EuroSDR

European Spatial Data Research

Deep Learning for Remote Sensing – EduServ Course

Sébastien Lefèvre, Loic Landrieu

Mars 2020

www.eurosdr.net

Presentation outline

1 Deep Learning Basics

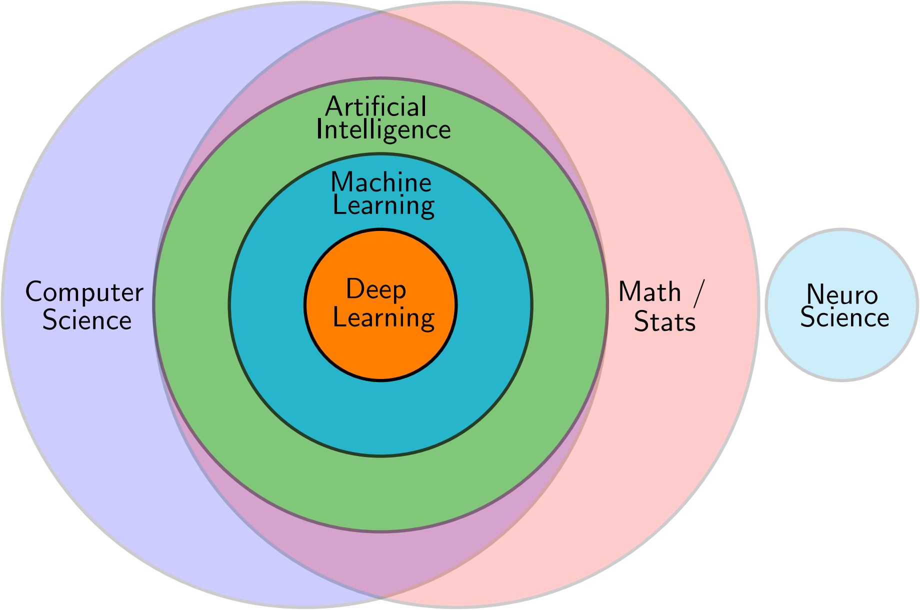
Outline

1 Deep Learning Basics

Presentation Layout

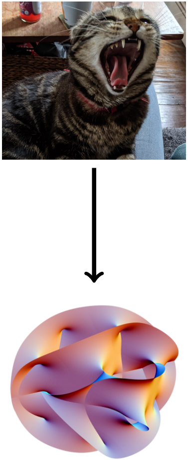
1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

Math vs CS vs AI vs ML vs DL

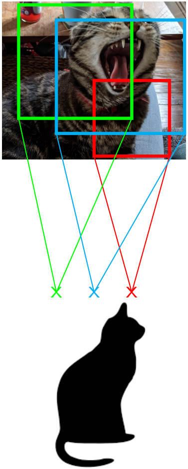


Beyond the Hype: AI vs ML vs DL

- **Math/Stats:** Problem is an ill-posed problem.
  Eg: projection onto the cat picture manifold (?) [impractical].



- **Math/Stats:** Problem is an ill-posed problem.
  Eg: projection onto the cat picture manifold (?) [impractical].
- **Computer Science:**
  Eg: template matching [impractical]

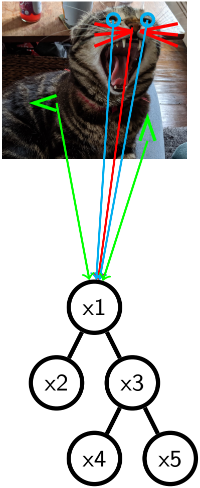


- **Math/Stats:** Problem is an ill-posed problem.

  Eg: projection onto the cat picture manifold (?) [impractical].
- **Computer Science:**

  Eg: template matching [impractical]
- **Artificial Intelligence (pre-ML)**

  Eg: we build a whiskers/eyes/ears detectors and combine them through a set of rules.



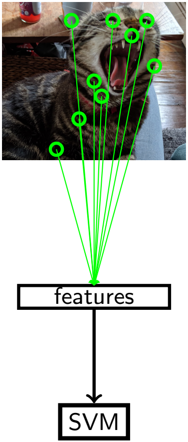
- **Math/Stats:** Problem is an ill-posed problem.

  Eg: projection onto the cat picture manifold (?) [impractical].
- **Computer Science:**

  Eg: template matching [impractical]
- **Artificial Intelligence (pre-ML)**

  Eg: we build a whiskers/eyes/ears detectors and combine them through a set of rules.
- **Machine Learning (pre-DL):**

  Eg: We train a cat picture classifier from 10 000 images of cat/non-cat using our hand-made image descriptors based on interest points.



Beyond the Hype: AI vs ML vs DL

- **Math/Stats:** Problem is an ill-posed problem.

  Eg: projection onto the cat picture manifold (?) [impractical].
- **Computer Science:**

  Eg: template matching [impractical]
- **Artificial Intelligence (pre-ML)**

  Eg: we build a whiskers/eyes/ears detectors and combine them through a set of rules.
- **Machine Learning (pre-DL):**

  Eg: We train a cat picture classifier from 10 000 images of cat/non-cat using our hand-made image descriptors based on interest points.
- **Deep Learning:**

  Eg: We train a network from 1, 000, 000 raw cat/non-cat images.

Presentation Layout

1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

Objective Statement of DL

- Data: $[x_i]_{i=1}^N$, Target:$[y_i]_{i=1}^N$
- **Objective:** learn a function f such that: $f(x_i)$ is *close to* y_i for most i.
- In classical ML, instead of working with x we work with features handcrafted by experts to be as expressive as possible.
- In the DL paradigm, we operate directly on the raw data.
- **Two key questions:**
- What kind of function is f?
- How to chose it such such that $f(x_i) \sim y_i$

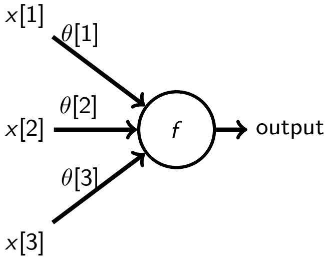
x 

y a cat

Artificial Neuron

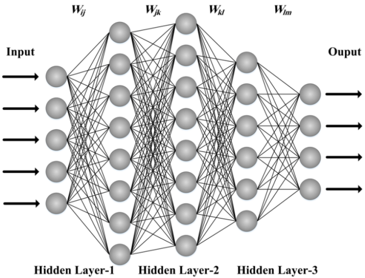
- Functions f typically composed of elementary bricks: neurons.
- $n(x) = \sigma(\sum_k \theta[k]x[k])$
- x : inputs
- θ_k : weights
- σ : non-linearity
- $n(x)$: output
- **In short:** a matrix product $x^\intercal\theta$ and non-linearity.
- Non linearity essential (or else networks simplify to matrix products).

  σ = sigmoid, Relu=max$(0, x)$.



Network Architecture

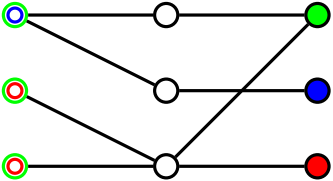
- Organization in layers, ie Multilayer Perceptron.
- **Feature map:** neurons activation at each layer = *a learned descriptor of the data.*
- The deeper layers extracts more complicated / abstract features.



credit : pubs.sciepub.com/ajmm

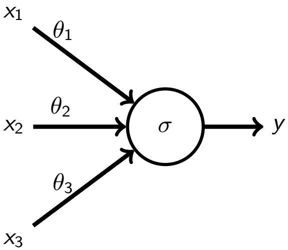
Network Architecture

- Organization in layers, ie Multilayer Perceptron.
- **Feature map:** neurons activation at each layer $=$ *a learned descriptor of the data.*
- The deeper layers extracts more complicated / abstract features.
- **Receptive Fields:** of a neuron: input which will influence its activation function.
- **Network architecture must reflects the nature of the data**.
- **Universal Approximation Theorem:** any functions (continuous, on a compact set) can be approximated to arbitrary precision by a network with sufficient width.



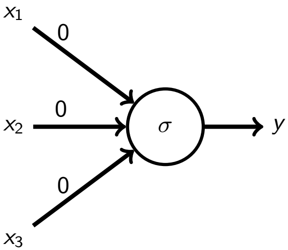
Training a network

- What a neural network outputs depends on the weights of each of its neuron: f_θ.



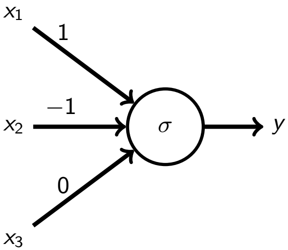
Training a network

- What a neural network outputs depends on the weights of each of its neuron: f_θ.



Training a network

- What a neural network outputs depends on the weights of each of its neuron: f_θ.
- For a given architecture, we call Ω the set of all possible weights θ.
- **Training a neural network:** finding a good parameterization $\theta \in \Omega$.



Choosing a Loss function

- "$f(x_i)$ *close to* y_i " can be hard to deal with directly.
- We formulate it as an optimization problem using a *Loss function*: $\mathcal{L}$.
- $\mathcal{L}$ chosen such that the closer $f(x_i)$ is to y_i, the closer $\mathcal{L}(f(x_i), y_i)$ is to 0.
- Eg. MSE: $\|f(x_i) - y_i\|^2$, cross entropy :$\log([x_i[y_i])$.
- Now we can formulate the problem as:

$$\arg\min_{\theta \in \Omega} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$

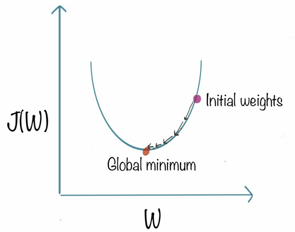
- An optimization problem with many parameters.

$$\arg\min_{\theta \in \Omega} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$

- Gradient $\nabla_\theta \mathcal{L}(f_\theta(x_i), y_i) =$ *slope* of $\mathcal{L}$ at θ, ie a direction in which we can change the parameters θ to increase a $\mathcal{L}$.
- By going in the opposite direction, we decrease $\mathcal{L}$.
- We can minimize $\mathcal{L}$ with the Gradient Descent algorithm:

$\theta^{(0)}$ `initialization`
`for i = 1..ite_max`
$\quad \theta^{(i)} \leftarrow \theta^{(i)} - \lambda \nabla \mathcal{L}(f_\theta(x), \hat{z})$

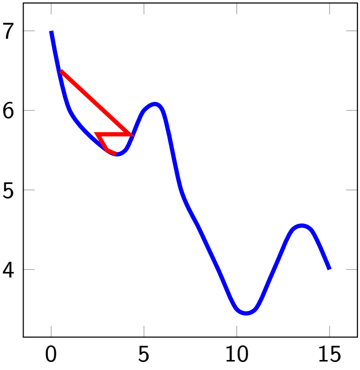
- λ: learning rate.
- If $\mathcal{L}$ is convex wrt. θ we have guarantees to find global optimum with adapted λ.



credit: kdnuggets.com

Gradient Descent

$$\arg\min_{\theta\in\Omega} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$



- Gradient $\nabla_\theta \mathcal{L}(f_\theta(x_i), y_i) =$ *slope* of $\mathcal{L}$ at θ, ie a direction in which we can change the parameters θ to increase a $\mathcal{L}$.
- By going in the opposite direction, we decrease $\mathcal{L}$.
- We can minimize $\mathcal{L}$ with the Gradient Descent algorithm:

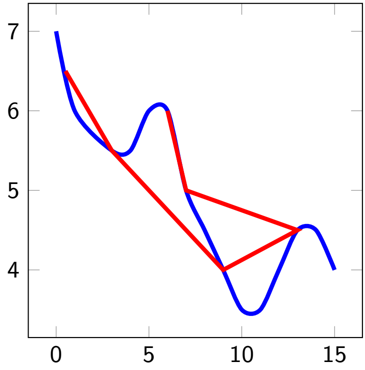
$\theta^{(0)}$ `initialization`
`for i = 1..ite_max`
    $\theta^{(i)} \leftarrow \theta^{(i)} - \lambda \nabla \mathcal{L}(f_\theta(x), \hat{z})$

- λ: learning rate.
- If $\mathcal{L}$ is convex wrt. θ we have guarantees to find global optimum with adapted λ.
- However, $\mathcal{L}$ is typically **not convex**.
- We get stuck in local optima.

Stochastic Gradient Descent (SGD)

- An adaptation of gradient descent: compute gradients using a single random data point at the time:
- Stochastic Gradient descent:

$\theta^{(0)}$ `initialization`
`for i = 1..ite_max`
    $p \leftarrow$ `random permutation 1 .. n`
    `for j = 1..n`
        $k \leftarrow p[j]$
        $\theta^{(i)} \leftarrow \theta^{(i)} - \lambda \nabla \mathcal{L}(f_\theta(x_k), \hat{z}_k)$



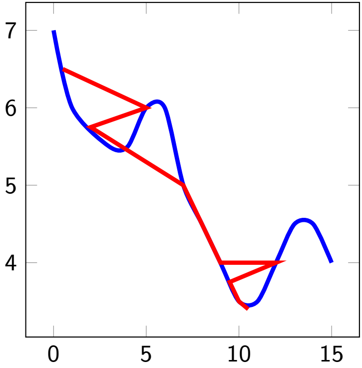
- **Problem:** $\nabla \mathcal{L}(f_\theta(x_j), \hat{z})$ is a poor substitute for $\nabla \mathcal{L}(f_\theta(x), \hat{z})$.
- Usually won't converge at all.

Batch-Stochastic Gradient Descent (BSGD)

- A compromise: we group elements by batches of size B.
- Batch-Stochastic Gradient descent:

$\theta^{(0)}$ `initialization`
`for i = 1..ite_max`
    $p \leftarrow$ `random permutation 1 .. n`
    `for` $j = \lfloor B \rfloor$
        $b = p[j \times B \cdots j \times (B+1)]$
        $\theta^{(i)} \leftarrow \theta^{(i)} - \lambda \nabla \mathcal{L}(f_\theta(x_b), \hat{z}_b)$

- $B = n$: GD, $B = 1$ SGD
- Large batches: fast convergence, can get stuck.
- Small batches: slow convergences, better minima found.
- Must be chosen in accord to λ.



Accelerated Schemes

- In practice, practitioners use accelerated schemes
- Heuristics based on **momentum** and **adaptive learning rate**, ie some

  Momentum: (fading) memory of gradients past. Helps overcome

  Adaptive learning rate: each parameter has its own learning rate. Helps
- In practice, use ADAM: Adaptive Moment Estimation for most problems.
- Already integrated in modern framework, requires zero effort to switch.

- **Objective:** efficiently compute $\nabla_\theta \mathcal{L}(f_\theta(x)), y)$.
- Organization in L layers: $\theta = (\theta_1, \cdots, \theta_L)$, with $f_\theta = f^{(L)}_{\theta_L} \circ \cdots \circ f^{(1)}_{\theta_1}$.
- Intermediary features: $z^{(i)} = f^{(i)}_{\theta_i} \circ \cdots \circ f^{(1)}_{\theta_1}(x)$
- Chain Rule of Differentiation:

$$\nabla_{\mathcal{L}} = \frac{\partial \mathcal{L}(x, \hat{z})}{\partial x}$$

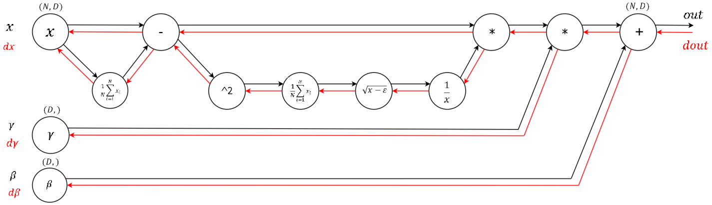
$$\nabla_{\theta_L} \mathcal{L}(z^{(L)}, \hat{z}) = \frac{\partial z^{(L)}}{\partial \theta_L} \frac{\partial \mathcal{L}(z^{(L)}, \hat{z})}{z^{(L)}} = \frac{\partial z^{(L)}}{\partial \theta_L} \nabla_{\mathcal{L}}$$

$$\nabla_{\theta_{L-1}} \mathcal{L}(z^{(L)}, \hat{z}) = \frac{\partial z^{(L-1)}}{\partial \theta_{L-1}} \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \frac{\partial \mathcal{L}(z^{(L)}, \hat{z})}{z^{(L)}} = \frac{\partial z^{(L-1)}}{\partial \theta_{L-1}} \nabla_L$$

$$\vdots$$

$$\frac{\partial \mathcal{L}(f(x), \hat{z})}{\partial \theta_1} = \frac{\partial z^{(1)}}{\partial \theta_1} \nabla_2$$

Backpropagation, cont'd



credit: medium.com/@karpathy

- A forward-backward approach:

- compute the feature map $z^{(1)} \cdots z^{(L)}$

- compute the gradients $\nabla_{\mathcal{L}}, \nabla_L \cdots \nabla_1$.

- Requires to keep all intermediate values in memory.
- Memory-mongers can be used to reduce memory impact.
- An automated process, requires zero code or calculation when using standard layers and modern frameworks.

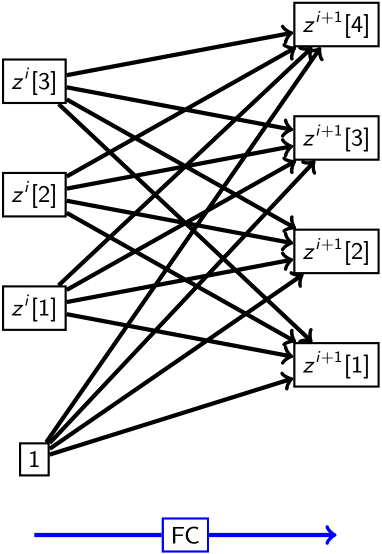
Presentation Layout

1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

- Simplest building block.
- Equivalent to learn a matrix multiplication M and a bias b:

$$z^{i+1} = Mz^{i} + b.$$

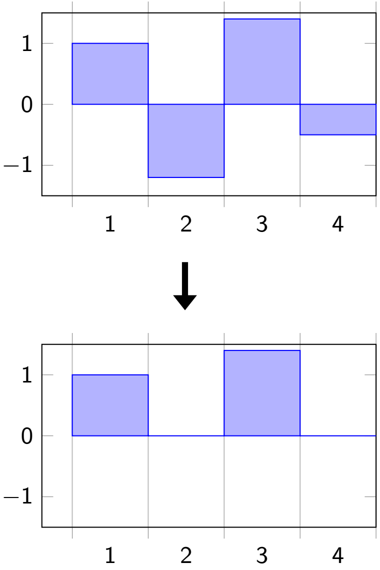
- Number of parameters:
- d_i size of z^i
- d_{i+1} size of z^i
- # parameters: $(d_i + 1) \times d_{i+1}$



Non-Linearity

- Necessary to alternate linear and non linear operations.
- **Rectified Linear Unit (ReLu)**: simplest non-linearity.

$$\sigma(x) \mapsto \max(0, x)$$



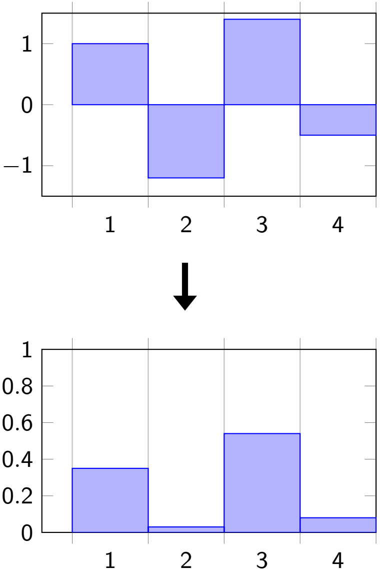
Non-Linearity

- Necessary to alternate linear and non linear operations.
- **Rectified Linear Unit (ReLu)**: simplest non-linearity.

$$\sigma(x) \mapsto \max(0, x)$$

- **Softmax**: maps a vector to a distribution.

$$\sigma([x]_{i=1}^{n}) \mapsto \left[\frac{\exp(x_i)}{\sum_i \exp(x_i)}\right]_{i=1}^{n}$$



Non-Linearity

- Necessary to alternate linear and non linear operations.
- **Rectified Linear Unit (ReLu)**: simplest non-linearity.

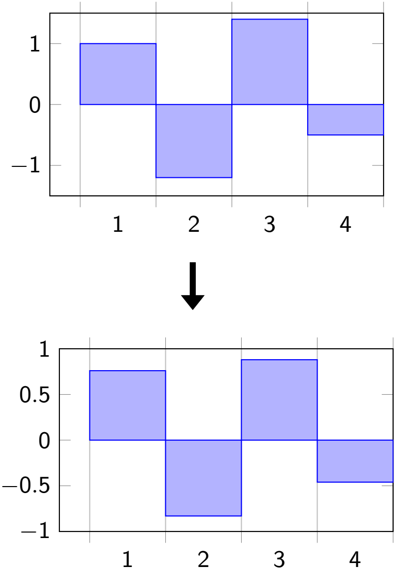
$$\sigma(x) \mapsto \max(0, x)$$

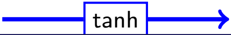
- **Softmax**: maps a vector to a distribution.

$$\sigma([x]_{i=1}^{n}) \mapsto \left[\frac{\exp(x_i)}{\sum_i \exp(x_i)}\right]_{i=1}^{n}$$

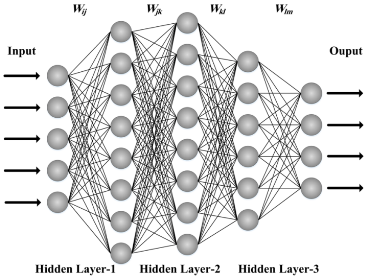
- **Sigmoid**: smooth, maps to [-1,1]

$$\sigma(x) \mapsto \tanh(x)$$



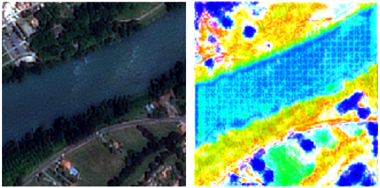


- **Problem:** the size of W increase quadratically for FC layers.
- 100×100 image : 10^8 parameters per layer (for grey scale images, more for RGB). **Unmanageable!**



credit : pubs.sciepub.com/ajmm

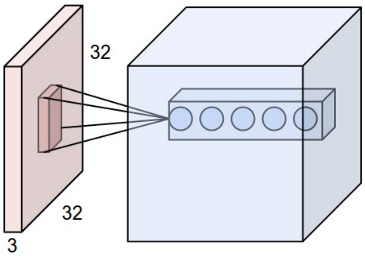
- **Problem:** the size of W increase quadratically for FC layers.
- 100×100 image : 10^8 parameters per layer (for grey scale images, more for RGB). **Unmanageable!**
- **Solution:** Organize layer into image-like cubes.



- **Problem:** the size of W increase quadratically for FC layers.
- 100×100 image : 10^8 parameters per layer (for grey scale images, more for RGB). **Unmanageable!**
- **Solution:** Organize layer into image-like cubes.
- Local convolutions: values only depend on a small number of points in previous layer (*receptive field*).

$$f^{(l+1)}[i,j] = \sum_{k,j \in [-1,0,1]^2} w_{k,l} f^{(l)}[i+k, j+l]$$

- Number of parameters of convolution $k \times k$ from map of depth d_1 to a map of depth $d_2 = \left(\underbrace{k \times k \times d_1}_{input} + 1 \right) \underbrace{\times d_2}_{output}$
- Very successful for images, exploits the spatial structure

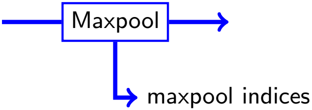


Pooling Layers

- **Objective:** decrease the size of a feature map.
- **Maxpool:**

$$f^{(l+1)}[i,j] = \max_{k,j \in [-1,0,1]^2} f^{(l)}[2i+k, 2j+l]$$

- Divides the size of the feature map by 4 for images.



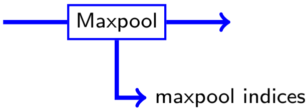
Pooling Layers

- **Objective:** decrease the size of a feature map.
- **Maxpool:**

$$f^{(l+1)}[i,j] = \max_{k,j \in [-1,0,1]^2} f^{(l)}[2i+k, 2j+l]$$

- Divides the size of the feature map by 4 for images.
- **Meanpool, medianpool**: viable alternatives for noisy data.

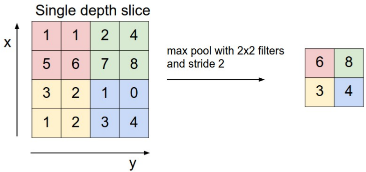
credit : cs231n.github.io

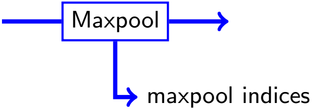


- **Objective:** decrease the size of a feature map.
- **Maxpool:**

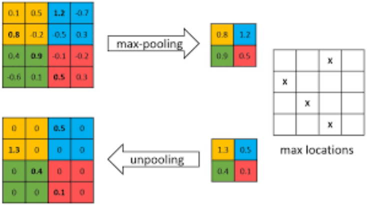
$$f^{(l+1)}[i,j] = \max_{k,j \in [-1,0,1]^2} f^{(l)}[2i+k, 2j+l]$$

- Divides the size of the feature map by 4 for images.
- **Meanpool, medianpool**: viable alternatives for noisy data.

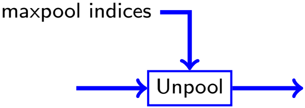




- **Unpool:** reverse of pooling layer: increse size of feature maps.
- Requires the max indices from corresponding maxpool layer.
- Operate jointly with a *twin* maxpool layer.
- **Deconvolution**: learns the reverse mapping from convolutions, accumulate in the overlap.

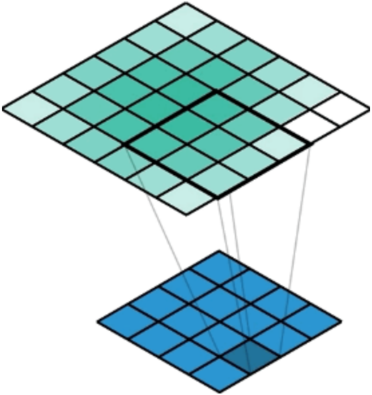


credit : DeepPainter: Painter Classification Using Deep Convolutional Autoencoders



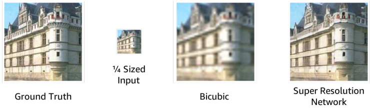
Upsampling Layers

- **Unpool:** reverse of pooling layer: increse size of feature maps.
- Requires the max indices from corresponding maxpool layer.
- Operate jointly with a *twin* maxpool layer.
- **Deconvolution**: learns the reverse mapping from convolutions, accumulate in the overlap.
- Can be used for super-resolution for example.



credit : medium.com/apache-mxnet

- **Unpool:** reverse of pooling layer: increse size of feature maps.
- Requires the max indices from corresponding maxpool layer.
- Operate jointly with a *twin* maxpool layer.
- **Deconvolution**: learns the reverse mapping from convolutions, accumulate in the overlap.
- Can be used for super-resolution for example.

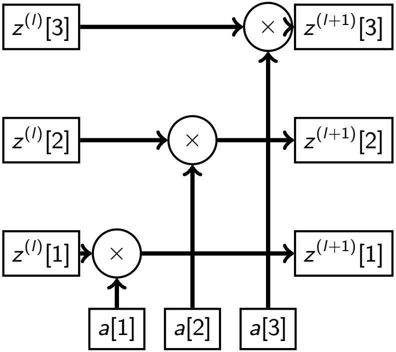


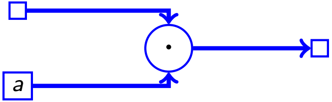
credit : SRFeat: Single Image Super-Resolution with Feature Discrimination

- **Objective:** mute some channels of feature map.
- Indirectly, which channels are the most important?
- With $a \in [0,1]^d$:

$$f^{(l+1)} = f^{(l)} \odot a$$

- a can also be a distribution (obtained from a softmax) to represent a limited amount of attention.





Attention Layers

- **Objective:** mute some channels of feature map.
- Indirectly, which channels are the most important?
- With $a \in [0, 1]^d$:

$$f^{(l+1)} = f^{(l)} \odot a$$

- a can also be a distribution (obtained from a softmax) to represent a limited amount of attention.
- The attention layer is typically obtained from another modality, eg. f from satellite images, a from cloud cover.

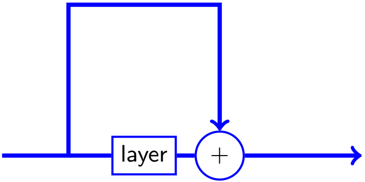
credit : www.gislounge.com

Residual Layers

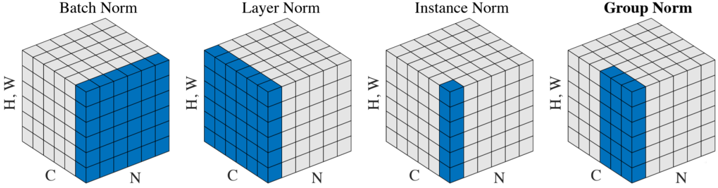
- **Observation:** Deeper network performs better.
- However, each layer can lose a small part of the information $\implies$ exponential decrease of information.
- **Solution** Residual connections:

$$x^{(l+1)} = \phi\left(x^{(l)}\right) + x^{(l)}$$

- If not needed, $\phi = 0$ will produce identity.
- ϕ can focus on **adding to / rectifying** the feature map without having to convey all information.
- Can be used in truly deep network (eg. 1000 layers)



Normalization-Layer

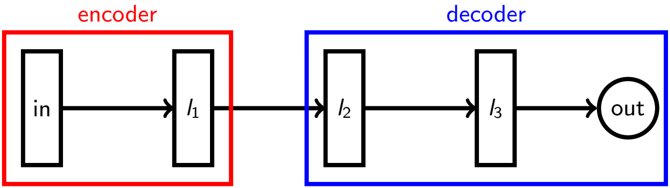


H,W : size of feature map C : Number of channels N : size of batch
credit: GroupNorm

- **BatchNorm:** each channel is normalized batchwise.
- **LayerNorm:** each layer is normalized channelwise.
- **InstanceNorm:** each channel is normalized layerwise.
- **Group Norm:** each layer is broken own in normalized groups.

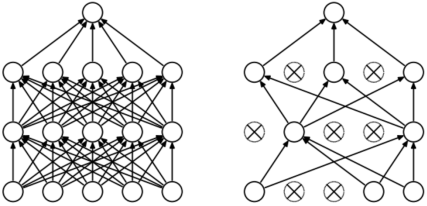
Batch-Normalization

- **Batchnorm:** scale the activation of neurons for a given layer to be of 0 mean and 1 deviation in a batch.
- **Justification:**
  - we always normalize the **inputs** of any ML approach
  - in DL, we can see the first k layers as an encoder (feature extractor), and the rest as a decoder (classifier/regressor), for any k
  - consequently, activation maps at each layers must be normalized
  - can't normalize on all training set (weights are changing) $\Rightarrow$ normalize per batch.
- **Other Benefits:**
  - Act as feature augmentation for each layer, increase robustness
  - More efficient step size when all activation/weights are comparable.
- **Batches must be sampled independently!**



Drop Out

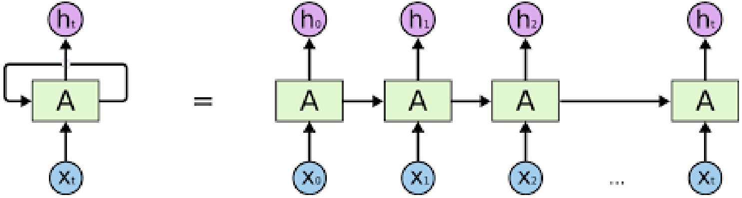
- A popular regularization layer, decrease overfiting
- Random neurons are de-activated during training.
- During inference, dropout deactivated.
- Increase resilience, promotes redundancy.
- Can be interpreted as an **ensemble method**.



(a) Standard Neural Net (b) After applying dropout.

credit towardsdatascience

Recurrent Neural Network

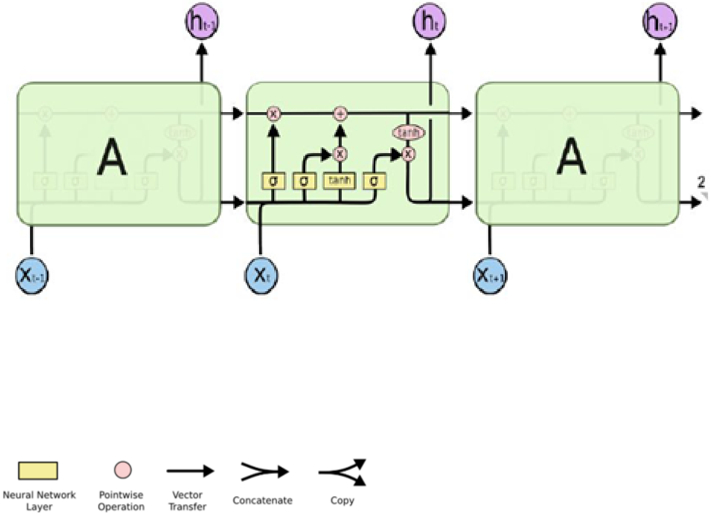


- **Objective:** modeling temporal structure.
- **General idea:** the network maintains a hidden state h_t called *memory*, which remember useful information at a given time.
- **Update:** $h_{t+1} = f(h_t, x_t)$.
- Have stability problem: vanishing / exploding gradients:

$$h_T = f(f(\cdots f(f(h_0, x_0), x_1) \cdots, x_{T-1}), x_T)$$

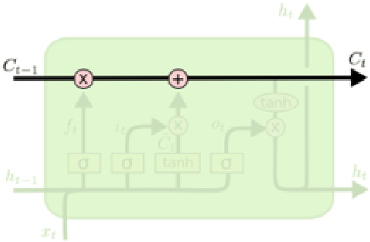
- If a small fraction of information is lost each time, exponentially decreasing information. Memory is unstable!

Long-Short Term Memory



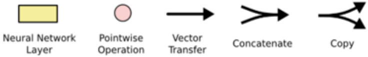
credit: https://towardsdatascience.com/

Long-Short Term Memory



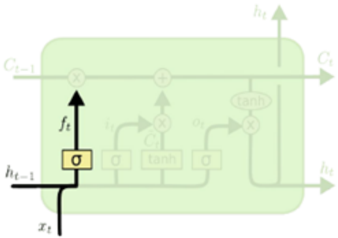
Cell State

Hidden State

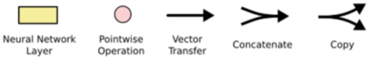


credit: https://towardsdatascience.com/

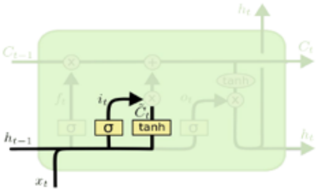
Long-Short Term Memory



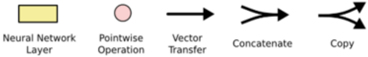
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$



credit: https://towardsdatascience.com/

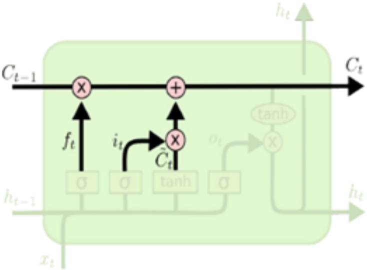


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

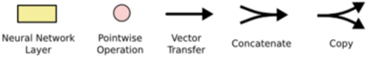


credit: https://towardsdatascience.com/

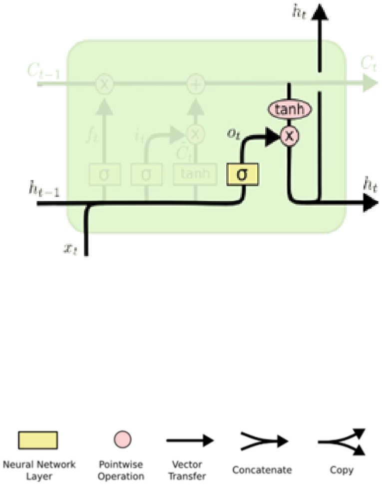
Long-Short Term Memory



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

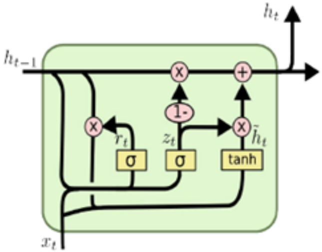


credit: https://towardsdatascience.com/



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

credit: https://towardsdatascience.com/

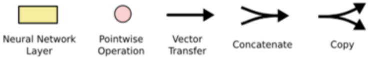


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



credit: https://towardsdatascience.com/

Presentation Layout

1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

Examples of Loss Functions

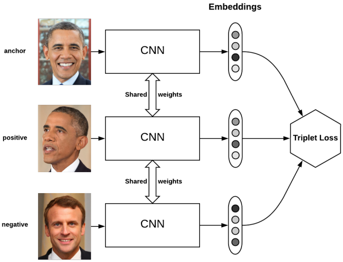
- **Reminder:** loss function: differentiable surrogate to target metrics.
- **Cross-Entropy:** surrogate to overall accuracy. With one-hot-encoding:

$$\text{Cross entropy}(z, \hat{z}) = -\frac{1}{n}\sum_{i=1}^{n} \log(z_i[\hat{z}_i[k]])$$

- **Square Difference:** for regression tasks.

$$\text{MSE}(z, \hat{z}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} \|\hat{z}_i[k] - z_i[k]\|^2$$

- **Metric Learning:** learning an embedding z such that similar elements have similar z.
  - **Anchor** the target element.
  - **Positive** element similar to the target.
  - **Negative** element different from the target.



credit: omoindrot.github.io

- Triplet Loss:

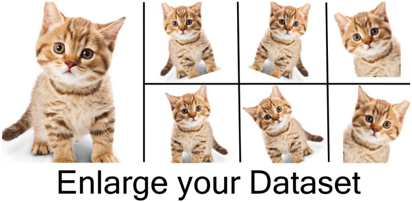
$$\mathcal{L}(A,P,N) = \max\left(\|(z(A) - z(P)\| - \|(z(A) - z(N)\| + \alpha, 0\right) .$$

- **Inference:** nearest neighbor.
- Hard example mining.

Presentation Layout

1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

- Most problems have **invariance**:
  - to noise
  - to rotation
  - to symmetries
  - etc...
- Our training set may not reflect these invariance.
- **Idea:** at each iteration, we add random noise/rotation/symmetries to the training examples.
- **Effects:**
  - artificially increase the dataset size
  - prevent network to learns unwanted training set characteristic

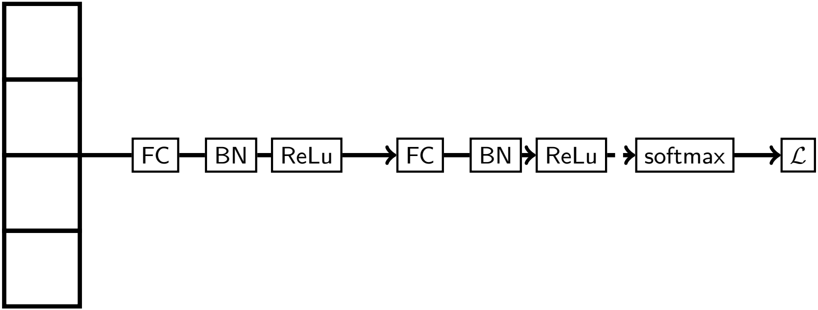


credit: medium.com

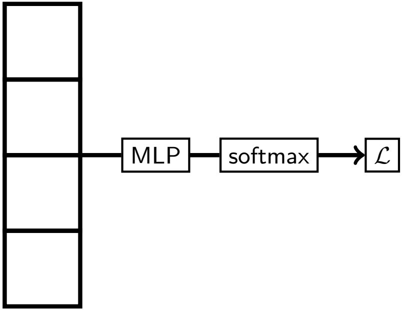
Presentation Layout

1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

Multilayer Perceptron



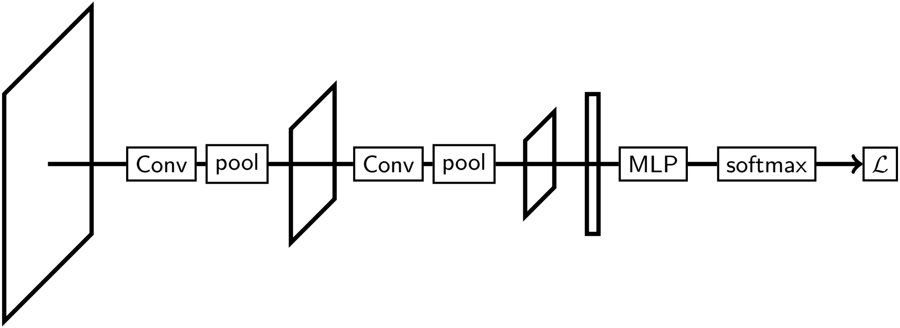
Input

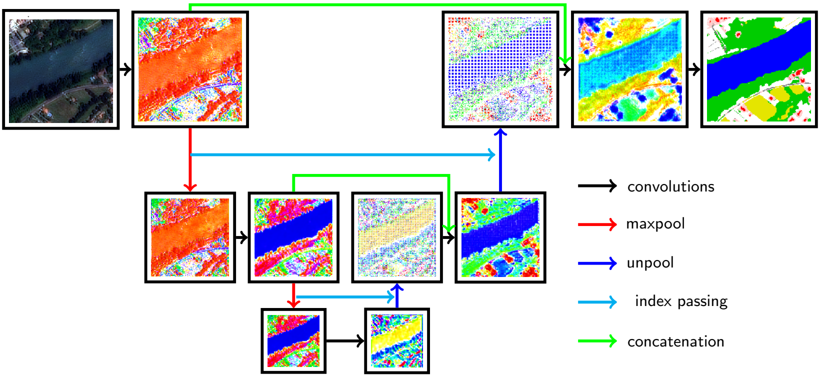


Input

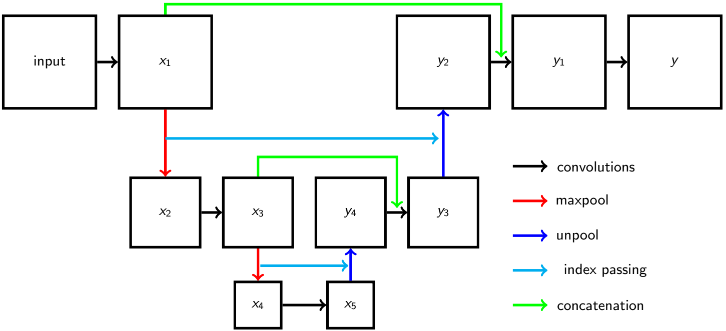
Batchnorms and ReLu are implied most of the time.

Convolutional Encoder



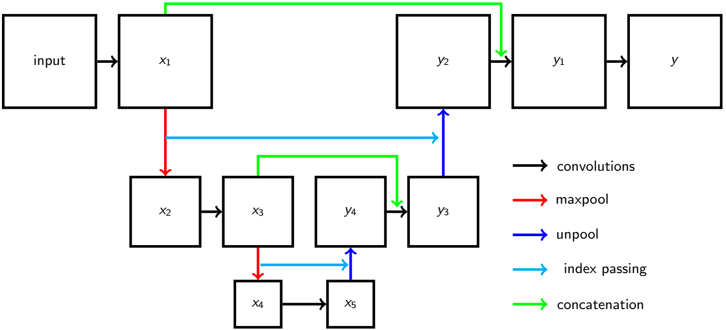

convolutions
maxpool
unpool
index passing
concatenation

Convolutional Encoder-Decoder



Tensor	Size	Tensor	Size
input	$H \times W \times 4$		
x_1	$H \times W \times d_2$		
x_2	$\lceil H/2 \rceil \times \lceil W/2 \rceil \times d_2$	y	$H \times W \times K$
x_3	$\lceil H/2 \rceil \times \lceil W/2 \rceil \times d_4$	y_2	$H \times W \times d_{10}$
x_4	$\lceil H/4 \rceil \times \lceil W/4 \rceil \times d_4$	y_3	$\lceil H/2 \rceil \times \lceil W/2 \rceil \times d_8$
x_5	$\lceil H/4 \rceil \times \lceil W/4 \rceil \times d_6$	y_4	$\lceil H/2 \rceil \times \lceil W/2 \rceil \times d_6$

Convolutional Encoder-Decoder



Encoder	Decoder
$x_1 = \text{Conv}_2(\text{Conv}_1(\text{input})))$	$y = \text{Conv}_D(y_1))$
$x_2, \text{indices}_{1\to 2} = \text{Maxpool}(x_1))$	$y_1 = \text{Conv}_{10}(\text{Conv}_9([y_2, x_1]))))$
$x_3 = \text{Conv}_4(\text{Conv}_3(x_2))))$	$y_2 = \text{Unpool}(y_3, \text{indices}_{1\to 2}))$
$x_4, \text{indices}_{2\to 3} = \text{Maxpool}(x_3))$	$y_3 = \text{Conv}_8(\text{Conv}_7([y_4, x_3]))))$
$x_5 = \text{Conv}_6(\text{Conv}_5(x_4))))$	$y_4 = \text{Unpool}(x_4, \text{indices}_{2\to 3}))$

Presentation Layout

1. Deep Learning Basics
   - Beyond the Hype
   - Deep Learning
   - Layer Bestiary
   - Loss Functions
   - Misc.
   - General Architecture
   - Implementation

- Tensorflow, Pytorch: high level python-based language doing all the tedious work.
- Example for a 1 hidden layer MLP to classify D-dimension data between K classes.
- `softmax`: goes from activation/class scores (logits) to probablities:

$$[\texttt{softmax}(v)]_i = \frac{\exp(v_i)}{\sum_j \exp(v_j)}$$

```
layer1 = Linear(D,L), layer2= Linear(L,K)
optimizer = SGD()
for i_epoch in range(n_epoch):
  for batch, gt in enumerate(training_set) % batch :B × D, gt : B
    feat = Relu(layer1(batch))                      % feat : B × L
    feat = Relu(layer2(feat))                       % feat : B × K
    output = softmax(feat)                          % output: B × K
    loss = cross_entropy(output, gt)                % compute the loss
    loss.backward()                                 % compute the gradient
    optimizer.step(i_epoch)                         % gradient step
  print("epoch = epoch, loss = avg_loss)            % monitor loss decrease
return softmax(Relu(layer2(Relu(layer1(test_set)))))
```