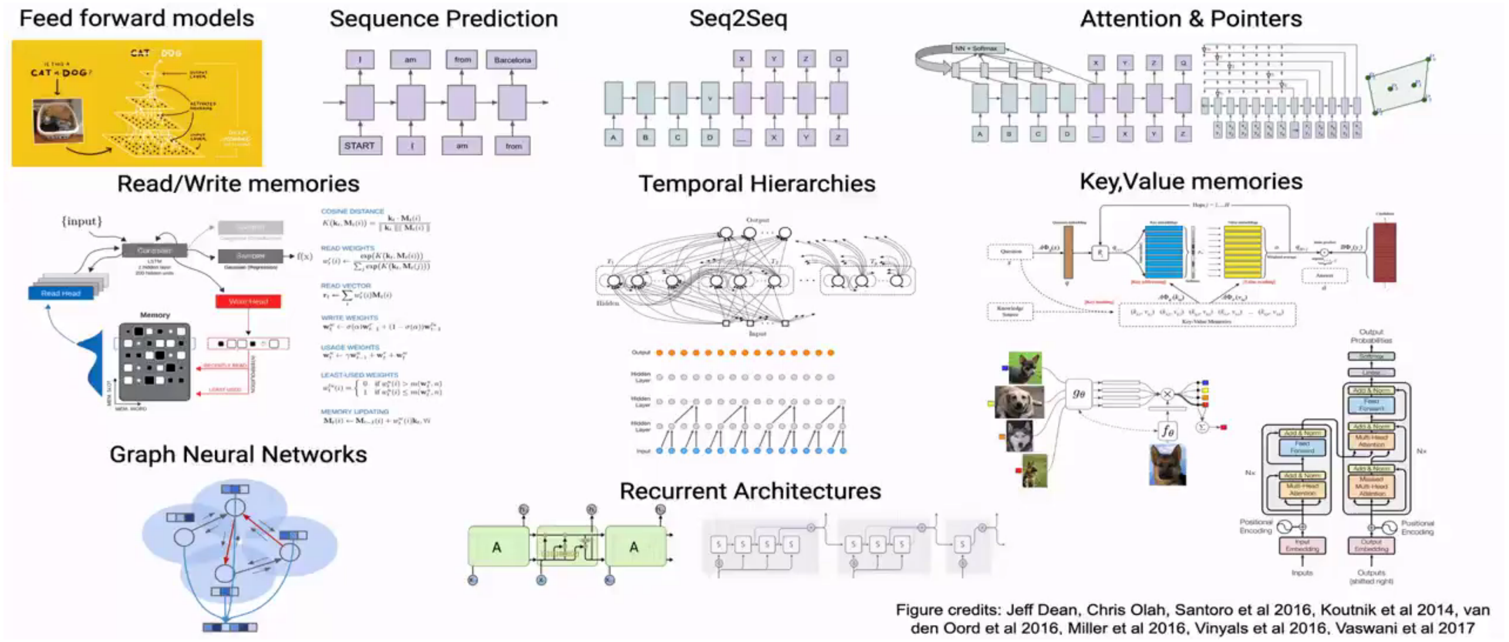
LEGO® Deep Learning Medium Brick Box (23 pcs)

ML4Jets 2020, January 15, New York University
Gilles Louppe, g.louppe@uliege.be



The toolbox

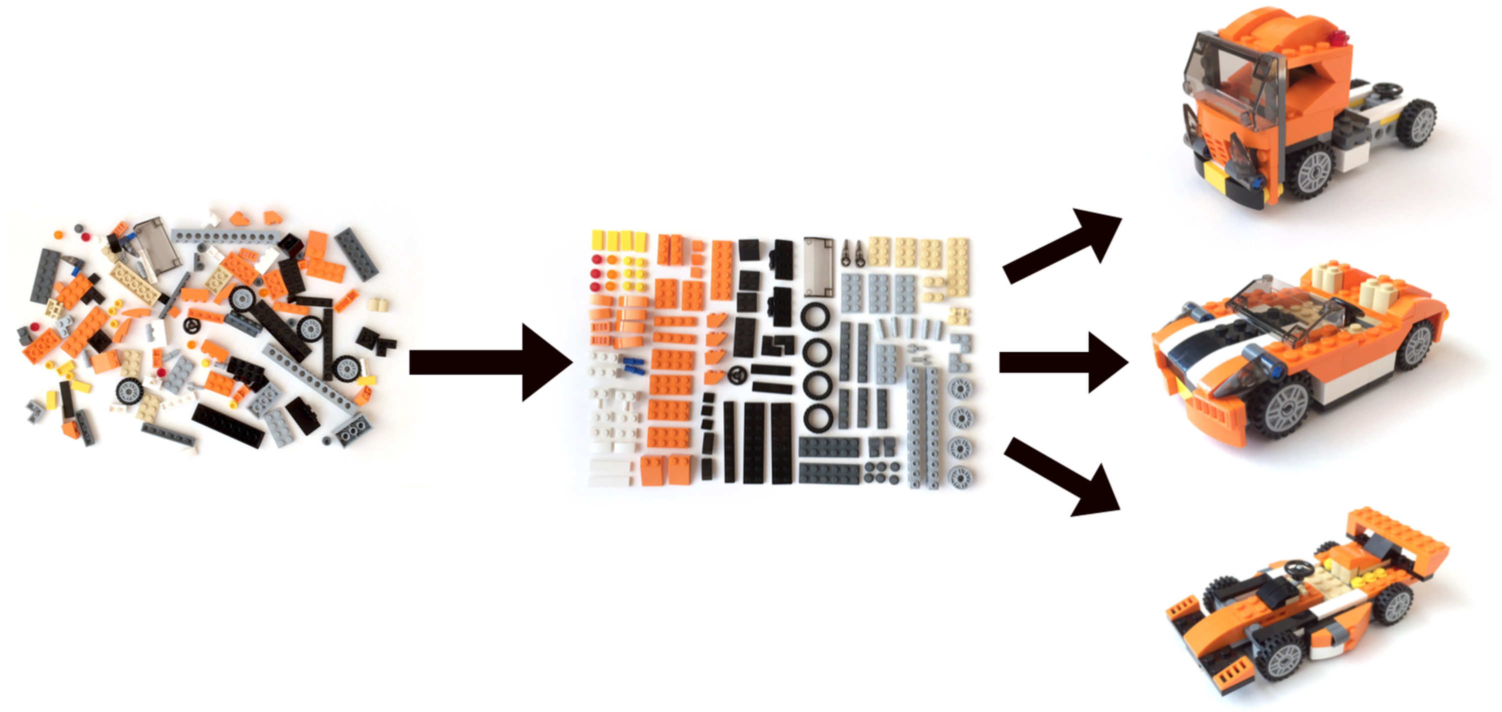
Credits: Oriol Vinyals, 2020.

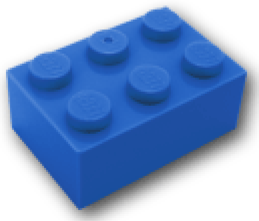
*People are now building a new kind of software by **assembling networks of parameterized functional blocks** and by **training them from examples using some form of gradient-based optimization**.*

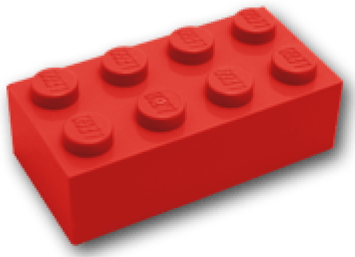
An increasingly large number of people are defining the networks procedurally in a data-dependent way (with loops and conditionals), allowing them to change dynamically as a function of the input data fed to them.

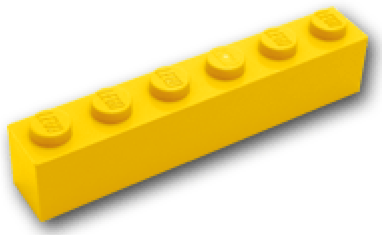
Yann LeCun, 2018.

DL as an architectural language




Inputs and outputs


Architectures

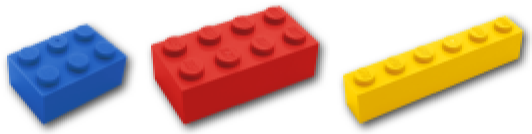

Losses

Inputs/Outputs

- Vectorized attributes, tabular data
- Images
- Sequences (words, speech, images, videos)
- Graphs
- Structured data

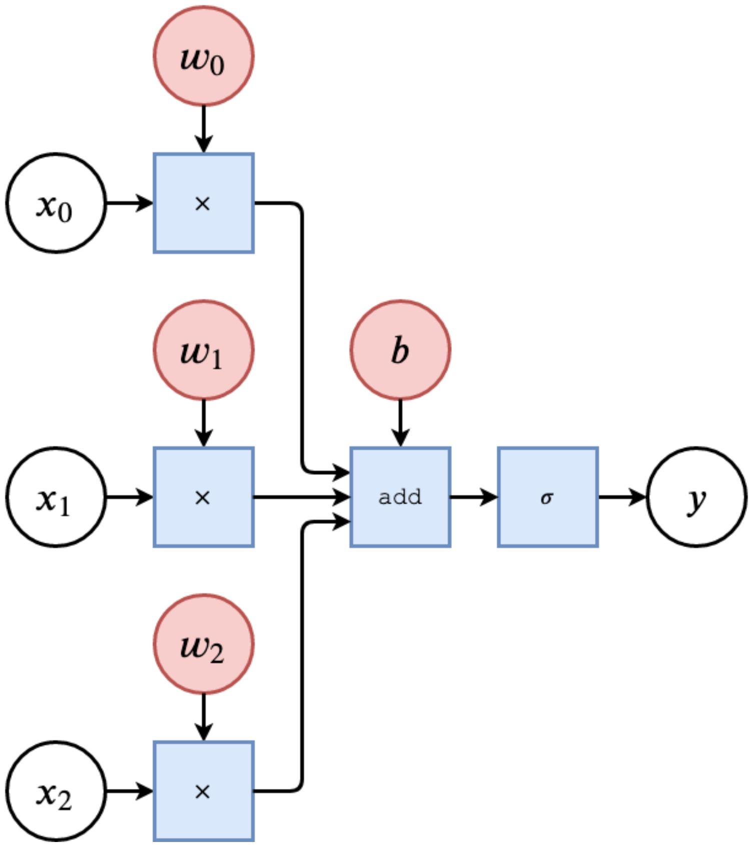
All may serve as both inputs and outputs.

Logistic regression



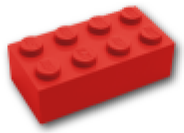
- I/O: vectorized inputs $\longrightarrow$ class labels
- Architecture: logistic unit
- Loss: cross-entropy

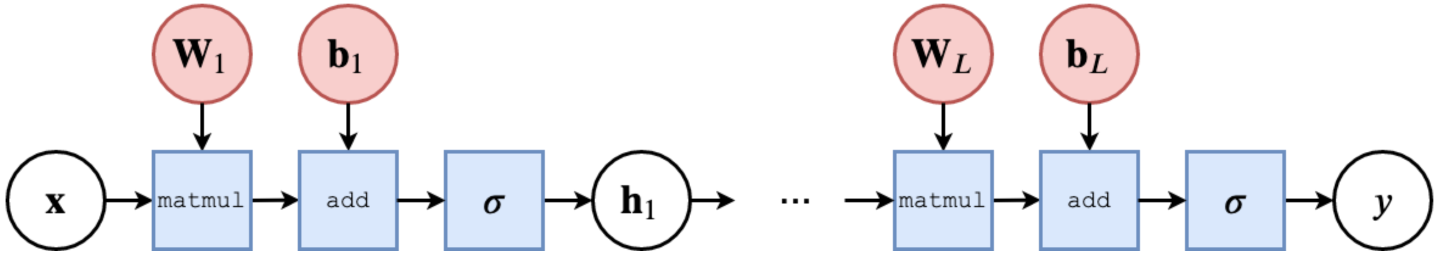
$$y = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

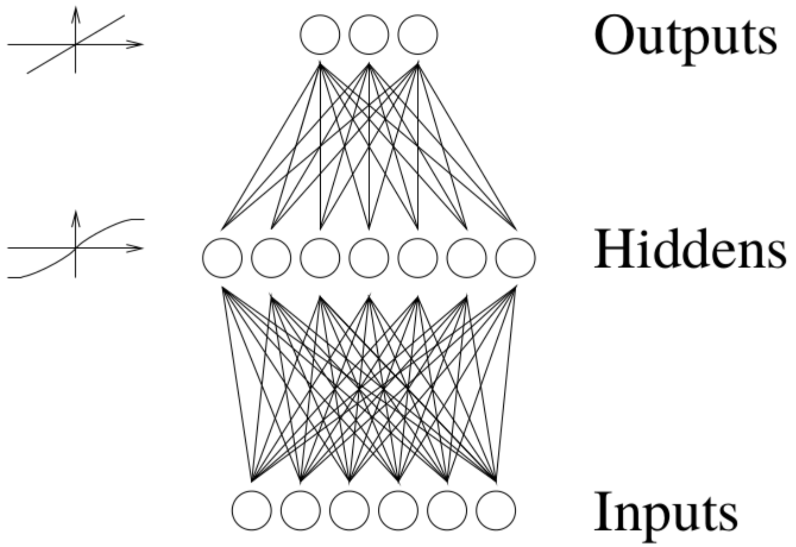


The logistic unit

Multi-layer perceptron



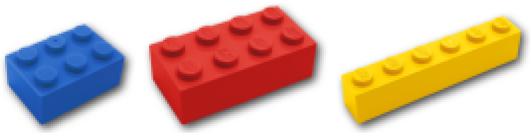




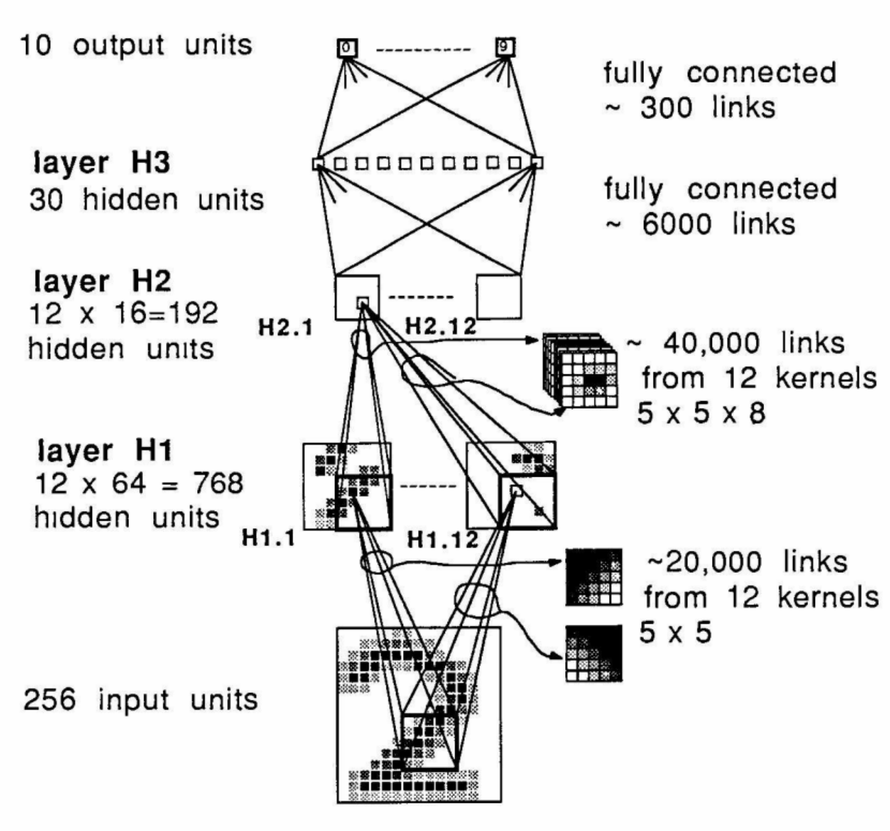
Stacking logistic units in parallel results in a layer.

Composing layers in series results in the MLP (or fully connected feedforwards nets).

Convolutional networks

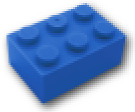


- I/O: images, audio, text $\longrightarrow$ class labels or regression targets
- Architecture: convolutions, pooling
- Loss: cross-entropy, negative log-likelihood



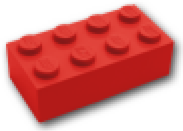
Convolutional networks extend fully connected nets with convolutional and pooling layers.

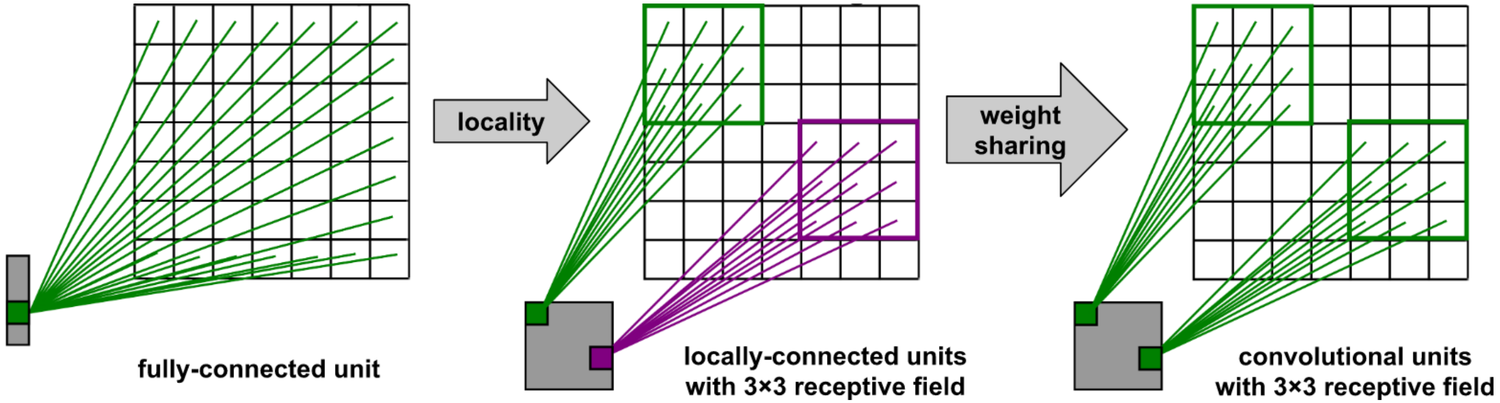
Locality and translation invariance



- Locality: objects tend to have a local spatial support.
- Translation invariance: object appearence is independent of location.

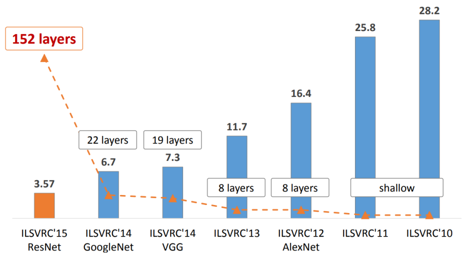
Convolution




locality
weight sharing
fully-connected unit
locally-connected units with 3×3 receptive field
convolutional units with 3×3 receptive field

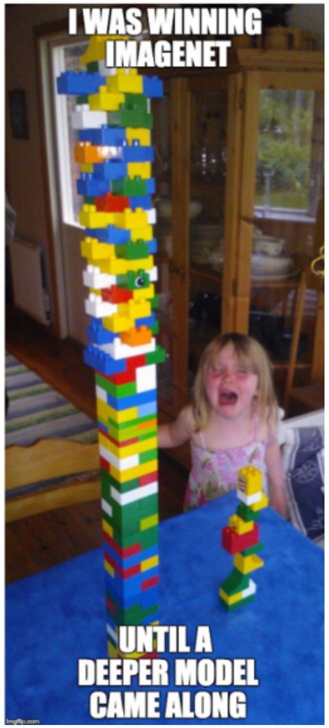
Credits: Reed, de Freitas, and Vinyals, "Deep Learning: Practice and Trends", 2017.

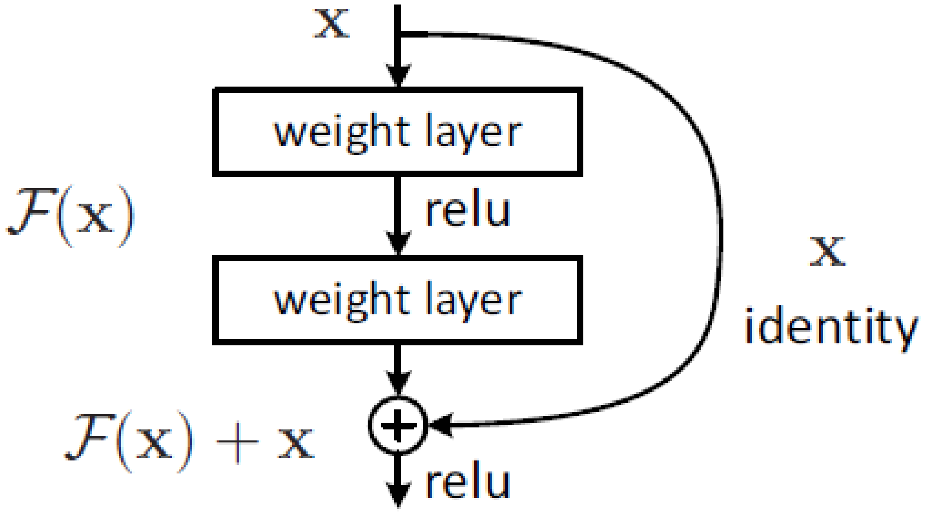
Skip connections



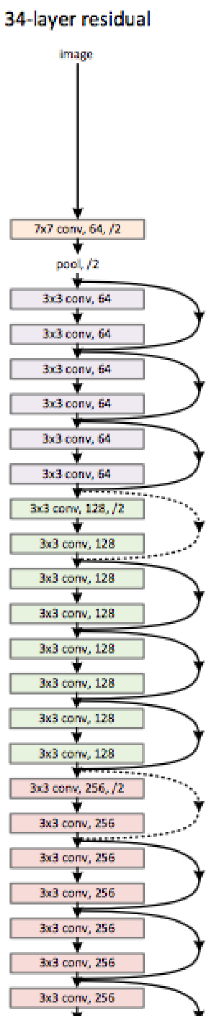
Training deep networks leads to vanishing gradients.

Skip connections provide a brick to make them trainable.

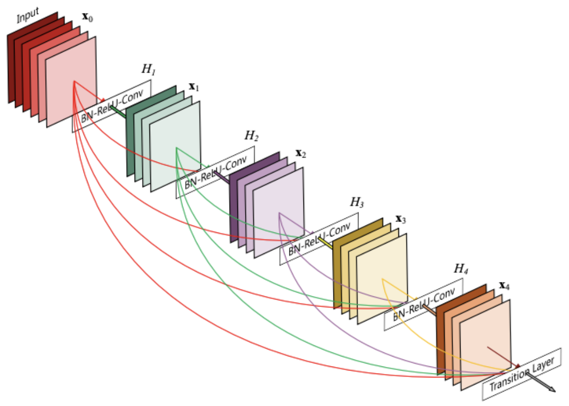


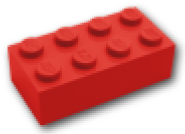


ResNet-34

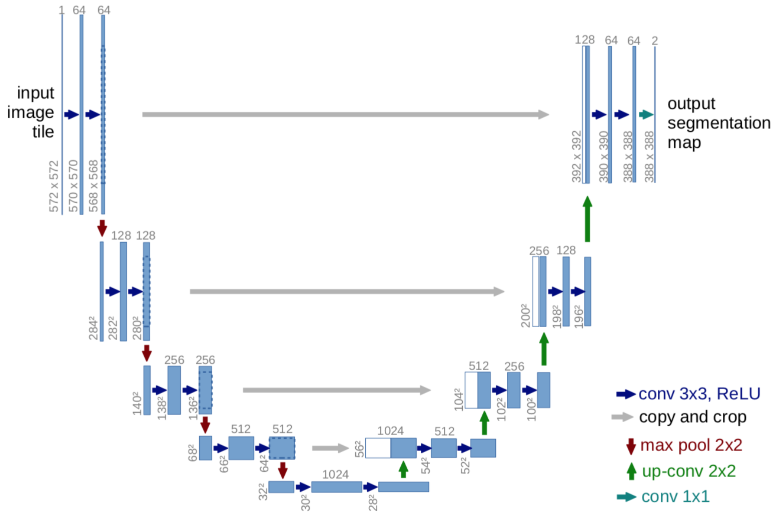


DenseNet

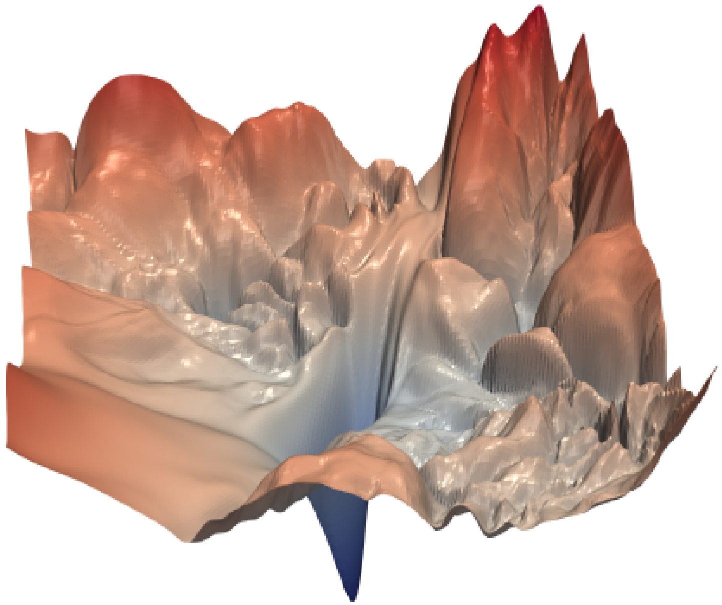




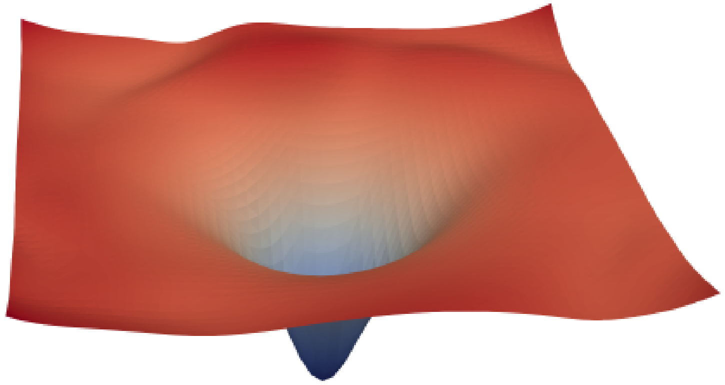
Unet



Credits: He et al, 2015, Huang et al, 2016, Ronneberger et al, 2015.



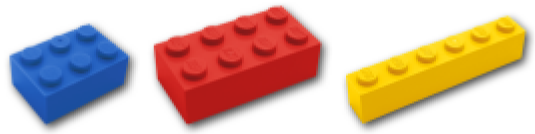
(a) without skip connections



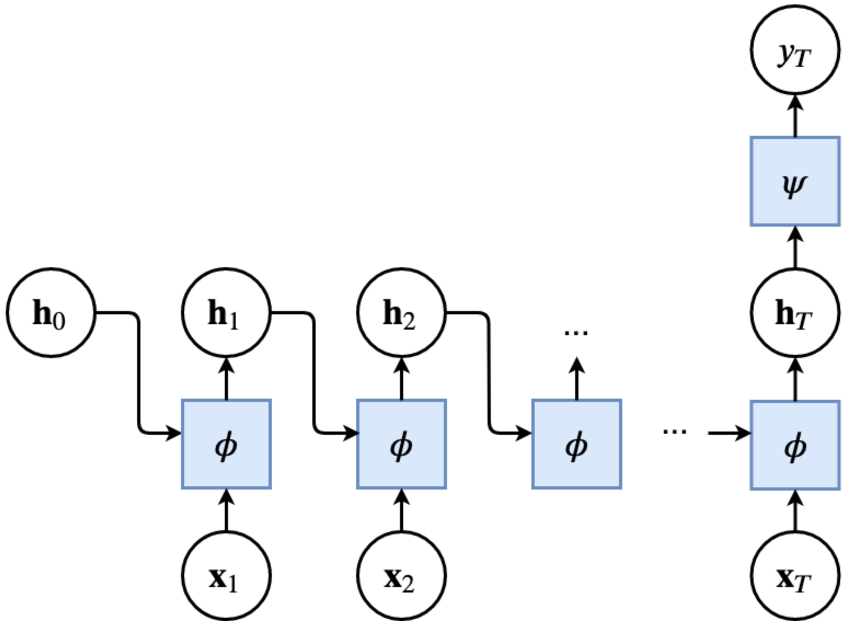
(b) with skip connections

Credits: Li et al, 2017.

Recurrent and attention networks

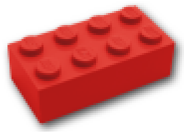


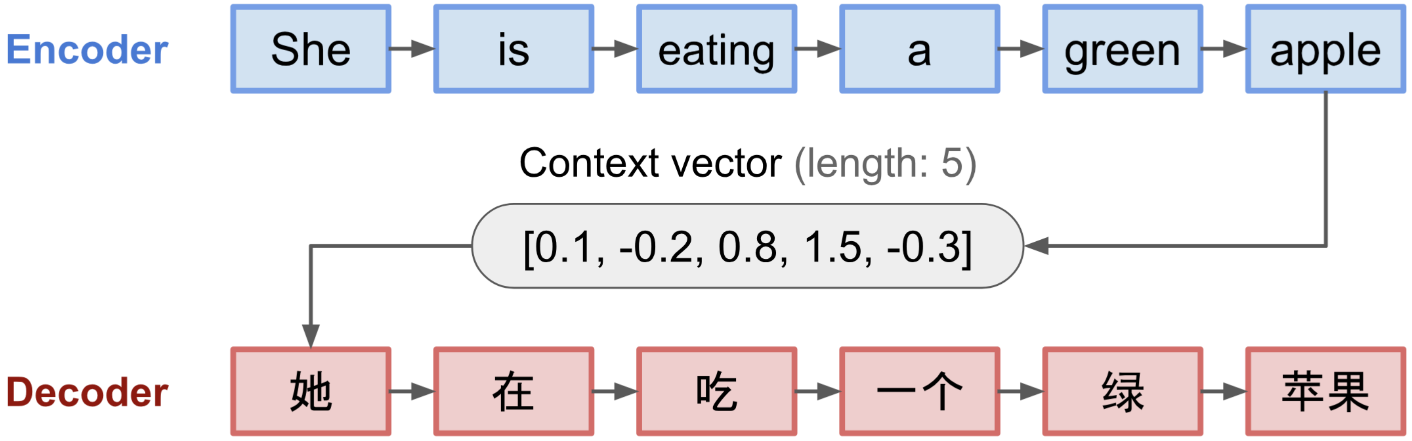
- I/O: sequences $\longrightarrow$ class labels, regression targets, or sequences
- Architecture: recurrent networks, attention
- Loss: cross-entropy, negative log-likelihood



RNNs are equipped with an internal state, which acts as memory cell. They are equivalent to feedforward nets but with shared layers.

Sequence-to-sequence



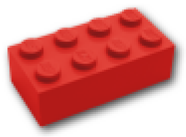


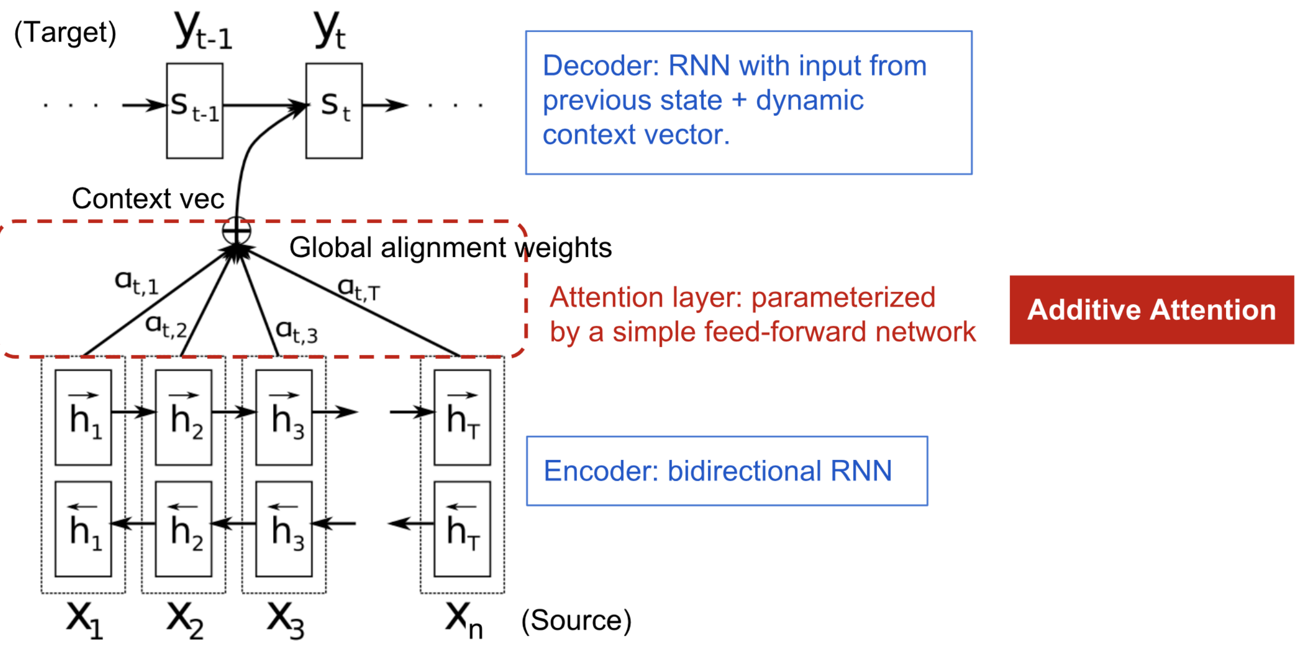
The seq2seq model is an encoder-decoder architecture composed of:

- an encoder RNN that processes the input sequence and compresses the information into a context vector $\mathbf{c}$ of fixed length;
- a decoder RNN initialized with the context vector that emits an output sequence.

Credits: Lilian Weng, 2018.

Attention



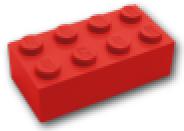

(Target)
y_{t-1}
y_t
s_{t-1}
s_t
Decoder: RNN with input from previous state + dynamic context vector.
Context vec
Global alignment weights
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
Attention layer: parameterized by a simple feed-forward network
Additive Attention
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
Encoder: bidirectional RNN
X_1 X_2 X_3 X_n
(Source)

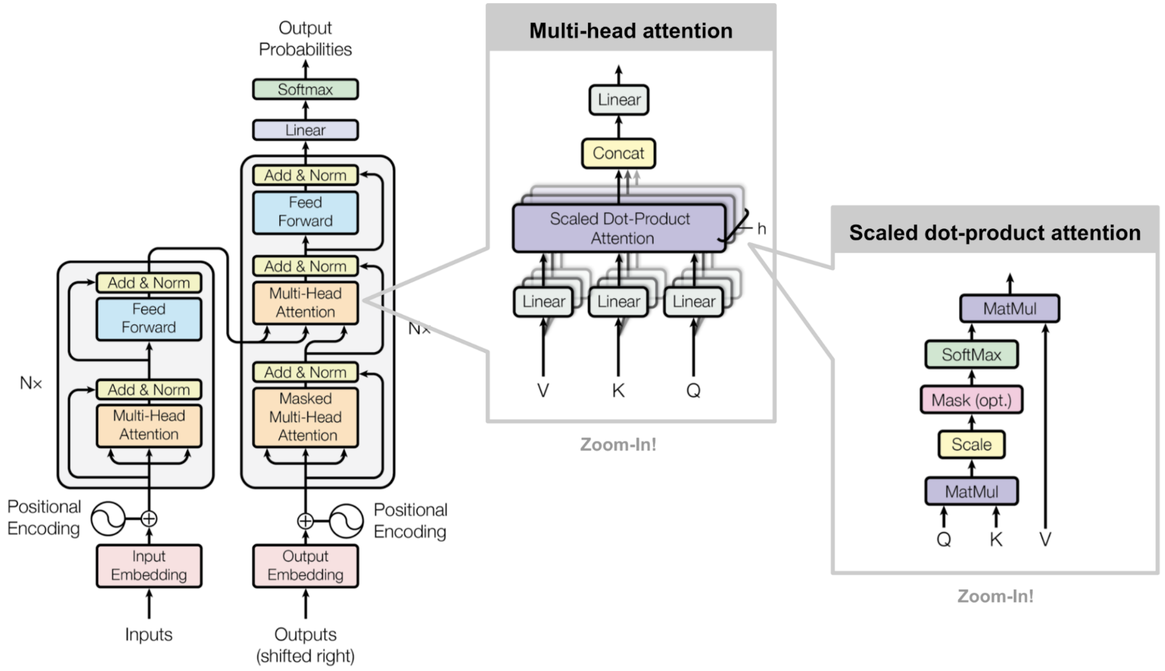
For infering the next state $\mathbf{s}_t$, the attention layer builds the dynamic context $\mathbf{c}_t$ which estimates how strongly the current state is correlated with (or attends to) other elements.

$$\mathbf{c}_t = \sum_i \alpha_{t,i}\mathbf{h}_i \qquad \alpha_{t,i} = \mathrm{softmax}(\mathrm{score}(\mathbf{s}_{t-1}, \mathbf{h}_i))$$

Credits: Bahdanau et al, 2015.

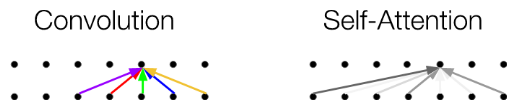
Transformers





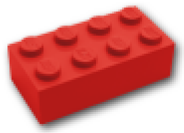
Transformer networks compose multi-head self-attention mechanisms for sequence-to-sequence modeling, without recurrent units.

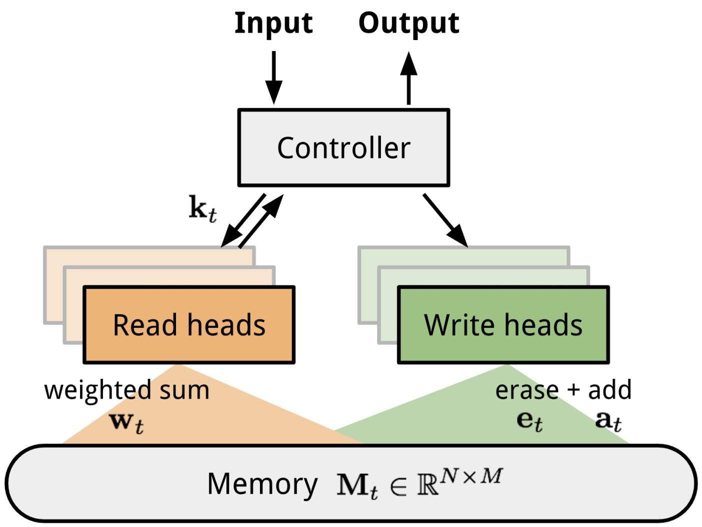
$$\mathrm{softmax}\left(\frac{QK^T}{\sqrt{n}}\right)V$$



Credits: Vaswani et al, 2017.

Neural computers

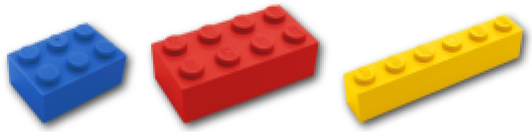




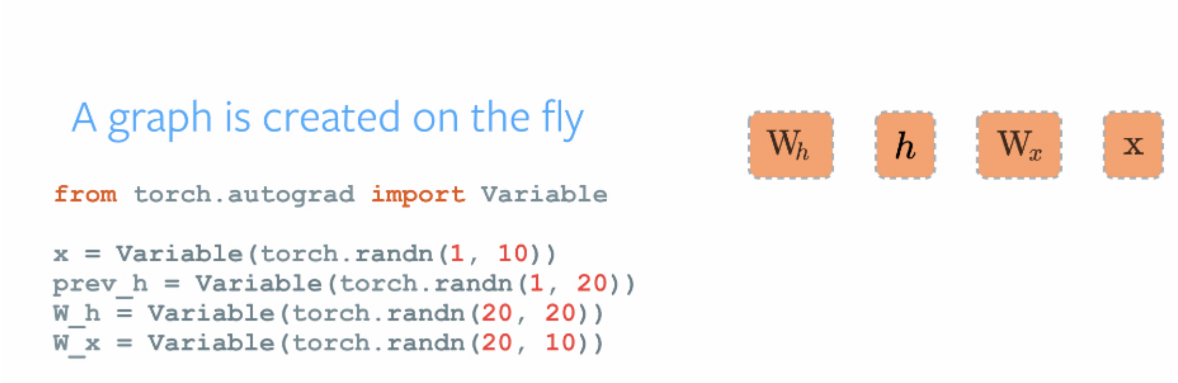
Networks can be coupled with memory storage to produce neural computers:

- The controller processes the input sequence and interacts with the memory to generate the output.
- The read and write operations attend to all the memory addresses.

Programs as neural nets



- I/O: structured data (one or more sources) $\longrightarrow$ class labels, regression targets or structured data
- Architecture: graph networks, dynamic graphs of computation
- Loss: cross-entropy, negative log-likelihood



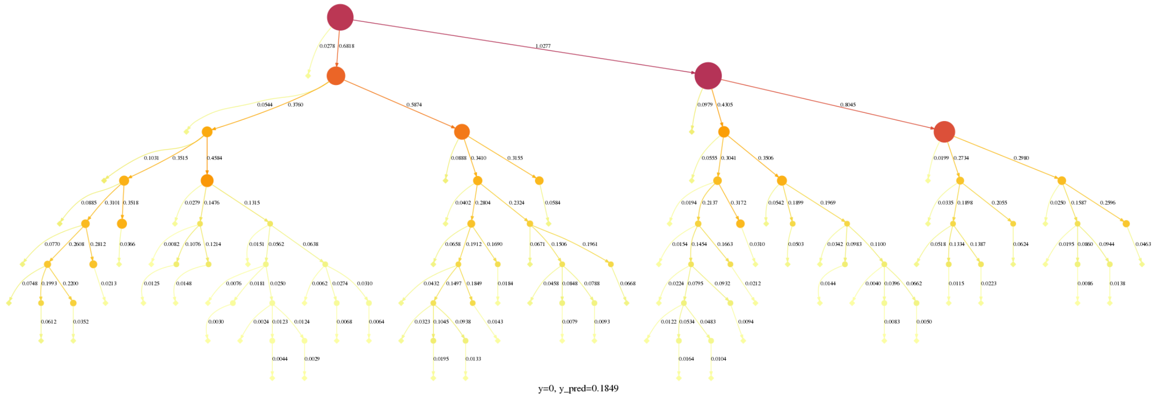
A graph is created on the fly

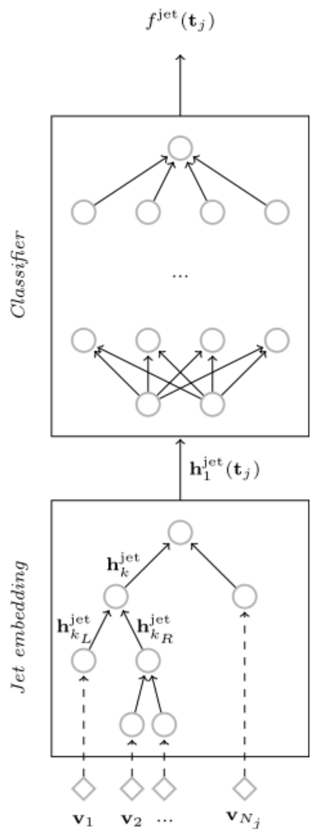
```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

Recursive nets for jet physics

- Use sequential recombination jet algorithms (kt, anti-kt, etc) to define computational graphs (on a per-jet basis).
- The root node in the graph provides a fixed-length embedding of a jet.

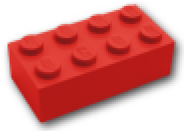


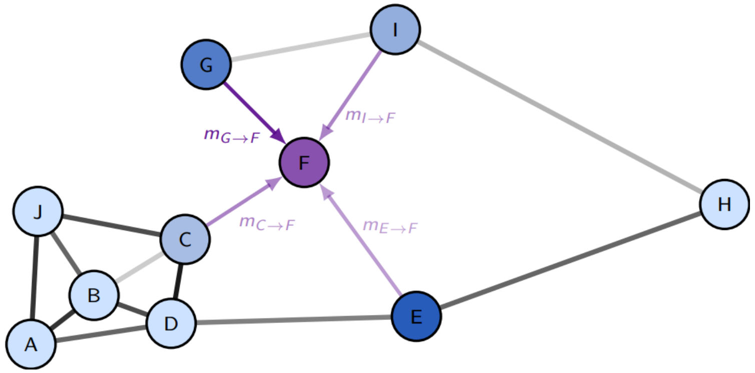


$f^{\text{jet}}(\mathbf{t}_j)$
Classifier
$\mathbf{h}_1^{\text{jet}}(\mathbf{t}_j)$
Jet embedding
$\mathbf{h}_k^{\text{jet}}$
$\mathbf{h}_{k_L}^{\text{jet}}$
$\mathbf{h}_{k_R}^{\text{jet}}$
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_{N_j}$

Credits: Louppe et al, 2017.

Neural message passing





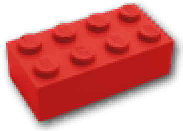
Algorithm 1 Message passing neural network

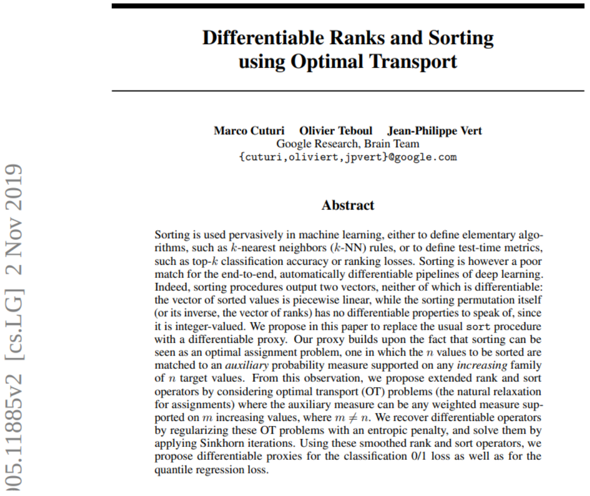
```
Require: $N \times D$ nodes $\mathbf{x}$, adjacency matrix $A$
  $\mathbf{h} \leftarrow$Embed$(\mathbf{x})$
  for $t = 1, \ldots, T$ do
      $\mathbf{m} \leftarrow$ Message$(A, \mathbf{h})$
      $\mathbf{h} \leftarrow$ VertexUpdate$(\mathbf{h}, \mathbf{m})$
  end for
  $\mathbf{r} =$ Readout$(\mathbf{h})$
  return Classify$(\mathbf{r})$
```

Even though the graph topology is dynamic, the unrolled computation is fully differentiable. The program is trainable.

Credits: Henrion et al, 2017.

Further examples of differentiable programs





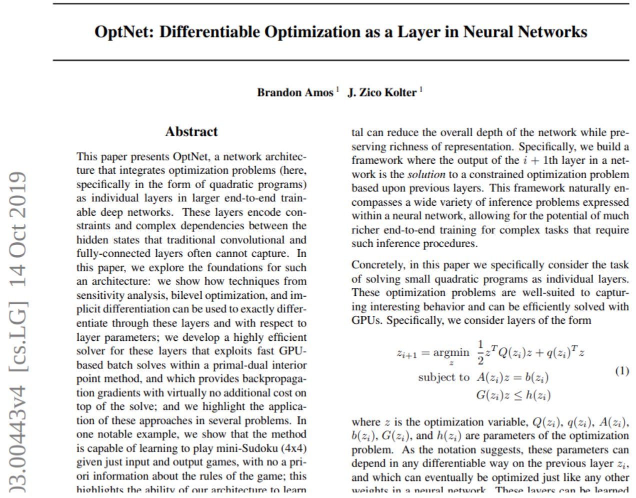
)05.11885v2 [cs.LG] 2 Nov 2019

Differentiable Ranks and Sorting using Optimal Transport

Marco Cuturi **Olivier Teboul** **Jean-Philippe Vert**
Google Research, Brain Team
`{cuturi,oliviert,jpvert}@google.com`

Abstract

Sorting is used pervasively in machine learning, either to define elementary algorithms, such as k-nearest neighbors (k-NN) rules, or to define test-time metrics, such as top-k classification accuracy or ranking losses. Sorting is however a poor match for the end-to-end, automatically differentiable pipelines of deep learning. Indeed, sorting procedures output two vectors, neither of which is differentiable: the vector of sorted values is piecewise linear, while the sorting permutation itself (or its inverse, the vector of ranks) has no differentiable properties to speak of, since it is integer-valued. We propose in this paper to replace the usual `sort` procedure with a differentiable proxy. Our proxy builds upon the fact that sorting can be seen as an optimal assignment problem, one in which the n values to be sorted are matched to an *auxiliary* probability measure supported on any *increasing* family of n target values. From this observation, we propose extended rank and sort operators by considering optimal transport (OT) problems (the natural relaxation for assignments) where the auxiliary measure can be any weighted measure supported on m increasing values, where $m \neq n$. We recover differentiable operators by regularizing these OT problems with an entropic penalty, and solve them by applying Sinkhorn iterations. Using these smoothed rank and sort operators, we propose differentiable proxies for the classification 0/1 loss as well as for the quantile regression loss.



)3.00443v4 [cs.LG] 14 Oct 2019

OptNet: Differentiable Optimization as a Layer in Neural Networks

Brandon Amos [1] **J. Zico Kolter** [1]

Abstract

This paper presents OptNet, a network architecture that integrates optimization problems (here, specifically in the form of quadratic programs) as individual layers in larger end-to-end trainable deep networks. These layers encode constraints and complex dependencies between the hidden states that traditional convolutional and fully-connected layers often cannot capture. In this paper, we explore the foundations for such an architecture: we show how techniques from sensitivity analysis, bilevel optimization, and implicit differentiation can be used to exactly differentiate through these layers and with respect to layer parameters; we develop a highly efficient solver for these layers that exploits fast GPU-based batch solves within a primal-dual interior point method, and which provides backpropagation gradients with virtually no additional cost on top of the solve; and we highlight the application of these approaches in several problems. In one notable example, we show that the method is capable of learning to play mini-Sudoku (4x4) given just input and output games, with no a priori information about the rules of the game; this highlights the ability of our architecture to learn

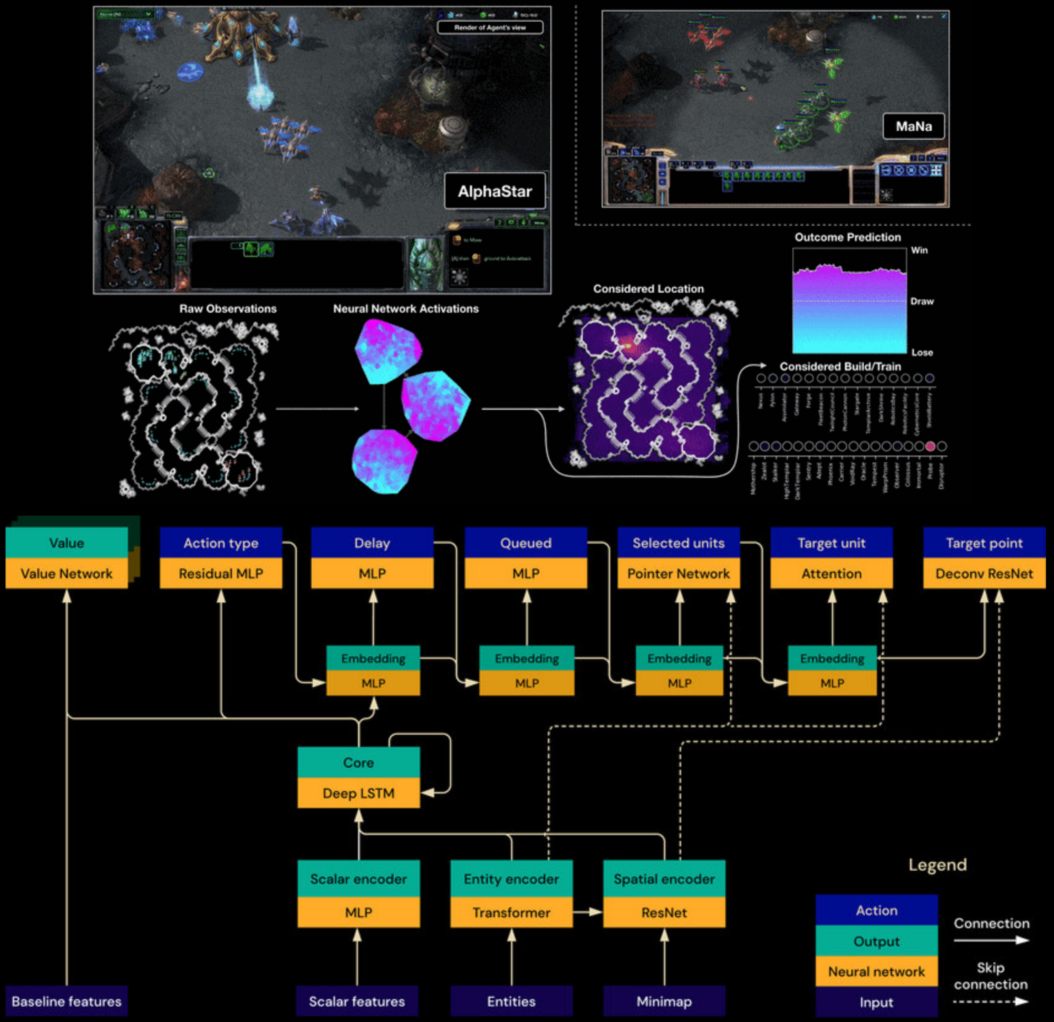
tal can reduce the overall depth of the network while preserving richness of representation. Specifically, we build a framework where the output of the $i + 1$th layer in a network is the *solution* to a constrained optimization problem based upon previous layers. This framework naturally encompasses a wide variety of inference problems expressed within a neural network, allowing for the potential of much richer end-to-end training for complex tasks that require such inference procedures.

Concretely, in this paper we specifically consider the task of solving small quadratic programs as individual layers. These optimization problems are well-suited to capturing interesting behavior and can be efficiently solved with GPUs. Specifically, we consider layers of the form

$$\begin{aligned} z_{i+1} = \underset{z}{\operatorname{argmin}} \quad & \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z \\ \text{subject to} \quad & A(z_i) z = b(z_i) \\ & G(z_i) z \leq h(z_i) \end{aligned} \tag{1}$$

where z is the optimization variable, $Q(z_i)$, $q(z_i)$, $A(z_i)$, $b(z_i)$, $G(z_i)$, and $h(z_i)$ are parameters of the optimization problem. As the notation suggests, these parameters can depend in any differentiable way on the previous layer z_i, and which can eventually be optimized just like any other weights in a neural network. These layers can be learned

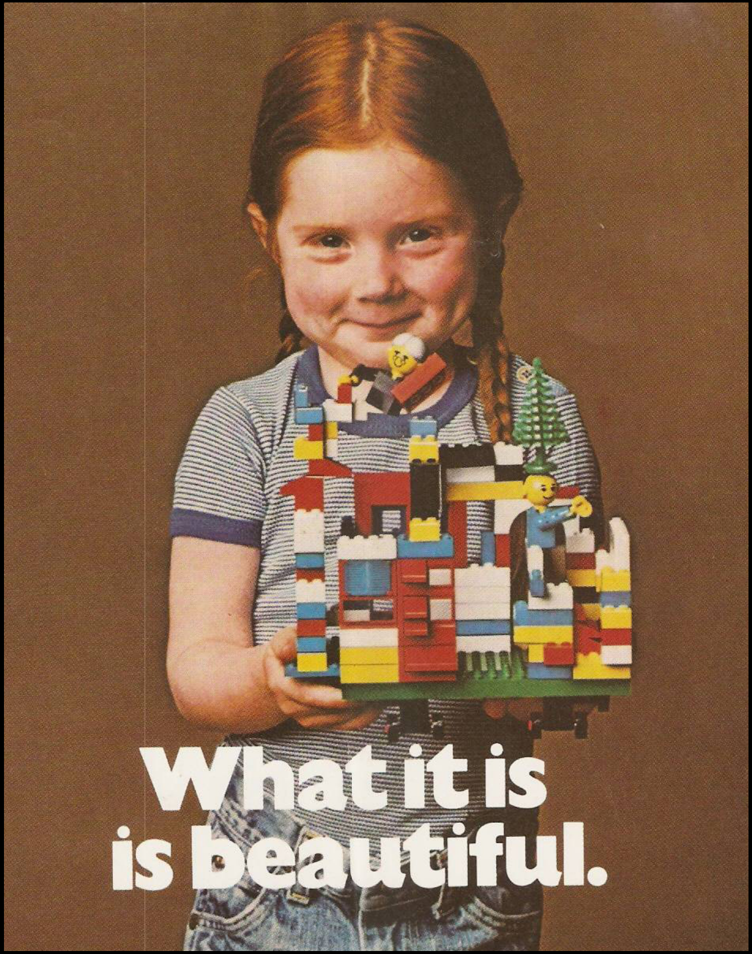
LEGO® Creator Expert



Credits: Vinyals et al, 2019.

Conclusions

- Deep Learning is more than feedforward networks.
- It is a **methodology**:
  - assemble networks of parameterized functional blocks
  - train them from examples using some form of gradient-based optimisation.
- Bricks are simple, but their nested composition can be arbitrarily complicated.
- The Large Creative Brick Box includes many more bricks!



Credits

This talk is strongly inspired from "Deep Learning: Practice and Trends" (NIPS 2017 Tutorial) by Scott Reed, Nando de Freitas and Oriol Vinyals.

References

- Scott Reed, Nando de Freitas and Oriol Vynials, "Deep Learning: Practice and Trends", 2017. [url]
- Gilles Louppe, "INFO8010 Deep Learning, ULiège", 2018-2019. [url]
- Lilian Weng, "Attention? Attention!", 2018. [url]
- Jay Alammar, "The Illustrated Transformer", 2018. [url]