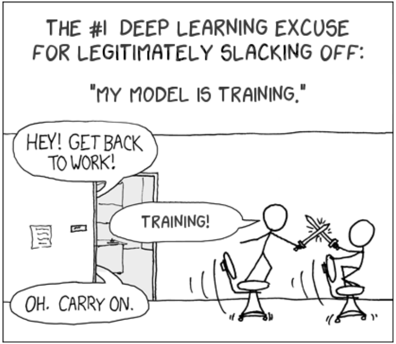
Deep Learning

Krystian Mikolajczyk & Seyed Moosavi

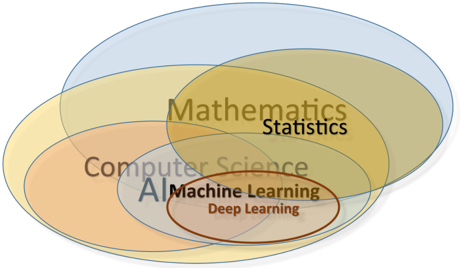
Department of Electrical and Electronic Engineering

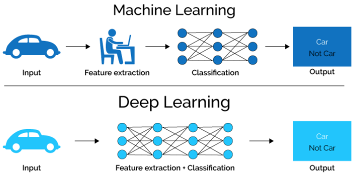
Imperial College London



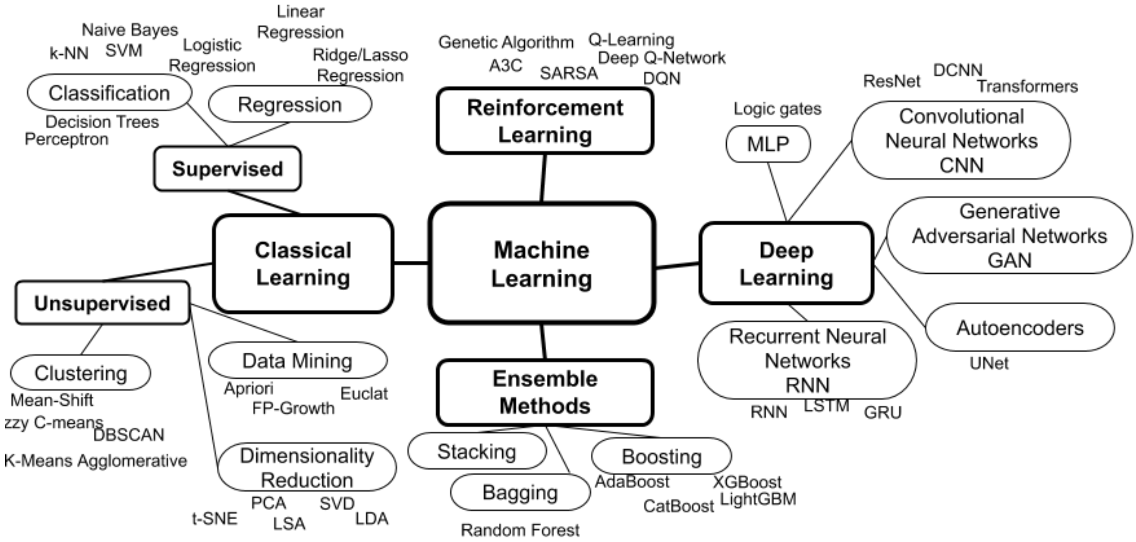
Deep Learning

- AI, Machine Intelligence
  - Intelligent agents with perception and actions to achieve goals
- Machine Learning
  - Ability to learn: Data $\rightarrow$ Hypothesis
- Deep Learning
  - Ability to learn data representation (features) and predictors





DL Summary


Naive Bayes
k-NN
SVM
Logistic Regression
Linear Regression
Ridge/Lasso Regression
Classification
Regression
Decision Trees
Perceptron
Supervised
Genetic Algorithm
A3C
SARSA
Q-Learning
Deep Q-Network
DQN
Reinforcement Learning
Logic gates
MLP
ResNet
DCNN
Transformers
Convolutional Neural Networks CNN
Generative Adversarial Networks GAN
Classical Learning
Machine Learning
Deep Learning
Unsupervised
Autoencoders
UNet
Recurrent Neural Networks RNN
RNN
LSTM
GRU
Clustering
Mean-Shift
zzy C-means
DBSCAN
K-Means
Agglomerative
Data Mining
Apriori
FP-Growth
Euclat
Ensemble Methods
Stacking
Bagging
Random Forest
Boosting
AdaBoost
XGBoost
CatBoost
LightGBM
Dimensionality Reduction
PCA
SVD
t-SNE
LSA
LDA

Goal

- To introduce fundamental principles, theory and approaches for learning with deep neural networks.
- To offer practical course on implementing and experimenting with deep learning.
  - Learn to apply different types of networks in various DL tasks.
- Part of a course on machine learning and related topics:
  - (Autumn): Machine Learning (prerequisite)
  - (Autumn): Maths for Signals and Systems
  - (Spring): Deep Learning
  - (Spring): Advanced Signal Processing
  - (Spring): Computer Vision and Pattern Recognition
  - (Spring): Final Year Project

Course Information

- Lectures: Panopto videos, slides on Blackboard
  - Book: *Deep Learning*, Ian Goodfellow, Yoshua Bengio & Aaron Courville, 2016. http://www.deeplearningbook.org
  - https://towardsdatascience.com
- Weekly practical exercises
  - Self studying on your PC or laptop
  - Colab - online python environment with Keras and TensorFlow backend https://github.com/MatchLab-Imperial/deep-learning-course
    - Colab free account - sufficient for all exercises
    - Colab Pro account - reimbursed for up to 3 months e.g. Feb-March
- Coursework (100%)
  - Work: online exercises from DL github, experiments and reports
  - Assessment: 1 page interim report 20%, Deadline: see Blackboard
  - Assessment: 4 page final report 70%, Deadline: see Blackboard
  - Online quiz 10% in last session.

Course Information

- Prof Krystian Mikolajczyk
  - Office hour: See details on Blackboard
  - Email: k.mikolajczyk@imperial.ac.uk
- Dr Seyed Moosavi
  - Office hour: See details on Blackboard
  - Email: seyed.moosavi@imperial.ac.uk
- Dr Abdalrahman Abu Ebayyeh
  - Office hour: See details on Blackboard
  - Email: a.abu-ebayyeh@imperial.ac.uk
- TAs
  - See details on Blackboard

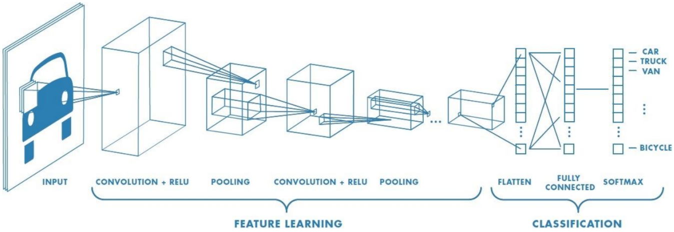
Lectures by Prof Krystian Mikolajczyk

- Part 1: Introduction to deep learning
- Part 2: Convolutional Neural Networks (CNN)
- Part 3: Network Training
- Part 4: CNN architectures
- Part 5: Recurrent Neural Networks

Part 2: Convolutional Neural Networks (CNN)

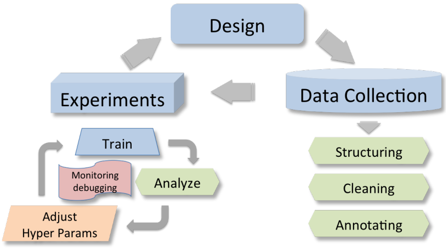
Type of neural network with a special architecture

- Convolution
- Filters, strides, padding
- Pooling
- FC layer
- Loss layer
- Practicals



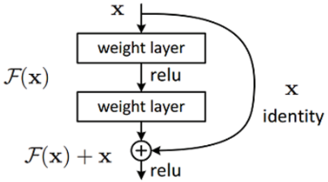
Part 3: Network Training

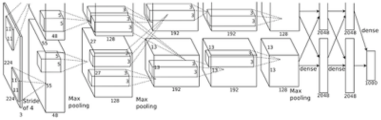
- Backpropagation (reminder)
  - Vanishing and exploding gradients
- Optimizers
  - Nesterov, Adagrad, RMSProp, Adadelta, Adam
- Regularisation
  - Dropout, initialisation, batch normalisation
- Practical development process
  - System design/choice, data collection and augmentation, hyper parameters, monitoring and debugging

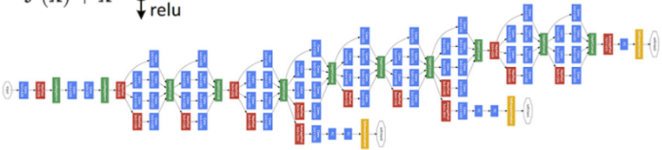


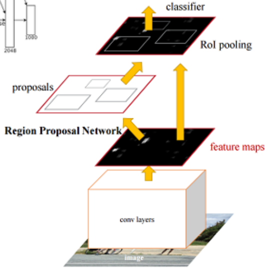
Part 4: CNN architectures

- LeNet, AlexNet, VGG Net, GoogLeNet, ResNet, ResNeXt, Siamese Architecture, Multi-task networks, Transformers



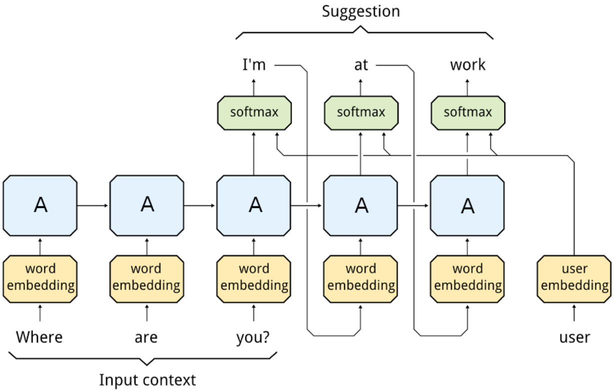






Part 5: Recurrent Neural Networks

- Recurrent Neural Networks
  - Word embedding
- RNN Unit
- LSTM Unit
- GRU Unit
- Architectures & applications



Lectures by Dr Seyed Moosavi

- Part 6: Representation Learning and Autoencoders
- Part 7: Generative models
- Part 8: Reinforcement Learning I
- Part 9: Reinforcement Learning II
- Part 10: Interpretable and Trustworthy Models

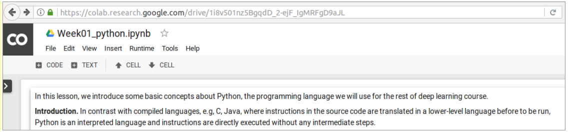
Practical Experiments & Coursework

https://github.com/MatchLab-Imperial/deep-learning-course

- Week 1-2
  - Introduction to Python, Keras and some frameworks (NumPy, Pandas, etc..)
  - Introduction to CNNs
- Week 3-6
  - Fundamentals of deep learning: handling different type of data (text, image, audio, etc), feedforward of artificial neural networks, introduction to last generation of CNN architectures (VGG, Inception, ResNet, UNets etc...)
- Week 7-10
  - Advanced deep learning: LSTM sequence modelling, Generative Adversarial Networks, Neural style transfer (CycleGan, Pix2Pix), Reinforcement Learning

Practical Experiments & Coursework

- Environment: Colaboratory [*1,2]
  - Repository: `https://github.com/MatchLab-Imperial/deep-learning-course`
  - Jupyter notebook environment which requires not setup and supported from most major browsers, e.g, Chrome and Firefox.
  - Code is run in virtual machines with free GPU.
  - Files are stored securely in your own Google Drive account.
  - Supports developing Python applications using popular deep learning libraries, e.g, **Keras**, Tensorflow, Pytorch.



[*1] https://colab.research.google.com/notebooks/welcome.ipynb

[*2] https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d

Practical Experiments & Coursework

- Format: Jupyter [*3]
  - Notebooks are documents produced by the Jupyter Notebook Apps, e.g., Colaboratory, containing both python code and rich text elements (paragraph, equations, figures, links, etc. . .)

```
In [0]: N=5
start_val = 0# pick an element for the code to plot the following N**2 values
fig, axes = plt.subplots(N,N)
for row in range(N):
  for col in range(N):
    idx = start_val+row+N*col

    im = np.concatenate((np.clip(X_test_noise[idx], 0, 1), np.clip(pred[idx], 0, 1)), 1)
    axes[row,col].imshow(im)
    y_target = int(y_train[idx])
    axes[row,col].set_xticks([])
    axes[row,col].set_yticks([])
```

[*3] https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

Practical Experiments & Coursework

- Deep Learning Framework: Keras [*4]
  - modular, minimalist framework, especially good for beginners
  - along with Colab environment allows to set a neural network and start prototyping in no time.

[*4] https://pypi.org/project/Keras/

We listen to your questions, feedback, suggestions

- BlackBoard Q&A forum
- Office hour, Lecturers and GTAs (see BB)
- Emails - risk of getting missed/delayed

And we act on it

- provide more guidance on how to report ML experiments
- introduced Wiseflow test
- moved the CW deadlines forward

ML Summary

Goal

Learning with generalisation

Machine Learning Revision[a]

[a] Lecture slides and videos from Autumn term are available on BlackBoard and Panopto

- Components of Learning
- ML Tasks
- Types of learning
- Types of data
- Learning setup
- Error/Loss Measures
- Perceptron
- Neural Networks
- Gradient descent
- Backpropagation
- Learning curves
- Regularization

Components of learning

- Theory
  - ML problem formulation
  - Errors, loss and bounds
- Predictors and Learners
  - Linear and non linear
  - SVM
  - **Neural Networks**
- Learning frameworks
  - Supervised
  - Reinforcement learning

Domain $\mathcal{D}$ (test/out-of-sample)

Data $\mathbf{x} \in \mathcal{D}$

Target outputs $\mathbf{y} \in \mathcal{Y}$

Target function $f : \mathcal{X} \to \mathcal{Y}$

Experience $\mathcal{X}$ (observations/train/in-sample)

$\mathbf{x} \in \mathcal{X} \subset \mathcal{D}$

$(\mathbf{x}_1, y_1), (\mathbf{x}_n, y_n), \ldots, (\mathbf{x}_n, y_n)$

with generalisation!

Learners

Expertise

Predictors, Models

Hypothesis $g : \mathcal{X} \to \mathcal{Y}$

Learning

Hypothesis class

$\mathcal{H} \subset \{h : \mathcal{X} \to \mathcal{Y}\}$

Learning algorithm

Goal $g \approx f$

Error $g \approx f$

$\{l : (g, \mathcal{Y}) \to \mathbb{R}^+\}$

$\hat{l}[g(\mathbf{w}, \mathbf{x}), f(\mathbf{x})] \in \mathbb{R}^+$

$\widehat{R}_n = \frac{1}{n} \sum_{i=1}^{n} \hat{l}[g(\mathbf{w}, \mathbf{x}), f(\mathbf{x})]$

Error Measures/Loss Functions

- How to quantify $h \approx f$?
- Usually pointwise error: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$

$$\ell(h(\mathbf{x}), f(\mathbf{x}))$$

Defined by the user for the ML task!

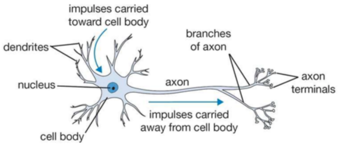
- Examples:

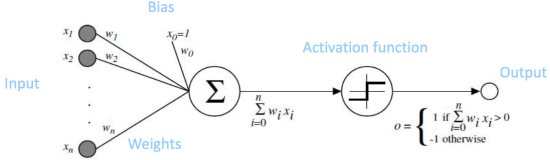
squared error $L2$	$\ell(\hat{y}, y) = (\hat{y} - y)^2$	(regression)
binary error	$\ell(\hat{y}, y) = \mathbb{I}(\hat{y} \neq y)$	(classification)
cross-entropy error	$\ell(\hat{y}, y) = \log\left(1 + e^{-y_i \mathbf{w}^\top \mathbf{x}_i}\right)$	(see logistic regression)

- Training error: $\widehat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(\mathbf{x}_i), y_i)$
- Test error: $R(h) = \mathbb{E}\left[\ell(h(\mathbf{x}), y)\right]$

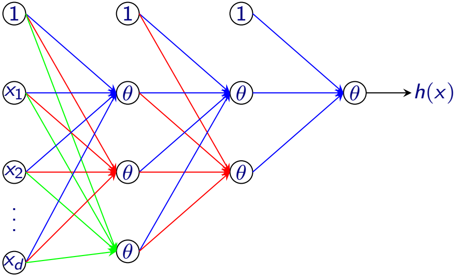
Neural Network: Perceptron

- Binary class perceptron
  - biologically inspired model of a single neuron

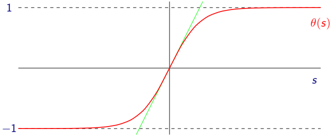




Neural Network



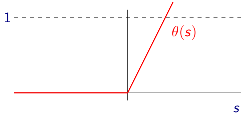
Linear vs. Non-linear activation function $\theta(s)$

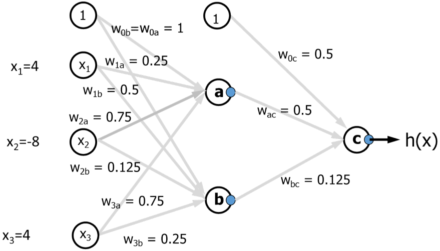


if $\theta(s) = s$ then $h(x) = \mathbf{w}_L^\top W_{L-1} W_{L-2} \ldots W_1 \mathbf{x} = \mathbf{w}_*^\top \mathbf{x}$

Neural Network Forward Pass (inference)

- Input $\mathbf{x} = (4, -8, 4)$
- Non linear activation ReLU $\theta(s) = s_+ = \max\{0, s\}$.





- $f = \max(0, \sum_i w_i x_i)$
- $f_a = \max(0, 1 + 0.25 \cdot 4 + 0.75 \cdot (-8) + 0.75 \cdot 4) = 0$
- $f_b = \max(0, 1 + 0.5 \cdot 4 + 0.125 \cdot (-8) + 0.25 \cdot 4) = 3$
- $f_c = h(\mathbf{x}) = \max(0, 0.5 + 0.5 \cdot 0 + 0.125 \cdot 3) = 0.875$
- Ground truth $y = 2$
- Error $\widehat{R}_n(\mathbf{w}_t) = (y - h(\mathbf{x}))^2 = (2 - 0.875)^2 \approx 1.27$

Learning Setup with $\mathbf{x} \sim P$ and $P(y|\mathbf{x})$

- Input: $\mathbf{x} \in \mathcal{X}$
- Output: $y \in \mathcal{Y}$
- Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n) \sim P$ (P is the joint distribution of $(\mathbf{x}, y)$)

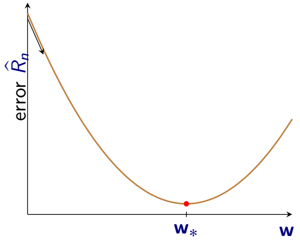
Learning

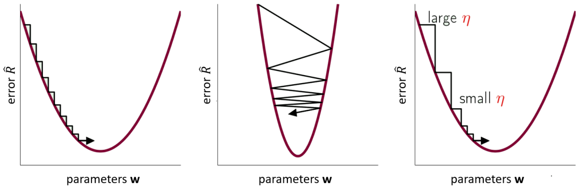
- Hypothesis class: $\mathcal{H} \subset \{h(\mathbf{w}) : \mathcal{X} \to \mathcal{Y}, w \in \mathbb{R}\}$
- Loss function: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$
- Find $\mathbf{w}^* \in \mathbb{R}^{\|\mathbf{w}\|}$ such that $g \approx P(y|\mathbf{x})$

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \left\{ R(\mathbf{w}) = \mathbb{E}\left[\ell(h(\mathbf{wx}), y)\right] \right\}$$

Gradient descent

- General method for non-linear optimization: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \mathbf{v}$
- Direction $\mathbf{v}$: starting from $\mathbf{w}_t$, step along the steepest slope
  - $\mathbf{v} = -\frac{\nabla \widehat{R}_n(\mathbf{w}_t)}{\|\nabla \widehat{R}_n(\mathbf{w}_t)\|}$ is a unit vector.
- Step size η: how quickly find the minimum
  - η is a scalar.





η is small $\qquad$ η too large $\qquad$ variable η

Heuristic: step size should increase with the slope $\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w}$

$$\Delta \mathbf{w} = -\eta \nabla \widehat{R}_n(\mathbf{w}_t) \text{ (with } \eta \text{ redefined)}$$

Neural Network Backpropagation (training)

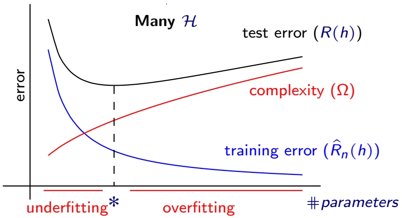
Backpropagation

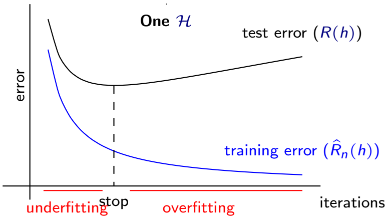
- Intialize all weights $w_{ij}^{(l)}$ at random
- for $t = 1, 2, \ldots$ do
  - Pick a data point (x_k, y_k)
  - **Forward:** Compute all $x_j^{(l)}$
  - **Backward:** Compute all $\delta_j^{(l)}$
  - **Update:** $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta_t x_i^{(l-1)} \delta_j^{(l)}$
    - single point (SGD), minibatch, batch
- Return final weights $w_{ij}^{(l)}$

Learning from noisy data

Overfitting

- Fitting to noise instead of the underlying target function/distribution.
- Occurs when $\widehat{R}_n(h) \downarrow \quad R(h) \uparrow$, moving away from the target function.





Remember n matters too!

Noise types: stochastic ($N \sim \mathcal{N}(0, \sigma^2)$), deterministic (complexity)

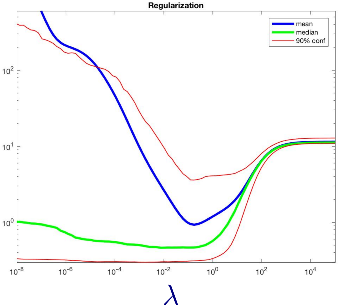
deterministic noise	$\uparrow$	overfitting	$\uparrow$
stochastic noise	$\uparrow$	overfitting	$\uparrow$
number of data points	$\uparrow$	overfitting	$\downarrow$

Regularisation

Regularised Loss (Augmented Error)

$$\mathcal{L}_n(h) = \widehat{R}(h) + \lambda \underbrace{\Omega(h)}_{\textit{overfit penalty}}$$

- $\Omega(h)$ – regulariser with parameter λ
  - $\|\mathbf{w}\|_2^2$: L2
  - $\|\mathbf{w}\|_1$: L1
  - $\|\mathbf{w}\|_1 + \|\mathbf{w}\|_2^2$: elastic net
  - $\mathbf{w}^\top \Gamma^\top \Gamma \mathbf{w}$: Tikhonov



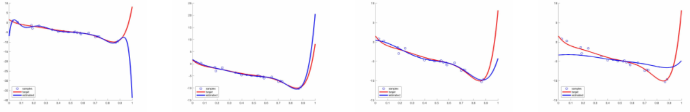
Occam's razor: Simpler is usually better

Regularize towards smoother, simpler functions. Why?

Because noise is not smooth!

Regularized Loss Minimization

$\lambda = 0$ $\lambda = 0.0001$ $\lambda = 1$ $\lambda = 100$



overfitting $\longrightarrow$ $\longrightarrow$ underfitting

Regularized Loss Minimization (RLM)

- Hypothesis class $\mathcal{H} = \cup_i(\mathcal{H}_i, \lambda_i)$, with $i \in \mathbb{N}$ e.g. $\lambda_i \in \{0.0001, 0.001, \ldots\}$
- Augmented error:

$$\mathcal{L}_{\bar{k}}(\mathbf{w}, \lambda) = \widehat{R}_{\bar{k}}(\mathbf{w}) + \lambda\Omega(\mathbf{w}) \qquad \text{e.g. } \Omega(\mathbf{w}) \in \{\|\mathbf{w}\|_2^2, \|\mathbf{w}\|_1, \|\mathbf{w}\|_2^2 + \|\mathbf{w}\|_1, \ldots\}$$

- RLM solution:
  - for all i, **train** on $\mathcal{D}_{\bar{k}}$: $\quad g(\mathbf{w}_{\lambda_i}) = \text{argmin}_{\mathbf{w}} \mathcal{L}_{\bar{k}}(\mathbf{w}, \lambda_i)$
  - from all i, **select** on $\mathcal{D}_k$: $\quad g(\mathbf{w}_{\lambda*}) = \text{argmin}_{\lambda_i} \check{R}_k(g(\mathbf{w}_{\lambda_i}))$

Summary

- Organization of DL course
- Exercises and coursework
- Revision of fundamentals of Machine Learning