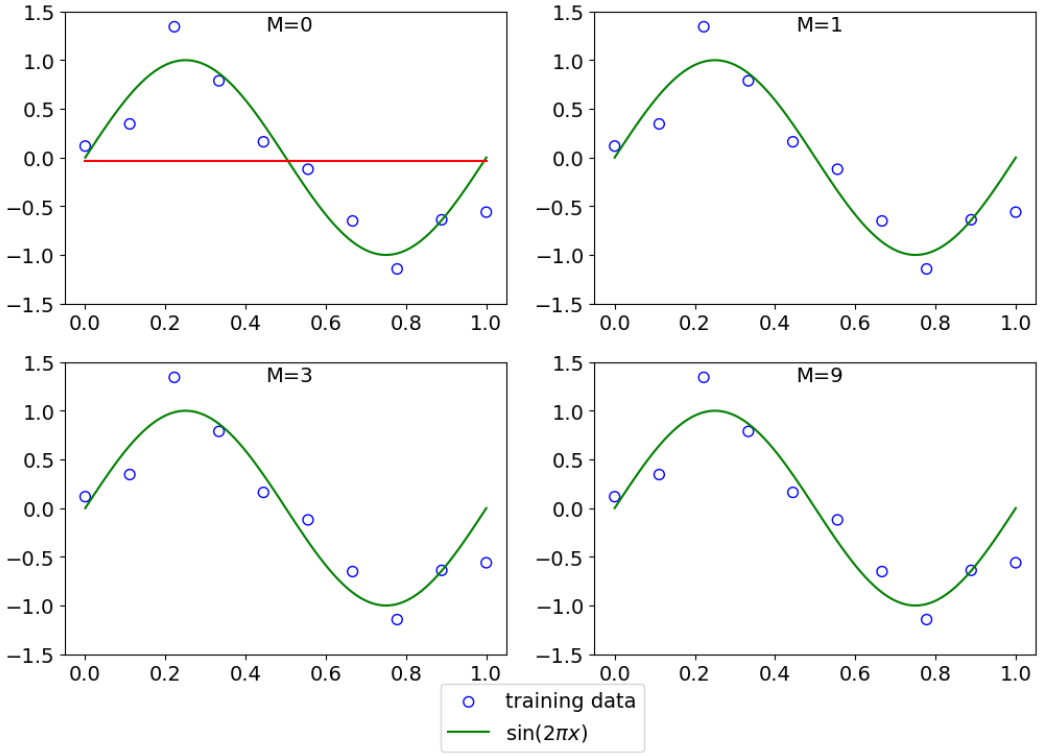
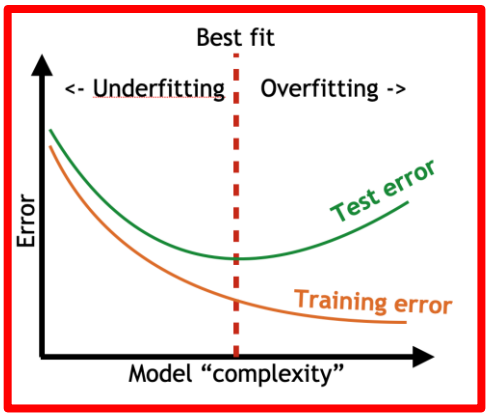
Fitting and model capacity

The following figure shows (noisy) measurements in blue, sampled from a ground truth function shown in green. Consider the use of different polynomial-curve-based models of order M.



1. Complete the sketch (you can draw directly into it) by adding estimations of how the fitted function would look like for the other model orders. An example fit is shown for M=0 as a red curve.
2. Explain under/overfitting on this example.
3. What methods could help to overcome overfitting. Discuss one method in detail.

- Model order corresponds to model complexity
- The main objective is to reduce the test error, depending on the model complexity but also data set size
- Distinguish between underfitting and overfitting
  underfitting: test error and training error improving with additional model complexity, model doesn't generalize because it's too simple
  Overfitting: test error increases and training error decreases, model doesn't generalize because it's also learning the noise
- Optional: error can be decomposed into bias and variance, bias error mainly follows training error, variance error mainly test error



Regularization (shrinkage methods), e.g. based on L1 or L2 norm, which penalizes large weights

- Other examples: data augmentation, early stopping, dropout, BN

Inception

Inception blocks have become a fundamental component in many deep neural network architectures. Answer the following questions:

1. Explain the concept of an Inception block. Provide an overview of what an Inception block is, its structure, its key components, and the motivation behind its design in neural networks.
2. What are the advantages of using Inception blocks in deep neural networks? Discuss the benefits of incorporating Inception blocks, such as improved model performance and computational efficiency, in comparison to traditional architectures.
3. What is the role of 1x1 convolutions in an Inception block? Elaborate on the significance of 1x1 convolutions and why this is essential in Inception blocks.

1. **Concept of an Inception Block**: The key idea behind an Inception block is to improve the network's ability to capture features of different sizes by using multiple convolutional filters and pooling operations in parallel. This architecture is named "Inception" because it can perform an "inception" or extraction of features at multiple scales within a single layer.
   Structure of a Typical Inception Block: Several parallel paths (branches) that apply different operations to the input. The most common components found within an Inception block are:
   1x1 Convolution: reduce the dimensionality of the input and capture fine-grained features. It serves as a bottleneck layer to reduce the number of channels
   3x3 Convolution: The 3x3 convolution path captures medium-sized features within the input, such as edges, textures, and some shapes.
   5x5 Convolution: The 5x5 convolution path focuses on capturing larger features and structures present in the input.
   Max Pooling: The pooling path down-samples the input, capturing the most significant features and aiding in translation invariance.
2. **Advantages of Inception Blocks**:
   a. **Improved Model Performance**: Inception blocks allow the network to capture a wide range of features, from fine details to larger patterns. This leads to improved performance as the network learns local and global features effectively.
   b. **Computational Efficiency**: Despite having multiple branches with different kernel sizes, often require fewer parameters than stacking multiple convolutional layers. Less overfitting and computational demands.
   c. **Combining filter sizes**: The outputs from a typical inception block are concatenated along the channel dimension, creating a feature map that incorporates information from various filter sizes.
3. **Role of 1x1 Convolutions:** Often referred to as the "network-in-network" or "bottleneck layer":
   a. **Dimension Reduction**: reduce the number of channels in the feature map, thus reducing the computational cost.

b. Capture fine-grained features

By combining 1x1 convolutions with larger convolutional filters (3x3, 5x5) and pooling operations in an Inception block, the network can efficiently capture features at different scales, improving its ability to recognize both fine details and broader patterns within the input image.

Quantization

Consider quantization as a model compression technique to reduce the overall complexity (compute, memory) of a neural architecture.

1. Discuss uniform and non-uniform quantization with regard to their advantages and disadvantages.
2. Provide and explain a uniform and a non-uniform quantizer using an equation.
3. Optional: For both types of quantization, provide an example demonstrating how to perform calculations using these number formats. You can either describe a specific calculation example, or describe how such computations would be carried out using typical hardware resources.

Uniform: equidistant quantization levels/quantization intervals of identical size

- Easy store and compute based on $\log_2(L)$ bits, additionally store quantization step ∂ (single element)
- Limited model capacity, representing W/A distributions is limited
- Computation:
  - use of standard HW components if W & A quantized identically
- $$Q(x) = \begin{cases} +1 : x \geq 0 \\ -1 : x < 0 \end{cases}$$

Nonuniform: non-equidistant quantization levels

- Each level requires storage of quantization step ∂, => scaling problems
- Computation requires re-coding into standard number format for most cases
- Improved model capacity, scaling factors allow to better represent W/A distributions
- Computation
  - RaS: trick to overcome the problem by reformulating the dot product
  - **c** = $\sum w_i a_i$ => $\sum(\{w\}\sum a_i)$
  - Otherwise, convert back into standard number format before computing
- $$w_l^i = \begin{cases} W_l^p : w_l > \Delta_l \\ 0 : |w_l| \leq \Delta_l \\ -W_l^n : w_l < -\Delta_l \end{cases}$$

Computational intensity

A plethora of hardware architectures exist that are tailored to particular forms of neural architectures. Different operations used in neural networks have different requirements on hardware architectures. One way to quantify this is the use of abstract HW metrics, describing the load put on different components of a hardware architecture. Consider computational intensity (FLOPs/Byte) as a key metric here.

1. Discuss the computational intensity of fully-connected and convolutional operations. If possible, provide a quantification (absolute or relative)
2. Explain the use of the roofline model of a given hardware architecture in this context.

1. Computational complexity

Fully Connected

$MAC_f = WHC \cdot O$
$W_f = WHC \cdot O$

Convolutional Layer

$MAC_c = \big((H - R + 1)(W - S + 1) \cdot RSC\big) \cdot M$
given R,S << H,W: MACc ~ O(HWCM)
$MAC_c = (EF \cdot RSC) \cdot M$
$W_c = RSCM$

Computational intensity

$$r_f = \frac{MAC_f}{W_f} = 1$$
$$r_c = \frac{MAC_c}{W_c} = EF$$

2. Be careful now about units, r up to now was number of elements, not bytes