Neural Networks 1 - Multilayer neural networks

18NES1 - Lecture 9 (First part), Summer semester 2024/25

Zuzana Petříčková

April 15th, 2025

What We Covered Last Week

Multilayer Neural Network (MLP)

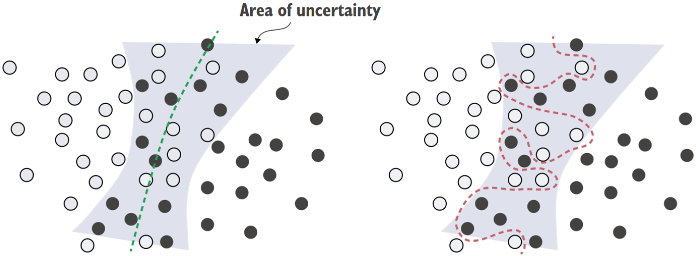
1. More notes on hyperparameter setting.
   - Learning algorithms for multilayer neural networks
2. Examples of various types of tasks:
   - Binary classification (already covered), multiclass classification, regression, time series prediction
   - Specifics of each task and data preprocessing
3. Generalization in MLPs and techniques for preventing overfitting (with demonstrations and examples) - Introduction

This Week

1. Generalization in MLPs and techniques for preventing overfitting (with demonstrations and examples)
2. Introduction to Convolutional Neural Networks

Generalization of Neural Networks

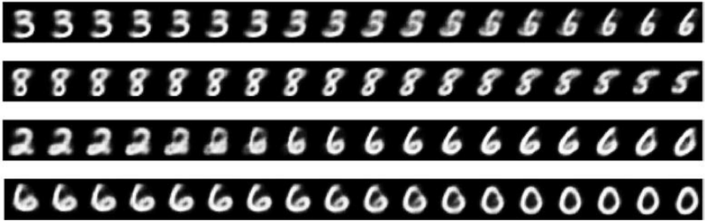
- The ability to produce correct outputs for inputs not seen during training
- Illustration: well-trained model vs. overfitted model



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

Generalization of Neural Networks

- Class boundaries are often hard to define:

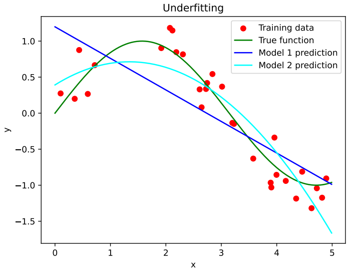


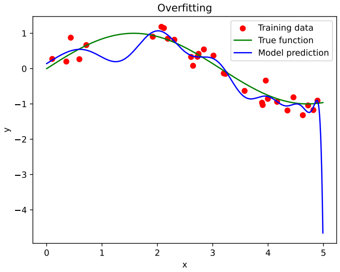
F.

Chollet: Deep Learning with Python, Fig. 5.7

Underfitting vs. Overfitting – Regression Example

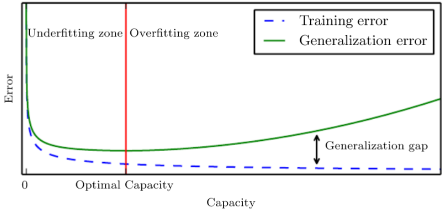
- Typical illustration of underfitting and overfitting in regression tasks:





Generalization and Model Capacity

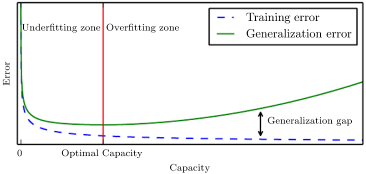
- Generalization depends on the network's architecture and model capacity (i.e., number of parameters)
- **Small model:**
  - Potentially stable but inaccurate predictions
  - Risk of **underfitting**
- **Large model:**
  - Greater variability in performance
  - Risk of **overfitting** – poor generalization



https://www.deeplearningbook.org/, Figure 5.3

Model Capacity and Dataset Size

- The required training set size depends on model capacity
- **Small model:**
  - Stable but potentially underfit
  - Needs fewer training samples to generalize well
- **Large model:**
  - Risk of overfitting
  - Requires more training data to generalize properly



https://www.deeplearningbook.org/, Figure 5.3

Theoretical Insight: Generalization and Training Set Size

Theorem: Relationship between model capacity and required number of training examples

- For a network with one hidden layer, w parameters, h hidden units, and generalization error ϵ, the minimum number of training samples N should satisfy:

$$N \geq \frac{w}{\epsilon} log_2(\frac{h}{\epsilon})$$

$\rightarrow$ If $N < \frac{w}{\epsilon}$, the model cannot generalize properly

- For target accuracy $\geq 90\%$, choose at least $10 \cdot w$ training samples

Generalization in Deep Networks

Estimated training set size for deeper architectures:

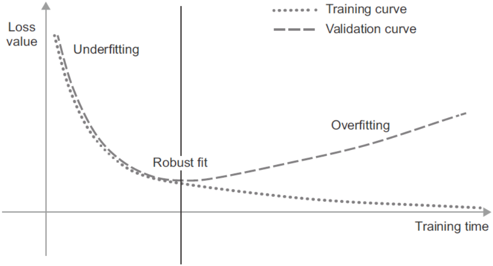
$$N \geq O\left(\frac{w \cdot \log w}{\epsilon}\right)$$

- More layers $\rightarrow$ more parameters $\rightarrow$ more data needed
- Empirical rule: **Often we need significantly more training samples than parameters**
- To achieve good generalization:
  - Use a sufficiently large training set, or
  - Apply suitable regularization techniques

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

Monte Carlo Cross-Validation

Basic principle: random repeated splitting

1. For $i = 1, \ldots, k$:
    - Randomly split dataset T into T_1 (training) and T_2 (test), e.g. 70:30
    - Train the model on T_1, evaluate on T_2
    - Record the test error
2. Compute mean and standard deviation of errors over k runs (typically $k = 100$)

k-Fold Cross-Validation

- Compared to Monte Carlo, it systematically covers the entire dataset (no sample is left out)

Basic principle:

1. Split training data T into k equally sized disjoint subsets $T_1, \dots, T_k$
2. For $i = 1, \dots, k$:
   - Train on $T \setminus T_i$, evaluate on T_i
   - Record the test error
3. Compute the average and standard deviation over all k runs (commonly $k = 10$)

Practical example

regularization_mnist.ipynb

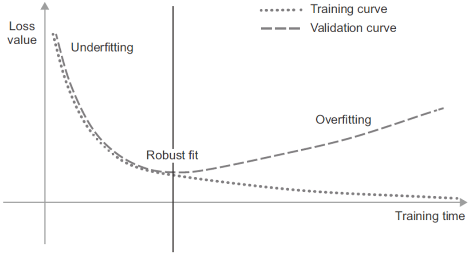
- Practical example: MNIST digit dataset with limited training set size
- Task: experiment with model size and training set size
- Observe how validation and test error increase (i.e., generalization performance decreases) as the training set becomes smaller or the model becomes larger

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

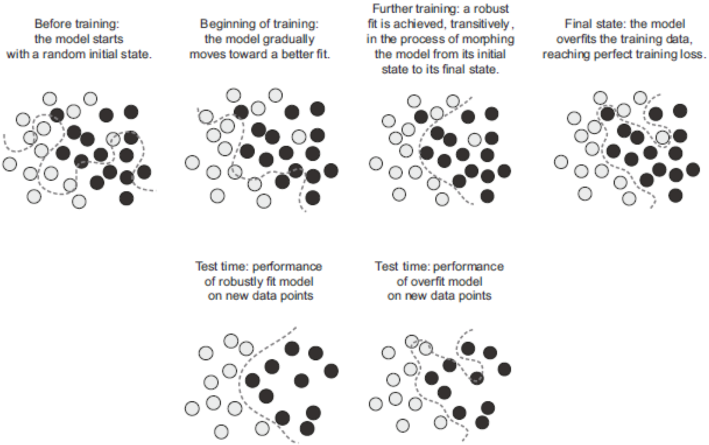
Early Stopping

- 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:

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

- 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:

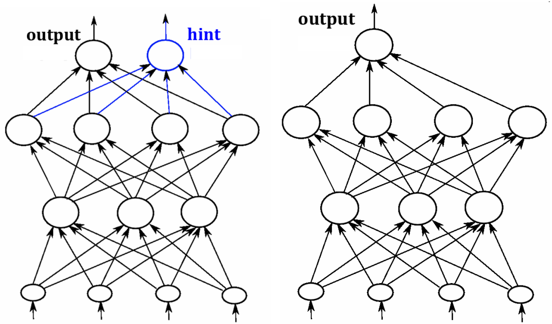
$$w_i(t+1) = w_i(t) - \alpha \frac{\partial E_{loss}}{\partial w_i} - \alpha_r \cdot \mathrm{sign}(w_i)$$

Adding Gaussian noise to the training set

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

Hint-Based Learning

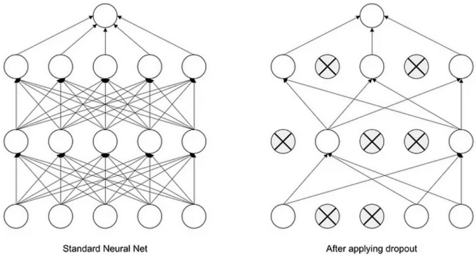
(Mostafa, 1993; Suddarth, 1990)



- Enhances generalization and accelerates training
- Leads to smoother learned functions, supports pruning

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 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

Pruning of Multilayer Neural Networks

Idea:

- Evaluate which parts of the model are important:
  - Connections (weights)
  - Hidden neurons
  - Input features
- Remove redundant parts of the model

Motivation:

- Faster inference, reduced memory requirements
- Improve generalization and reduce overfitting
- Produce a more interpretable model
- Automatically detect the most important input features

Pruning of Multilayer Neural Networks

Algorithm:

1. Train a model with a sufficiently large architecture
2. While validation error continues to decrease (or remains below a threshold):
    1. Compute relevance scores for hidden units or connections
    2. Remove the least relevant neuron(s) or connection(s)
    3. Fine-tune (retrain) the pruned network

Challenges:

- How to define neuron/connection relevance
- Choosing an appropriate pruning strategy

Pruning Criteria – Measuring Relevance of Neurons

Common relevance scores for hidden neurons:

- **Sum of outgoing weights:** $W_i = \sum_j w_{ij}^2$
  $\rightarrow$ Simple yet effective
- **Goodness factor:** $G_i = \sum_p \sum_j (y_i^p w_{ij})^2$
  $\rightarrow$ Rewards neurons with strong connections and frequent activation
- **Consuming energy:** $E_i = \sum_p \sum_j y_i^p w_{ij} y_j$
  $\rightarrow$ Highlights neurons that are often co-activated with the next layer
- **Sensitivity coefficients:** $S_{ij} = \frac{\partial y_j}{\partial w_i}$ or $\frac{\partial y_j}{\partial y_i}$

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

- Improve training data or extract better features
- Reduce model size, use adaptive learning rate, tune hyperparameters
- 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_mnist.ipynb

- MNIST digits dataset with limited training size
- Demonstrates core techniques for generalization: early stopping, regularization, dropout
- Includes cross-validation example

images_simple_mlp_autoencoder.ipynb

- Simple autoencoder (input = output)
- Illustrates how regularization (e.g., dataset augmentation) influences learning
- Try varying training set size and number of neurons to observe effects

Optional Homework

images_simple_mlp_autoencoder.ipynb

- Use your own images and modify the notebook to strongly alter their color palette (differently than in the example)
- Choose your own training image and create a custom data augmentation strategy
- You may need to adjust the number of neurons to fit your input
- Submit the modified notebook, original image(s), and transformed output(s)