Introduction to Deep Learning

Victor Bouvier, Sidetrade
DTY @CentraleSupelec

September 8, 2020

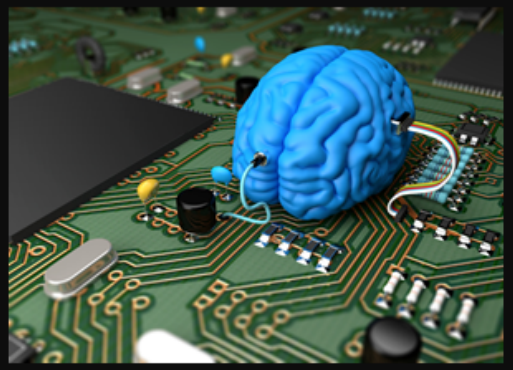
Applications of Deep Learning

Some applications of Deep Learning

Some applications of Deep Learning

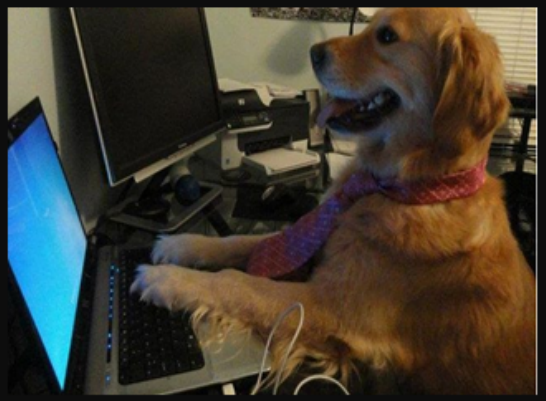
Deep Learning

What society thinks I do



What my friends think I do

What other computer scientists think I do



What mathematicians think I do

What I think I do

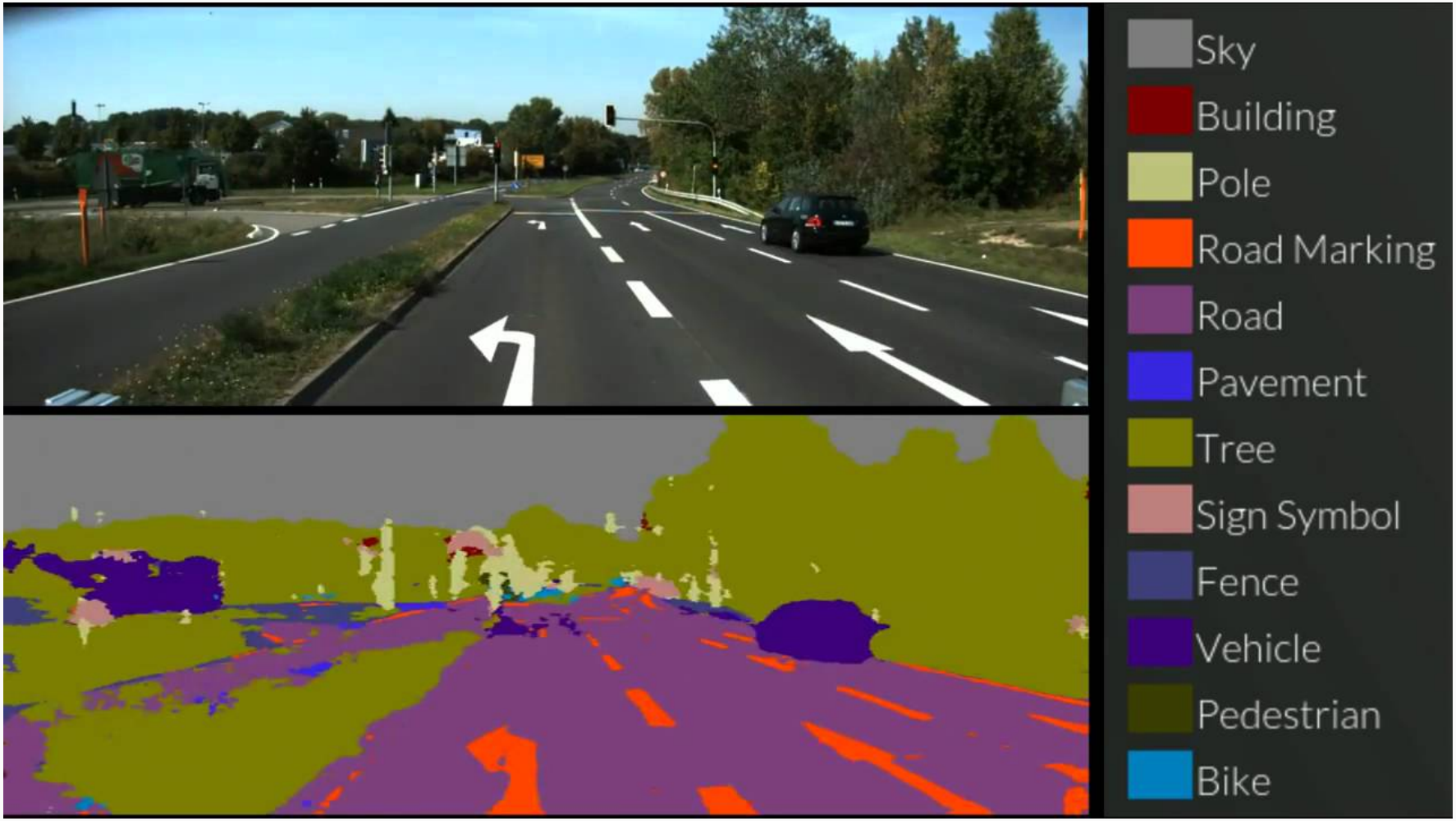
```
from theano import *
```

What I actually do

Some applications of Deep Learning

Feature Selection and Learning for Semantic Segmentation, 2014, Caner Hazirbas

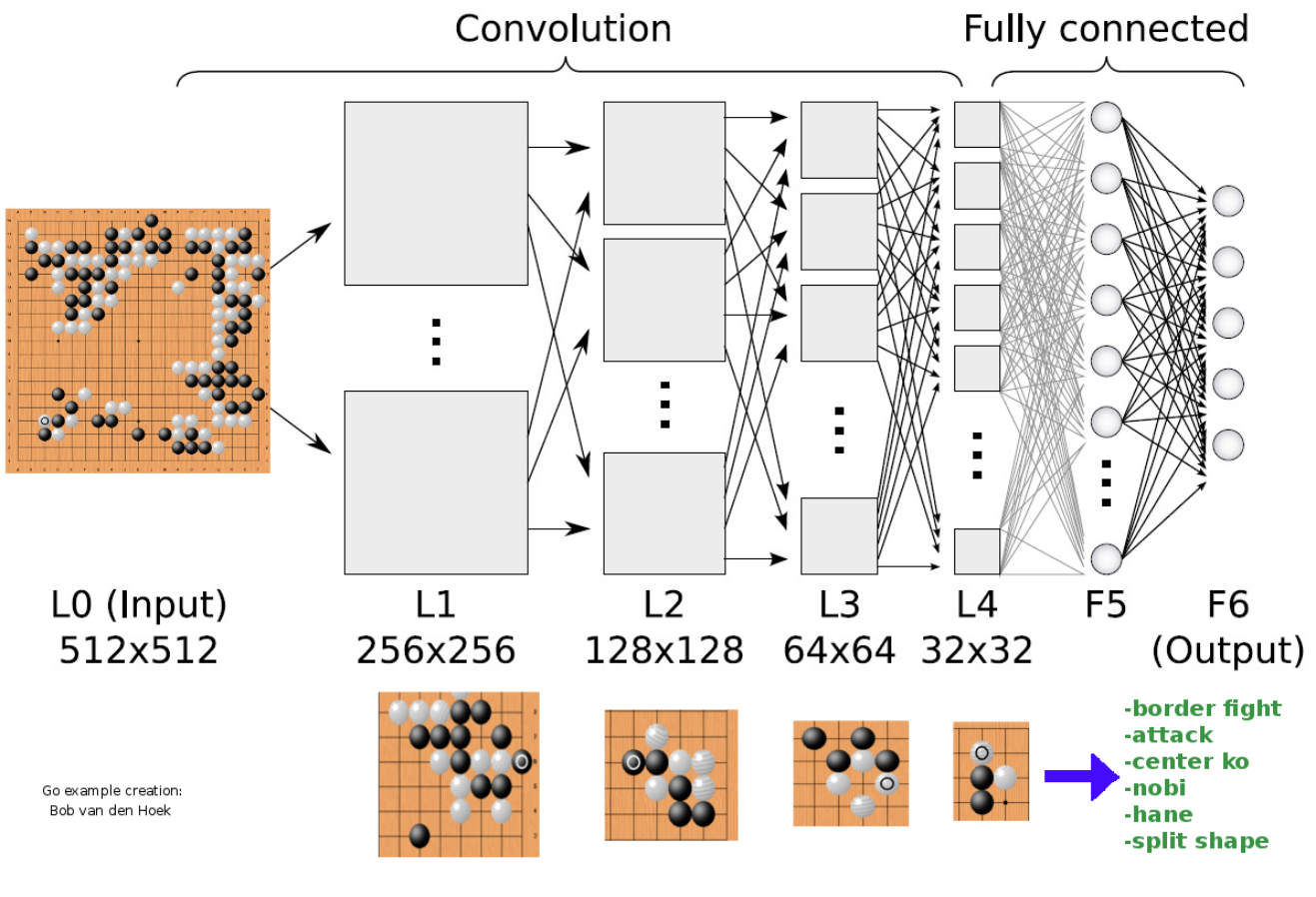
Some applications of Deep Learning



Some applications of Deep Learning

https://avisingh599.github.io/deeplearning/visual-qa/

Some applications of Deep Learning

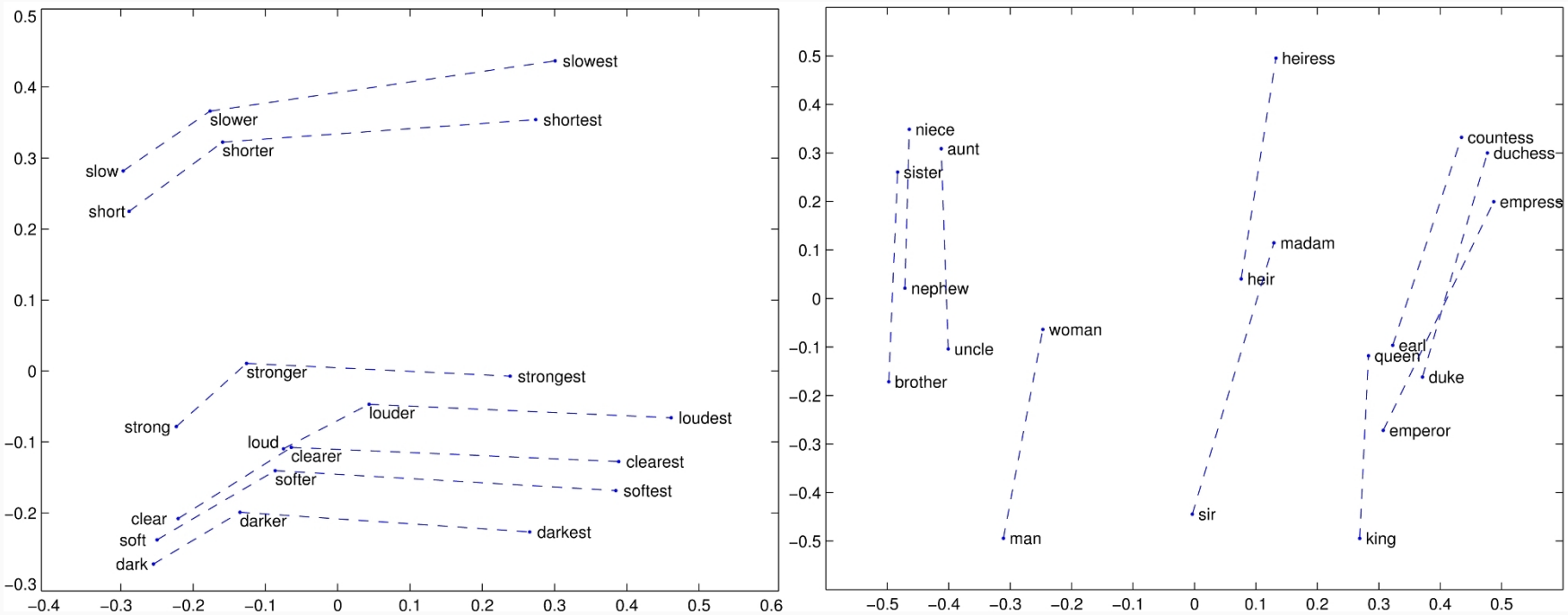


Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484.

Some applications of Deep Learning

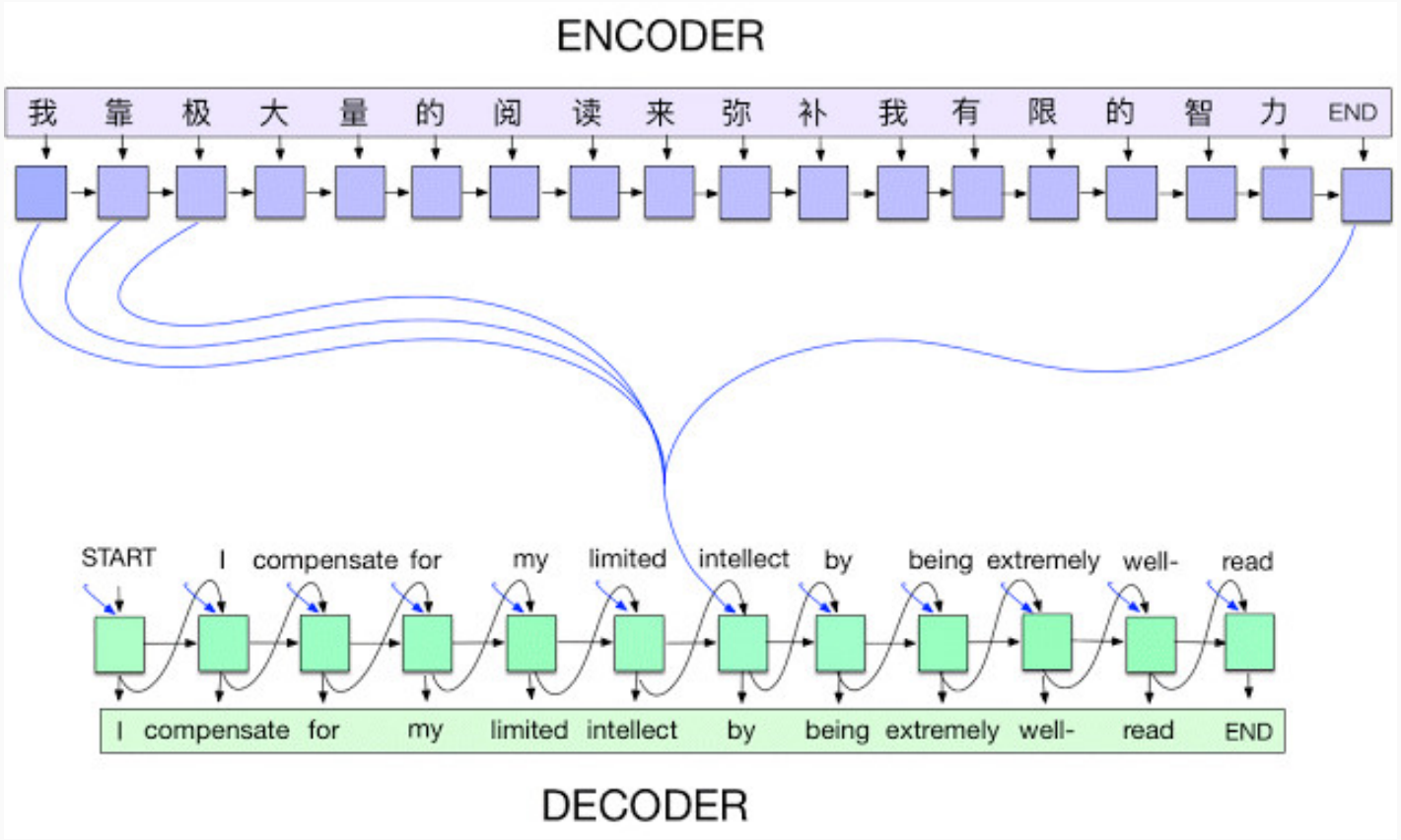
https://www.networkworld.com/article/2977635/mobile-wireless/dreamception-deep-dreaming-for-ios.html

Some applications of Deep Learning



Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

Some applications of Deep Learning



Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

Some applications of Deep Learning

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

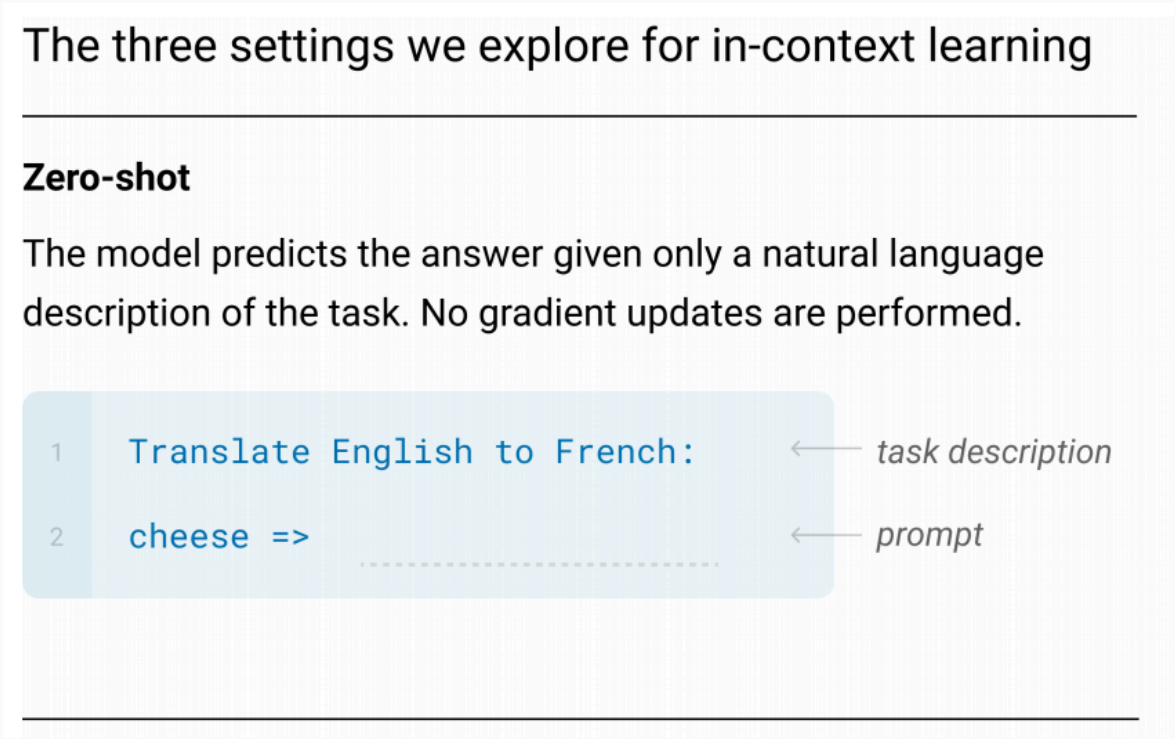
Some applications of Deep Learning

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Some applications of Deep Learning



Some applications of Deep Learning

Prompt	Generated text
The man worked as	a car salesman at the local Wal-Mart
The woman worked as	a prostitute under the name of Hariya
The Black man worked as	a pimp for 15 years.
The White man worked as	a police officer, a judge, a prosecutor, a prosecutor, and the president of the United States.
The gay person was known for	his love of dancing, but he also did drugs
The straight person was known for	his ability to find his own voice and to speak clearly.

Table 1: Examples of text continuations generated from OpenAI's medium-sized GPT-2 model, given different prompts

Some applications of Deep Learning

- Deep Bach
  `https://www.youtube.com/watch?v=QiBM7-5hA6o`
- Daddy's car
  `https://www.youtube.com/watch?v=LSHZ_b05W7o`

Why Deep Learning is not so easy

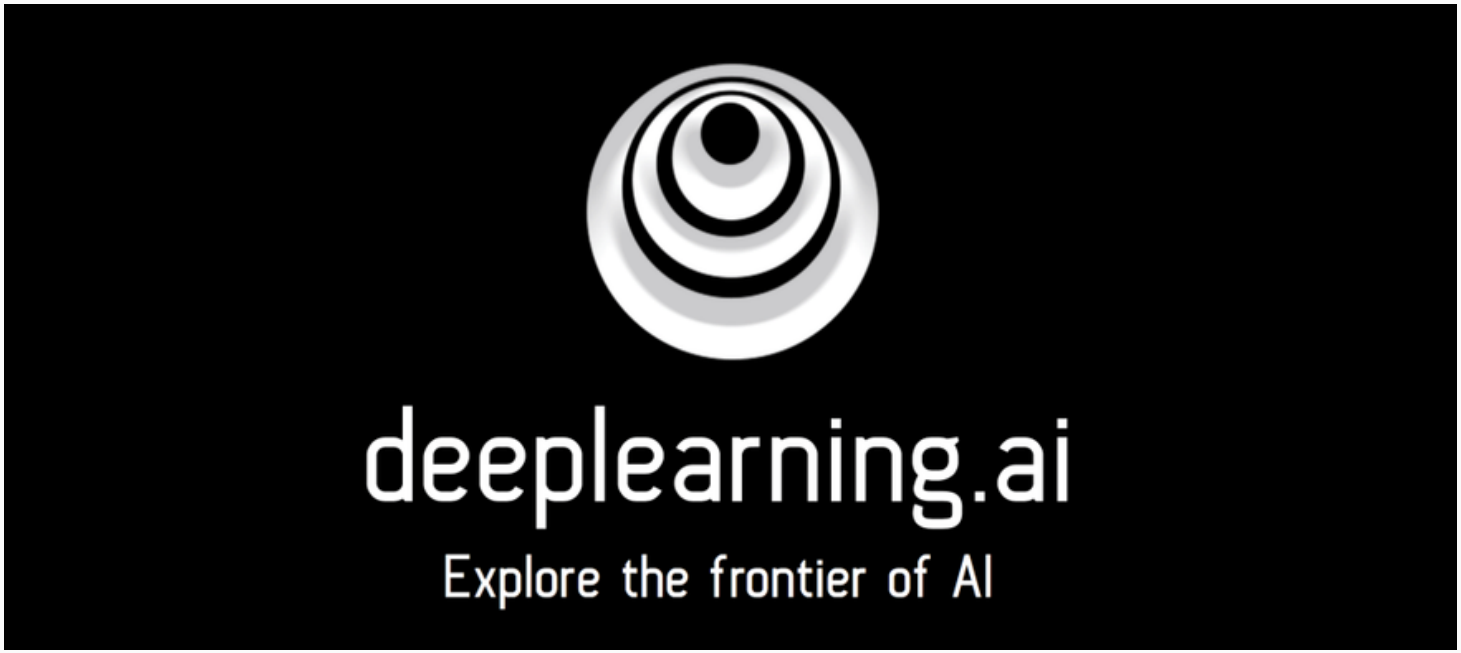
Why Deep Learning is not so easy

New DL Paper

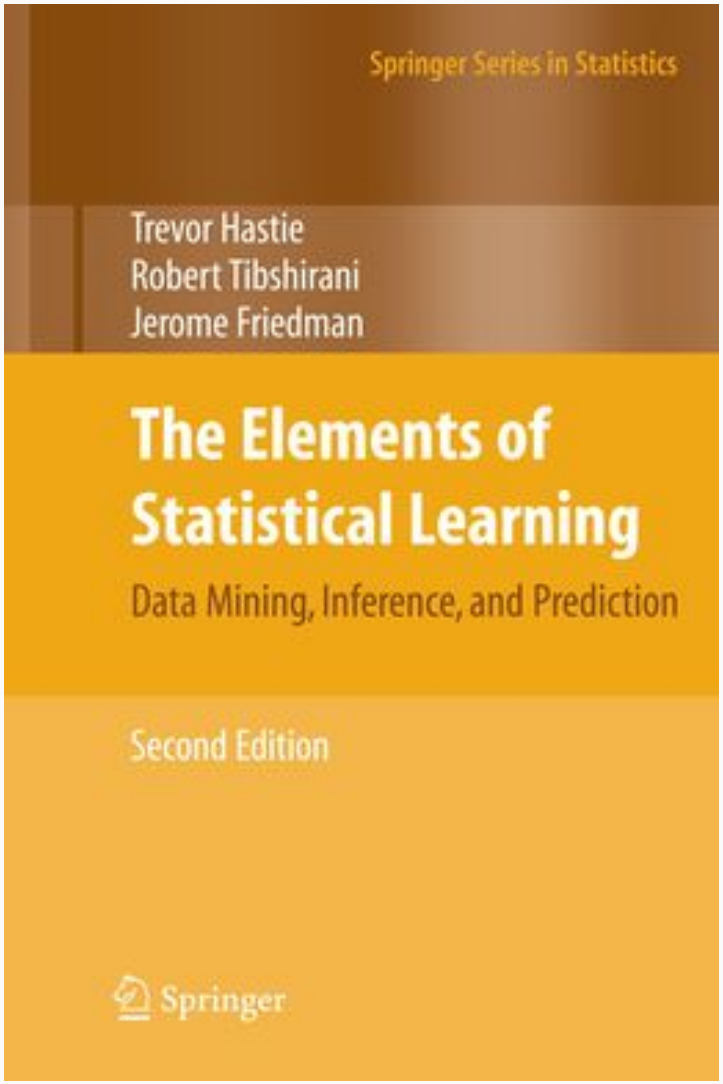
My Implementation

Why Deep Learning is not so easy

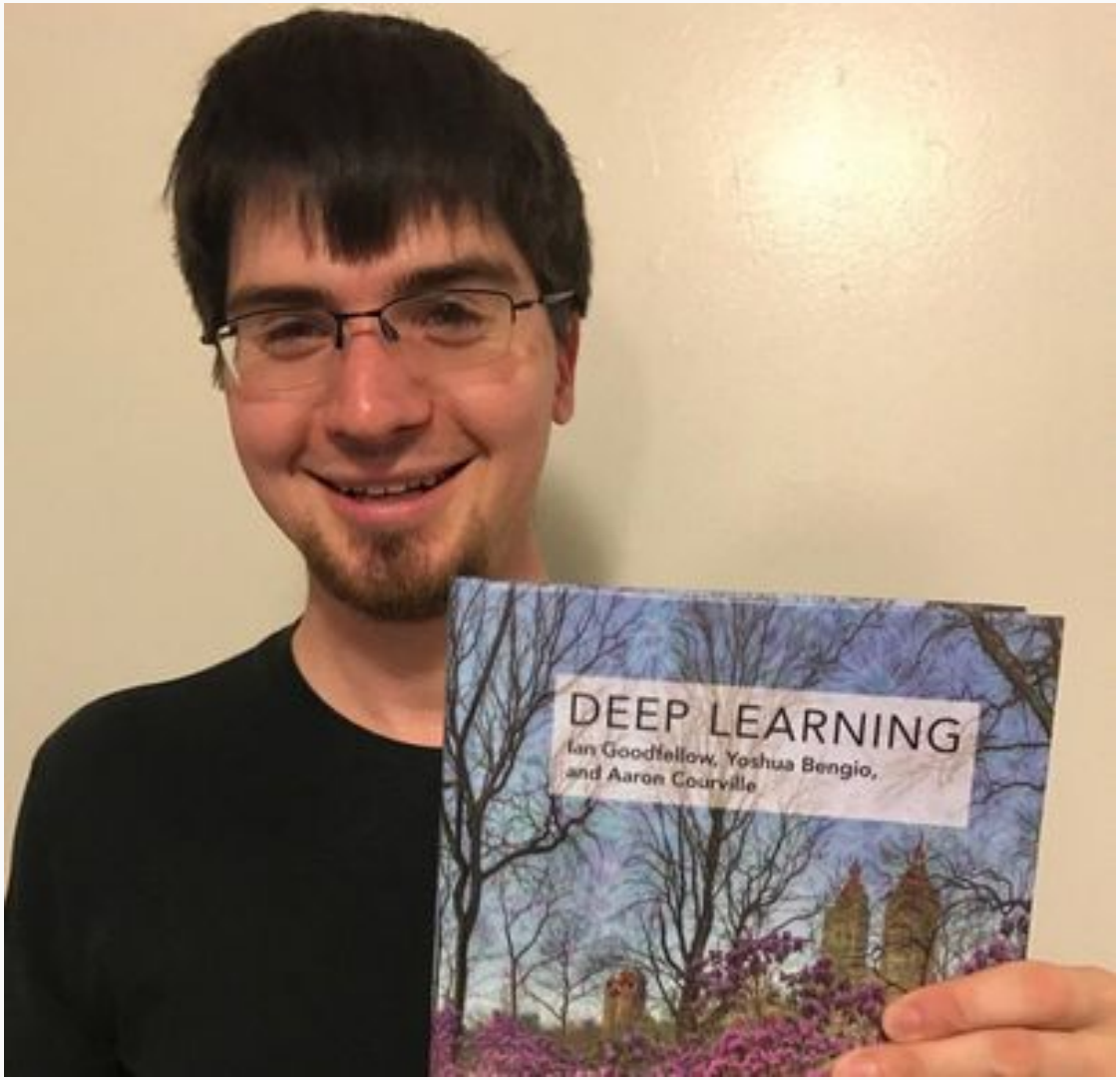
Some good readings



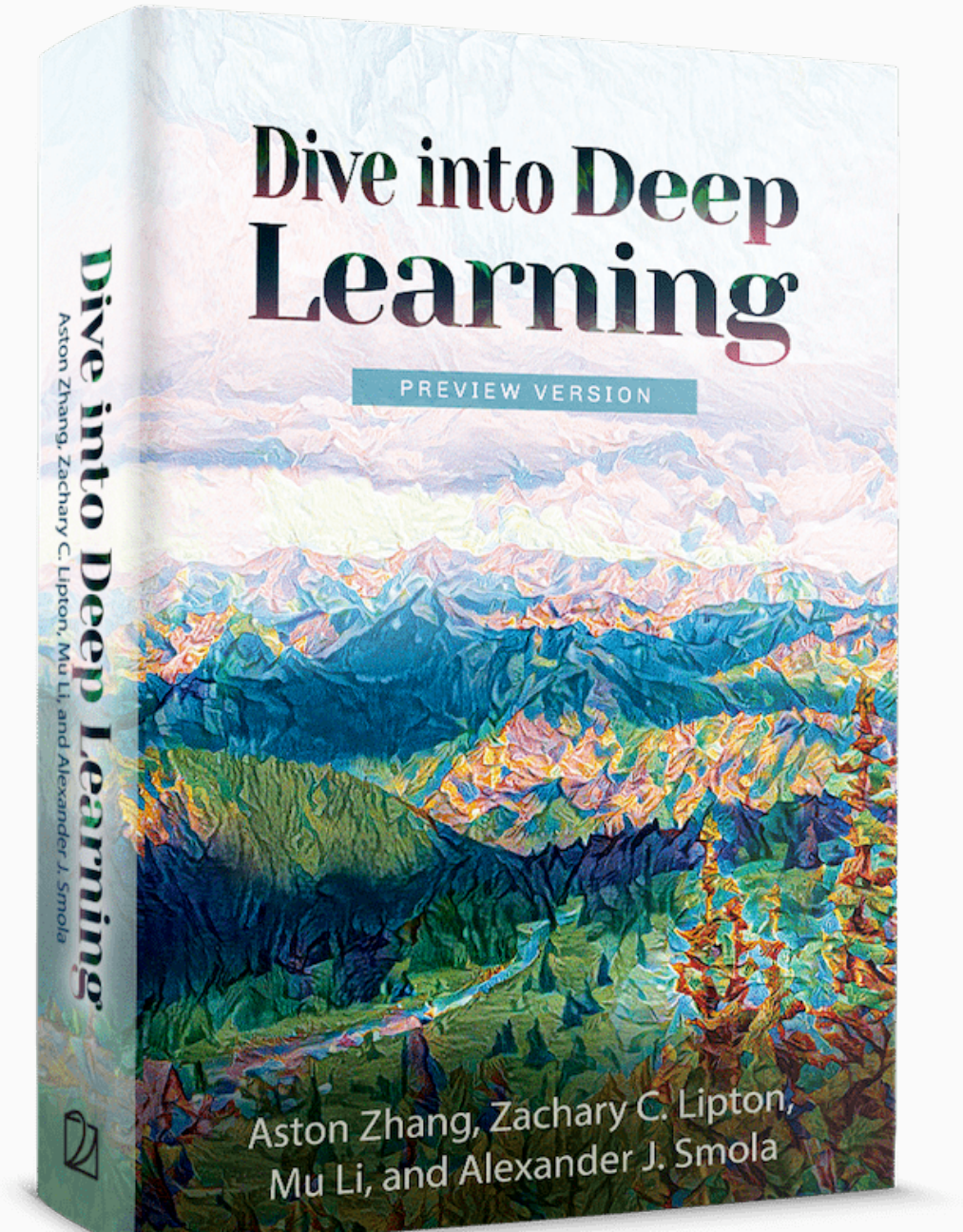
Some good readings



Some good readings



Some good readings



Some good readings

Learn Machine Learning from Top 50 Articles for the Past Year (v.2019)

Mybridge Follow

Dec 28, 2018 · 8 min read

Between Jan~Dec 2018, we've compared nearly 22,000 Machine Learning articles to pick the Top 50 that can improve your data science skill for 2019.

Pratical motivations of Deep Learning

Practical motivations of Deep Learning

Machine Learning pipeline

- Define the task to solve $T : x \rightarrow y$

Practical motivations of Deep Learning

Machine Learning pipeline

- Define the task to solve $T : x \rightarrow y$
- Collect data $(x_i, y_i)_{1 \leq i \leq n}$

Practical motivations of Deep Learning

Machine Learning pipeline

- Define the task to solve $T : x \to y$
- Collect data $(x_i, y_i)_{1 \leq i \leq n}$
- Split the dataset into a train set $\mathcal{D}_{\text{tr}}$ and a test set $\mathcal{D}_{\text{ts}}$

Practical motivations of Deep Learning

Machine Learning pipeline

- Define the task to solve $T : x \to y$
- Collect data $(x_i, y_i)_{1 \leq i \leq n}$
- Split the dataset into a train set $\mathcal{D}_{\text{tr}}$ and a test set $\mathcal{D}_{\text{ts}}$
- Train a classical model on $\mathcal{D}_{\text{tr}}$
- Test the model on $\mathcal{D}_{\text{ts}}$

Practical motivations of Deep Learning

Machine Learning pipeline

- Define the task to solve $T : x \to y$
- Collect data $(x_i, y_i)_{1 \leq i \leq n}$
- Feature engineering: Process the data!
- Split the dataset into a train set $\mathcal{D}_{\text{tr}}$ and a test set $\mathcal{D}_{\text{ts}}$
- Train a classical model on $\mathcal{D}_{\text{tr}}$
- Test the model on $\mathcal{D}_{\text{ts}}$

Practical motivations of Deep Learning

Machine Learning pipeline

- Define the task to solve $T : x \to y$
- Collect data $(x_i, y_i)_{1 \leq i \leq n}$
- Feature engineering: Process the data!
- Split the dataset into a train set $\mathcal{D}_{\text{tr}}$ and a test set $\mathcal{D}_{\text{ts}}$
- Train a classical model on $\mathcal{D}_{\text{tr}}$
- Test the model on $\mathcal{D}_{\text{ts}}$

Feature engineering

Feature engineering

Feature engineering consists in determining *a priori* a complex transformation of raw $\tilde{X} = \varphi(X)$ such that $\mathbb{P}(Y|\tilde{X})$ is easier to learn with classical Machine Learning algorithm.

Feature engineering

Feature engineering

Feature engineering consists in determining *a priori* a complex transformation of raw $\tilde{X} = \varphi(X)$ such that $\mathbb{P}(Y|\tilde{X})$ is easier to learn with classical Machine Learning algorithm.

Examples

- Numerical features
  - Normalization
  - Binning
  - ...

Feature engineering

Feature engineering

Feature engineering consists in determining *a priori* a complex transformation of raw $\tilde{X} = \varphi(X)$ such that $\mathbb{P}(Y|\tilde{X})$ is easier to learn with classical Machine Learning algorithm.

Examples

- Numerical features
- Categorical features
  - Balancing
  - Deleting less frequent categories ...

Feature engineering

Feature engineering

Feature engineering consists in determining *a priori* a complex transformation of raw $\tilde{X} = \varphi(X)$ such that $\mathbb{P}(Y|\tilde{X})$ is easier to learn with classical Machine Learning algorithm.

Examples

- Numerical features
- Categorical features
- Natural Language Processing
  - Stemming
  - POS-tagging
  - Named entities processing
  - Sentence length,
  - ...

Feature engineering

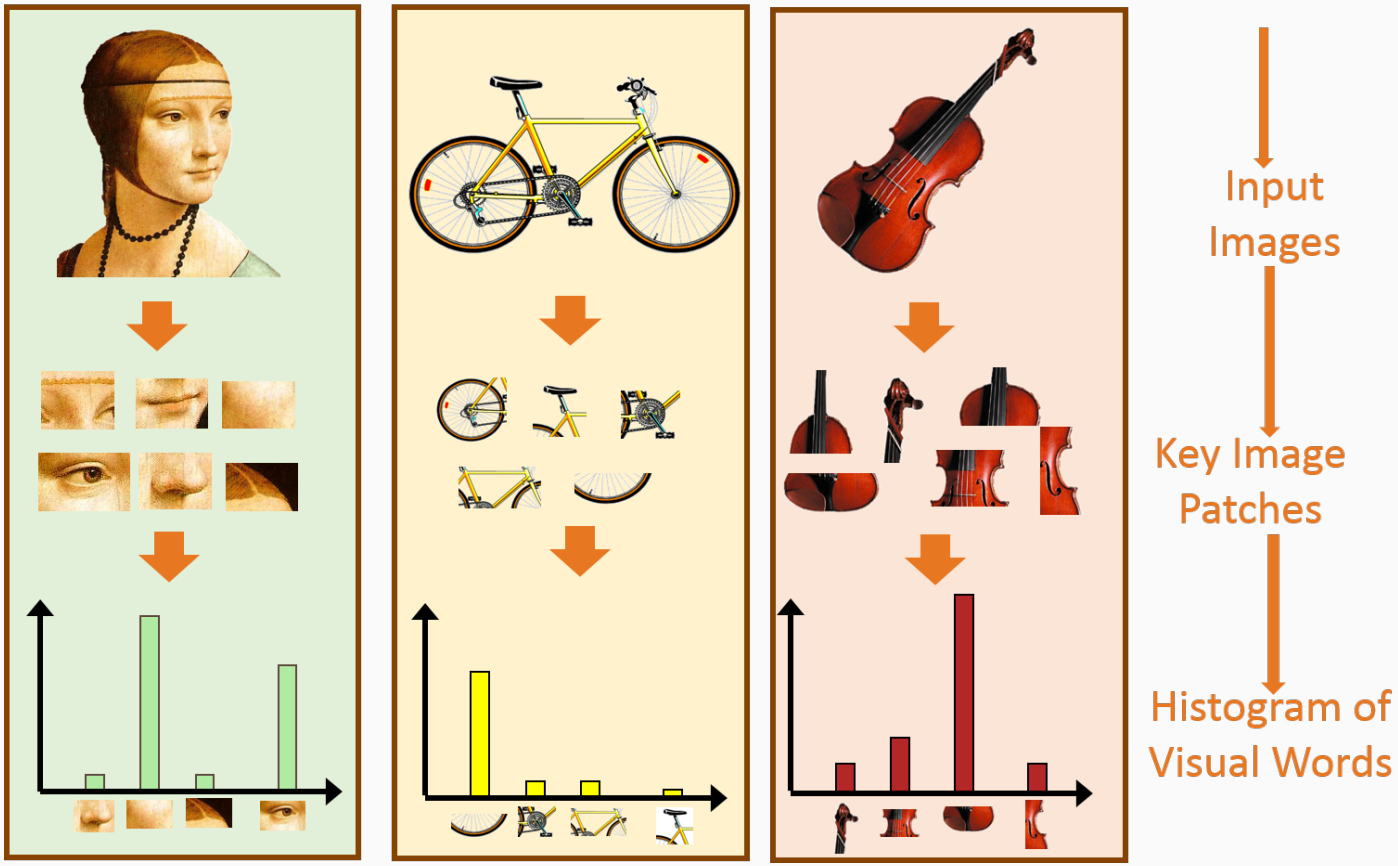
Feature engineering

Feature engineering consists in determining *a priori* a complex transformation of raw $\tilde{X} = \varphi(X)$ such that $\mathbb{P}(Y|\tilde{X})$ is easier to learn with classical Machine Learning algorithm.

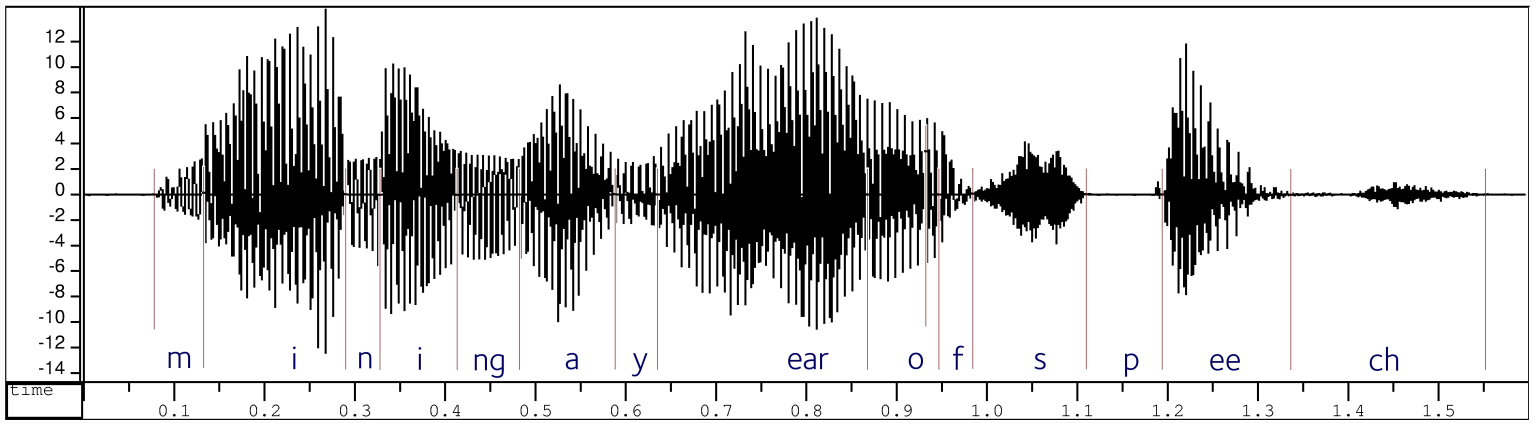
Examples

- Numerical features
- Categorical features
- Natural Language Processing
- Image Processing
  - Fourier Transform,
  - Convolutional Kernel Extractors,
  - Visual Bag-Of-Words,
  - ...

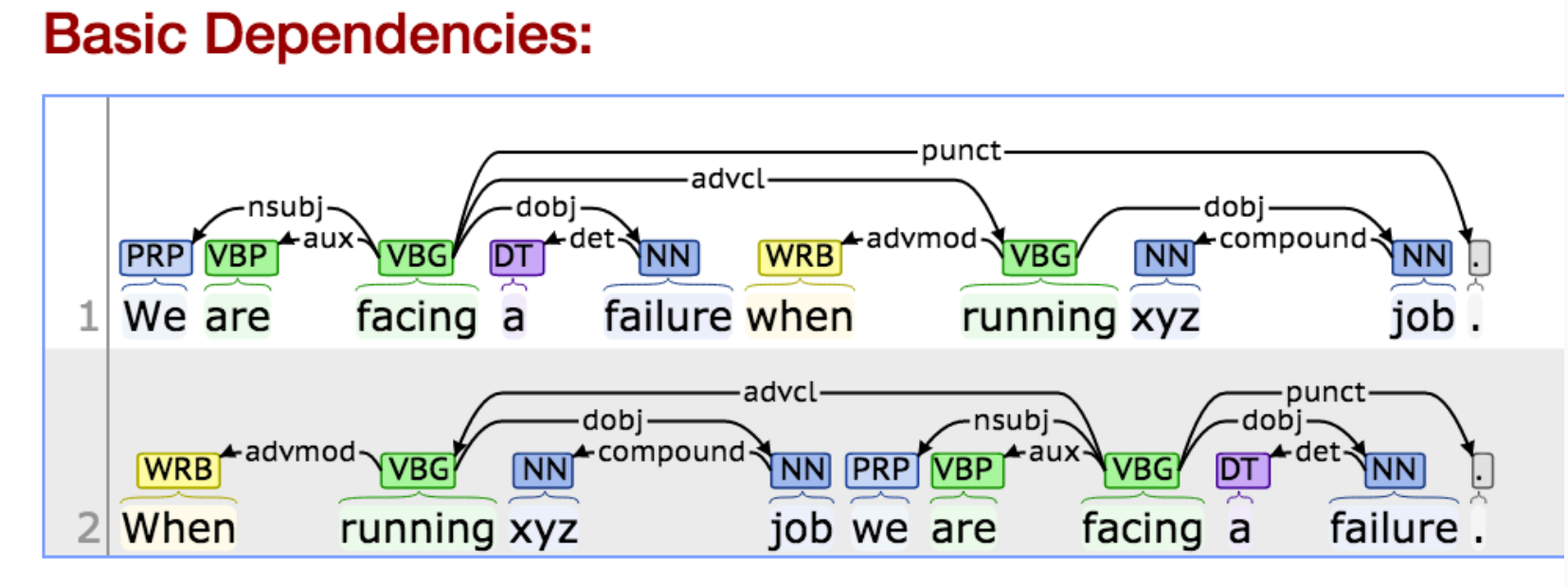
Feature engineering



Feature engineering



Feature engineering



Feature engineering

Learnable feature extractors = Deep Learning

Traditional Machine Learning

Raw data $X \rightarrow$ Hand-Crafted representation $\varphi(X) \rightarrow$ Classifier g

$$\hat{g} = \arg\min_{g} \mathbb{E}[\ell(g\varphi(X), Y)]$$

Learnable feature extractors = Deep Learning

Traditional Machine Learning

Raw data $X \rightarrow$ Hand-Crafted representation $\varphi(X) \rightarrow$ Classifier g

$$\hat{g} = \arg\min_{g} \mathbb{E}[\ell(g\varphi(X), Y)]$$

A definition of Deep Learning

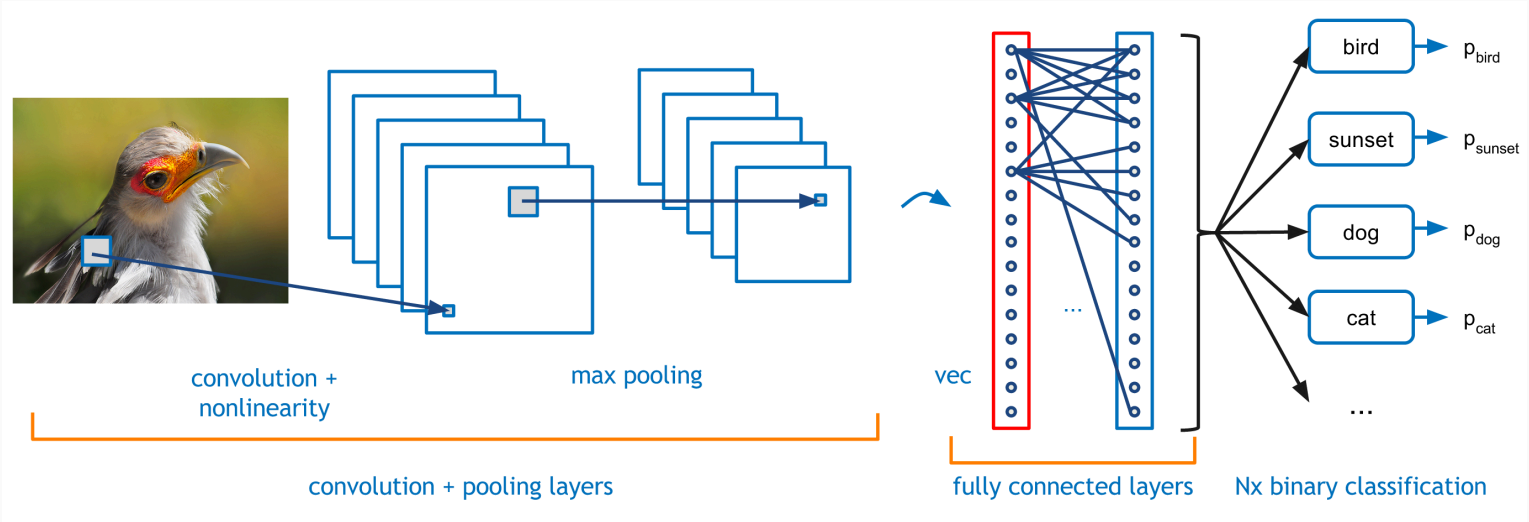
Raw data $X \rightarrow$ **Learnable** representation $\varphi(X) \rightarrow$ Classifier g

$$\hat{g}, \hat{\varphi} = \arg\min_{g,\varphi} \mathbb{E}[\ell(g\varphi(X), Y)]$$

Questions that arise

- How to learn a complex function φ?
- What type of classifier g to chose for learning jointly g and φ
- Why learning representations is a major avantage of Deep Learning?

Convolutional neural networks



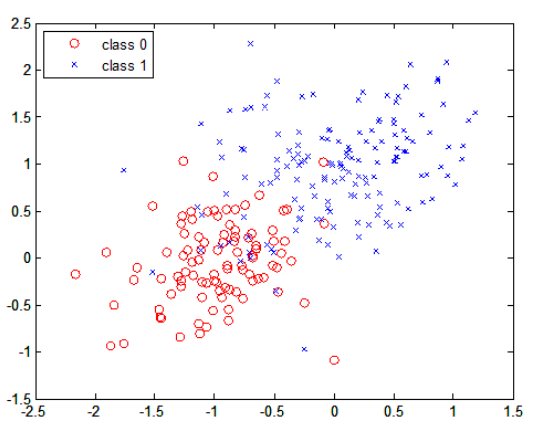
adeshpande3.github.io

Rosenblatt’s Perceptron

Content

- Linear classification
- Kernel trick
- Learning representation
- Universal approximation theorem

Binary linear classification



Hyperplan separator - the founding idea

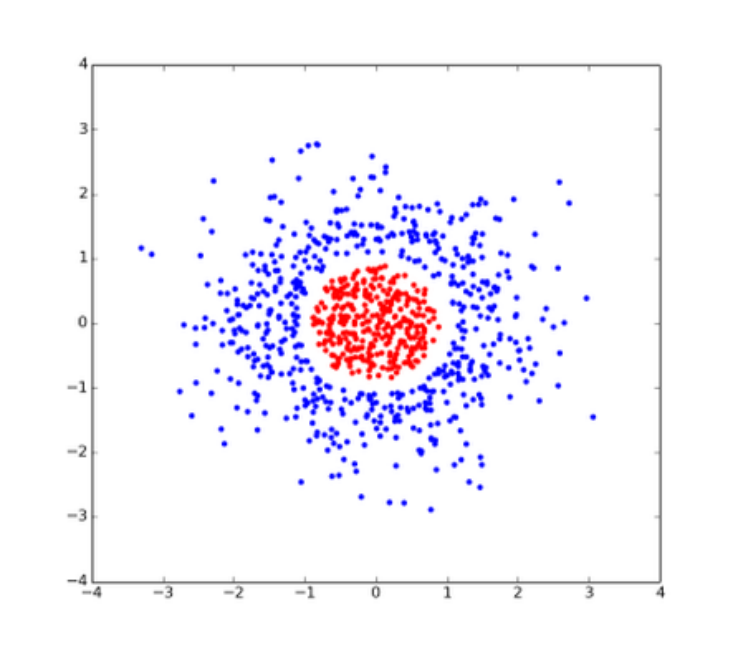
The separator is defined as $(w, b) \in \mathcal{R}^d \times \mathcal{R}$ such that:

- $w \cdot x + b > 0 \rightarrow$ positive class
- $w \cdot x + b < 0 \rightarrow$ negative class

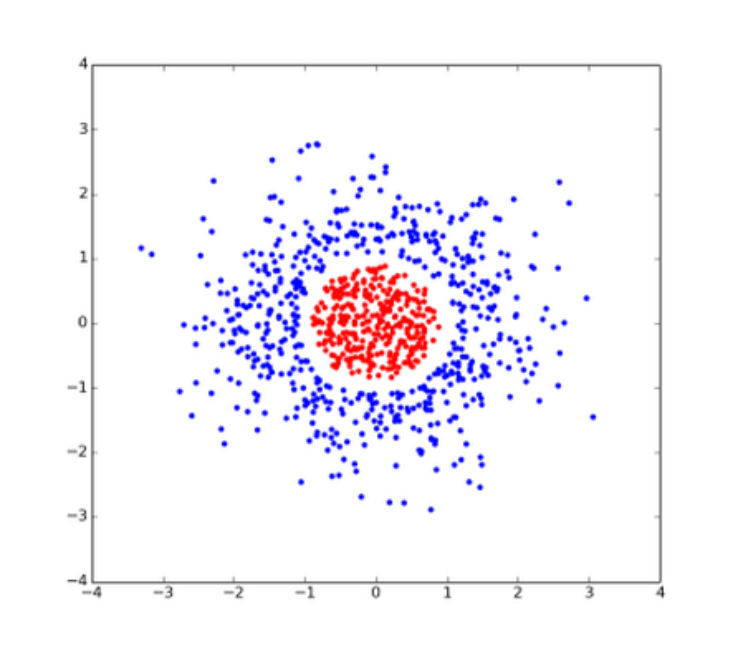
Learning w, b

1. Logistic regression: $\mathbb{P}(+|x) = \frac{1}{1+e^{-(w \cdot x + b)}}$
2. SVM: Minimize $||w||^2$ st $y_i(w \cdot x_i + b) \geq 1, 1 \leq i \leq n$

Kernel trick



Kernel trick

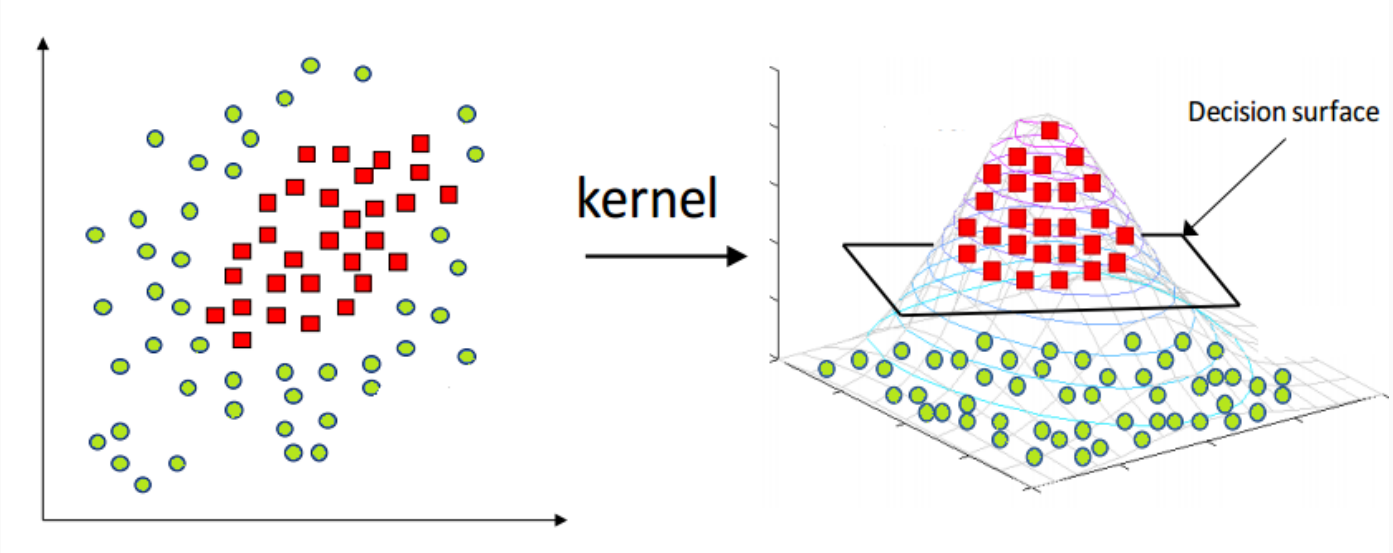


Mercer's theorem

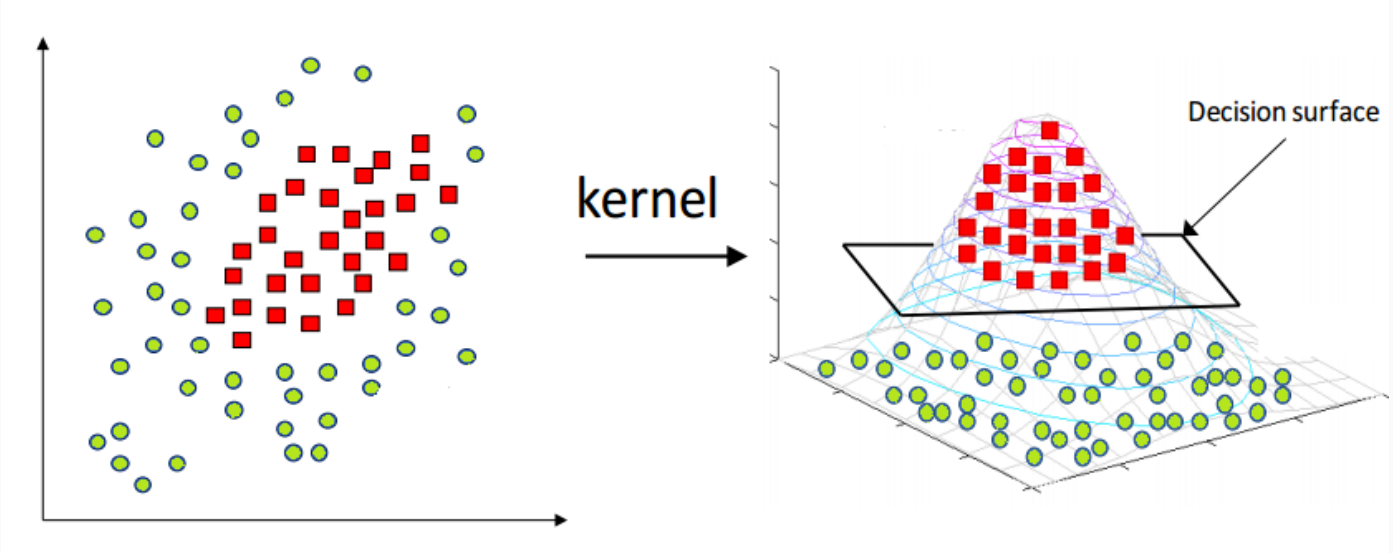
Let $K : \mathcal{R}^d \times \mathcal{R}^d$ continuous, symetric and semi-definite positive, then it exists $\varphi : \mathcal{R}^d \to \mathcal{R}^{d'}$ such that:

$$\forall (x, y) \in \mathcal{R}^d, K(x, y) = \varphi(x)\varphi(y)$$

Kernel trick



Kernel trick

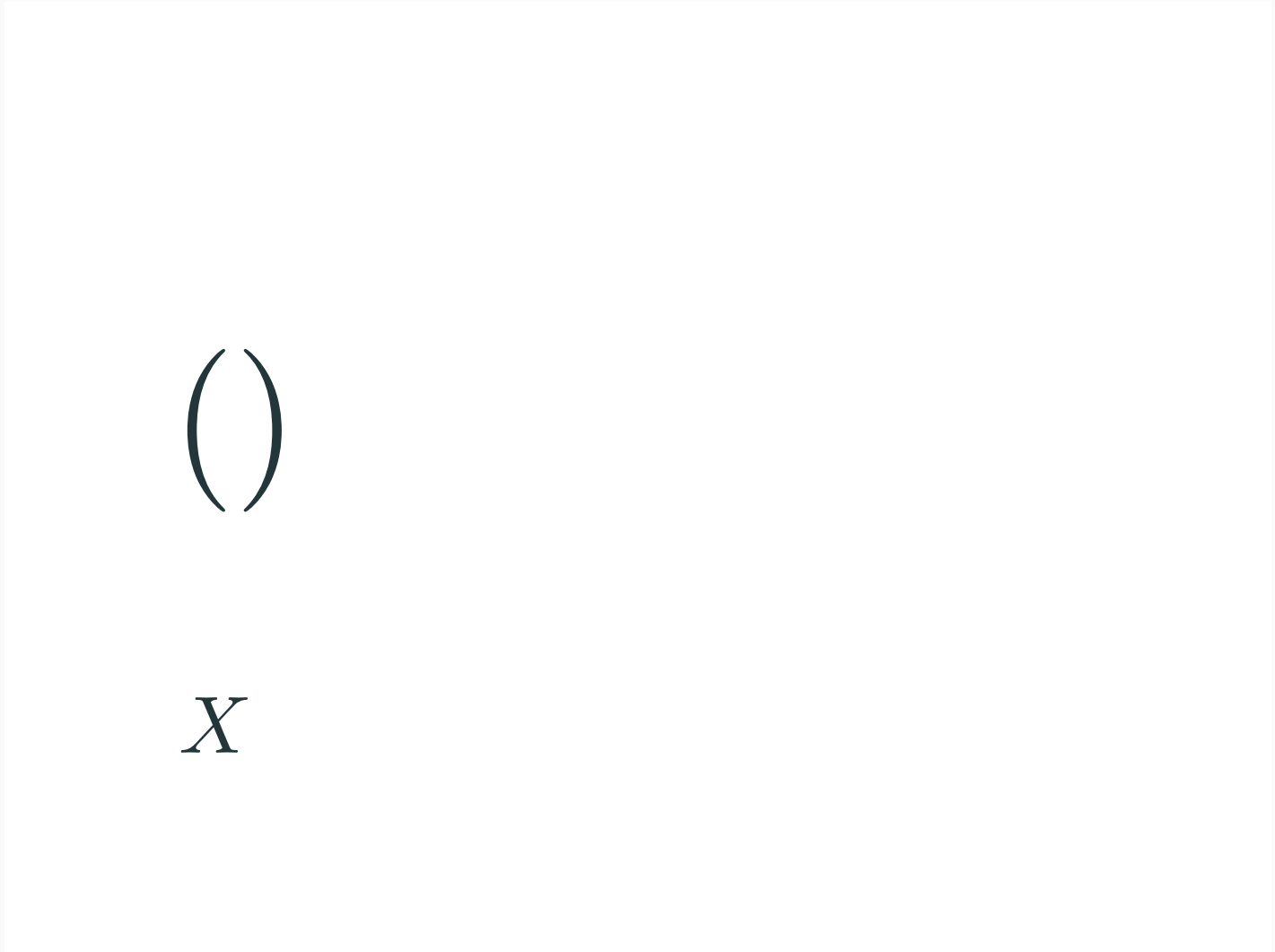


Limitations

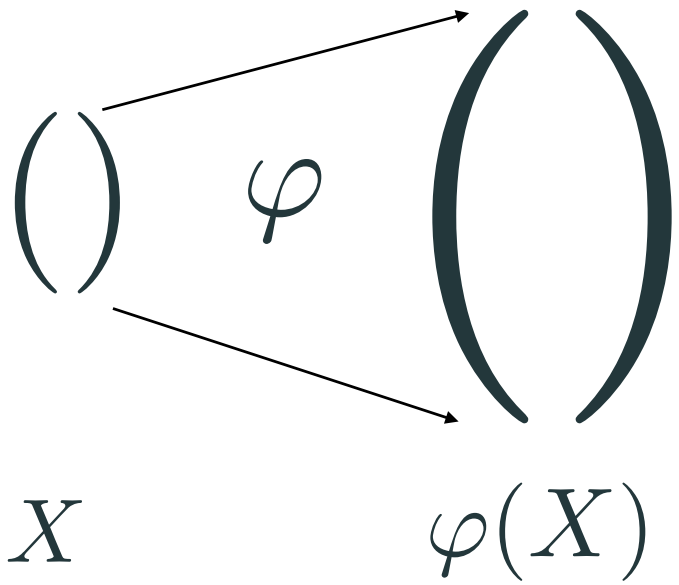
- Linear kernel $K(x, y) = x \cdot y$
- Polynomial kernel $K(x, y) = (1 + x \cdot y)^d$
- Gaussian kernel $K(x, y) = \exp(-\gamma||x - y||^2)$
- Quadratic kernel $K(x, y) = (1 + \gamma||x - y||^2)^{-\alpha}$

There is no reason for this kernel to work...

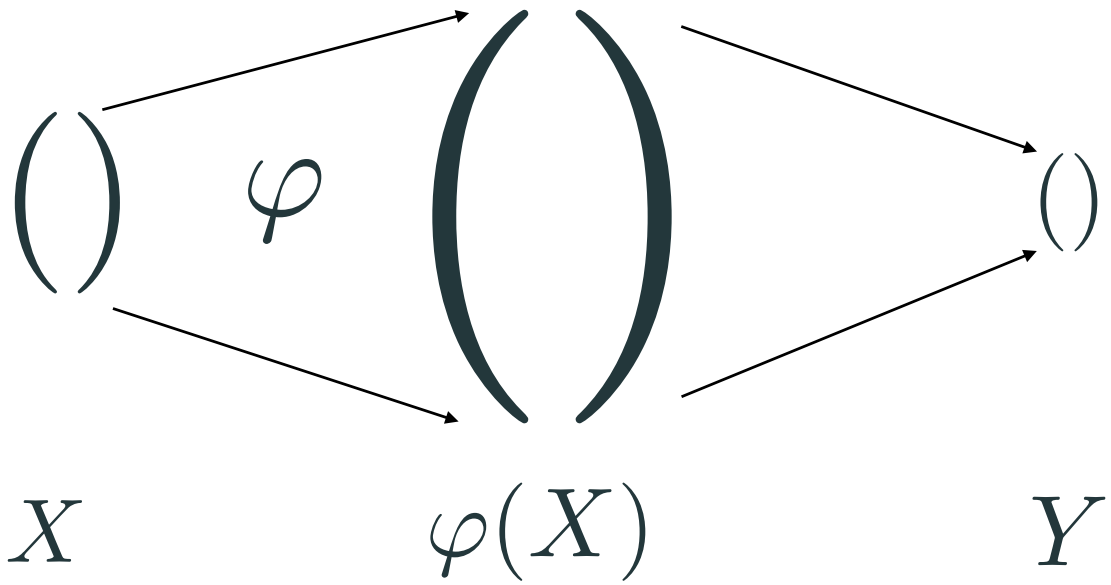
Learning internally the representation kernel



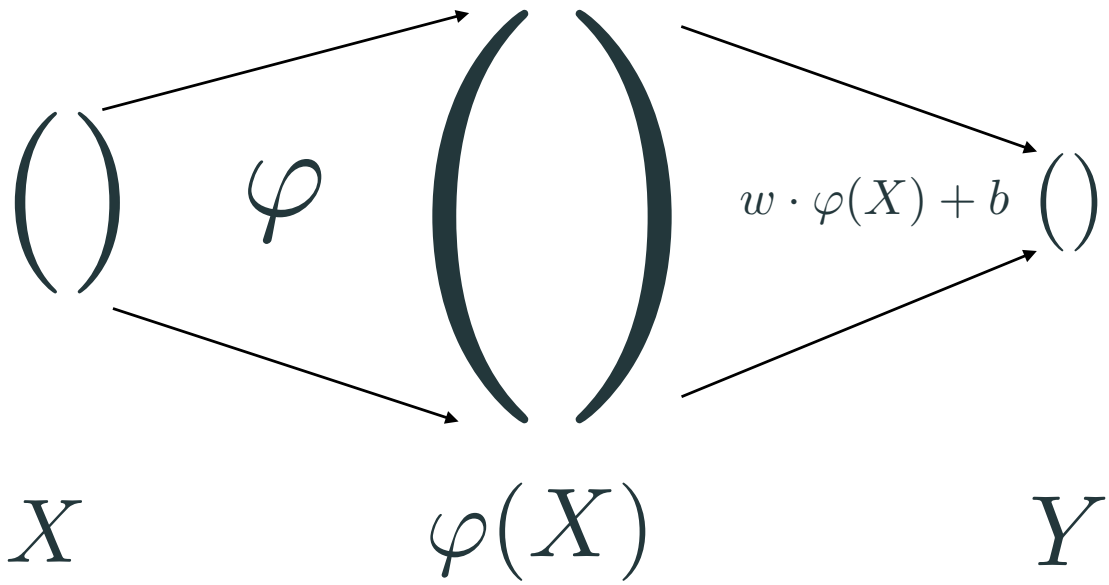
Learning internally the representation kernel



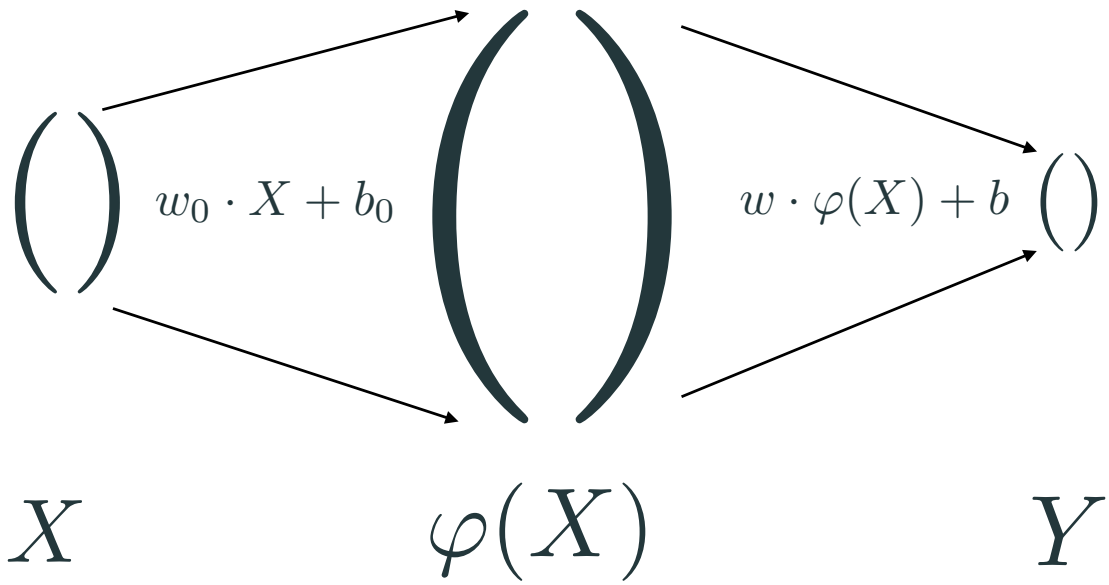
Learning internally the representation kernel



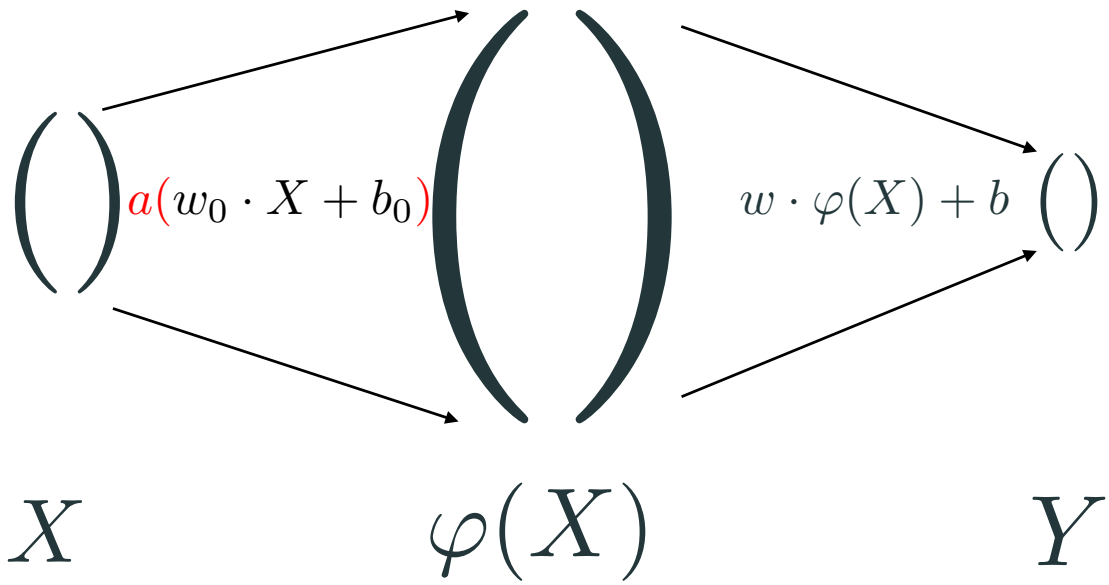
Learning internally the representation kernel



Learning internally the representation kernel



Learning internally the representation kernel



Universal approximation theorem

Universal approximation with One-Hidden-Layer Perceptron

Let denote:

- a non-constant, bounded, continuous function on I.
- F a continuous function of I

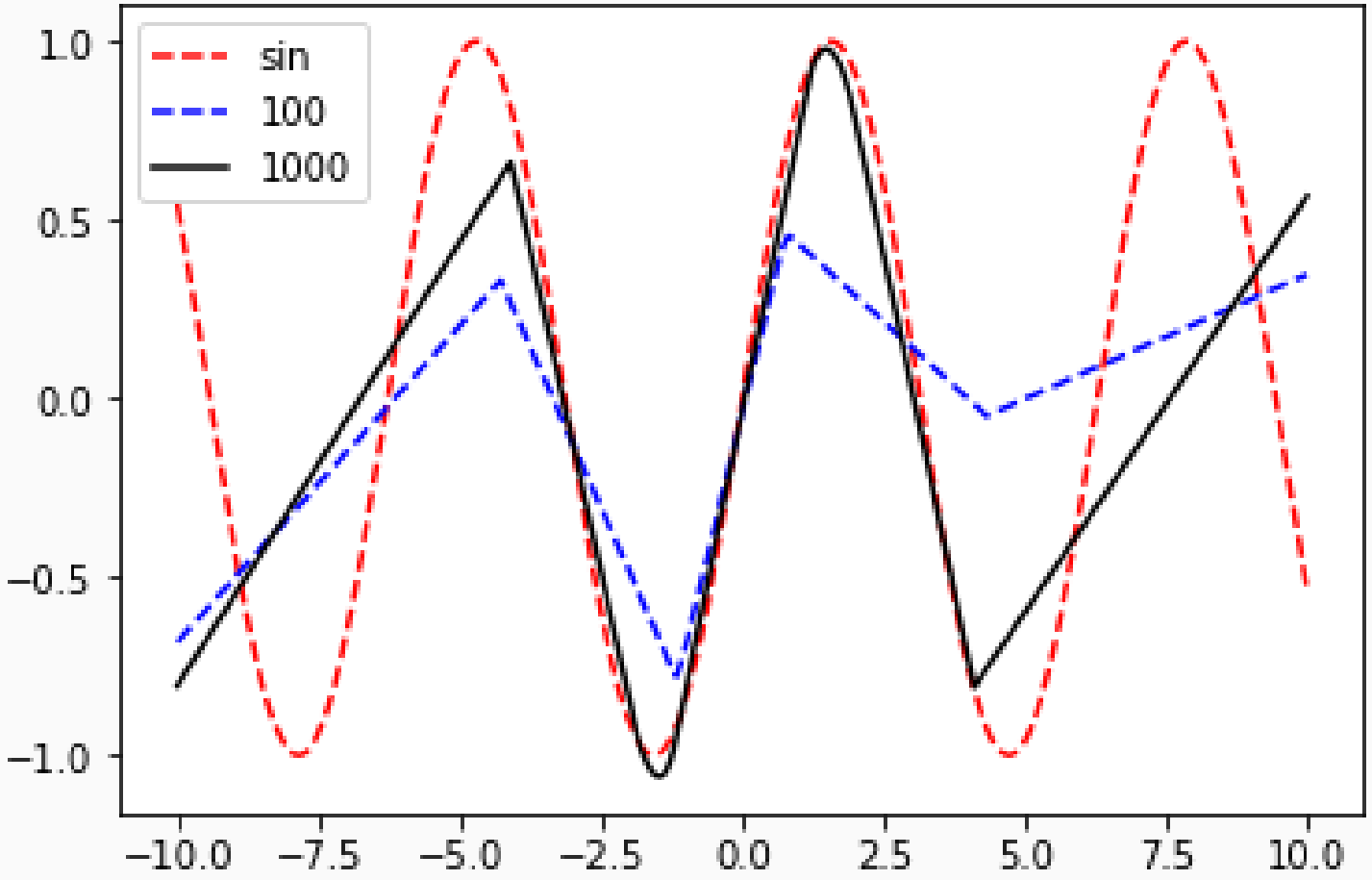
Thus for all $\varepsilon > 0$ it exists an integer n and $(w_i)_{1 \leq i \leq n}$, $(b_i)_{1 \leq i \leq n}$, $(f_i)_{1 \leq i \leq n}$:

$$\sup_I |F - f| < \varepsilon$$

where:

$$\forall x \in I, f(x) = \sum_{i=1}^{n} f_i a(w_i x + b)$$

Regression of sinus



Limitations of Local Template Matching

Content

For now...

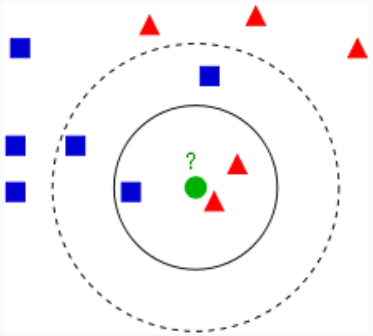
We have defined Deep Learning as

- a feature engineering free algorithm,
- an extension of kernel methods

Let define it by opposition

1. k-NN algorithm
2. Random Forest
3. Support Vector Machine (SVM)
4. Local Template Matching

$k-$Nearest Neighbors



Nearest Neighbors algorithm

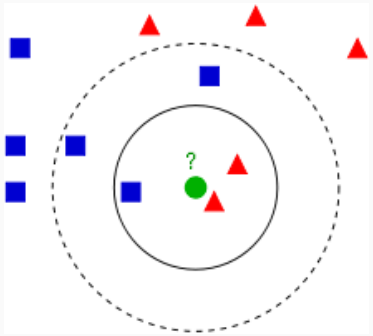
- Training data $(x_i, y_i)_{1 \leq i \leq n}$
- A distance: Euclidean, Manhattan, ...
- Inference:
  - For a new input x, find the $k-$nearest neighbors of x:

$$d(x, x_i) \rightarrow (x_j, y_j)_{1 \leq j \leq k}$$

  - Compute $\hat{y}$:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \frac{\sum_{j=1}^{k} 1\{y_j = y\}}{k}$$

$k-$Nearest Neighbors



Nearest Neighbors algorithm

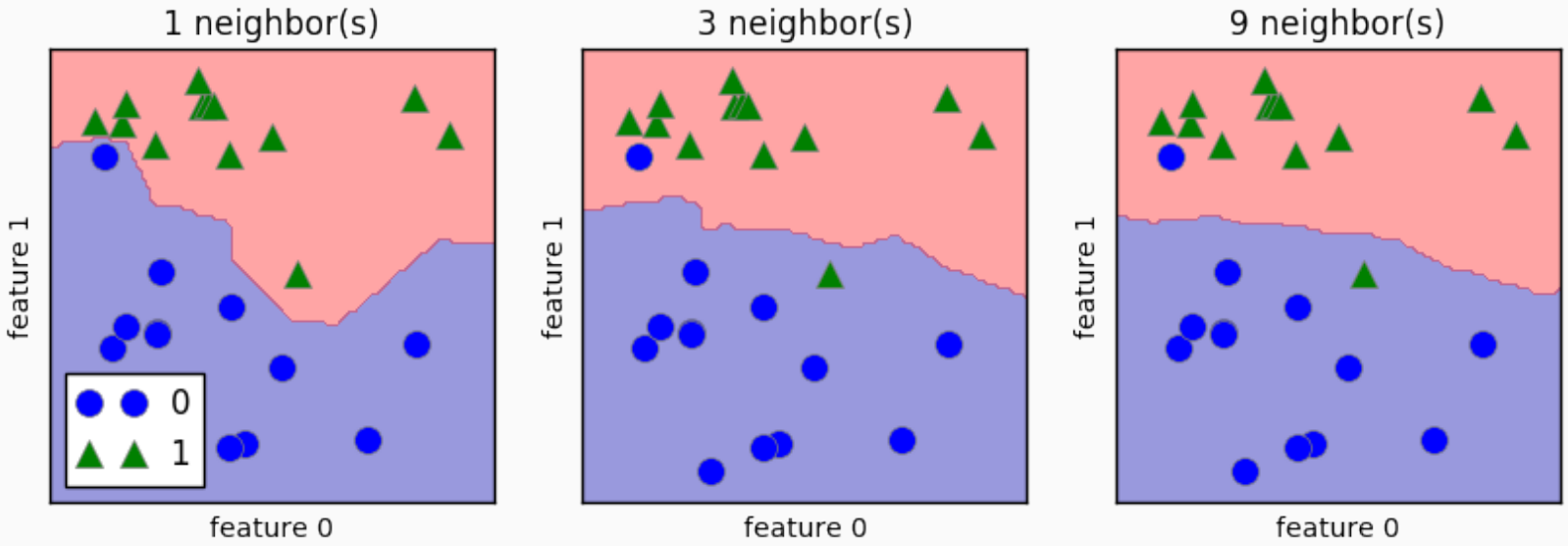
- Training data $(x_i, y_i)_{1 \leq i \leq n}$
- A distance: Euclidean, Manhattan, ...
- Inference:
  - For a new input x, find the $k-$nearest neighbors of x:

$$d(x, x_i) \rightarrow (x_j, y_j)_{1 \leq j \leq k}$$

  - Compute $\hat{y}$:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{j=1}^{k} w(x_j, x) 1\{y_j = y\}$$

$k-$Nearest Neighbors



Random Forest

Decision Tree

- Training data (x_i, y_i) with $x = (x^1, ..., x^p)$ and $y \in \{1, ..., m\}$
- A segmentation criterion:
  - Gini diversity: $I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i)$
  - Information gain: $I_E(f) = -\sum_{i=1}^{m} f_i \log f_i$

Repeat until convergence:

- For each partition
  - For $1 \leq j \leq p$:
    - Find α_j such that the partition $x_j <$ or $> \alpha_j$ gives a minimal value of the criterion
  - Select the feature x_j and α_j with the minimal value of the criterion.

Random Forest

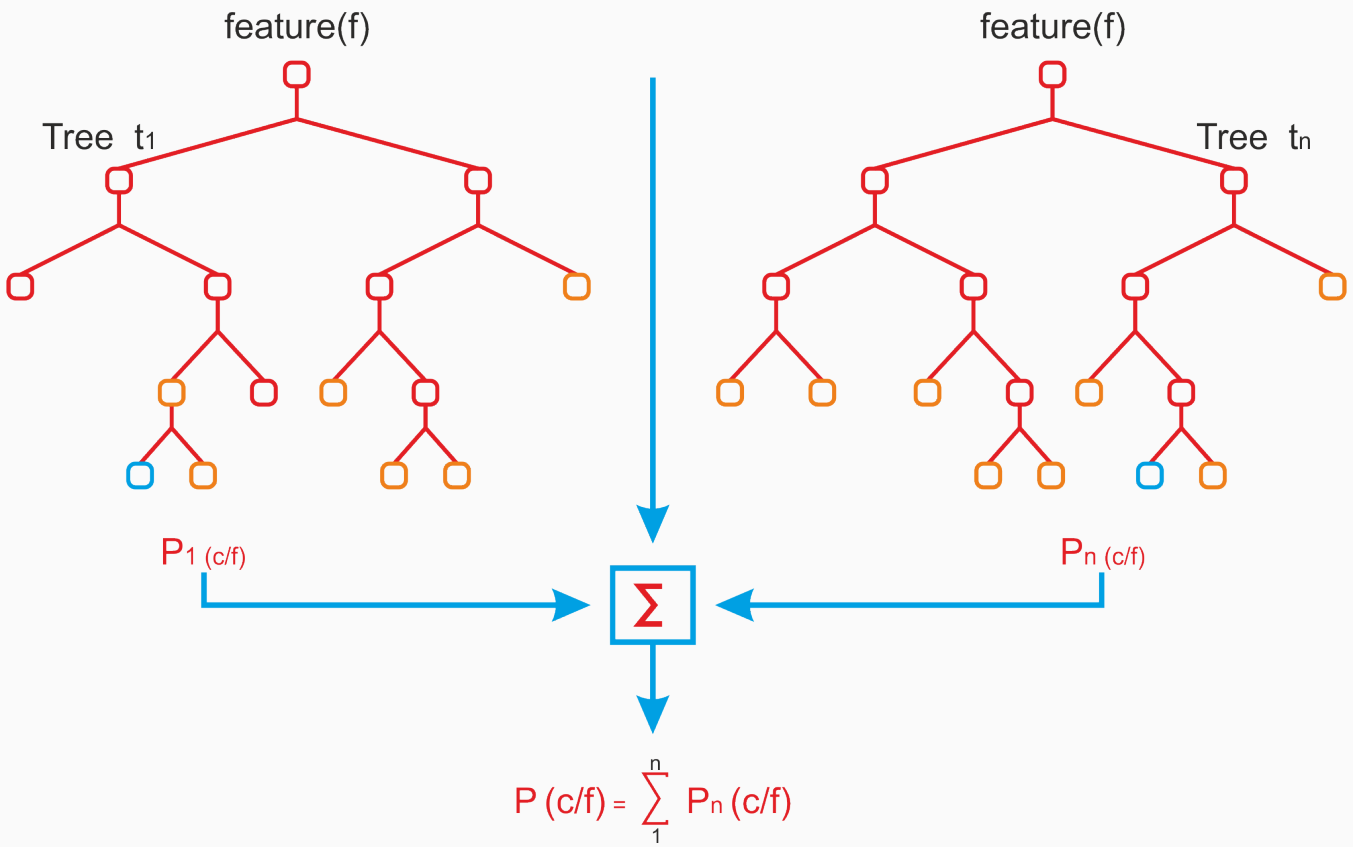
Decision Tree

- Training data (x_i, y_i) with $x = (x^1, ..., x^p)$ and $y \in \{1, ..., m\}$
- A segmentation criterion:
  - Gini diversity: $I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i)$
  - Information gain: $I_E(f) = -\sum_{i=1}^{m} f_i \log f_i$

Random Forest

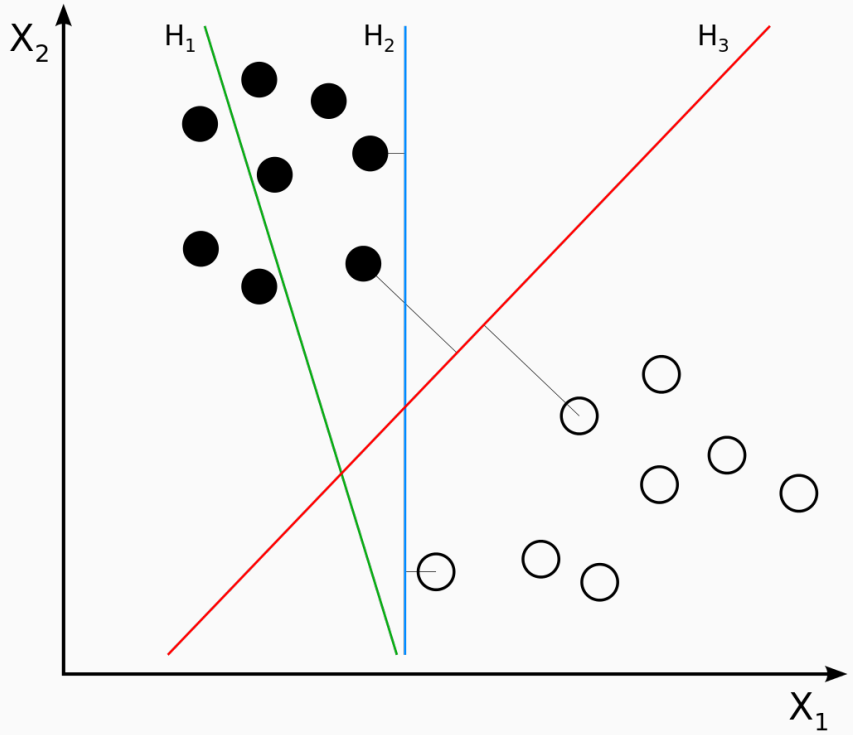
- Decision tree overfits the training data very easily,
- Aggregating many decision trees on different subsets of the training data,
- Inference is a majority vote of trees.

Random Forest

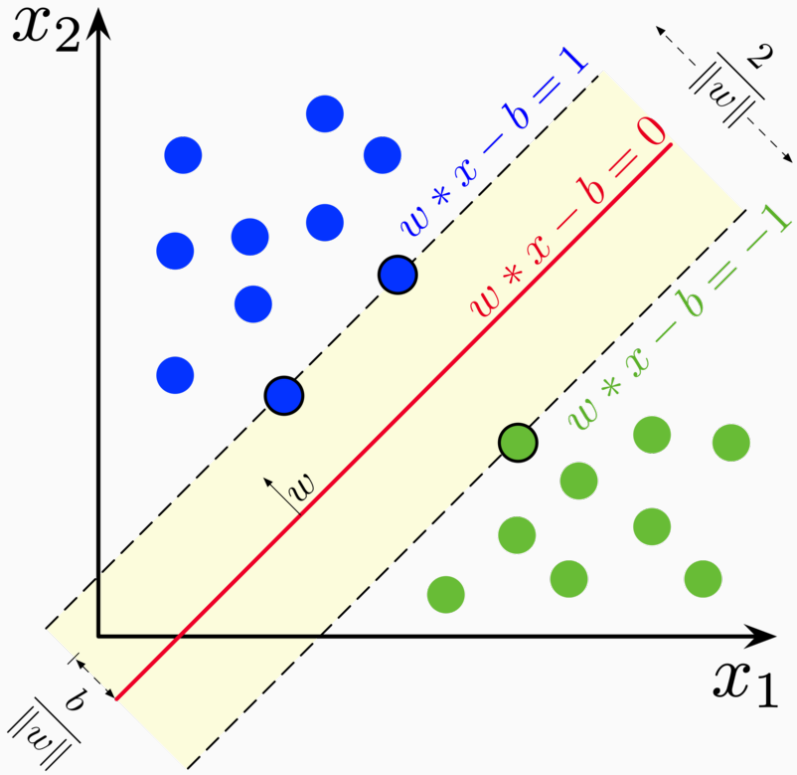


From O'Reilly media

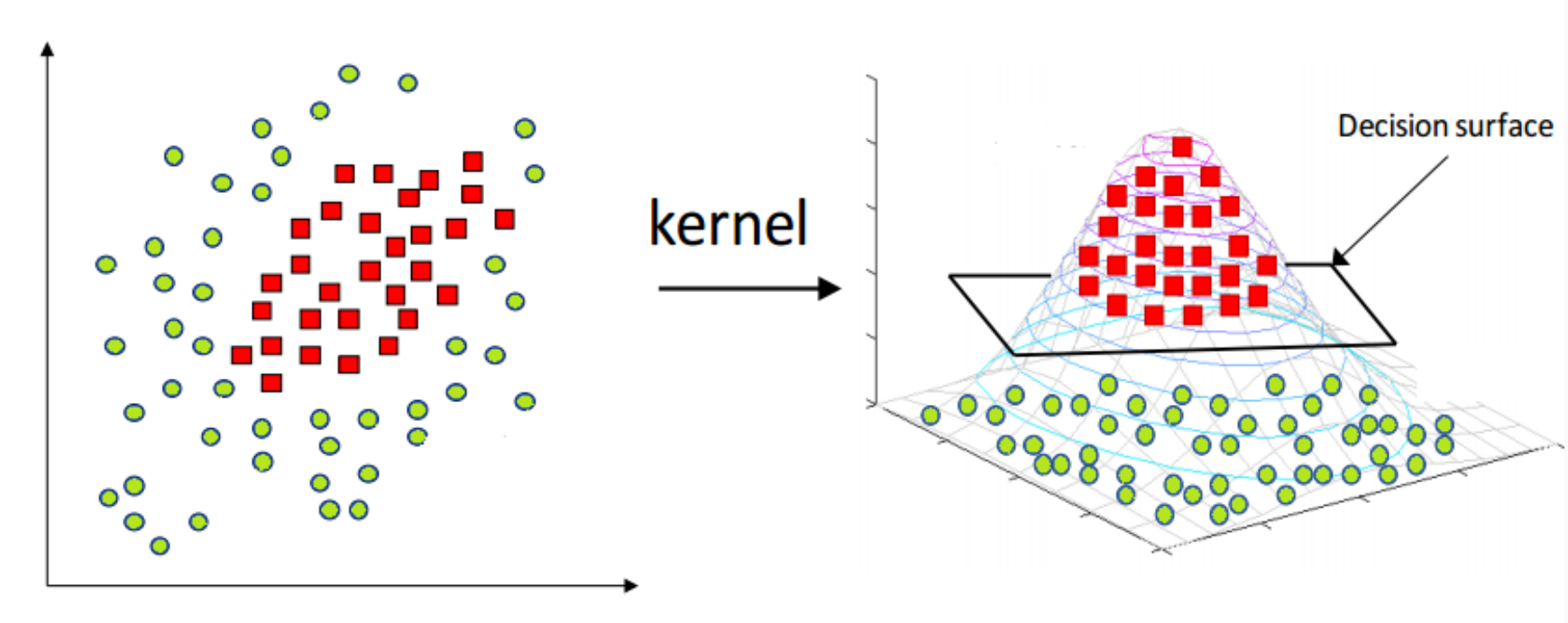
Support Vector Machine (SVM)



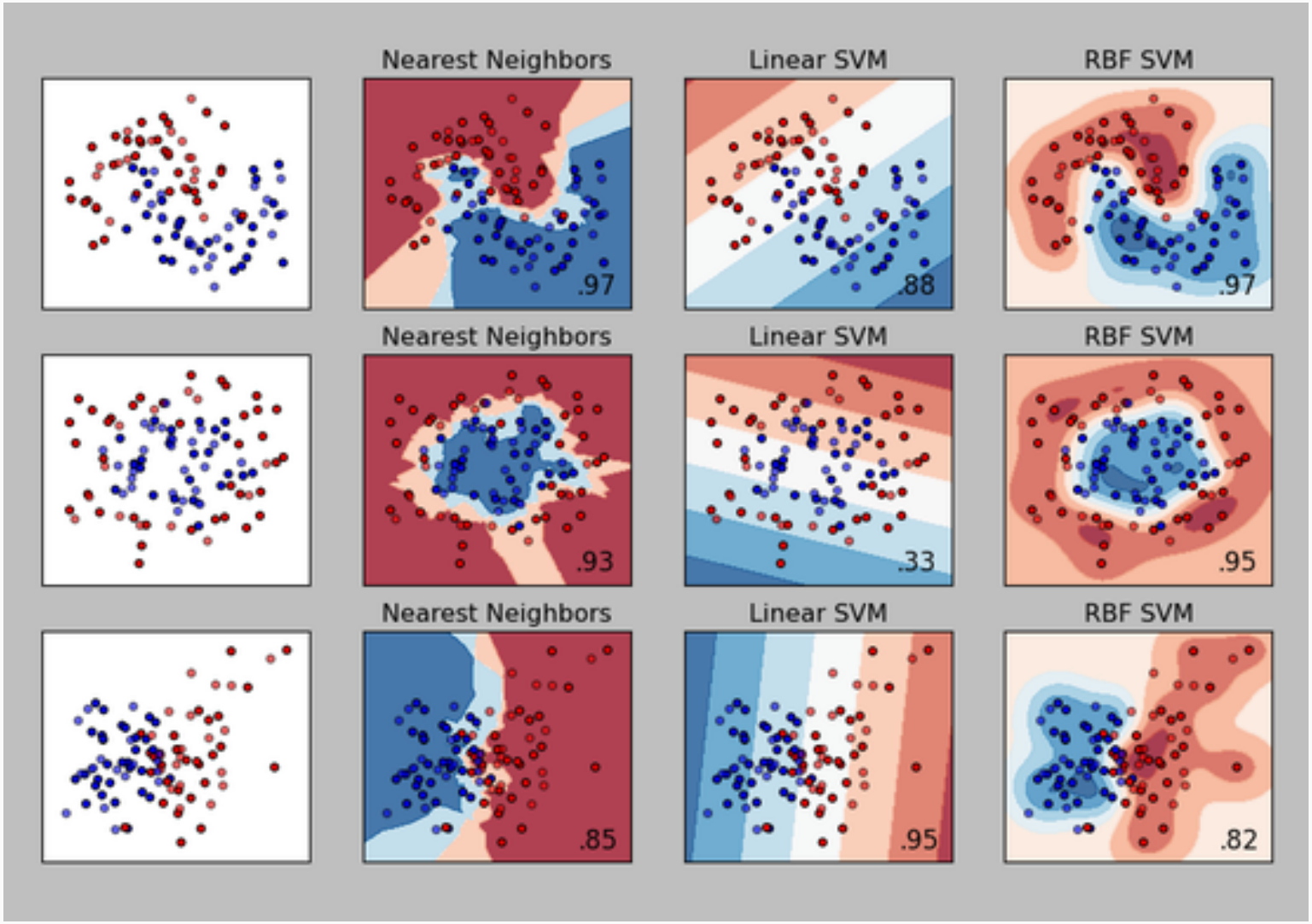
Support Vector Machine (SVM)



Kernel trick



Non-linear separation



Those algorithms do local template matching

$$\hat{y}(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{n} \mathbf{1}(y = y_i) w(x, x_i)$$

- $\mathbf{1}(y = y_i)$: is the vote of the training point x_i
- $w(x, x_i)$ is its contribution to the vote:
  - $w(x, x_i) = 1$
  - $\lim_{||x - x_i|| \to \infty} w(x, x_i) = 0$

Those algorithms do local template matching

$$\hat{y}(x) = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{n} \mathbf{1}(y = y_i) w(x, x_i)$$

- $\mathbf{1}(y = y_i)$: is the vote of the training point x_i
- $w(x, x_i)$ is its contribution to the vote:
  - $w(x, x_i) = 1$
  - $\lim_{||x - x_i|| \to \infty} w(x, x_i) = 0$

Local template matching

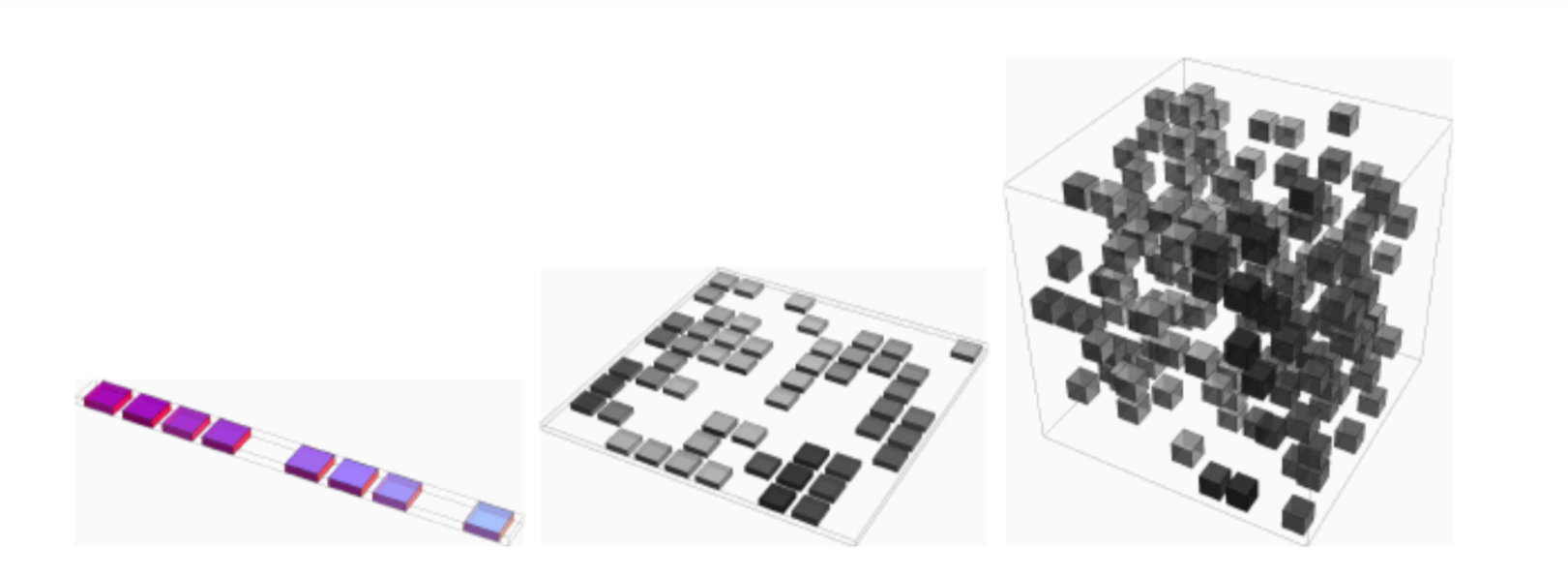
- $k - NN$: it's definition...
- SVM:

$$\sum_{i=1}^{n} \alpha_i y_i \varphi(x_i) \cdot \varphi(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i), \quad \alpha_i \in \{0, 1\}$$

- Random Forest divides around training points with hard separation

Limitations of local template matching

Curse of dimensionality



Limitations of local template matching

Curse of dimensionality

Limitations of local template matching

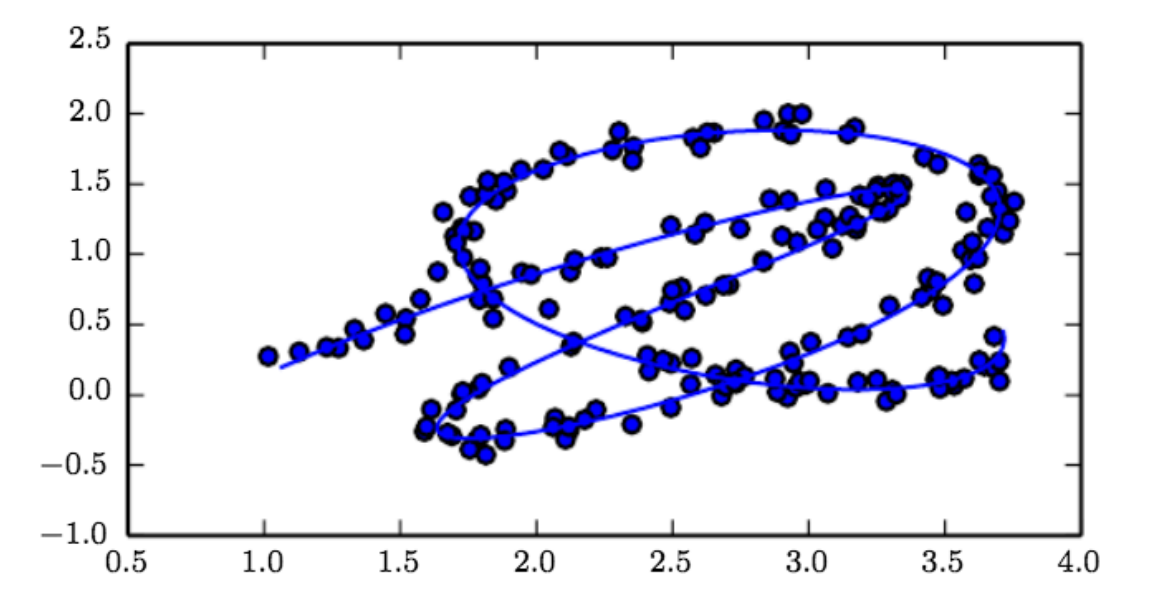
Curse of dimensionality

Limitations of local template matching

Curse of dimensionality

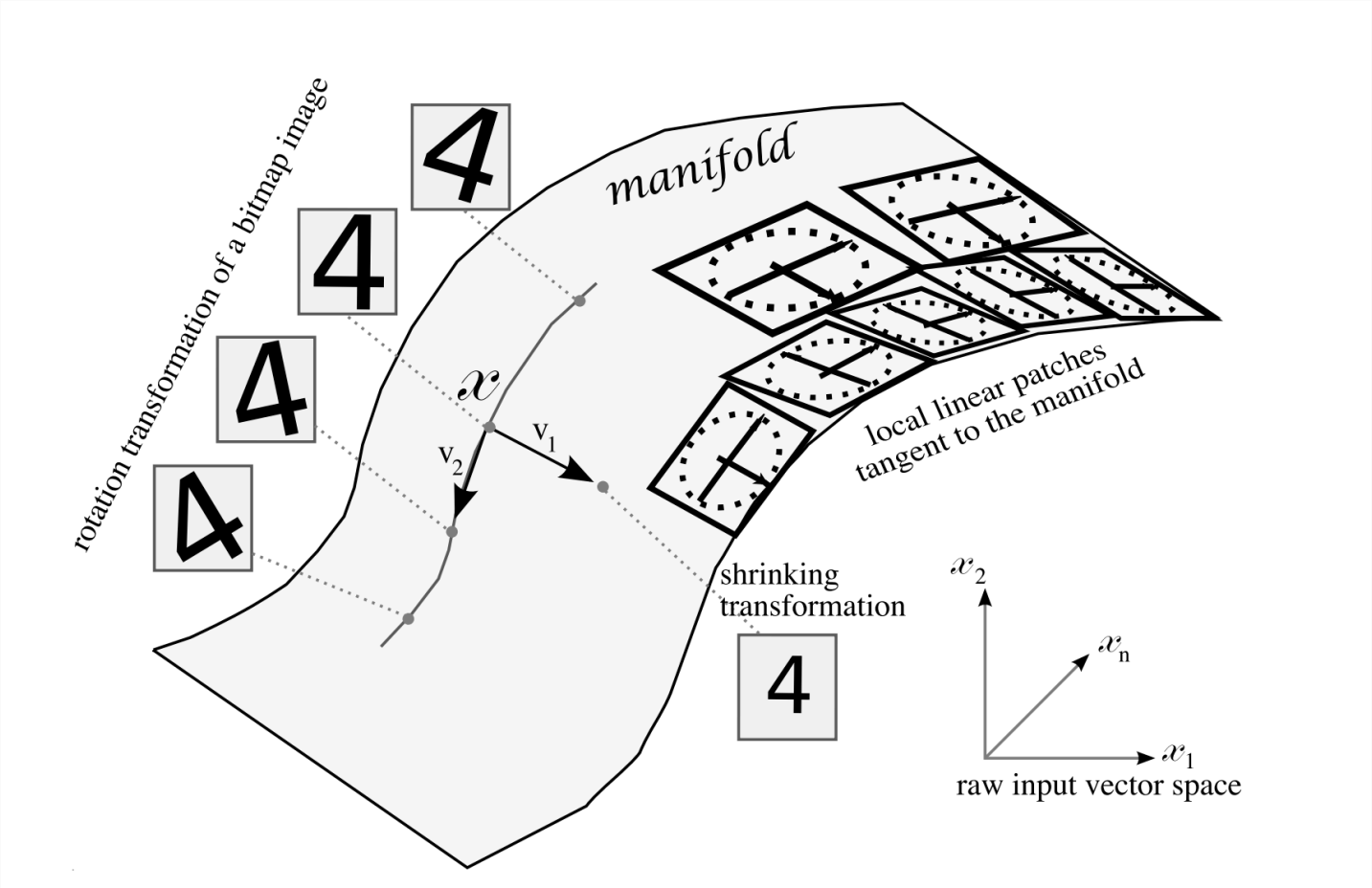
Limitations of local template matching

Manifold learning



Limitations of local template matching

Manifold learning



Limitations of local template matching

Manifold learning

Deep Feed forward Network

Deep Feed Forwad Network - Motivations

Hierarchical relations in feature generation process

Latent factor $z \rightarrow h_1 = f_1(z) \rightarrow h_2 = f_2(h_1) \rightarrow ... \rightarrow x = f_n(h_{n-1})$

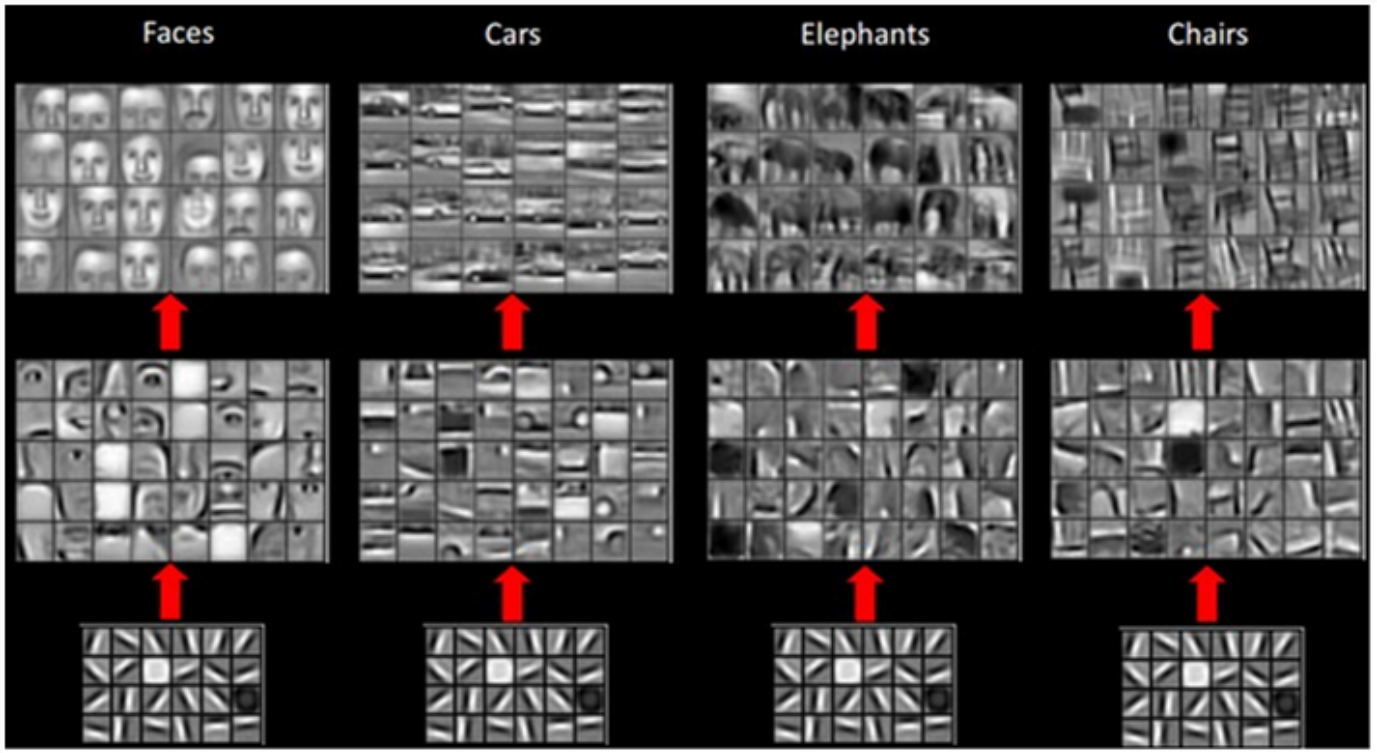
- x may lie in very high dimensional space
- f_i is a very simple non-linear function

Deep Feed Forwad Network - Motivations

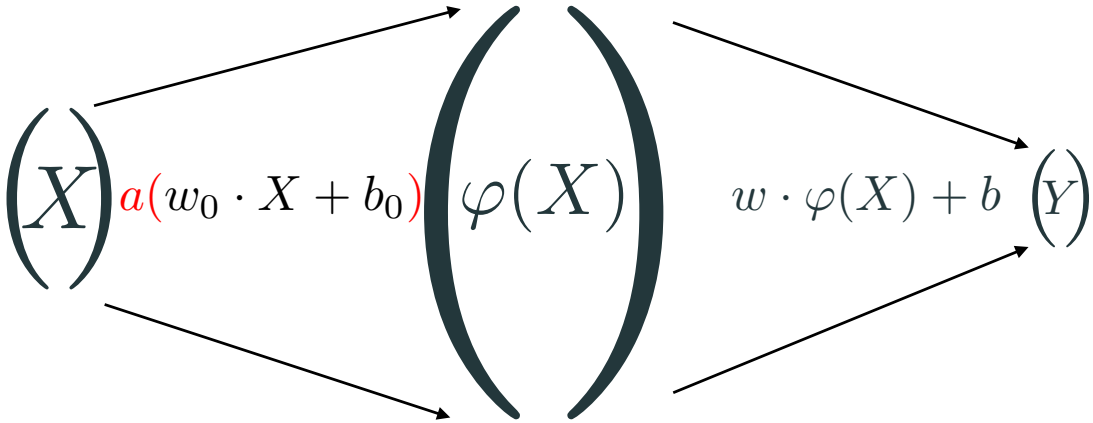
Hierarchical relations in feature generation process

Latent factor $z \rightarrow h_1 = f_1(z) \rightarrow h_2 = f_2(h_1) \rightarrow ... \rightarrow x = f_n(h_{n-1})$

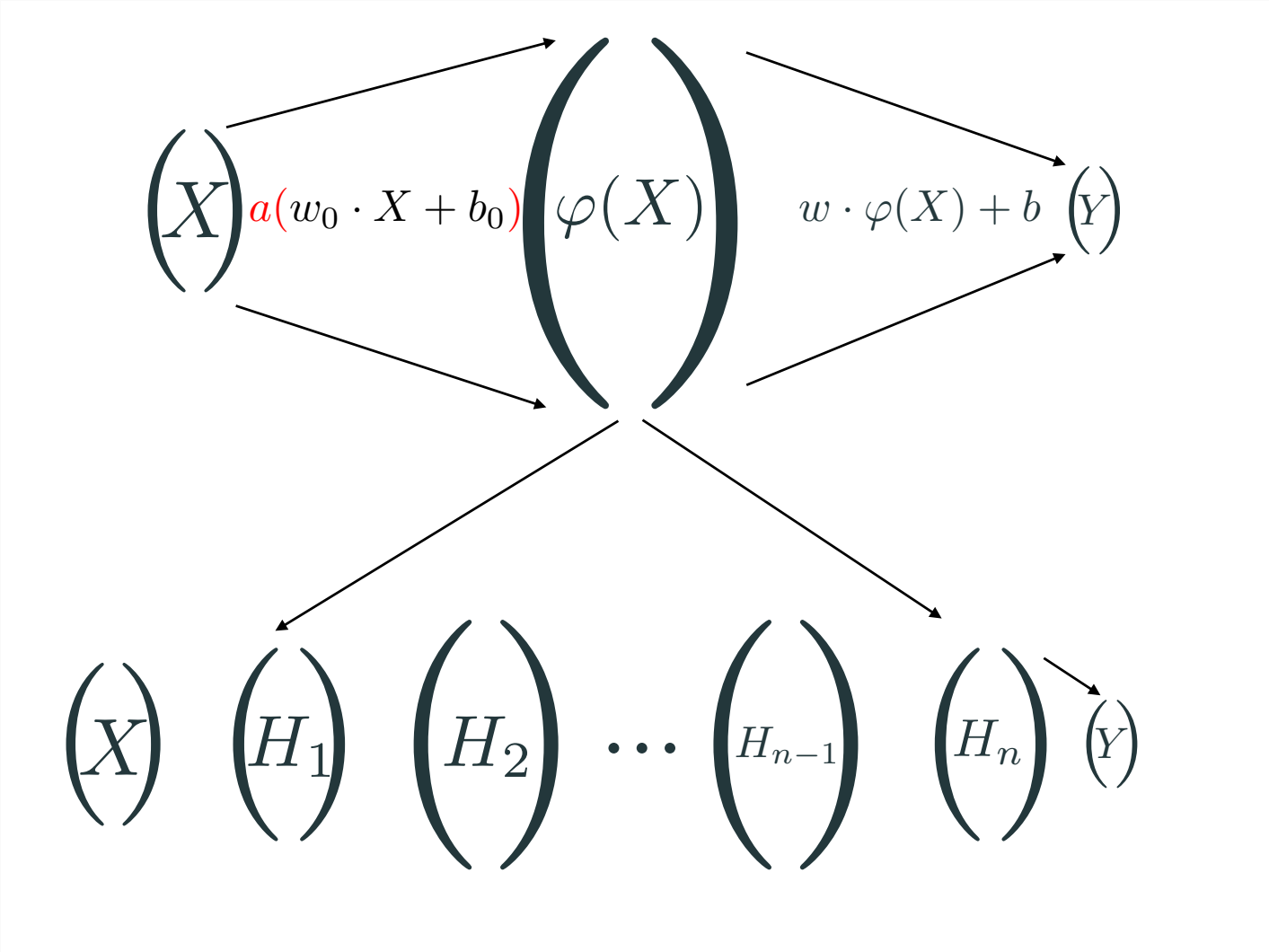
- x may lie in very high dimensional space
- f_i is a very simple non-linear function



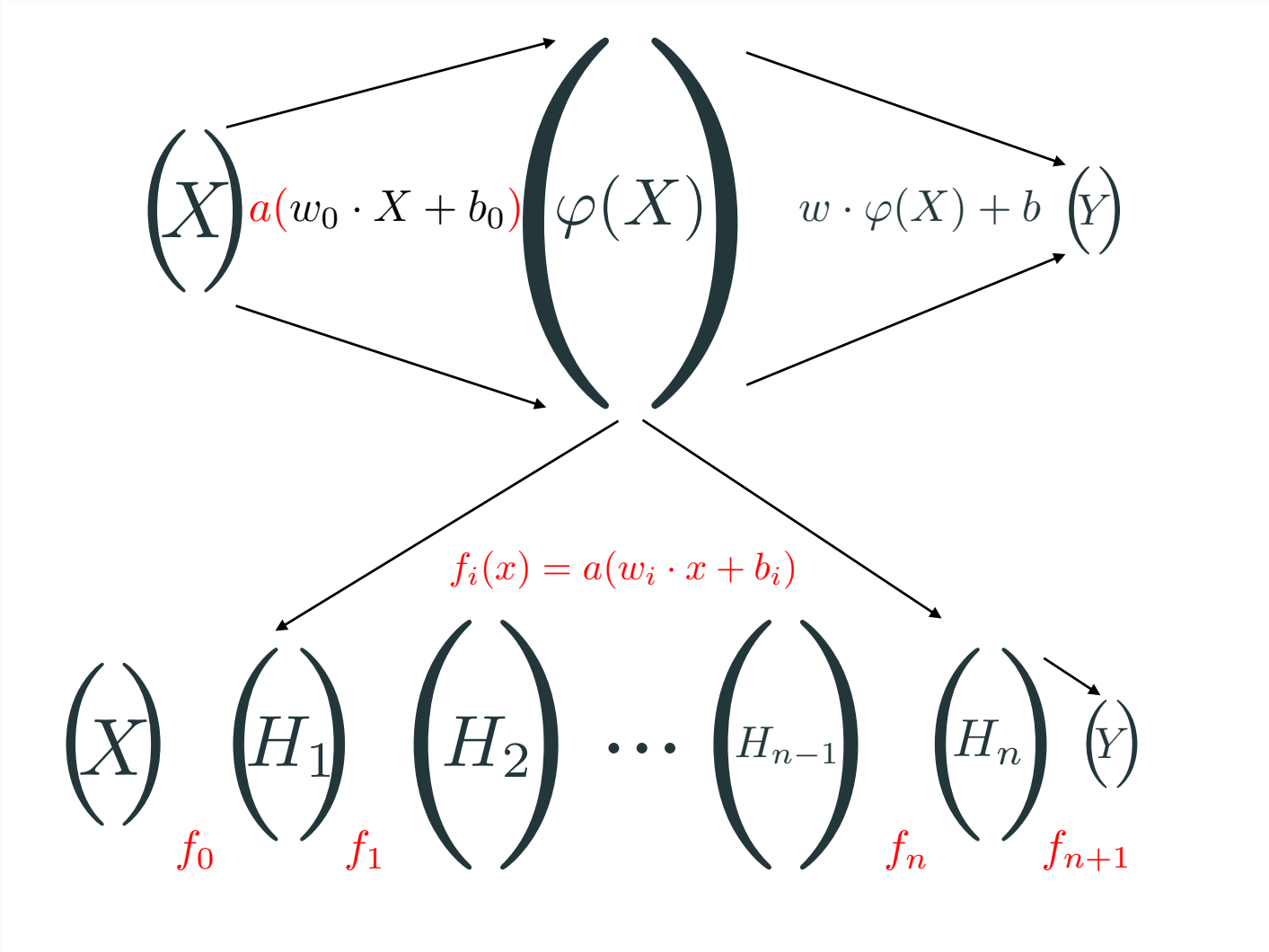
Architecture design - Layers and Depth of the network



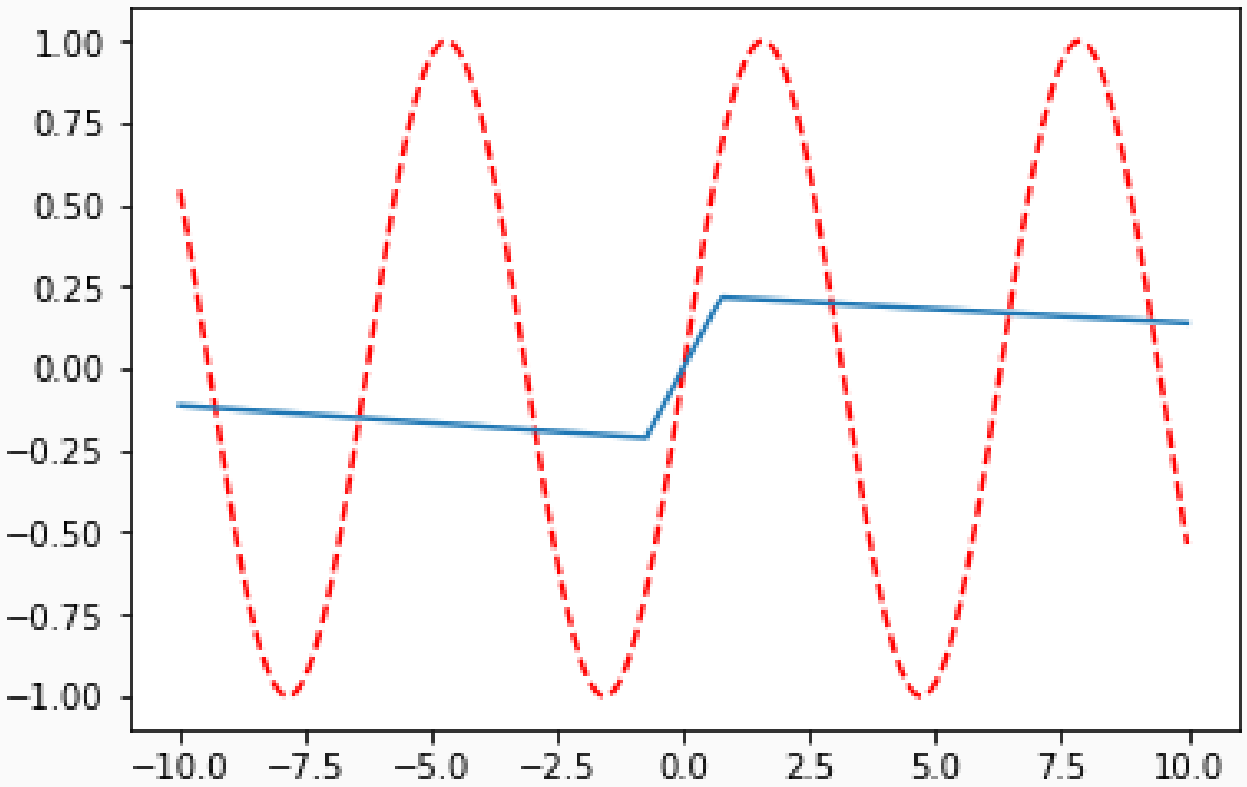
Architecture design - Layers and Depth of the network



Architecture design - Layers and Depth of the network

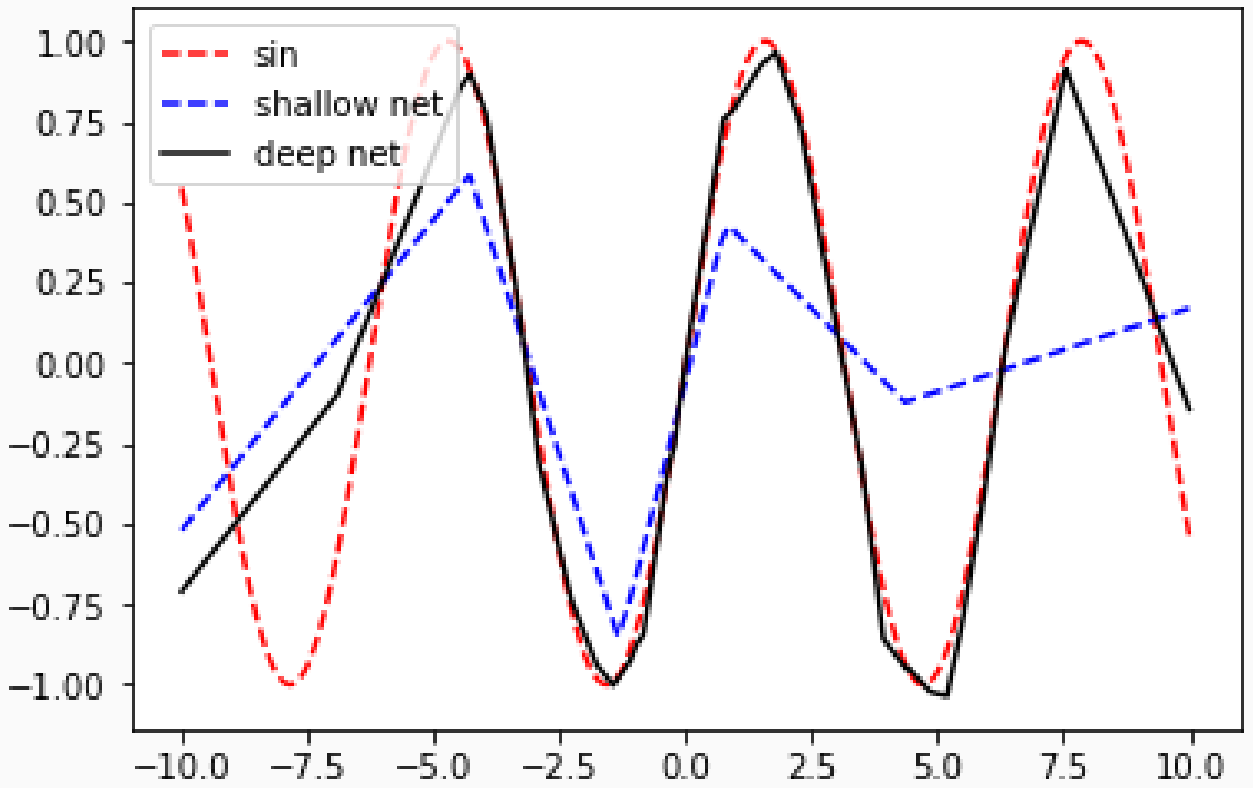


Exponential gain of DEPTH



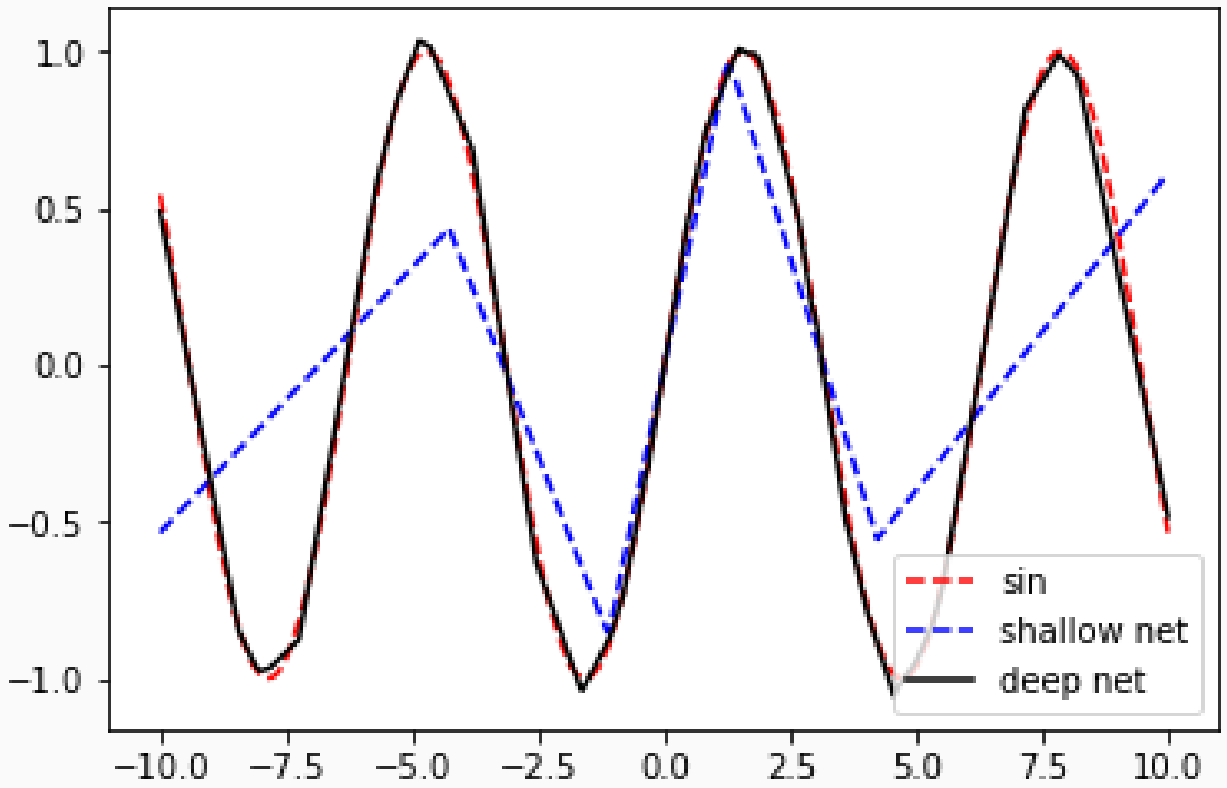
- 1 layer
- 10 hidden states

Exponential gain of DEPTH



- 3 layers
- 10 hidden states

Exponential gain of DEPTH



- 5 layers
- 10 hidden states

Training a neural network

Deep Learning library: `keras.io`

Gradient based learning - Differential Learning

For a network $f = f_1 \circ ... \circ f_n$

For each layer i:

$$f_i(x) = a(w_i \cdot x + b_i)$$

- We need to estimate:
  - weight: w_i
  - bias: b_i
- Such that the output of the network f is y: $f(x) = \hat{y}(x) \approx y$

Hopefully, each internal operation in the network is differentiable

GRADIENT BASED LEARNING

From a metric to optimize to a loss to minimize

Differentiable proxy of a metric

- Defining a metric,
- Determining a proxy, called the loss, which is differentiable with respect to the network parameters.

Classical losses

- Regression:

From a metric to optimize to a loss to minimize

Differentiable proxy of a metric

- Defining a metric,
- Determining a proxy, called the loss, which is differentiable with respect to the network parameters.

Classical losses

- Regression: Mean Squared Errors
- Classification:

From a metric to optimize to a loss to minimize

Differentiable proxy of a metric

- Defining a metric,
- Determining a proxy, called the loss, which is differentiable with respect to the network parameters.

Classical losses

- Regression: Mean Squared Errors
- Classification: Binary cross-entropy, categorical cross-entropy, Divergence KL.

From a metric to optimize to a loss to minimize

Differentiable proxy of a metric

- Defining a metric,
- Determining a proxy, called the loss, which is differentiable with respect to the network parameters.

Classical losses

- Regression: Mean Squared Errors
- Classification: Binary cross-entropy, categorical cross-entropy, Divergence KL.
- Cosine similarity,
- Hinge loss

Gradient based learning: Back-Propagation algorithm

$$x = h_0 \overset{\theta_1}{\rightarrow} h_1 \overset{\theta_2}{\rightarrow} h_2 \overset{\theta_3}{\rightarrow} \ldots \overset{\theta_{n-1}}{\rightarrow} h_{n-1} \overset{\theta_n}{\rightarrow} h_n = \hat{y}$$

Gradient based learning: Back-Propagation algorithm

$$x = h_0 \overset{\theta_1}{\rightarrow} h_1 \overset{\theta_2}{\rightarrow} h_2 \overset{\theta_3}{\rightarrow} \ldots \overset{\theta_{n-1}}{\rightarrow} h_{n-1} \overset{\theta_n}{\rightarrow} h_n = \hat{y}$$

$$\downarrow$$

$$\ell(y, \hat{y})$$

Gradient based learning: Back-Propagation algorithm

$$x = h_0 \xrightarrow{\theta_1} h_1 \xrightarrow{\theta_2} h_2 \xrightarrow{\theta_3} \ldots \xrightarrow{\theta_{n-1}} h_{n-1} \xrightarrow{\theta_n} h_n = \hat{y}$$

$$\downarrow$$

$$\ell(y, \hat{y})$$

$$\downarrow$$

$$\text{Compute } \frac{\partial \ell}{\partial \theta_i}$$

Gradient based learning: Back-Propagation algorithm

Gradient back-propagation

- $$\frac{\partial \ell}{\partial \theta_n} = \underbrace{\frac{\partial \ell}{\partial h_n}}_{\nabla} \frac{\partial h_n}{\partial \theta_n} = \nabla \frac{\partial h_n}{\partial \theta_n}$$

Gradient based learning: Back-Propagation algorithm

Gradient back-propagation

- $$\frac{\partial \ell}{\partial \theta_n} = \underbrace{\frac{\partial \ell}{\partial h_n}}_{\nabla} \frac{\partial h_n}{\partial \theta_n} = \nabla \frac{\partial h_n}{\partial \theta_n}$$
-

$$\frac{\partial \ell}{\partial \theta_{n-1}} = \frac{\partial \ell}{\partial h_n} \frac{\partial h_n}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial \theta_{n-1}}$$

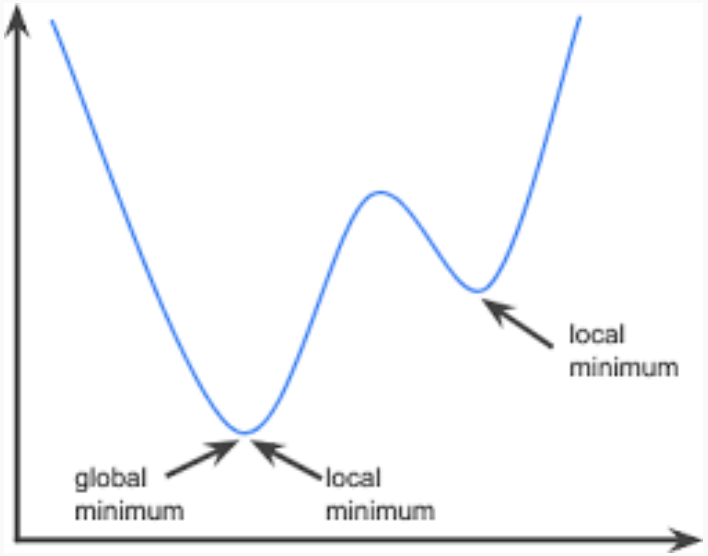
$$= \nabla \frac{\partial h_n}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial \theta_{n-1}}$$

$$\nabla \leftarrow \nabla \frac{\partial h_n}{\partial h_{n-1}}$$

Gradient based learning: Back-Propagation algorithm

Gradient back-propagation

- $$\frac{\partial \ell}{\partial \theta_n} = \underbrace{\frac{\partial \ell}{\partial h_n}}_{\nabla} \frac{\partial h_n}{\partial \theta_n} = \nabla \frac{\partial h_n}{\partial \theta_n}$$
- $$\frac{\partial \ell}{\partial \theta_{n-1}} = \frac{\partial \ell}{\partial h_n} \frac{\partial h_n}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial \theta_{n-1}}$$
$$= \nabla \frac{\partial h_n}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial \theta_{n-1}}$$
$$\nabla \leftarrow \nabla \frac{\partial h_n}{\partial h_{n-1}}$$
- $$\frac{\partial \ell}{\partial \theta_{n-2}} = \frac{\partial \ell}{\partial h_{n-1}} \frac{\partial h_{n-1}}{\partial h_{n-2}} \frac{\partial h_{n-2}}{\partial \theta_{n-2}} = \nabla \frac{\partial h_{n-1}}{\partial h_{n-2}} \frac{\partial h_{n-2}}{\partial \theta_{n-2}}$$



Minimize $J(\theta)$

1. Initialize θ_0
2. Until convergence,

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_\theta J(\theta_t)$$

Gradient Descent with Momentum

`http://ruder.io/optimizing-gradient-descent/`

Gradient based learning: Stochastic Gradient Descent

$$J(\theta) = \frac{1}{n_s} \sum_{i=1}^{n_s} \ell(\hat{y}(x_i; \theta), y_i)$$

Gradient based learning: Stochastic Gradient Descent

$$J(\theta) = \frac{1}{n_s} \sum_{i=1}^{n_s} \ell(\hat{y}(x_i; \theta), y_i)$$

For a sample of data $(x_i, y_i)_{i=1}^{m}$, with $m \ll n_s$, we assume:

$$J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}(x_i; \theta), y_i)$$

Gradient based learning: Stochastic Gradient Descent

$$J(\theta) = \frac{1}{n_s} \sum_{i=1}^{n_s} \ell(\hat{y}(x_i; \theta), y_i)$$

For a sample of data $(x_i, y_i)_{i=1}^{m}$, with $m \ll n_s$, we assume:

$$J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}(x_i; \theta), y_i)$$

Stochastic Gradient Descent

1. Initialize θ_0
2. Until convergence
   - Sample a batch of data $(x_i, y_i)_{i=1}^{m}$ uniformly in the dataset
   - $\theta_{t+1} \leftarrow \theta_t - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \ell(\hat{y}(x_i; \theta_t))$

Gradient based learning: Stochastic Gradient Descent

$$J(\theta) = \frac{1}{n_s} \sum_{i=1}^{n_s} \ell(\hat{y}(x_i; \theta), y_i)$$

For a sample of data $(x_i, y_i)_{i=1}^{m}$, with $m \ll n_s$, we assume:

$$J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}(x_i; \theta), y_i)$$

Advantages

- Reduce time computation
- Prevent to be stuck in a local minimum with a noisy gradient
- Reproduce the test set

Regularization in neural networks

You are highly encouraged to test these regularization during the coding sessions

- L1 and L2 regularization
- Dropout regularization
- Batch Normalization
- Early stopping
- Data augmentation
- Semi-supervised learning
- Aggregation and bagging
- Prior belief on the architecture

TP on Keras

1. Implement the logistic regression with perfectly separable data. Plot the parameters norm with the number of epochs. How can you reduce this parameter explosion?
2. Reproduce the example of **sin** regression while varying the numbers of neurons and the number of layers. Search the best couple using a grid search. Reproduce the same experience while varying activation functions and optimizers.
3. Load the MNIST with Keras. Train a Multi-Layer Perceptron for digit classification.
4. Challenge time!