# ML 2023 Project

Authors: Ludovico Iannello, Federico Fattorini.

Team name: NikolaTesla

Master degree: Physics; Curriculum: Complex Systems

Emails: l.iannello@studenti.unipi.it, f.fattorini4@studenti.unipi.it.

Date 15/01/2024

Type of project: **B**

# Objectives

The aim is to compare 4 different models (K-nn, SVM, MLP from 2 different libraries with SGD training algorithm) in order to assess which one best performs on a regression task with pattern of input dimension 10 and target of dimension 3.

The correct functioning of each model is assessed on the 3 MONK dataset, for which state of art result were obtained.

Finally, the best model among the ones tried in the regression task is chosen with a classical model selection, and the best model's performance is estimated using an internal set. In the regression task, plots regarding the functioning of each model are provided to give more insights.

# Methods and models

The used libraries are numpy, matplotlib, tensorflow [3] , Keras [2], pandas, Scikit-learn [1], scikeras.

The 4 model implemented are:

- **K-nn** from the Scikit-learn library (*KneighborsClassifier* and *KneighborsRegressor*): the tool allows to change the number of neighbors, the metrics and if the neighbors are weighted with the distance.
- **SVM** from the Scikit-learn library (*SVC* and *SVR*): the tool allows to change the kind of kernel with the corresponding parameters of SVM, the regularization parameter and the epsilon tube in the regression task.
- **MLP** from the Keras and scikeras library (*KerasClassifier* and *KerasRegressor*) using SGD algorithm, 1 or 2 layer varying the number of units, and varying other hyperparameters (see next slides).
- **MLP** from the Scikit-learn library (*MLPClassifier* and *MLPRegressor*) with the same features of Keras

In all cases the models selection and the grid search has been performed with Scikit-learn tool *GridSearchCV*.

# Methods and models: MLP

For both MLP models the activation functions tried were ReLU and Tanh, the batch size was varied trying full batch, online, mini-batch. The weights were initialised randomly and the regularization schema used was Tikhonov regularization.

**Keras MLP vs Scikit-learn MLP**

- In Keras the loss to be minimized can be changed, while in scikit-learn is fixed by the library in both the classification and the regression task.
- In Keras the learning rate is changed with an exponential decay, while in Scikit-learn the adaptive learning rate used means that the learning rate is divided by 5 once if the loss does not decrease in 2 iterations. Still in Scikit-learn, invscaling learning rate gradually decreases the learning rate at each epoch using an inverse scaling exponent of *power_t*
- In Scikit-learn the stopping condition is automatically implemented, blocking the training when the training loss does not decrease more than a tolerance (*tol* parameter) in *n_iter_no_change* epochs, while in Keras the number of epochs was manually fixed (tuning them as others hyperparameters and by the use of learning curves).
- In Keras the random initialisation was varied between uniform and gaussian, while in Scikit-learn is fixed as uniform.

# Loss functions for the MLP

For the classification task the loss function used for both sklearn MLP and keras MLP was the log loss function:

$$logloss = -\frac{1}{l}\sum_{p=1}^{l} t_p \log(P(o_p = 1 \mid x_p)) + (1 - t_p)\log(1 - P(o_p = 1 \mid x_p))$$

where $x_p$ is the pattern p, t corresponding target, *l* is the total number of patterns and $P(o_p = 1|x_p)$ is the probability that the prediction for $x_p$belongs to the positive class.

For the regression task the loss function used for the training was the mean squared error (for both MLPs), and the results (validation error and training error) were computed using the mean euclidean error:

$$E_{MSE} = \frac{1}{l}\sum_{p=1}^{l} (\boldsymbol{o_p} - \boldsymbol{t_p})^2 \qquad E_{MEE} = \frac{1}{l}\sum_{p=1}^{l} \|\boldsymbol{o_p} - \boldsymbol{t_p}\|_2$$

# Methods and Models: Validation schema

The dataset was divided in Training+Validation set (80%) and Test set (20%) using an Hold Out schema for model assessment (only for the Cup, for the monk tasks the dataset was already splitted into TR/VAL and TS).

The models are chosen using a 5-fold CV (stratified CV in the MONKs case) on the Training+Validation set, repeated 5 times. The results were expressed with the mean and std among the repetitions.

An unique grid-search was used to select the best parameters, first trying a wide range of parameters, than varying between the best which found  with the first approach.

In the case of SVM, the three different kernels were optimized with 3 separated grid-search, for efficiency reasons and to better tune the parameter for each kernel.

In both the MLP models, the neural network with 1 layer and the one with 2 were optimized separately, again with a grid-search.

Finally, once the model was selected, a final retraining was performed , and the performance were estimated on the test set.

# Monk Results K-nn

A model selection with a 5-fold stratified approach (repeated 5 times) was performed among the following choices/hyperparameters, then the best model is refitted on the design set:

- If the encoding for the category was One Hot or not
- Possible metrics: L1, L2, Cosine Similarity, Minkowski with p=3, 5, 7
- Number of neighbors K: integer in range [1,50]
- If the contributions of the neighbors were weighted with the distance

| **Task** | **Parameters** | **Accuracy ± Std (TR/VAL/TS)** |
|---|---|---|
| MONK 1 | Not Encoded, metric=Cosine similarity, K=6, weighted distance | 1 ± 0 / 0.84 ± 0.07 / 0.84 |
| MONK 2 | Not Encoded, metric=Cosine similarity, K=1, uniform distance | 1 ± 0 / 0.80 ± 0.06 / 0.89 |
| MONK 3 | Encoded, metric=L1, K=47, weighted distance | 1 ± 0 / 0.92 ± 0.05 / 0.95 |

# Monk Results SVM

A model selection with a 5-fold stratified approach (repeated 5 times) was performed among the following choices/hyperparameters, then the best model is refitted on the design set:

- If the encoding for the category was One Hot or not
- Type of kernels among Polynomial $k(\boldsymbol{x}, \boldsymbol{x_i}) = (\gamma\langle \boldsymbol{x}, \boldsymbol{x_i}\rangle + r)^p$ , RBF $k(\boldsymbol{x},\boldsymbol{x_i}) = e^{-\frac{1}{2\sigma^2}\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}$

  Sigmoid $k(\boldsymbol{x}, \boldsymbol{x_i}) = \tanh\left(\beta_0 \left\langle \boldsymbol{x}, \boldsymbol{x_i} \right\rangle + \beta_1\right)$

- C as a regularization hyperparameter

| **Task** | **Parameters** | **Accuracy ± Std (TR/VAL/TS)** |
|---|---|---|
| MONK 1 | Encoded, Polynomial, C=0.001, p=2, r=-10, $\gamma$ = 63 | 1 ± 0 / 1 ± 0 / 1 |
| MONK 2 | Encoded, Polynomial, C=10, p=2, r=-5, $\gamma$ = 4 | 1 ± 0 / 0.93 ± 0.08 / 1 |
| MONK 3 | Encoded, Polynomial, C=0.06, p=3, r=-133, $\gamma$ = 0.04 | 0.93 ± 0.02 / 0.94 ± 0.05 / 0.97 |

# Monk Results MLP Scikit-learn

One hot encoding was used for the data. The MLP had only one hidden layer, with ReLU or Tanh as possible activation functions.

The algorithm was SGD using the log-loss function. No early stopping was used.

A model selection with a 5-fold stratified approach (repeated 5 time) was performed among the choices/hyperparameters, then the best model is refitted on the design set.

| **Task** | **Parameters: units, activation function, λ, batch size, learning rate, η, epochs, α, Nesterov** | **Accuracy ± Std (TR/VAL/TS)** |
|---|---|---|
| MONK 1 | 5, reLU, 0.0001, 5, adaptive, 0.05, 500, 0, no Nesterov | 1 ± 0 / 1 ± 0 / 1 |
| MONK 2 | 3, reLU, 0.0001, 10, adaptive, 0.05, 500, 0.1 , Nesterov | 1 ± 0 / 1 ± 0 / 1 |
| MONK 3 | 3, reLU, 0.0001, batch, adaptive, 0.01, 1500, 0.1, Nesterov | 0.93 ± 0.01 / 0.93 ± 0.05 / 0.97 |

# Monk 1: learning curves for best Scikit-learn MLP

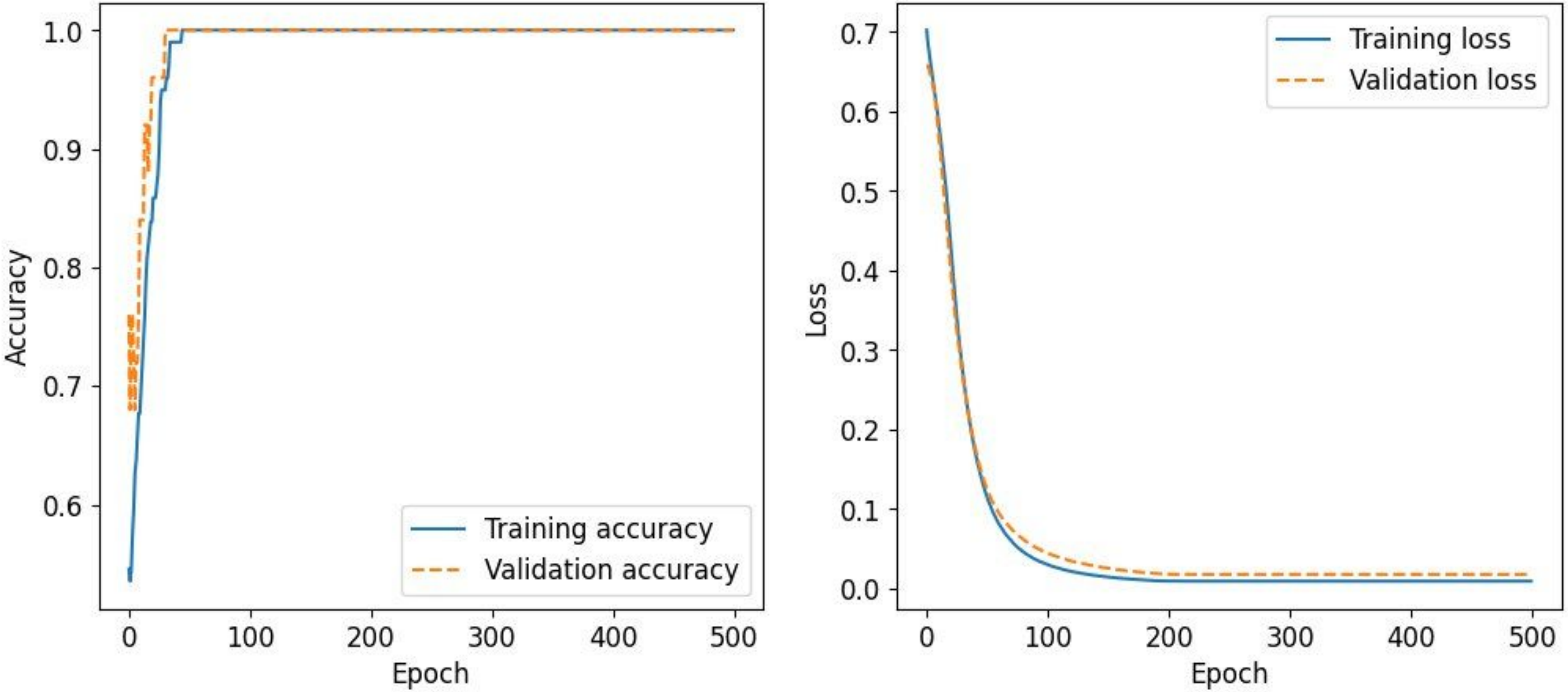

Figure 1: Accuracy and log-loss function as training epochs increase, for training and validation set

# Monk 2: learning curves for best Scikit-learn MLP

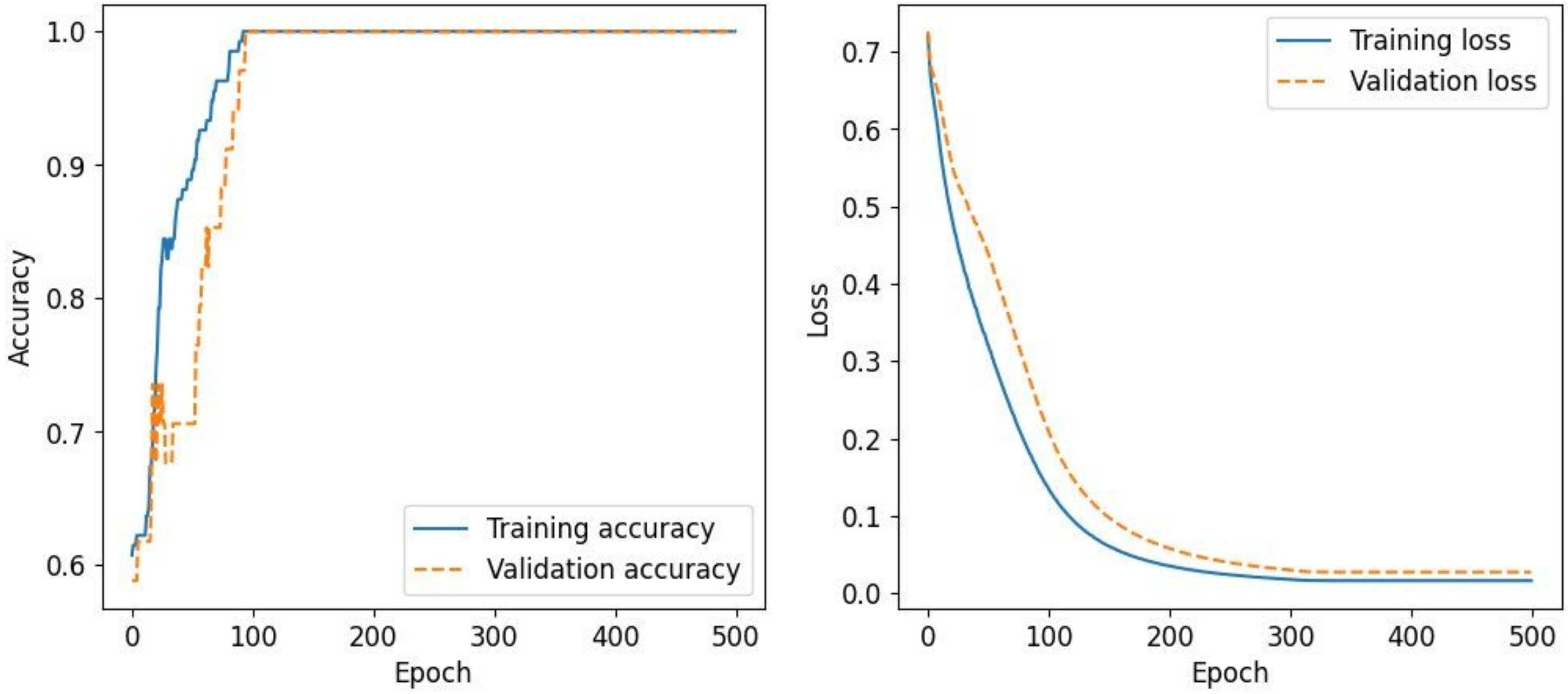

Figure 2: Accuracy and log-loss function as training epochs increase, for training and validation set

# Monk 3: learning curves for best Scikit-learn MLP

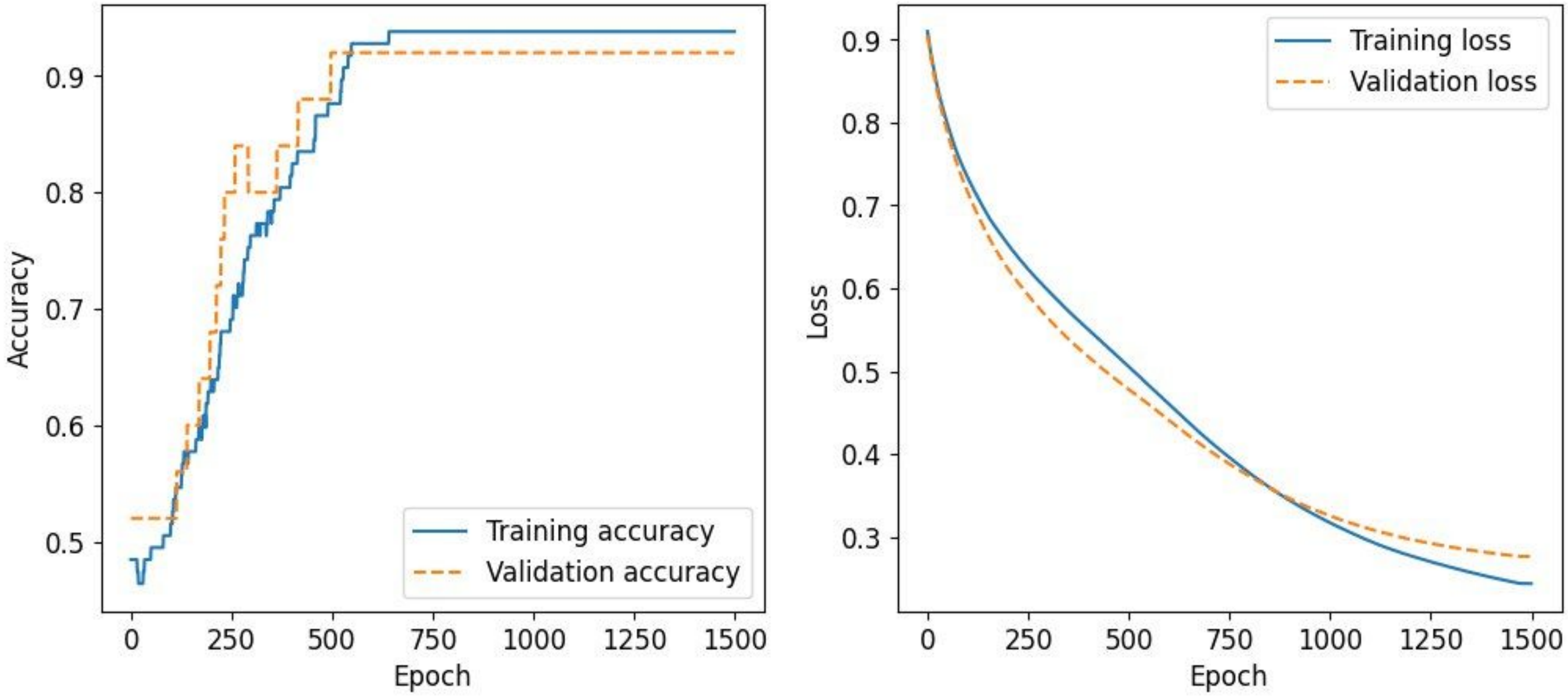

Figure 3: Accuracy and log-loss function as training epochs increase, for training and validation set

# Monk Results MLP keras

One hot encoding was used for the data. The MLP had only one hidden layer, with ReLU or Tanh as possible activation functions.

The algorithm was SGD using the logloss function. No early stopping was used.

A model selection with a 5-fold stratified approach (repeated 5 times) was performed among the choices/hyperparameters, then the best model is refitted on the design set

| **Task** | **Parameters: units, activation function, λ, batch size, η, epochs, α, Nesterov** | **Accuracy ± Std (TR/VAL/TS)** |
|---|---|---|
| MONK 1 | 3, tanh, 0.01, 100, 0.1, 1500, 0.8, Nesterov | 1 ± 0 / 0.998 ± 0.007 / 1 |
| MONK 2 | 5, reLU, 0.001, 50, 0.05, 600, 0.4 , no Nesterov | 1 ± 0 / 1 ± 0 / 1 |
| MONK 3 | 5, reLU, 0.1, 100,  0.05, 300, 0.7, no Nesterov | 0.93 ± 0.01 / 0.93 ± 0.05 / 0.97 |

# CUP Results: K-nn

The K-nn regressor was implemented with the model KNeighborsRegressor of Scikit-learn, using a unique model to compute the 3 outputs.

The hyperparameters varied are:

- The metric: among L1, L2, Cosine similarity and Minkowski with p>2  (p<10 were tried)
- The number of neighbors K from 1 to 40
- If the neighbors are weighted with the distance or not

Effects of each hyperparameter were studied also with the plots showed in the next slide.

The best configuration found was:

| | | |
|---|---|---|
| K=6 | metric=Cosine similarity | weighted distance |

| Training error | Validation error |
|---|---|
| $(8.8 \pm 0.3)\cdot 10^{-14}$ | $2.7 \pm 0.2$ |

# CUP Results: K-nn

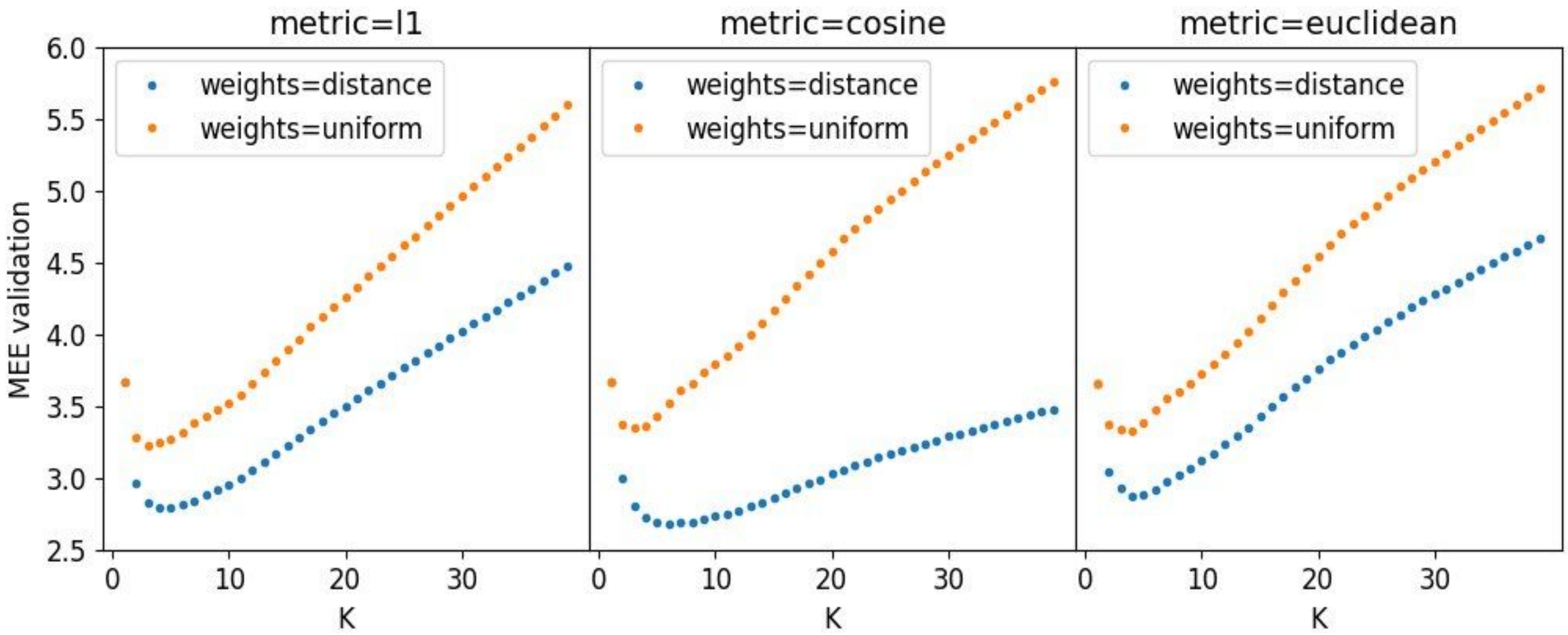

Figure 4: MEE error on the validation set, for different number of neighbors, three metrics and with 2 different weighting of the distance

# CUP Results: SVM

The SVM regressor was implemented with the model SVR of Scikit-learn, using the class MultiOutputRegressor to fit 1 regressor (SVM) per target and extend the SVM model to compute the 3 outputs.

The hyperparameters C, $\gamma$, $\beta_0$ were varied logarithmically in the range [$10^{-5}$,$10^{3}$], the epsilon-tube parameter was varied in the range [0.1,0.7], the degree of the polynomial kernel was varied in the range [1,10], the coefficient r in the range [-100,100] and $\beta_1$ in [-10,0). For RBF and sigmoid, the grid search was supported with the plots showed in the next slides.

The best configurations found varying the type of kernels are the following:

| **Kernel** | **Parameters** | **MEE± Std (TR/VAL)** |
|---|---|---|
| RBF | $C = 10^4$, $\varepsilon = 0.1$, $\gamma = 1/(2\sigma^2) = 0.2$ | 0.170 ± 0.006 / 0.55 ± 0.05 |
| Polynomial | $C = 10^{-5}$, $\varepsilon = 0.1$, $p = 8$, $r = 6.5$, $\gamma = 0.88$ | 0.158 ± 0.001 / 0.53 ± 0.04 |
| **Sigmoid** | $C = 5\cdot10^4$, $\varepsilon = 0.1$, $\beta_0 = 0.3$, $\beta_1 = -4$ | 0.162 ± 0.001 / 0.51 ± 0.04 |

# CUP Results: SVM with gaussian kernel

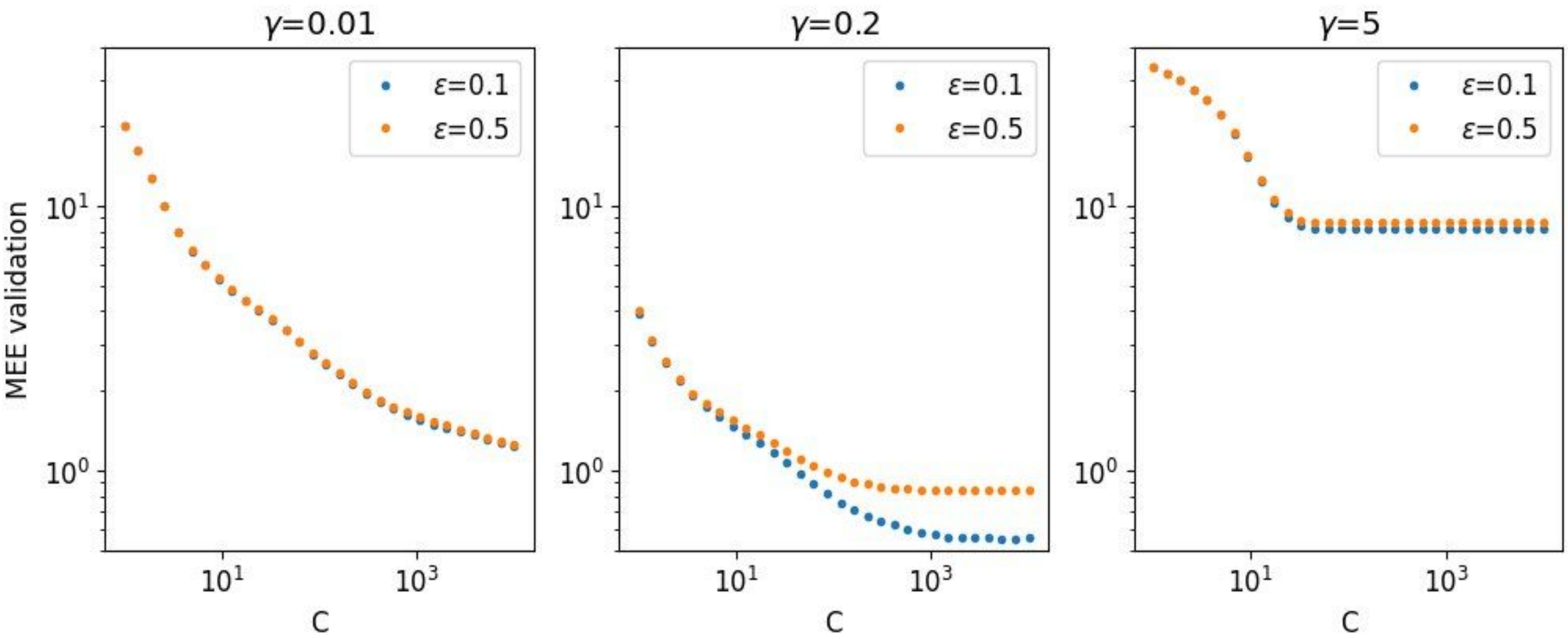

Figure 5: MEE error on the validation set varying C, for 3 different $\gamma$, and with 2 epsilon-tube parameters.

# CUP Results: SVM with sigmoidal kernel

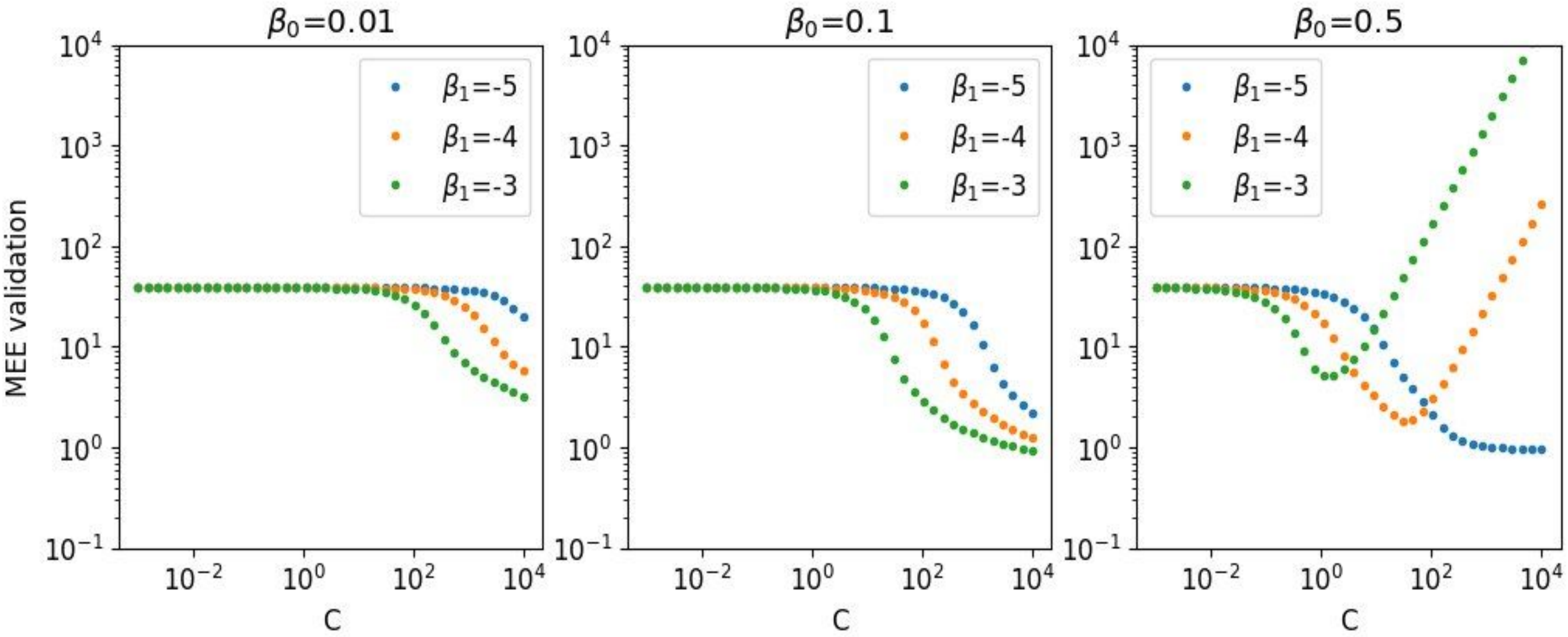

Figure 6: MEE error on the validation set varying C, for 3 different $\beta_0$ , and with 3 $\beta_1$

# CUP Results: MLP Hyperparameters

1 and 2 hidden layers were tried as architectures, tuning the hyperparameters separately:

- The number of units in each layer was increased from 10/15 units up to 100
- The initial learning rate was modified in the range [0.001,0.5]. For the Keras model, the learning rate decreased by a factor $d_r$ in range [0.5,0.95] in $d_{step}$ steps in range [100,1000] according to:

$$\eta_i = \eta_0 \, d_r^{\frac{1}{d_{step}}}$$

- The weight decay was varied in [$10^{-9}$,$10^{-1}$]
- The momentum $\alpha$ was varied (with and without Nesterov) in [0,1]
- Batch size was varied from 1 to 500
- The number of epochs were tuned from 500 up to 2000, depending on the other hyperparameters
- Early stopping was tried (using a fraction of 0.2 of the original training set), but validation results were worse than without it

# CUP Results: MLP scikit-learn and MLP Keras

**MLP scikit-learn results.**

| **N. hidden layers** | **Parameters: units, activation function, λ, batch size, learning rate, η, epochs, α, Nesterov** | **MEE ± Std (TR/VAL)** |
|---|---|---|
| 1 | 100, tanh, 0.001, 5, adaptive, 0.005, 1000, 0.7, no | 0.35 ± 0.02 / 0.67 ± 0.05 |
| 2 | 100, tanh, 0.0001, 10, invscaling (power_t=0.05), 0.005, 1200, 0.4, no | 0.31 ± 0.03 / 0.61 ± 0.05 |

**MLP Keras results.**

| **N. hidden layers** | **Parameters: units, activation function, λ, batch size, $\eta_0$, $d_r$ , $d_{step}$ , epochs, α, Nesterov** | **MEE ± Std (TR/VAL)** |
|---|---|---|
| 1 | 100, tanh, 0.001, 20, 0.02, 0.9 , 1000, 1500, 0.5, no | 0.53 ± 0.01 / 0.78 ± 0.05 |
| 2 | 110, tanh, 0.001, 200, 0.01, 0.9 , 500, 3000, 0.8 , yes | 0.30 ± 0.01 / 0.65 ± 0.05 |

# CUP Results: MLP learning curves

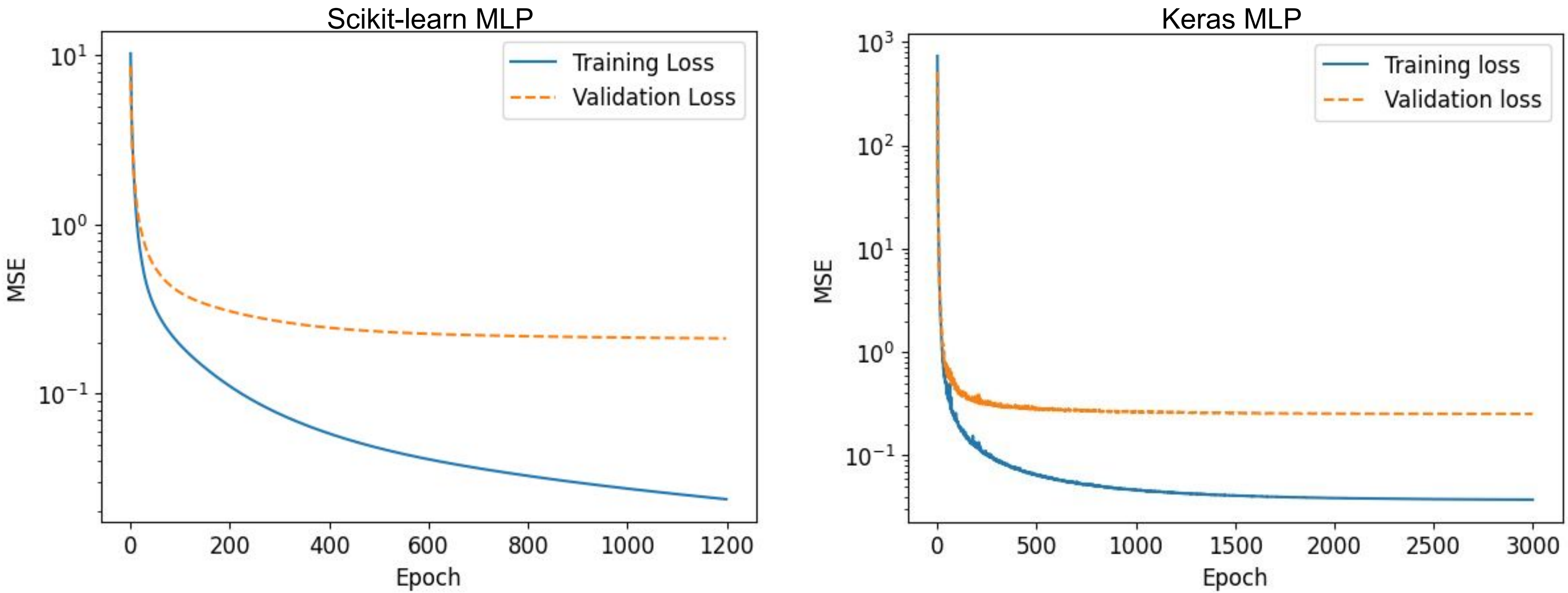

Figure 6: MSE error on the training and validation set, for the best Scikit-learn and Keras 2-layers MLP

# CUP: model selection and model assessment

The best model was selected among the best configurations of parameters obtained with the grid search for each kind of model (K-nn, SVM, MLP from scikit and MLP from Keras), choosing the one with the lowest MEE on the validation set.

The best model is SVR

| **Kernel** | **Parameters** | **MEE± Std (TR/VAL)** |
|---|---|---|
| **Sigmoid** | $C = 5 \cdot 10^4$, $\varepsilon = 0.1$, $\beta_0 = 0.3$, $\beta_1 = -4$ | 0.162 ± 0.001 / 0.51 ± 0.04 |

After a retraining on the design set, the MEE test score is 0.49.

The model performances were evaluated on the following device:

Lenovo IdeaPad Flex Flex5, AMD Ryzen 4000 series 7, 8 GB RAM, (with Ubuntu 22.04),

with training time 23 s.

# Conclusions

The model selection procedure on the validation set showed that the best model was a SVR with sigmoid kernel, with test set MEE 0.49.

For the particular set of data both Keras and Scikit-learn MLP performed similarly, with better results for Scikit-learn. In both cases, 2 layers and a large number of units (around 100) were needed.

K-nn results were the worst out of all the models.

The blind Test results were provided after a final retraining of the SVR model on the whole dataset.

Name of the blind test result file: "*NikolaTesla_ML-CUP23-TS.csv*"

Nickname of the team: "*NikolaTesla*".

# Bibliography

1. Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830. Retrieved from https://scikit-learn.org
2. Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras
3. Abadi, Martin, Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … others. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation ($OSDI$ 16)* (pp. 265–283). Retrieved from https://www.tensorflow.org/?hl=it
4. Garcia Badaracco. (2020). SciKeras. GitHub. Retrieved from https://github.com/adriangb/scikeras

# Appendix

In the following supplementary slides the learning curves on the 3 MONK dataset for the 3 best Keras MLP models are showed.

# Monk 1: best MLP keras learning curve

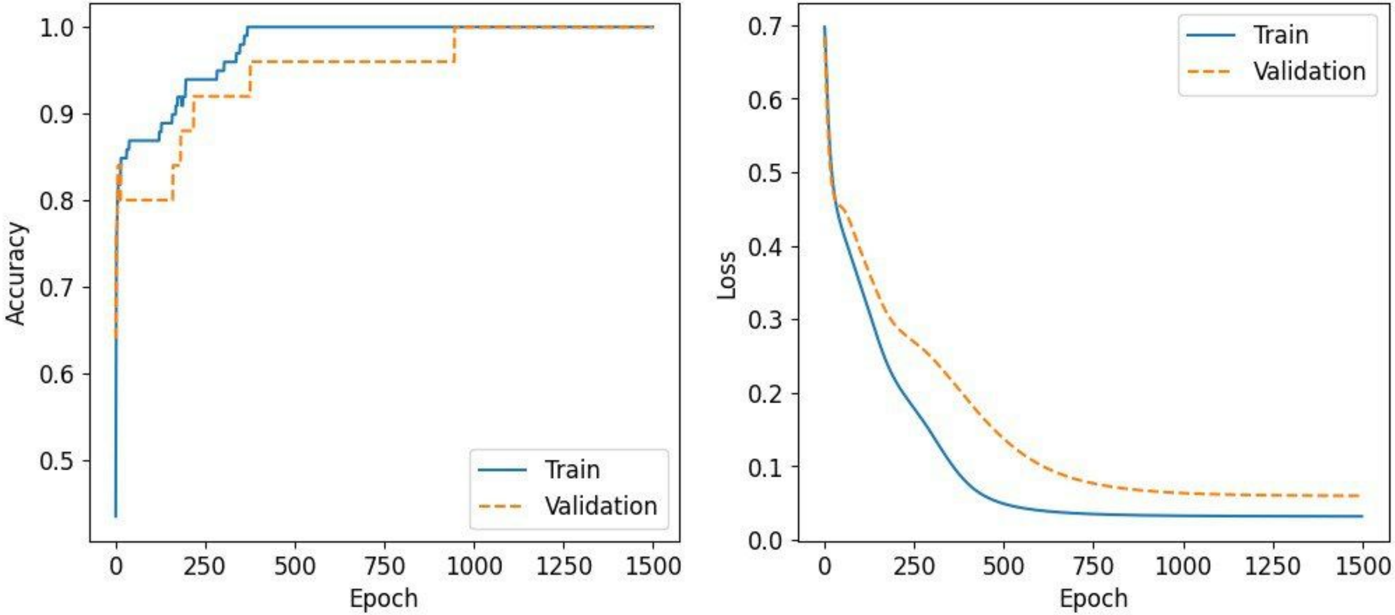

Figure 7: Accuracy and logloss as training epochs increase, for training and validation set

# Monk 2 MLP: best keras learning curve

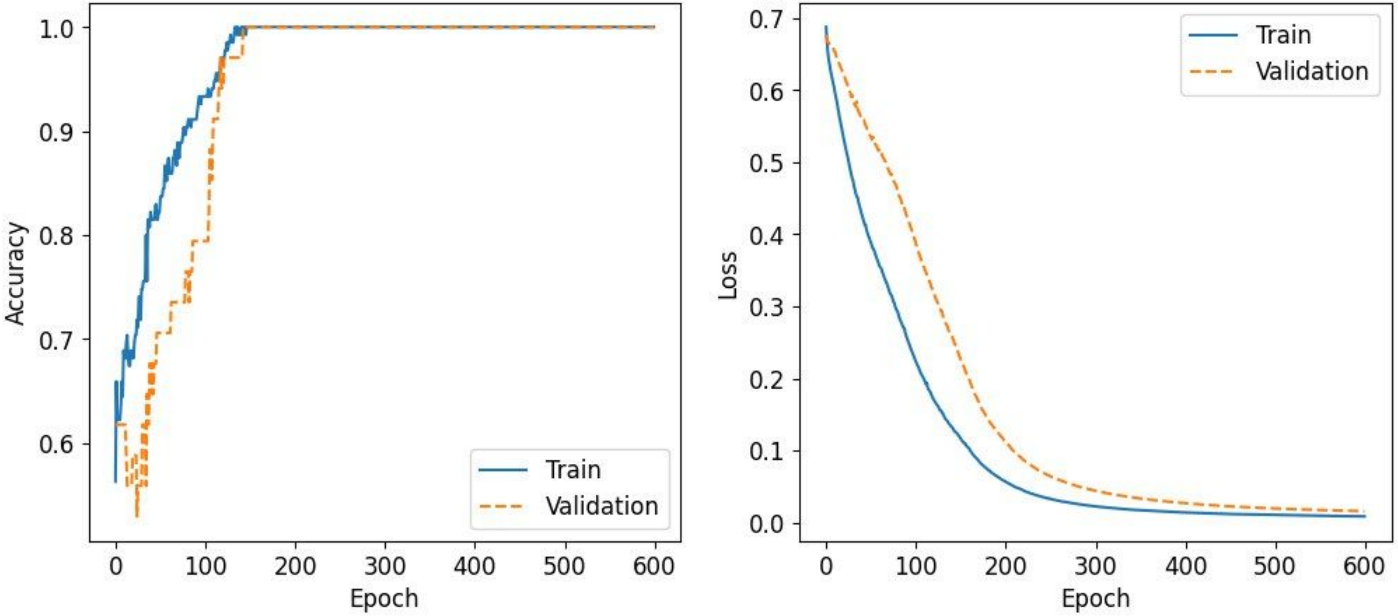

Figure 8: Accuracy and logloss as training epochs increase, for training and validation set

# Monk 3 MLP: best keras learning curve

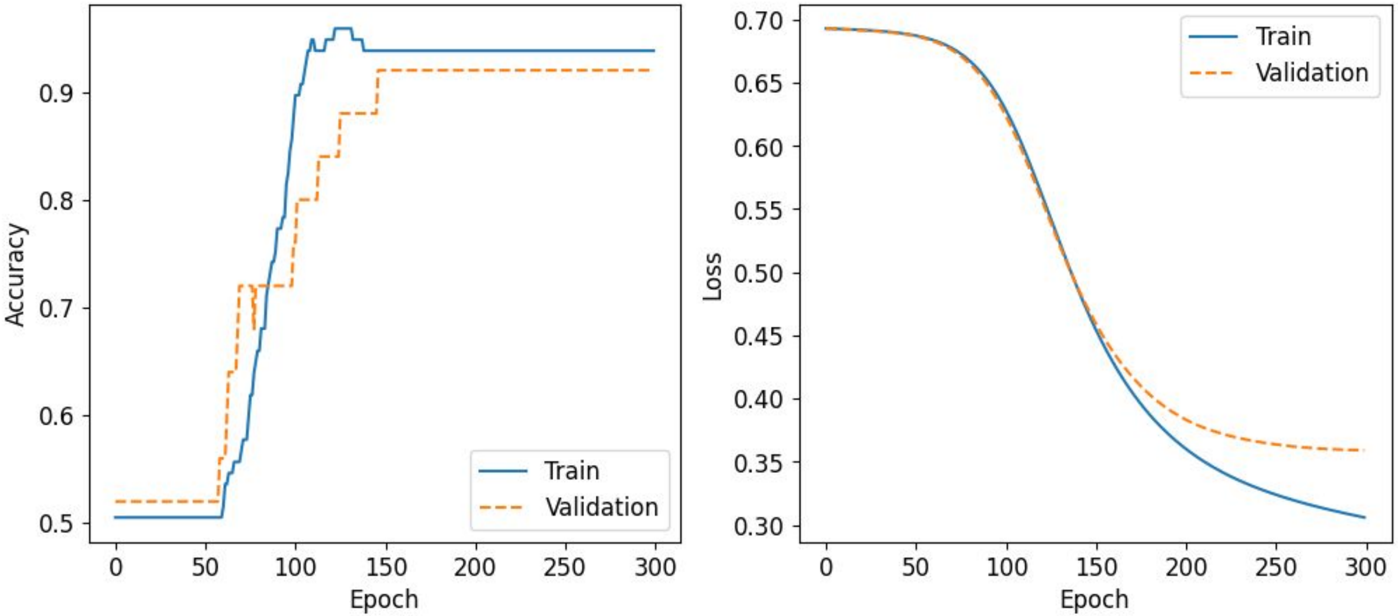

Figure 9: Accuracy and logloss as training epochs increase, for training and validation set