A Deep Dive into Deep Learning Hyperparameter Tuning

...

Evan Johnson

Late Summer 2021

Overview

The purpose of this project was to get familiar with the process of *hyperparameter tuning*. We use TensorFlow and Keras to implement and train our neural networks.

Hyperparameter tuning is an important part of any machine learning project because it allows optimization of a chosen algorithm or neural network beyond the standard parameters

In this project, we choose a neural network approach and attempt to find an optimal set or sets of parameters to solve a classification problem with a simple dataset

https://github.com/evan-wes/A-Deep-Dive-into-Deep-Learning-Hyperparameter-Tuning

The Forest Cover Type dataset

The dataset consists of a mix of categorical and numerical features:

- `Elevation` - Elevation in meters
- `Aspect` - Aspect in degrees azimuth
- `Slope` - Slope in degrees
- `Horizontal_Distance_To_Hydrology` - Horz Dist to nearest surface water features
- `Vertical_Distance_To_Hydrology` - Vert Dist to nearest surface water features
- `Horizontal_Distance_To_Roadways` - Horz Dist to nearest roadway
- `Hillshade_9am` - (0 to 255 index) - Hillshade index at 9am, summer solstice
- `Hillshade_Noon` - (0 to 255 index) - Hillshade index at noon, summer solstice
- `Hillshade_3pm` - (0 to 255 index) - Hillshade index at 3pm, summer solstice
- `Horizontal_Distance_To_Fire_Points` - Horz Dist to nearest wildfire ignition points
- `Wilderness_Area` - (4 binary columns, 0 = absence or 1 = presence) - Wilderness area designation
- `Soil_Type` - (40 binary columns, 0 = absence or 1 = presence) - Soil Type designation
- `Cover_Type` - (7 types, integers 1 to 7) - Forest Cover Type designation

The Forest Cover Type dataset

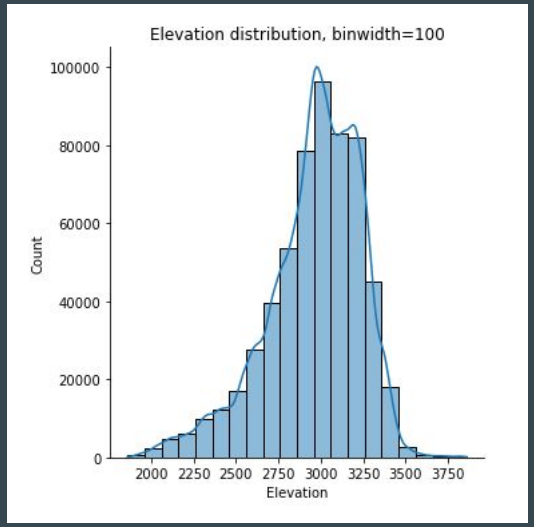
The target or response variable is the forest Cover Type:

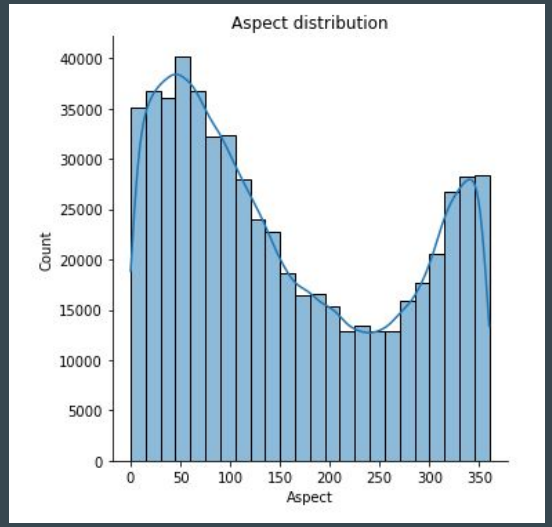
The `Cover_Type` is the response variable representing the type of forest cover. The seven values represent

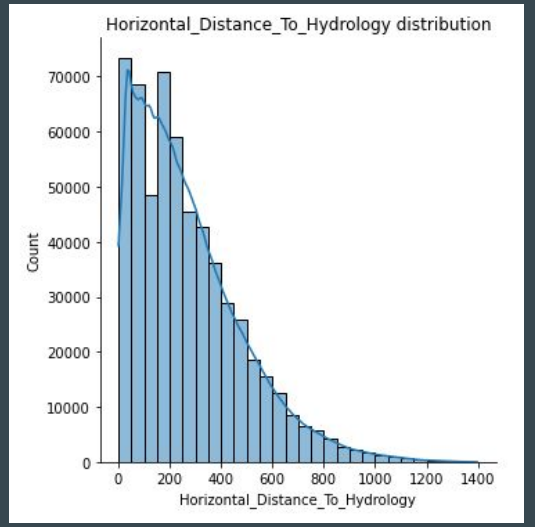
- `1` - Spruce/Fir
- `2` - Lodgepole Pine
- `3` - Ponderosa Pine
- `4` - Cottonwood/Willow
- `5` - Aspen
- `6` - Douglas-fir
- `7` - Krummholz

Exploratory Data Analysis

We can create histograms of the numerical features to get a sense of their distributions. We find symmetric, multi-modal, and asymmetric features:

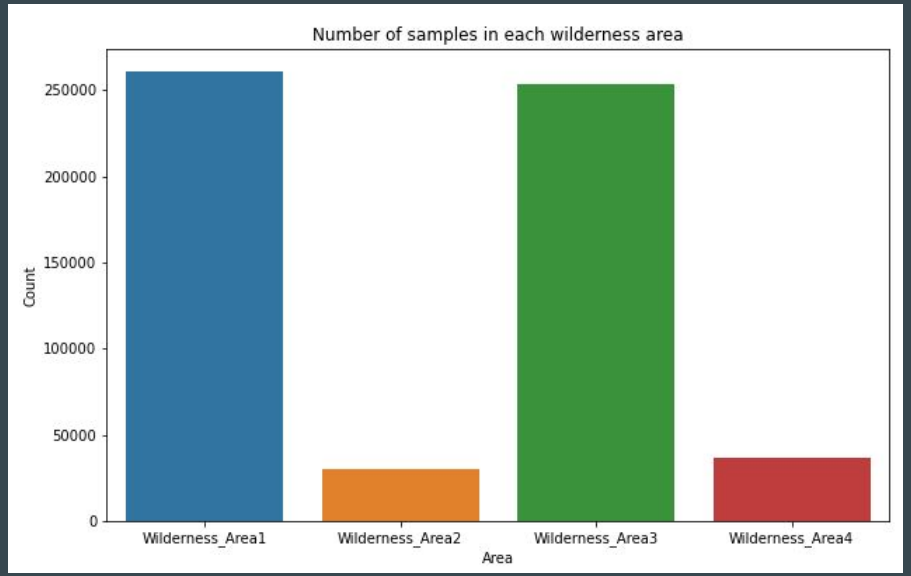


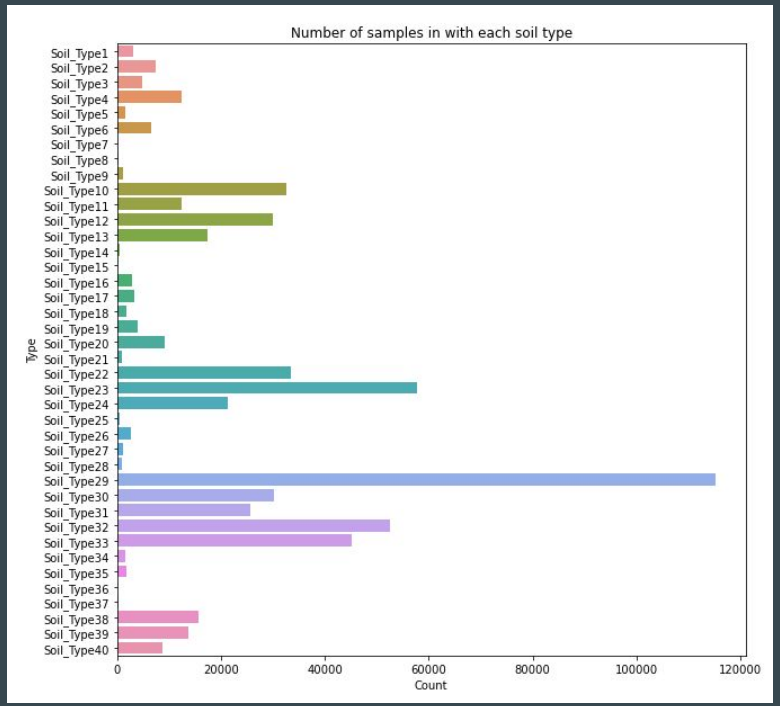




Exploratory Data Analysis

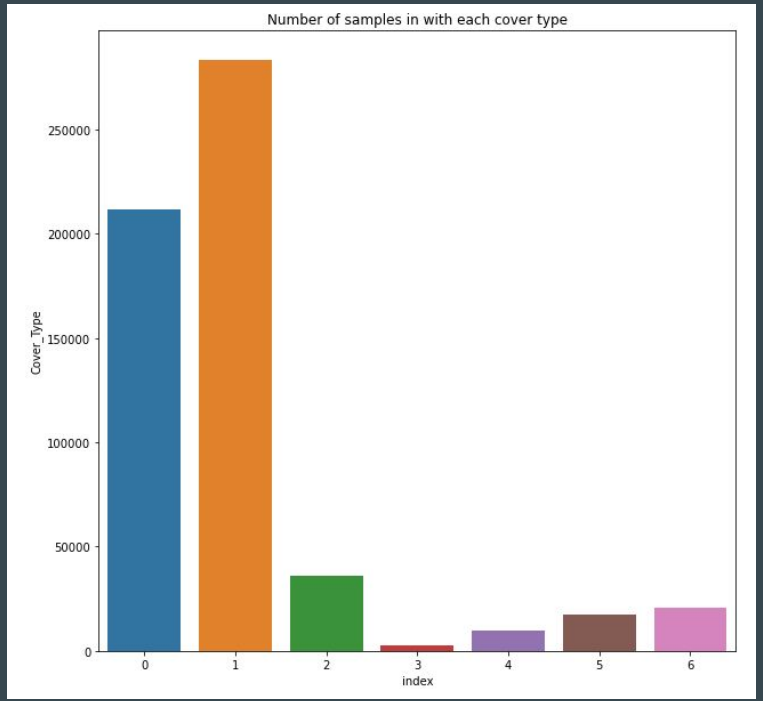
We can create histograms of the categorical features to get a sense of their distributions





The Target Variable

In this dataset, the target variable is the forest Cover Type. It turns out to not have a uniform distribution:



We can use *stratified sampling* when we split the data into training, validation, and testing data to preserve the distribution and ensure all values are represented

Feature Engineering/Data Preprocessing

The various numerical features have different ranges. We can use *min-max* normalization to bring them into a standard range from 0 to 1.

The categorical features can be *one-hot-encoded* to produce numerical values corresponding to each category.

Overall, are nearly 600k samples to work with, and 54 features.

Project objective:
Find optimal sets of hyperparameters to maximize classification accuracy

Types of parameters

- **Model architecture** - The number of hidden layers and their respective sizes and dropout rates
- **Optimizer** - The algorithm for updating the model weights after each pass through the network. We will test out stochastic gradient descent (SGD) and Adam (adaptive moment estimation) optimizers
- **Epochs** - Number of full passes of our data through the network during training
- **Batch Size** - how many training samples are passed each time through the network before the optimizer is updated
- **Splitting proportion** - Parameters that determine the initial training/testing split, and also the validation size during training
- **Dropout** - A regularization technique meant to limit or reduce overfitting. Controllable parameters are the probability for nodes to 'drop out', i.e. have their influence reduced to zero during training. We can apply dropout on the input layer, which we refer to as 'input dropout', and on hidden layers, which we refer to as 'hidden dropout'. Another parameter is whether or not to implement a max normalization constraint on the layers that have dropout applied. This can prevent weights from overcompensating for the dropout by becoming too large. The constraint value is another parameter.

- **Model intrinsic parameters**: Parameters that define how a model is built and compiled. They include the number and size of hidden layers, the properties of dropout layers, and the properties of the optimizer. Each variation of these parameters defines a different model, and changing them requires rebuilding and recompiling the model.
- **Model extrinsic parameters**: Parameters that are independent of the model itself. They include parameters that affect the training of the model like batch size, epoch number, and validation split, as well as the proportions of the initial training and testing splitting

Some paramters, like the model architecure, or batch size, need to be changed for each training run. Others, like the number of epochs, can be tuned in a single training run by looking at the training history.

Types of Parameters

Network Parameters

- Model Architecture (number of size of hidden layers)
- Hidden layer activation function
- Dropout parameters

Optimizer Parameters

- Optimizer algorithm
- Learning Rate

Training Parameters

- Batch Size
- Number of Epochs

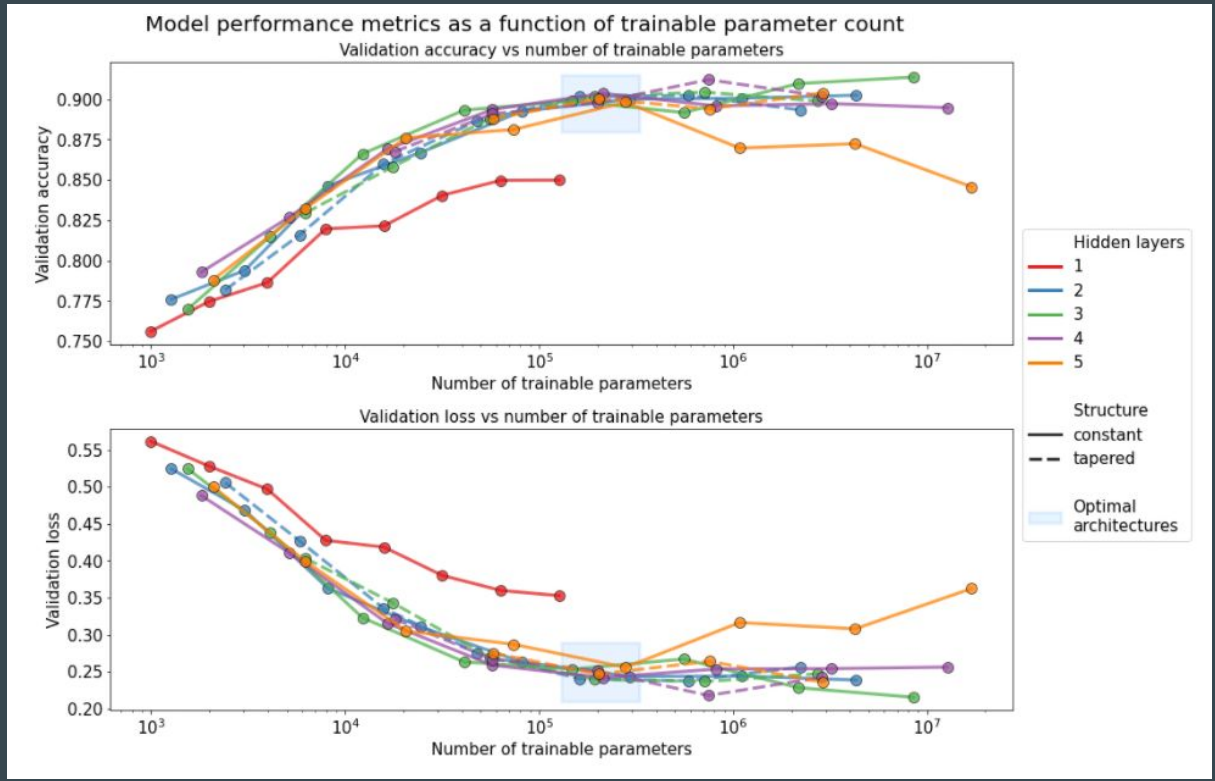
Strategy

Explore the multi-dimensional parameter space to determine sets of optimal parameters that produce the highest classification accuracy and lowest loss. For this problem, we use the categorical cross-entropy loss function.

Tuning the model architecture

We test several options for the network architecture. We vary the number of layers, the number of nodes in each layer, and the “structure” of the layers: i.e. constant layer size or tapered.

We identify a group of networks with around 10^5 parameters and use these as our starting point



Tuning the model architecture

We found a cluster of models with order 10^5 trainable parameters.

Models with `trainable_count` in the range [130k, 330k] ordered by decreasing `accuracy`:

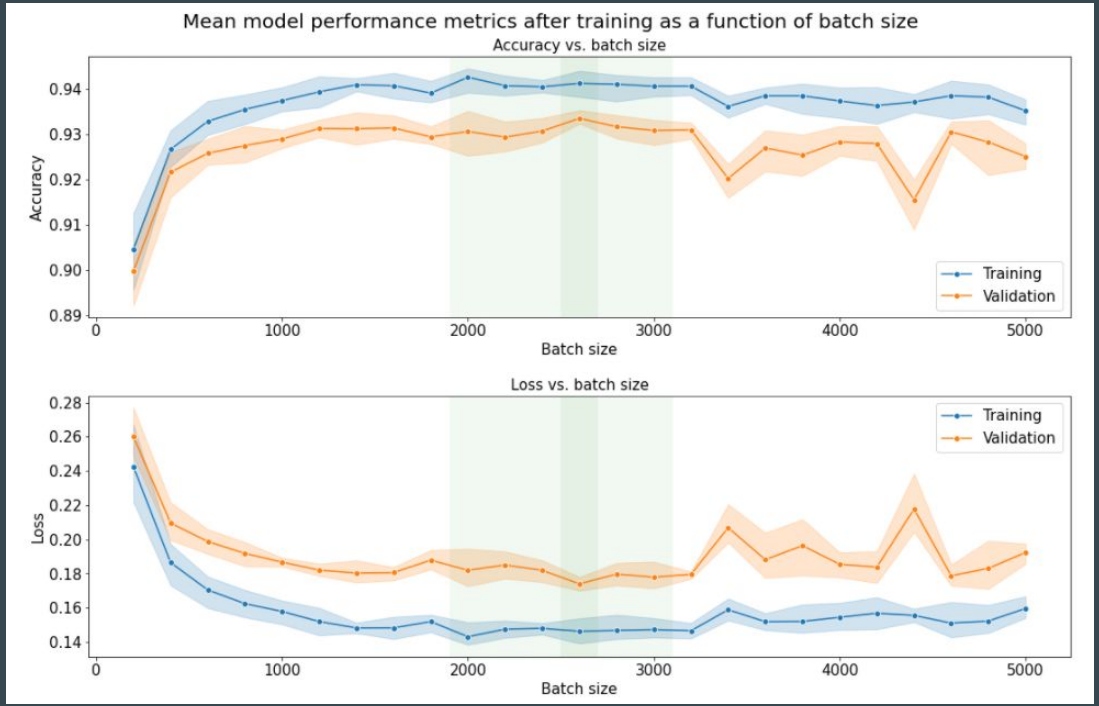
	model_id	nodes	layer_count	structure	trainable_count	non_trainable_count	loss	accuracy	val_loss	val_accuracy	training_time
32	256-256-256	[256, 256, 256]	3	constant	147463	0	0.230197	0.905882	0.253507	0.899077	160.677540
40	512-512	[512, 512]	2	constant	294407	0	0.229072	0.905880	0.243809	0.900110	164.923423
37	512-256-128	[512, 256, 128]	3	tapered	193287	0	0.232166	0.904692	0.239899	0.901809	160.676362
36	512-256	[512, 256]	2	tapered	161287	0	0.237309	0.902660	0.240695	0.901530	157.393599
39	512-256-128-64-32	[512, 256, 128, 64, 32]	5	tapered	202951	0	0.239118	0.902149	0.246892	0.900497	169.245430
33	256-256-256-256	[256, 256, 256, 256]	4	constant	213255	0	0.240900	0.901592	0.242219	0.903746	164.173456
34	256-256-256-256-256	[256, 256, 256, 256, 256]	5	constant	279047	0	0.241939	0.901149	0.256009	0.898216	172.347774
38	512-256-128-64	[512, 256, 128, 64]	4	tapered	201095	0	0.245037	0.899624	0.251196	0.898001	166.170155

We choose the top four, which also happen to be shallow models with two or three hidden layers.

	model_id	nodes	layer_count	structure	trainable_count
0	256-256-256	[256, 256, 256]	3	constant	147463
1	512-256	[512, 256]	2	tapered	161287
2	512-256-128	[512, 256, 128]	3	tapered	193287
3	512-512	[512, 512]	2	constant	294407

Tuning the batch size: Aggregate metrics

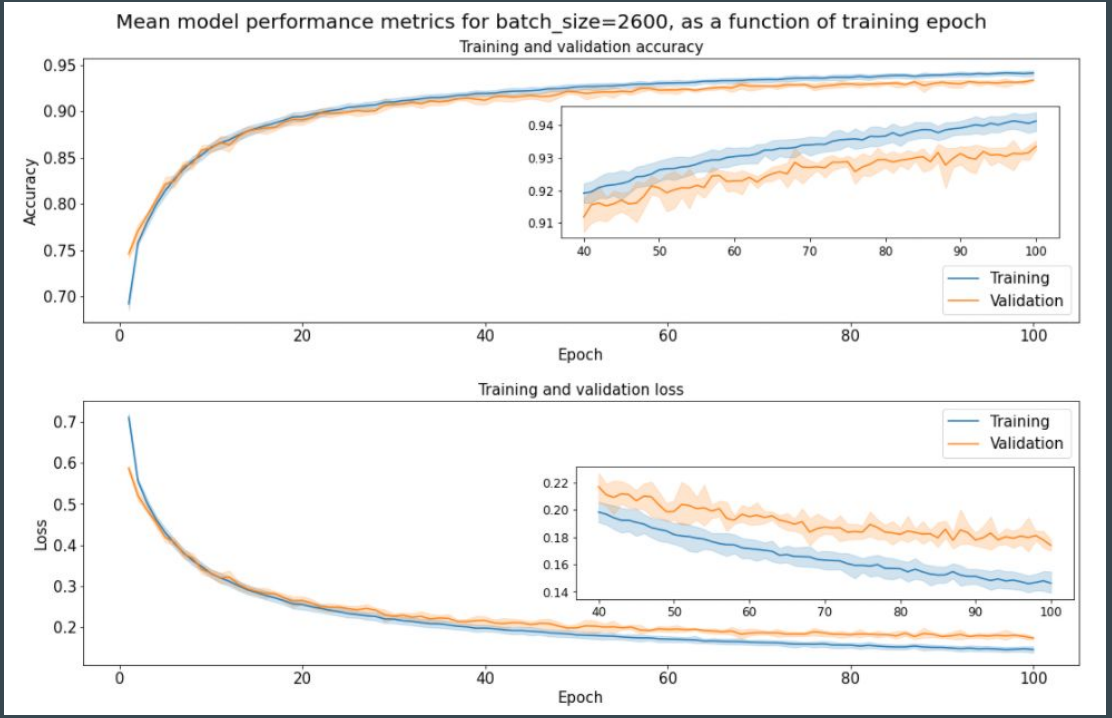
We narrowed our models down to four architectures. Next, we train these models over a wide range of batch sizes, plotting the aggregated model performance metrics. The solid lines are the average metrics and the bands are the one sigma width



Tuning the batch size: Aggregate metrics

If we fix the batch size, we can get an idea of the level of overfitting when we train beyond some optimal number of epochs.

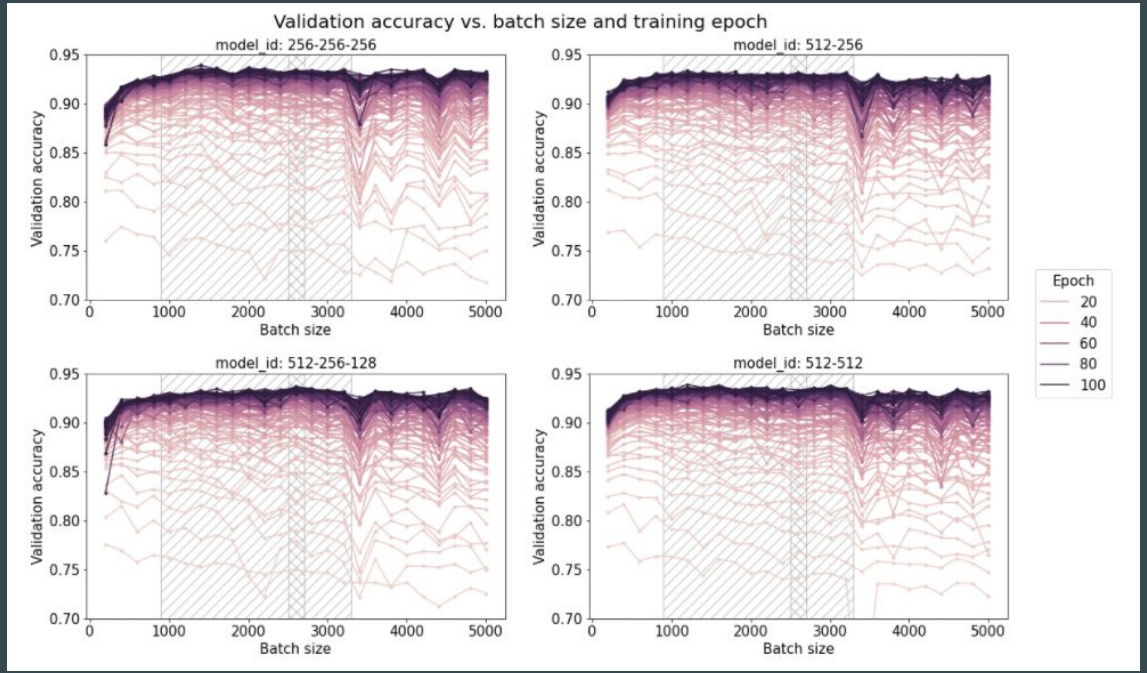
Overfitting would be signalled by a flattening in the validation performance while the training performance continues to increase



Tuning the batch size: Differentiating by architecture

We can explore the model-specific performance metrics, also showing the training progression.

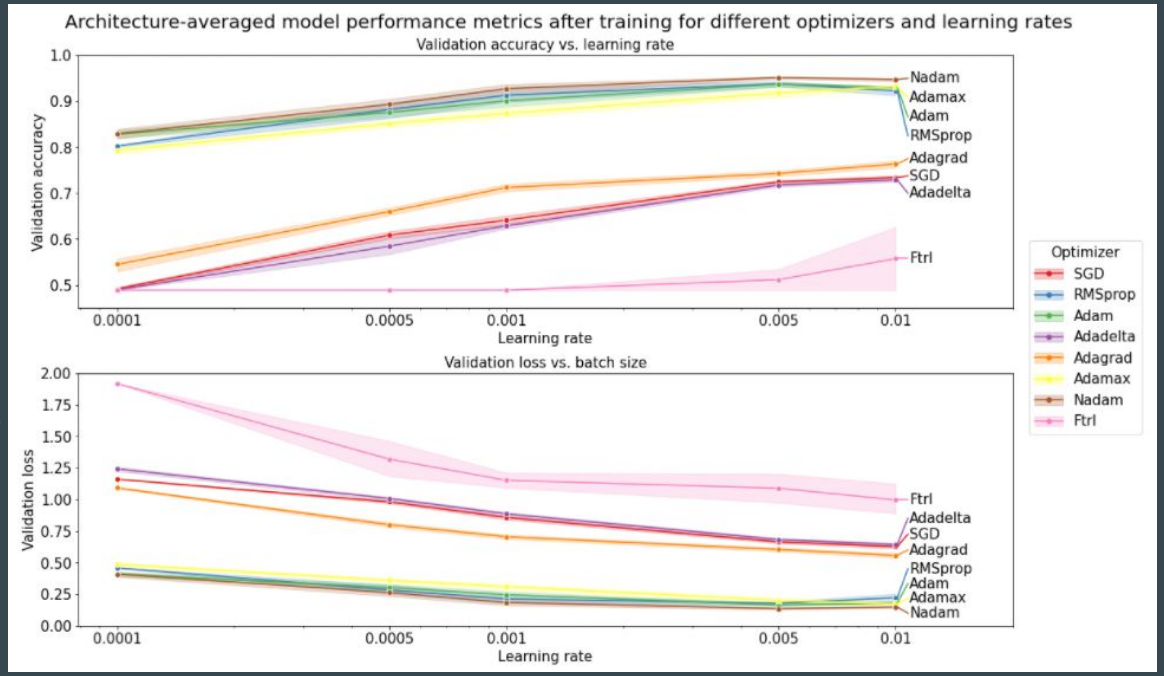
The conclusion is there seems to be a range of batch sizes where the metrics plateau. We choose a value of 1000 to tune the remaining parameters.



Tuning the optimizer

The optimizer parameters include the algorithm and the learning rate.

We have tested a variety of optimizers implemented. From the labeling on the plots, we can easily identify a list of "optimal optimizers", consisting of the Nadam, Adamax, Adam, and RMSprop algorithms.



Tuning the optimizer: Differentiating by Optimizer and Architecture and Learning Rate

We can explore each combination by making colorized tables

	First	Second	Third	Fourth	Fifth
Color	4	3	2	1	0

Optimizer: Nadam

learning_rate	0.0001	0.0005	0.001	0.005	0.01
model_id					
256-256-256	0.829113	0.904563	0.937372	0.954089	0.950474
512-256	0.817001	0.874271	0.909404	0.947893	0.941869
512-256-128	0.843291	0.907962	0.937458	0.954713	0.949807
512-512	0.828317	0.886922	0.926873	0.950216	0.949097

Optimizer: RMSprop

learning_rate	0.0001	0.0005	0.001	0.005	0.01
model_id					
256-256-256	0.804114	0.901702	0.920806	0.946064	0.922334
512-256	0.802048	0.853747	0.892989	0.929756	0.928207
512-256-128	0.800499	0.892580	0.935522	0.936791	0.908091
512-512	0.805125	0.881371	0.904283	0.940148	0.932790

Optimizer: Adamax

learning_rate	0.0001	0.0005	0.001	0.005	0.01
model_id					
256-256-256	0.796842	0.856199	0.877154	0.930036	0.934812
512-256	0.782040	0.840688	0.861513	0.904090	0.924421
512-256-128	0.801855	0.860524	0.889396	0.925561	0.936727
512-512	0.792173	0.849767	0.867752	0.913728	0.934124

Optimizer: Adam

learning_rate	0.0001	0.0005	0.001	0.005	0.01
model_id					
256-256-256	0.833437	0.884039	0.915299	0.941632	0.930273
512-256	0.815753	0.860373	0.887631	0.928358	0.927153
512-256-128	0.840795	0.885502	0.910286	0.938599	0.931585
512-512	0.823089	0.873066	0.892300	0.941804	0.932682

We find learning rates of 0.005 and 0.01 perform the best over almost all architectures and optimizers. When we tune the dropout rate, we also examine a learning rate of 0.001.

Tuning the Dropout Rates

Dropout is a regularization technique designed to reduce overfitting and increase network generalization. Dropout refers to randomly “dropping out” nodes in a neural network during each training epoch. This has the effect of “averaging over model architectures”, because the network was trained on a variety of random subsets of its nodes.

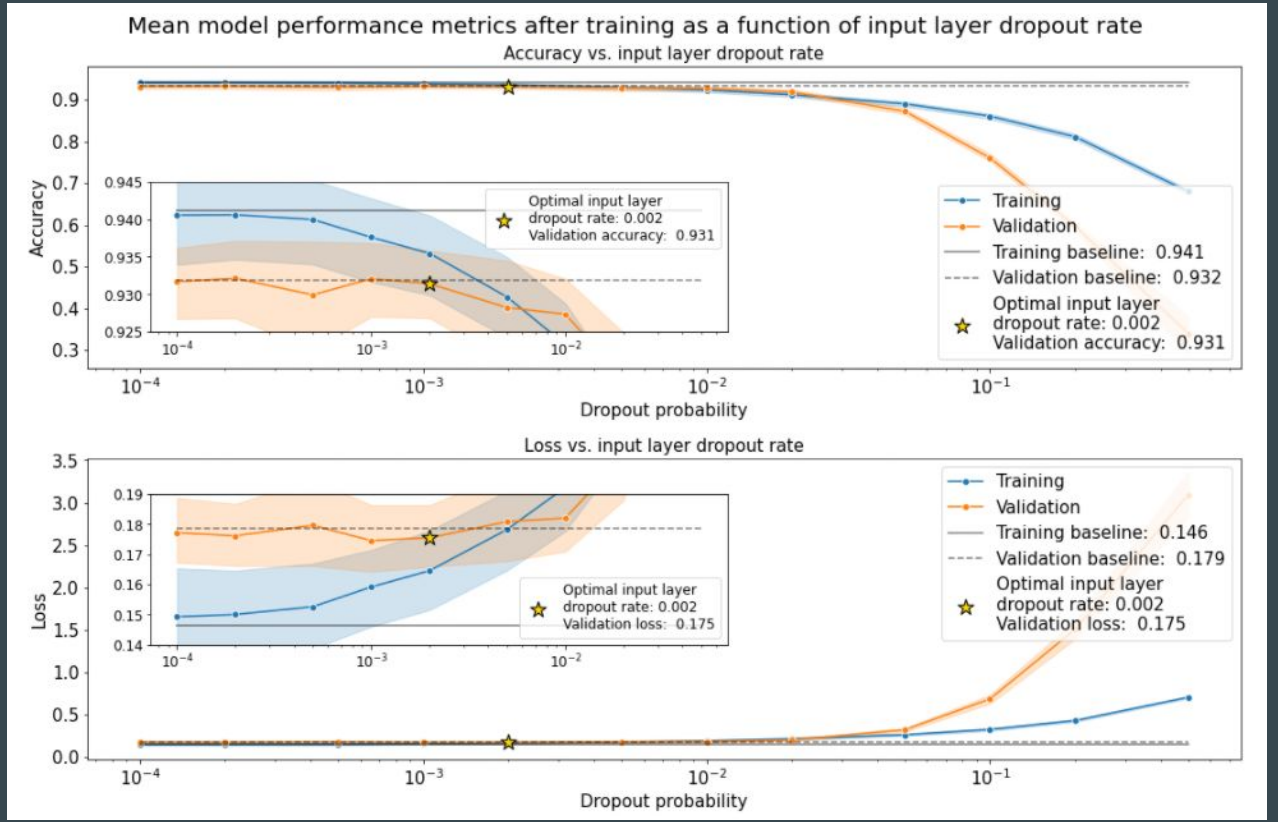
There are three parameters:

- Input Layer dropout probability: How often we drop out a node on the input layer
- Hidden Layer dropout probability: How often we drop out a node on the hidden layers
- Weight Constraint value: A parameter used to constrain the norm of the weights, so that weights do not become to large to overcompensate for dropped nodes

For tuning dropout, we use the Adam optimizer

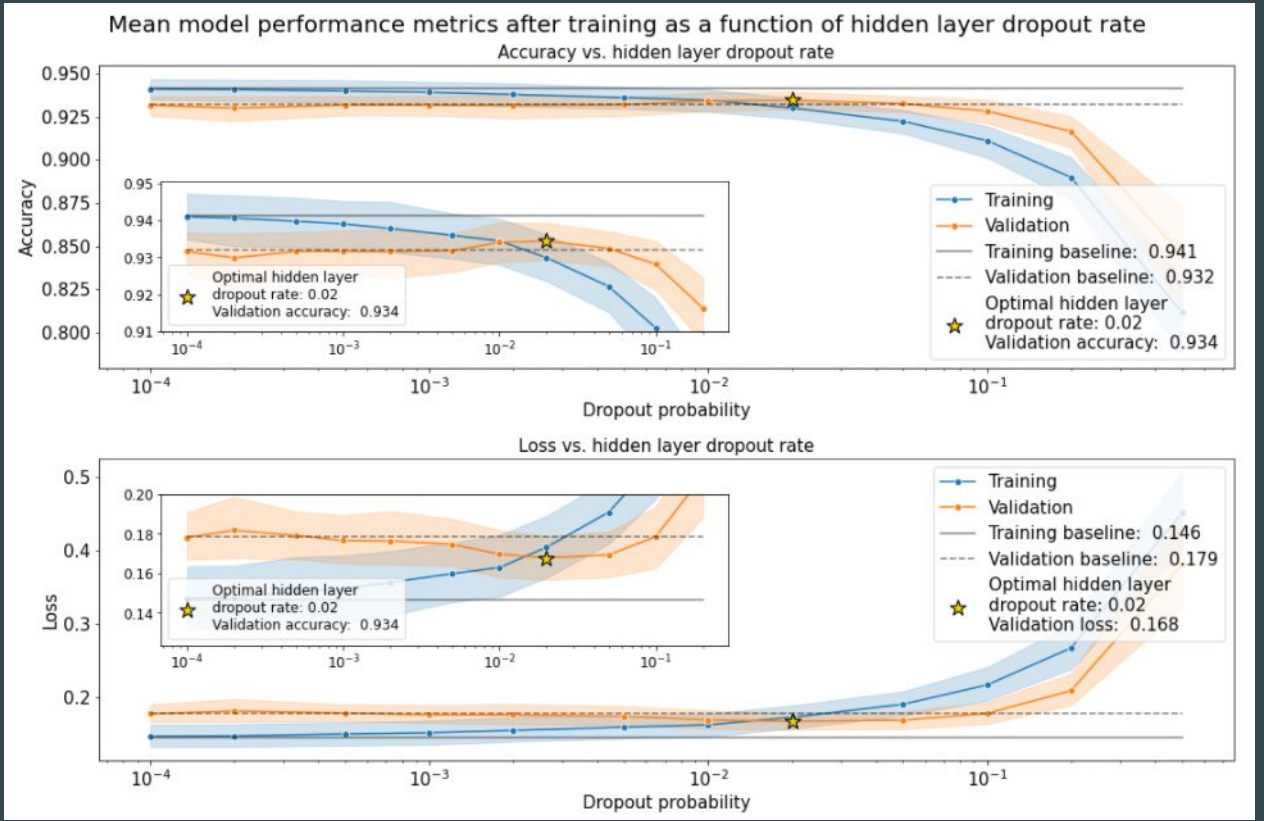
Tuning the Input Layer Dropout Rates: 1-D analysis

We test all four model architectures and all three learning rates and vary the input layer dropout, compared to the baseline performances with no dropout. We want the largest value for the dropout rate before the performance deteriorates



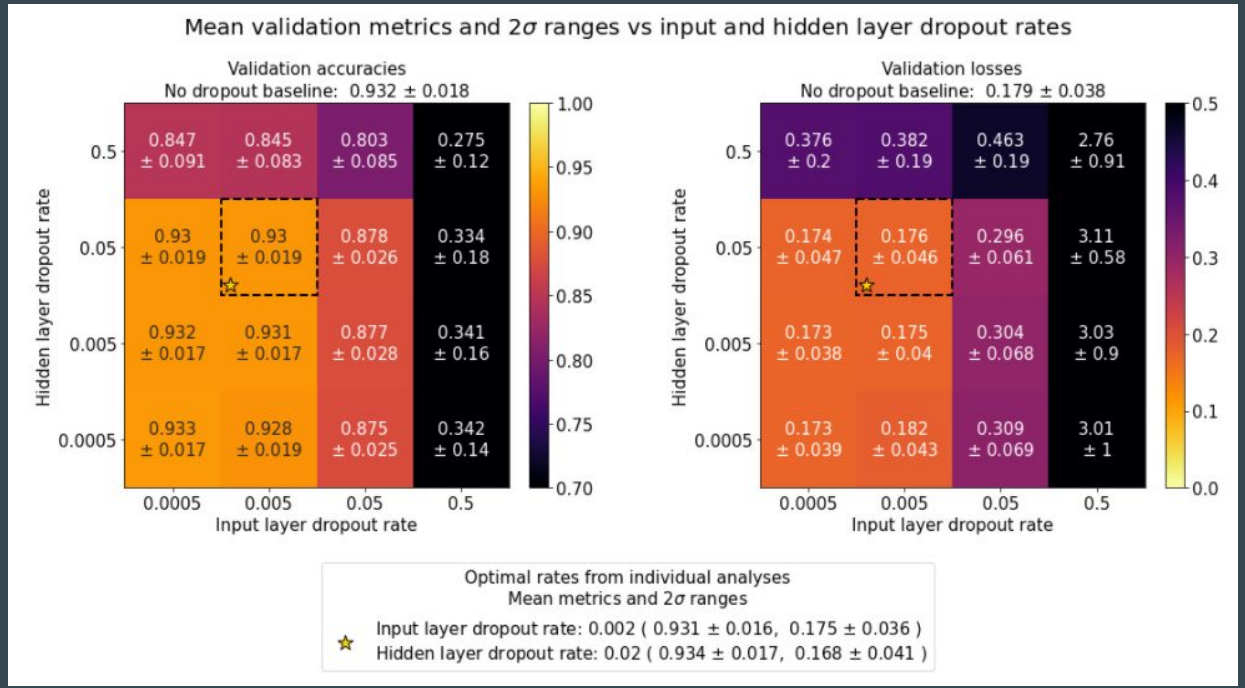
Tuning the Hidden Layer Dropout Rates: 1-D analysis

We test all four model architectures and all three learning rates and vary the input layer dropout, compared to the baseline performances with no dropout. We want the largest value for the dropout rate before the performance deteriorates



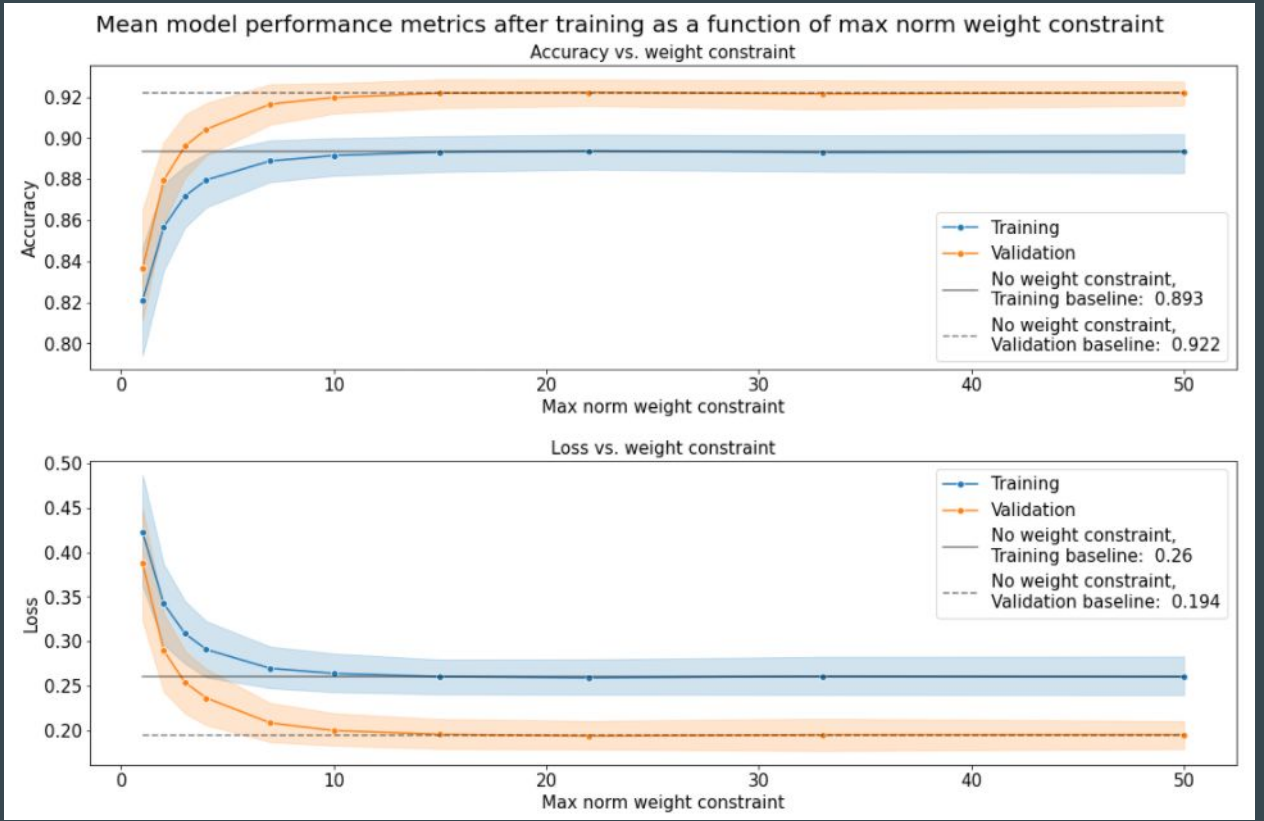
Tuning the Dropout Rates: 2-D analysis

We test all four model architectures and all three learning rates and vary the input and hidden layer dropout rate together, compared to the baseline performances with no dropout. We want the largest value for the dropout rate before the performance deteriorates



Tuning the Weight Constraint: 1-D analysis

We fix a level of dropout and vary the weight constraint. A small weight constraint greatly inhibits the performance of the network on this dataset, and loosening the constraint allows the performance to reach the baseline (no weight constraint). We conclude that constraining the weights does not give an advantage for this dataset.



Optimal Dropout Parameters

We conclude our dropout parameter study by identifying two sets of parameters:

Optimal dropout parameters

The analysis of the dropout rates and max norm weight constraints, and the effect of changing the learning rates, has suggested the following two sets of dropout parameters, one for each learning rate we have included:

- **Dropout set 1**:
  - Learning rate: `0.001`
  - Input layer dropout rate: `0.01`
  - Hidden layer dropout rate: `0.1`
  - Max norm weight constraint: None
- **Dropout set 2**:
  - Learning rate: `0.01`
  - Input layer dropout rate: `0.005`
  - Hidden layer dropout rate: `0.05`
  - Max norm weight constraint: None

We can combine these sets of dropout parameters with the other optimal parameters we have determined, to create one final list of optimal parameter choices. Then, we can train models with each set of optimal parameters to determing the number of epochs to train for each to arrive at a full set of optimal parameters.

Remaining Analysis

We combine the results for the parameter searches to find orthogonal optimal sets for different types of hyperparameters. We have identified 4 model architectures, 4 optimizers, and two sets of dropout parameters (with different learning rates).

What is left to do is to test the specific combinations to see what the final best set of parameters (or top two or three).

Finally, we would evaluate these networks on the test set, which we have so far not touched.

Final conclusions to come at a later date