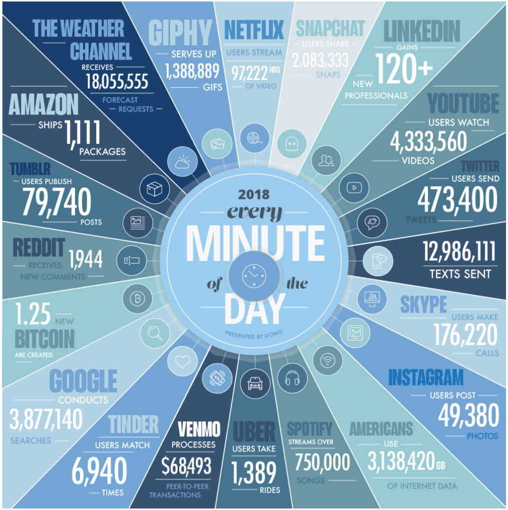
Deep learning: an introduction

Gilles Gasso

INSA Rouen - ITI Department
Laboratory LITIS

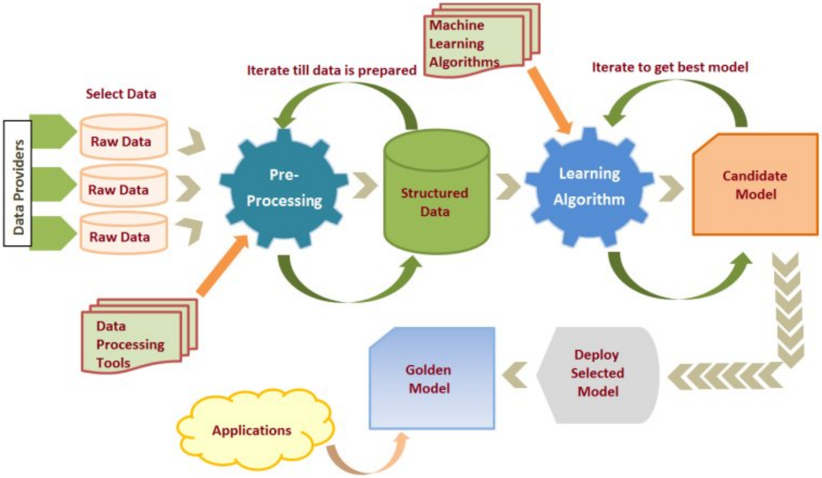
December 8, 2025

From Data...

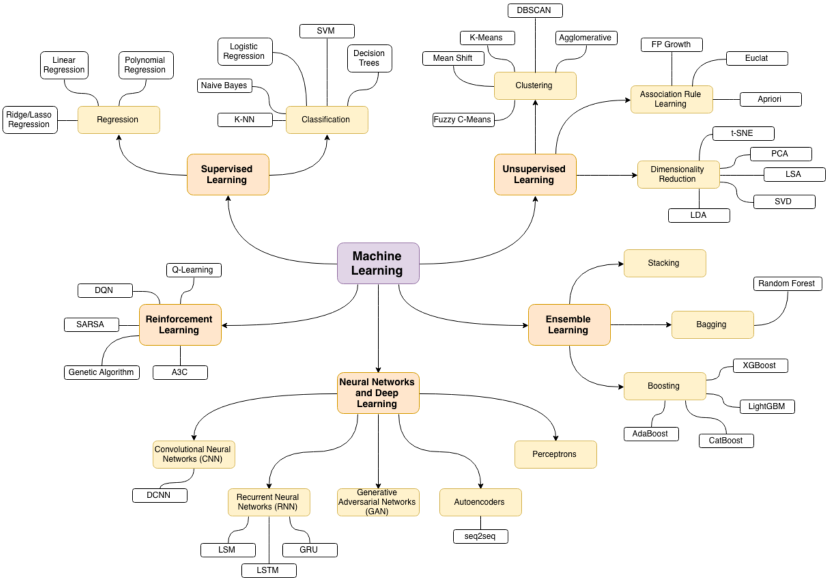


- Text
- Audio
- Images
- Videos
- Graphs ...

to processing...

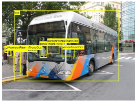


using algorithms

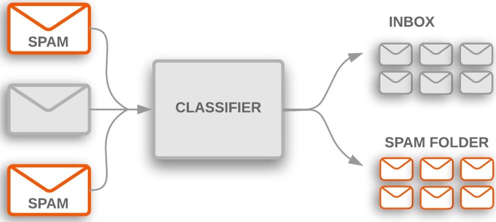


Complex, structured data

Image classification : bus $y = 1$ vs train $y = 0$

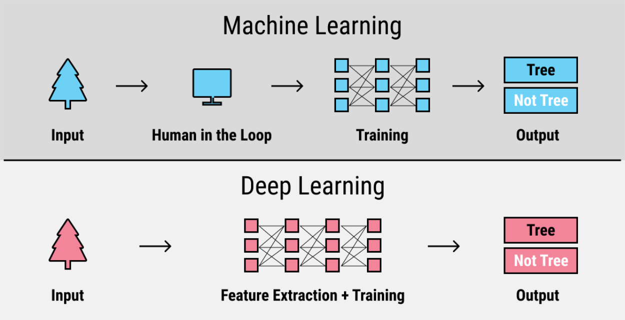

,

,


Text classification : spam vs non-spam



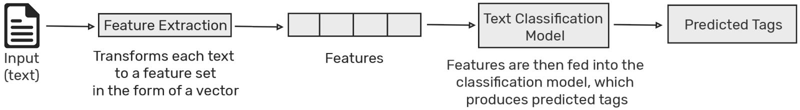
Machine Learning vs Deep Learning

Image

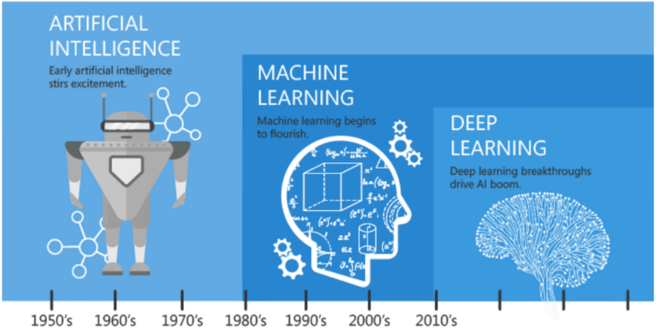


https://labelyourdata.com/articles/machine-learning-and-training-data

Text



Machine Learning vs Deep Learning



Today's IA is Deep Learning (a technique of Machine Learning)

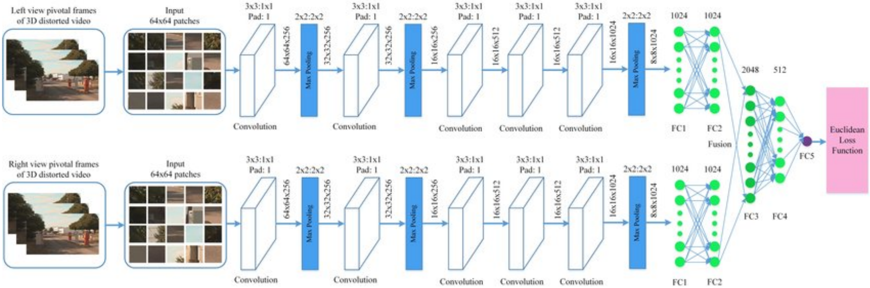
https://blog.alore.io/machine-learning-and-artificial-intelligence/

- Deep Learning: nowadays most popular paradigm in machine learning
- Its rise dates back to 2006 (ImageNet dataset challenge)
- Has introduced a paradigm shift in the way one will exploit data

Deep Learning paradigm

End-to-end learning

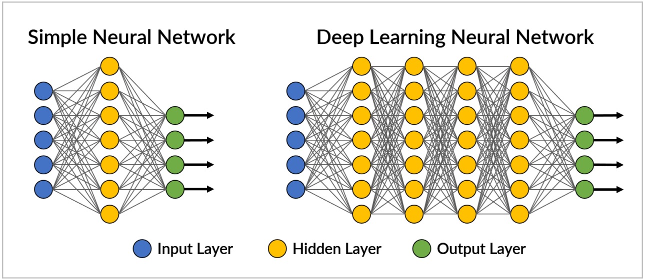
- Learn automatically (and simultaneously) the data representation $\Phi(x)$ and the decision function f



Deep learning paradigm

Principle

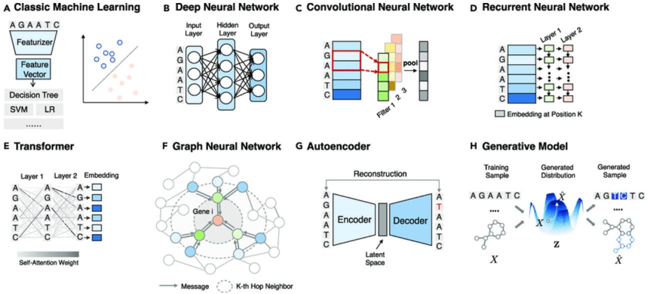
- Neural networks: models with many layers to learn hierarchical representations of data
- Composition of simple functions $f = f_1 \circ f_2 \circ \cdots \circ f_L$
- Learning based on data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^N$
- Scales with data and compute



https://www.go-rbcs.com/columns/deep-learning-to-the-rescue

Various deep learning models I

- Mulitlayer perceptron
- Convolutional networks
- Recurrent networks
- Transformers and Attention
- Diffusion models...

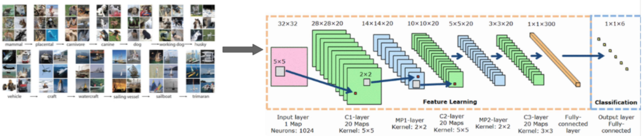


https://www.researchgate.net/figure/Ilustrations-of-machine-learning-models-Details-about-each-model-can-be-found-in_fig3_353783898

Various deep learning models II

Deep learning for computer vision tasks

Labeled training images — Deep classification architecture



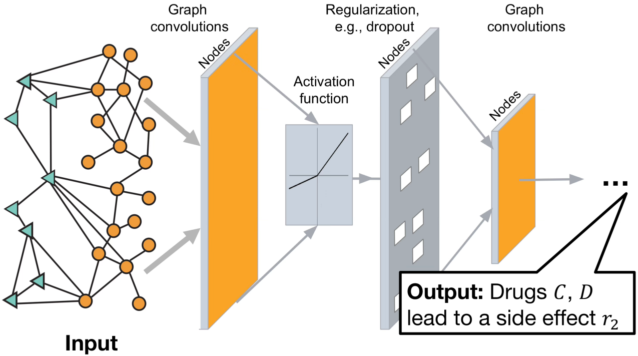
https://www.researchgate.net/profile/Y_Nikitin/publication/270163511/figure/download/fig5/AS:
295194831409153@1447391340221/MPCNN-architecture-using-alternating-convolutional-and-max-pooling-layers-13.png

Input images and predicted category

https://adriancolyer.files.wordpress.com/2016/04/imagenet-fig4l.png?w=656

Various deep learning models III

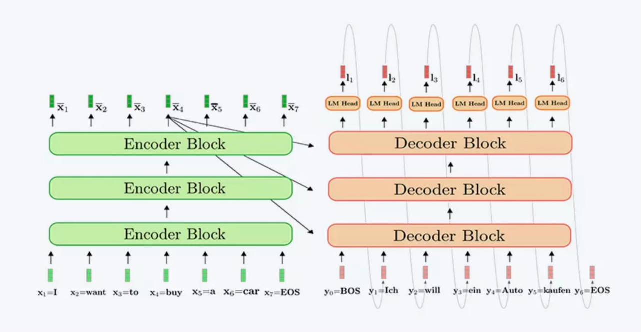
Deep learning for graphs



http://snap.stanford.edu/decagon/decagon-overview.png

Various deep learning models IV

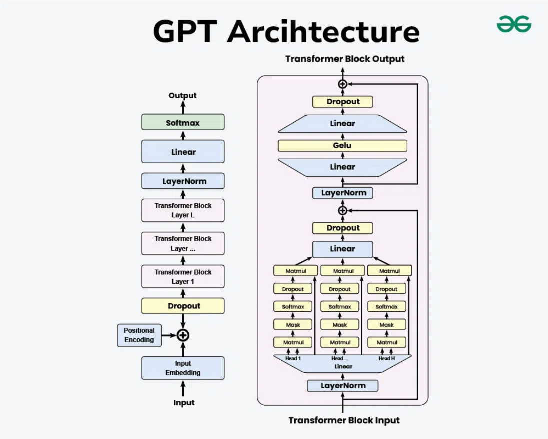
Deep learning for texts



https://raw.githubusercontent.com/patrickvonplaten/scientific_images/master/encoder_decoder/EncoderDecoder.png

Various deep learning models V

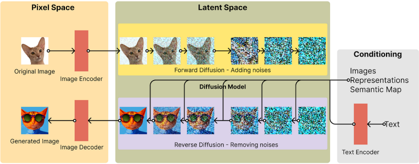
Deep learning for texts

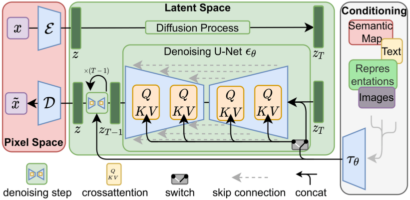


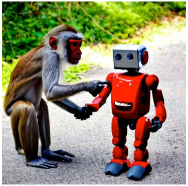
https://www.geeksforgeeks.org/artificial-intelligence/introduction-to-generative-pre-trained-transformer-gpt/

Various deep learning models VI

Deep generative models (image generation)



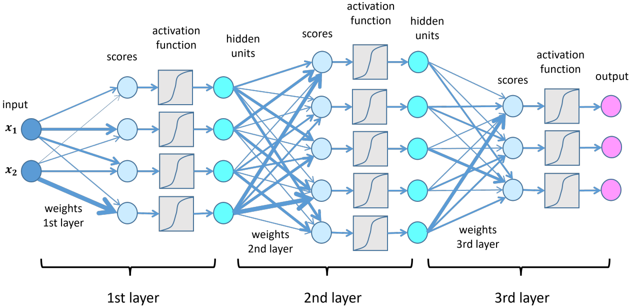




prompt: generate a monkey with red sunglasses shaking hands of a robot

Neural network

- Dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^N$
- Goal: train a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which prediction $f(\boldsymbol{x})$ approximates y as best as possible
- f is designed as a composition of functions, inspiring from human brains: $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) \circ f_2(\boldsymbol{x}) \circ \cdots \circ f_L(\boldsymbol{x})$
- Each function f_ℓ represents a layer



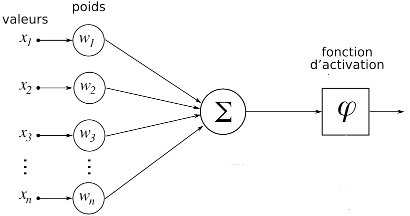
Artificial neuron [McCulloch et Pitts, 1943]

Formal neuron

- Input: $\boldsymbol{x} \in \mathbb{R}^d$, Output: y
- Input-output relationship

$$f(\boldsymbol{x}) = \varphi\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right)$$

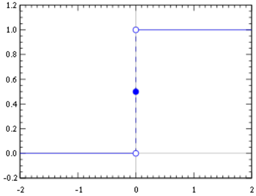
- φ: (non-linear) activation fonction



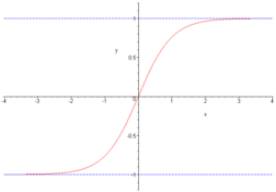
Activation functions

Examples

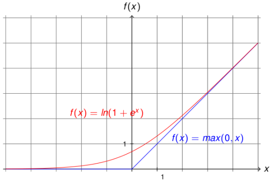
- Identity function: $\varphi(z) = z$
- Heaviside: $\varphi(z) = 0$ si $z < 0$, 1 sinon
- sigmoid: $\varphi(z) = \frac{1}{1+e^{-z}}$
- $\tanh$: $\varphi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}-1}{e^{2z}+1}$
- ReLU: $\varphi(z) = \max(0, z)$



heaviside



tanh



ReLU

The formal neuron as a learning machine (Perceptron 1958)

- Dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}\}_{i=1}^N$
- Decision function: $f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right)$

Algorithm 1 Perceptron Algorithm

Initialize $\boldsymbol{w}^0$, b^0, set the learning rate η, $t = 0$
repeat
  Draw randomly a sample $(\boldsymbol{x}_j, y_j)$
  if $y_j \left((\boldsymbol{w}^t)^\top \boldsymbol{x}_j + b^t\right) \leq 0$ **then**
    Update: $\begin{pmatrix} \boldsymbol{w}^{t+1} \\ b^{t+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}^t \\ b^t \end{pmatrix} + \eta \times y_j \times \begin{pmatrix} \boldsymbol{x}_j \\ 1 \end{pmatrix}, \qquad t = t + 1$
  end if
until convergence

- The learning rule is a **stochastic gradient algorithm** for minimizing the number of wrongly predicted labels
- Under some mild conditions, the algorithm is guaranteed to converge after a finite number of iterations (Novikof, 1962)

Perceptron algorithm and stochastic gradient descent (SGD)

- Objective function: $J(\boldsymbol{w}) = \sum_{i=1}^{N} -\mathbf{1}_{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq 0} \; y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = \sum_{i=1}^{N} \text{cost}(y_i, \boldsymbol{x}_i)$
- Goal: minimize the overall classification error
- Gradient $\nabla_{\boldsymbol{w}} J = -\sum_{i=1}^{N} \mathbf{1}_{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq 0} \; y_i \boldsymbol{x}_i$
- Learning scheme
  - Peform a gradient descent on a sample at a time
    - Pick a sample $(\boldsymbol{x}_j, y_j)$ at random
    - Update $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \nabla_{\boldsymbol{w}} \text{cost}(y_j, \boldsymbol{x}_j)$. b is updated similarly
  - Perceptron algorithm performs a **stochastic gradient descent (SGD)**

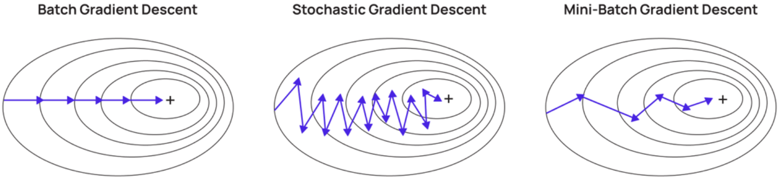
Adaline (Widrow - Hoff 1959)

Similar algorithm has been derived for least squares problem
$J(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \varphi\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right)\right)^2$

Batch vs Stochastic Gradient Descent

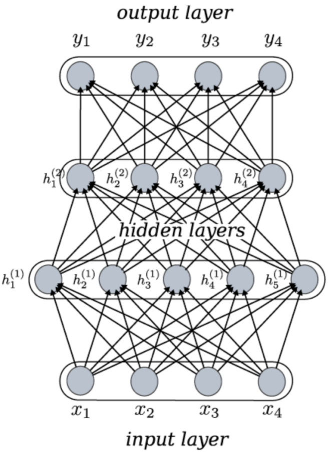
Global loss $J(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} \text{cost}(y_i, \boldsymbol{x}_i)$

Batch gradient	Stochastic gradient	Mini-batch gradient
$\mathbf{g} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{w}}\text{cost}(y_i, x_i)$	$\mathbf{g}_i = \frac{1}{N}\nabla_{\boldsymbol{w}}\text{cost}(y_i, x_i)$	$\mathbf{g}_{\mathcal{B}} = \frac{1}{N}\sum_{i\in\mathcal{B}} \nabla_{\boldsymbol{w}}\text{cost}(y_i, x_i)$



https://alwaysai.co/blog/what-is-gradient-descent

Multi-layer perceptron (MLP)



$$y = \varphi(W_3 h^{(2)})$$

$$h^{(2)} = \varphi(W_2 h^{(1)})$$

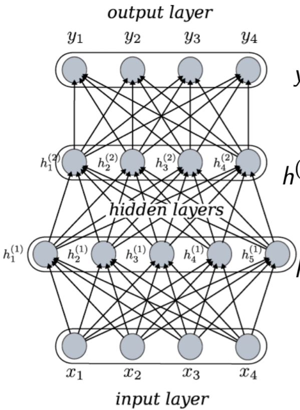
$$h^{(1)} = \varphi(W_1 \mathbf{x})$$

$$\mathbf{x}$$

Learning

Use backpropagation algorithm to compute the parameters (weights) $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3$

Forward and Backward Propagation



$$y = \varphi(W_3 h^{(2)})$$

$$\uparrow$$

$$h^{(2)} = \varphi(W_2 h^{(1)})$$

$$\uparrow$$

$$h^{(1)} = \varphi(W_1 \mathbf{x})$$

$$\uparrow$$

$$\mathbf{x}$$

$$\nabla_{W_3} J = (y - y_a)\varphi'(W_3 h^{(2)})h^{(2}$$

$$\downarrow$$

$$\nabla_{W_2} J = \nabla_{h^{(2)}} J \varphi'(W_2 h^{(1)})h^{(1)}$$

$$\downarrow$$

$$\nabla_{W_1} J =$$

Update of the parameters

$$\mathbf{W}_\ell^{t+1} \leftarrow \mathbf{W}_\ell^t - \eta \nabla_{\mathbf{W}_\ell^t} J(\mathbf{W}_\ell^t) \quad \text{pour} \quad \ell = 1, 2, \cdots, L$$

Training algorithm

Algorithm 2 Backpropagation algorithm

Initialize the weights $\mathbf{W}_\ell$, set L, η the learning rate, the mini-batch size B
while non convergence **do**
  for $t = 1 \rightarrow \text{round}(N/B)$ **do**
    Draw randomly a set of B samples $\mathcal{B}_t = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{B}$
    #Forward pass
    Compute the hidden vectors $\boldsymbol{h}^{(\ell)} = \varphi(\mathbf{W}^t \boldsymbol{h}^{(\ell-1)}), \quad \forall \ell = 1, \cdots, L$ and the loss $J(\mathbf{W}^t, \mathcal{B}_t)$ based on mini-batch $\mathcal{B}_t$
    #Backward pass
    Compute the gradients $\nabla_{\mathbf{W}_\ell^t} J(\mathbf{W}^t, \mathcal{B}_t) \quad \forall \ell$ based on mini-batch $\mathcal{B}_t$
    Update weights $\mathbf{W}_\ell^{t+1} \leftarrow \mathbf{W}_\ell^t - \eta \nabla_{\mathbf{W}_\ell} J(\mathbf{W}^t, \mathcal{B}_t) \quad \forall \ell = 1, \cdots, L$
  end for
end while

Learning rate

Several methods exist to set up the learning rate η: fixed, adaptive, momentum...

Optimization in deep learning: optimizer I

Optimizer Adagrad: SGD algorithms with Adaptive learning rate

AdaGrad adapts learning rates for each parameter w_k at each step t based on historical squared gradients. Let g_k^t the gradient of J w.r.t w_k, one computes:

$$G_k^t = G_k^{t-1} + (g_k^t)^2$$

with update rule:

$$w_k^{t+1} = w_k^t - \frac{\eta}{\sqrt{G_k^t + \epsilon}} g_k^t$$

Pros:

- Good for sparse data
- Per-parameter adaptive learning rate

Cons:

- Learning rate decays too aggressively.

Optimization in deep learning: optimizer II

Optimizer RMS Prop

RMSProp uses an exponential decaying average of past gradients:

$$E[g_k^2]_t = \rho E[g_k^2]_{t-1} + (1-\rho)(g_k^t)^2$$

Update rule:

$$w_k^{t+1} = w_k^t - \frac{\eta}{\sqrt{E[g_k^2]_t + \epsilon}} g_k^t$$

Characteristics:

- Controls the unbounded growth of AdaGrad's accumulator.
- Works well for deep learning tasks where gradients vary over time.

Optimization in deep learning: optimizer III

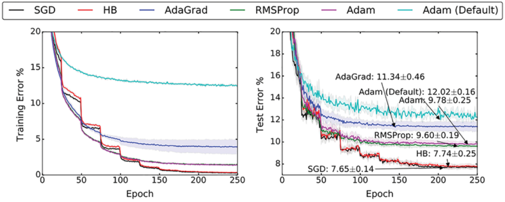
Optimizer Adam

Adam combines Momentum and RMSProp:

$$m_k^t = \beta_1 m_k^{t-1} + (1-\beta_1) g_k^t$$

$$v_k^t = \beta_2 v_k^{t-1} + (1-\beta_2) (g_k^t)^2$$

Bias-corrected estimates: $\hat{m}_k^t = \frac{m_k^t}{1-\beta_1}$, $\hat{v}_k^t = \frac{v_k^t}{1-\beta_2}$, Update: $w_k^{t+1} = w_k^t - \eta \frac{\hat{m}_k^t}{\sqrt{\hat{v}_k^t} + \epsilon}$



(a) CIFAR-10 (Train) (b) CIFAR-10 (Test)

https://www.researchgate.net/publication/368951831/figure/fig71/AS:
11431281233495765@1712057231036/Standard-SGD-and-SGD-with-momentum-vs-AdaGrad-RMSProp-Adam-on-CIFAR-10-dataset.tif

Optimization in deep learning: regularization

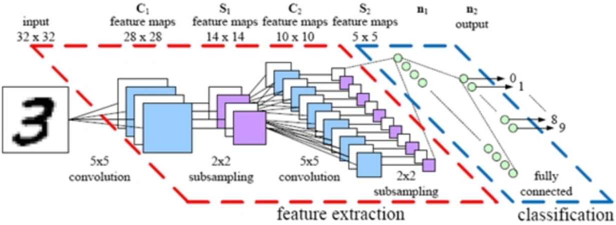
Regularization schemes

- Explicit regularization: rather minimize $J(\mathbf{W}) + \lambda\|\mathbf{W}\|^2$ with $\lambda > 0$
- Dropout: Randomly drop some weights at training time
  - Parameter: dropout percentage p (percentage of parameters to deactivate)
  - Each weigth is dropped with probability p (Bernouilli) i.e. is inactive in the forward and backward pass
- Batch Normalization

$$\hat{\boldsymbol{x}} = (\boldsymbol{x} - \boldsymbol{\mu}_{\mathcal{B}}) \oslash (\sqrt{\boldsymbol{\sigma}_{\mathcal{B}}^2 + \epsilon})$$

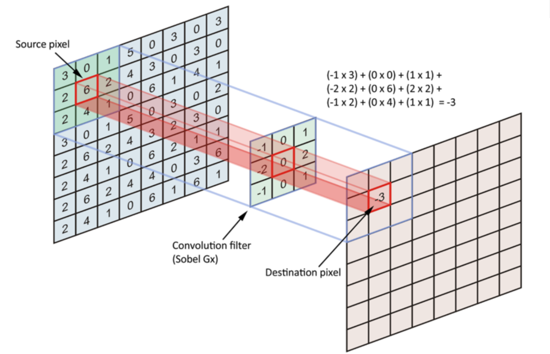
$\oslash$ elementwise division

What if the inputs are images?



Use Convolutional Neural Network (CNN)

More on CNN I



Convolution: Given input I and filter K,

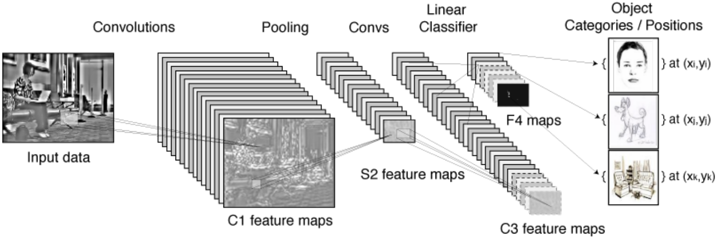
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Followed by pooling (max pooling): $\text{pool}(i,j) = \max_{(m,n)\in\text{window}} I(i+m, j+n)$

More on CNN II

Stacking several layers

- Each layer includes Convolution and Pooling to summarize spatial information
- Last layers are fully connected layers (MLP-like) to yield the output



animation : `http://cs231n.github.io/convolutional-networks/`

Recurrent networks

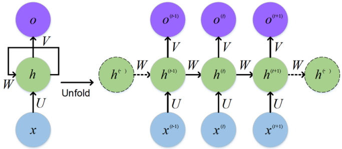
Many tasks involve sequential or temporal structure:

- Natural language: sentences, documents
- Time-series prediction: stock prices, weather
- Audio processing: speech, music
- Video: sequence of frames

RNNs introduce **recurrence**: each output depends on the current input *and* previous hidden state.

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

This allows information to persist across time.



https://www.researchgate.net/publication/318332317/figure/fig1/AS:614309562437664@1523474221928/The-standard-RNN-and-unfolded-RNN.png

Packages

- TensorFlow (Google, python) `https://www.tensorflow.org`
- Keras (Google, python) `https://keras.io/` (+ Theano ou TF)
- Pytorch