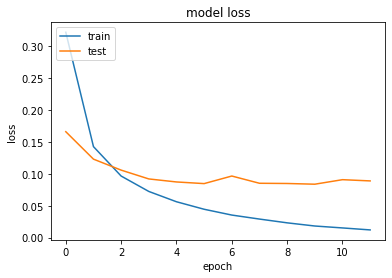
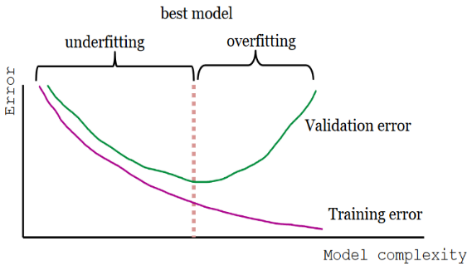
**- Question 1 -**

**(1) Underfitting.** When starting the training process, the loss value (or prediction error), will be high during the first few epochs and the model is thought to **underfit** the data. When underfitting, the model is too simplistic, hence is not performing well on training data nor on the unseen data (high training and validation prediction error). However, when both the training and the validation loss decrease, the model will gain on its predictive power.

**(2) Overfitting.** However, when continuing with the trainings process, at some point (the crossing in figure 1), the model is trained to strongly on the trainings set, it pays much attention to idiosyncrasies (or noise) that is uniquely inherent to every trainings set. Hence the model is **overfitted** to the trainings data set. In figure 2, when the validation loss in the plot stagnates, or even raises again, while the training loss remains constant or decreases, the model is overfitting. The larger the difference between validation and trainings error the strong the degree of overfitting. Therefore, to select an optimal model a balance between underfitting and overfitting must be maintained.

* **2.1 Implications of overfitting.** Overfitting will be compromising a model’s capacity to generalize its current ‘knowledge’ about the world states to other similar world states that were not previously encountered (e.g. unseen data set; or other noisy samples of the target population). This implies not only a reduction in performance to classify or predict future instances (or states) correctly (e.g. on the holdout test or validation set) but also affect the reliability of the model prediction and the validity of the interpretation of the models result.

**(3) Model constraints: Dropout.** To prevent overfitting, constrains can be introduced to reduce the ‘complexity’ of the model that is tailored to the trainings set. One possible constrain is to introduce another hyperparameter – the **dropout** (rate).

* **(3.1) Co-adaptions contributed to overfitting.** With respect to a model’s optimizer, backpropagation and gradient descent is improving the model’s performance throughout the training session. For each epoch, ‘gradients’ are computed from the loss in order to adjust the weights of each feature (their impact), so that the network performs slightly better during the next iteration of the training process. Computing the gradient is done with respect to the error, but also with respect to adjustment of the of weights of all other units (Srivastava et al., 2014). These changes in their weights, may fix the mistakes of other neurons, leading to complex **co-adaptations** that may not generalize to unseen data, hence contributed to overfitting.
* **(3.2) How dropout prevents coadoptions.** With dropout, the training process essentially prunes neurons in a neural network, including their connections or synapses, and hence no data flows through these neurons anymore. Dropout, then, prevents these co-adaptations by making the presence of other hidden neurons unreliable, because the pruning of neurons (and thus their connections) is performed at random: Neurons simply cannot rely on other units to correct their mistakes, which reduces the number of co-adaptations that do not generalize to unseen data. The units are temporarily removed from the network during every epoch, which generally results in sampling thinned networks (e.g. Srivastava, 2014). The last step is to sample the ‘thinned’ network from the global architecture and used it for training.
* **(3.3) Conclusion.** Concluding, the reduction of co-adoption during the dropout process leads to significantly lower generalization error rates (i.e., overfitting), as the presence of neurons is made unreliable, improving the generalizability, reducing overfitting and hence validity of the interpretation of model output with respect to the target population.

**- Question 2 –**

**(1) Method 1:** First, 168 biological IT neural responses are extracted by a multi electrode array while the animal (rhesus macaque) was exposed to a benchmark test set, containing photorealistic 3D objects of 8 natural categories that are placed on randomly selected cluttered natural scenes; variation of object position, scale, and pose were used to ensures viewpoint invariant object recognition. Second, a myriad of candidate DCNN models, each is unique within a large parameter space (*N* = 57), are trained on the same images. A total of 5935 models were selected for evaluation that were either based on random draws from the whole parameter space (*N* = 2016), explicit IT predictivity-optimization (*N* = 1876) and object categorization performance-optimization (*N* = 2043). Third, the models were evaluated on object categorization accuracy, and IT neural predictivity; the latter by testing the response predictions of ‘the best biological-matching’ synthetic neuron in the network against actual neural site’s output on novel images.

* **Result I:** The results demonstrate that invariant object recognition performance does strongly correlates with IT neural predictivity (produce more similar IT neural responses) and this was true across all 3 selection regimes (or model classes). However, the models that were optimized for performance had the highest correlation with IT predictivity from all 3 model classes, with the best model within this class did predict neural output as well as those models directly selected for neural predictivity, although the reverse is not true. It was concluded that performance optimization is an efficient mean to identify regions in parameter space containing IT-like models.

**(2)** **Method 2:** In a follow up, the model class was extended to deeper DCNN networks that, intuitively, corresponded more closely to the visual stream architecture, by using hierarchical modular optimization (HMO). In an initial pre training, the HMO-optimized models were then tested on a screening task and the model with the highest generalization performance to other naturalistic images (e.g. present in the test set) was selected. After completing the training set, ‘the HMO model’ was tested on three different test sets, consisting an increasing difficulty of orientation, size and position (increasing the degree of view-point invariance) and compared against multiple contender: **(1)** a variety of published state-of-the-art ventral stream models targeting several levels of the ventral hierarchy (low level (V1), intermediate (V2 and V4) and high level (IT) models) that were trained on the same training set as the HMO model, **(2)** two neural reference models that were, unlike the other models, trained directly on the responses of the IT and V4 neuronal population of the animal and **(3)** sampled performance of humans.

* **(2.1) Result II:** Across all levels of variation, the IT population model was similar to human performance levels, while the V4 population and other popular low-level as well as high-level tailored models (e.g. HMAX) matched human performance only at low levels of viewpoint variation. For instance, HMAX model performed less well than the V4 population sample, even when pretrained directly on the test dataset. The HMO model’s object recognition performance, however, was similar to the IT neural sample, even for high degree of viewpoint variation, its high predictivity for individual IT neural sites (roughly an 100% improvement over the other models) and comparable prediction accuracy at lower lever demonstrate superior performance of the ‘HMO model’.
* **(2.2) Result III:** When measuring the HMO model’s IT predictivity across all layers, showed that each successive layer predicted IT units increasingly well, which verifies that the trend of a positive correlation between classification performance and IT predictability continues to hold in higher performance regimes. In addition, the model layers showed that category selectivity and tolerance to more drastic image transformations emerge gradually along the hierarchy. In addition, the performance of IT predictivity was also significantly higher than the ideal semantic ideal observer (knowing all category labels), showing that high IT predictivity does not automatically follow from category selectivity and that there must be significant noncategorical structure in IT responses attributable to intrinsic aspects of hierarchical network structure. Both, high categorization performance and the hierarchical model architecture class are necessary to produce IT like populations.
* **(2.3) Result IV** When using a representation dissimilarity matrix (RDM) to investigating the global IT neural representation at the population level, the neural population predicted by the output layer of the HMO model demonstrates a very high similarity to the actual IT population structure, that is close to the split-half noise ceiling of the IT population. Both, the RDM for HMO and the measured IT neural population exhibit a clear block-diagonal structure, which is associated with high categorization performance. This implies that much of the residual variance unexplained at the single-site level may not be relevant for object recognition in the IT population level code. Furthermore, after performing an category-level and object-level generalization test, the HMO model prediction were found to generalizes robustly, capturing the IT population’s layout for completely novel objects and categories.
* **(2.4) Result V:** When comparing certain layers of the HMO model’s neural predictivity for the V4 neural population with other HMO layers, the HMO model’s intermediate layer demonstrate a higher predictive capacity of V4 neural responses (51:7± 2:3% explained V4 variance), than the most top or bottom layer. This supports the claim that V4 corresponds to an intermediate layer in a hierarchical model whose top layer is an effective model of IT. When compared to the performance of other models, V2-like model predicts the most V4 variation 34:1 ±2:4%, while semantic models explain effectively no variance in V4, which is consistent with V4’s lack of category selectivity. Therefore, performance optimization does not only drive top-level output model layers to resemble IT, but also imposes biologically consistent constraints on the intermediate feature representations that can support downstream performance.

**- Question 3 -**

**(1) Data set.** A more complex data sets need mores epochs and demands a more complex model ( more or better features, more neurons and hidden layers). The interpretation might be that a complex problem as input (e.g. spiral shape compared to two clusters of data’s), might have more complex patterns the model needs to be able to capture, resulting in more complex decision boundaries (sine-function transformation of the input instead of linear transformations). In essence a complex problem can be solved when the **right** transformations (by the filters) are performed over and over again, assuming enough (complex) filters and a certain magnitude of filter iterations.

**(2) Feature selection.** The (degree of complexity of) problem to be analyzed is closely connected to the selection of the **right** filters. Adequate feature selection has crucial implications for achieving good performance measure (minimal loss, amount of overfitting and number of epochs) or even achieving an acceptable classification at all.

* For instance, X12 and X22 is best used for the first data set, while X1X2 is best used for the second data set. For the latter a test loss of 0.02 and a training loss of 0.013 is reached already at the 8th epoch (compared to 0.463 and 0.466 when using X1 and X2).

Again, the right transformations must be made, such that adequate decision boundaries can be drawn for classifications solution (e.g. transforming from a cartesian coordinate system to a polar coordinate system).

**(3)** **Neurons and Layers.** As aforementioned, more nodes (e.g. 8 instead of 4 in the first hidden layer) and layers (e.g. 6), do not necessarily increase the networks performance when the optimal features are selected (e.g. when results are compared at a fixed number of epochs). But more neurons, layers and neurons per layer can improve performance when the features are not optimal, which, however, comes at the cost of constructing a very large network.

* For example, a train/validation loss of < 0.1 for the spiral without sine-feature can be achieved with 6 hidden layers and 33 nodes.

A larger network consists of more nested features that can be iteratively recombined allows for more transformation steps using less optimal features from which a more diverse combination of the transformation operation can be achieved that might arrive at complex features, which are necessary to capture the relevant patterns in the input image that are necessary for good classification. This might be only true for relatively simple inputs (e.g. somewhat more complex shapes) but not for complex ones (e.g. objects); some features might never be derived from very complex combinations of simpler features 9e.g. nonlinear features from linear ones) or would demand a network size that is computationally unbearable. In addition, similar or even better performance can be achieved with less layers (e.g. 3 - 4 instead of 6) when the right features are chosen. Here, feature combinations do not have to compensate to make the more appropriate transformation, because the necessary transformation was already made by the appropriate filter in previous layers. Concluding, if the features are not correctly selected adding additional hidden layer can help, to some extent, to classify the output correctly and with less epochs.

**(4) Activation function.** The type of activation threshold does determine what type of input can be processed along with different theoretical and computational properties affecting the performance measures of the model.

* For example, when using linear activation, nonlinear patters like the spiral can never be classified accordingly. A combination of linear functions will be a linear function, which is unable to describe non-linear dependencies.

**(5) Regularization.** Regularization is useful to reduce overfitting in the model and depends on the regularization hyperparameter λ (here the regularization rate); λ is the tuning parameter that controls the bias-variance trade-off; if λ = 0, the penalty term becomes 0 and no regularization will be applied, with λ increases to infinite, the impact of the shrinkage penalty grows and the ridge regression (L2) and the lasso regression (L1) coefficients will get closer to zero.

* For example, selecting L2 with a λ = 0.003, overfitting of the model will be less than when no regularization is applied. However, with a λ = 10, the model is unable to learn from epoch one (because the weights of all features are close to 0 - are considered to be unimportant).

Importantly, the penalty term shrinks the weights differently for Lasso and ridge regularization. Lasso regularization reduces large and small weights equally, while ridge regularization reduces larger weights more than smaller weights. Therefore, some features (the ones with smaller weights) in the weight vector reach 0 before others: these are "pruned", and stop contributing to the problem, that is the classification of the target variable (blue and yellow dots; e.g. the 4-square feature would not be useful at all if the input feature space consists only sine and circular features). However only Lasso but not ridge regularization enables this type of feature selection during the trainings process, because the shrinking magnitude is equal for all weights in Lasso regularization, but is proportional to the magnitude of the weights in ridge regularization (larger weights lose more, smaller ones lose less). Therefore, the "force" pushing small weights to 0 is very small for ridge regularization and pruning of node only performed simultaneously (when λ is very high), essentially pruning all features at once).

* For example, selecting L1 with a λ = 0.003, will not only reduce overfitting of the model but also prune some nodes in the network. However, this assumes a sufficiently large network, otherwise the model might not learn well, because to many nodes are pruned before they can demonstrate to be useful.

**(6.1) Training-Test ratio. (1)** The ratio of training to test size has implication for model’s trainings process, and hence for the performance measures. When setting the train:test ratio very high (90:10), the trainings loss seems to be more robust (achieving the same accuracy reliably at a fixed number of epochs), while the validation loss and validation-trainings loss difference (overfitting) had mixed results. This, indication of overfitting, might, however, might be problematic since the model is only validated on a very small test set (hence a smaller feature space), such that generalization performance is better or worse due to chance of the sampling error in the validation set. **(2)** When the ratio train:test is low (30:70), performance in terms of minimal loss, quality of classification and number of necessary epochs suffer and due to the bad overall generalization performance the model tend to overfit more, which is even further increased when the data contains noise. **(3)** However, when the test set is too small (10:90) the model cannot learn enough, if at all, and many epochs are necessary; here, the model might never achieve good generalization, since only very little ‘knowledge’ (datapoints) of an actually way larger feature space is provided for the model. This was especially true when a more complex data set (4) was used. A 70:30 or an 80:20 ratio seem to yield the best performance measures (e.g. number of epochs). Ultimately, the ratio between test and trainings set depends on the overall size of the data set, which might be the most crucial factor for how well the training and validation data sets describe the feature space. If the number of data points in the whole data set is large then any division may work fine but when the data set is limited, division ratio may play a crucial role, with the ratio of 70:30 yielding the best results.

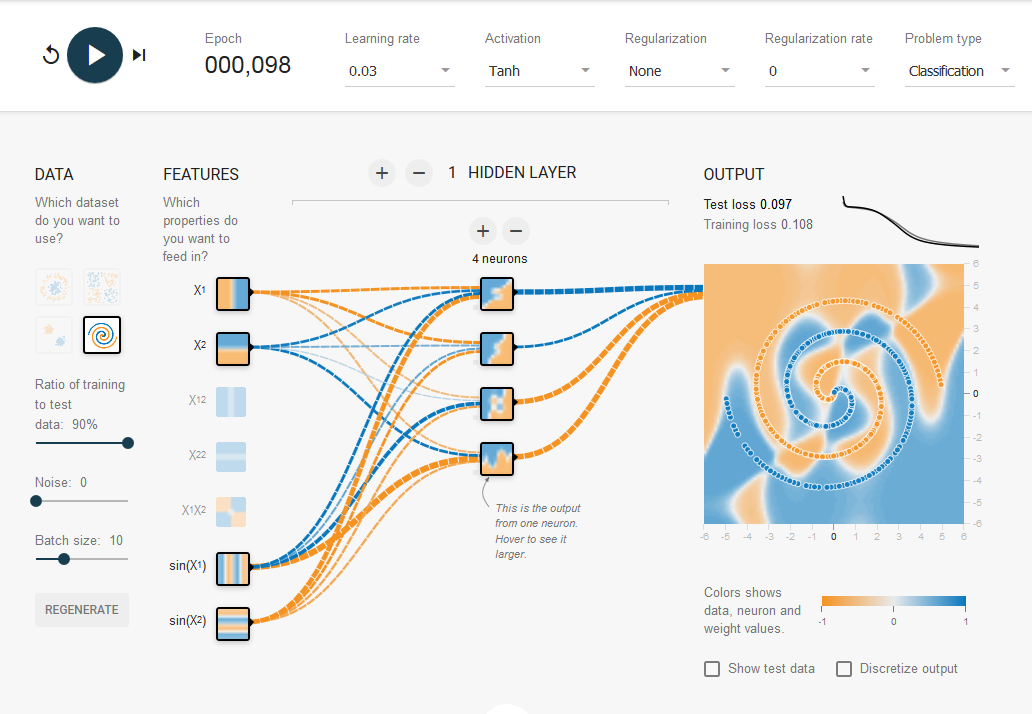
**(6.2) Noise.** Increasing noise in the data will reduce the generalization of the model and hence increase the overfitting of the model. This is a logical consequence in all measurement instruments or models that provide input for later decision making.

* For example, in signal detection theory, if the noise and signal data set overlap largely, the (measurement) sensitivity (or d-prime) is low, the noise and signal distribution strongly overlap, hence the ratio of false negative and positive will increase. Therefore, if less noise is contained in the data (noise and signal distribution overlap less), the neuronal network it can achieve better classifications independent of whether the right transformations are made, given that the noise follows a normal distribution and is independent (0-correlation) from the patterns in the data (or the signal).

**(6.3) Batch size. (1)** Smaller batch size (e.g. 2-4) can reach a low trainings and test loss with less epochs. However, this seem to be unreliable and is based on the chance that the model accidentally performs well given ‘the right’ batch sample (training examples) for one iteration. Only if by chance (sampling chance) the right examples are presented in the batch, the model achieves good classification. A robust result is achieved by using a higher batch size. **(2)** When the batch size is unnecessarily high (20-30), the model might suffer in terms of run-time efficiency. The model could have learned the same features with less data, since learning them multiple times is redundant; due to sampling of the batches, if the batch size is large I would expect the features to be represented in multiple batch samples.

**- Question 4 –**

Since there is no definition of a minimum provided in the question, to answer this question the following definition of minimal is assumed: Minimal describes a model with minimal network structure that is able to classify the non-noisy spiral data set reliably, with a test performance of <0.1, while taking memory and run-time efficient into account. If no noise exists in the data four features (1,2,6,7.) and one hidden layer with four neurons have to be selected to reliably achieve a test loss lower than 0.1. Importantly, the training to test data ratio should be at least 80:20 to achieve a fast result (fewer epochs). A similar result can be obtained unreliably when only feature 1,2 and 7 is selected or with a neuron less in the first hidden layer. This, however, comes at the cost of run-time efficiency (>800 epochs). Therefore, the first model is considered to be superior. Although, Tanh which is computationally more costly (run time within the epochs), instead of ReLu is used as activation function, a clear advantage for Tanh activation is observed for the amount of epochs. Although a smaller batch size does sometimes result in an extremely fast classification (<100 epochs), this is at cost of its robustness (model might not always attain a test loss of below 0.1). To avoid this, a batch size of at least 10 is selected. Regularization does not improve the run-time in terms of the number of epochs nor does it increase the test set performance (but reduces overfitting, which is no objective for this problem), therefore, no regularization is selected to avoid additional computational steps, ensure memory efficiency and keep the network structure minimal.



**- Question 5 & 6 –**

Backward-propagation is used to improve prediction accuracy of the forward method by utilizing gradient descent to optimize the weights such that the prediction error of the all weight is minimal.

**(1)** First the difference between the prediction and the real state (or label) must be quantified to provide an indicator of the prediction quality, which is the deviation of the prediction made by forward method from the true value **(y – ŷ)**. This is done by a cost function **J** that computes the overall cost by adding squared prediction error of W2 (the output of the previous layer) together, which can be summarized as:

The backward method of the model training is in essence about minimizing the costs function. Since the weights and features (filters) are often initialized are random, the costs of the top layer result from suboptimal weighing of the input or inappropriate features. Both can be changed to minimize the costs.

Since linear and nonlinear transformation of the data can be formulated in one equation, the gradient of a cost function can be computed by computing the 1-order partial derivative of **J** with respect to **W(N) (dJ/dW)** **–** therate of change in the **J** (thecosts) with respect to **W(N)** (all the weights). If the partial derivative of a given weight is a positive gradient (a positive rate of change of a given weight **w** in **W(N))**, then the optimal value is smaller than the current value of the weights and vice versa. The search for the minimum of a given weight function **w** stops if the cost minimal for that weight (where the partial derivative of **w** is 0). This greedy approach is called gradient decent.

The advantage of gradient descent is that this approach eliminates exactly one half of all possible values by knowing the direction of the gradient of **w** (+/-) and the computational time is decreased by a stepwise iterative search in the direction of the expected value (below or above the current value). Backward propagation is necessary, since a brute force search is computationally costly at its best and impossible to perform when the network structure is complex, containing many neurons (high dimensionality). For example, to finding the optimal value in a large set of values for a given weight in order to adjusting it, grows exponentially with an increasing number of weights.

The partial derivative for a set of weights **W(N)** is computed per layer (**W(1) W(2) W(3)** … **W(N)**), hence the weights can be represented separately. For example, when considering the gradient of the loss function **(dJ/dW)** for **W(N)**, with **N** = 2, hence the weights can be represented separately for the two weight matrices **W(1)** and **W(2)**. For **W(2)**, this can besummarized by:



To solve the equation, first the partial derivative is taken for each error (that is **(y – ŷ)2**) individually using the sum rule of partial derivatives. Then for each error term, the power rule and the chain rule is applied, which eliminates the exponent (by multiplying it with ½, used for mathematical convienience) then multiplying the partial derivative of the outer expression with the partial derivative of the inner expression. Since y is a constant (does not change with respect to **w**) the product of y becomes 0, leaving only **dŷ/dW(2)** \* **(y – ŷ)**. Next the derivative of **dŷ/dW(2)** is broken down into **dŷ/dz(3) \* dz(3)/dW(2)** using the chain rule, since **ŷ** is the activation function **z(3)** (e.g. sigmoid, ReLU or Tanh) applied to the values of the previous layer. Each activation function describes a specific function. For instance, ReLU is just **f(x)=max(0,x)**, while sigmoid is **f(x) = 1/(1 + e-x).** Again the partial derivative of the sigmoid function(**dŷ/dz(3)**)must be differentiated with respect to **z**. The last partial derivative is **dz(3)/dW(2),** of which **z(3)** is the matrix product of activities in the nodes of the previous layer (**a2**) and the weights, **w2**. For each synapse **dz(3)/dW(2)** is the activation **a2** on that synapse.

All these computational steps performed during the feedforward process need to be considered in error backward propagation. When considering the weights of one give layer, the backpropagation error is the difference between the predicted value and the true value multiplied by the differentiated activation function of the same size. Then the activity in every synapse (**a**) needs to be multiplied by the backpropagation of the error to adjust the weights (done by transposed matrix multiplication) in order to retrieve the adjustment of the weights for every neuron in this layer. This amount activity and the weighting of the activity differs for each neuron of a given layer. However, every neuron, contains a certain degree activation due to the input and each input example has a certain cost and a certain gradient that contribute to the total gradient decent in a weighted manner.

This mathematical operation results in the backpropagation of the error of the most top layer (the output layer) to each weight of the previous layer. The weights (multiplied with the activity in each synapse) that contribute the most to the total error at the output level of the prediction, therefore have larger activations, yield larger derivative values and will therefore change more than less contributing weights. Here the derivative is calculated with respect to the weight of the synapse. When the summed derivative for a given layer is computed the same process is repeated for the layer that feeds in the current layer. However, the derivative must also be computed across weights of the synapses. The linear relationship along each synapse is the rate of change in the highest layer with respect to each weight. Ultimately, each gradient is then subtracted from the weight of the neurons.

In a nonconvex loss function, however, the gradient decent might chose a value in a local minimum over the value in the global minimum, which would reduce the cost for the model the most and allow the prediction to approximate the true most for that given weight. To avoid this the cost function is expressed as the sum of squared errors, which will always be a parabola (e.g. x2), even in multidimensional space. In addition, because of stochastic gradient descent, performing the calculation one at a time (to avoid multidimensionality), the convex nature of the cost function might not matter that much.

**Reference**

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014, June 15). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Retrieved from <http://jmlr.org/papers/v15/srivastava14a.html>