networks are improved by a considerable amount.

4.5 Pancreat

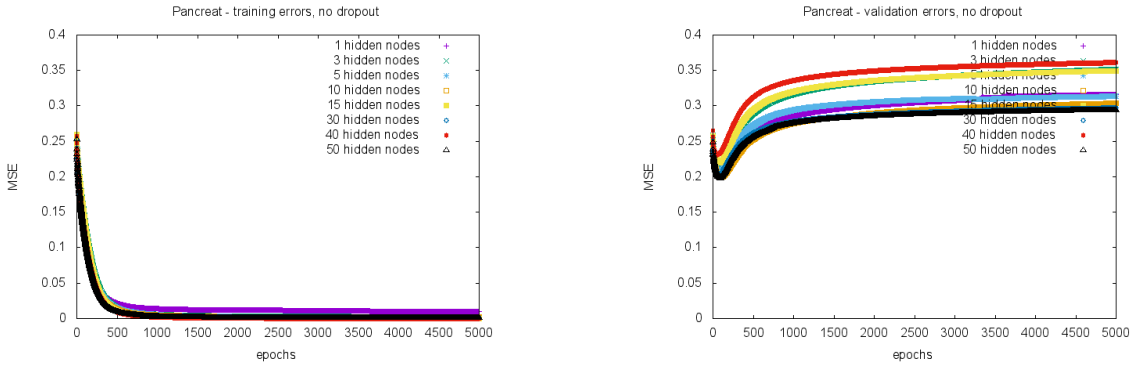


Figure 32: The training errors (left) and validation errors (right) when training on the Pancreat data set without dropout.

The huge number inputs make the networks very susceptible to overtraining when training on the Pancreat data set (fig. 32). The training errors are almost zero whereas all validation errors were above 0.25 after 5000 epochs.

For this data set, the best result without dropout on the inputs was achieved with a 70% drop rate on the hidden nodes, however, a lot of overtraining is still present (fig. 33 and fig. 34).

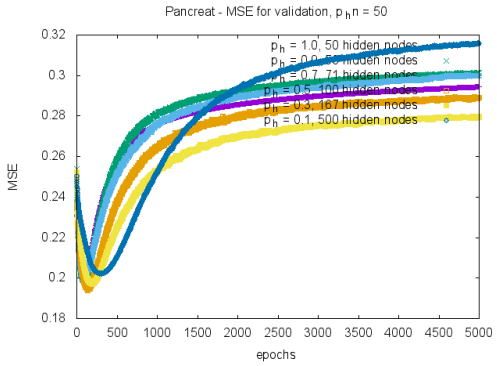


Figure 33: The validation errors for varying drop rates for the hidden layer, and no dropout on the input layer. p_hn was held constant at $p_hn = 50$.

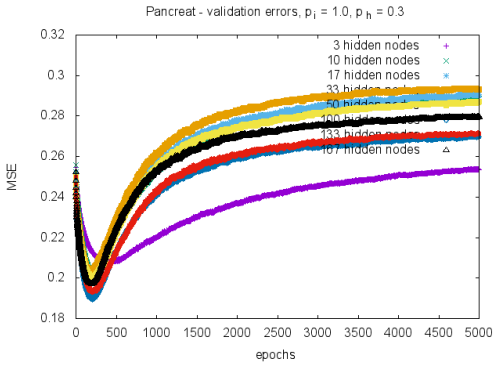
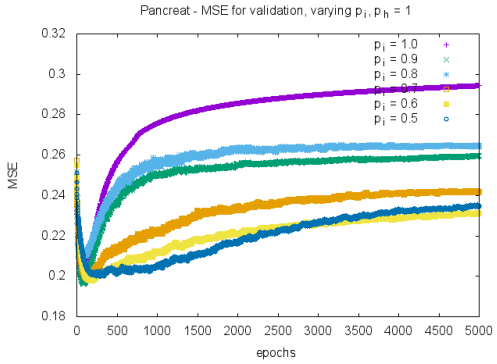
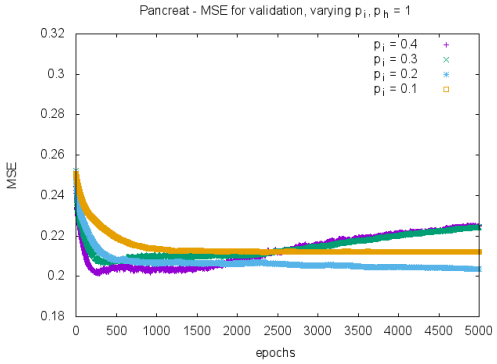


Figure 34: Validation errors when training on the Pancreat data set with 70% dropout on the hidden layer and no dropout on the input layer.



(a) p_i of the interval [0.5,1.0].



(b) p_i of the interval [0.1,0.4].

Figure 35: The validation errors for varying drop rates on the input layer. All networks used in this test had 50 hidden nodes.

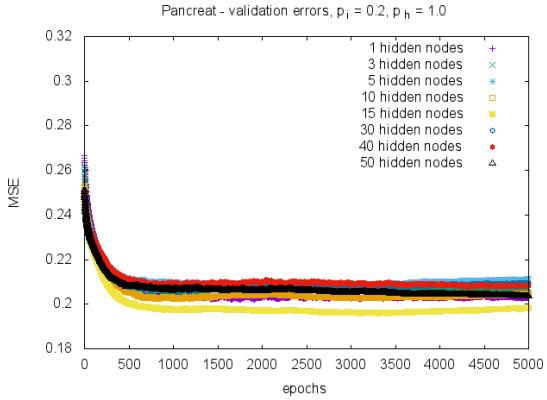


Figure 36: Validation errors when training on the Pancreat data set with 80% dropout on the input layer.

Dropout on the inputs showed significantly better results. The best drop rate for the inputs was 80% (fig. 35). Figure 36 shows that, with 80% dropout on the inputs, no sign of overtraining can be observed. Another thing to notice from this figure is that almost the same validation error is reached for all network sizes, even with only one hidden node, meaning that this was a linear problem.

Table 7: The second column lists the best validation errors reached for each network size with no dropout applied to the networks. The columns 3, 4, and 5 lists the validation errors after 5000 epochs.

$p_h n$	best validation errors without dropout	validation errors without dropout	validation errors with $p_h = 0.3$	validation errors with $p_i = 0.2$
1	0.20557	0.31634	0.25343	0.20293
3	0.21611	0.35149	0.28860	0.20698
5	0.20503	0.31352	0.29059	0.21121
10	0.19875	0.30367	0.29315	0.20480
15	0.22025	0.34972	0.28711	0.19823
30	0.20130	0.29701	0.26975	0.20937
40	0.23143	0.36113	0.27140	0.20818
50	0.19794	0.29436	0.27967	0.20365

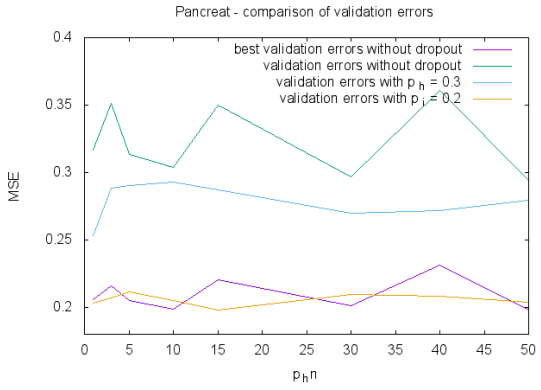


Figure 37: The purple line shows the best validation errors reached with no dropout applied to the networks. The other lines show the validation errors after 5000 epochs. The exact values are listed in table 7.

The validation errors after 5000 epochs with and without dropout were compared to the optimal validation errors reached without any dropout (table 7 and fig. 37). As also seen in fig. 34, dropout on the hidden layer reduced the overtraining, but was not enough to remove the overtraining. However, fig. 37 shows that with dropout on the input layer, the validation errors are as low as the optimal errors without any dropout.

4.6 Discussion and Conclusion

Dropout is a versatile method, capable of reducing overtraining on a variety of problems, with varying number of inputs, complexity and data set sizes, without limiting the performance of the network.

A downside with dropout networks is that they take a longer time to train compared to regular MLPs without any dropout. A higher drop rate on the hidden layer reduces the probability of overtraining but requires more time. Dropout on the input layer does not increase the time as much as dropout on the hidden layer, since the size of the networks stays constant, however, a more careful choice of the drop rate has to be made, since, with a higher drop rate, although overtraining is reduced, the performance of the network may be worse. This effect could be seen for the non-linear problems. A likely reason for this is that with a drop rate that is too high, the networks are not always capable of finding the non-linear relations, since they require several inputs to be present.

For the data sets used in this project, dropout on the inputs proved to be better than dropout on the hidden layer (with good values on the drop rates). By applying dropout on the input layer the validation errors were as good as or better than the validation errors produced by applying dropout to the hidden layer, whilst taking significantly less time to train.

No improvements could be made by applying dropout on both inputs and hidden nodes, since, with good values on the drop rates, the networks had almost no problem of overtraining either when only applying dropout on the hidden nodes (all data sets except Pancreat) or only on the inputs (all data sets). This also shows that no additional regularization is needed.

An improvement on the study could have been made by comparing the dropout method to ensembles. However, this was not done due to limited time.

References

[1] Warren S. McCulloch, Walter Pitts (1943). *A Logical Calculus of Ideas Immanent in Nervous Activity.*

[2] Neuron, `https://en.wikipedia.org/w/index.php?title=Neuron&oldid=721347018` (last visited May 21, 2016.)

[3] Mattias Ohlsson (2015). *Lecture Notes on Artificial Neural Networks (FYTN06).*

[4] An overview of gradient descent optimization algorithms (2016), `http://sebastianruder.com/optimizing-gradient-descent/index.html#gradientdescentvariants` (last visited January 2, 2017)

[5] Yann LeCun, Yoshua Bengio, Geoffrey Hinton (2015). *Deep learning, Nature, vol. 521, 28 May 2015, pp. 436-444.*

[6] B. Yegnanarayana (2006). *Artificial Neural Networks, p. 378.*

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.*

[8] `https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes` (last visited January 2, 2017.)