AML-CW-4
A Project on Human Hand Tracking

Youngbin Lee
Department of Industrial Engineering
UNIST
young@unist.ac.kr

Abstract

Regression algorithms are used to conduct prediction tasks by estimating functions where the target variables are continuous. In this project, we use 3D Hand Pose Estimation images to predict hand joint locations. We constructed different machine learning regression models and neural network regression models to experiment how they perform in our prediction tasks. In experiment, we compared different hyperparameters and components to effectively examine performance of different algorithms. We also provide some discussions on the effect of such variants of algorithms.

1 Introduction

1.1 Problem Definition

We aim to compare different regression algorithms on the image dataset. Our prediction task is a human hand pose estimation. That is, the models are trained to predict hand joint locations when a set of depth maps depicting hands.

Comparing different combinations of hyperparameters and ablation studies are important in our project because they are known to affect model performance greatly (1). Hyperparameters are defined by users and they determine the hypothesis and architecture of the models. Ablation studies are conducted to see the effects of adding or removing specific components in model architectures.

1.2 Data Description

The ICVL Hand Posture Dataset is used for the problem of human hand pose estimation where the goal is to predict 21 hand joint locations from a single input depth map. Because the ground-truth target output is multi-dimensional, we construct vector-valued regression models.

The inputs are single depth images with different hand poses (i.e., each image has 1 channel with 240 height and 320 width). And they are flattened into 1-dimension to be entered by the models except CNNs. Each pixel values indicating depths are normalized with its mean and standard deviation (i.e., standardization) before putting into the model.

1.3 Regression Algorithms

In this project, four different machine learning models and neural network models are used for experiments. The hypothesis and architecture of each models will be determined by different hyperparameters in experiment, section 2

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Gaussian Process (GP) regressor is a non-parametric, Bayesian approach to regression where the prediction is probablistic. This model produces flexible predictions as it combines training and testing, but it takes a long time to predict.

Random Forests (RF) is an ensemble model that uses multiple random trees to prevent overfitting and enhance generalization performance. A tree-based models are known to perform well with tabular data and RFs are not so complicated yet perform well.

Multi-layer Perceptrons (MLPs) are the most basic form of neural networks. With varying number of layers and neurons that users define, it can approximate any function. In this project, considering the size of dataset and complexity of task, 3 fully-connected layers are used. In this project, we trained this model with 100 epochs throughout the whole experiments as it seems to converge enough.

Convolutional Neural Networks (CNNs) are at the core of most state-of-the-art computer vision solutions for a wide variety of tasks (2). They are designed to extract low-level features and high-level features from images by using filters in convolutional layers. Unlike MLPs, each neurons are not fully-connected but they focus on local features. In experiment, we used a framework of LeNet (3) as the size of model is not so big yet it performs fairly well on image data. More specifically, a 3-layers with convolution, ReLU activation, and max-pooling are used, followed by a fully-connected layer to produce an output. In this project, we trained this model with 30 epochs throughout the whole experiments as it seems to converge enough.

2 Experiment

2.1 Data

The dataset consists of 16008 training set and 1596 testing set. In this project, the training set is divided into a training set and a validation set to find the best hyperparameters. And we finally evaluate the performance of models with testing set to find the best model.

To see how models perform with varying number of training instances, we randomly sampled 30%, 60%, and 100% of training set. Then, we randomly sampled 20% of training set for a validation set. For example, with 30% sample, the original 16008 training instances become 4802 instances. Then, they are separated into 3841 training set and 960 validation set. The testing set always remains the same in any experiment because we use it is used for a final evaluation regardless of the number of training instances that the model have seen.

2.2 Hyperparameter Optimization: Search

To find the optimal set of hyperparameters, we used a separate validation dataset. With several considered hyperparameters and corresponding search ranges in figure 1, we performed a grid search to find the best combination of hyperparameters. In other words, models with the lowest validation error are going to be used for final evaluation with the testing set.

Gaussian Process (GP) regressor We used a Radial-basis function (RBF) kernel and a length scale l using scalar is searched in a range. And a value α is searched as well, which is added to the diagonal of the kernel matrix.

Random Forest The number of trees (N_estimators), minimum number of samples to split an internal node (Min_samples_split), maximum depth of the trees (Max_depth), and the number of features to consider to determine the best split (Max_features) are searched. They all determine the architecture of the trees or forests.

MLPs will search the number of nodes in hidden layers (Hidden_size) in a range of [80, 100]. This range is considered because they don't make the model too complex yet they are bigger than the output nodes which is 63. Activation functions that are applied to the first two fully connected layers are also searched using a sigmoid function and a hyperbolic tangent (tanh) function.

Model	Hyperparameter	Range
Gaussian Process	l (RBF kernel)	[0.1, 10]
	α	[1e-10, 1e-3, 1e-2]
Random Forests	N_estimators	[50, 100, 150]
	Min_samples_split	[2, 5, 10]
	Max_depth	[None, 2, 5]
	Max_features	[None, sqrt, log2]
MLPs	Hidden_size	[80, 100]
	Activation	[sigmoid, tanh]
CNNs	Channels	[[6,6,16], [32,32,64]]
	Kernel_size	[3, 6]

Figure 1: Hyperparameter search range

CNNs In this experiment, CNNs follow an architecture of LeNet, the number of channels in each convolutional layer is searched with the same values in LeNet and the other values that are bigger than that. This is because we want to see if a larger number of channels will be effective to extract features from input images. And the kernel size is searched.

2.3 Hyperparameter Optimization: Results

The performance of hyperparameters from each models is shown in figures [2 - 4]. For a basic evaluation of models, a Mean Squared Error (MSE) is used to calculate error. And runtime is recorded in seconds during training and testing on validation set.

Gaussian Process (GP) regressor The considered hyperparameters did not make a significant difference on runtime and accuracy in results in figure 2. In fact, the length scale of the kernel (l) had no effect on accuracy and only α made a difference lowering the error as it goes larger. Thus, we conclude Gaussian Process model best works with our dataset with hyperparameters producing the lowest MSE (in bold).

Model	*Kernel*	l	α	Runtime (train)	Runtime (valid)	MSE (valid)
Gaussian Process	RBF	0.1	1e-10	2418	581	4055.4497e-05
			1e-3	2396	563	4055.4496e-05
			1e-2	2355	573	**4055.4488e-05**
		10	1e-10	2363	570	4055.4497e-05
			1e-3	2376	569	4055.4496e-05
			1e-2	2358	562	4055.4488e-05

Figure 2: Hyperparameter-tuning results: GP

Random Forest In figure 13, we can see more trees in Random Forest are helpful, as MSE generally goes down as N_estimators goes up. And the minimum number of samples to split an internal node should be small to get a better accuracy. Max depth determines the size of each trees, and the result says it is better to let the trees grow untill all leaves are pure ('None'). The last hyperparameter

'Max_features' gives the most clear difference to the model: the number of features to randomly consider in trees works best with square root function ('sqrt'), because it always leads to the best MSE values that are marked in bold. On the other hand, using all features ('None') is inefficient as it takes too much time to train, yet the error is still higher than the one with 'sqrt'. We find the best combination of hyperparameters with the lowest MSE on valid set, 5.89e-05, and the corresponding combination is [N_estimators=150, Min_samples_split=5, Max_depth=None, Max_features=sqrt].

MLPs From the results in figure 3, more nodes in hidden layers turned out to be beneficial because MSE goes down. Activation functions works a bit randomly in MSE, but its runtime for training is much shorter when Sigmoid function is used. In the end, Sigmoid function showed the best MSE (in bold) with the largest hidden_size, 130.

Model	*Hidden_size*	*Activation*	**Runtime (train)**	**MSE (valid)**
MLPs	30	Sigmoid	455	0.2e-05
		Tanh	1072	0.3e-05
	80	Sigmoid	732	0.1e-05
		Tanh	1069	0.2e-05
	130	Sigmoid	714	**00.8e-05**
		Tanh	1487	0.2e-05

Figure 3: Hyperparameter-tuning results: MLP

CNNs Results in figure 4 show that larger number of channels do not help lowering MSE but lead to much more time to train. Thus, We conclude that we don't need too many channels to effectively extract features from our input images that are depth maps simply depicting hands with different poses. And the kernel size works better with larger value as the lowset MSE is shown in the kernel size of 6. It would be great to search for more hyperparameters and larger range of parameters, that this project couldn't have done due to computing power and time for training. If more kernel size was combined with different hyperparameters such as the size of stride, there might be bigger difference in MSE.

Model	*Channels*	*Kernel_size*	**Runtime (train)**	**MSE (valid)**
CNNs	[6, 6, 16]	3	409	6.6e-05
		6	482	**6.5e-05**
	[32, 32, 64]	3	2343	6.5e-05

Figure 4: Hyperparameter-tuning results: CNN

2.4 Accuracy and Runtime

With the hyperparameters that showed the best performance, we finally evaluated each models on a test dataset and the results can be found in figure 5. We trained each models with varying number of training instances that are specified in 'Training size'. Note that in Gaussian Process, we could not train the model with 100% training size due to the lack of computing power. Except this model, in general, MSE on test set goes down as the model is trained with more data. In other words, the generalization performance gets better with larger number of training data. Runtime also goes up as we use more training samples, but this difference was particularly big in Gaussian Process. Other models showed tolerable training time even if more samples are used.

Model	Hyperparameter	Training size	Runtime (train)	MSE (test)
Gaussian Process	(RBF kernel, l=0.1, α=1e-2)	30%	573	4507.5e-05
		60%	2402	**4507.5e-05**
Random Forests	(N_estimators=150, Min_samples_split=5, Max_depth=None, Max_features=sqrt)	30%	32	43.2e-05
		60%	72	41.9e-05
		100%	128	**40.8e-05**
MLPs	(Hidden_size=130, Activation=sigmoid)	30%	113	131.1e-05
		60%	215	124.5e-05
		100%	358	**124.0e-05**
CNNs	(Channels=[6, 6, 16], Kernel_size=6)	30%	103	4510e-05
		60%	195	4511e-05
		100%	331	**4510e-05**

Figure 5: Accuracy and Runtime

However, the results show that the models are overfitted to training data, except Gaussian Process where the validation loss and test loss are similar. Since validation dataset is made from training data and validation loss actually goes down as we use more training set, it may be probable that the testing images have different characteristics to training images.

2.5 Ablation Studies

In field of neural networks, there have been attempts to design its architecture for a better performance in terms of overfitting and computational cost. Thus, to see their effects on models, we performed ablation studies on MLPs and CNNs by adding and removing dropout and normalization techniques. In figure 6 9, experiments of hyperparameter-tuning are shown with some new components. In this figure, you can see MSE on test set only for model with the best hyperparameters that showed the lowest MSE on valid set (marked in bold).

MLPs Figure 6 shows the effect of layer normalization and dropout on MLPs that we experimented. When adding layer normalization layer on after each three fully-connected layers, the runtime is reduced but the model performance in terms of MSE did not change when compared to the results of original MLPs in figure 3. Also, its overfitting problem still exists as MSE on test set is still higher than validation error. When adding dropout with probability of 40%, the model performance got worse but the runtime got much better compared to the original MLPs as we can see in figure 7. Thus, we conclude that dropout brings a shorter runtime but the performance goes down when dropout probability is too high.

CNNs In CNNs, we applied different normalization technique that is known to work well with CNNs: batch normalization. However, due to the lack of computing power, we added a batch normalization layer only once after the first convolutional layer and we didn't experiment with larger number of channels [32, 32, 64]. Our results in figure 8 says that a batch normalization helps reducing runtime but the model accuracies remains almost the same in terms of both valid set and test set, when compared to the results of original CNNs in figure 4. Its performance could get better if more normalization layers are added. Dropout with probability of 40% reduced overall runtime significantly too, as shown in figure 9. However, model performance did not change much than the original CNNs. It would have been better if experiments are done with lower dropout probability.

Model	Hidden_size	Activation	Runtime (train)	MSE (valid)	MSE (test)
MLPs with *LayerNorm*	30	Sigmoid	120	0.2e-05	-
		Tanh	118	**0.2e-05**	**172.93e-05**
	80	Sigmoid	118	0.2e-05	-
		Tanh	147	0.2e-05	-
	130	Sigmoid	121	0.2e-05	-
		Tanh	416	0.2e-05	-

Figure 6: Hyperparameter-tuning results: MLPs with Layer Normalization

Model	Hidden_size	Activation	Runtime (train)	MSE (valid)	MSE (test)
MLPs with *DropOut*	30	Sigmoid	119	1.1e-05	-
		Tanh	118	**0.2e-05**	**765.30e-05**
	80	Sigmoid	121	1.0e-05	-
		Tanh	120	1.5e-05	-
	130	Sigmoid	121	1.1e-05	-
		Tanh	174	1.5e-05	-

Figure 7: Hyperparameter-tuning results: MLPs with DropOut

2.6 Other models

Extra Trees Results of the accuracy and runtime and ablation studies say that the best model so far is Random Forest as it shows the lowest MSE on testing data. Since random forest worked well, we did the sample experiment with some other tree-based ensemble methods to see if a better model exists for our dataset. Extra Trees, also known as an extremely randomized trees, are a variant of Random Forest that has more randomness of algorithm. In figure 10, we found that the model performance with the best hyperparameter is comparable with Random Forest. Note that we only used 30% of training dataset for this model and the best hyperparameters are the same with Random Forest. Thus, if the model is trained with more data, it is highly likely to beat Random Forest.

Support Vector Machines (SVM) We also experimented SVM to explore other regression models. In figure 12, we can see that the linear kernel outperformed non-linear kernel and the regularization parameter C has no effect on model performance. But still, its accuracy on test set is lower than Random Forest and MLPs, according to the figure 5.

3 Conclusion

With a 3D Hand Pose Estimation dataset, we constructed regression models to predict hand joint locations from single depth images with hand poses. We first compared hyperparameters of each models by calculating MSE on validation set separaed from training set. Results on hyperparameter-tuning show that hyperparameter selection plays an important role for specific models in improving model performance. Then, with the best hyperparameters selected, models are evaluated by MSE on testing dataset. When testing, we not only used model trained with a full training data but we trained model with varying number of training instances and analyzed corresponding runtime and accuracies.

In ablation studies, we added layer normalization, batch normalization, and dropout into neural network regression algorithms (MLPs and CNNs) and compared the results with original algorithms without them. The results say that applied techniques reduced training time significantly but they did not have an impact on prediction errors.

When evaluating four models (GP, RF, MLPs, and CNNs) with their best hyperparameters, RF showed the best accuracy on test set with minimum MSE of 40.8e-05. Then, we further examined another tree-based ensemble algorithm. Extra trees showed competitive performance on test set (with MSE of 44.20e-05) even when trained with much smaller training instances than Random Forest.

Model	Channels	Kernel_size	**Runtime (train)**	**MSE (valid)**	**MSE (test)**
CNNs with *BatchNorm*	[6, 6, 16]	3	199	6.6e-05	-
		6	343	**6.5e-05**	**4509.9e-05**
	[32, 32, 64]	3	644	6.5e-05	
		6	767	6.5e-05	

Figure 8: Hyperparameter-tuning results: CNNs with Batch Normalization

Model	Channels	Kernel_size	**Runtime (train)**	**MSE (valid)**	**MSE (test)**
CNNs with *DropOut*	[6, 6, 16]	3	194	6.7e-05	-
		6	345	**6.5e-05**	**4509.5e-05**

Figure 9: Hyperparameter-tuning results: CNNs with DropOut

References

[1] Claesen, M., De Moor, B. (2015). Hyperparameter search in machine learning.arXiv preprint arXiv:1502.0212

[2] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

[3] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86 (11), 2278-2324.

A Appendix

Model	*N_estimators*	*Min_samples_split*	*Max_depth*	*Max_features*	Runtime (train)	MSE (valid)	MSE (test)
Extra Trees	100	2	None	Sqrt	41	8.50e-05	-
				Log2	6	9.44e-05	-
			2	Sqrt	9	101.56e-05	-
				Log2	1	121.37e-05	-
		5	None	Sqrt	36	8.45e-05	-
				Log2	5	10.36e-05	-
			2	Sqrt	9	101.56e-05	-
				Log2	1	121.37e-05	-
	150	2	None	Sqrt	63	8.43e-05	-
				Log2	10	9.35e-05	-
			2	Sqrt	13	100.25e-05	-
				Log2	1	121.24e-05	-
		5	None	Sqrt	54	**8.39e-05**	**44.66e-05**
				Log2	9	10.31e-05	-
			2	Sqrt	13	100.25e-05	-
				Log2	1	121.24e-05	-

Figure 10: Hyperparameter-tuning results: Extra Trees

Model	Hyperparameter	Training size	Runtime (train)	MSE (test)
Extra Trees	(N_estimators=150, Min_samples_split=5, Max_depth=None, Max_features=sqrt)	10%	12	48.40e-05
		20%	31	**44.20e-05**
		30%	54	44.66e-05

Figure 11: Accuracy and Runtime: Extra Trees

Model	*Kernel*	*C*	Runtime (train)	MSE (valid)	MSE (test)
Support Vector Machine	Linear	1.0	1457	**155.2e-05**	**165.5e-05**
		10	1506	155.2e-05	-
	RBF	1.0	1773	165.4e-05	-
		10	1827	165.4e-05	-

Figure 12: Hyperparameter-tuning results: SVM

Model	*N_estimators*	*Min_samples_split*	*Max_depth*	*Max_features*	**Runtime (train)**	**Runtime (valid)**	**MSE (valid)**
Random Forest	50	2	None	None	6700	0.60	6.53e-05
				Sqrt	58	0.58	**6.04e-05**
				Log2	10	0.61	6.09e-05
			2	None	1377	0.24	0.0010
				Sqrt	7	0.26	0.0009
				Log2	1	0.24	0.0011
			5	None	2268	0.27	0.0004
				Sqrt	12	0.28	0.0004
				Log2	2	0.29	0.0005
		5	None	None	6246	0.56	6.66e-05
				Sqrt	47	0.56	**6.03e-05**
				Log2	7	0.57	6.27e-05
			2	None	1379	0.25	0.0010
				Sqrt	7	0.24	0.0009
				Log2	1	0.24	0.0011
			5	None	2315	0.37	0.0004
				Sqrt	11	0.27	0.0004
				Log2	2	0.35	0.0005
		10	None	None	4835	0.50	7.18e-05
				Sqrt	42	0.58	**6.43e-05**
				Log2	6	0.56	6.92e-05
			2	None	1381	0.23	0.0010
				Sqrt	7	0.24	0.0009
				Log2	1	0.24	0.0011
			5	None	2592	0.29	0.0004
				Sqrt	11	0.27	0.0004
				Log2	2	0.30	0.0005
	100	2	None	None	13393	0.76	6.45e-05
				Sqrt	117	0.77	**5.92e-05**
				Log2	19	0.76	5.97e-05
			2	None	2750	0.27	0.0010
				Sqrt	13	0.25	0.0009
				Log2	2	0.29	0.0011
			5	None	4469	0.31	0.0004
				Sqrt	23	0.29	0.0004
				Log2	3	0.31	0.0005
		5	None	None	12847	0.88	6.59e-05
				Sqrt	94	0.72	**5.92e-05**
				Log2	14	0.74	6.15e-05
			2	None	3351	0.34	0.0010
				Sqrt	13	0.26	0.0009
				Log2	2	0.28	0.0011
			5	None	4855	0.48	0.0004
				Sqrt	22	0.31	0.0004
				Log2	3	0.33	0.0005
		10	None	None	10075	0.61	7.10e-05
				Sqrt	83	0.70	**6.32e-05**
				Log2	12	0.70	6.84e-05
			2	None	2844	0.28	0.0010
				Sqrt	13	0.27	0.0009
				Log2	2	0.26	0.0011
			5	None	4472	0.32	0.0004
				Sqrt	23	0.30	0.0004
				Log2	3	0.31	0.0005
	150	2	None	None	21789	0.94	6.40e-05
				Sqrt	173	0.93	**5.89e-05**
				Log2	28	0.98	5.93e-05
			2	None	4151	0.25	0.0010
				Sqrt	19	0.29	0.0009
				Log2	2	0.27	0.0011
			5	None	6779	0.40	0.0004
				Sqrt	33	0.33	0.0004
				Log2	5	0.34	0.0005
		5	None	None	21698	0.32	6.54e-05
				Sqrt	141	0.87	**5.89e-05**
				Log2	20	0.88	6.10e-05
			2	None	5779	0.26	0.0010
				Sqrt	19	0.28	0.0009
				Log2	2	0.30	0.0011
			5	None	9527	0.71	0.0004
				Sqrt	33	0.33	0.0004
				Log2	5	0.33	0.0005
		10	None	None	11451	0.40	8.92e-05
				Sqrt	123	0.84	**6.28e-05**
				Log2	17	0.84	6.81e-05
			2	None	4284	0.30	0.0010
				Sqrt	19	0.29	0.0009
				Log2	2	0.26	0.0011
			5	None	6734	0.33	0.0004
				Sqrt	36	0.37	0.0004
				Log2	5	0.38	0.0005

Figure 13: Hyperparameter-tuning results: RF