Data Science - Module 5

Machine Learning
Supervised Learning

Andrea Cossu – andrea.cossu@sns.it
Scuola Normale Superiore
University of Pisa

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

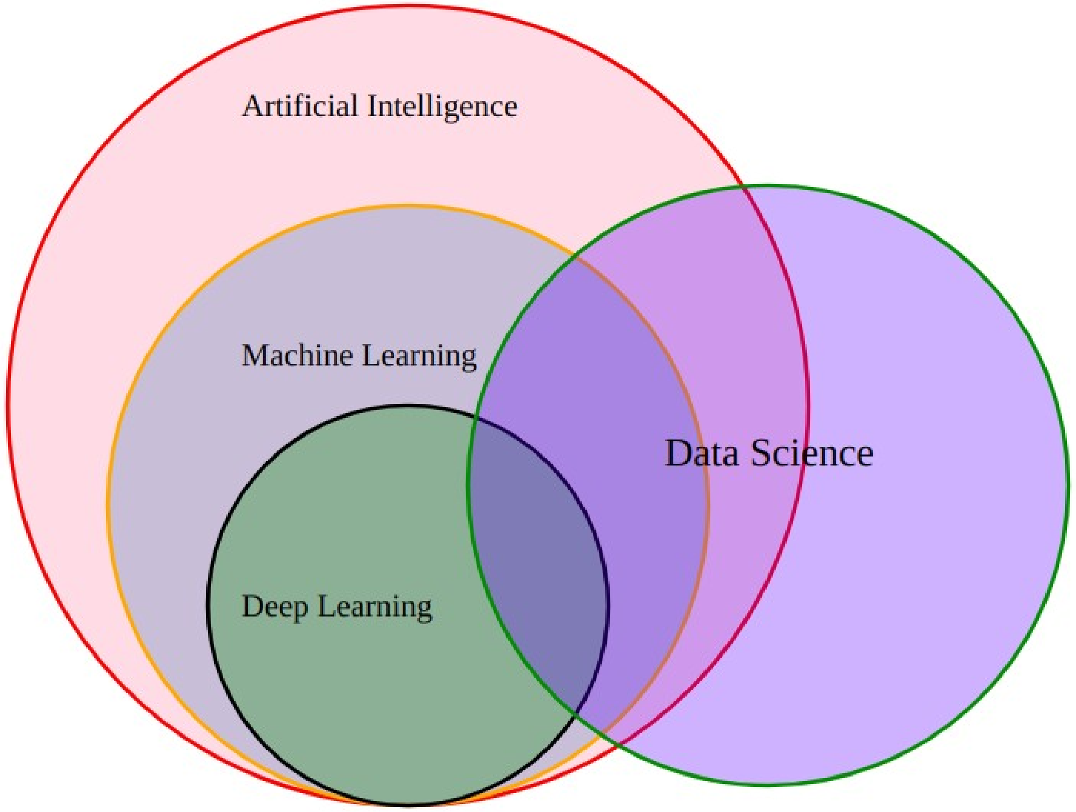
Lab Outline

- Machine / Deep Learning frameworks
- Perceptron from scratch with PyTorch
- Beyond perceptrons with MultiLayer Perceptron
  - Autograd feature of PyTorch
- Metrics (accuracy, F1, ROC curve)
- Tensorboard
- Optional: Quick overview of Keras

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

The AI family



https://jatogayama.blogspot.com/2020/05/ai-ml-dl-venn-diagram.html

From Artificial Intelligence to Deep Learning

- Good Old Fashioned Artificial Intelligence – GOFAI
  - Symbolic approach
  - Knowledge base, expert systems, rule-based decision support systems
  - **Symbol grounding problem**
    The problem is analogous to trying to learn Chinese from a Chinese/Chinese dictionary alone
    [Stevan Harnad, The symbol grounding problem, Physica D: Nonlinear Phenomena, Volume 42, Issues 1–3, June 1990, Pages 335-346]

From Artificial Intelligence to Deep Learning

- Machine Learning (ML) *[Tom Mitchell, Machine Learning, McGraw Hill, 1997]*
  - A ML model improves over task T with respect to performance measure P based on experience E
  - Experience E is a set of *data features*
  - ML model extracts patterns from features
  - The model must *generalize* well to unseen experience

From Artificial Intelligence to Deep Learning

- Deep Learning (DL) *[Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016]*
  - A DL model “understands” the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts.
  - The hierarchy of concepts is *deep*
  - Features are created hierarchically from raw data
  - The model must *generalize* well to unseen experience

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

Supervised Learning ingredients

- Data → a set of patterns: (features, **target**) = (input, **desired output**)
- Model (hypothesis) → defines the search space
- Learning algorithm → navigate the search space to find the optimal solution to the…
- … Task → classification, regression, …
- Empirical Risk Minimization $\mathbb{E}[\mathcal{L}(target, output)]$

- Learning is supervised due to the **target** information provided to the model for each input pattern

Ok… What??
Iris example

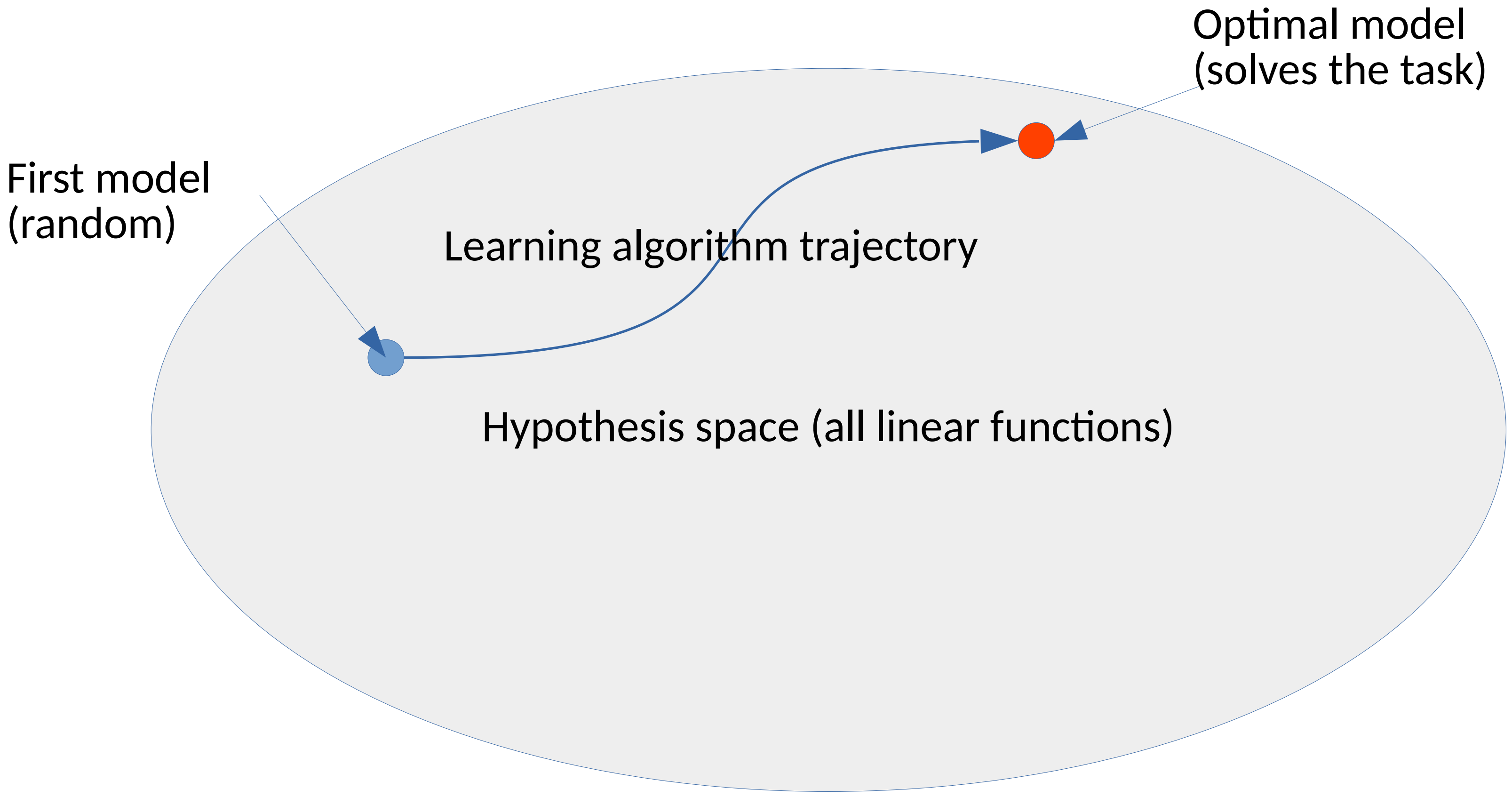
Let's clarify!

- Data → Iris dataset, a set of
  features=(sepal length/width, petal length/width),
  target=flower class
- Model (hypothesis) → you choose this! Suppose logistic regression.
- Learning algorithm → gradient descent
- Task → correctly classify each flower based on its features

Interaction between model and learning algorithm

- The model defines the search (hypothesis) space.
  - *INDUCTIVE BIAS: set of assumptions used to guide learning process*
- Logistic regression restricts search space to linear relationships between features and target
  - A learning algorithm will be able to search within *that* search space
- Neural networks are a more expressive model which takes into account nonlinear relationships
  - The *same* learning algorithm will be able to search within *that* search space
- VERY DIFFERENT RESULTS!

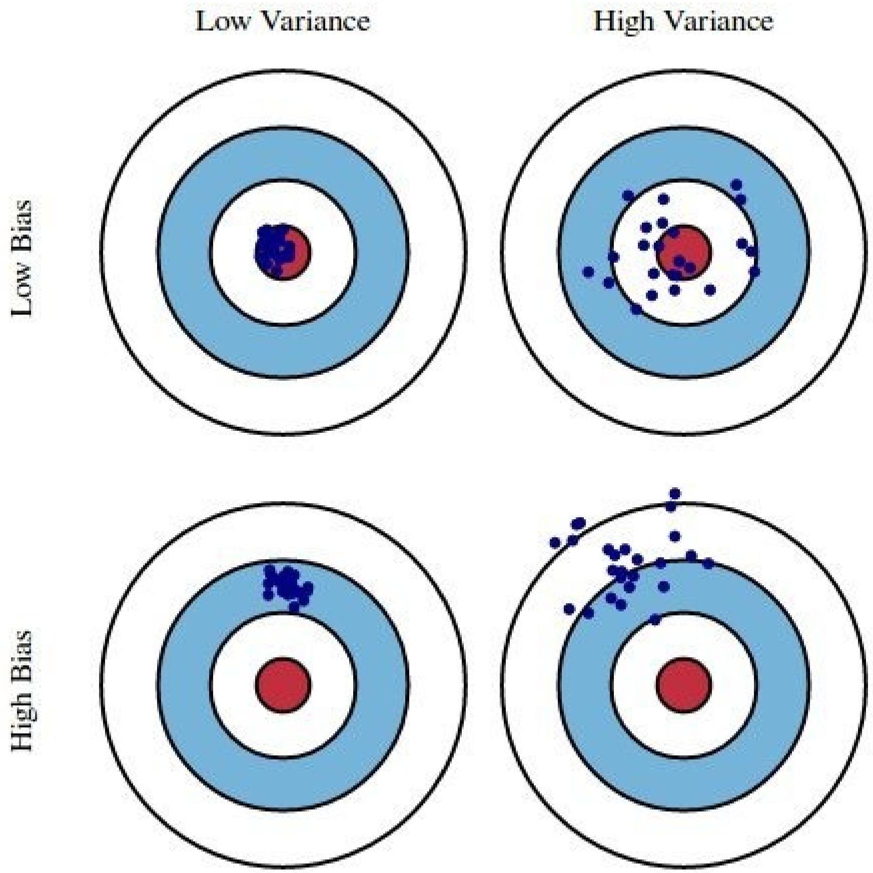
Search in the hypothesis space


Optimal model
(solves the task)
First model
(random)
Learning algorithm trajectory
Hypothesis space (all linear functions)

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

Bias Variance tradeoff intuitively



https://www.kdnuggets.com/wp-content/uploads/bias-and-variance.jpg

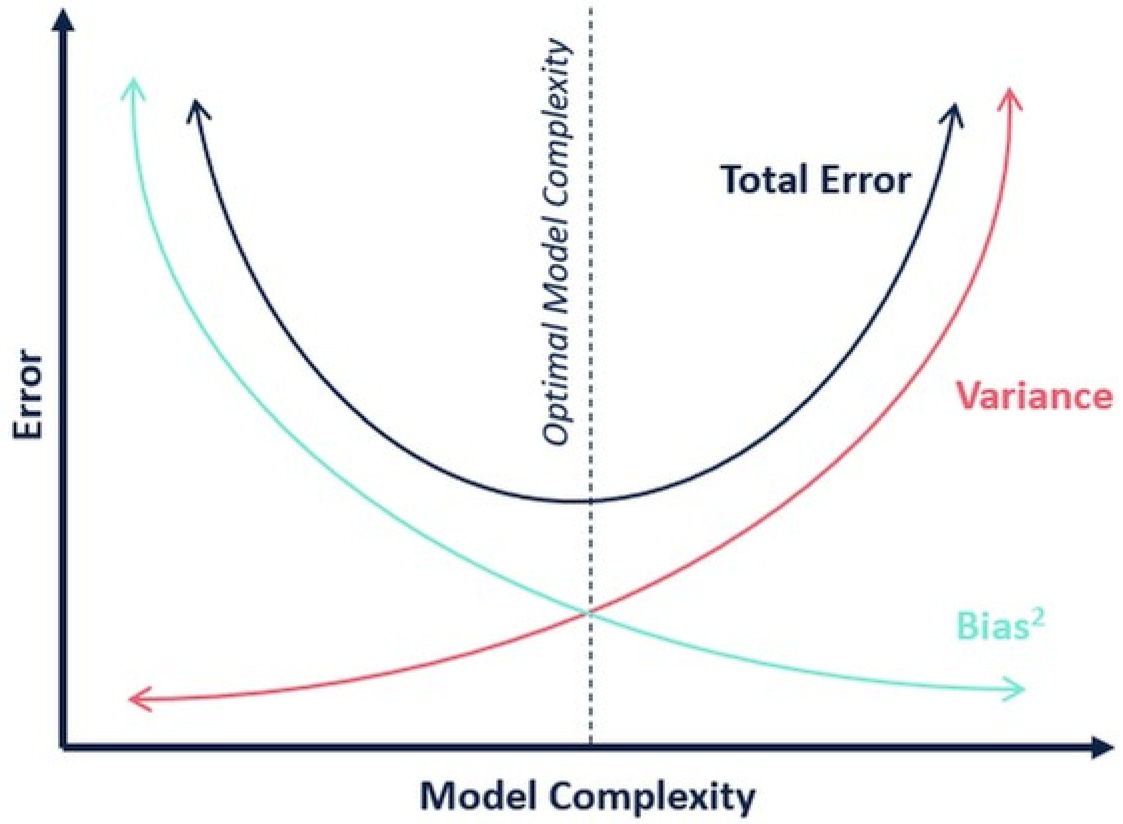
Bias Variance decomposition

- True function: $Y = f(x) + \epsilon$
- Mean Square Error (can be used as loss function in ERM)

  $$\mathbb{E}[(Y - \hat{f}(x))^2]$$

- MSE can be decomposed in Bias^2 + Variance + Irreducible error
  - Bias: $\mathbb{E}[\hat{f}(x) - f(x)]$
  - Variance: $\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$
  - Irreducible error: constant term

ERM in light of Bias Variance



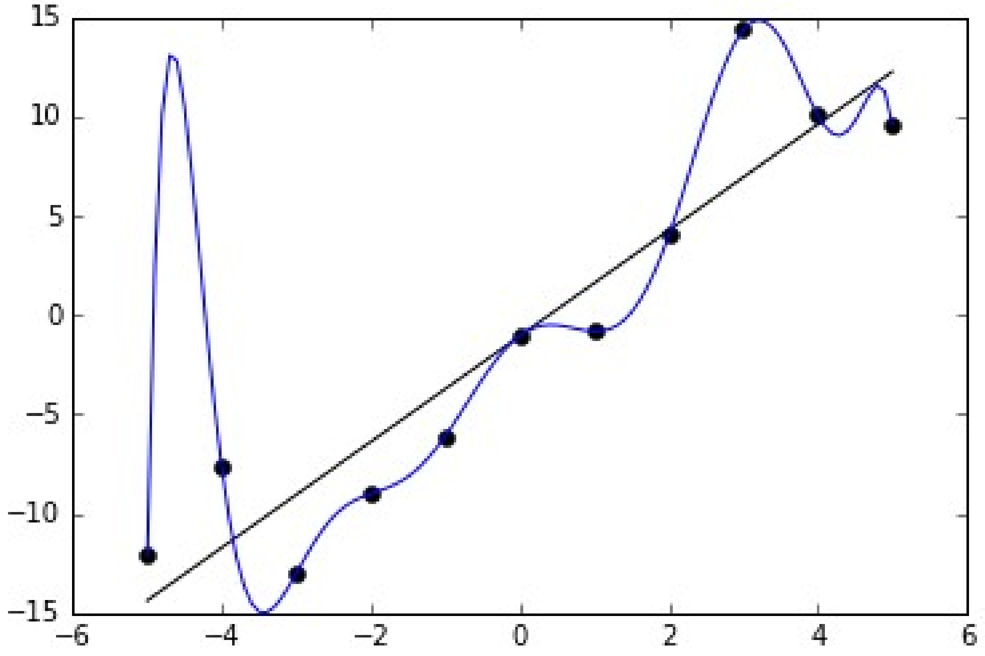
https://ai-pool.com/a/s/bias-variance-tradeoff-in-machine-learning

Tweak Bias and Variance in a model

- Large bias means restricting the hypothesis space
- Large variance means allowing for greater flexibility
- Reduce variance → increase bias (and viceversa)
- The more expressive a model is, the less the bias, the more the variance
- Linear regression → large bias, small variance
- Neural networks → small bias, large variance

Overfitting vs underfitting

- Current trend is to have large variance model
- Problem:



Possible solutions

- Avoid underfitting? Easy $\rightarrow$ use a more expressive model
- Avoid overfitting? Difficult!
  - constrain (i.e. regularize) your expressive model (prevent polynomial coefficients > 5)
  - use more data (large datasets require the use of expressive models, if the task is hard)

Lecture outline

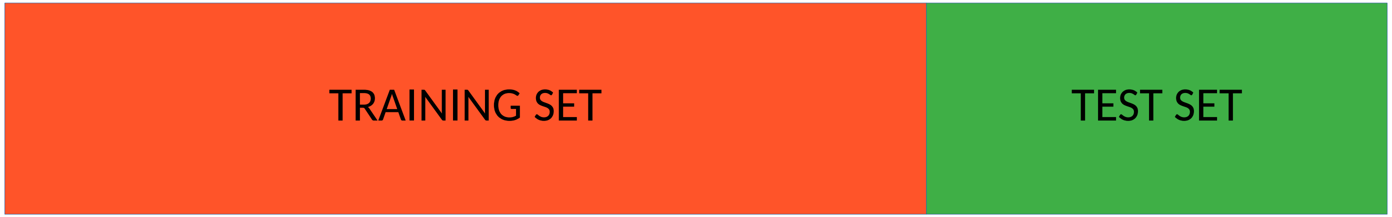
- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

Overfitting the data, a different one

- The objective of supervised learning is to find a model which performs well on *unseen* data (generalization)
- If you *train* a model on a dataset, what can you say about its generalization capabilities?
- Training ≈ fitting
  - What if your dataset is not representative of the true distribution?
  - What if your model becomes overconfident on the dataset?
  - Overfitting! → decrease in performances on *held-out* test set

Split 1 – Train Test

- Fit your model on the *training* set and test it on the *test* set

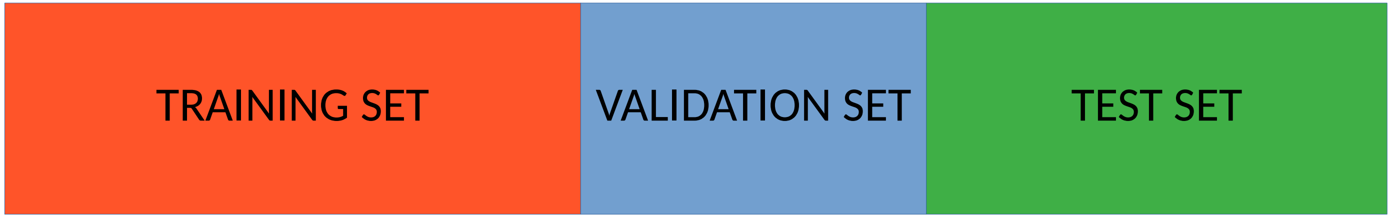


- If performance on test set << performance on training set… Overfitting!
- The performance of your model **must** be estimated on a separate test set.
- You cannot access, see, smell, hear, touch or even think about test set until you selected your final model. Otherwise your final evaluation will not be reliable.

Choose the best

- Select the best model for the current task
- Say, you create 10 different models by training them on the training set
- Which one should you use in your application?
- Test the performances of each of them on the test set and select the best → NEVER DO THAT
- You are overfitting on the test set!
- We need another split.

Split 2 – Train Validation Test



- Train N different models on the training set
- Choose the best based on the performance on the validation set
- Evaluate the best model on the test set to get generalization estimate

Model selection vs model assessment

- *Model selection*: among N models, choose the best one
- *Model assessment*: given one model, estimate its performance on unseen data
  - The final model can be retrained on training + validation set
- Pseudocode
  - For each model:
    - Use TRAINSET to train separately N models
    - Measure performance on VALSET
  - Select best model according to performance on VALSET
  - Optional: Retrain best model on TRAINSET + VALSET
  - Test generalization performance of best model on TESTSET

Bonus track: K-fold Cross validation

- Used for model selection → get the test set and throw it far, far away
- Split the dataset into K equally-sized parts (folds)
- Use K-1 folds for training and 1 for validation → repeat K times by changing validation fold
- Compute avg and std performance on the K runs and select the best model, then test it on the held-out test set
- Used for model assessment → use K-1 folds for training, 1 for test
  - Obtain avg and std performance for generalization

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

Linear regression for supervised learning

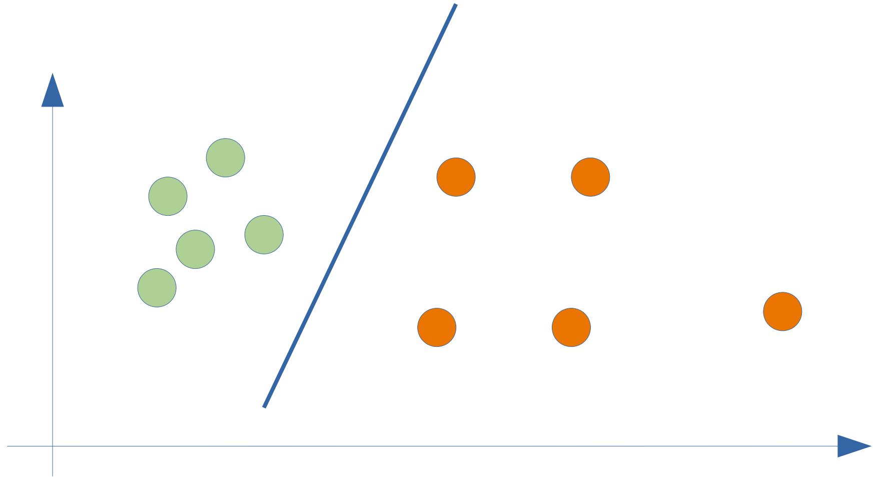
- A useful form: $\hat{y} = wx + b$
- One-shot solution with pseudoinverse (still, supervised)

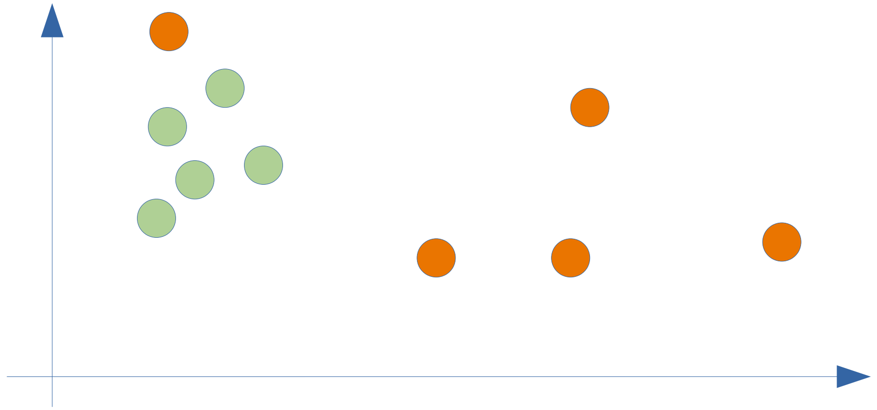
$$w = (X^T X)^{-1} X^T y = X^\dagger y$$

- or… gradient descent: $w = \text{argmin}_w \mathbb{E}[(wx + b - y)^2] = \text{argmin}_w \mathbb{E}[\mathcal{L}(\hat{y}, y)]$
  - Initialize *w* randomly
  - Compute $\nabla_w \mathcal{L}(\hat{y}, y)$
  - Update $w = w - \eta \nabla_w \mathcal{L}(\hat{y}, y)$
  - … until convergence

From linear regression to Perceptron

- A stepping stone towards modern neural networks
- Classify a set of points in N-dimensional space (N features)
- Binary classification (-1, 1)
- Classify *linearly separable* points

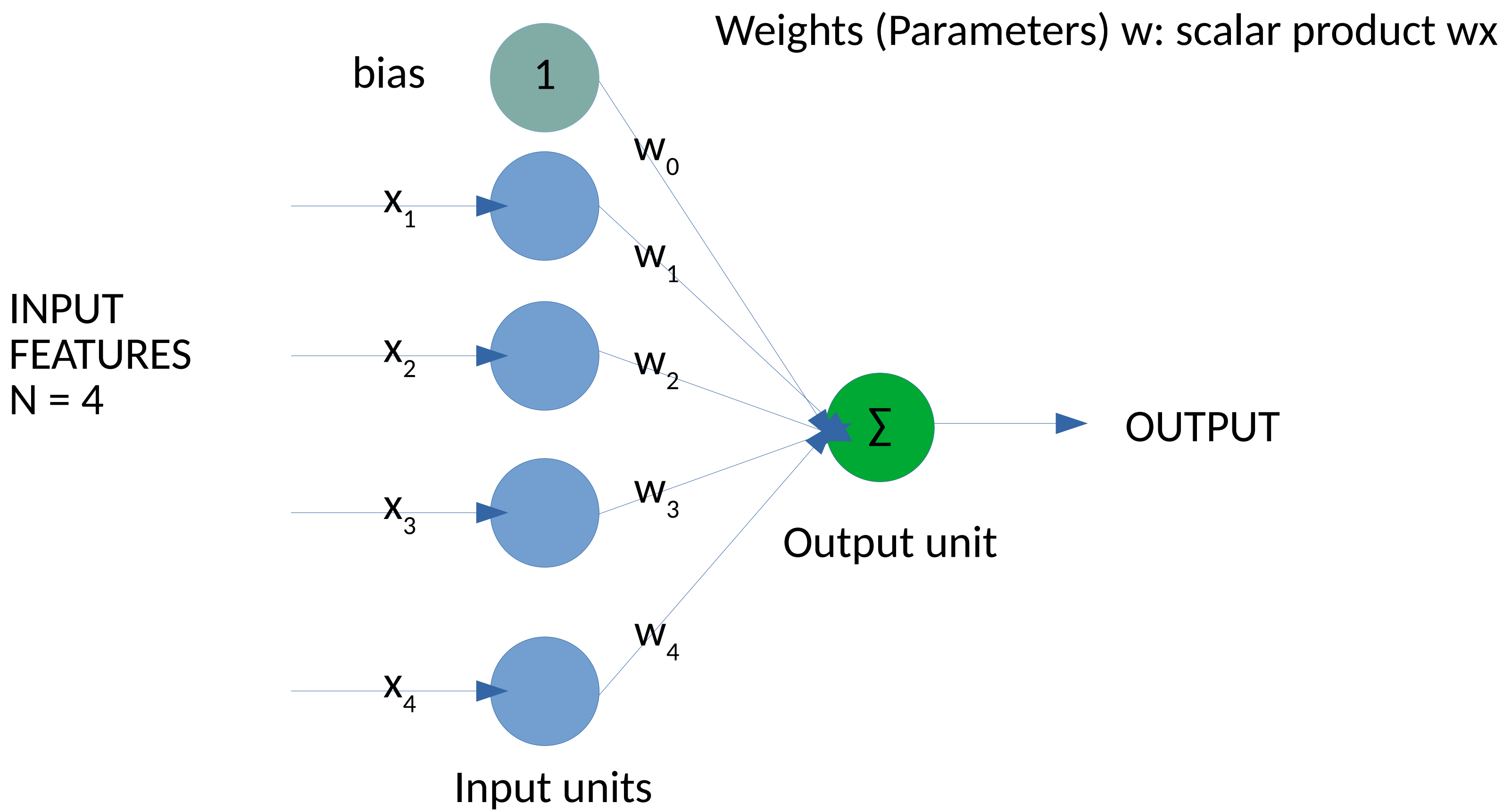




Perceptron in equations

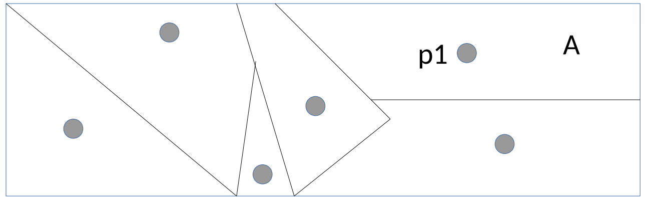
- Perceptron = Linear Threshold Unit (LTU): $\hat{y} = wx + b$
- Initialize randomly *w*, *b* (bias here has nothing to do with inductive bias or bias-variance)
- For each input pattern x_p (epoch):
  - Compute sign(wx_p+b)
  - Update $w_i = w_i + \eta(y_p - \hat{y}_p)x_{p,i} \quad \forall i \in [1, N]$
- If average error over epoch < threshold → stop
- Convergence is guaranteed if patterns are linearly separable!
- Update rule:
  - If target > output → move towards +
  - If target < output → move towards -

Perceptron as a single neuron (unit)



Bonus track: K Nearest Neighbours (K-NN)

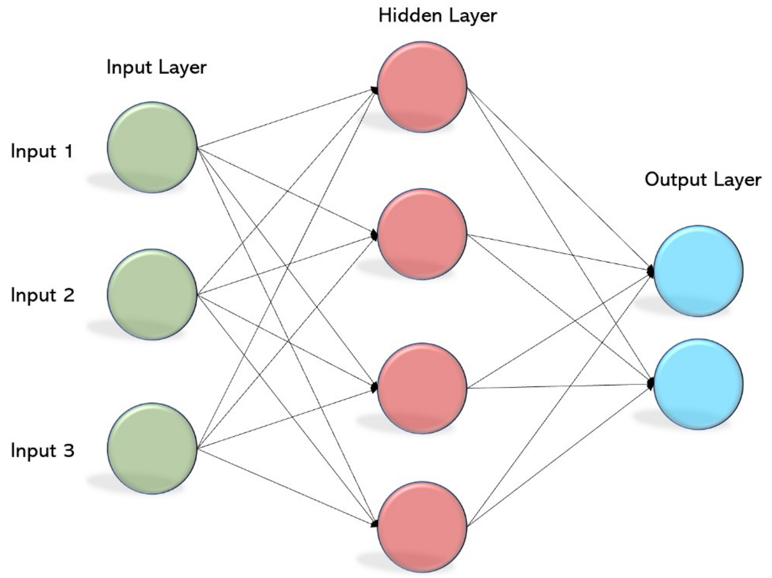
- A very simple form of supervised learning
- No training phase → just collect data
- Predict a novel input
  - look at the closest K points
  - Voting: take the class of the majority (or the average value for regression)
- **Distance-based method** $d(x_1, x_2) = \sqrt{\sum_{i=1}^{N} (x_{1,i} - x_{2,i})^2} = \|x_1 - x_2\|$



Voronoi diagram:
any point in A is closer to p1 than
to any other point outside A

MultiLayer Perceptron (MLP) or feedforward network

- Yes, a neural network, at last!
- It is a densely connected network of perceptron-like units, stacked in layers

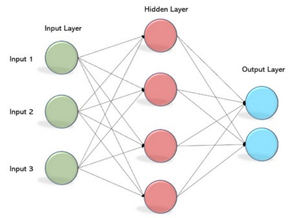


https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f

MLP components

$$h^1 = f(W^1 x + b^1)$$
$$h^2 = f(W^2 h^1 + b^2)$$
$$\hat{y} = g(W^3 h^2 + b^3)$$

- f is an *activation function*
  - it is **non linear**
  - MLP separates also points which are not linearly separable
  - multiple linear layers = single linear layer → need f nonlinear
- W is a **matrix,** b is a **vector**
  - W^1_{ij} is the weight from unit *i (layer 0)* to unit *j (layer 1)*
  - W^2_{ij} is the weight from unit *i* (layer 1) to unit *j* (layer 2)
  - 3rd row of W^1 is the vector of weights from unit *3* (layer 0) to all units of layer 1



MLP is a function approximator

- One hidden layer is sufficient to approximate any continuous function
  - However, the number of units grows exponentially → stack more layers!
- What function does MLP approximate? Well…

$$\hat{y} = g(W^3 f(W^2 f(W^1 x + b^1) + b^2) + b^3)$$

- (W, b) are the adaptive parameters (weights) that needs to be tuned on the dataset

Activation functions

- Rectified Linear Unit (ReLU): $max(0, x)$
- Hyperbolic tangent (tanh): $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Sigmoid (logistic): $\frac{1}{1 + e^{-x}}$
- Softmax: $\frac{e^{x_i}}{\sum_k e^{x_k}} \quad \forall i$

Loss functions

- Mean Squared Error (regression): $\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$

- Cross Entropy (classification): $-\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i$

Bonus: L2 regularization

- MLP are very prone to overfitting, since they can approximate any continuous function
- If you don’t have enough data you will overfit
- L2 regularization prevents weights from taking large values
- Modified loss function
  - $w = \text{argmin}_w \; \mathcal{L}(\hat{y}, y) + \lambda ||w||^2$
  - Cfr. Tykhonov regularization, ridge regression
- If weights explode, total loss goes up.

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

Training a neural network: SGD

- Similar to what happens for linear regression
- Compute gradients of the loss with respect to all parameters
- Update each parameter in the direction of negative gradient (component-wise)
  - More complicated optimization techniques exist (and work better).
- Problem: how to compute gradient?

Solution: backpropagation

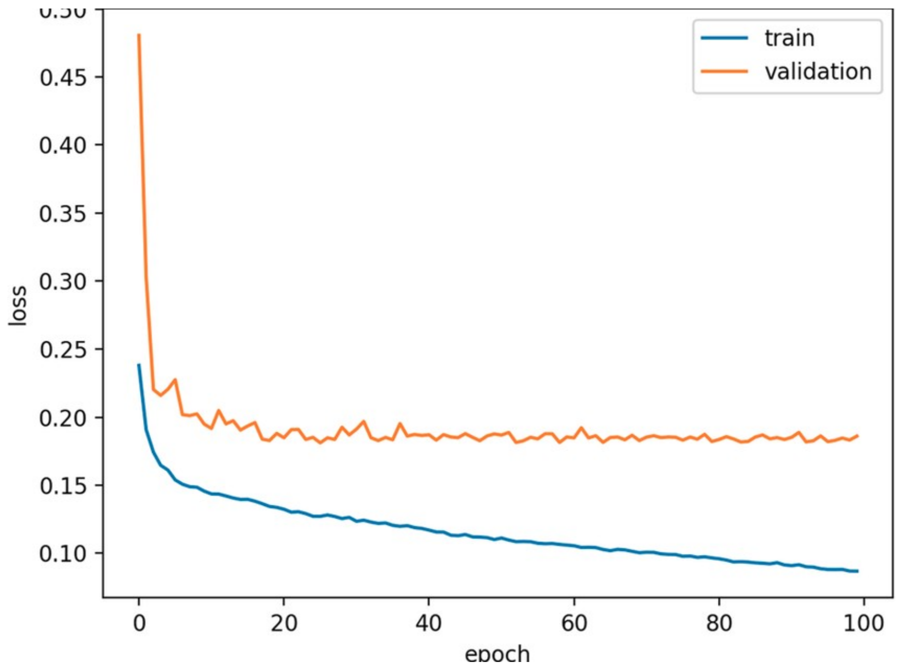
- A strict derivation allows to compute the gradient for each parameter in a MLP.
- We will see how this is done automatically with ML frameworks in Python
- Idea
  - Start from last (output) layer
  - Compute the derivative there (it’s easy!)
  - Use the derivative computed at the output layer to get the derivative at the lower layer
  - Use that derivative to compute the derivative at the lower layer
  - And so on down to the first layer. *Propagate* information *back*wards!

Lecture outline

- Machine Learning in a nutshell
- Supervised learning framework
- Bias Variance tradeoff
- Model selection vs. model assessment
- Supervised models
  - Linear regression (heating up)
  - Perceptron
  - Multi-Layer Perceptron
- How to train neural networks
- How to monitor performances

How to monitor overfitting?

- Plot performance on training and validation set



https://mc.ai/training-on-detectron2-with-a-validation-set-and-plot-loss-on-it-to-avoid-overfitting/

Accuracy

- *When* to monitor: usually, at the end of each epoch compute average on train and on validation set

- Accuracy (for classification): percentage of correctly classified patterns
- There is no equivalent for regression, the MSE is usually a good measure

Confusion matrix

- True Positive → # correctly classified as 1
- True Negative → # correctly classified as 0
- False Positive → # misclassified as 1 (it was 0)
- False Negative → # misclassified as 0 (it was 1)

Accuracy in terms of TP, TN, FP, FN?

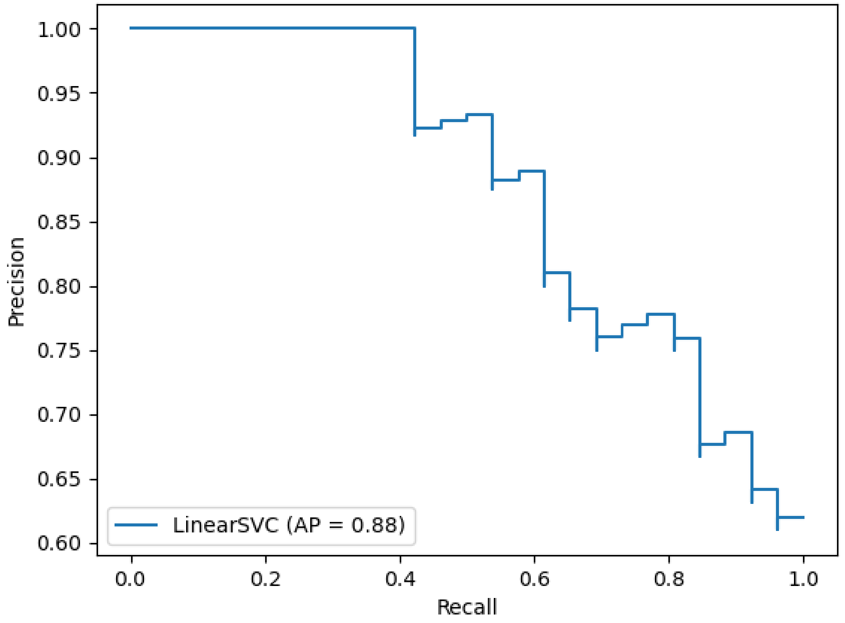
True target → Predicted ↓	**1**	**0**
1	*5 (TP)*	*2 (FP)*
0	*3 (FN)*	*3 (TN)*

F1 score (a single number!)

- Precision $\frac{TP}{TP + FP}$ % detected positives among all detected positives
- Recall Sensitivity $\frac{TP}{TP + FN}$ % detected positive among all real positives
- F1 Score: harmonic mean of P and R = $2\frac{PR}{P + R}$ = $\frac{TP}{TP + 0.5(FP + FN)}$
- Specificity $\frac{TN}{TN + FP}$ % detected negatives among all real negatives

Precision Recall curve

- What is P and R if I vary the threshold which tells apart 0s from 1s?
  - Standard threshold: 0.5
- Area Under the Curve (AUC)
  - the larger, the better
  - equals accuracy
- Random = horizontal line at % positive labels
- Used for class-imbalanced problems
  - e.g. Information retrieval, credit-card fraud
  - Focus on positive class
    (look at numerator of P and R)



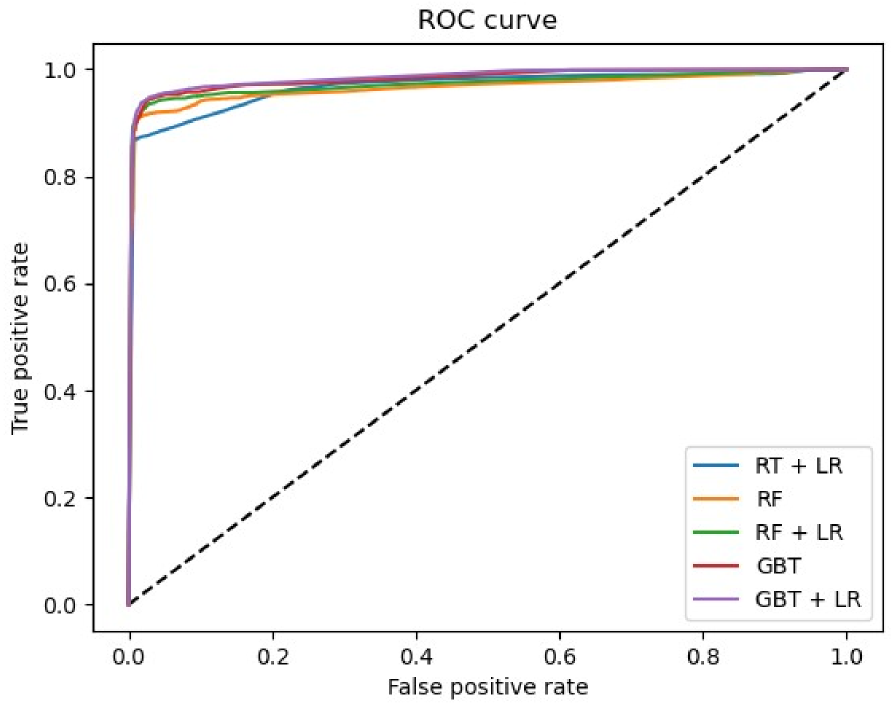
https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py

ROC Curve

- AUC → same meaning as in P-R curve
- TPR = Recall / Sensitivity

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

- Not useful when dataset has many more negative samples (use P-R)
- Use ROC when dataset is balanced



Hidden bonus: Support Vector Machine

- Classifier (SVR for regression)
- Often a strong baseline against which compare performances
- Requires less data than neural networks
- It is a linear classifier in the *vanilla* version
  - Can be made non linear with kernels (functions projecting data into a higher dimensional feature space)
- Matematically involved to describe
- Uses only a limited number of support vectors (i.e. data points) which lie near the decision boundary to discriminate new patterns
- Effective also when #dimensions > #patterns (due to the support vectors)