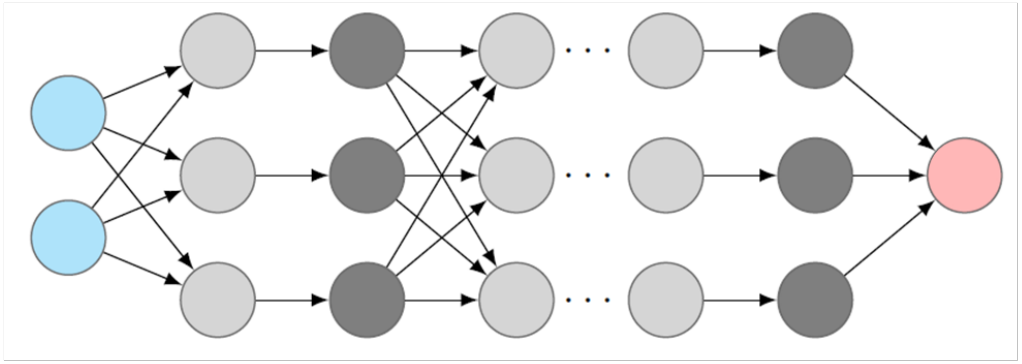
Generalization

- Validate, augment, regularize, generalize! -

Lecturer: Michael Möller – michael.moeller@uni-siegen.de
Exercises: Hartmut Bauermeister – hartmut.bauermeister@uni-siegen.de



Reminder

What (supervised) "Deep Learning" is:
A fancy word for function approximation

Assume there is an unknown function G that maps some kind of input data x to some kind of desired output y.

Assume we are given some evaluations of this (unknown) function G.

1. Choose a parameterized function $\mathcal{N}(x;\theta)$ in the hope that for the right choice of parameters θ it approximates the unknown function G well.

2. Try to determine suitable weights θ in such a way that $\mathcal{N}(x_i;\theta) \approx y_i$ holds for all examples (x_i, y_i) from your training data set, i.e. ***train the network***.

3. Try to ensure that both, the architecture as well as the training are chosen in such as way that the network makes good predictions during inference, i.e. on previously unseen data x : $\mathcal{N}(x;\theta) \approx G(x)$. We refer to this property as ***generalization***.

Towards generalization

So far our efforts have solely been focused on minimizing

$$E(\theta) = \sum_j \mathcal{L}(\mathcal{N}(x_j, \theta), y_j)$$

For a suitable network $\mathcal{N}$ and a suitable loss function $\mathcal{L}$ on our set of training examples (x_j, y_j).

Urgent question:

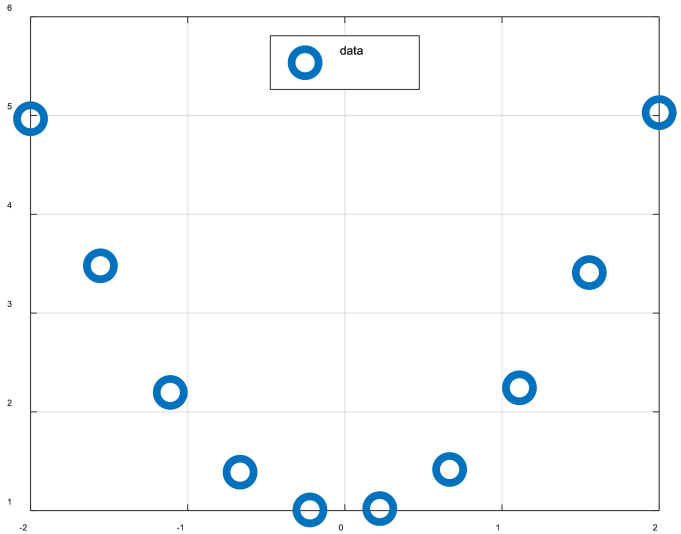
1) Do we even want to find the best possible fit to the training data?
2) What role does the network architecture play? How to choose it?

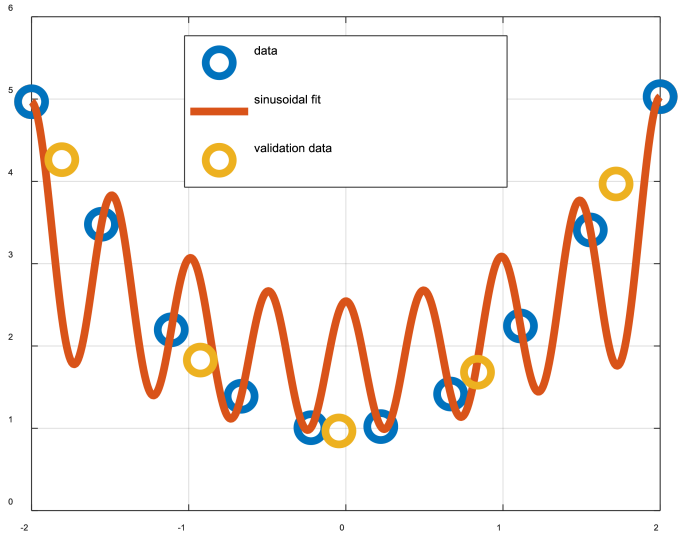
Ultimate goal: **Generalization**!

$$\mathcal{N}(x; \theta) \approx G(x), \qquad \forall x$$

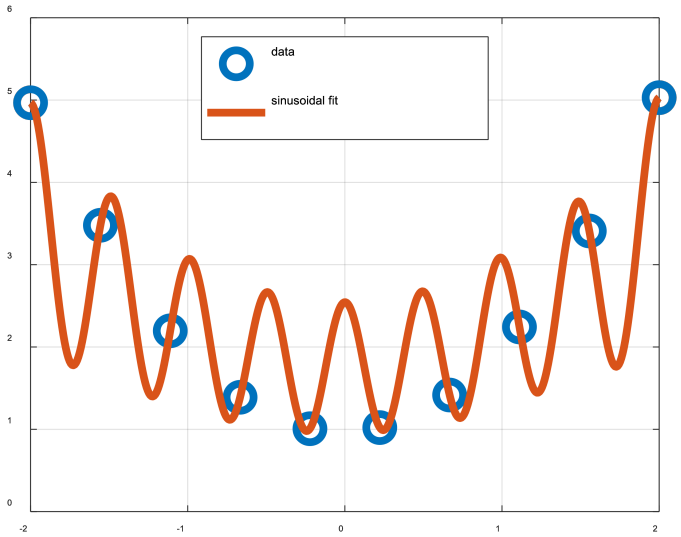
This often differs from $\mathcal{N}(x_j; \theta) \approx G(x_j), \qquad \forall \text{ training examples } j$

Generalization is difficult to judge

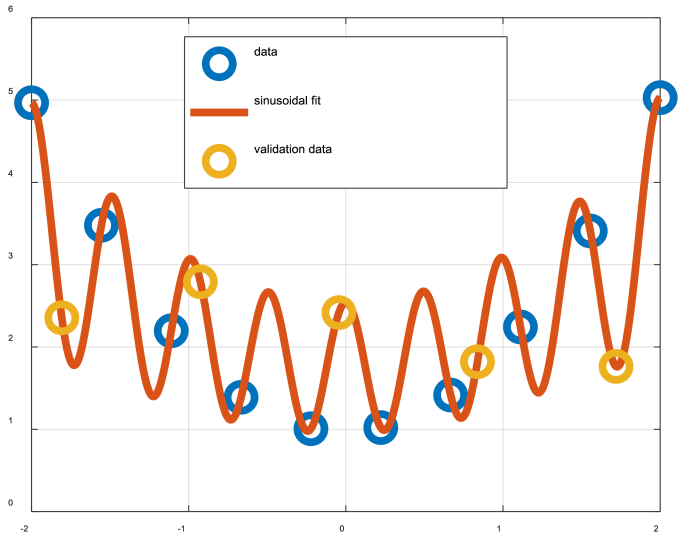




NO!



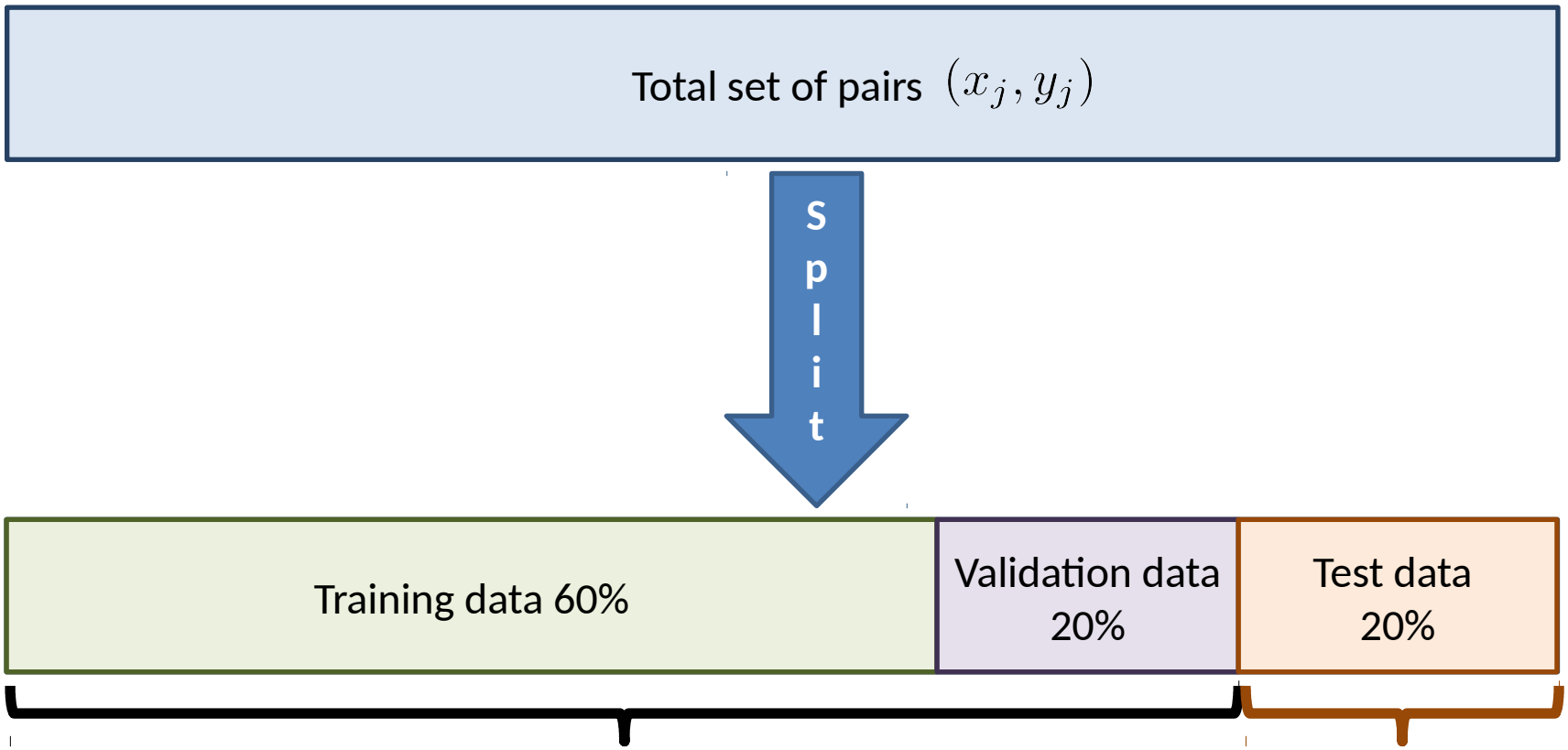
YES!



Is this a good fit for the given data?

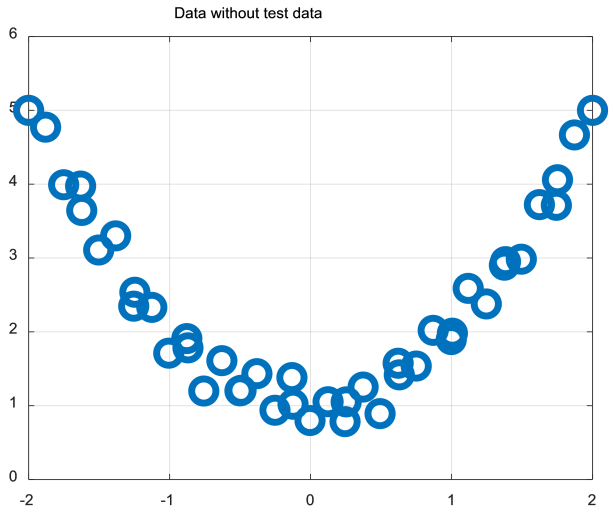
Good practice: Validate!

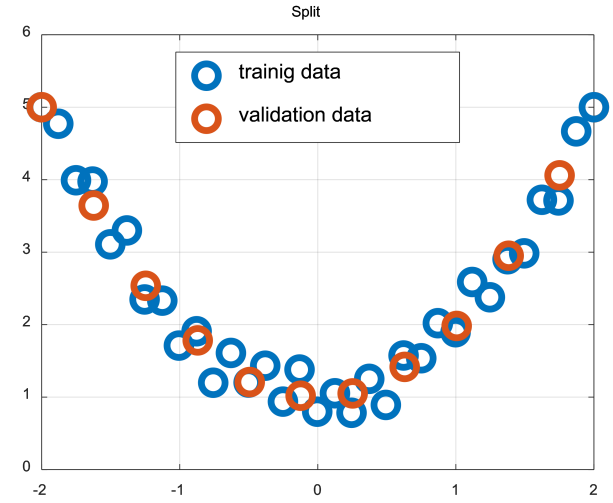
Without validation it is impossible to judge whether your model is reasonable or starts overfitting the data! Therefore:



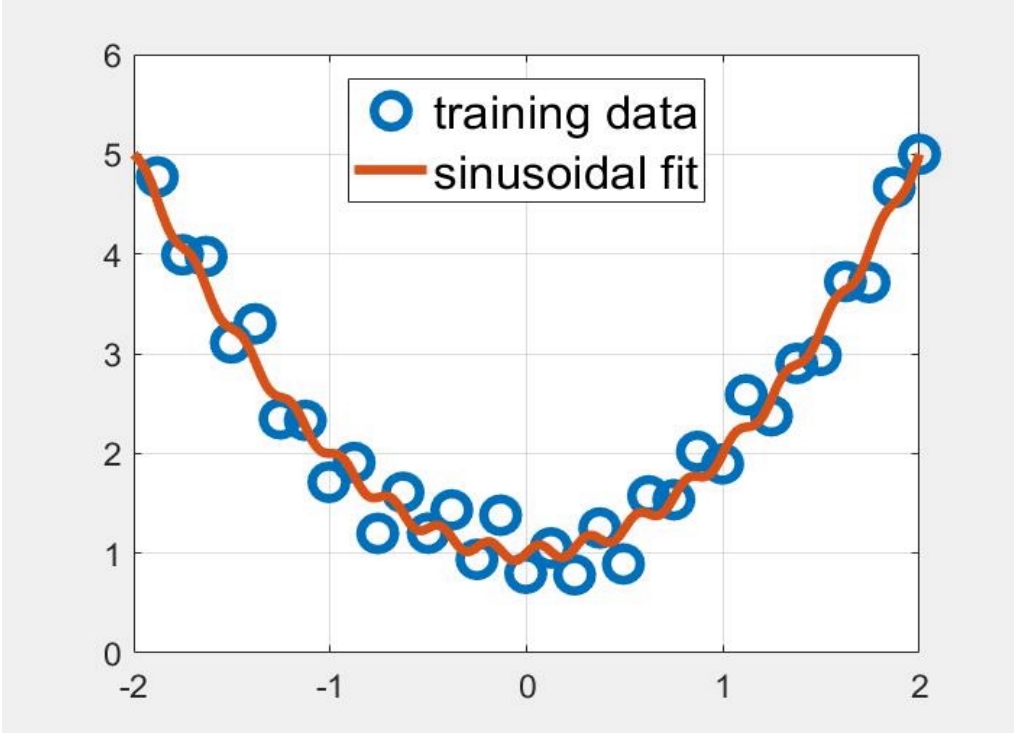
Good practice: Validate!

How do you use the validation set?





Progress of your optimizer



Good practice: Validate!

How do you use the validation set?

Good practice: Validate!

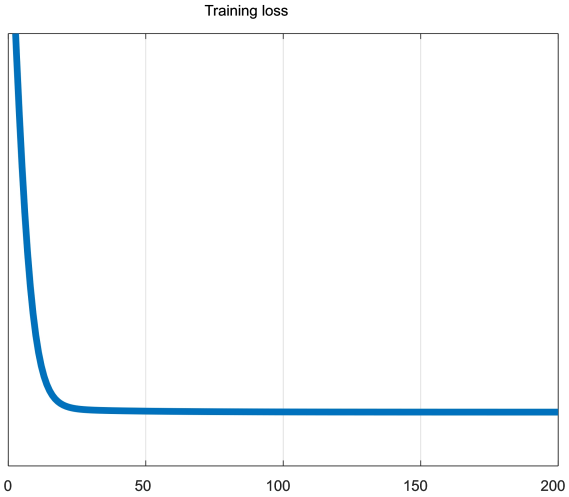
Stopping the optimizer early can be interpreted as a regularization technique. It prevents a too irregular shape of the trained network! Validation sets are crucial to detect overfitting!

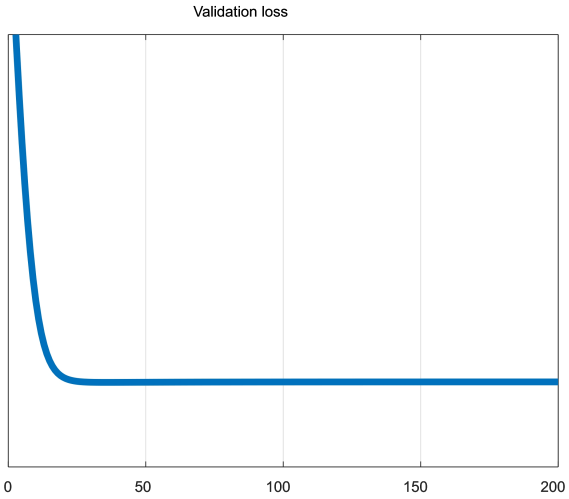
An additional frequently exploited technique for preventing irregular shapes of the trained network is to regularize the parameters of your layers to prevent them from becoming too large. The most common way to do this is to introduce a weighted quadratic penalty on the parameters of the network into the cost function:

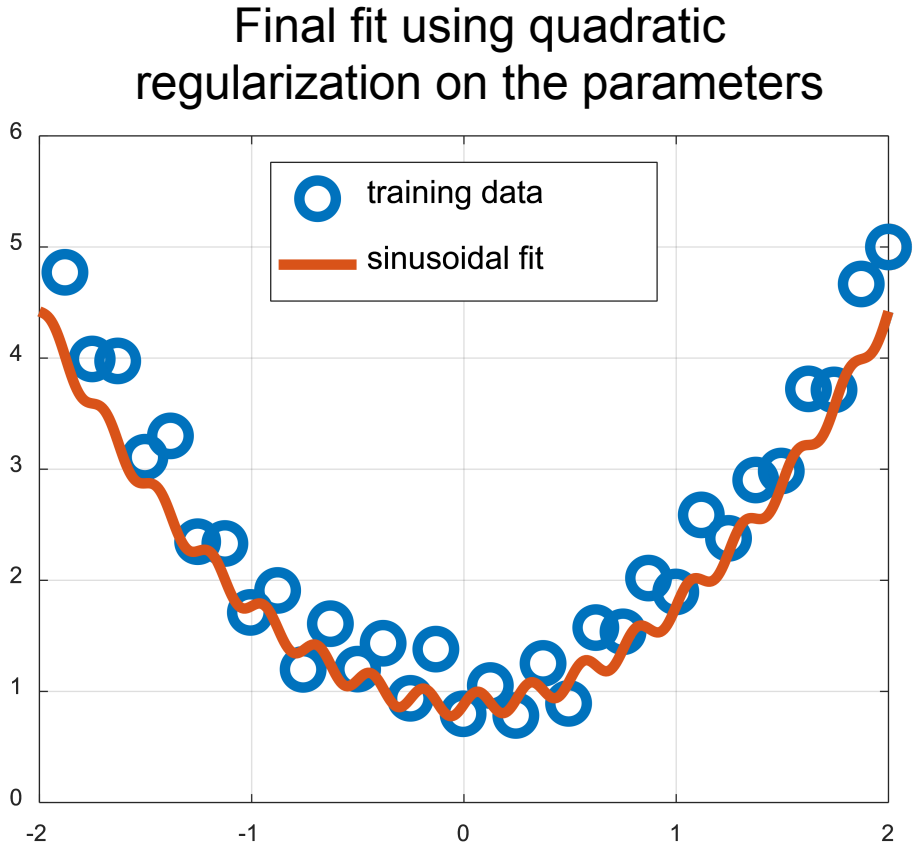
$$E(\theta) = \sum_j \|\mathcal{N}(x_j, \theta) - y_j\|^2 \; + \; \underbrace{\gamma \; \|\theta\|^2}_{=\gamma \sum_k \theta_k^2}$$

In the deep learning language, this is sometimes called *weight-decay*.

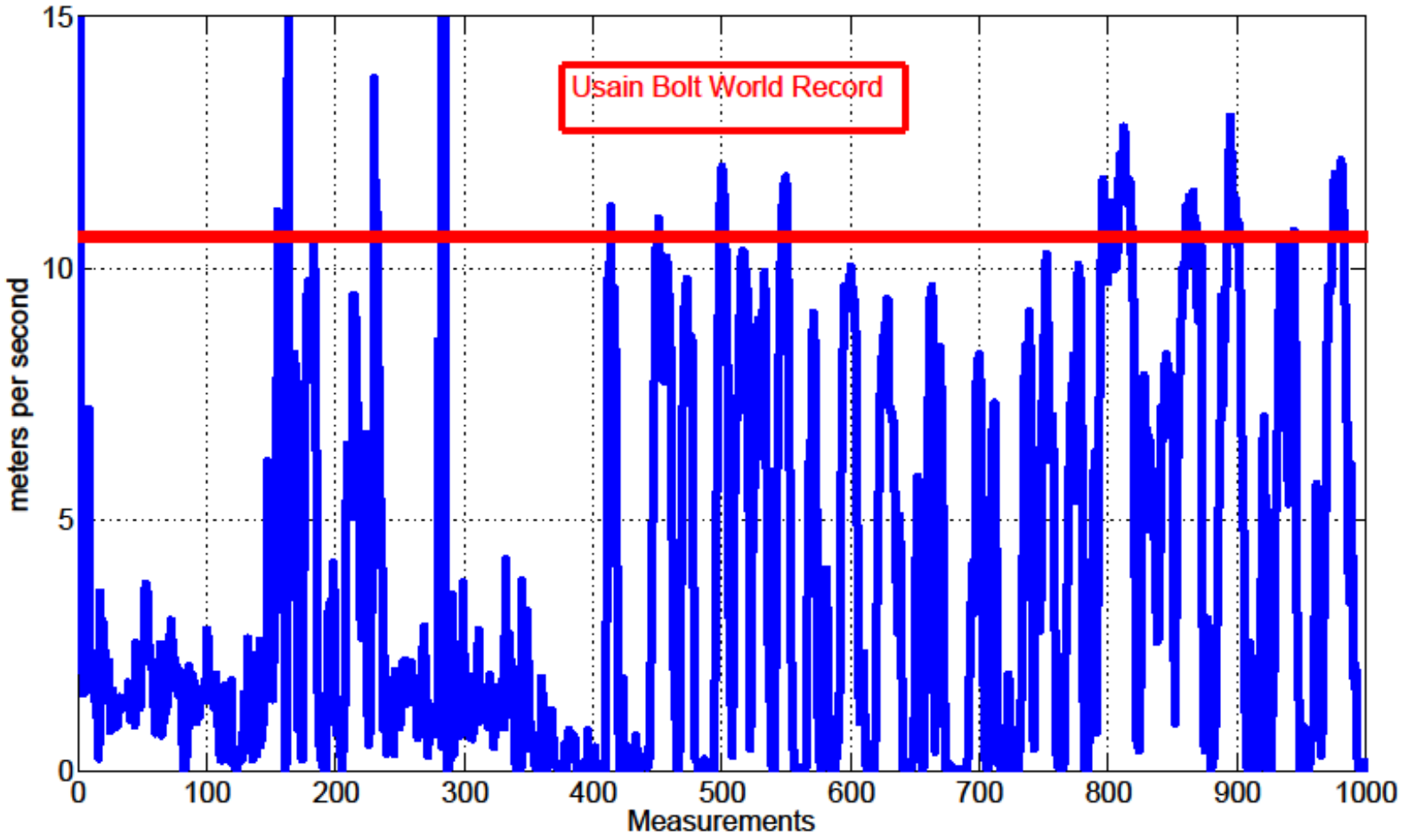
Regularization







Regularization beyond learning



Ways to improve generalization

Augmentation: If you know transformations your algorithm should be invariant against, create artificial training examples from the existing ones!

Original image – a giraffe

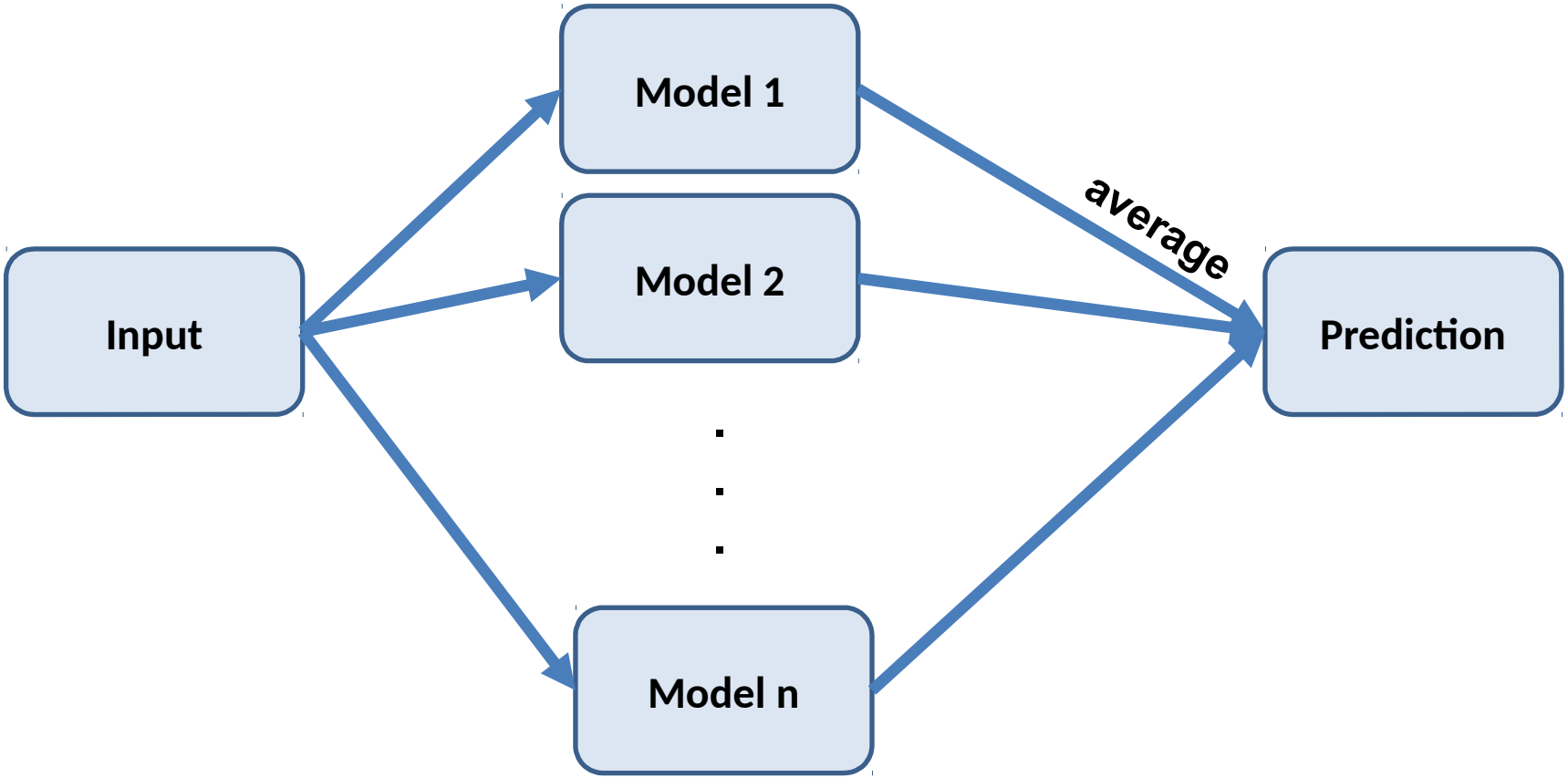
Flipped – still a giraffe

Changes of contrast – still a giraffe

Shifted image - still a giraffe, noisy image – still a giraffe, rotated image – still a giraffe, …

Ways to improve generalization

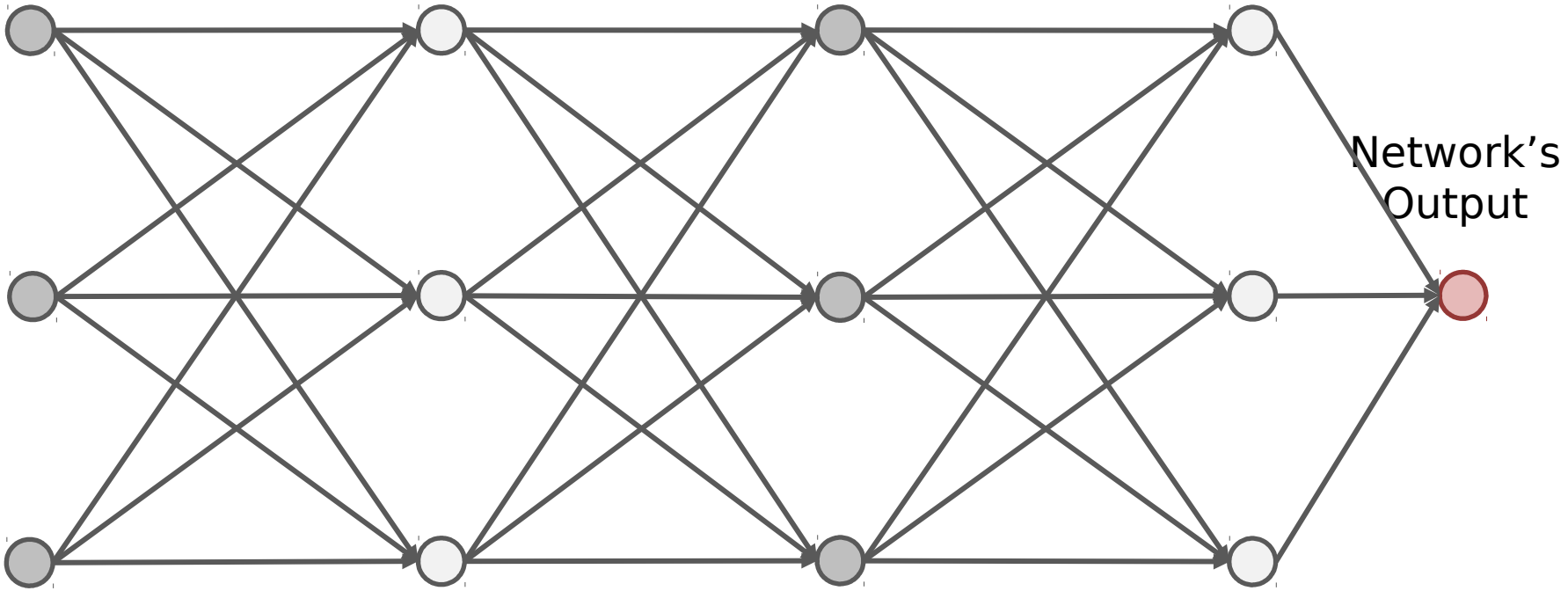
Ensemble learning: Train multiple classifiers using different training algorithms, initializations, loss functions, or even different architectures. Average their result during inference. If errors have little correlation, your results will improve!



See e.g. *Boosting* or *Bagging*

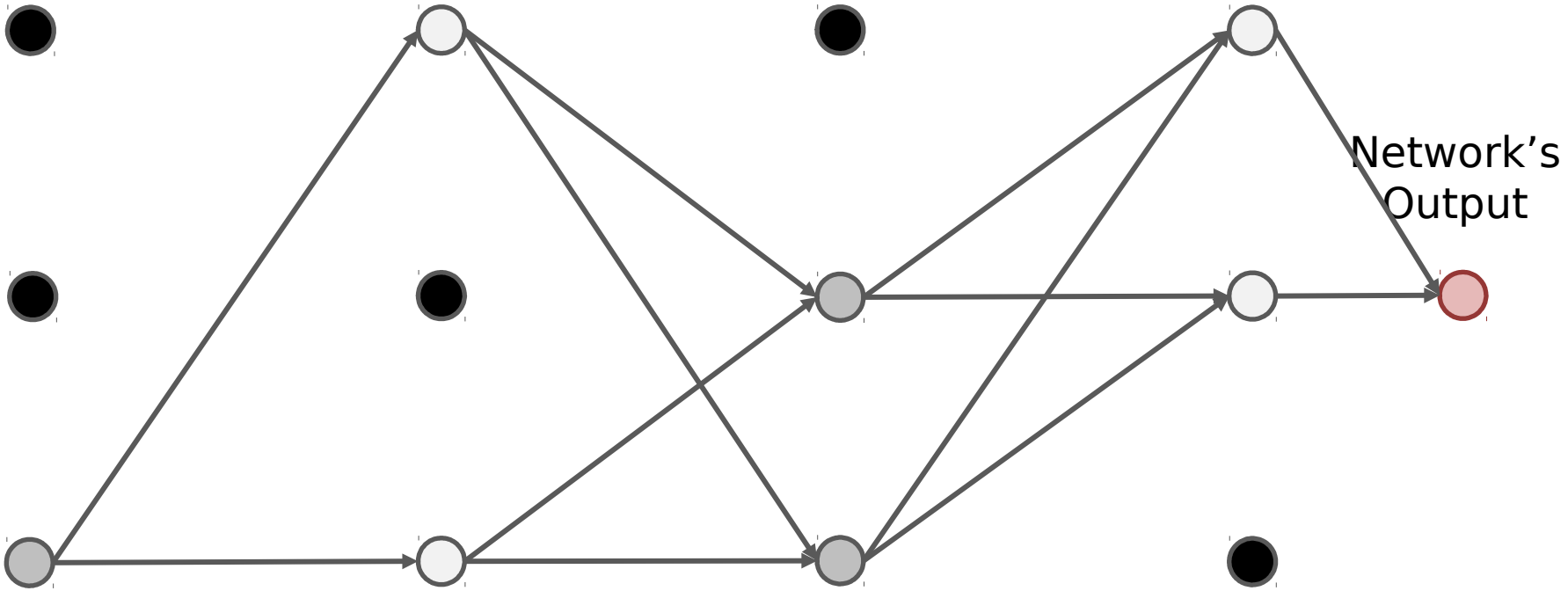
Ways to improve generalization

Dropout (Srivastava et al. 2014): Crazy and very effective! Randomly set about 40% of the neurons in a layer to zero!



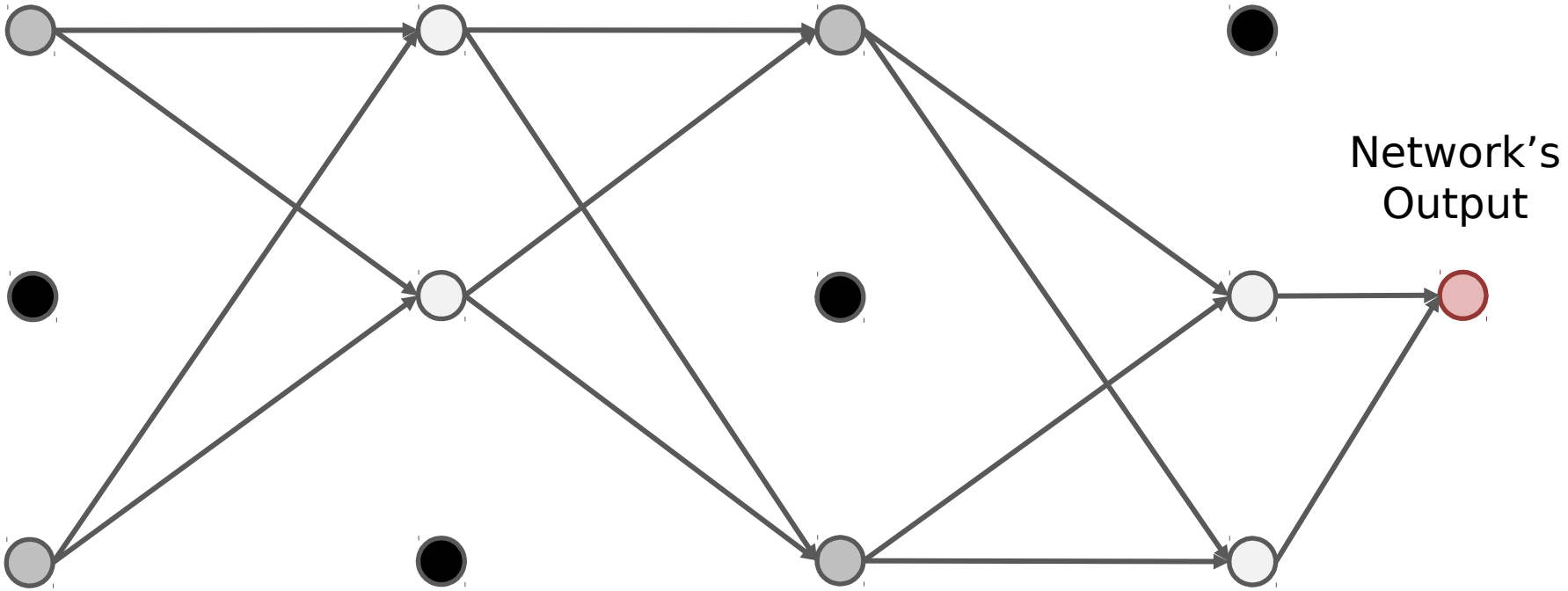
Ways to improve generalization

Dropout (Srivastava et al. 2014): Crazy and very effective! Randomly set about 40% of the neurons in a layer to zero!



Ways to improve generalization

Dropout (Srivastava et al. 2014): Crazy and very effective! Randomly set about 40% of the neurons in a layer to zero!



Ways to improve generalization

Intuition: Learn a redundant representation!

~~"great again"~~

"fake news"

~~"fake media"~~

"which hunt"

"caravan"

"Mexico will pay for the wall"

~~"covfefe"~~

Ways to improve generalization

Intuition: Learn a redundant representation!

- The network should learn to make the right prediction even if not all features are present. This helps to generalize to previously unseen data!
- Dropout could be interpreted as ensemble learning with shared parameters.
- Can also be understood as an architecture-dependent regularization penalty, see “Dropout Training as Adaptive Regularization” by Wagner et al.

Implementation details:

- During training, dropout randomly changes the nodes to be set to zero in every forward pass. The output of the non-zero nodes is scaled with p^{-1} where p is the fraction of elements dropout sets to zero. This roughly preserves the magnitude of the signals.
- During test time all nodes are used. There is no dropout.

Ways to improve generalization

Some practical advice:

- Use dropout rates in the range of 20 - 50%
- Use dropout on multiple (fully connected) layers
- Increase the size of your model – dropout helps you in preventing overfitting.

  A larger network has more freedom to learn redundant representation.
- Dropout can require more training time.

For extensions, see e.g. “Regularization of Neural Networks using DropConnect” by Wan et al.

Ways to improve generalization

We have seen the following generalization techniques

Early stopping
Stop optimizing once the validation loss increases!

Data augmentation
If you know transformations that should not alter your prediction, use them to generate more training data

Regularization with a penalty
Bound or penalize your weights, e.g. by a quadratic penalty. Use any kind of prior information!

Dropout
Train redundant networks that can accomplish a task with many different features!

Deep Learning Recipe

Do you have a small training error?

No → **Does the training error decay reasonably at the beginning of the training?**

- Yes → **Increase the network's expressiveness.**
- No → **Tune your training.**

Yes ↓

Do you have a small validation error?

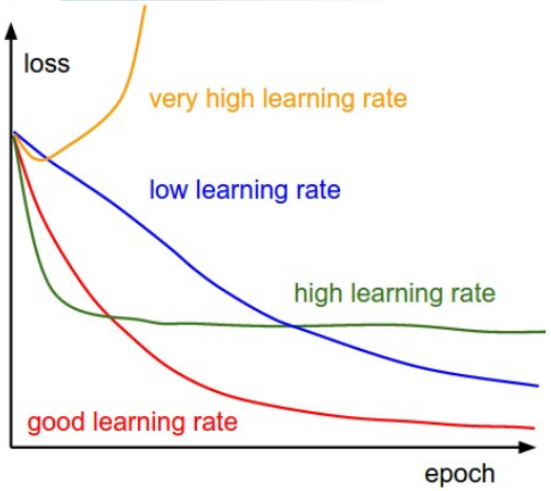
No → **Use more data (if possible).**

Regularization techniques.

Use prior knowledge (modelling parts of the network)

Yes ↓

Perfect! You're done!



If nothing works, try a different network architecture.