Network Analysis Individual Assignment – Mirthe Hendriks 6866999

Describe the principles of overfitting and how dropout can reduce this (Question 1, 5 points)

Data are observations from a real world process, but they represent only a sample of that reality. A common issue in training deep neural networks is overfitting. An overfitted model is inaccurate because the trend does not capture the real world process. The main issue is that, with the training of a neural network model, there is a trade-off between the models' ability to fit the training data accurately and do well on the (classification) task at hand, and the models generalizability to new data. Overfitting can be observed when the model predicts the data it trained on very accurately, but fails to generalize to unseen data. Specifically, an indication of this happening is if the accuracy and loss scores of the testing/validation dataset are noticeably worse than those of the training dataset. This is problematic as the goal of machine learning is to make accurate predictions on new data in future tasks.

Especially excessively complex models with too many parameters are prone to overfitting. A technique to prevent overfitting is regularization, which implements slight modifications to the learning algorithm for the model to generalize better. Dropout is one regularization technique. The technique entails the random "drop out" of a subset of neurons (i.e. nodes) in the (hidden) layer(s) of the network during training, in each iteration. The contribution of these neurons – along with their connections – will thus be ignored when running a prediction on the data. More specifically, the participation of these neurons in the feed forward activation of downstream neurons, and the weight modifications for neurons in preceding layers when propagating backwards, is temporarily removed. Without dropout, a neural network learns the optimal weights of neurons for features specifically for the training data. The neurons co-adapt too much; neighboring neurons become very reliant on this specialization and activation in one neuron is linked to a certain output. Dropout then ensures that the network becomes less sensitivity to particular neurons. It forces the network to use output of other neurons in surrounding layers to make predictions. Due to the fact that in each iteration a different random subset of neurons is "dropped out", the network is essentially trained on a different dataset each time. Thus, dropout reduces overfitting and allows the model to generalize better to unseen data.

In the Keras library for python, Dropout layers can simply be added in the model definition. The probability with which neuron are randomly dropped-out in each iteration can be specified.

Read the research paper "Performance-optimized hierarchical models predict neural responses in higher visual cortex" and write a short (~500 word) summary of the experimental approach and results. (Question 2, 10 points)

Yamins et al. (2014) introduce an experimental approach to improve predictive models on neural responses in higher ventral visual cortex, specifically the inferior temporal (IT) cortex. Humans are extremely skilled at visual object recognition, but our understanding of the ventral stream is limited. The purpose of this study was to train an artificial neural network model to equal human performance. The understanding is that if a model predicts neural responses in IT perfectly it by definition shows high performance, because IT shows high performance. This paper discovers that for a biologically plausible hierarchical neural network model, the opposite is equally valid: the more accurate the categorization performance of a convolutional neural network (CNN) model, the higher the percentage of IT neural explained variance. Yamins et al. (2014, p. 8623) operationalized a hypothesis to show that two

biological constraints define the ventral visual cortex: (1) “the functional constraint of recognition performance” and (2) “the structural constraint imposed by the hierarchical network architecture”.

The experimental approach entailed computing a wide range of candidate CNN models drawn from a large parameter space, and evaluating the models’ predictivity on object categorization tasks. More specifically, a standard linear regression method was applied to assess this predictivity: a match was formed between the target IT neuron and an artificial neuron composed from a linearly weighted model output, then for new image the classification by the artificial neuron is compared to the output of the human neural sites. Each model is defined by parameter choices (the number of layers, and parameter at each layer, activation threshold, pooling, and local receptive field size) which impact their recognition performance and neural predictivity.

For the selected models – either randomly selected, IT fitting optimized or categorization performance optimized - there was a significant correlation between performance and neural predictivity. The performance optimized models performed best. The results indicate that model performance optimization based on regions in parameter space is valuable in constructing IT-resembling models. The next step was to improve the model using hierarchical modular optimization (HMO), which efficiently discovers high-performing combinations of functionally specialized hierarchical architectures without a pre-specified subtask. With the HMO procedure a four-layer CNN with 1,250 top-level outputs was identified: named the HMO model.

Next, the HMO model was tested to examine whether its performance generalized to the testing image set. The authors measured its predictivity in individual IT neural sites and found that the higher layers predicted IT units increasingly well and outperformed control models. The layers demonstrate that category selectivity and tolerance to high variation tasks improves progressively along the hierarchy. The findings indicate that high categorization performance and a hierarchical model architecture together produce IT-resembling populations. Additionally, the paper found that the HMO model generalizes robustly to images of new objects and categories, and captures significant similarity between the predicted neural population and the IT population structure.

In addition, the intermediate layers of HMO were used to predict responses in V4, the dominant cortical input to IT (which is less categorical in its neural representation compared to IT). The results show that HMO’s penultimate layer is a good match for V4 with high predictivity of its neural responses. Given these findings, there is significant support for the hypothesis that V4 is effectively predicted by the intermediate layer in a hierarchical model in which the top layer resembles IT neural responses.

In conclusion, performance optimization not only shapes top-level output model layers that successfully mimic IT functional recognition performance, the hierarchical network architecture also introduces biologically consistent structural constraints on the intermediate layer’s feature representation to promote downstream performance. The process of biological performance optimization can be used as inspiration to construct hierarchical neural network models that successfully predict neural processing in the highest ventral cortical area.

Tenserflow playground: Play around with these settings and see how they affect your ability to learn classification of different data sets. Write down what you found and how you interpret the effects of these settings. (Question 3, 8 points)

Deep convolutional neural networks (CNN) are commonly used to perform classification tasks of varying complexities. The TensorFlow playground allows us to discover how model parameters influence the performance of the network; the different datasets, input features, number of layers, number of neurons in each layer (i.e. feature maps), ratio of training and test data, noise, the activation function, and regularization techniques will be explored.

The playground simulates a classification task where points in a dataset need to be classified as blue or orange. In the datasets the colored points are distributed differently along the x and y coordinates. Certain classification tasks will be more difficult due to the distribution of the colored points; the Gaussian should be relatively easy because a single decision boundary can be used, while a spiral is obviously more complex. Nevertheless, we see that all data sets can be classified accurately as long as the network applies a suitable (mathematical) transformation to the input and extracts the relevant features. Each piece of input information is known as a feature which can be represented in different formats. The circle data was accurately and efficiently classified when using X_1^2 and X_2^2 as inputs in the feature column, most likely because these transform the input in a way that suits the data at hand (potentially using distance from center to separate the colored points). Alternatively, an input X_1 value on the horizontal and X_2 on the vertical axis work well for the Gaussian data; this feature representation results in an accurate and quick classification. Similarly, the feature X_1X_2 seems to transform the input in a way that makes the classification task on the 'exclusive or' dataset easy. The great benefit of deep learning is, of course, that the machine leans how to transform the input; what important features to extract to accurately classify the data.

A neural network can solve a simple classification task by selecting the right set of features. However, for more complicated tasks, deep learning is needed to determine which features are relevant. A deep network might contain multiple hierarchical layers. In the playground we can select up to 6 hidden layers with 8 neurons per layer. Each neuron in a hidden layer represents a feature map, where a certain filter pattern of the input is seen. Each representation in a higher layer is built from a representation of the input in preceding layers. Especially in higher levels of network (top layers), the representations become increasingly abstract, because filter structures operate over multiple feature maps and we are looking at a pattern of a pattern of a pattern. We can observe this in the playground, because within the neuron squares we see a small representation of the input (a certain area in the input map, a line or a circle) which becomes more abstract in subsequent layers. The playground also confirms that, while for some more complicated tasks it is beneficial to use multiple neurons and multiple hidden layers, for other tasks this is not necessary. For classification tasks where the relevant features are unknown, multiple hidden layers might be beneficial. The playground network shows lines connecting the neurons, where the thickness of the line reflects the weight of the connection (the thicker the line the more influential the weight). These weights indicate how much a certain neuron contributes in the prediction task. From this we can deduce if the network can reach the same prediction using fewer neurons.

In the TensorFlow playground we can also adjust the learning rate. The learning rate is essentially the step size in the path towards minimized loss in a gradient decent. Exploring the playground I found that a higher learning rate produces the output loss faster, but from the theory I know it could miss the

minimum loss because of these larger steps. Most of the time, the runs I tried with a lower learning rate simply needed more epochs to achieve a lower loss output.

In the playground, the activation function can be specified. The purpose of the activation function is to only activate an output feature map if its value reaches a predetermined threshold, thereby introducing nonlinearity in the model. Rectified Linear Unit (ReLU) activation is a particularly popular. With ReLU the outputs equals the input for every value higher than zero. What I noticed in the playground is that for most models the ReLU function performed well, but that for some operations a different activation function might produce better results. When classifying the spiral data, the sigmoid activation function overall gave me better and more stable results.

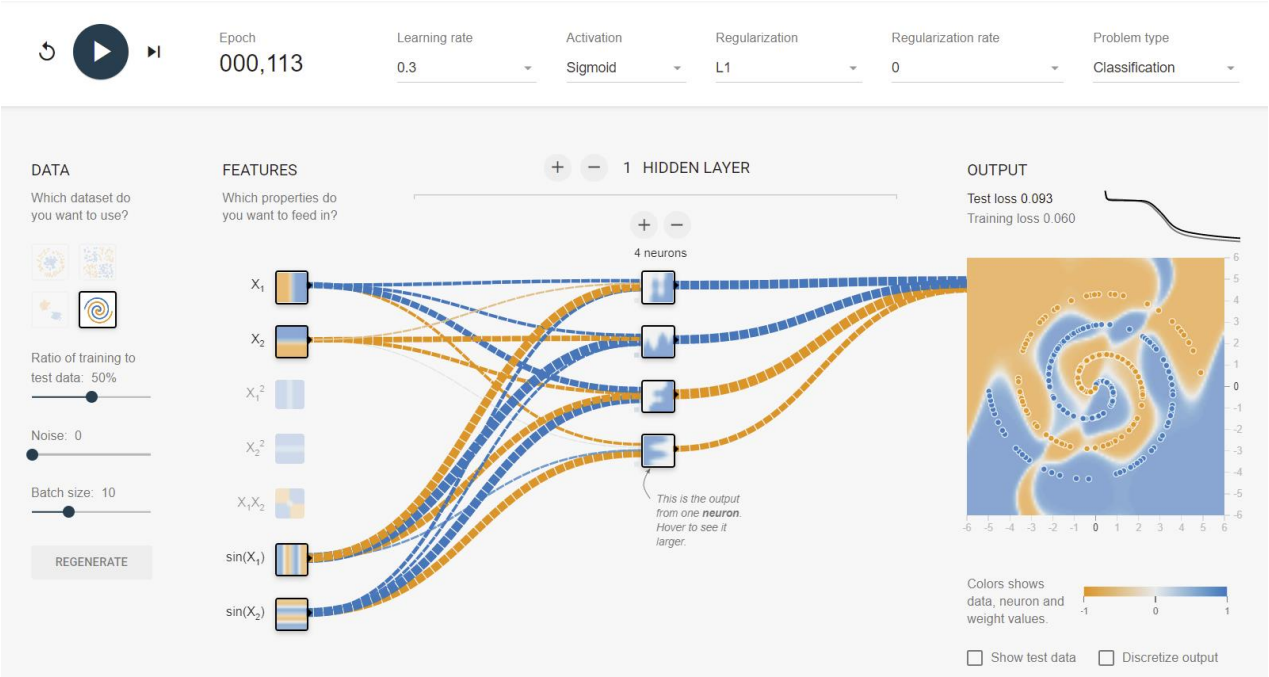
Furthermore, the playground enables regularization, a technique to prevent overfitting. Regularization implements slight modifications to the learning algorithm – slightly reducing or increasing the weights of weak or strong connections – for the network to generalize better to new data. In the playground, you can choose the L1 or L2 method. L1 works well for sparse feature spaces, when we need to select a few neurons among many. L2 is more effective when inputs are correlated. The regularization rate can also be specified, where a higher rate entail that weights are more limited in range. I tried both L1 and L2 in different models, and overall I noticed that the one technique was not consistently better than the other. I did notice that when applying regularization, the output loss score for the training and test data (i.e. the gray and the black line) were often closer together. This makes sense as regularization should prevent the network from overfitting and should allow for a more accurate generalization from the training data to the test data.

Lastly, in the playground we see ratio of training to test data and noise. Ratio of training to test data controls the percentage of data points in the training set and test set as the data must be divided between the two. 50% gives us an equal split. I noticed that in several test runs a lower loss score was reached with less epoch when there was more training data (70%). Nevertheless, in many instances, especially with simpler tasks, the ratio did not seem to matter that much. Furthermore, the playground allows you to add noise to the data. Noise makes the data pattern more irregular, so more difficult to classify accurately. The loss score on average were a little higher if there was more noise in the data, which is logical as we see orange points appearing in generally blue areas and vise versa.

Overall, my conclusion would be that there are many parameters that should be tuned and tested for a neural network to be the most accurate and computationally efficient as it can possibly be for a specific classification task.

<u>What is the minimum you need in the network to classify the spiral shape with a test set loss of below 0.1? (Question 4, 7 points)</u>

The minimum I used in the network to classify the spiral shape with a test set loss below 0.1: a network with the features X_1, X_2, $\sin(X_1)$ and $\sin(X_2)$, with 1 hidden layer containing 4 neurons, with a learning rate of 0.3, a sigmoid activation, and regularization L1. The network outputs a test loss score of 0.093 with 113 epochs (see below).



The spiral is of course a more complex dataset to classify. I started trying out different features to classify the input. I realized that X_1 and X_2 by themselves would not be able to transform the input in a way that would lead to an accurate outcome. I discovered that a combination of $\sin(X_1)$, $\sin(X_2)$ and X_1, X_2 worked very well. I figured I should try sin (with its wave pattern) because it might allow for a more complex transformation of the input. I tested $\sin(X_1)$ and $\sin(X_2)$ in combination with other features, but these resulted in higher loss scores.

First I selected multiple hidden layers for the network, however adding layers did not improve performance. I also tried two layers with varying numbers of neurons, but the output became unstable over the different epochs and overall loss scores remained relatively high. A single layer with only 4 neurons turned out to be sufficient; I realized no deep complex network was needed. I selected a learning rate of 0.3 to speed up the gradient decent algorithm and reach a lower loss score more quicky (but without missing the minimum loss). Keeping other network parameters stable, a learning rate of 0.1 output a low loss score but needed more epochs to get there. Furthermore, I tested the various activation functions. With ReLU the output became unstable with a lot of spikes in the loss scores among the epochs. This network with a sigmoid activation function consistently showed good performance. I decided that regularization would prevent overfitting and would improve the models' generalizability from the training data to test data. Regularization with L1 did not seem to differ significantly from L2 with regards to the network output.

This network is able to accurately classify the spiral and has high computational efficiency, needing only around 100 epochs to obtain a test loss score of below 0.1.

Explain the principle of backpropagation of error in plain English in about 500 words. This can be answered with minimal mathematical content and should be IN YOUR OWN WORDS. What is backpropagation trying to achieve, and how does it do so? (Question 5, 8 points)

Backpropagation is the essential algorithm for how neural networks learn. The algorithm uses supervised data to fine-tune filter parameters and minimize the error, in order for a neural network to learn to produce the desired output on a tasks. Before we get to backpropagation, let us take a step back to look at deep learning. Deep learning is commonly used in more complex classification and regression tasks, because a machine can learn to transform data input in a way that it will produce a desired output. Especially in the higher layers of a neural network, machine learning can figure out what abstract feature representation to use. The layers in the network consists of neurons, with neurons in lower layers activating neurons in higher layers. The filter structure is essentially the weight of connections between the neurons. The network preforms a feed forwards sweep and produces an output. Because we are using supervised learning, we know what the correct output is. If the predicted output does not equal the correct output, we can deduce a prediction error (i.e. loss) from the difference. The ultimate goal is for the neural network to perform well and make accurate predictions. Thus, network parameters can be updated to reduce the prediction error. In practice, weights and biases are often fine-tuned to minimize a cost function (i.e. the sum of squared errors) and improve model accuracy and generalizability. This is where backpropagation comes in.

The backpropagation algorithm computes the gradient (the derivative) of the cost function to minimize the loss. The weights and biases are updated by gradient decent. This entail that each individual weight and bias is adjusted with an small amount in the direction opposite to the error gradient. The backpropagation algorithms uses the chain rule to iterate backwards through the network; it begins at the top layer and propagates backwards down to the first layer, computing the gradient of the loss function for each layer sequentially. Basically, the weights of connections between neurons are modified depending on each neuron's activity and the amount of error. If a neuron's activity is high and the error is high, the weights need to be modified significantly. The activation of each neuron is inspected to see how erroneous the output was in order for the network to learn how much to change the structure. For every neuron, the error on the connection with neurons in the preceding layer is computed and the weight for those connections are updated. Thus, the derivatives of the error with regards to the filter structure of a layer can be efficiently deduced from the previous layer in the hierarchy. As such, through backpropagation, the network learns how sensitive the cost function is to each weight and bias and what change to apply to the filter structure to minimize the cost function. In short, backpropagation allows the deep neural network to learn.

Nowadays, large-scale backpropagation has become feasible because of stochastic gradient descent (SGD). SGD entails splitting up the training data into smaller, random batches. The gradient decent is computed sequentially for each batch where multiple iterations are used to make new batches. SGD improves the computational efficiency of backpropagation on very large datasets and generally converges to a stable prediction.