5. NEURAL NETWORKS MONITORING

Stephan Robert-Nicoud
HEIG-VD/HES-SO

Credit: Andres Perez-Uribe

Objectives

- ❑ Understand the effect of hyper-parameters in a model
- ❑ Understand how to monitor the training of a neural network during the hyper-parameter tuning process
- ❑ Apply the "generic" Machine-Learning model selection methodology to neural networks using training, validation, and test datasets

Data-driven modeling methodology

- What sort of data do I have ? is it noisy ? do I have missing data ? outliers ? redundant variables ?
  - Exploratory analysis of data
  - variable selection, noise reduction, outlier filtering
- Do I have variables whose values are in very different ranges ?
  - Data normalization
- Do I have an unbalanced dataset (number of examples per class) ?

- Create a train-validation-test partition of the dataset
- Baseline of neural network performance
    - identify number of epochs and learning rate using a first ANN configuration
- Neural network model selection
    - cross-validation evaluation using different hyper-parameters (e.g., number of hidden neurons)
- Compute final performances using the test dataset

Data normalization (1)

Data normalization is needed if variables have different ranges. The simplest way to deal with this problem is to re-scale inputs using:

$$\boxed{x' = \frac{(x - x_{min})}{(x_{max} - x_{min})}}$$

Rescaled into [0,1] range

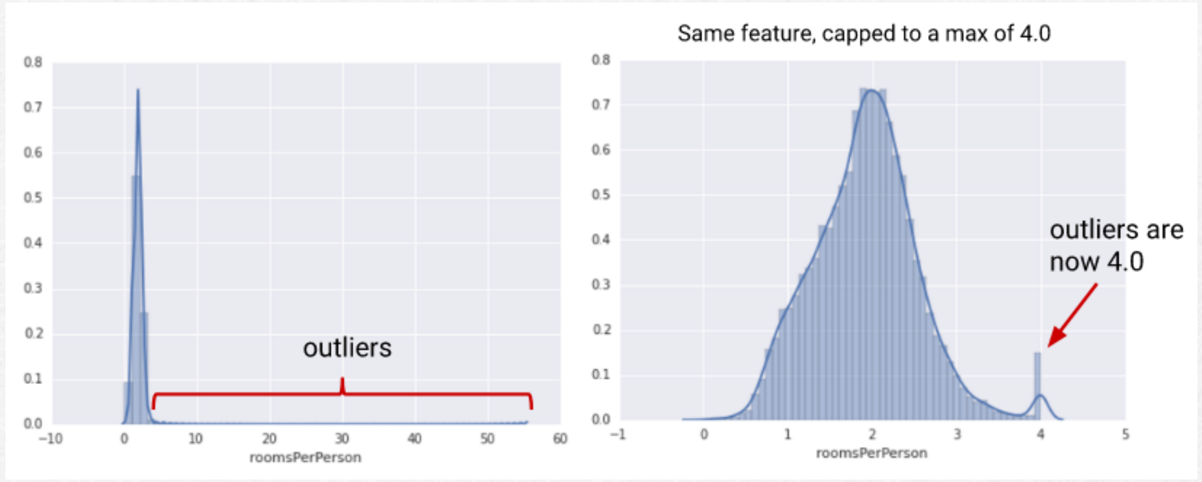
$$\boxed{x' = 2\frac{(x - x_{min})}{(x_{max} - x_{min})} - 1}$$

Rescaled into [-1,1] range

But, we should only use the training dataset to compute the minimum and maximum values.

Data normalization (2)

MinMax scaler faces a problem when the data contains outliers:



We have to first filter-out outliers or if there are not too many outliers use a z-normalization:

$$\boxed{x' = \frac{(x - \mu)}{\sigma}}$$

where μ is the mean and σ the st-dev

from https://developers.google.com/machine-learning/data-prep/transform/normalization

Weight initialization

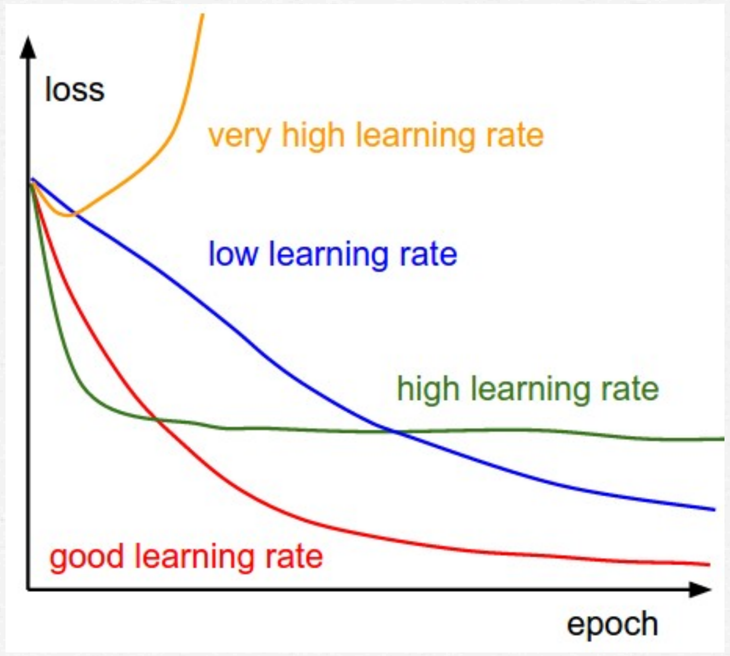
- zero initialization: this places us somewhere in the loss error function we want to minimize. If every time we start in the same place, we may always face the same difficulties to get to the minimum. Moreover, if all weights are equal, all neurons in the first layer will compute the same thing… in conclusion it is a bad idea.

- small random numbers: it has been proposed that weights should be initialized to small random values scaled by a function of the number of inputs (n), as follows:

  w = np.random.randn(n) * sqrt(2.0/n)

Effect of learning rate

With low learning rates (blue line) the improvements look linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line).

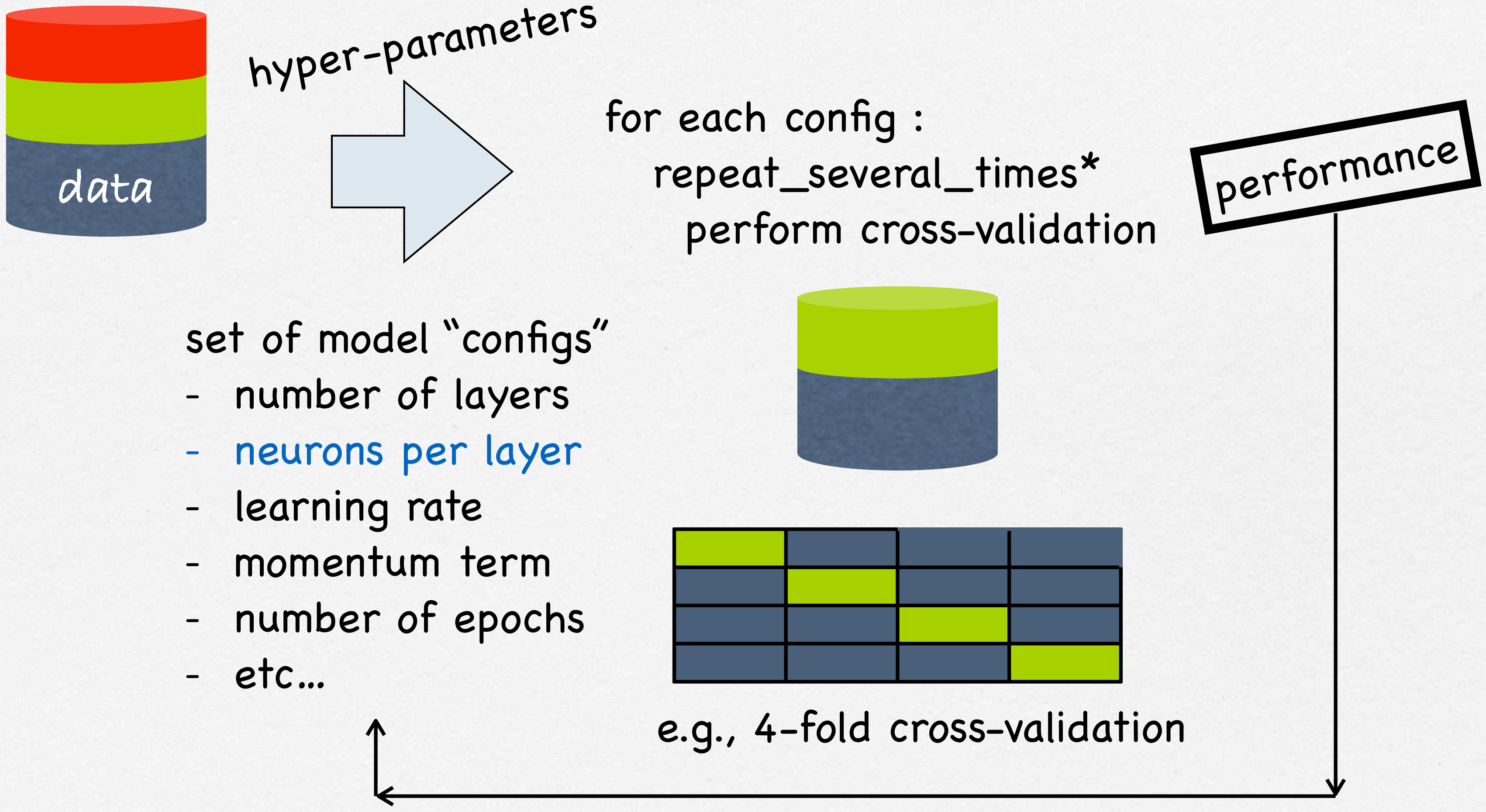


from https://cs231n.github.io/neural-networks-3/

Hyper-parameter tuning

- Rule of thumb: try several log-spaced values (0.1, 0.01, 0.001) and narrow the grid search to the region where you obtain the lowest validation error.
- For some parameters use the *3 strategy: e.g., number of hidden neurons: 1, 3, ~10, 30 …
- Scikit-learn implements GridSearchCV and RandomizedSearchCV
- Keras implements KerasTuner for hyperparameter tuning
- AutoML: the process of automating the development of a ML solution

Model selection



```
for each config :
    repeat_several_times*
        perform cross-validation
```

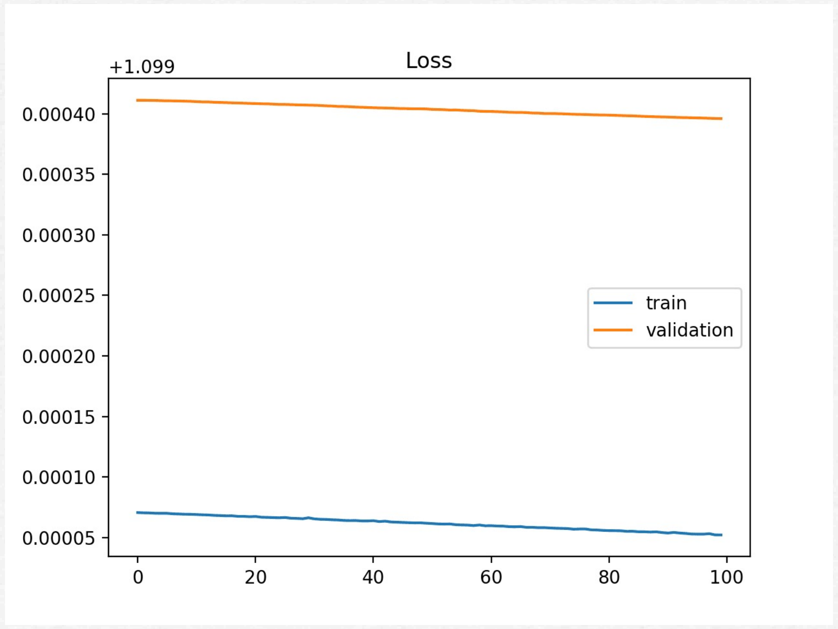
set of model "configs"
- number of layers
- neurons per layer
- learning rate
- momentum term
- number of epochs
- etc...

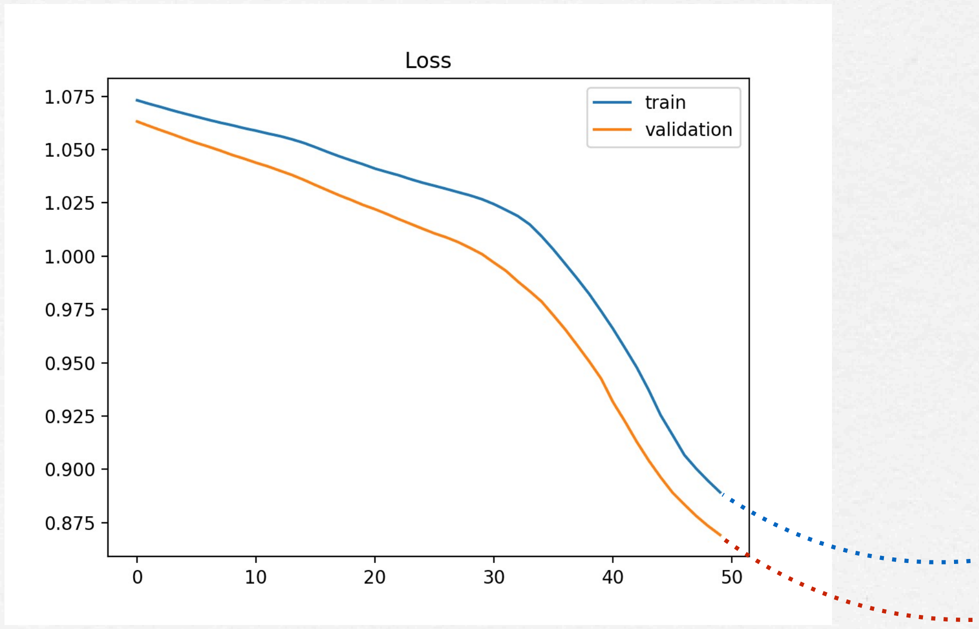
e.g., 4-fold cross-validation

* we may need to try the response of the model for different weight initializations

Monitoring ANN training (1)

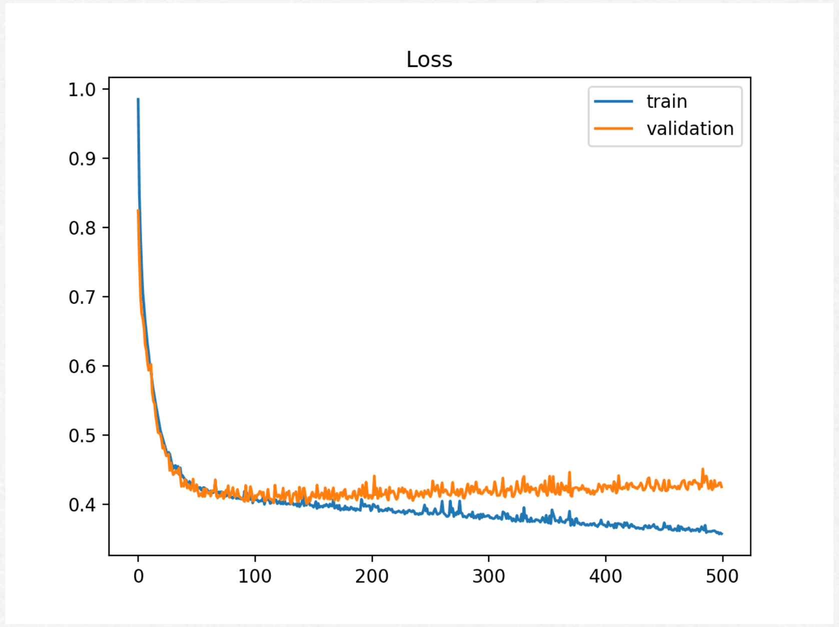
underfitting, probably because the model is too simple (left) or because there is a need for more training iterations (right).





Monitoring ANN training (2)

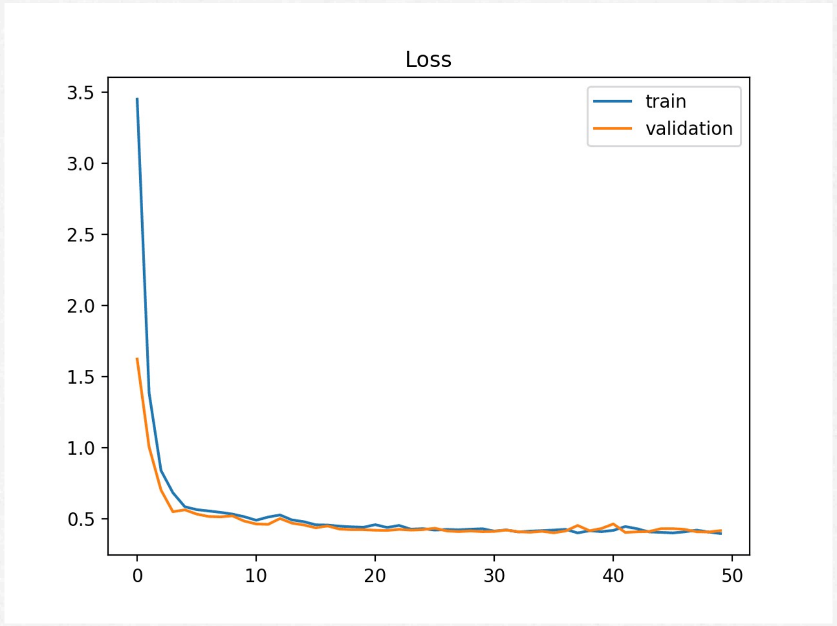
loss vs epochs plot: overfitting, probably because the model is too complex; there is an increasing gap between the evolution of the loss evaluated on the training dataset and the loss evaluated on the validation dataset



from https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Monitoring ANN training (3)

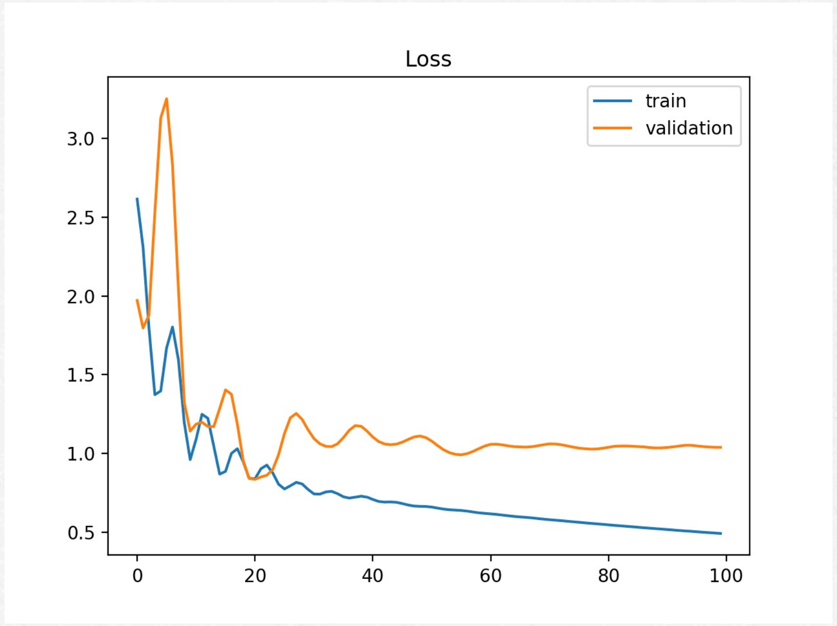
good fit; the loss function reaches a point of stability and there is a small gap between the training and the validation errors



from https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Monitoring ANN training (4)

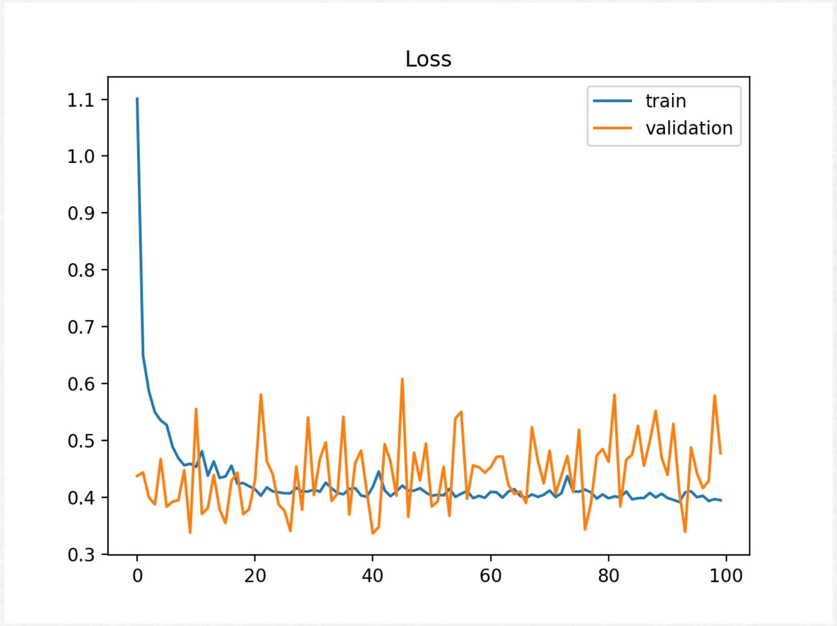
Unrepresentative training dataset: it does not provide sufficient information to learn the problem (e.g., too few examples or training data exhibits low diversity)



from https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

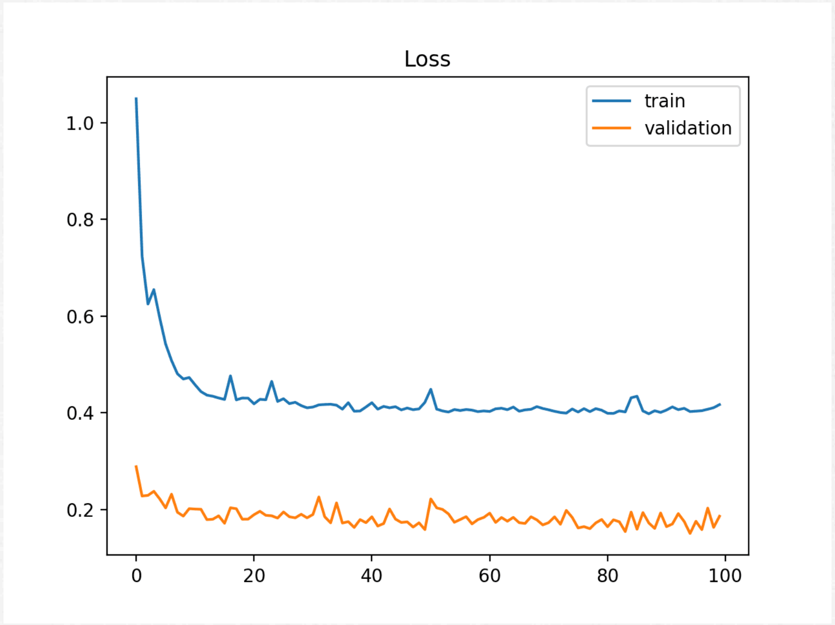
Monitoring ANN training (5)

Unrepresentative validation dataset. Here a randomly initialized model (epoch=0) performs similarly (on the validation dataset) to models that have been trained, even after 100 epochs.



Monitoring ANN training (6)

Unrepresentative validation dataset. Here, surprisingly the performance of the model on the validation dataset is better than on the training one. This means the validation dataset is too easy compared to the learning problem represented by the training dataset.


Loss
train
validation