This project aims to train a vortex-induced vibration (VIV) predictor, mapping flow and response parameters (*Re*, *f*, *A/D*, *i*) to lift coefficient distributions (*CL*(*x*)) on flexible risers. 74 datasets are applied to perform supervised learnings. A fully-connected neural network (FCNN) and a Gaussian process regression (GPR) model are trained, tuned and tested, using Keras and MATLAB `fitrgp` respectively. They are further applied to higher-mode and multi-mode cases to test robustness.

## 1. Data Preprocessing (MATLAB)

Based on prior knowledge, mode *i* (1:1:5) largely determines the *CL*(*x*) shape. Thus, the 74 datasets are organized according to modes and each mode has ~14 datasets. They are randomly sliced into 10 groups (with 4 leave-out sets) and it's guaranteed that each includes all 5 modes. Therefore, we have 63 training sets and 7 testing sets in each of the 10 folds. *Re*, *f* and *A/D* are standardized based on training sets to have zero mean and unit variance.

## 2. Loss Function and Mode Encoding

To compute losses, predicting lift coefficients directly and comparing with *CL*(*x*) is hardly a good idea. The models will learn the spatial sampling information which undermines generalizations. By prior knowledge, lift coefficients are mostly close to zeros at $x = 0, L$ and have very harmonic-like distributions along *x* axis. Therefore, *CL*(*x*) is approximated by sum of *N* sine functions as:

$$CL(x) = \sum_{j=1}^{N} c_j \sin(\frac{j\pi x}{L}) \tag{1}$$

where $c_j$ are fitted approximation coefficients.

However, comparing predicted and fitted $c_j$ in loss computations is still not a perfect choice. It'd be better to compare predicted $c_j$ with the original *CL*(*x*) curve. Therefore, in FCNN, the coefficients $\hat{c}_j$ from the output layer are fed into a new loss function. The loss function, as a user-defined function in Keras, will map $\hat{c}_j$ to $\widehat{CL}(x)$ by Eq. (1) and calculate its mean squared error to *CL*(*x*) (averaging SE over 99 sample points). This arrangement makes *N* a hyper-parameter (output size) in the learning algorithm. It reduces iterative work in *N* tuning and makes it drastically easier. The losses can also give straightforward sense of the prediction accuracy which is of interest.

The input modes (*i* = 1~5) are kept as they are, but I cannot find obvious ordering features from the processed data. I was considering encoding them as one-hot vectors. However, it can hamper generalizing the predictor to higher modes, although low modes are of most interest. In order to investigate the trade-off, a 10-fold cross validation is performed with the tuned FCNN. The two encodings have average testing losses of 0.011 (original) and 0.010 (one-hot). Therefore, regardless of slightly higher loss, the original modes *i* will be adopted for the sake of generalizations.

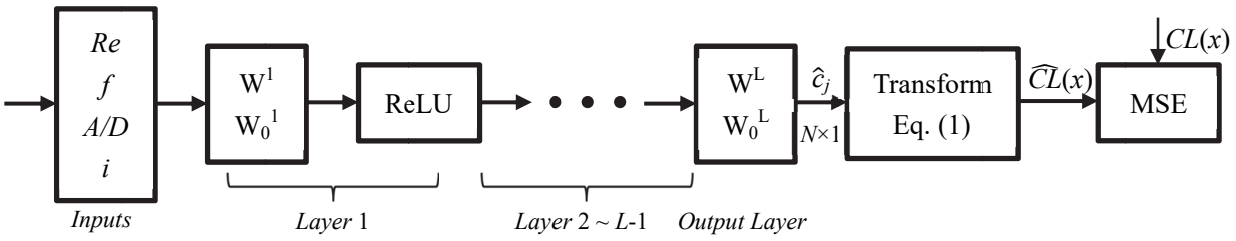


Fig. 1: Architecture of FCNN

## 3. Fully Connected Neural Networks (Keras)

A FCNN is trained in Keras, as illustrated in Fig. 1. The input dimension is 4, and output dimension is *N*. There are $N_h$ hidden layers with ReLU non-linearity, and each layer has $N_u$ units. The loss is a user-defined function as described in section 2. Full-batch (size = 63) is used in each update iteration.

A cross-validation function is implemented to tune *N*, $N_h$, $N_u$ and the choice of optimizer (from GD, Adadelta, Adam). The training and testing losses are evaluated after each epoch. The loss convergences of three example cases are illustrated in Fig. 2. I discovered that even after thousands of epochs, no obvious overfitting happened due to the relatively small training set. But to save time in tuning, I chose to run 100 epochs for each parameter, by which the losses have largely flattened in most cases (as seen in Fig. 2).

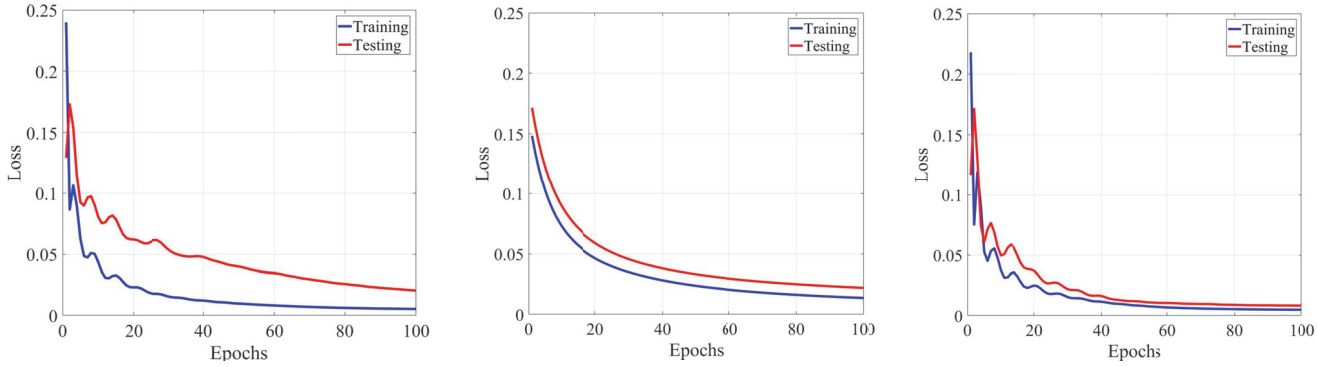


Fig. 2: Training and testing losses during training using GD (left), Adadelta (middle) and Adam (right) as optimizer

The tuning is carried out iteratively to find out the best parameter combinations. In each cross validation, the temporarily best parameters are applied and the tuning parameter is varied in a range to determine optima. The tuning iteration stops when no further modifications can be made.

Figure 3 shows average training and testing losses with $N$ ranging from 2 to 18. The losses decrease with $N$ as it's less than 10 but stabilize as $N$ grows even larger. So $N = 10$ is already good for approximating $CL(x)$.

Figure 4 shows losses with $N_h$ ranging from 1 to 9. Obviously, $N_h = 2$ has the best performance. The deeper neural nets, by closer examination, take more epochs to converge to a similar loss level. So there may be too many weights to update or too small a learning rate given this training set size for deep NN.

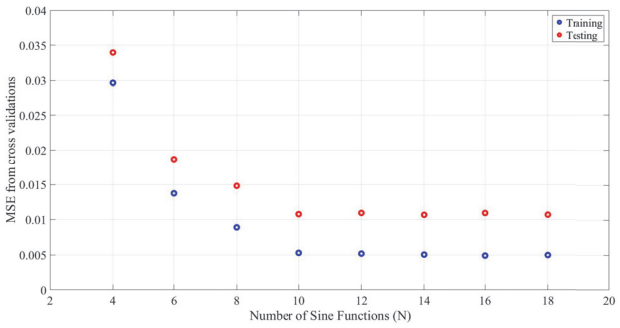


Fig. 3: Tuning number of sine functions *

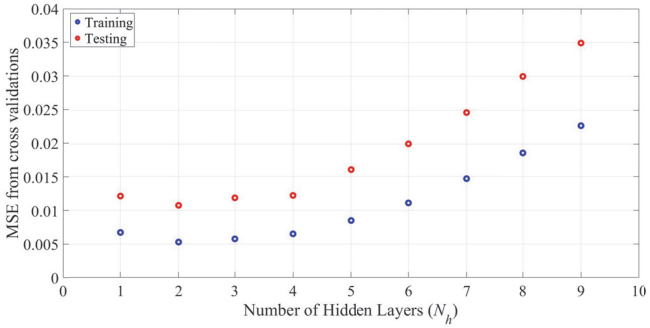


Fig. 4: Tuning number of hidden layers

Figure 5 shows losses with $N_u$ ranging from 100 to 800. There is a decreasing trend as $N_u$ is less than around 400~500 but stabilize for higher numbers. $N_u = 500$ is chosen at the end.

Figure 6 shows losses with different optimizers. The optimizer parameters are as follows: GD (learning rate = 0.01, decay = 1e-6, momentum = 0.9); Adadelta (learning rate = 0.1**, rho (decay rate) = 0.95) [1]; Adam (learning rate = 0.001, beta_1 (decay rate) = 0.9, beta_2 (decay rate) = 0.999)[2]. Obviously, the Adam optimizer has the best performance. As the case in Fig. 2, Adam converges faster and to a lower loss for most cases in this study.

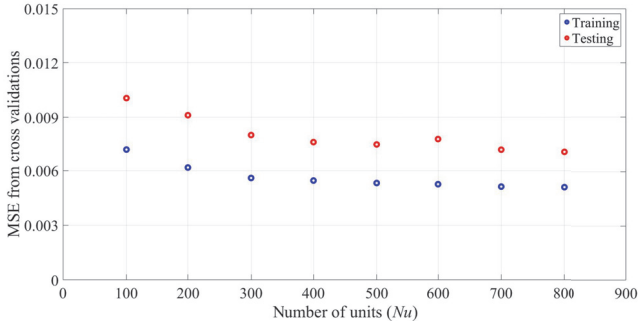


Fig. 5: Tuning number of units in each layer

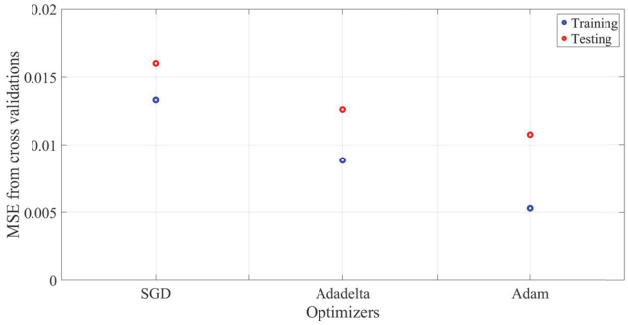


Fig. 6: Tuning choice of optimizers

* The points are average MSE losses calculated by 10-fold cross validations after training (100 epochs). Blue points are training losses after the last epoch. Red points are testing losses.

** Adadelta's learning rate is adjusted to smooth loss fluctuations during training. Since its average MSE losses from cross validations only slightly changed after the adjustment and the tuning result was not influenced, Fig. 6 from the progress report was not updated.

Therefore, the best parameters so far are $N = 10$, $N_h = 2$, $N_u = 500$ with Adam optimizer. To train a predictor, the cross validation with optimal parameters is carried out again but with 300 epochs. The average training and testing losses are listed in Tab. 1. The predictor with the lowest testing loss is selected at the end. The predictions it made for testing sets are illustrated in Fig. 9 (blue curves).

The errors are mainly structural errors. The "sum of sines" approximation cannot perfectly (even with large $N$) account for the shape of original data. For instance, $CL(0)$ and $CL(L)$ are not necessarily close to zero. Another consideration is that the predictions for mode 2,3,4 (closer to center of [1,5]) are better, possibly because the model sees more samples at the center and there may be a bias.

## 4. Gaussian Process Regression (MATLAB)

To also reveal confidence of predictions, GPR is implemented using `fitrgp` in MATLAB. The layout is illustrated in Fig. 7. Unlike FCNN, the GPR function doesn't support the above user-defined loss function or multiple outputs. Thus, $N$ GPR models are trained, $CL(x)$ is approximated (using `fit` with Eq. (1) as `fittype`), yielding $N$ coefficients, and they are fed into the models as ground truths[*].

For ease of parameter tuning and result comparing, the estimated coefficients $\hat{c}_j$ are mapped[**] to $\widehat{CL}(x)$ again and MSEs to $CL(x)$ are calculated. Figure 9 illustrates the tuning of $N$ by 10-fold cross validations. Similarly, the decrease trend largely vanishes at around 10~12. For comparison, $N$ is selected to be the same as in FCNN: 10. Other GPR parameters remain as default (fit method: exact; basis function: constant; kernel function: squared exponential) [3]. The average training and testing losses are listed in Tab. 1.

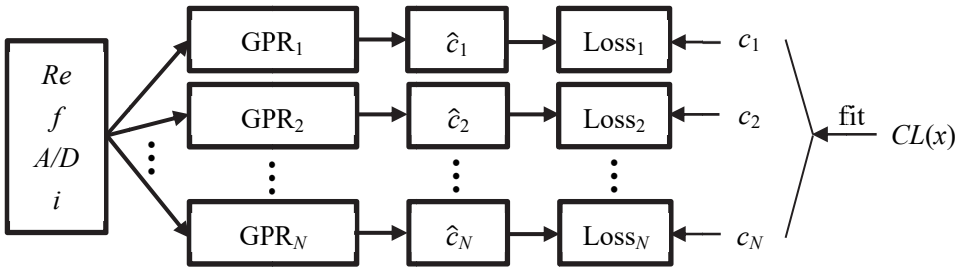


Fig. 7: Layout for Gaussian process regression model training

For comparison, the GPR predictor trained from the same fold as in FCNN is applied to make predictions in the same testing sets. Some results are illustrated in Fig. 9 (red curves) with 95% confidence intervals (grey areas). Likewise, the predictions are more accurate and confident for modes "in the center" (2~4), which implies a bias in training. The aforementioned structural error from "sum of sines" approximation is made clearer by the confidence intervals. The model is very confident about $\widehat{CL}(0)$ and $\widehat{CL}(L)$, but there are discrepancies.

Compared to FCNN, the GPR trained predictor has a slightly lower testing loss (as shown in Tab. 1). It also produces confidence intervals that reveal predictions' confidence level. In training, GPR takes a shorter computation time (within 1 min, ~ 5 times faster). Therefore, GPR is a relatively favorable method for this problem.

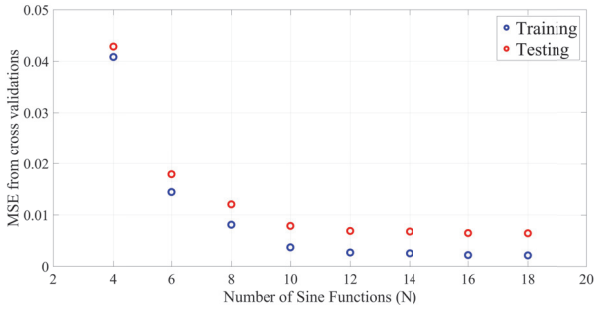


Fig. 8: Tuning number of sine functions in GPR

Tab. 1: Losses (MSE to $CL$) from cross validations

| 10-fold cross validation | FCNN[#] | GPR |
|---|---|---|
| Average training loss | 0.0036 | 0.0037 |
| Average testing loss | 0.0093 | 0.0079 |

[#] losses evaluated after training 300 epochs

[*] An assumption made here is that the output coefficients are independent to each other.
[**] The 95% confidence intervals are mapped in a similar way by Eq. (1) to be $x$-dependent.

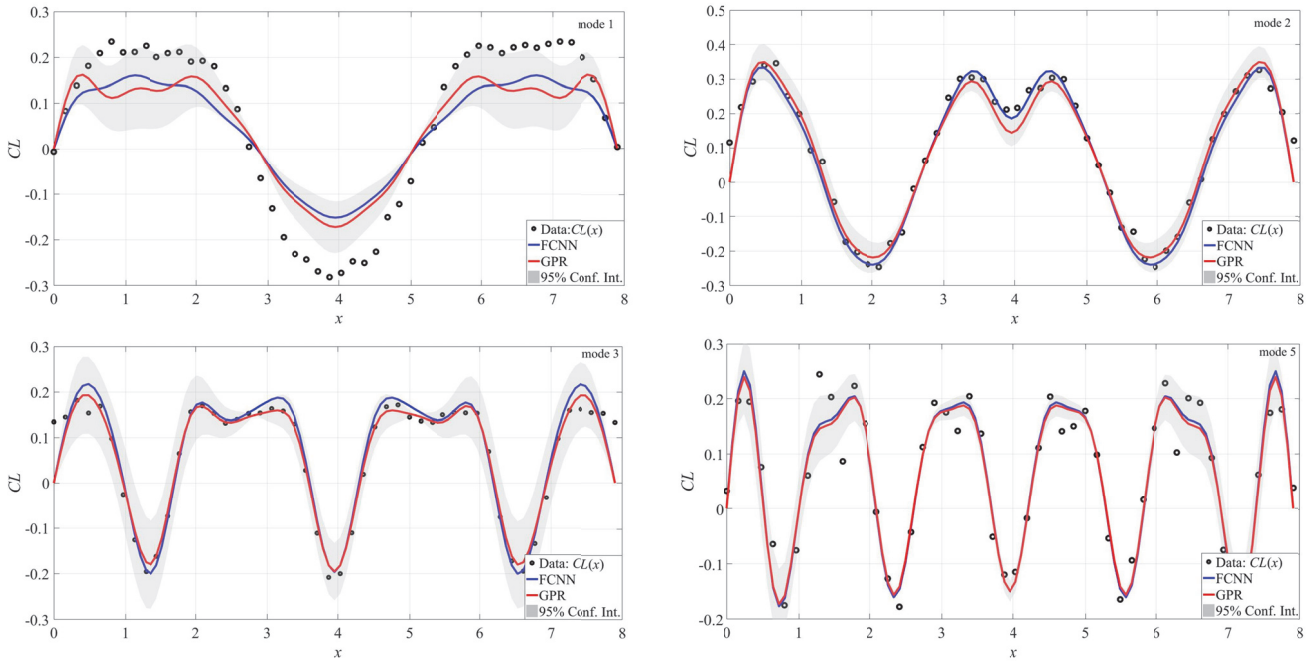


Fig. 9: Predictions by (optimal) FCNN and GPR predictors in test datasets (mode 1,2,3,5)

## 5. Generalizations

The above FCNN and GPR predictors are applied to predict $CL(x)$ in higher-mode ($i > 5$) and multi-mode cases*.

Figure 10 shows predictions in a mode 6 case. The FCNN prediction is 2~3 times higher than $CL$. The GPR predictor is more robust relatively, but it's much less confident due to the unexpected mode number.

In multi-mode VIV, as the component of a mode is extracted using band-pass filtering, other modes still have influences or "coupling effects" on that mode. In a mode 4, 5 coupling case, predictions are made to the filtered-out mode 5 component, as shown in Fig. 11. The FCNN and GPR predictions show similar performances (MSE = 0.0062 vs 0.0080). GPR predictor has more confidence because it sees $i = 5$ a lot in training. But obviously this is "blind confidence", in that it can't consider coupling effects without training on multi-mode cases.

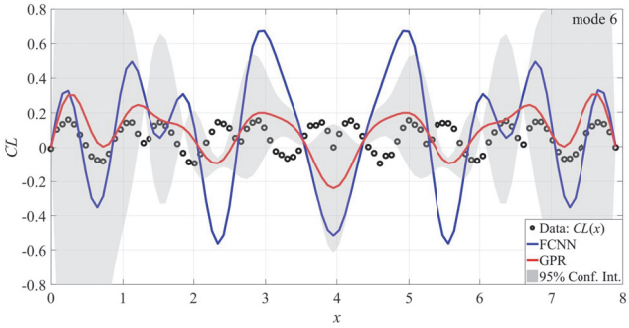


Fig. 10: $CL$ predictions in a mode 6 case

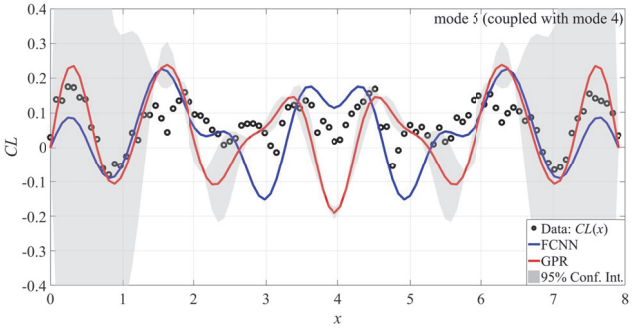


Fig. 11: mode 5 predictions with coupling effect from mode 4

## 6. Conclusions

Supervised learnings over 74 datasets (4 leave-outs) are performed to train a VIV predictor mapping parameters: $Re$, $f$, $A/D$, $i$ to lift coefficient distribution $CL(x)$. A FCNN predictor is firstly trained with no obvious overfitting observed due to the small training set. The number of sine functions, hidden layers and units per layer $N$, $N_h$, $N_u$, together with choice of optimizer are tuned iteratively by 10-fold cross-validations. The best parameter combination found is (10, 2, 500, Adam). Subsequently, a GPR predictor is trained and $N$ tuning yields similar results.

As shown in Fig. 9, the prediction errors are mainly structural errors from the "sum of sines" approximation. And better performances for mode 2~4 also imply a bias in training. The FCNN and GPR predictions have same-level performances in testing sets. But considering runtime and productions of confidence levels, GPR should be a more favorable choice in this problem. In higher-mode cases (as Fig. 10), the predictors fail to produce reliable predictions. In multi-mode cases (as Fig. 11), the predictions are more acceptable but GPR predictor is "overconfident". In general, the models entail extra training sets to perform more robustly.

* Before feeding inputs into predictors, $Re$, $f$ and $A/D$ are standardized by means and STDs of the training-set inputs.

## Nomenclature

| | |
|---|---|
| $x$ | position along riser length |
| $L$ | riser length |
| $D$ | riser diameter |
| $Re$ | Reynolds number |
| $f$ | non-dimensional frequency |
| $A/D$ | non-dimensional amplitude |
| $i$ | mode number |
| $CL(x)$ | lift coefficient distribution |
| $c_j$ | fitted approximation coefficients for $CL(x)$ |
| $N$ | number of sine functions to approximate $CL(x)$ |
| $N_h$ | number of hidden layers in FCNN |
| $N_u$ | number of units per layer in FCNN |
| $\hat{c}_j$ | predicted approximation coefficients |
| $\widehat{CL}(x)$ | predicted lift coefficient distribution |

## References

[1] Matthew D. Zeiler, 2012. Adadelta: An Adaptive Learning Rate Method. arXiv:1212.5701v1.

[2] Diederik P. Kingma, Jimmy L. Ba, 2017. Adam: A Method For Stochastic Optimization. arXiv:1412.6980v9.

[3] Carl E. Rasmussen, Christopher K. I. Williams, 2006. Gaussian Processes for Machine Learning. MIT Press. Cambridge, Massachusetts.