Traitement des données massives et apprentissage
École d'été, 5-9 juin 2023

Introduction to deep learning and neural networks

Emanuele Dalsasso (CÉDRIC, CNAM, Paris)

le cnam

June, 2023

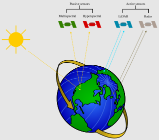
Context leading to deep learning rise

Big Data

- Explosion of data availability: images, videos, audio, text, *etc*
- Sources: Media, Social Networks, Open data policies for commercial and/or scientific purposes

Digital data

Biomedical instruments



Remote Sensing sensors

- Need to access, search, analysis, visualise, classify these data
- Huge number of applications: medicine, economy, science *etc*
- Leading track in major ML/CV conferences during the last decade

- Visual Recognition: archetype of low-level signal understanding
- Supposed to be a master class problem in the early 80's
- Certainly the most impacted topic by deep learning
  - Scene categorization
  - Object localization
  - Context & Attribute recognition
  - Rough 3D layout, depth ordering
  - Rich description of scene, *e.g.* sentences

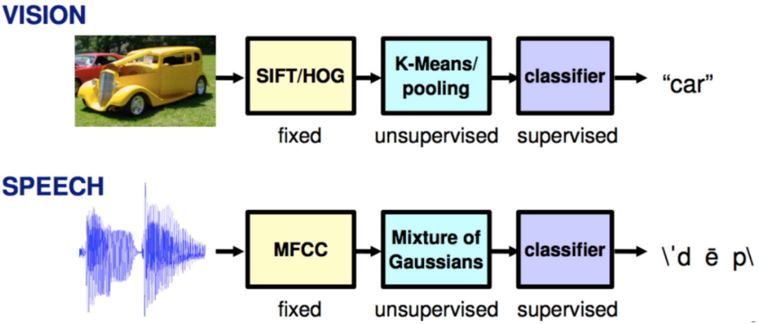
Example of Semantic Segmentation in Earth Observation†

Example of Visual Question Answering in Remote Sensing‡

† *Illustration from* ***Semi-Supervised Semantic Segmentation in Earth Observation: The MiniFrance suite, dataset analysis and multi-task network study****, Castillo-Navarro, Javiera, et al., Machine Learning 2021*
‡ *Illustration from* ***RSVQA: Visual question answering for remote sensing data****, Lobry, Sylvain, et al., IEEE TGRS 2020*

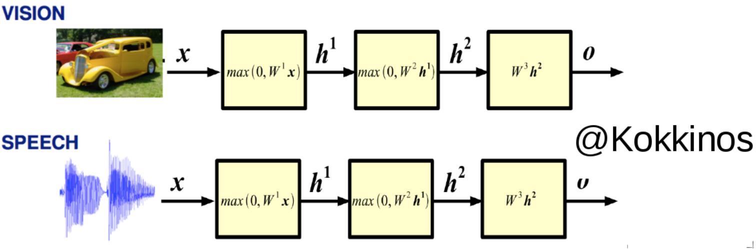
Deep Learning (DL) & Recognition of low-level signals

- DL: breakthrough for the recognition of low-level signal data
- Before DL: handcrafted intermediate representations for each task
  - ⊖ Needs expertise (PhD level) in each field
  - ⊖ Weak level of semantics in the representation

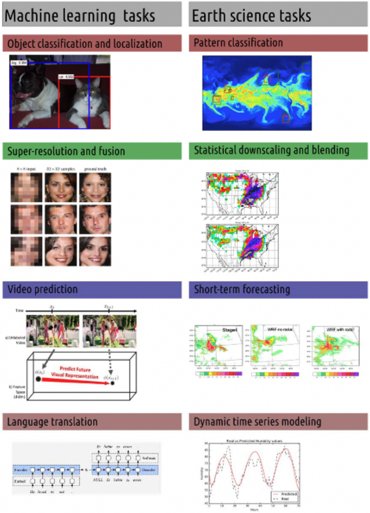


@Kokkinos

- DL: breakthrough for the recognition of low-level signal data
- Since DL: automatically **learning intermediate representations**
  - $\oplus$ Outstanding experimental performances $>>$ handcrafted features
  - $\oplus$ Able to learn high level intermediate representations
  - $\oplus$ Common learning methodology $\Rightarrow$ field independent, no expertise



From Machine Learning to Earth Sciences



† *Illustration from* ***Deep learning and process understanding for data-driven Earth system science****, Reichstein, Markus, et al., Nature 2019*

Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

The origins

History

- **1943:** The formal neuron [McCulloch and Pitts, 1943]
- **1958:** First perceptron [Rosenblatt, 1958]
- **1974:** Backpropagation algorithm [Werbos, 1974]
- **1980:** First deep feedforward network [Fukushima, 1980]
- **1989:** First convolutional neural network [LeCun et al., 1989]

The formal neuron, basis of deep feedforward neural networks

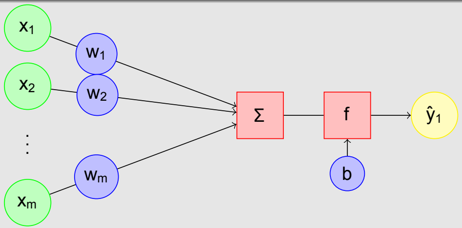


Figure: The formal neuron – Credits: R. Herault

x_i: inputs
w_i, b: weights and biases
f: activation function
y: output of the neuron

$$y = f(w^\top x + b)$$

Outline

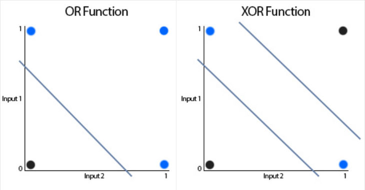
1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

Example: Learning XOR

The XOR function

$$y = f^\star(\mathbf{x}) = x_1 \oplus x_2$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

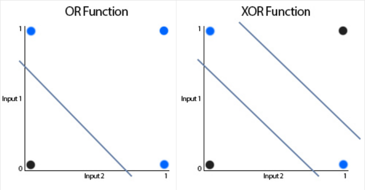


Example: Learning XOR

The XOR function

$$y = f^\star(\mathbf{x}) = x_1 \oplus x_2$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Single perceptron

- Define MSE loss function to learn parameters $\boldsymbol{\theta}$ of $f^{\boldsymbol{\theta}}(\mathbf{x})$:

$$\mathcal{L} = \frac{1}{4}\sum_{i=1}^{4}(f^\star(\mathbf{x}_i) - f^{\boldsymbol{\theta}}(\mathbf{x}_i))^2$$

- Linear model defined by

$$f^{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

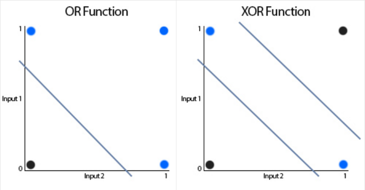
- The solution gives $\mathbf{w} = \mathbf{0}$ and $b = \frac{1}{2}$: $f^{\mathbf{w},b}(\mathbf{x}) = \frac{1}{2}$
- Data are not linearly separable

Example: Learning XOR

The XOR function

$$y = f^\star(\mathbf{x}) = x_1 \oplus x_2$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Multi-layer perceptron

- Define MSE loss function to learn parameters $\boldsymbol{\theta}$ of $f^{\boldsymbol{\theta}}(\mathbf{x})$:

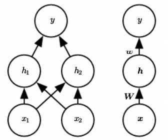
$$\mathcal{L} = \frac{1}{4}\sum_{i=1}^{4}(f^\star(\mathbf{x}_i) - f^{\boldsymbol{\theta}}(\mathbf{x}_i))^2$$

- Define different feature space where the linear model is able to represent the solution
- Nonlinear model defined by

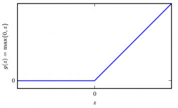
$$f(\mathbf{x})^{(\mathbf{W},\mathbf{b},\mathbf{w},b)} = \mathbf{w}^T \max\{0, \mathbf{W}^T\mathbf{x}+\mathbf{b}\}+b$$

- The solution gives $\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ and $b = 0$, which perfectly fits with data

Learning XOR



Two ways of representing a perceptron with 1 hidden layer



The rectified linear unit **ReLU** activation function

- Single perceptron can only represent linear functions (linear regression, logistic regression):

$$f(\mathbf{x})^{(\mathbf{w},b)} = \mathbf{x}^T\mathbf{w} + b$$

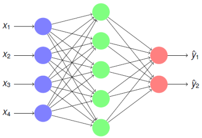
- Represent nonlinear functions by apply linear model to a nonlinear transformation of the input $\mathbf{x}$:

$$f(\mathbf{x})^{(\mathbf{W},\mathbf{b},\mathbf{w},b)} = \mathbf{w}^T \max\{0, \mathbf{W}^T\mathbf{x} + \mathbf{b}\} + b$$

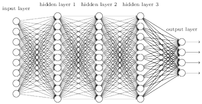
$\Rightarrow$ learn XOR by breaking it down to OR, NAND and AND

*Illustrations from **Deep learning**, Goodfellow, Ian, Yoshua Bengio, and Aaron Courville, MIT press, 2016.*

Deep Feedforward Networks



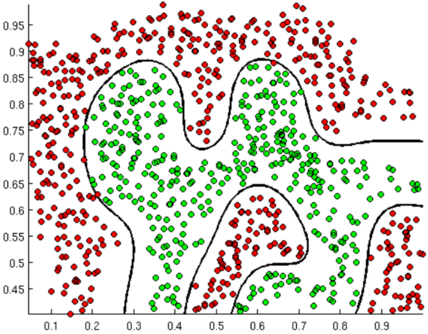
Perceptron with 1 hidden layer – Credits: R. Herault



Stacking more layers, toward "deep learning" – Credits: M. Nielsen

- **Feedforward**: flow of information from $\mathbf{x}$ to $\mathbf{y}$
- **Neural**: ispired by neuroscience. The elementary element is called *neuron*
- **Network**: stack of several different functions (layers). For instance:
  - $\mathbf{y} = f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$
  - The length of the chain gives the **depth** (*deep learning*): *different levels of abstraction* from low-level features to the high-level ones
  - $f^{(3)}$ is the **output layer**
  - $f^{(1)}$ and $f^{(2)}$ are **hidden layers**
- Activation functions (nonlinearities): ReLU, Sigmoid, Tanh, Softmax, ..
  $\Rightarrow$ Build universal function approximators from simple components

- Basis of the "deep learning" field
- Principle: Stacking layers of neural networks to allow more complex and rich functions
- Can be seen as **different levels of abstraction** from low-level features to the high-level ones
- **Neural network with one single hidden layer** $\Rightarrow$ **universal approximator [Cybenko, 1989]**
  - Ex for classification: any decision boundaries can be expressed



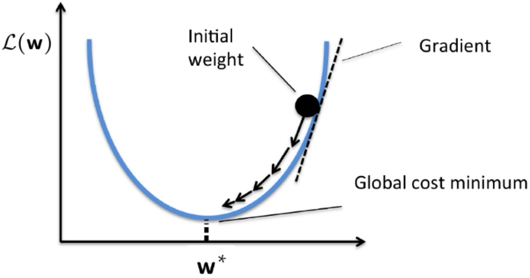
1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

Training Multi-Layer Perceptron (MLP)

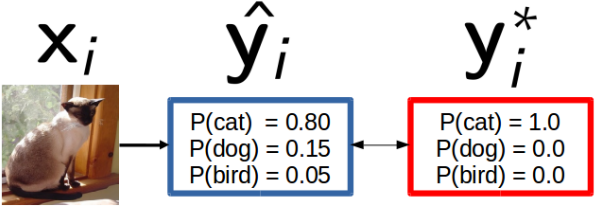
- Input $\mathbf{x}$, output $\mathbf{y}$
- A parametrized model $\mathbf{x} \Rightarrow \mathbf{y}$: $f_{\mathbf{w}}(\mathbf{x}_i) = \hat{\mathbf{y}}_i$
- Supervised context: training set $\mathcal{A} = \left\{(\mathbf{x}_i, \mathbf{y}_i^*)\right\}_{i \in \{1,2,\dots,N\}}$
  - A loss function $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$ for each annotated pair $(\mathbf{x}_i, \mathbf{y}_i^*)$
- Assumptions: parameters $\mathbf{w} \in \mathbb{R}^d$ continuous, $\mathcal{L}$ differentiable
- Gradient $\nabla_{\mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$: steepest direction to decrease loss $\mathcal{L}$

MLP Training

- Gradient descent algorithm:
  - Initialyze parameters $\mathbf{w}$
  - Update: $\boxed{\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \boldsymbol{\eta} \frac{\partial \mathcal{L}}{\partial \mathbf{w}}}$
  - Until convergence, *e.g.* $||\nabla_{\mathbf{w}}||^2 \approx 0$



MLP Training: loss function

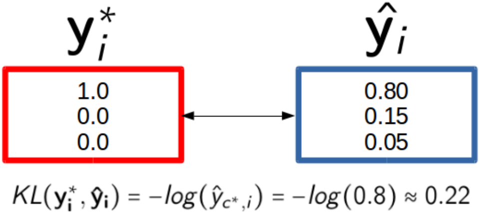


- **Input $\mathbf{x}_i$, ground truth output supervision $\mathbf{y}_i^*$**
- One hot-encoding for $\mathbf{y}_i^*$:

$$y_{c,i}^* = \begin{cases} 1 & \text{if c is the groud truth class for } \mathbf{x}_i \\ 0 & \text{otherwise} \end{cases}$$

- Loss function: multi-class Cross-Entropy (CE) ℓ_{CE}
- ℓ_{CE}: Kullback-Leiber divergence between $\mathbf{y_i^*}$ and $\mathbf{\hat{y}_i}$

$$\ell_{CE}(\mathbf{\hat{y}_i}, \mathbf{y_i^*}) = KL(\mathbf{y_i^*}, \mathbf{\hat{y}_i}) = -\sum_{c=1}^{K} y_{c,i}^* log(\hat{y}_{c,i}) = -log(\hat{y}_{c^*,i})$$

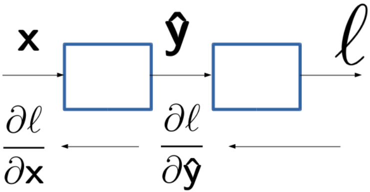


MLP Training: Backpropagation

- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
- ℓ_{CE} smooth convex upper bound of $\ell_{0/1}$
  $\Rightarrow$ **gradient descent optimization**
- Gradient descent: $\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \boldsymbol{\eta}\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$ $\quad(\mathbf{b}^{(t+1)} = \mathbf{b}^{(t)} - \boldsymbol{\eta}\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{b}})$
- Computing $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}}$?

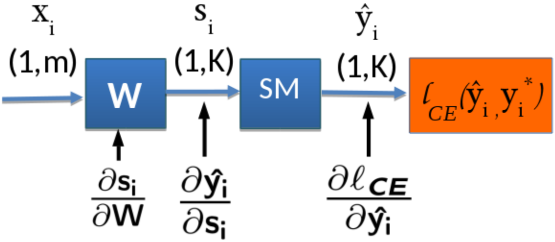
  $\Rightarrow$ **Backpropagation of gradient error!**

  $\Rightarrow$ **<u>Key Property:</u>** chain rule $\dfrac{\partial \mathbf{x}}{\partial z} = \dfrac{\partial \mathbf{x}}{\partial \mathbf{y}}\dfrac{\partial \mathbf{y}}{\partial z}$



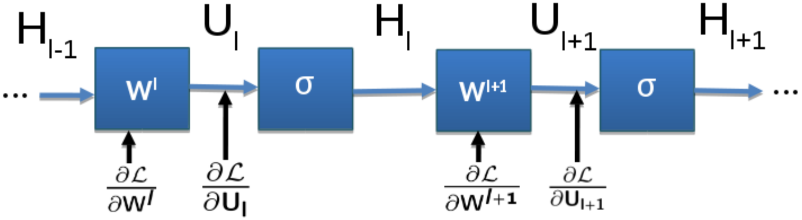
$$\boxed{\frac{\partial \ell}{\partial \mathbf{x}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}}}$$

- Logistic regression: $\boxed{\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_\mathbf{i}} \frac{\partial \hat{\mathbf{y}}_\mathbf{i}}{\partial \mathbf{s}_\mathbf{i}} \frac{\partial \mathbf{s}_\mathbf{i}}{\partial \mathbf{W}}}$



Deep Neural Network Training: Backpropagation

- Multi-Layer Perceptron (MLP): adding more hidden layers
- Backpropagation update $\sim$ application of chain rule recursively through all network layers



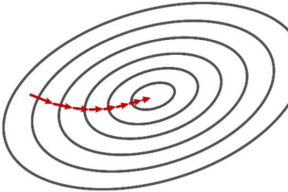
Neural Network Training: Optimization Issues

- Classification loss over training set ($\mathbf{w}$):

$$\mathcal{L}_{CE}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{c^*,i})$$

- Gradient descent optimization:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \boldsymbol{\eta}\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{w}}\left(\mathbf{w}^{(t)}\right) = \mathbf{w}^{(t)} - \boldsymbol{\eta}\nabla_{\mathbf{w}}^{(t)}$$



- Gradient $\nabla_{\mathbf{w}}^{(t)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}}\left(\mathbf{w}^{(t)}\right)$ linearly scales wrt:
  - $\mathbf{w}$ dimension
  - Training set size

$\Rightarrow$ **Too slow even for moderate dimensionality & dataset size!**

Stochastic Gradient Descent

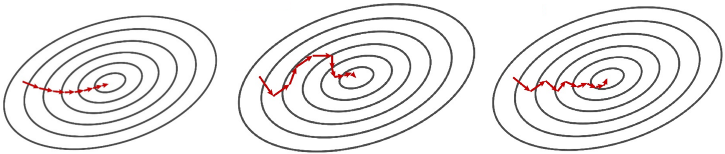
- **Solution:** approximate $\nabla_{\mathbf{w}}^{(t)} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}}\left(\mathbf{w}^{(t)}\right)$ with subset

  $\Rightarrow$ **Stochastic Gradient Descent (SGD)**
  - Use a single example (online):

$$\nabla_{\mathbf{w}}^{(t)} \approx \frac{\partial \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}}\left(\mathbf{w}^{(t)}\right)$$

  - Mini-batch: use $B < N$ examples (avoids redundancy):

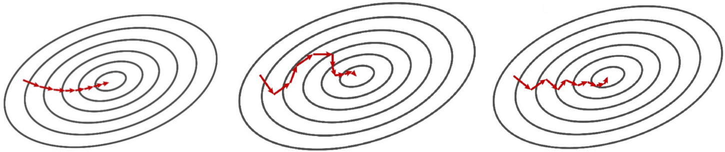
$$\nabla_{\mathbf{w}}^{(t)} \approx \frac{1}{B}\sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}}\left(\mathbf{w}^{(t)}\right)$$



Full gradient SGD (online) SGD (mini-batch)

Stochastic Gradient Descent

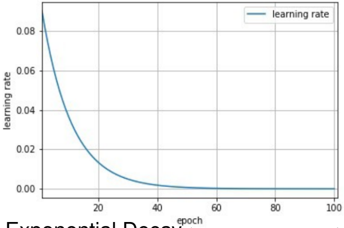
- **SGD: approximation of the true Gradient $\nabla_{\mathbf{w}}$!**
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT:** much more parameter updates: online $\times N$, mini-batch $\times \frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets



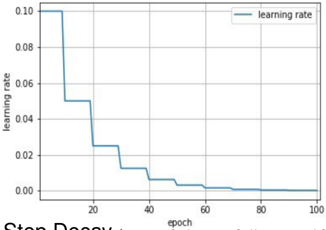
Full gradient | SGD (online) | SGD (mini-batch)

Optimization: Learning Rate Decay

- Gradient descent optimization: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \boldsymbol{\eta}\nabla_{\mathbf{w}}^{(t)}$
- $\boldsymbol{\eta}$ setup ? $\Rightarrow$ open question
- Learning Rate Decay: decrease $\boldsymbol{\eta}$ during training progress
  - Inverse (time-based) decay: $\boldsymbol{\eta}_t = \frac{\boldsymbol{\eta}_0}{1+r\cdot t}$, r decay rate
  - Exponential decay: $\boldsymbol{\eta}_t = \boldsymbol{\eta}_0 \cdot e^{-\lambda t}$
  - Step Decay $\boldsymbol{\eta}_t = \boldsymbol{\eta}_0 \cdot r^{\frac{t}{t_u}}$ ...

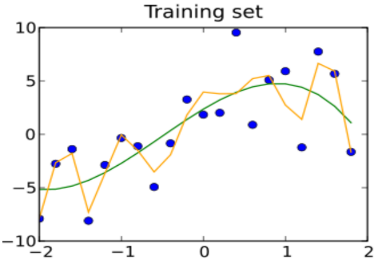


Exponential Decay $(\boldsymbol{\eta}_0 = 0.1, \lambda = 0.1\text{s})$

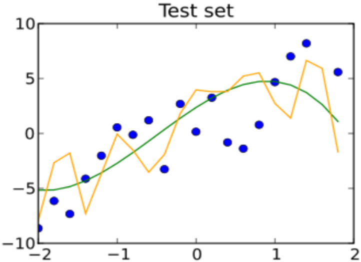


Step Decay $(\boldsymbol{\eta}_0 = 0.1, r = 0.5, t_u = 10)$

- **Learning:** minimizing classification loss $\mathcal{L}_{CE}$ over training set
  - Training set: sample representing data *vs* labels distributions
  - **Ultimate goal:** train a prediction function with low prediction error on the **true (unknown) data distribution**



$\mathcal{L}_{train} = 4$, $\mathcal{L}_{train} = 9$



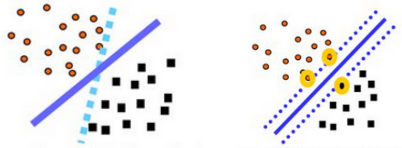
$\mathcal{L}_{test} = 15$, $\mathcal{L}_{test} = 13$

$\Rightarrow$ **Optimization $\neq$ Machine Learning!**
$\Rightarrow$ **Generalization / Overfitting**!

- **Regularization:** improving generalization, *i.e.* test ($\neq train$) performances
- Structural regularization: add ***Prior*** $R(\mathbf{w})$ in training objective:

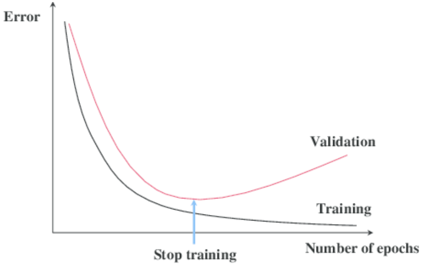
$$\mathcal{L}(\mathbf{w}) = \mathcal{L}_{CE}(\mathbf{w}) + \alpha R(\mathbf{w})$$

- L^2 **regularization: weight decay**, $R(\mathbf{w}) = ||\mathbf{w}||^2$
  - Commonly used in neural networks
  - Theoretical justifications, generalization bounds (SVM)
- Other possible $R(\mathbf{w})$: L^1 regularization, dropout, *etc*



Regularization and hyper-parameters

- **Neural networks:** hyper-parameters to tune:
  - **Training parameters:** learning rate, weight decay, learning rate decay, # epochs, *etc*
  - **Architectural parameters:** number of layers, number neurones, non-linearity type, *etc*
- **Hyper-parameters tuning:** $\Rightarrow$ improve generalization: estimate performances on a validation set



References I

[Cybenko, 1989] Cybenko, G. (1989).
Approximation by superpositions of a sigmoidal function.
Mathematics of control, signals and systems, 2(4):303–314.

[Fukushima, 1980] Fukushima, K. (1980).
Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.
Biological cybernetics, 36(4):193–202.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep Learning.
MIT Press.
http://www.deeplearningbook.org.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989).
Backpropagation applied to handwritten zip code recognition.
Neural computation, 1(4):541–551.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943).
A logical calculus of the ideas immanent in nervous activity.
The bulletin of mathematical biophysics, 5(4):115–133.

[Rosenblatt, 1958] Rosenblatt, F. (1958).
The perceptron: A probabilistic model for information storage and organization in the brain.
Psychological Review, 65(6):386–408.

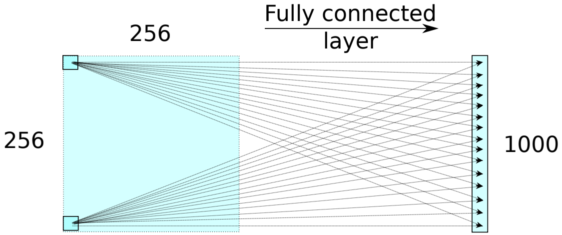
[Werbos, 1974] Werbos, P. (1974).
Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.
PhD thesis, Harvard University.

Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

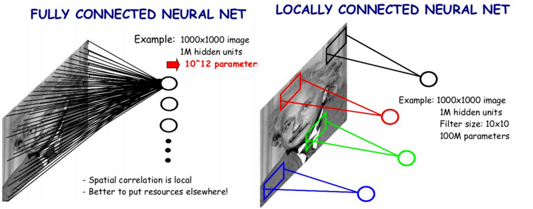
Introduction to CNN

- In MLPs each layer of the network contained **fully connected** layers
- Unfortunately, there are great drawbacks with such an approach



- Each hidden unit is connected to each input unit
- There is high redundancy in these weights :
  - In the above example, 65 million weights are required

- For many types of data with **grid-like topological structures** (eg. images), it is not necessary to have so many weights
- For these data, the **convolution** operation is often extremely useful
- Reduces the number of parameters to train
  - Training is faster
  - Convergence is easier : smaller parameter space



- A neural network with convolution operations is known as a **Convolutional Neural Network** (CNN)

- "Neocognitron" of Fukushima* : first to incorporate notion of receptive field into a neural network, based on work on animal perception of Hubert and Weisel†
- Yann LeCun first to propose **back-propagation** for training convolutional neural networks‡
  - Automatic learning of parameters instead of hand-crafted weights
  - However, training was very long : required 3 days (in 1990)
- In the years 1998-2012, research continued on shallow and deep neural networks, but other machine learning approaches were preferred (GMMs, SVMs etc.)
- In 2012, Alex Krizhevsky et al. used **Graphics Processing Units** (GPUs) to carry out backpropagation on a very deep convolutional neural network
  - Greatly outperformed classic approaches in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

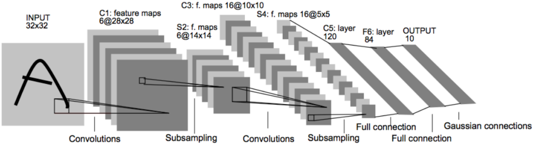


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

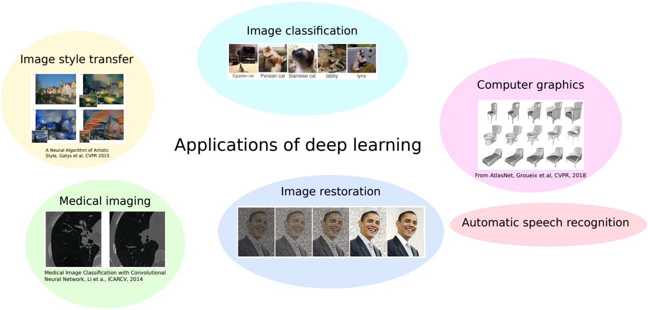
* ***Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position***, *Fukushima, K., Biological Cybernetics, 1980*

† ***Receptive fields and functional architecture of monkey striate cortex***, *Hubel, D. H. and Wiesel, T. N, 1968*

‡ ***Backpropagation Applied to Handwritten Zip Code Recognition***, *LeCun, Y. et al., AT&T Bell Laboratories*

Introduction - some history

- Since 2012, CNNs have completely revolutionised many domains
- CNNs produce competetive/best results for most problems in image processing and computer vision



- Being applied to an ever-increasing number of problems

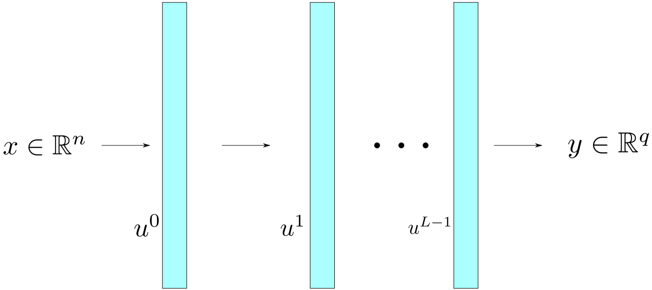
Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

Introduction - some notation

Notations

- $x \in \mathbb{R}^n$: input vector
- $y \in \mathbb{R}^q$: output vector
- u_ℓ : feature vector at layer ℓ
- $\boldsymbol{\theta}_\ell$: network parameters at layer ℓ



Neural network with L layers

- A "**Convolutional Neural Network**" (CNN) is simply a concatenation of :
  1. Convolutions (filters)
  2. Additive biases
  3. Down-sampling ("Max-Pooling" etc.)
  4. Non-linearities

Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

Convolution operator

Let f and g be two integrable functions. The **convolution operator** $*$ takes as its input two such functions, and outputs another function $h = f * g$, which is defined at any point $t \in \mathbb{R}$ as :

$$h(t) = (f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau.$$

- Intuitively, the function h is defined as the inner product between f and a *shifted* version of g

Convolutional Layers

- In many practical applications, in particular for CNNs, we use the **discrete convolution** operator, which acts on discretised functions;

Discrete convolution operator

Let f_n and g_n be two summable series, with $n \in \mathbb{Z}$. The discrete convolution operator is defined as :

$$(f * g)(n) = \sum_{i=-\infty}^{+\infty} f(i)g(n-i)$$

- Intuitively, the function h is defined as the inner product between f and a *shifted* version of g
- In practice, the filter is of small spatial support, around 3×3, or 5×5
- Therefore, only a **small number** of parameters need to be trained (9 or 25 for these filters)

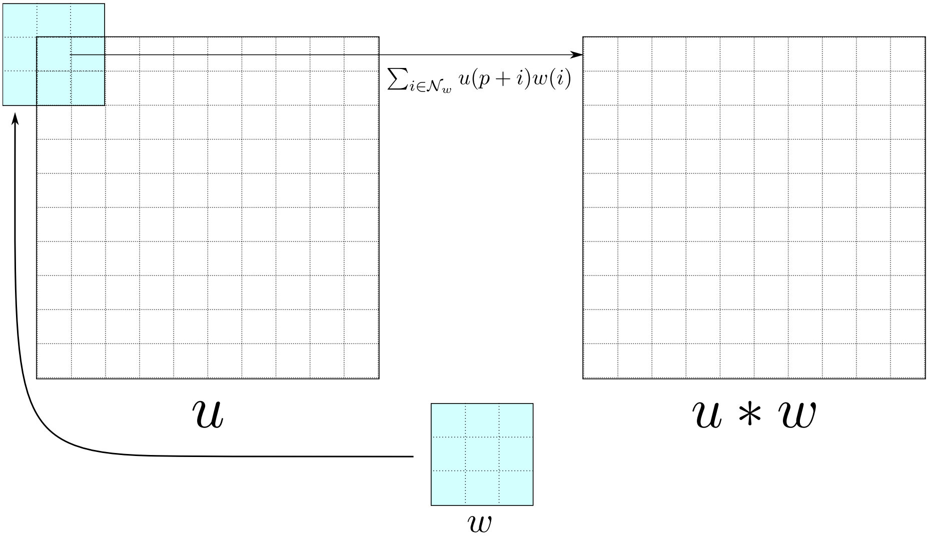
- Most often, we are going to be working with **images**
- Therefore, we require a 2D convolution operator : this is defined in a very similar manner to 1D convolution :

2D convolution operator

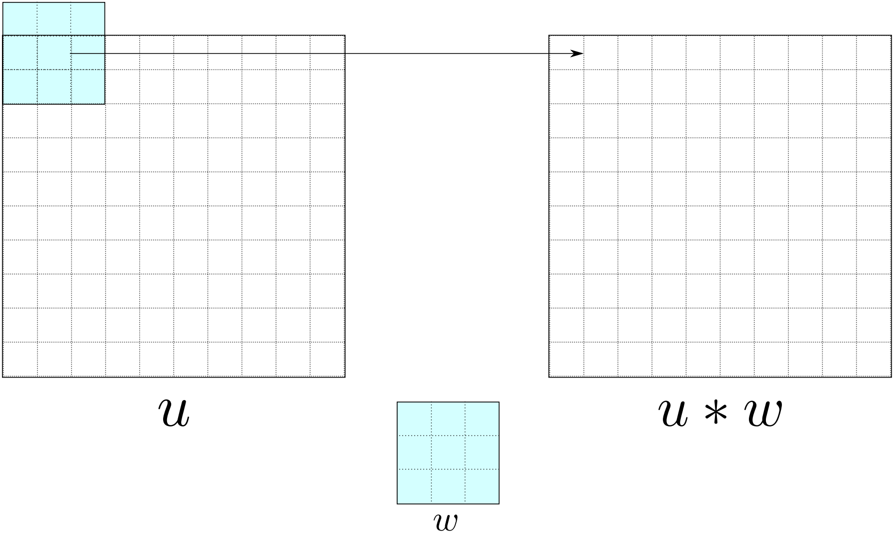
$$(f * g)(s,t) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f(i,j)g(s-i,t-j)$$

- We are going to denote the **filters with** w
- For lighter notation, we write $w(i) =: w_i$ (and the same for x_i etc.)

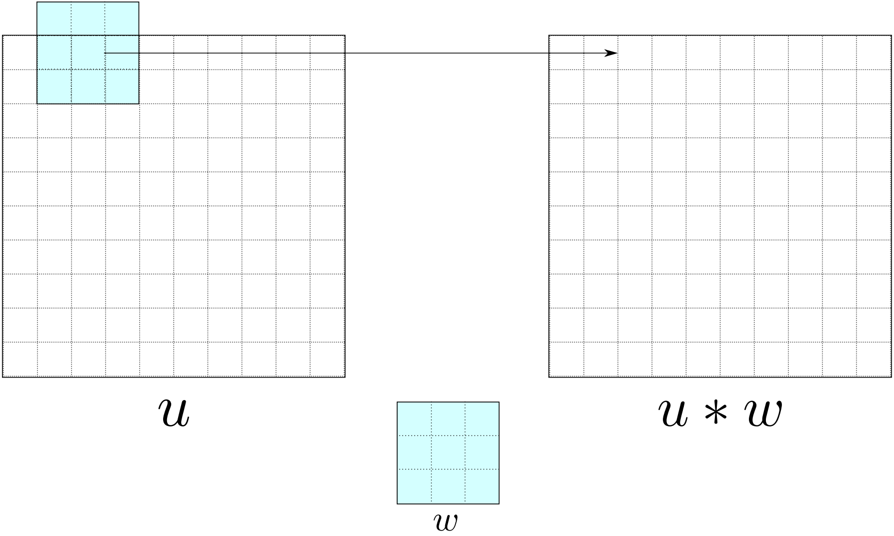
Convolutional Layers : Visual Illustration



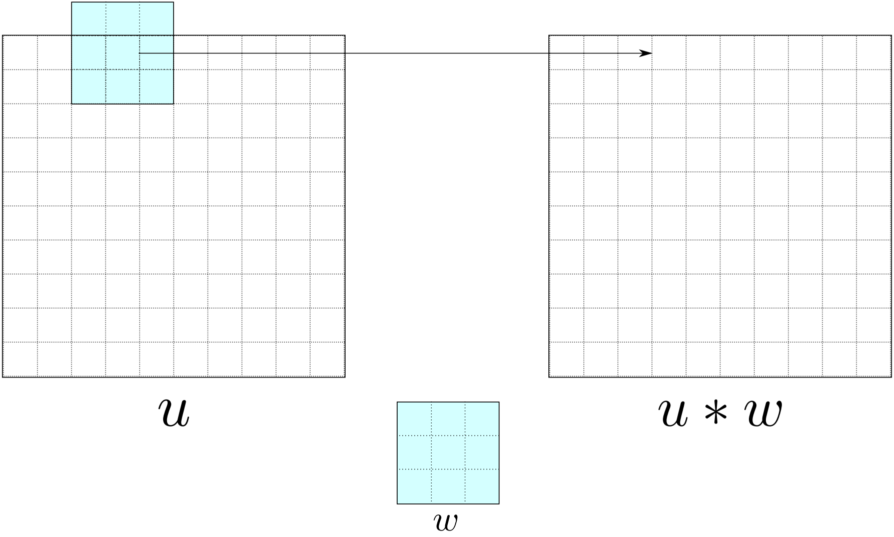
Convolutional Layers : Visual Illustration



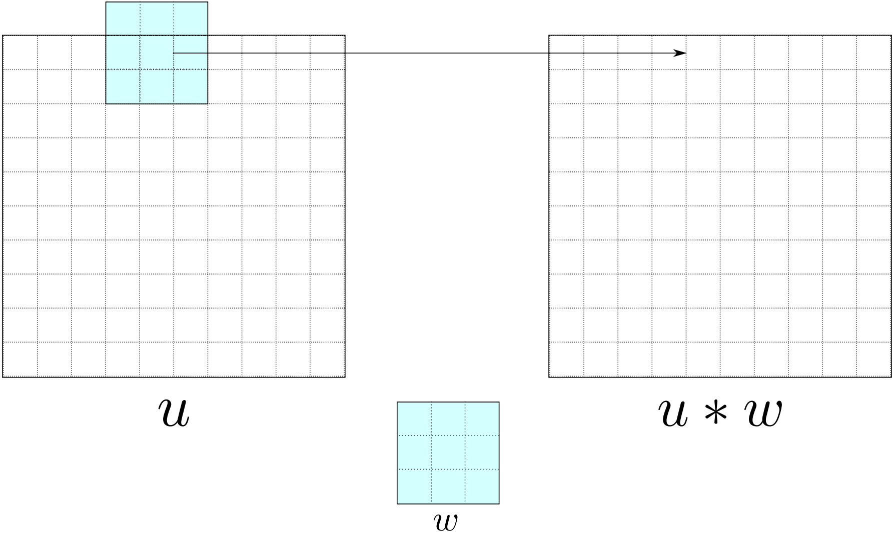
Convolutional Layers : Visual Illustration



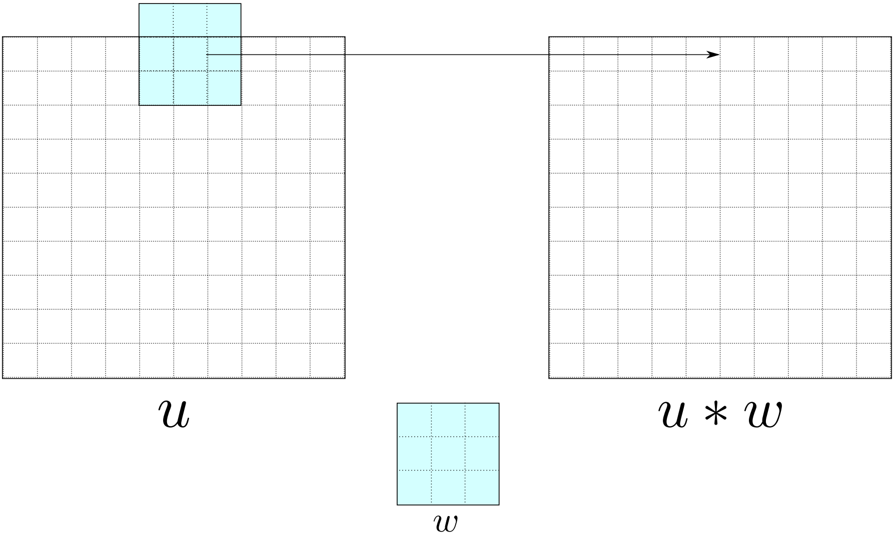
Convolutional Layers : Visual Illustration



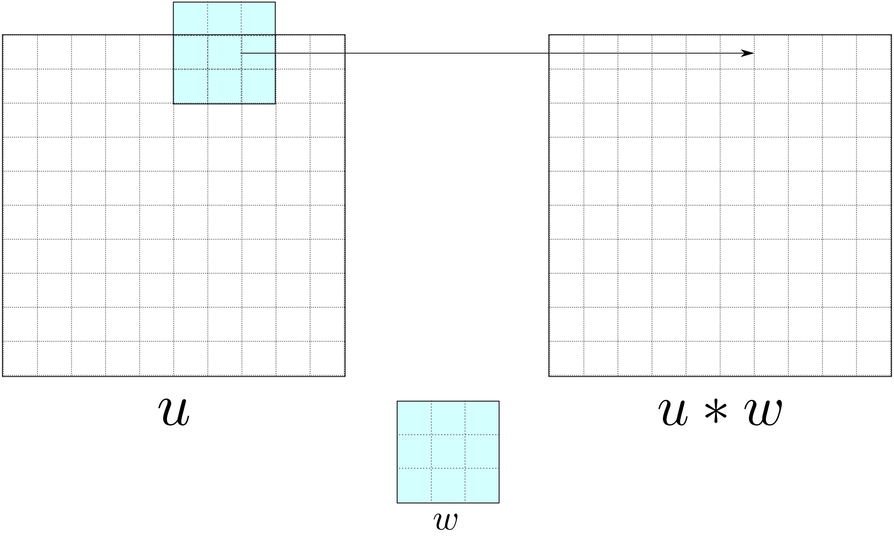
Convolutional Layers : Visual Illustration



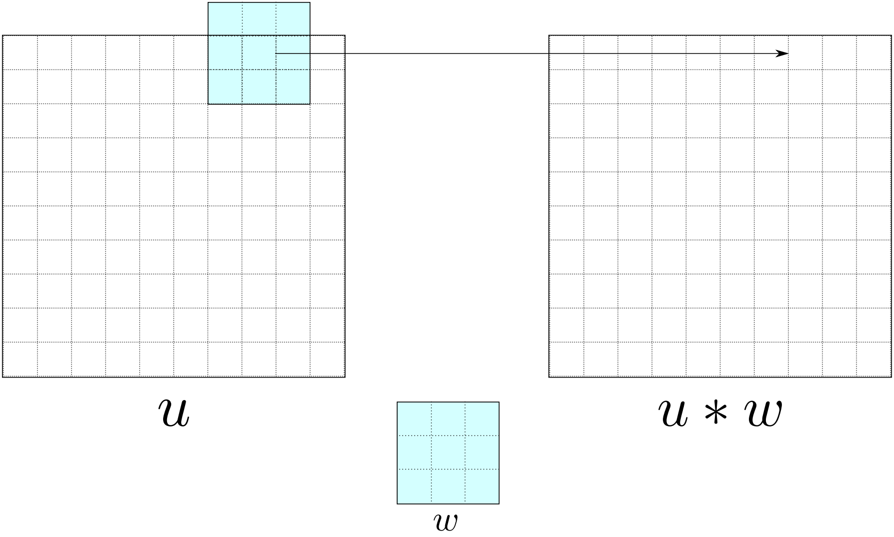
Convolutional Layers : Visual Illustration



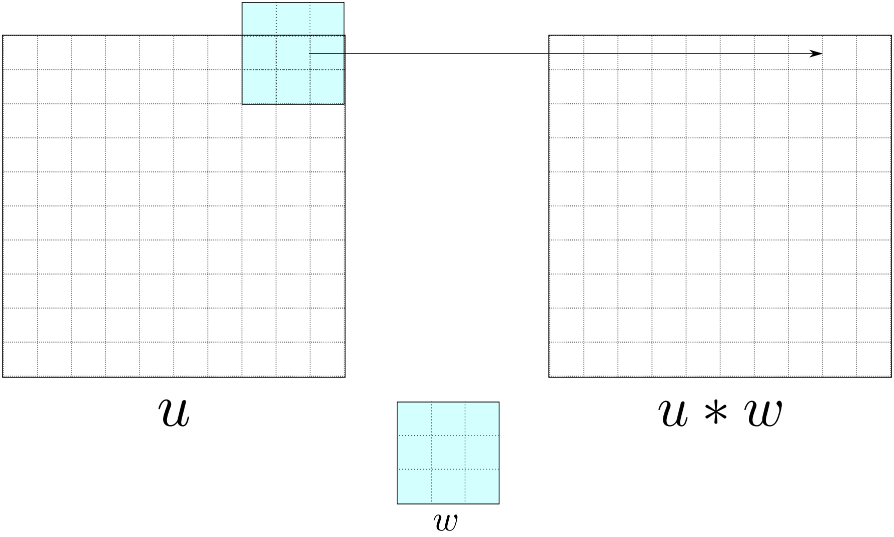
Convolutional Layers : Visual Illustration



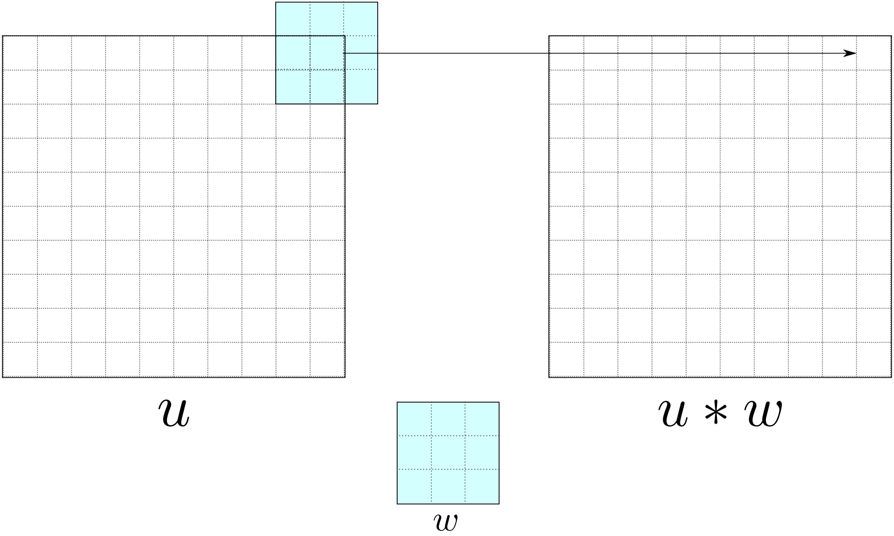
Convolutional Layers : Visual Illustration



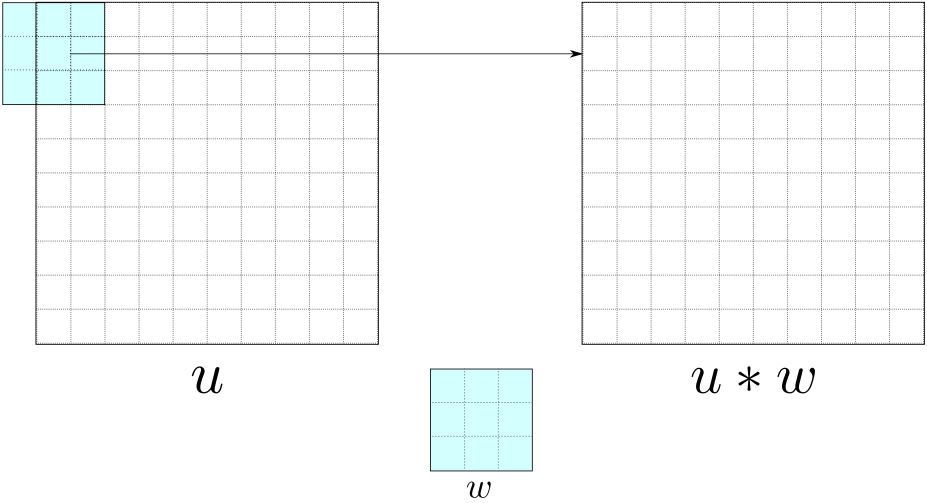
Convolutional Layers : Visual Illustration



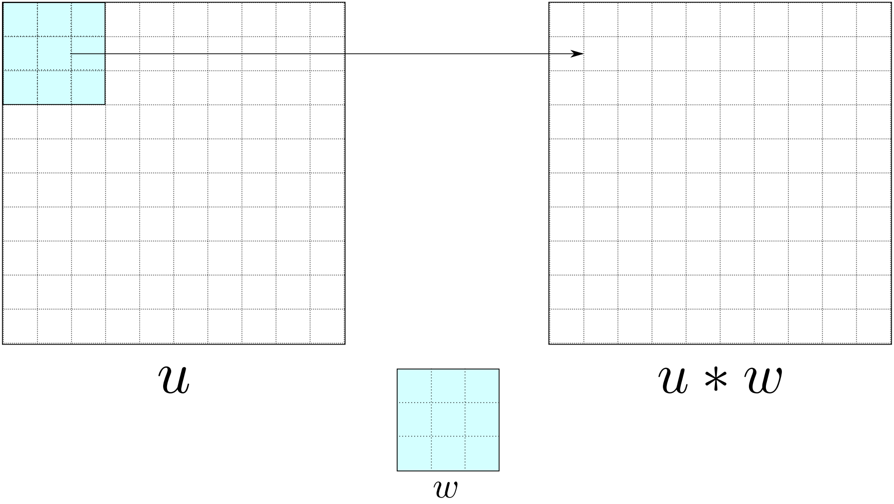
Convolutional Layers : Visual Illustration



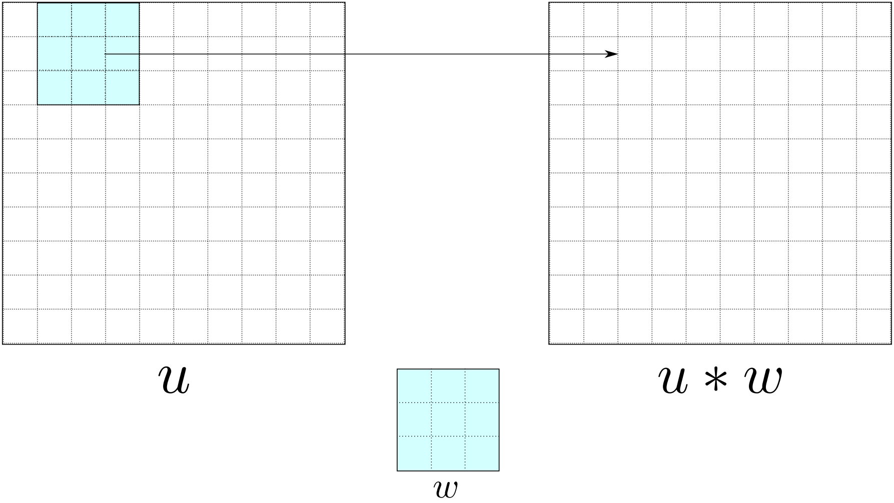
Convolutional Layers : Visual Illustration



Convolutional Layers : Visual Illustration

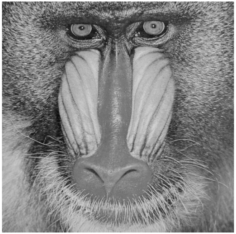


Convolutional Layers : Visual Illustration



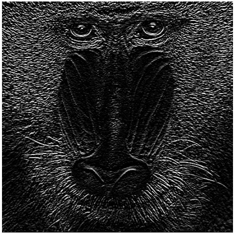
Convolutional Layers

- The filter weights w_i determine what type of "feature" can be detected by convolutional layers;
- Example, sobel filters :



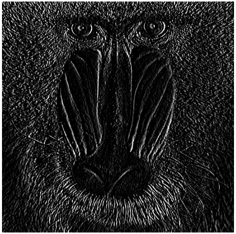
Horizontal edge

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



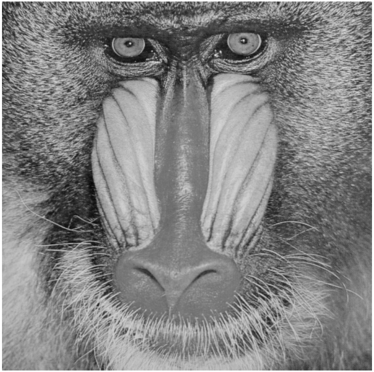
Vertical edge

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

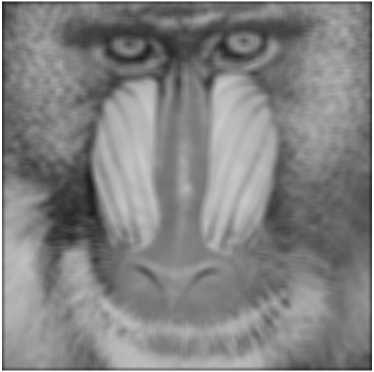


Convolutional Layers

- Convolutional filters can also act as low-pass/smoothing filters



Input image



Low-pass filtered image

Convolutional Layers

- We can also write convolution as a matrix/vector product, as in the case of fully connected layers

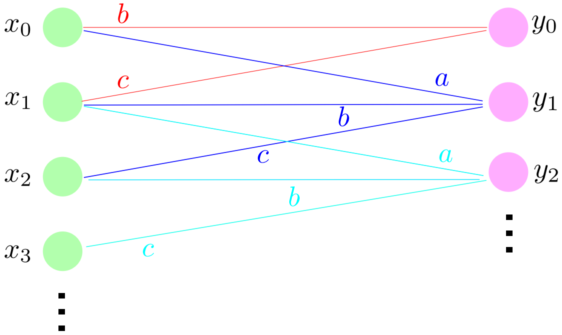
Example : discrete Laplacian operator

$$w = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \to A_w = \mathsf{K} \Bigg\downarrow \overset{\xrightarrow{\qquad\qquad \mathsf{n} \qquad\qquad}}{\begin{pmatrix} 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & & \\ -1 & 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & \\ 0 & -1 & 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ -1 & & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & -1 & 4 \end{pmatrix}}$$

- This further illustrates the drastic reduction in weight parameters (9 instead of Kn)
- Can be useful to view convolution in this manner $\to$ Backpropagation

Convolutional Layers

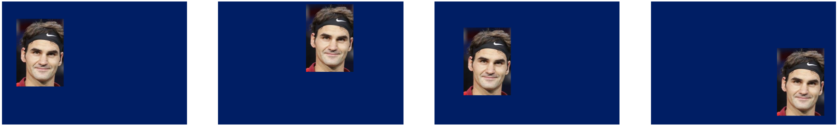
- At this point, it is good to have a more "neural network"-based illustration of CNNs



- We can see the main justifications for CNNs
  1. Sparse connectivity
  2. Weight sharing
  3. Equivariance to translation
- **optimization** of a neural network with convolutional layers through **back-propagation**

Convolutional Layers

- In many cases, we are primarily interested in **detection**;
- We would like to detect objects **wherever they are in the image**



- Formally, we would like to have some **shift invariance property**;
- This is done in CNNs by using **subsampling**, or some variant :
  - Strided convolutions
  - Max pooling
- We explain these now

Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

The Receptive Field

- The region of the image which an individual filter responds to is known as the "receptive field" of that filter
- The receptive field of a deep networks corresponds to that of the filters contained in the last layer

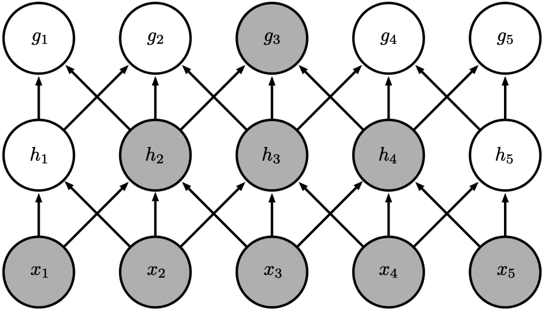


Illustration from: ***Deep learning book****, Goodfellow, I., Bengio, Y., & Courville, A., MIT Press 2016,*
https://www.deeplearningbook.org/slides/09_conv.pdf/

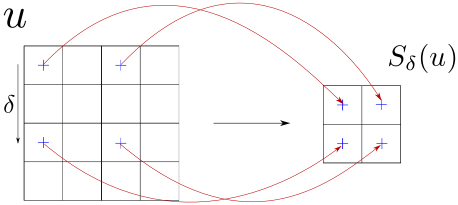
Strided convolution

- **Strided convolution** is simply convolution, followed by **subsampling**

Subsampling operator (for 1D case)

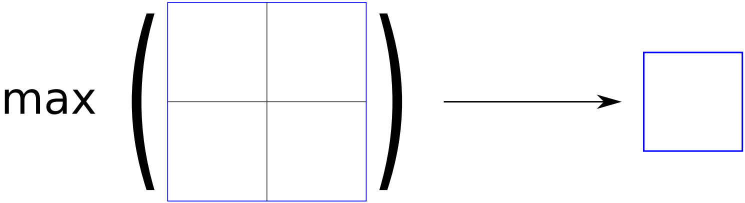
Let $x \in \mathbb{R}^n$. We define the subsampling step as $\delta > 1$, and the subsampling operator $S\delta : \mathbb{R}^n \to \mathbb{R}^{\frac{n}{\delta}}$, applied to x, as

$$S_\delta(x)\ (t) = x(\delta t), \ \text{ for } t = 0 \dots \frac{n}{\delta} - 1$$



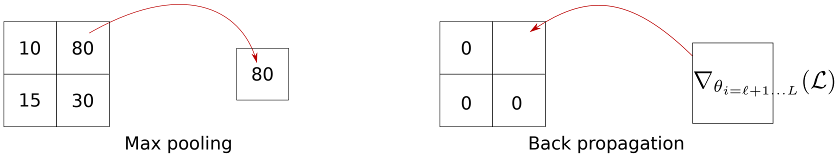
Max pooling

- **Max pooling** subsampling consists in taking the **maximum** value over a certain region
- This maximum value is the new subsampled value
- We will indicate the max pooling operator with S_m



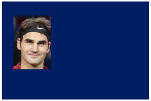
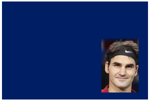
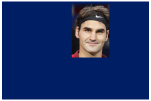
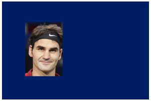
Max pooling

- Back propagation of max pooling only passes the gradient through the **maximum**



Down-sampling

- Conclusion : cascade of convolution, non-linearities and subsampling produces **shift-invariant** classification/detection
- We can detect Roger wherever he is in the image !

$$u * w$$

Convolution + non-linearity +max pooling

✓ ✓ ✓ ✓

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

How to build your CNN ?

- We have looked at the following operations : convolutions, additive biases, non-linearities
- All of these elements make up convolutional neural networks
- However, how do we put these together to create our own CNN ?
  - Programming tools ? Pytorch, Tensorflow,..
  - Datasets ?
  - **Architecture ?**
  - **Loss function ?**

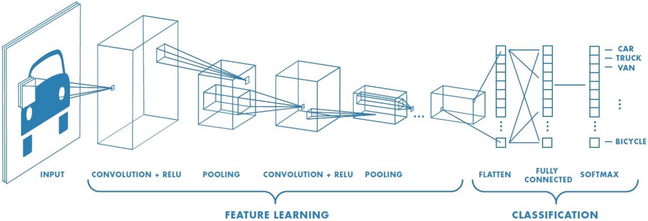
Main datasets

MNIST dataset: 60,000 28×28 pixel grey-level images containing hand-written digits. “simple” dataset, still used to display performance of modern CNNs

Caltech 101: recongnition dataset. 9,146 images, 101 object categories, each category contains between 40 and 800 images

ImageNet: 14,197,122 images, hand-annotated. Used for the ImageNet Large Scale Visual Recognition Challenge, an annual benchmark competition for object recognition algorithms

- We have input datapoints x, which we wish to classify into several, predefined classes $\{c_i\}, i = 1 \ldots K$, where K is the number of classes
- As we have seen, convolution, non-linearities, subsampling allow for robust classification that is invariant to many perturbations

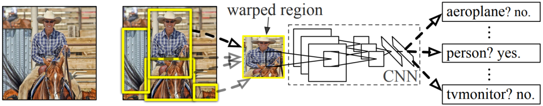


- Vast majority of CNN classification networks follow this general architecture and use the following **Cross-Entropy** loss function:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{K} y_{c_i} \log \hat{y}_{c_i} \ , \quad \hat{y}_{c_i} = \mathrm{Softmax}(z_{c_i}) = \frac{e^{z_{c_i}}}{\sum_{i=1}^{K} e^{z_{c_i}}}$$

- We can also **detect the position** of objects in images
- RNN* proposes a simple approach :
  1. Propose a list of bounding boxes in the image
  2. Pass the resized sub-images through a powerful classification network
  3. Classify each sub-image with your favourite classifier

R-CNN: ***Regions with CNN features***

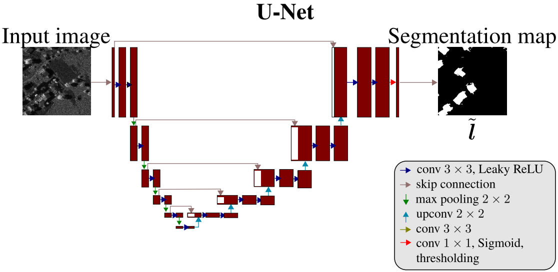


1. Input image | **2.** Extract region proposals (~2k) | **3.** Compute CNN features | **4.** Classify regions

- Many variants on this work (Fast R-NN, Faster R-CNN) etc.

* ***Rich feature hierarchies for accurate object detection and semantic segmentation***, *Girschik, R. et al. CVPR 2014*

- **Image segmentation** is a particular case of image classification, where a label is assigned to each image pixel
- Most of architectures are based on U-Net*
  1. Pooling layers to encode image semantic
  2. Residual (or skip) connections allow the flow of information to preserve fine details from the input image

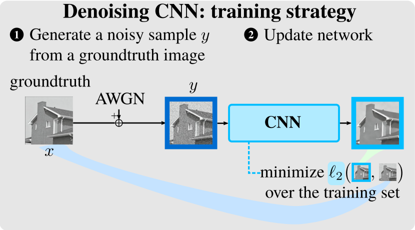


* ***U-net: Convolutional networks for biomedical image segmentation**, Ronneberger, Olaf, et al.,MICCAI 2015*

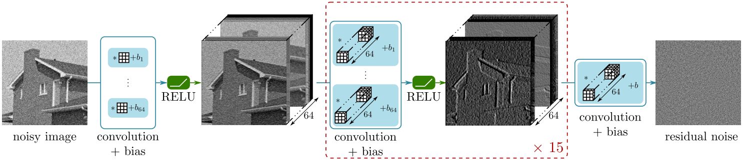
- **Goal**: reconstruct the clean signal $\mathbf{x}$ from a noisy measurement $\mathbf{y} = \mathbf{x} + \mathbf{n}$, with $\mathbf{n}$ being the noise
- Achieved by mimizing the distance between $f_{\boldsymbol{\theta}}(\mathbf{y})$ and $\mathbf{x}$ for each image pixel indexed by i:

$$\mathcal{L}_2 = \sum_i \|f_{\boldsymbol{\theta}}(y_i) - x_i\|^2$$

$$\mathcal{L}_1 = \sum_i \|f_{\boldsymbol{\theta}}(y_i) - x_i\|$$

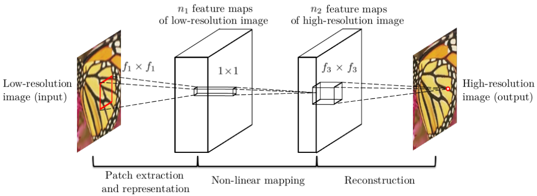


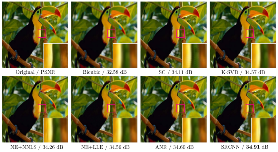
- Majority of architectures employ a residual learning strategy*



* ***Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising***, *Zhang, Kai, et al., IEEE TIP 2017*

- Image super-resolution : go from a low-resolution image to a higher-resolution one
- Relatively straightforward approach with a CNN*





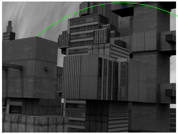
- Drawback, highly dependent on degradation used in lower-resolution images in database

* ***Learning a deep convolutional network for image super-resolution****, Chao et al, ECCV 2014*

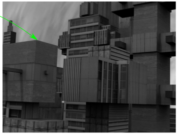
Motion estimation

- **Motion estimation** is a central task for many image processing and computer vision problems : tracking, video editing
- **Optical flow** involves estimating a vector field $(u, v) : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ where each vector points to the displacement of pixel (x, y) from an image I_1 to I_2

$$I_1(x, y) = I_2(x + u(x, y), y + v(x, y))$$



(a) Frame 1



(b) Frame 2

Optical flow

*Illustration from : **BriefMatch: Dense binary feature matching for real-time optical flow estimation**, Eilertsen, G, Forssén, P-E, Unger, J., Scandinavian Conference on Image Analysis, 2017*

Attention mechanism in image networks

- **Attention mechanism** originally developed in RNNs : addresses problem of long range dependency

  $\rightarrow$Networks exist with attention only : **transformer***

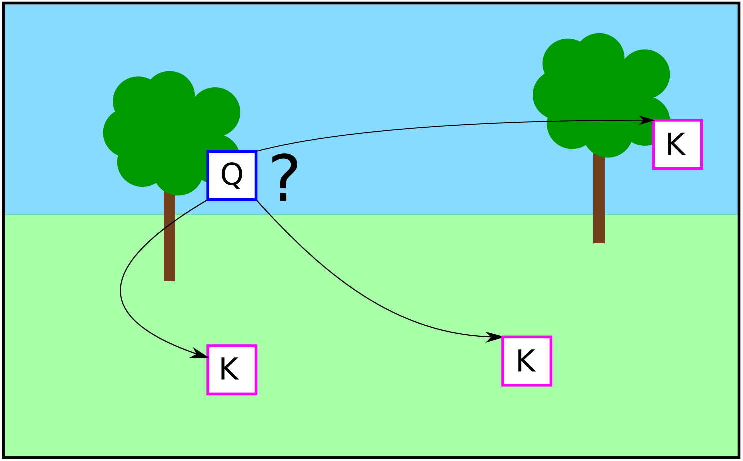
- Also used in image network architectures (usually **self-attention**)

$$Attention(Q, K, V) = \text{Softmax}(QK^T)V$$

- This equation says that the attention is a weighted version of V
- The weights are given by a softmax of the dot products between patches in Q and those in K

* ***Attention is all you need****, Vaswani et al, NIPS, 2017*

Attention mechanism in image networks



* ***Attention is all you need**, Vaswani et al, NIPS, 2017*

Attention mechanism in image networks

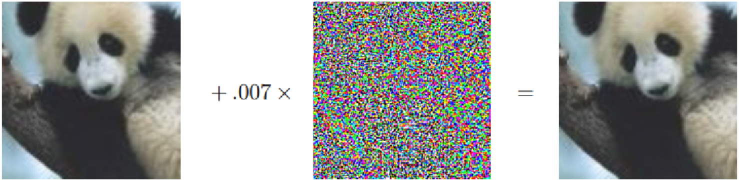
- Combined attention/convolution archtictures present the best accuracies on ImageNeT (to date*)
    - CoAt-Net7: 90.88% accuracy on ImageNet

Rank	Model	Top 1 Accuracy	Top 5 Accuracy	of params	Training Data	Paper	Code	Result	Year	Tags
1	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
2	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer JFT-3B
3	**CoAtNet-6**	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
4	**ViT-MoE-15B** (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	Transformer JFT-3B
5	**Meta Pseudo Labels** (EfficientNet-L2)	90.2%	98.8	480M	✓	Meta Pseudo Labels			2020	EfficientNet JFT-300M

* ***https://paperswithcode.com/sota/image-classification-on-imagenet***

Adversarial examples

- We often get the impression that CNNs are the end all and be all of AI
- Consistently produce state-of-the-art results on images
- However, CNNs **are not infallible** : adversarial examples[†] !



- How was this image created ???

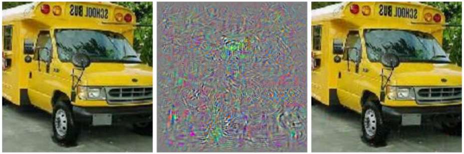
[†] ***Intriguing properties of neural networks**, Szegedy, C. et al, arXiv preprint arXiv:1312.6199, 2013*

Adversarial examples

- We often get the impression that CNNs are the end all and be all of AI
- Consistently produce state-of-the-art results on images
- However, CNNs **are not infallible** : adversarial examples[†] !
- Szegedy et al. propose[‡] add a small perturbation r that fools the classifier network f into choosing the wrong class c for $\hat{x} = x + r$

$$\arg\min_r |r|_2^2, \;\; \text{s.t}\; f(x+r) = c, \; x + r \in [0,1]^n$$

- $\hat{x}$ is the closest example to x s.t $\hat{x}$ is classified as in class c
- Minimisation with box-constrained L-BFGS algorithm

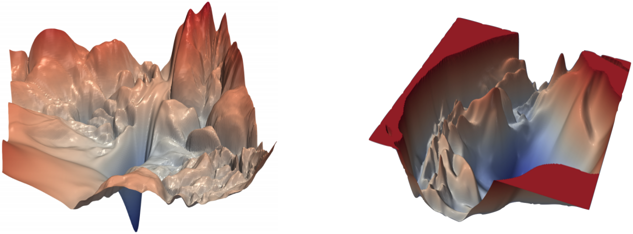


x $\quad r \quad$ $\hat{x}$

† ***Intriguing properties of neural networks**, Szegedy, C. et al, arXiv preprint arXiv:1312.6199, 2013*

‡ ***Intriguing properties of neural networks**, Szegedy, C. et al, arXiv preprint arXiv:1312.6199, 2013*

- Common explanation : the space of images is very high-dimensional, and contains many areas that are unexplored during training time



Example of loss surfaces in commonly used networks (Res-Nets)

Illustration from ***Visualizing the Loss Landscape of Neural Nets****, Li, H et al, NIPS, 2018*

Summary

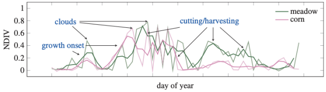
- CNNs represent the state-of-the art in many different domains/problems
- If you have an unsolved problem, there is a good chance CNNs will produce a good/excellent result
- **However : theoretical understanding is still relatively limited**
  - This leads to problems such as adversarial examples
  - It is not clear whether CNNs are truly robust/generalisable
  - This is a hot research topic, important if CNNs are to be used in industrial applications

Outline

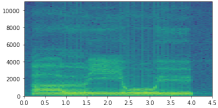
1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

Context

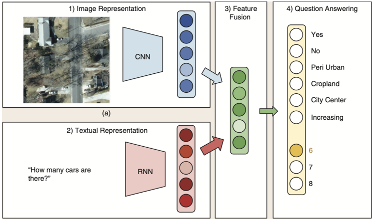
- CNNs are designed to capture spatial relationships for matrix-like data
- Signals that evolves through time are modeled as sequences



Example of Earth observation sequential data†



Example of spectrogram of an audio signal



Visual Question Answering in Remote Sensing‡

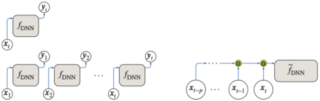
† *Illustration from* ***Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences****, Camps-Valls, Gustau, et al., John Wiley & Sons, 2021.*

‡ *Illustration from* ***Toward a collective agenda on ai for earth science data analysis****, Tuia, Devis, et al., IEEE GRSM 2021*

Context

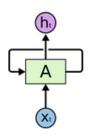
- Deep Feedforward Network can process single elements or fixed-length sequences

$$\mathbf{y} = f(\mathbf{x}_t) \quad \text{or} \quad \mathbf{y} = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-p})$$



Single-time and multi-time feedforward neural networks†

→ They fail to dynamically capture the temporal context: **Recurrent Neural Networks (RNNs)** do that: $\mathbf{y} = f(\{\mathbf{x}_t\}_{t=0}^{\tau})$

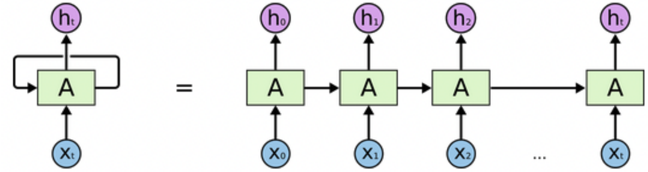


Structure of a recurrent neural network‡

† *Illustration from* ***Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences****, Camps-Valls, Gustau, et al., John Wiley & Sons, 2021.*

‡ *Illustration from* `http://ai.stanford.edu/~quocle/tutorial2.pdf`

- Relaxation of Feedforward Neural Networks to allow feedback loops
  $\rightarrow$ allows to process sequences of variable lenght
- Backpropagation is not directly applicable due to feedback loops
  $\rightarrow$ RNN can be reformulated as Deep Feedforward Networks



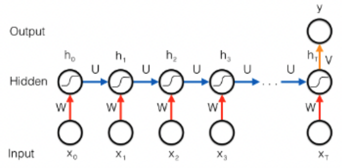
An unrolled recurrent neural network†

- Layers correspond to individual times

† *Illustration from* `http://ai.stanford.edu/~quocle/tutorial2.pdf`

Recurrent Neural Networks training

- RNN elements:
  - W: input hidden weights
  - U: hidden to hidden weigths
  - V: hidden to label weigths
  - h_t: hidden state



Structure of a recurrent neural network†

$$
\begin{aligned}
f(x) &= V h_T \\
h_t &= \sigma(U h_{t-1} + W x_t), \quad \text{for} \quad t = T, \dots, 1 \\
h_0 &= \sigma(W x_0)
\end{aligned}
$$

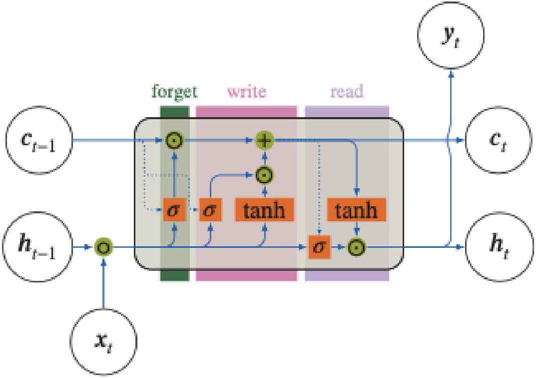
- Backpropagation through T layers involves multiplying $\times T$ the shared weights matrix $\rightarrow$ **Exploding** or **Vanishing Gradient** problem[1,2]

† *Illustration from* `http://ai.stanford.edu/~quocle/tutorial2.pdf`

[1] ***Long short-term memory.***, *Hochreiter, Sepp, and Jürgen Schmidhuber, Neural computation, 1997*

[2] ***Learning long-term dependencies with gradient descent is difficult.*** *Bengio, Yoshua et al., IEEE transactions on neural networks, 1994*

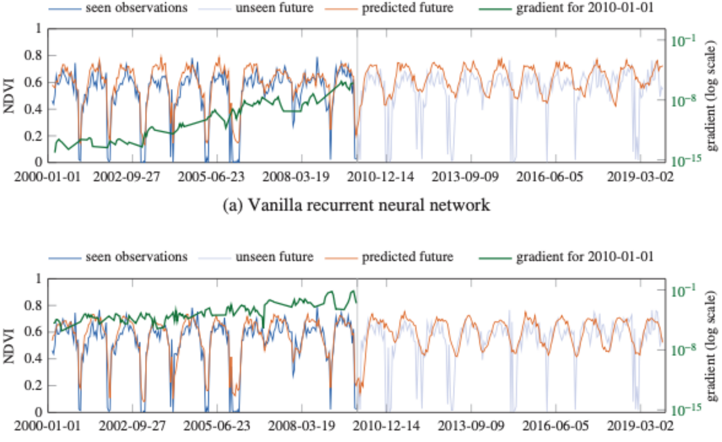
- The most popular variant of RNNs are **Long Short-Term Memory (LSTM)** networks
- Composed by several gates controlling the information flow



- The "memory" c can allow the gradient to pass through without vanishing

† *Illustration from* ***Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences****, Camps-Valls, Gustau, et al., John Wiley & Sons, 2021*

→ LSTM retrieves long-term temporal context to predict future states

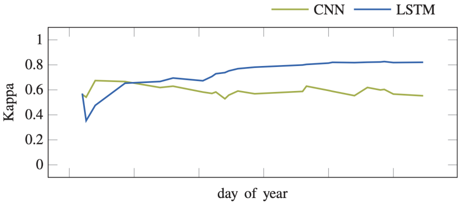


(a) Vanilla recurrent neural network

(b) Long short-term memory neural network

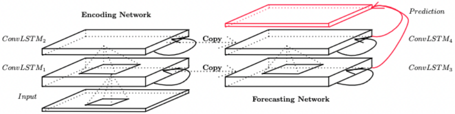
† *Illustration from* ***Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences****, Camps-Valls, Gustau, et al., John Wiley & Sons, 2021*

- LSTM are better at modeling temporal relationships than CNN



Crop classification accuracy[†]

- However, CNN can capture the spatial context
  → To account for spatio-temporal relationships, architectures combining CNNs and LSTM have been proposed

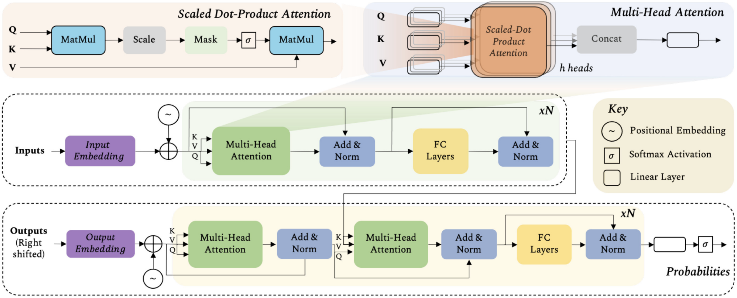


Convolutional LSTM Network[‡]

[†] *Illustration from* ***Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences****, Camps-Valls, Gustau, et al., John Wiley & Sons, 2021*

[‡] *Illustration from* ***Convolutional LSTM network: A machine learning approach for precipitation nowcasting****, Shi, Xingjian, et al., NeurIPS 2015.*

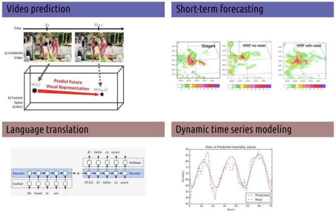
- The sequential nature of RNNs precludes parallelization within training examples: critical for long sequences
- Transformer architecture* relies entirely on attention mechanism to derive global dependences between input and oputput
- Every position in the decoder attends over all positions in the input sequence



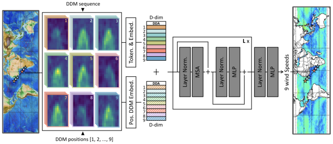
- Led to the development of Large Language Models

* ***Attention is all you need***, *Vaswani et al, NIPS, 2017*
Illustration from ***Transformers in vision: A survey***, *Khan, Salman, et al., ACM computing surveys (CSUR) 2022*

- Language Modeling
- Speech Recognition
- Music Generation/Synthesis
- Precipitation Nowcasting
- Crop classification



Examples of problems where data are modeled as sequences†



Ocean Wind speed estimation with transformers‡

† *Illustration from* ***Deep learning and process understanding for data-driven Earth system science****, Reichstein, Markus, et al., Nature 2019*
‡ *Illustration from* ***DDM-Former: Transformer networks for GNSS reflectometry global ocean wind speed estimation****, Zhao et al, Remote Sensing of Environment, 2023*

Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

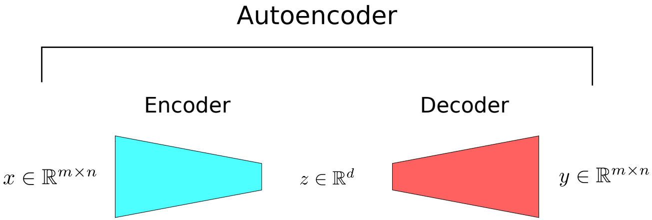
Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

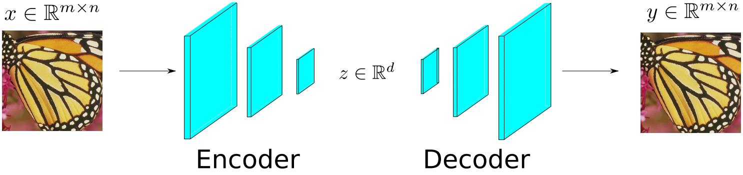
- Neural networks are often used for :
  - Classification/detection (MLPs, CNNs)
  - Modelling time-series, sequences (RNNs)
- All of these networks rely on the extraction of features to analyse data
- Idea : the network's internal representation of the data can be useful !
- **Autoencoders** and more generally **generative models** use this idea

Autoencoders

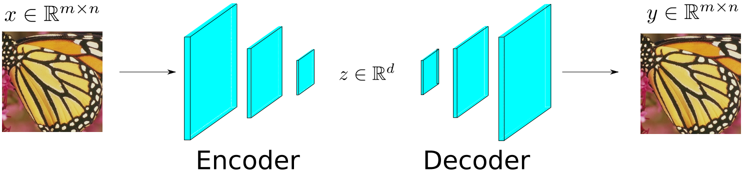
- Autoencoders consist of two networks : an **encoder** and a **decoder**
  - Encoder : map data x to a smaller **latent space**
  - Decoder : map point z back from latent space to original data space
- Main idea : the latent space is a space where it is **easier to manipulate/understand data**
- More powerful and compact representation of data



- The autoencoder is trained to minimise some norm between the input x and the output y of the decoder
- In almost all cases, we have $d << mn$
- This forces the autoencoder to learn a compact and powerful latent space



- The autoencoder is trained to minimise some norm between the input x and the output y of the decoder
- In almost all cases, we have $d << mn$
- This forces the autoencoder to learn a compact and powerful latent space



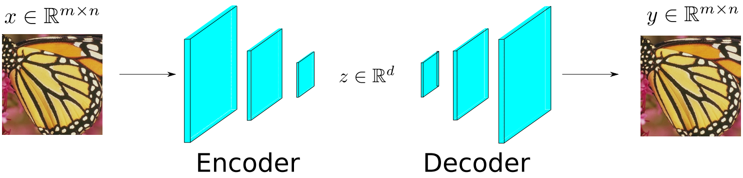
Autoencoding training minimisation problem

$$\begin{aligned}\mathcal{L}(x) &= \|y - x\|_2^2 \\ &= \sum_i^m \sum_j^n \left((\Phi_d \circ \Phi_e(x))_{i,j} - x_{i,j} \right)^2\end{aligned}$$

- Put simply : **output should look like input !**

Autoencoders

- Uses of autoencoders :
  - Data compression, dimensionality reduction
  - Classification (easier in latent space)
  - **Data generation/synthesis**



Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

Generative models

- In many applications, it is desirable to **synthesise** data
  - Video post-production
  - Data augmentation
  - Imrove performances of simulators
  - Domain adaptation

- Several types of generative models exist :
  - Restricted Bolzmann machines, Deep Belief models
  - **Variational autoencoders**
  - **Generative Adversarial Networks**;
  - **Diffusion models**;

- The common idea in these models is the **internal representation/latent space** of the network

- Modern generative models produce highly realistic, (relatively) high-definition images

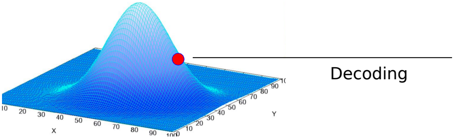
Synthesis examples from "Real NVP"†

- Before, let us take a step back to autoencoders

† ***Density estimation using Real NVP***, *L. Dinh, J. Sohl-Dickstein, S. Bengio, arXiv:1605.08803 2016*

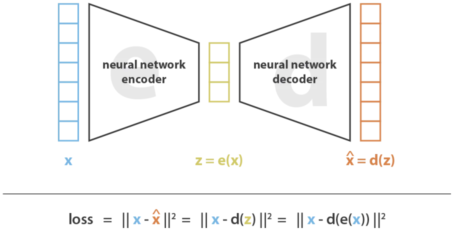
- Suppose we want to produce **random examples of data**, how would we go about this ?

- Suppose we want to produce **random examples of data**, how would we go about this ?
- We can model the latent space in a **probabilistic manner**
- Synthesis will then consist of :
  1. Sampling in the latent space
  2. Decoding to produce the random image



Probabilistic model in latent space

Synthesis of random image



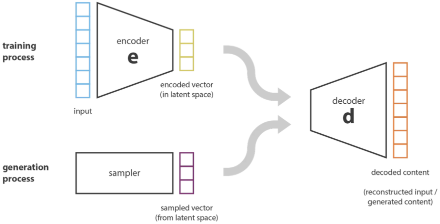
An autoencoder with its loss function†

- What is the link between autoencoders and sample generation?

† *Illustration from* `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`

Variational autoencoders - data generation

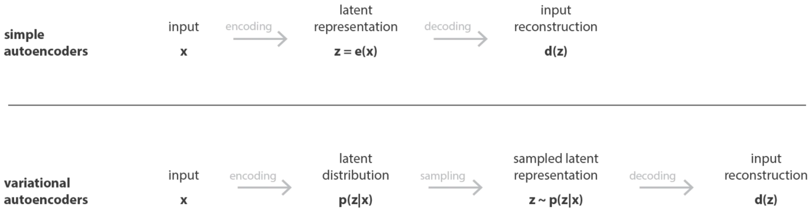
- Generate new examples by sampling into the latent space
- The quality of generated data depends on the regularity of the latent space
  →Enforce regularity with a probabilistic approach



† *Illustration from* `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`

Variational autoencoders - a probabilistic approach

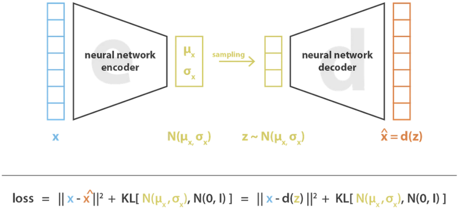
- Regularize the training of an autoencoder to avoid overfitting and ensure that the latent space has a specific organization
- The latent space distribution is often chosen to be Normal
- Regularization is enforced by an additional loss function based on the **Kullback-Leibler divergence** between the latent space distribution and the Normal distribution



† *Illustration from* `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`

Variational autoencoders - a probabilistic approach

- VAE loss function is the sum of a **Reconstruction error** and a **Regularization term** eforcing the chosen prior distribution on the latent space



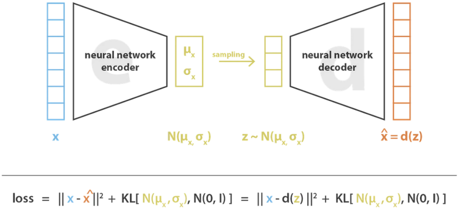
† *Illustration from* `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`

Variational Autoencoder

- There remains one more important detail : how to backpropagate through samples of z ? **Random variable, not differentiable**
- Solution : **"reparametrisation trick"**, make the random element an network input
- In the Gaussian case, where q_ϕ is a multivariate Gaussian vector, with mean $\boldsymbol{\mu}$ and diagonal covariance matrix σId, this gives

$$z = \boldsymbol{\mu} + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathrm{Id})$$

- $\boldsymbol{\mu}$ and σ are produced by the encoder
- Thus, backpropagation can be carried out w.r.t to network parameters

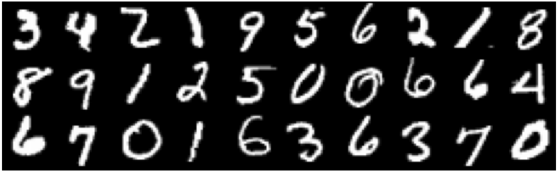


† *Illustration from* `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`

Variational autoencoder **in practice**

- Let us take the following case, well-adapted to the **mnist dataset** :
  - Prior : $p_{\boldsymbol{\theta}}(z) \sim \mathcal{N}(0, \text{Id})$
  - Variational approximation : $q_{\phi}(z|x) \sim \mathcal{N}(\boldsymbol{\mu}, \sigma\text{Id})$, where $(\boldsymbol{\mu}, \sigma) = \Phi_e(x)$
  - Likelihood : $p_{\boldsymbol{\theta}}(x|z) \sim Ber(y)$, where $y = \Phi_d(z)$

$$\mathcal{L} = \overbrace{\sum_{i=1}^{mn} x_i \log y_i + (1 - x_i)\log(1 - y_i)}^{\text{Reconstruction error}} - \underbrace{\frac{1}{2}\sum_{j=1}^{d}\left(\boldsymbol{\mu}_j^2 + \sigma_j^2 - 1 - \log\left(\sigma_j^2\right)\right)}_{\text{KL divergence}}$$



Variational autoencoder results

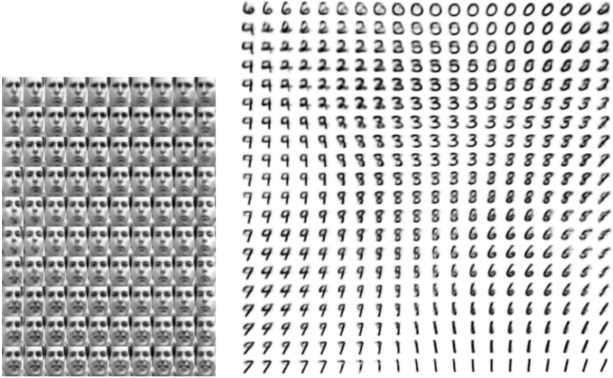
- Some results of variational autoencoders on mnist data : random samples

(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Auto-Encoding Variational Bayes, *D. P. Kingma, M. Welling, arXiv preprint arXiv:1312.6114, 2013*

Variational autoencoder results

- Some results of VAEs on mnist, face data : uniform samples



(a) Learned Frey Face manifold (b) Learned MNIST manifold

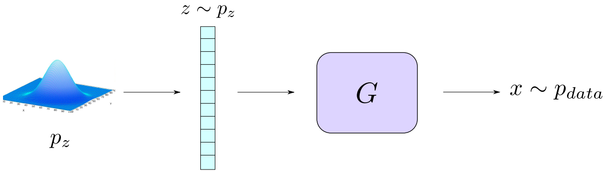
Auto-Encoding Variational Bayes, *D. P. Kingma, M. Welling, arXiv preprint arXiv:1312.6114, 2013*

Variational Autoencoders : summary

- Rigourous framework to autoencode data onto a probabilisitcally modelled latent space
- Advantages
  - Theoretically-motivated, loss function meaningful
  - Learn to and from mapping (encoder and decoder)
- Drawbacks
  - Have to re-write loss function for each different model, not always easy
  - In practice, do not produce as complex examples as **Generative Adversarial Networks**

Generative Adversarial networks

- The GAN contains only the **decoder** part of an autoencoder
  - The code z is **explicitly sampled** from a chosen distribution p_z (contrary to the VAE)
- The decoder is referred to here as the "Generator"



- We suppose that the data in the databse follows a distribution p_{data}
- We want to make the distribution of $y = G(z)$, p_G, similar to p_{data}*

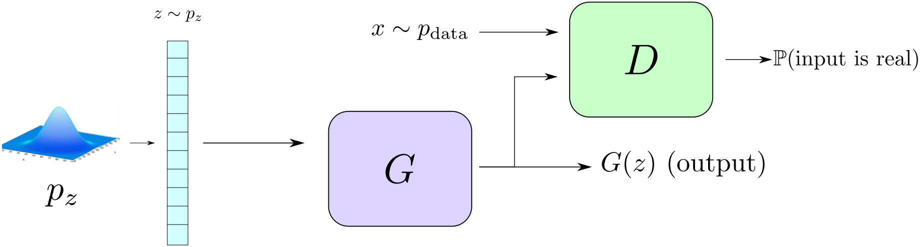
* *Why can we not do this via the KL divergence as before ? Too high dimensionality (previously, we worked in the latent space)*

- However, with no reconstruction error, how do we make x look like the data ?
- Answer : Train another network : a **Discriminator** D (or “Adversarial Network”)

Generative Adversarial networks

- However, with no reconstruction error, how do we make x look like the data ?
- Answer : Train another network : a **Discriminator** D (or "Adversarial Network")
- $D : \mathbb{R}^{mn} \rightarrow [0, 1]$ is trained to **identify "good" (or "true")** examples of the data
- $G : \mathbb{R}^{z} \rightarrow \mathbb{R}^{mn}$ is trained to **produce realistic data examples**
- The two networks are trained at the same time, and **each try to fool the other !**

- The full GAN architecture looks like this



Generative Adversarial networks

- The discriminator is a really interesting idea, why ?
- Reliable and powerful image/data models are difficult to establish
- It is difficult to say whether an image is “good” or not
  - The discriminator acts as a **learned image norm** !
- How is this is achieved ? Via a well-designed loss function

GAN loss

- Train generator G and the discriminator D in a minimax optimisation problem

$$\min_G \max_D \quad \overbrace{\mathbb{E}_{x\sim p_{data}}\left[\log D(x)\right]}^{D \text{ is trying to recognize true data}} \quad + \underbrace{\mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(z))\right)\right]}_{\substack{G \text{ is trying to fool } D, \\ \text{but } D \text{ is trying not to be fooled}}}$$

GAN loss

- Train generator G and the discriminator D in a minimax optimisation problem

$$\min_G \max_D \quad \overbrace{\mathbb{E}_{x\sim p_{data}}\left[\log D(x)\right]}^{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(z))\right)\right]}_{\substack{G \text{ is trying to fool } D, \\ \text{but } D \text{ is trying not to be fooled}}}$$

- Minimisation w.r.t G
  - Second term is low, $\implies 1 - D(G(z))$ is close to 0 $\implies$ D is recognising $G(z)$ as a true data example : G **has fooled** D

GAN loss

- Train generator G and the discriminator D in a minimax optimisation problem

$$\min_G \max_D \quad \overbrace{\mathbb{E}_{x\sim p_{data}}\left[\log D(x)\right]}^{D \text{ is trying to recognize true data}} \quad + \underbrace{\mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(z))\right)\right]}_{\substack{G \text{ is trying to fool } D, \\ \text{but } D \text{ is trying not to be fooled}}}$$

- Minimisation w.r.t G
  - Second term is low, $\implies 1 - D(G(z))$ is close to 0 $\implies D$ is recognising $G(z)$ as a true data example : G **has fooled** D
- Maximisation w.r.t D
  - First term is high $\implies D(x)$ is close to 1 : D **is learning to recognize true data**
  - Second term is high $\implies 1 - D(G(z))$ is close to 1 : D **is not getting fooled by** G

- At the beginning of the training, the examples from G are not very good : D can spot them easily
- At the end of training, the discriminator should not be able to tell the true data from the generated data : $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log(4)$
- Optimisation alternates between minimisation and maximisation steps

- Here are some results of the original GAN paper*

- In the space of four years, these results have been vastly improved on
- There are many, many GAN variants. We present a few now

* ***Generative Adversarial Nets***, *Goodfellow et al, NIPS 2014*

Conditional Generative Adversarial networks

- The Conditional GAN allows a label $\mathbf{c}$ to be added to the loss function
- It is then possible to generate examples of a given class

$$\min_G \max_D [\log D(x|\mathbf{c})] + [\log (1 - D(G(z|\mathbf{c})))]$$

* ***Conditional Generative Adversarial Nets***, *Mirza, M. and Osindero, S., arXiv preprint arXiv:1411.1784, 2014*

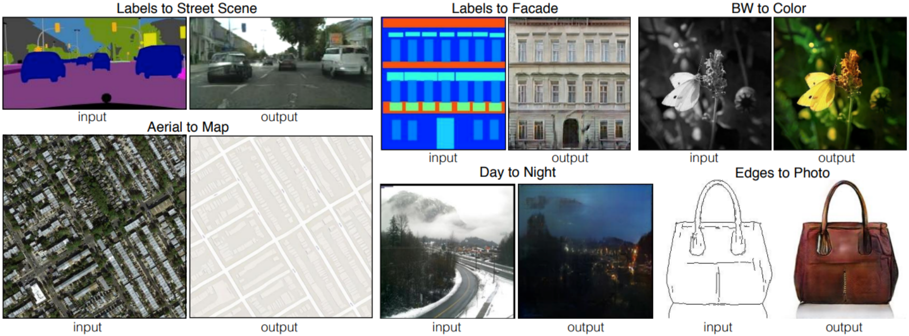
Generative Adversarial networks

- Examples of results of Conditional GAN

* ***Conditional Generative Adversarial Nets***, *Mirza, M. and Osindero, S., arXiv preprint arXiv:1411.1784, 2014*

Generative Adversarial networks

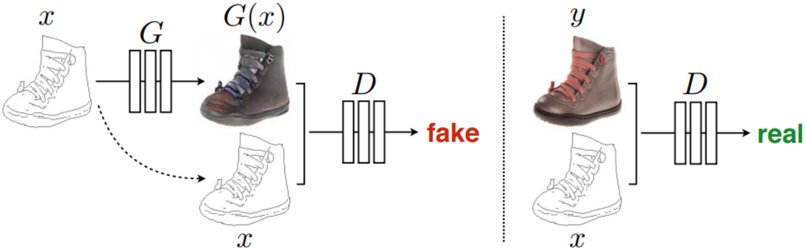
- GANs have also been modified to carried out **domain translation**
- One of the most well-known networks is **Pix-To-Pix**†



† ***Image-to-Image Translation with Conditional Adversarial Nets***, *P Isola, J.-Y. Zhu, T. Zhou, A. A. Efros, CVPR, 2017*

Generative Adversarial networks

- Instead of going from a random code to an image, the GAN learns to map one representation to another

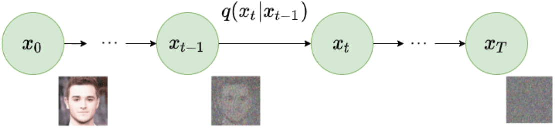


- Can be used for tasks such as data augmentation, image inpainting

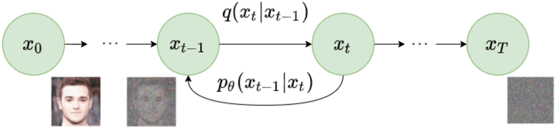
† ***Image-to-Image Translation with Conditional Adversarial Nets***, *P Isola, J.-Y. Zhu, T. Zhou, A. A. Efros, CVPR, 2017*

Denoising Diffusion Probabilistic Models (DDPM)

- Consider a forward diffusion process which progressively adds noise

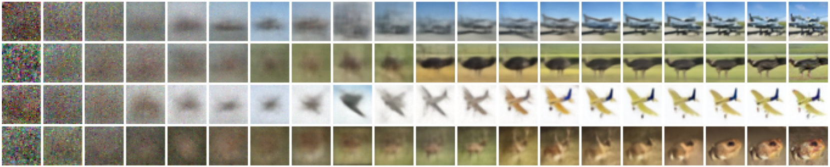


- reversing the process allows to generate new samples $\mathbf{x}_0 \sim q(\mathbf{x})$ starting from white noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$



***Deep unsupervised learning using nonequilibrium thermodynamics**, Sohl-Dickstein, Jascha, et al., ICML 2015*
***Denoising diffusion probabilistic models**, Ho, Jonathan, et al., NeurIPS 2020*

Denoising Diffusion Probabilistic Models (DDPM)



Unconditional CIFAR10 progressive generation

***Denoising diffusion probabilistic models**, Ho, Jonathan, et al., NeurIPS 2020*

Outline

1. Deep Feedforward Networks
   - Undestanding deep networks
   - Backpropagation and neural network training
2. Convolutional Neural Networks
   - Introduction, notation
   - Convolutional Layers
   - Down-sampling and the receptive field
   - Applications of CNN's
3. Recurrent Neural Networks for Sequence Modeling
4. Autoencoders and Generative Models
   - Autoencoders
   - Generative Models
5. Conclusion

From Deep Learning to the Earth Sciences

Opportunities

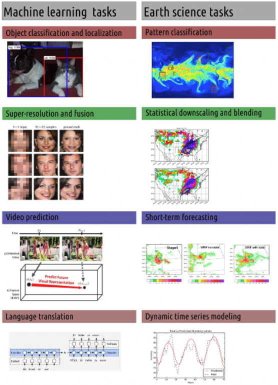
- Analogies between the types of data addressed with classical deep learning
  - Images $\rightarrow$ 2-D data fields
  - Videos $\rightarrow$ 2-D data fields evolving in time
  - Naural language speech signals $\rightarrow$ Dynamic time-series of Earth system variables
- Tasks such as classification, regression, anomaly detection, and dynamic modeling are typical problems in both computer vision and geosciences

Challenges

- RGB Images $\neq$ Hyperspectral images
- Variables are often not i.i.d.
- How to integrate multi-modal data?
  - Geometry
  - Temporal / Spatial resolution
- Sources of noise
- No benchmark: many unlabeled data $\rightarrow$ Self-Spervised Learning (SSL)

Deep learning and process understanding for data-driven Earth system science*, Reichstein, Markus, et al., Nature 2019*

From Machine Learning to Earth Sciences



These slides are inspired to the Deep Learning course of **Nicolas Thome** and the Convolutional Neural Network course of **Alasdair Newson**. Many thanks!

† *Illustration from **Deep learning and process understanding for data-driven Earth system science**, Reichstein, Markus, et al., Nature 2019*

Properties of convolution

1. Associativity : $(f * g) * h = f * (g * h)$
2. Commutativity : $f * g = g * f$
3. Bilinearity : $(\alpha f) * (\beta g) = \alpha\beta(f * g)$, for $(\alpha, \beta) \in \mathbb{R} \times \mathbb{R}$
4. Equivariance to translation : $(f * (g + \tau))\,(t) = (f * g)(t + \tau)$

Associativity, commutativity

- Associativity+commutativity implies that we can carry out convolution in any order
- There is no point in having two or more consecutive convolutions
  - This is true in fact for any linear map

Equivariance to translation

- Equivariance implies that the convolution of any shifted input $(f + \tau) * g$ contains the **same information** as $f * g$ †
- This is useful, since we want to detect objects **anywhere in the image**

† if we forget about border conditions for a moment

Convolutional Layers

- Note : optimisation of loss w.r.t one parameter w_k **involves entire image**
- **Weights are "shared" across the entire image**
- This notion of **weight sharing** is one of the main justifications of using CNNs
- In practice, we do not calculate dw_k and dx_k ourselves, we use the **automatic differentiation** tools of Tensorflow, Pytorch etc.

2D+feature convolution

- Several filters are used per layer, let us say K filters : $\{w_1, \ldots, w_K\}$
- The resulting vectors/images are then stacked together to produce the next layer's input $u^{\ell+1} \in \mathbb{R}^{m \times n \times K}$

$$u^{\ell+1} = [u * w_1, \ldots, u * w_K]$$

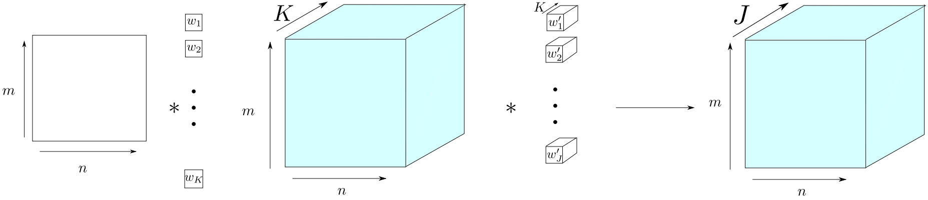
- Therefore, the next layer's weights must have a depth of K. The 2D convolution with an image of depth K is defined as

$$(u * w)_{y,x} = \sum_{i,j,k} u(i,j,\mathbf{k})\, w(y-i, x-j, \mathbf{k})$$

Useful explanation : `https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215`

Convolutional layers

- Illustration of several consecutive convolutional layers with different numbers of filter

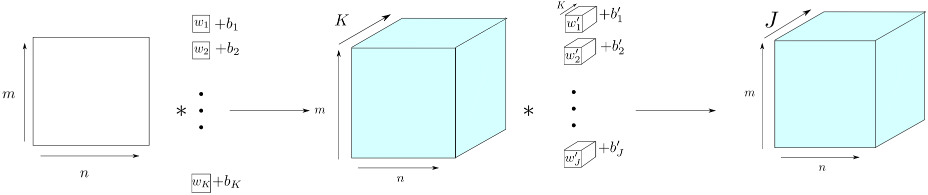


- Each layer contains “image” with a depth, where each channel corresponds to a different filter response
- Each layer is a concatenation of several features : rich information

Useful explanation : `https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215`

Convolutional layers - a note on Biases

- A note on biases in neural networks : **each output layer is associated with one bias**
- There is **not** one bias per pixel
- This is coherent with the idea of **weight sharing** (bias sharing)



The Receptive Field

- The region of the image which an individual filter responds to is known as the “receptive field” of that filter
- The receptive field of a deep networks corresponds to that of the filters contained in the last layer

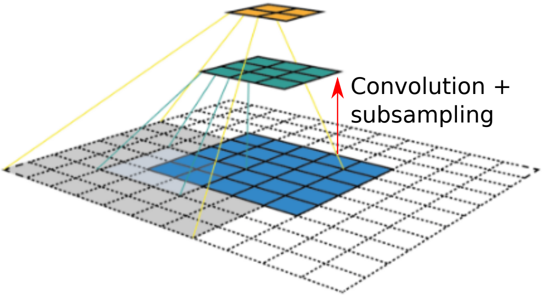


Illustration from : Applied Deep Learning, Andrei Bursuc, `https://www.di.ens.fr/~lelarge/dldiy/slides/lecture_7/`

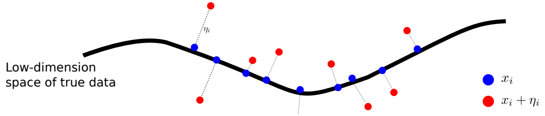
Denoising autoencoder

- Naive autoencoder may lead to overfitting, poor robustness and difficulty to interpret the latent space
- We would like to make the encoder/decoder robust to **small perturbations** in the input data
- One solution : the **denoising autoencoder**

Denoising autoencoder

- Idea : add noise $\boldsymbol{\eta}$ to the input

$$\mathcal{L}(x) = \|\Phi_d \circ \Phi_e(x + \boldsymbol{\eta}) - x\|_2^2$$



- Example of autoencoder use : interpolation of complex data

Interpolation of complex data†

† ***Generative Visual Manipulation on the Natural Image Manifold***, *J-Y. Zhu, P. Krähenbühl, E. Schechtman, A. Efros, CVPR 2016*