Lectures on Machine Learning

Lecture 2: from non-linear models to hyperparameter tune

Stefano Carrazza

TAE2023, 11-12 September 2023

University of Milan and INFN Milan (UNIMI)

Outline

Lecture 1 (yesterday)

- Artificial intelligence
- Machine learning
- Model representation
- Metrics
- Parameter learning

Lecture 2 (today)

- Non-linear models
- Beyond neural networks
- Clustering
- Cross-validation
- Hyperparameter tune

Artificial neural networks

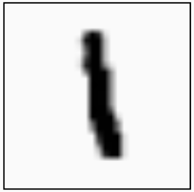
Limitations of linear models

Why not linear models everywhere?

Limitations of linear models

Why not linear models everywhere?

Example: consider 1 image from the MNIST database:



$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

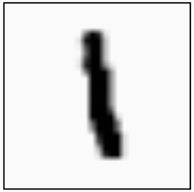
Each image has 28x28 pixels = 785 features (x3 if including RGB colors).

If consider quadratic function $\mathcal{O}(n^2)$ so linear models are impractical.

Limitations of linear models

Why not linear models everywhere?

Example: consider 1 image from the MNIST database:



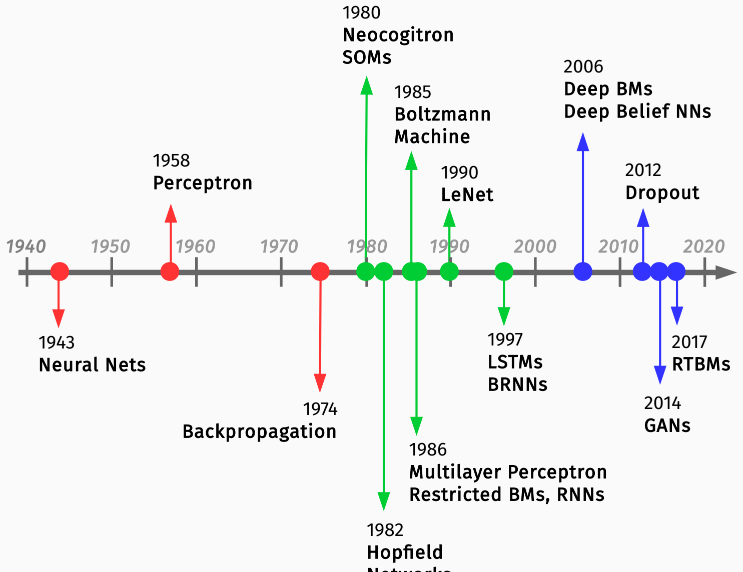
$$\simeq \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Each image has 28x28 pixels = 785 features (x3 if including RGB colors).

If consider quadratic function $\mathcal{O}(n^2)$ so linear models are impractical.

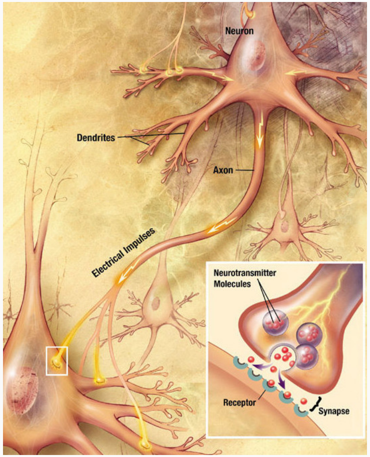
Solution: use non-linear models.

Non-linear models timeline



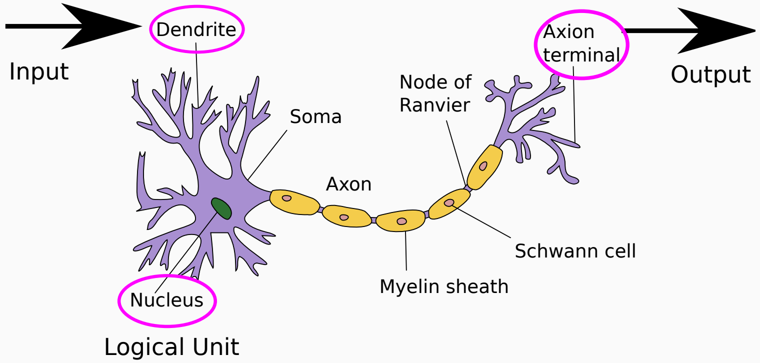
Neural networks

Artificial neural networks are computer systems inspired by the biological neural networks in the brain.



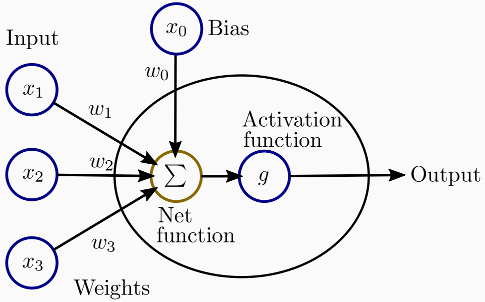
Neuron model

We can imagine the following data communication pattern:



Neuron model

Schematically:



where

- each **node** has an associate weights and bias w and inputs x,
- the output is modulated by an **activation function**, g.

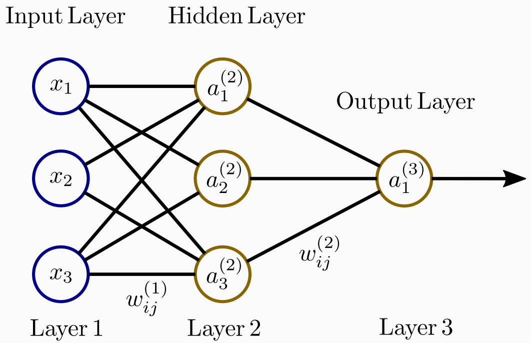
Some examples of activation functions: sigmoid, tanh, linear, ...

$$g_w(x) = \frac{1}{1 + e^{-w^T x}}, \quad \tanh(w^T x), \quad x.$$

Neural networks

In practice, we simplify the bias term with $x_0 = 1$.

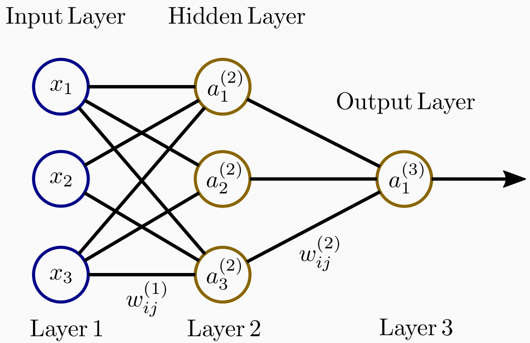
Neural network $\rightarrow$ connecting multiple units together.



where

- $a_i^{(l)}$ is the activation of unit i in layer l,
- $w_{ij}^{(l)}$ is the weight between nodes i, j from layers $l, l+1$ respectively.

Neural networks

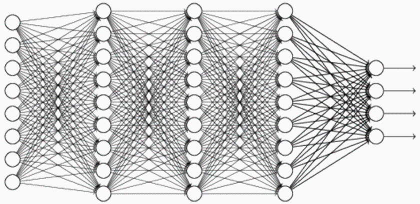


- $a_1^{(2)} = g(w_{10}^{(1)} + w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3)$
- $a_2^{(2)} = g(w_{20}^{(1)} + w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3)$
- $a_3^{(2)} = g(w_{30}^{(1)} + w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3)$
- **Output** $\rightarrow a_1^{(3)} = g(w_{10}^{(2)} + w_{11}^{(2)} a_1^{(2)} + w_{12}^{(2)} a_2^{(2)} + w_{13}^{(2)} a_3^{(2)})$

Neural networks

Some useful names:

- Feedforward neural network: no cyclic connections between nodes from the same layer (previous example).
- Multilayer perceptron (MLP): is a feedforward neural network with at least 3 layers.
- Deep neural networks: term referring to neural networks with more than one hidden layer.



Training neural networks

The training NNs is usually performed with gradient descent methods.

Following the previous section, we have to compute the cost function gradient with respect to parameters $w_{ij}^{(l)}$:

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \nabla_{ij}^{(l)} J \quad \rightarrow \quad \nabla_{ij}^{(l)} J = \frac{\partial}{\partial w_{ij}^{(l)}} J(\boldsymbol{w})$$

Training neural networks

The training NNs is usually performed with gradient descent methods.

Following the previous section, we have to compute the cost function gradient with respect to parameters $w_{ij}^{(l)}$:

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \nabla_{ij}^{(l)} J \quad \rightarrow \quad \nabla_{ij}^{(l)} J = \frac{\partial}{\partial w_{ij}^{(l)}} J(\boldsymbol{w})$$

Use the backpropagation algorithm to compute the gradient of a NN.

- can be used with any gradient-based optimizer, including quasi-Newton methods.
- reduces the large amount of computations thanks to chain rule
- requires the derivative of the cost function with respect to the output layer $w_{ij}^{(l)}$ with $l =$ output.

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

2: perform a backward propagation: evaluate for each node a "prediction error":

$$\delta_j^{(l)} = \text{"error" of node } j \text{ in layer } l.$$

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

2: perform a backward propagation: evaluate for each node a “prediction error”:

$$\delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l.$$

3: calculate $\nabla_{ij}^{(l)} J$ using erros $\delta_i^{(l)}$ and $a_i^{(l)}$.

Backpropagation algorithm

The backpropagation steps:

1: perform a forward propagation (calculate $a_i^{(l)}$)

2: perform a backward propagation: evaluate for each node a "prediction error":

$$\delta_j^{(l)} = \text{"error" of node } j \text{ in layer } l.$$

3: calculate $\nabla_{ij}^{(l)} J$ using erros $\delta_i^{(l)}$ and $a_i^{(l)}$.

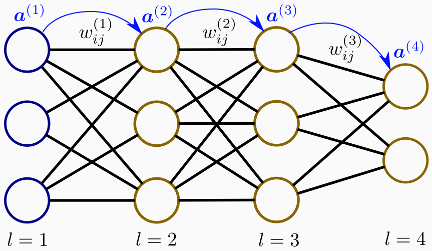
4: perform weight updates, $\Delta w_{ij}^{(l)}$, via gradient descent using $\nabla_{ij}^{(l)} J$.

Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$

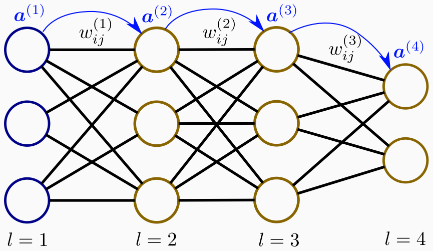


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$

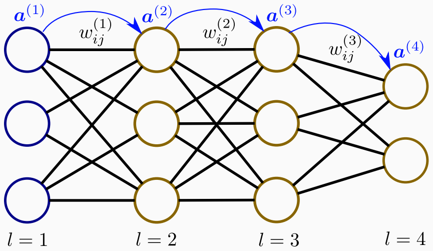


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$

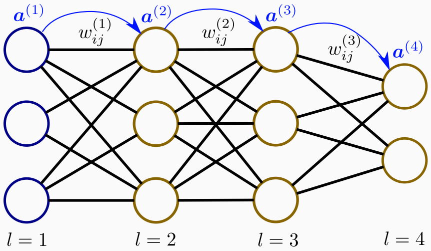


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$

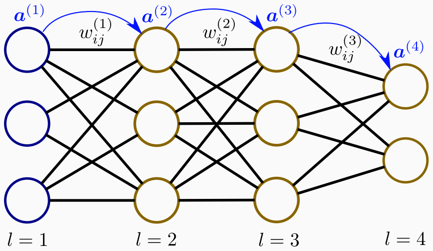


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$
- $\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$

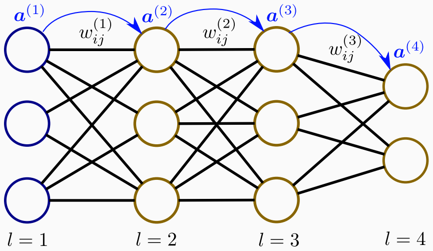


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$
- $\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$
- $\boldsymbol{z}^{(4)} = \boldsymbol{w}^{(3)}\boldsymbol{a}^{(3)}$

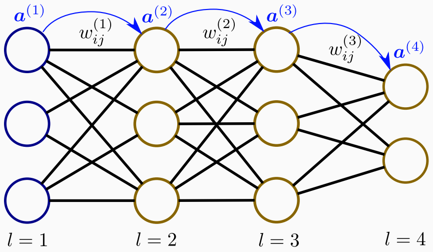


Backpropagation algorithm

Suppose we have a MLP and one training example $(\boldsymbol{x}, \boldsymbol{y})$.

Step 1: We first perform a forward propagation pass:

- $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
- $\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1)}\boldsymbol{a}^{(1)}$
- $\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)})$
- $\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(2)}\boldsymbol{a}^{(2)}$
- $\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$
- $\boldsymbol{z}^{(4)} = \boldsymbol{w}^{(3)}\boldsymbol{a}^{(3)}$
- Output $\boldsymbol{a}^{(4)} = g(\boldsymbol{z}^{(4)})$



At this step we know the output of the current MLP setup.

Backpropagation algorithm

2. evaluate for each node the error $\delta_j^{(k)}$ for $k = 2, 3, \dots, L$.

Some remarks:

It is possible to proof using derivative chain rules that:

$$\nabla_{ij}^{(l)} J = \frac{\partial J}{\partial z_i^{(l+1)}} a_j^{(l)} \equiv \delta_i^{(l+1)} a_j^{(l)},$$

for $l = 1, \dots, L-1$.

Backpropagation algorithm

2. evaluate for each node the error $\delta_j^{(k)}$ for $k = 2, 3, \ldots, L$.

Some remarks:

It is possible to proof using derivative chain rules that:

$$\nabla_{ij}^{(l)} J = \frac{\partial J}{\partial z_i^{(l+1)}} a_j^{(l)} \equiv \delta_i^{(l+1)} a_j^{(l)},$$

for $l = 1, \ldots, L-1$.

The recursive relation for the error is:

$$\delta_i^{(l)} = \sum_k w_{ki}^{(l)} \delta_k^{(l+1)} \cdot g'(z_i^{(l)})$$

and at $l = L$, *i.e.* the highest l index:

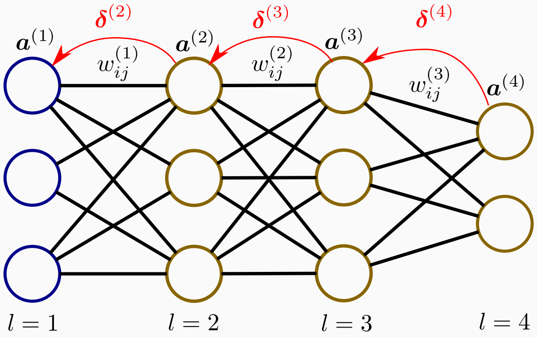
$$\delta_i^{(L)} = \frac{\partial J}{\partial a_i^{(L)}} \cdot g'(z_i^{(L)})$$

where $g'(z_i^{(l)}) = a_i^{(l)}(1 - a_i^{(l)})$ if g is the sigmoid function.

Backpropagation algorithm

Example: evaluating error $\delta_j^{(l)}$ for a MLP with sigmoids in the hidden layers and linear activation function in the output layer:

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \boldsymbol{y}$
- $\boldsymbol{\delta}^{(3)} = (\boldsymbol{w}^{(3)})^T \boldsymbol{\delta}^{(4)} \cdot (\boldsymbol{a}^{(3)}(1 - \boldsymbol{a}^{(3)}))$
- $\boldsymbol{\delta}^{(2)} = (\boldsymbol{w}^{(2)})^T \boldsymbol{\delta}^{(3)} \cdot (\boldsymbol{a}^{(2)}(1 - \boldsymbol{a}^{(2)}))$



Backpropagation algorithm summary

```
Data: training set $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ with $i = 1, \ldots, m$ examples.
Result: the trained neural network
Initialize network weights;
while stopping criterion is not satisfied do
    Set all $\Delta w_{ij}^{(l)} = 0$.
    for $k = 1$ to $m$ do
        Perform forward pass and compute $\boldsymbol{a}^{(l)}$ for $l = 1, 2, 3, \ldots, L$;
        Perform backward pass and compute $\boldsymbol{\delta}^{(l)}$ for $l = 2, \ldots, L$;
        $\Delta w_{ij}^{(l)} := \Delta w_{ij}^{(l)} + a_j^l \delta_i^{(l+1)}$
    end
    Update network weights using gradient descent;
end
```

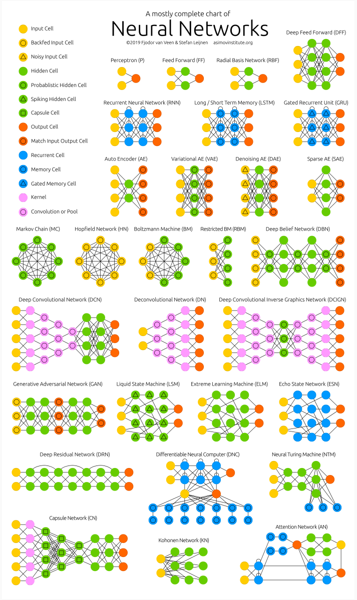
Training neural networks

Some remarks and example of neural network initialization:

- zero: all weights are set to zero so all neurons perform the same calculation. The complexity of the neural network is equivalent to a single neuron.
- random: breaks parameter symmetry.
- glorot/xavier: initialize each weight with a small Gaussian value with mean zero and variance based on the in/out size of the weight.
- he: avoid activation function saturation. Weights are random initialized considering the size of the previous layer.

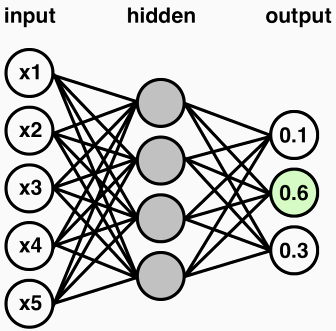
Neural networks zoo

Artificial neural networks architectures


A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org
Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Differentiable Neural Computer (DNC)
Neural Turing Machine (NTM)
Capsule Network (CN)
Kohonen Network (KN)
Attention Network (AN)

MLP

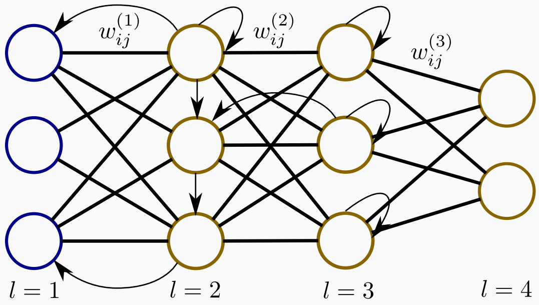
- Sequential model (MLP): regression and classification



Artificial neural networks architectures

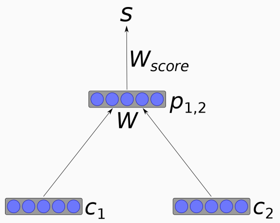
Some examples of neural network popular architectures:

- Recurrent neural networks: neural networks where connections between nodes form a directed cycle.
    - built-in internal state memory
    - built-in notion of time ordering for a time sequence



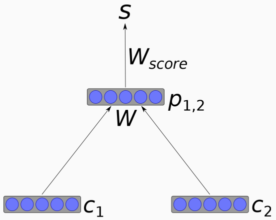
Artificial neural networks architectures

- Recursive neural networks: a variation of recurrent neural network where pairs of layers or nodes are merged recursively.
    - successful applications on natural language processing.
    - some recent applications for model inference.



Artificial neural networks architectures

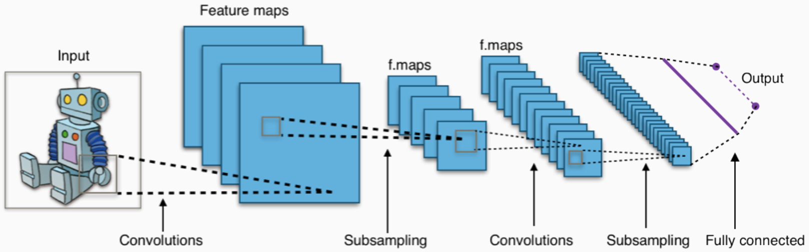
- Recursive neural networks: a variation of recurrent neural network where pairs of layers or nodes are merged recursively.
    - successful applications on natural language processing.
    - some recent applications for model inference.



- Long short-term memory: another variation of recurrent neural networks composed by custom units cells:
    - LSTM cells have an input gate, an output gate and a forget gate.
    - powerful when making predictions based on time series data.

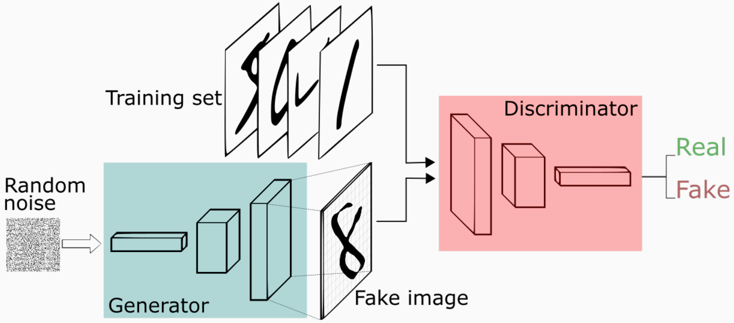
Artificial neural networks architectures

- Convolutional neural networks: multilayer perceptron designed to require minimal preprocessing, *i.e.* space invariant architecture.
    - the hidden layers consist of convolutional layers, pooling layer, fully connected layers and normalization layers
    - great successful applications in image and video recognition.



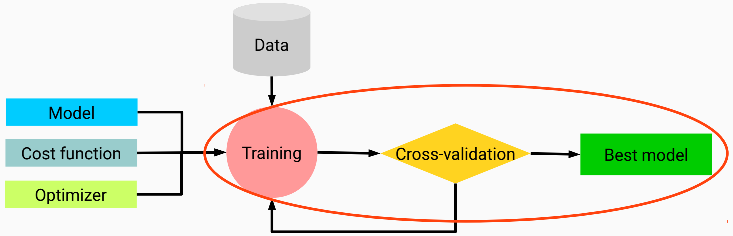
Artificial neural networks architectures

- Generative adversarial network: unsupervised machine learning system of two neural networks contesting with each other.
  - one network generate candidates while the other discriminates.



Hyperparameter tune

Outline


Data
Model
Cost function
Optimizer
Training
Cross-validation
Best model

Hyperparameters summary

So far we have encountered the following problems:

- Model:
  - model architecture / size
  - if NN: layers, nodes, activation functions
  - regularization techniques including early stopping techniques, weight decay, etc.
- Training:
  - performance metrics
  - optimizer configuration, e.g.: η, scheme, etc.
  - cross-validation split fractions
- Dataset:
  - size (gather more data?)
  - unbalanced data

Hyperparameters summary

So far we have encountered the following problems:

- Model:
  - model architecture / size
  - if NN: layers, nodes, activation functions
  - regularization techniques including early stopping techniques, weight decay, etc.
- Training:
  - performance metrics
  - optimizer configuration, e.g.: η, scheme, etc.
  - cross-validation split fractions
- Dataset:
  - size (gather more data?)
  - unbalanced data

Each choice should be tested $\rightarrow$ large space $\rightarrow$ difficult / time consuming.

Practical methodology

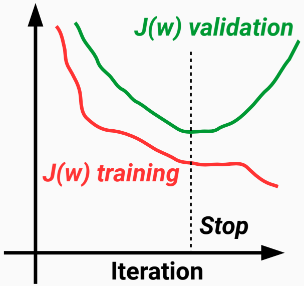
Don't get lost!

Designing a practical pipeline process:

- Estimate current state-of-the-art performance.
- Define realistic project goals, simplify / accelerate algorithms.
- Propose initial performance metrics matching the project goals.
- Perform incremental changes iteratively (data, hyperparameter, algorithms, etc.).

Example 1: auto-tuning model's capacity

How to simplify model capacity selection? Early stopping techniques



Monitor the cost function for the validation set and stop when it stops improving:

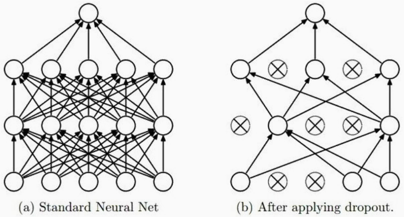
- look at the variation in a moving window
- stop at the minimum of the validation set (lookback method),

Example 2: auto-tuning model's capacity

How to simplify model capacity selection? Neural Network Dropout

At each training stage:

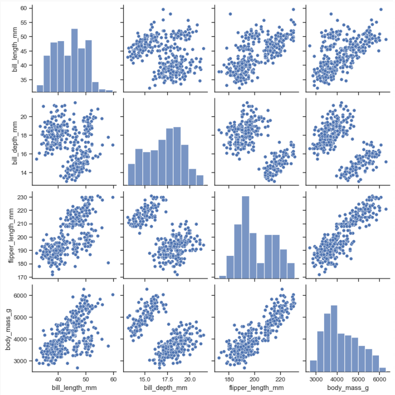
- individual nodes and related incoming and outgoing edges are dropped-out of the neural network with a fixed probability.
- the reduced NN is trained on the data.
- the removed nodes are reinserted in the NN with their original weights.

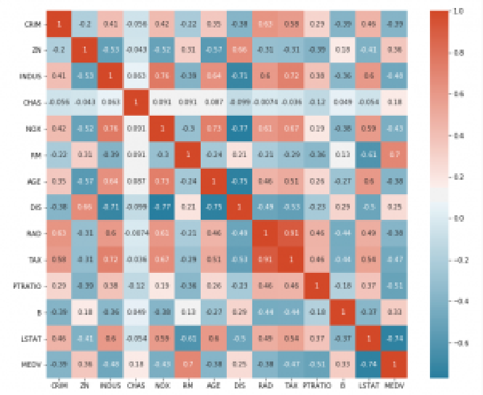


Data considerations

Data considerations

Step 1: Understand your data, extract correlations, perform minimal feature extraction.





Data considerations

Gathering more data is usually crucial but before:

Step 2:

- check the performance with the current training set, if its performance is:
    - **poor** → increase model size and tune the optimizer.
    - **acceptable** → check test set performance.

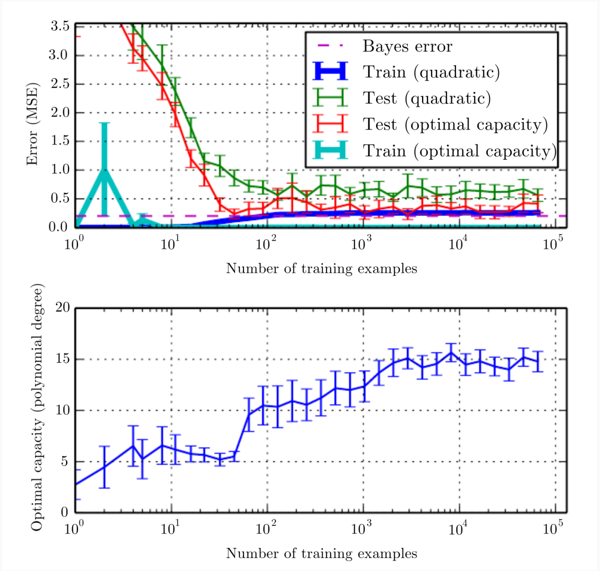
Step 3:

- if tuned models **fail** → check data for **noise or inconsistencies, collect new data**
- if test set performance is **poor** → **gather more data** if possible
    - if not possible **reduce the size of the model** or **improve hyperparameter tuning**.

Step 4:

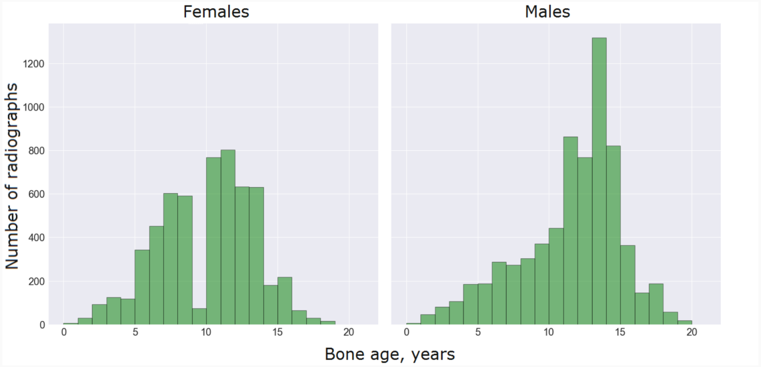
- estimate how much **additional training data** is needed.
- if gathering much more data is **not feasible** → improve the learning algorithm itself.

Data considerations



Unbalanced data

Unbalanced datasets are common issues:



Solution → perform **class weighting, oversampling, data augmentation**.

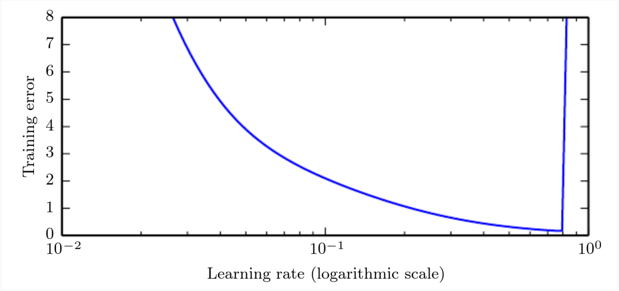
Hyperparameter tune

Manual hyperparameter tuning

Manual approach goal → achieve **good performance on the test set**.

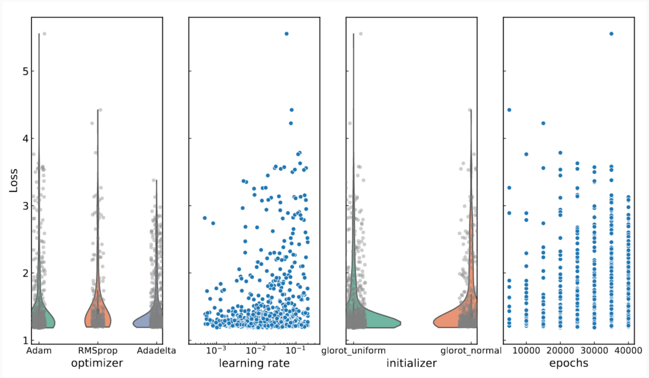
Some examples of effect of hyperparamaters on model capacity:

- Number of layers/nodes → increases capacity when increased.
- Learning rate → increases capacity when tuned optimally.
- Weight decay → increases capacity when decreased.
- Dropout rate → increases capacity when decreased.



Automatic hyperparameter optimization algorithms

Hyperparameter tuning is an **optimization problem** thus we can **automate the process**.



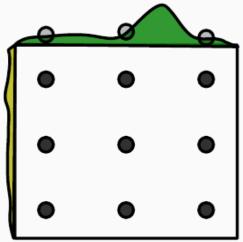
Common approaches:

- grid search
- random search
- bayesian optimization
- gradient-based optimization
- evolutionary optimization

Grid search

Grid search: searching through a manually subset range of the hyperparameter space.

- Train model for every grid point of the hyperparameter space.
- Allocate initial grids following a logarithmic scale, perform zoom in another round.
- Monitor the best validation set error $\rightarrow$ best hyperparameter values.



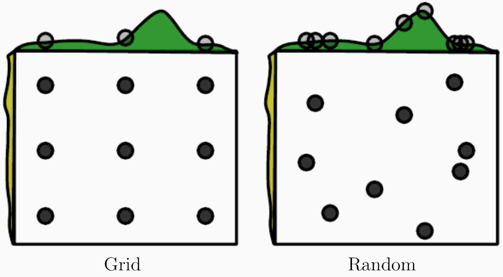
Grid

The disadvantage: with n values and m parameters the number of trials is $\mathcal{O}(n^m)$

Random search

Random search: sample trial points from a marginal distribution for each hyperparameter.

- Do not discretize or bin the values of the hyperparameters.
- The marginal distribution will perform independent explorations of hyperparameters.

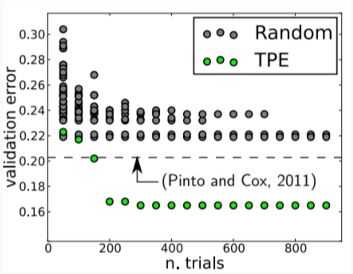


Model-based hyperparameter optimization

Idea:

- Perform a training using a set of hyperparameters.
- Define the cost function to be optimize as the validation set error.
- Use sequential model-based optimization (SMBO) approach, or algorithms which monitors the numerical gradient from the loss function.

Example: Tree-structured Parzen Estimator (TPE)

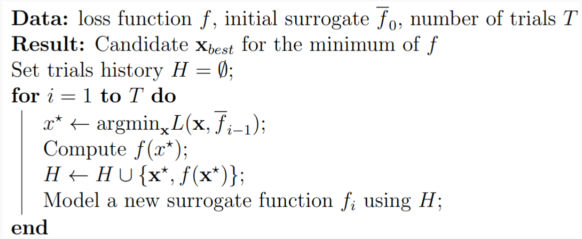


SMBO

SMBO minimizes functions $f : X \to \mathcal{R}$ where each evaluation is very expensive.

The f function is replaced by a **surrogate** function, $\bar{f}$, easier to manage.

The surrogate function proposes a new search point $\mathbf{x}_{i+1}$, $f(\mathbf{x}_{i+1})$ is computed and $\bar{f}$ updated or recomputed to approximate better the true loss function.



```
Data: loss function $f$, initial surrogate $\overline{f}_0$, number of trials $T$
Result: Candidate $\mathbf{x}_{best}$ for the minimum of $f$
Set trials history $H = \emptyset$;
for $i = 1$ to $T$ do
    $x^\star \leftarrow \text{argmin}_{\mathbf{x}} L(\mathbf{x}, \overline{f}_{i-1})$;
    Compute $f(x^\star)$;
    $H \leftarrow H \cup \{\mathbf{x}^\star, f(\mathbf{x}^\star)\}$;
    Model a new surrogate function $f_i$ using $H$;
end
```

Where $L(\mathbf{x}, \bar{f})$, the criterion, and $\bar{f}$ depend on the specific algorithm.

TPE

The Tree-structured Parzen Estimator (TPE) algorithm is a SMBO where the surrogate model is a probabilistic model $p(y|\mathbf{x})$, which chooses the next trial point by optimizing the **Expected Improvement** criterion:

$$\mathrm{EI}_{y^\star}(\mathbf{x}) = \int_{-\infty}^{\infty} \max(y^\star - y, 0) p(y|\mathbf{x}) dy$$

which measures how much the loss function is expected to be lower than a threshold value $y^\star$, chosen so that $p(y < y\star) = \gamma$ where γ is a parameter of the algorithm.

TPE

The $p(y|\mathbf{x})$ is computed via Bayes' theorem through $p(\mathbf{x}|y)$

$$p(\mathbf{x}|y) = l(\mathbf{x}) \quad \text{if } y < y^\star; \qquad g(\mathbf{x}) \quad \text{if } y \geq y^\star$$

where $l(\mathbf{x})$ and $g(\mathbf{x})$ are probability distributions estimated by using the trials $\mathbf{x}_i$ such that $f(\mathbf{x}_i)$ is respectively lower and higher or equal $y^\star$.

The Expected Improvement for the TPE admits a closed form solution:

$$\mathrm{EI}_{y^\star}(\mathbf{x}) = \int_{-\infty}^{\infty} \max(y^\star - y) \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} dy \propto \left(\gamma + \frac{g(\mathbf{x})}{l(\mathbf{x})}(1-\gamma)\right)^{-1}$$

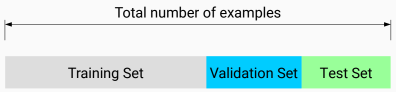
Code libraries and algorithms

- scikit-learn: grid and random search.
- Hyperopt: grid, random and TPE.
- Optuna: grid, random, TPE, CMAES.
- Ray Tune: grid, random, bandit, blended, cost-frugal, TPE, gradient-free, etc.

Cross-validation

Cross-validation

The hyperparameter tune procedure still requires the training/validation/test split to choose for the best model.

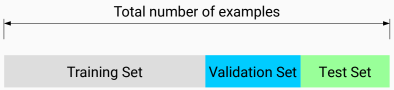


Problems:

- how to perform the data split when the available data set is small?
- how to define a suitable split?

Cross-validation

The hyperparameter tune procedure still requires the training/validation/test split to choose for the best model.



Problems:

- how to perform the data split when the available data set is small?
- how to define a suitable split?

Solution:

Use cross-validation algorithms to access the quality of your model + hyperparameter choice.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
    - Leave-p-out: uses p observations as validation set.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
  - Leave-p-out: uses p observations as validation set.
  - Leave-one-out: set $p = 1$.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
    - Leave-p-out: uses p observations as validation set.
    - Leave-one-out: set $p = 1$.
- Non-exhaustive cross-validation: do not test all possible ways to divide the original sample but use discrete subsamples.

Cross-validation

Cross-validation performs a rotation estimation by:

1. **partitioning data** into **training/validation** subsets
2. multiple rounds of cross-validation using different partitions
3. results are averaged over the rounds to give an estimate of the model performance

Common approaches to cross-validation:

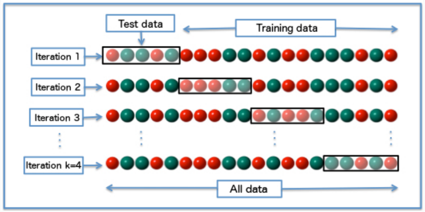
- Exhaustive cross-validation: test all possible ways to divide the original sample into a training and a validation set.
    - Leave-p-out: uses p observations as validation set.
    - Leave-one-out: set $p = 1$.
- Non-exhaustive cross-validation: do not test all possible ways to divide the original sample but use discrete subsamples.
    - k-fold cross-validation.

Example k-fold cross-validation

k-fold cross-validation:

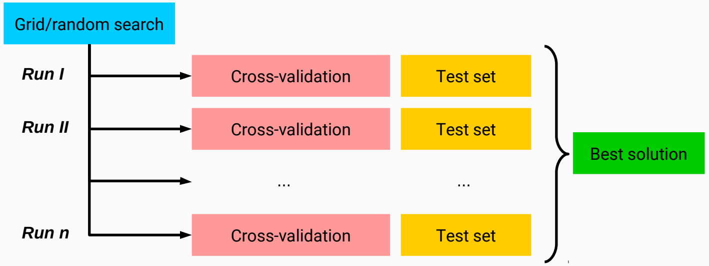
1. the original data is randomly partitioned into k equal sized subsamples.
2. from the k subsamples, a single subsample is used as validation data and the remaining $k-1$ subsamples are used as training data.
3. repeat the process k times by changing the validation and training partitions.
4. compute the average over the k results.

Example of k-fold with $k = 4$:



Complete recipe

Perform hyperparameter tune coupled to cross-validation:

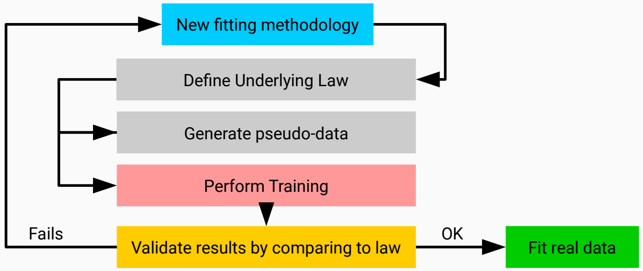


Easy parallelization at search and cross-validation stages.

Closure testing

Closure tests

Validation and optimization of fitting strategy performed on **closure test** with known underlying law.



ML in practice

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

For ML applications:

- Keras: a Python deep learning library.
- Theano: a Python library for optimization.
- PyTorch: a DL framework for fast, flexible experimentation.
- Caffe: speed oriented deep learning framework.
- MXNet: deep learning frameowrk for neural networks.
- CNTK: Microsoft Cognitive Toolkit.

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

For ML applications:

- Keras: a Python deep learning library.
- Theano: a Python library for optimization.
- PyTorch: a DL framework for fast, flexible experimentation.
- Caffe: speed oriented deep learning framework.
- MXNet: deep learning frameowrk for neural networks.
- CNTK: Microsoft Cognitive Toolkit.

For ML and beyond:

- TensorFlow: libray for numerical computation with data flow graphs.
- scikit-learn: general machine learning package.

Most popular public ML frameworks

For experimental HEP:

- TMVA: ROOT's builtin machine learning package.

For ML applications:

- Keras: a Python deep learning library.
- Theano: a Python library for optimization.
- PyTorch: a DL framework for fast, flexible experimentation.
- Caffe: speed oriented deep learning framework.
- MXNet: deep learning frameowrk for neural networks.
- CNTK: Microsoft Cognitive Toolkit.

For ML and beyond:

- TensorFlow: libray for numerical computation with data flow graphs.
- scikit-learn: general machine learning package.

Why use public codes? $\rightarrow$ builtin models and automatic differentiation

Keras

Keras is a high-level deep learning framework in Python which runs on top of TensorFlow, CNTK or Theano.

Keras

Keras is a high-level deep learning framework in Python which runs on top of TensorFlow, CNTK or Theano.

Pros:

- fast prototyping, user friendly, common code for multiple backends.
- support several NN architectures out-of-the-box.
- runs seamlessly on CPU and GPU.

Keras

Keras is a high-level deep learning framework in Python which runs on top of TensorFlow, CNTK or Theano.

Pros:

- fast prototyping, user friendly, common code for multiple backends.
- support several NN architectures out-of-the-box.
- runs seamlessly on CPU and GPU.

Cons:

- more tricky to extend when custom ML setups are required
- runs only in Python

TensorFlow

TensorFlow is a library for high performance numerical computation.

TensorFlow

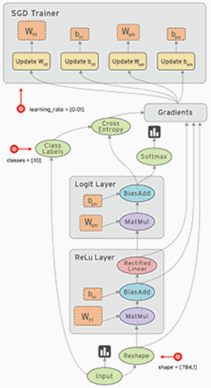
TensorFlow is a library for high performance numerical computation.

Pros:

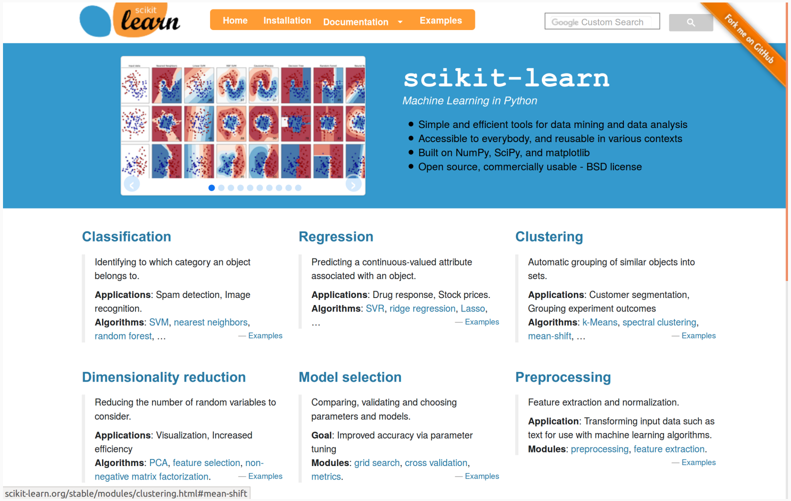
- solves optimization problems with automatic differentiation.
- can be extended in python and c/c++.
- runs seamlessly on CPU and GPU, and can uses JIT technology.

Cons:

- do not provides builtin models from the core framework
- less automation for cross-validation and hyperparameter tune



Scikit-learn



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

scikit-learn.org/stable/modules/clustering.html#mean-shift

Scikit-learn

Scikit-learn contains the most popular algorithms for:

- Supervised learning: neural networks, decision trees, etc.
- Unsupervised learning: density estimate, clustering, etc.
- Model selection: cross-validation, hyperparameter tune, etc.
- Dataset transformations: feature extractions, dim. reduction, etc.
- Dataset loading
- Strategies to scale computationally
- Computational performance

Questions?