Neural Networks 2 - Neural Networks

18NES2 - Lecture 5, Winter semester 2025/26

Zuzana Petříčková

October 21, 2025

Neural Networks 2 - Generalization

What We Covered Last Time

Practical Examples of MLP on Different Task Types

- **Binary classification** – IMDB movie reviews
  - Text data $\rightarrow$ sentiment analysis (positive / negative)
  - Representation: Bag-of-Words (vector of word occurrences)
- **Multiclass classification** – Fashion-MNIST
  - Image data $\rightarrow$ clothing type recognition
  - Representation: pixel intensity matrix (grayscale images)
- **Regression** – Boston Housing dataset
  - Tabular numerical data $\rightarrow$ prediction of house prices
  - Focus on model evaluation (cross-validation)

What We Covered Last Time

Regression – Boston Housing dataset

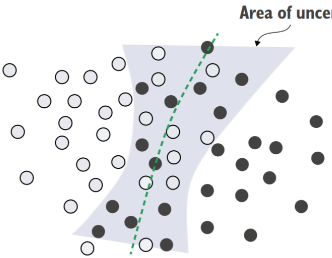
- We can quickly revisit this example today — we didn't have time to finish it last time.

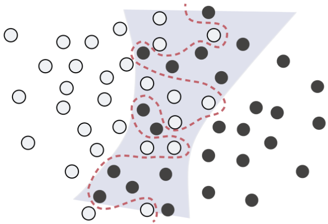
2nd homework assignment

- The assignment was due **today**.
- Those who have submitted can arrange a short consultation with me this week.
- **Points will be awarded only to those who complete the consultation by the end of this week.**

Generalization of Neural Networks

- The ability to produce correct outputs for inputs not seen during training
- Illustration: well-trained model vs. overfitted model

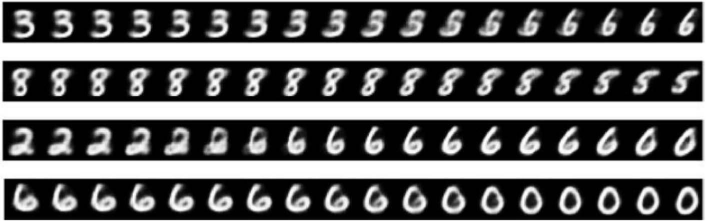




F. Chollet: Deep Learning with Python, Fig. 5.5

Generalization of Neural Networks

- Class boundaries are often hard to define:

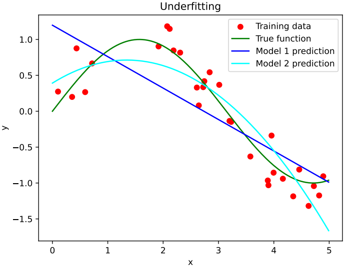


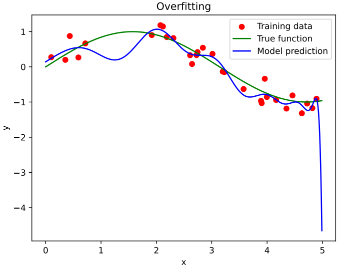
F.

Chollet: Deep Learning with Python, Fig. 5.7

Underfitting vs. Overfitting – Regression Example

- Typical illustration of underfitting and overfitting in regression tasks:





Neural Networks Tend to Overfit

Observation: Neural networks can easily memorize patterns from the training data — even those that are irrelevant or random.

Example (IMDB sentiment analysis):

- Suppose a rare word appears only once in the training set — in a negative review.
- The model may assign a **strong negative weight** to this word, assuming it indicates negative sentiment.
- As a result, it performs very well on the training data, but fails to generalize to unseen reviews.

General phenomenon:

- Overfitting means the model captures *noise instead of structure*.
- The model's internal representation becomes too specific to the training set.

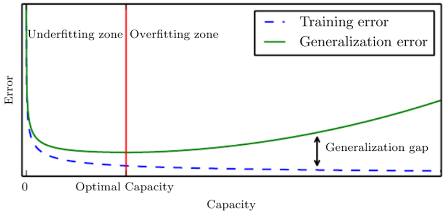
Practical example

irrelevant_features.ipynb

- Practical example: dataset with irrelevant input featues
  - We start from the **Fashion MNIST** dataset and extend each image with an additional channel of **irrelevant features**:
    - random noise (completely uninformative)
    - or constant zeros (no variation at all)
  - The task remains the same — classify the clothing item.
  - Compare models trained:
    - on the original data,
    - on data with noise features,
    - and on data with zero-valued features.
  - Observe how irrelevant input dimensions:
    - affect validation loss and accuracy,
    - influence overfitting and generalization,

Generalization and Model Capacity

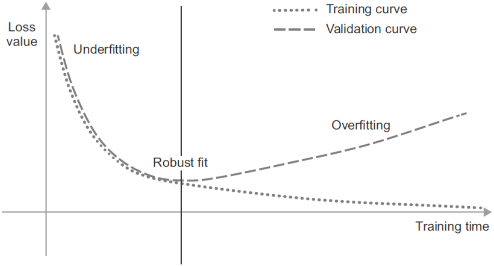
- Generalization depends on the network's architecture and model capacity (i.e., number of parameters/weights)
- **Small model:**
  - Potentially stable but inaccurate predictions, risk of **underfitting**
  - Needs fewer training samples to generalize well
- **Large model:**
  - Greater variability in performance, risk of **overfitting** – poor generalization
  - Requires more training data to generalize properly



How to Measure Generalization

Sampling-based techniques:

- **Validation set:**
  - Split training data into training (e.g., 70%) and validation/test (30%) subsets
  - Train on training subset only
  - Use validation/test data to estimate generalization error



F. Chollet: Deep Learning with Python, Fig. 5.1

- **Cross-validation** (e.g., k-fold CV)

Validation Set – Best Practices

Things to keep in mind:

- All subsets (training, validation, test) should be representative and balanced across classes
- For time series: validation/test data should follow training data chronologically
- Avoid redundancy – similar examples in training and validation/test sets may bias evaluation

Cross-Validation (CV)

- Allows reliable generalization error estimation, especially with small datasets
- Extends the basic train/test split principle
- Helps detect overfitting / underfitting
- Useful for model and hyperparameter comparisons

Common types of CV:

- **Monte Carlo CV** – random, flexible, suitable for mid-size datasets
- **k-fold CV** – systematic, ensures all samples are used, great for small datasets

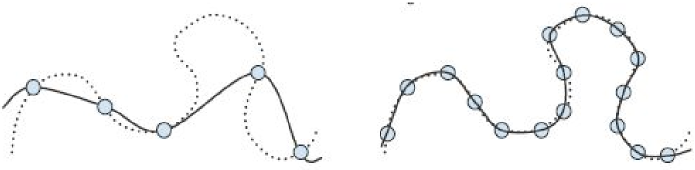
How to Improve Generalization in (Deep) Neural Networks?

- **Find the optimal architecture** for a given dataset (number of layers and neurons, activation functions)
  - **Neural Architecture Search (NAS)**, e.g., AutoKeras library
- **Increase training set size** (data augmentation)
- **Feature engineering** (extract more informative input features)
- **Early stopping** using a validation set
- **Regularization techniques**
  - **L1/L2 regularization, Dropout**, DropConnect
  - Label smoothing
- **Normalization** of data, weights, and layer outputs
- **Transfer learning** and **Ensembling**
- **Hyperparameter tuning** (Grid Search, Random Search, Bayesian Optimization), e.g., Keras Tuner

Increase the Training Set Size (Data Augmentation)

Why use data augmentation?

- When the dataset is **imbalanced** $\rightarrow$ augment underrepresented classes
  - undersampling vs. oversampling
- When there is **not enough data** $\rightarrow$ generate additional training samples
  - sparse vs. dense sampling:



F. Chollet: Deep Learning with Python, Fig. 5.1

How to Increase the Training Set Size?

Depending on data type:

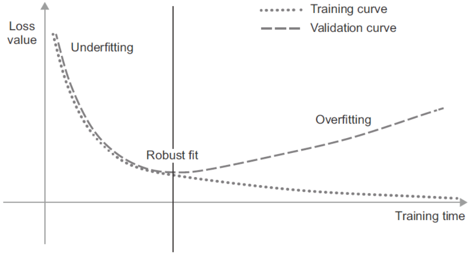
- **Image data:** geometric transformations (rotation, flipping), brightness and contrast changes, cropping, adding noise, image blending, etc. (`keras.ImageDataGenerator` can apply them in real time)
- **Text data:** synonym replacement, word deletion or swapping, paraphrasing
- **Audio data:** time shift, pitch or speed modification, adding background noise
- **Sequential data:** random cropping, temporal masking, or dropout of samples

Synthetic data generation:

- Adding controlled noise or small random perturbations
- Using random processes (e.g., Markov chains) or simulations
- Employing **generative models** (GANs, VAEs, diffusion models)

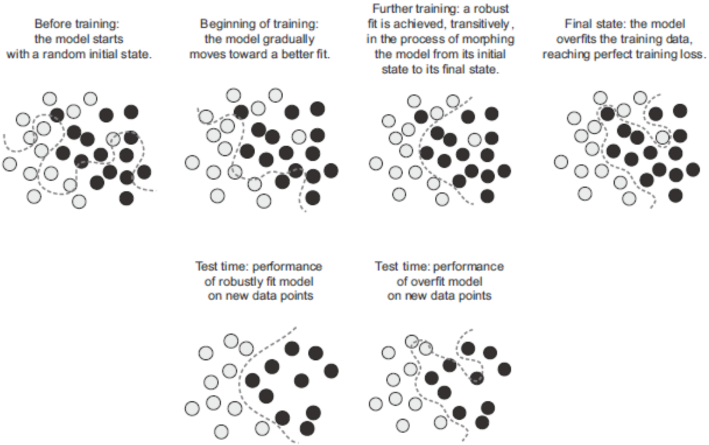
Early Stopping

- Split training data into training (e.g., 70–90%), validation and test subsets
- Train the model only on the training subset
- Stop training once validation loss starts increasing
- Evaluate the model performance on the test set
- Caution: validation and test sets must be completely independent!



F. Chollet: Deep Learning with Python, Fig. 5.1

Early Stopping – Visualization



F. Chollet: Deep Learning with Python, Fig. 5.10

Regularization Techniques

Core idea:

- Add penalty terms to the basic loss function (e.g., E_{loss}):

$$E = c_{loss} \cdot E_{loss} + c_A E_A + c_B E_B + \dots$$

- **Occam's Razor:** smaller networks with simpler, smoother functions generalize better
- Many penalty terms exist, from simple to complex

Regularization Techniques – L2 Regularization

L2 Regularization (Weight Decay) (Werbos, 1988)

- One of the most well-known penalty terms:

$$E = \beta E_{loss} + (1-\beta) \cdot \frac{1}{2} \|\vec{w}\|_2^2 = \beta E_{loss} + (1-\beta) \sum_i w_i^2$$

  - i indexes all weights and biases in the model
  - $0 \leq \beta \leq 1$ ... weights the error terms
- Weight update rule:

$$w_i(t+1) = w_i(t) - \alpha_r w_i(t)$$

- Penalizes large weights, helps prevent overfitting
- We can prune insignificant weights (i.e., weights with low magnitude)

Regularization Techniques – L1 Regularization

L1 Regularization (Lasso)

- Promotes sparsity by zeroing some weights:

$$E = \beta E_{loss} + (1 - \beta) \cdot \frac{1}{N_w} \sum_{i=1}^{N_w} |w_i|$$

- Weight update rule:

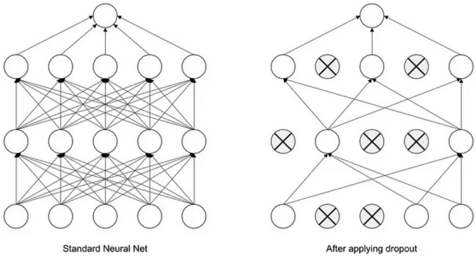
$$w_i(t+1) = w_i(t) - \alpha \frac{\partial E_{loss}}{\partial w_i} - \alpha_r \cdot \text{sign}(w_i)$$

Adding Gaussian noise to the training set

- Augment the training set with a bit “noised” samples
- Has a similar effect to L2 regularization

Dropout (Srivastava et al., 2014)

- Highly effective regularization method
- Randomly deactivates hidden neurons during training
- During inference (after the model is trained), all neurons are active
- Implemented by adding a **Dropout** layer after each fully connected hidden layer



Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014

Normalization Techniques

- Includes normalization of data, weights, and layer outputs
- Often implemented via dedicated normalization layers
- **Batch Normalization** – normalizes layer otputs across batch samples for each neuron (used in MLPs, CNNs)
- **Layer Normalization** – normalizes layer outputs across neurons per sample (used in RNNs, Transformers)
- Helps prevent saturation and vanishing gradients
- Overall, improves stability and convergence in deep networks

Other Generalization Techniques for Deep Networks

- **Label smoothing** – prevents overconfident predictions by softening the target distribution
- **Transfer learning** – reuses pretrained models on similar tasks
- **Ensembling** – combining multiple models improves accuracy and robustness
- ...

Practical Advice: When Your Model Fails to Generalize

- Extend training data or extract better features
- Reduce model size, use better optimizer, finetune hyperparameters (e.g. batch size, learning rate)
- Apply Dropout
- Alternatively:
  - Use Batch Normalization for large models
  - Use L2 Regularization for smaller models

Tip: Start simple. Monitor results. Regularize wisely.

Practical Examples

regularization_imdb.ipynb

- Demonstrates core techniques for generalization: early stopping, dropout, regularization, label smoothing, ensemble model, reduced model

regression_boston_housing.ipynb

- Regression example from last week
- Includes k-fold cross-validation example

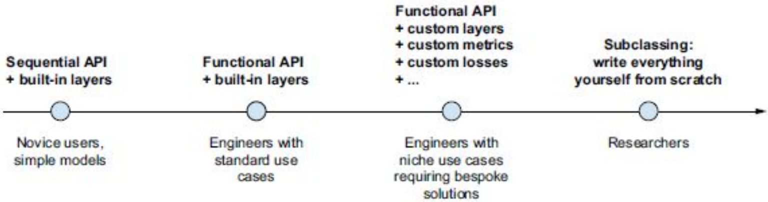
functional_model.ipynb

- Introduction to functional API in Keras (needed e.g., for ensembles)

Aside: Building Keras Models

Keras provides three main APIs for building models:

- **Sequential API** — the simplest option; layers are stacked one after another (linear stack)
- **Functional API** — the most commonly used; supports complex, graph-like model architectures
- **Model subclassing** — the most flexible, low-level approach giving full control



F. Chollet: Deep Learning with Python, Fig. 7.1

functional_model.ipynb

3rd Graded Homework: Generalization on Fashion MNIST

Goal: Start from the **Fashion MNIST** notebook and explore different techniques to improve model **generalization**.

- **Step 1:** Increase the model size or depth so that you can clearly observe **overfitting** (validation error increasing while training error decreases).
- **Step 2:** Apply and compare regularization techniques that help reduce overfitting.

3rd Graded Homework: Generalization on Fashion MNIST

Requirements

- Keep the pipeline: load $\rightarrow$ preprocess $\rightarrow$ build $\rightarrow$ train $\rightarrow$ evaluate.
- Visualize training curves (loss and accuracy) and discuss signs of overfitting / underfitting.
- Report **accuracy** and include a **confusion matrix**.
- Try at least **four** regularization techniques, e.g.:
  - early stopping,
  - dropout layers,
  - L_2 or L_1 (weight) regularization,
  - ensemble model,
  - label smoothing
- Apply each technique separately first, then optionally combine some of them.
- Provide a short summary of your findings in the notebook.

3rd Graded Homework: Generalization on Fashion MNIST

Submission

- Submit the notebook by **Nov 3, 2025**.
- **Consultation required by Nov 7, 2025** to receive points (short discussion after lab or individually).