Deep Learning

Jan Koutník

IDSIA
USI & SUPSI

July 18, 2016

Istituto Dalle Molle di studi sull'intelligenza artificiale (IDSIA)

- Swiss AI Institute in Lugano
- Jürgen Schmidhuber & Luca Gambardella
- Long Short-Term Memory (LSTM; 1997)
- Universal Optimal AI Theory (AIXI; 2001)
- Deep Convolutional Neural Networks on GPUs (2009)
- Natural Evolution Strategy (NES; 2010)
- Fast Scanning Convolutional NNs (2011)
- Highway Networks (2014)
- **http://www.idsia.ch**

What is Deep Learning?

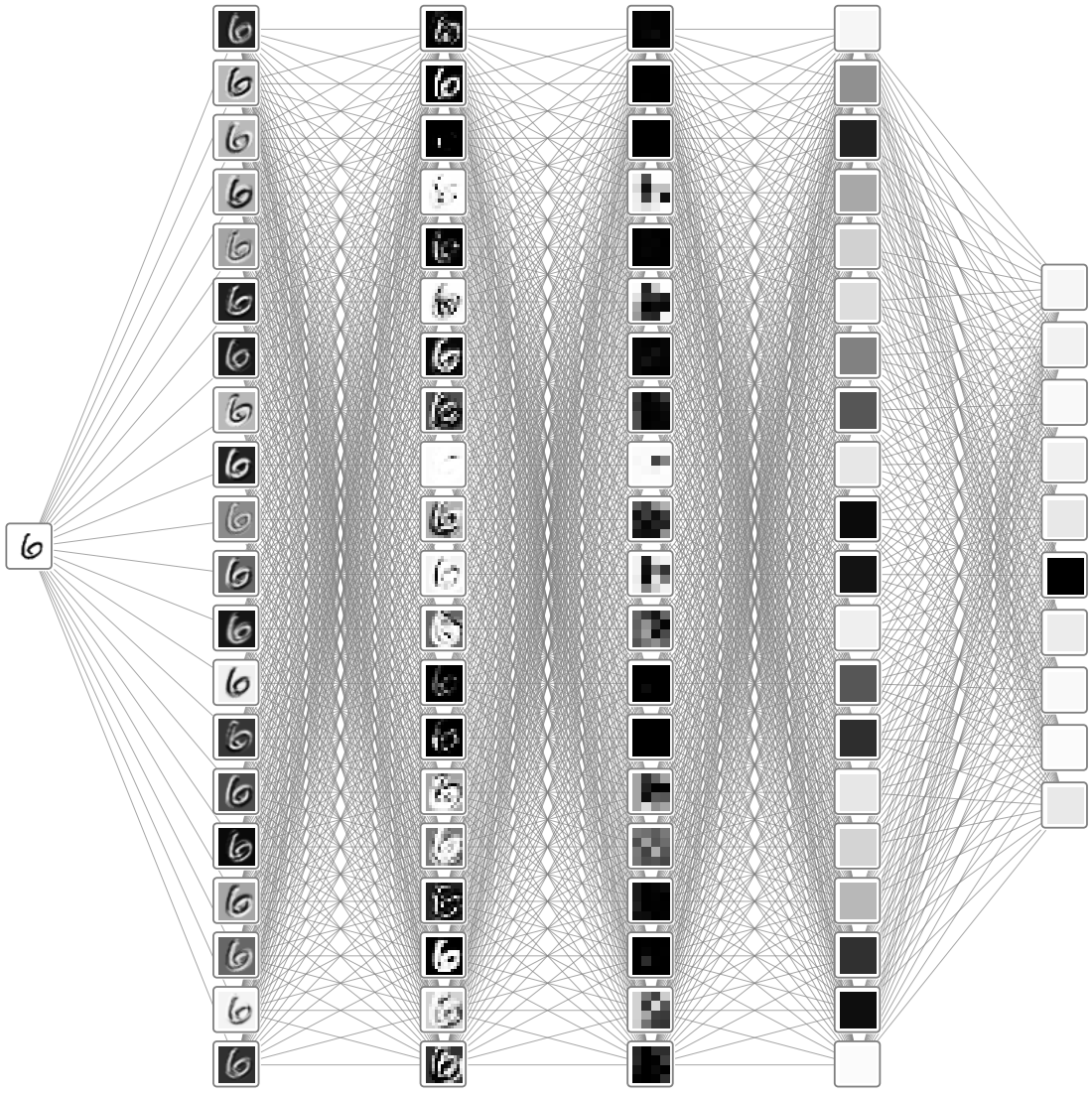
- A fancy *marketing* name for Artificial Neural Networks (NN)
- Namely Convolutional Neural Networks used for supervised perception tasks
- A few centuries old term
- Brought to machine learning in 1986[1]
- Used in NN context in 1999[2]
- The depth can be measured by *credit assignment path* length in NN
- ... number of layers

---

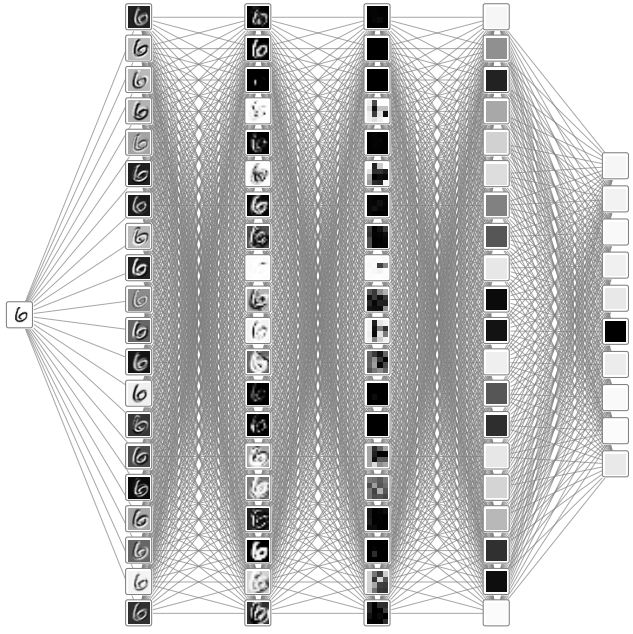
[1] R. Dechter, "Learning while searching in constraint-satisfaction-problems," in *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, T. Kehler, Ed., Morgan Kaufmann, 1986, pp. 178–185. [Online]. Available: http://www.aaai.org/Library/AAAI/1986/aaai86-029.php.

[2] I. N. Aizenberg, "Neural networks based on multi-valued and universal binary neurons: theory, application to image processing and recognition," in *Computational Intelligence, Theory and Applications, International Conference, 6th Fuzzy Days, Dortmund, Germany, May 25-28, 1999, Proceedings*, B. Reusch, Ed., ser. Lecture Notes in Computer Science, vol. 1625, Springer, 1999, pp. 306–316, ISBN: 3-540-66050-X. DOI: 10.1007/3-540-48774-3_36. [Online]. Available: http://dx.doi.org/10.1007/3-540-48774-3_36.

Convolutional Neural Networks



Convolutional Neural Networks



- Used for image classification (typical usage)
- Input filtered using convolutions in order to provide features
- Downsample the image in order to reduce dimension
- Features classified by a classifier (e.g. neural network)
- Features are not hand-crafted but trained from the data

- Deep Maxpooling CNNs, MLP on top
- 1 year on CPU = 1 week on GPU
- 5×10^9 weight updates/s
- First human-competitive result (0.2% error)[3]
- After a decade of 0.4% error

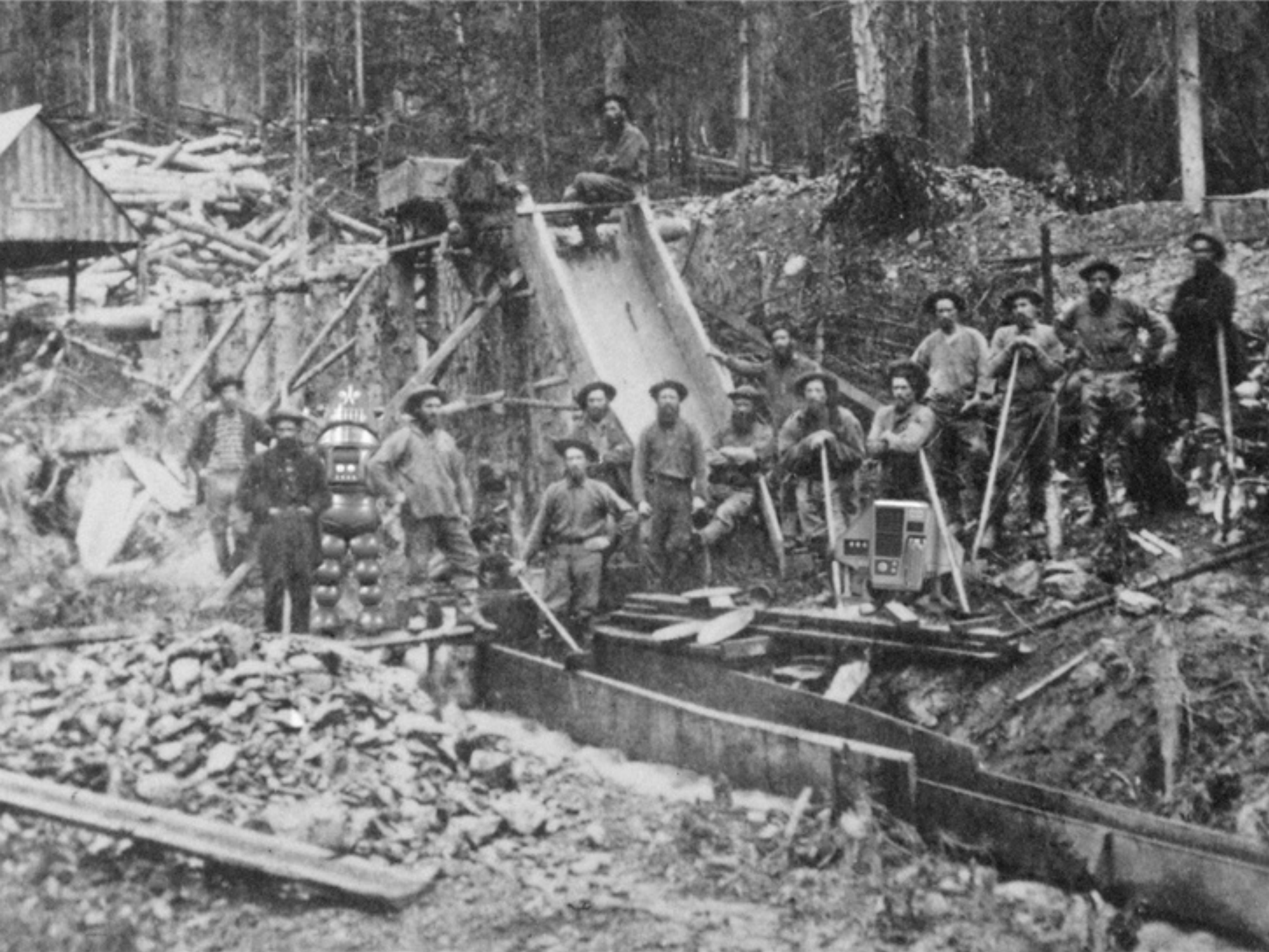
[3] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition CVPR 2012*, Long preprint arXiv:1202.2745v1 [cs.CV], 2012.

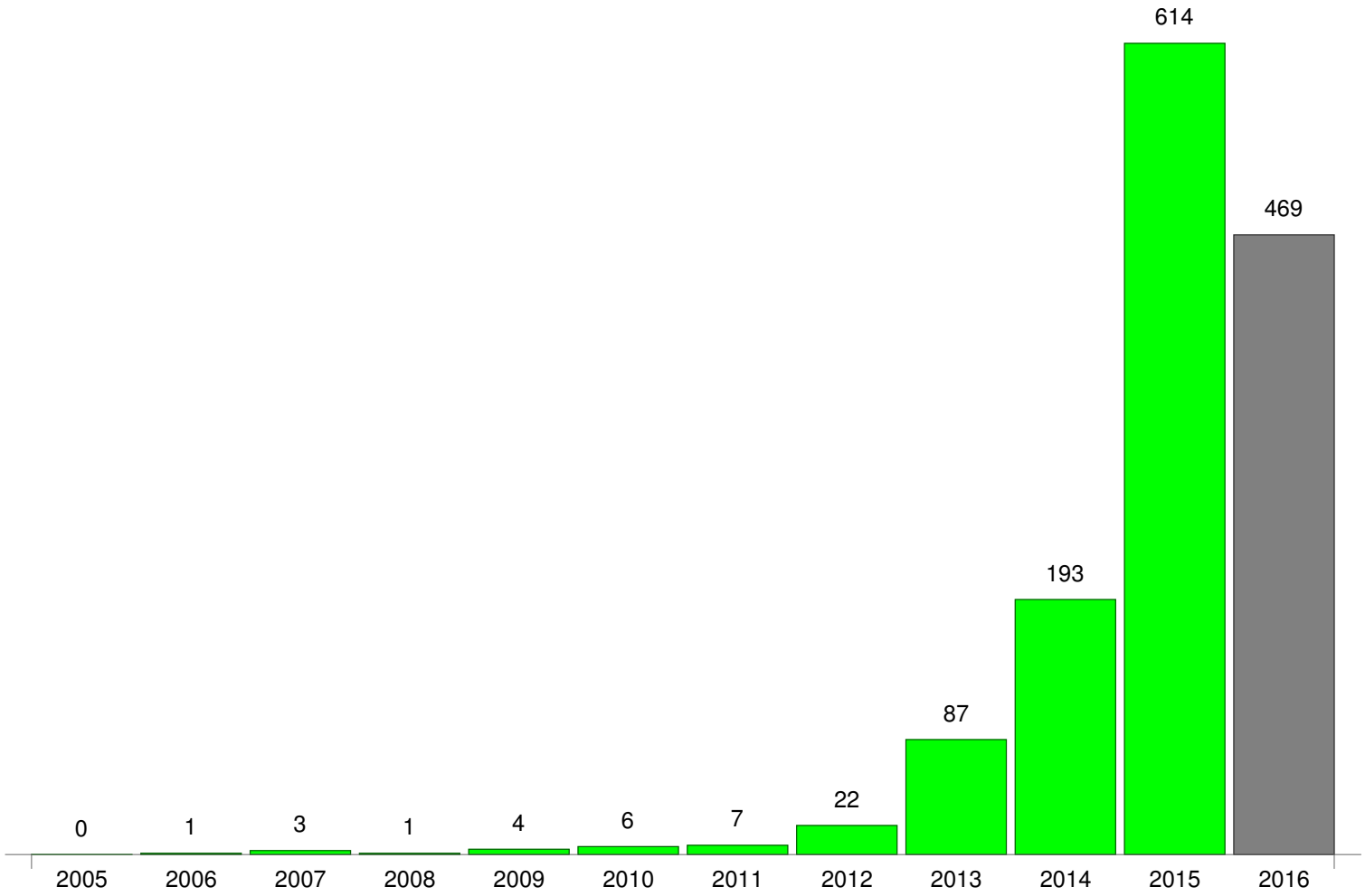
- IJCNN 2011, on site traffic sign recognition
- 1^{st} 0.56%[4,5]
- 2^{nd} 1.16% (Humans)
- 3^{rd} 1.69%
- 4^{th} 3.86%

[4] D. C. Ciresan, U. Meier, J. Masci, *et al.*, "A committee of neural networks for traffic sign classification," in *International Joint Conference on Neural Networks (IJCNN)*, 2011, pp. 1918–1921.

[5] D. C. Ciresan, U. Meier, J. Masci, *et al.*, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, vol. 32, pp. 333–338, 2012.

since 2012 ...





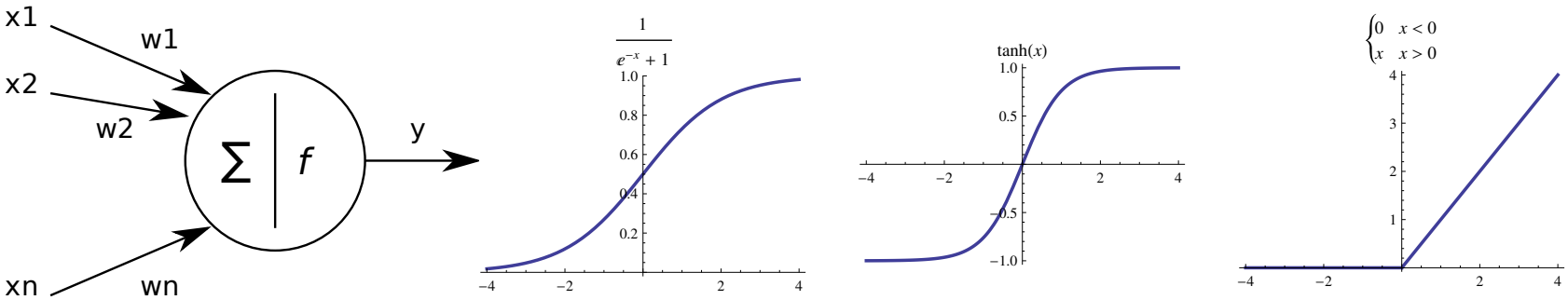
- Search for "Deep Learning" in abstracts on ArXiv[6]

[6] http://arxiv.org

- Major industries build their Deep Learning teams
- Deep Learning fills news papers
- Bold claims of Deep Learning solving the general AI problem
- News catching up on such claims
- 1B$ invested in 2014 in Deep learning
- ... but

- It is mostly solving perception problems
- Let's take a closer look

Artificial Neural Networks



- Originally a Linear Thresholding Unit[7]
- $y = f\sum_{i=1}^{N} w_i x_i = f(\mathbf{wx})$
- f is a non-linear function
- f in LTU is was a unit step

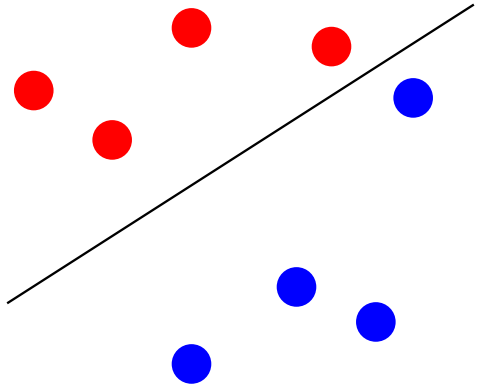
[7] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 7, pp. 115–133, 1943.

Perceptron

- In 1958 developed by Frank Rosenblatt in IBM[8]
- Simulated on IBM 704 (first commercial FPU)
- Supervised learning binary classifier
  $w_i(t+1) = w_i(t) + \alpha(z_j - y_j(t))x_{j,i}$
  for input i and j-th target z_j
- Step non-linearity
- Mark 1 Perceptron – custom-built hardware

[8] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

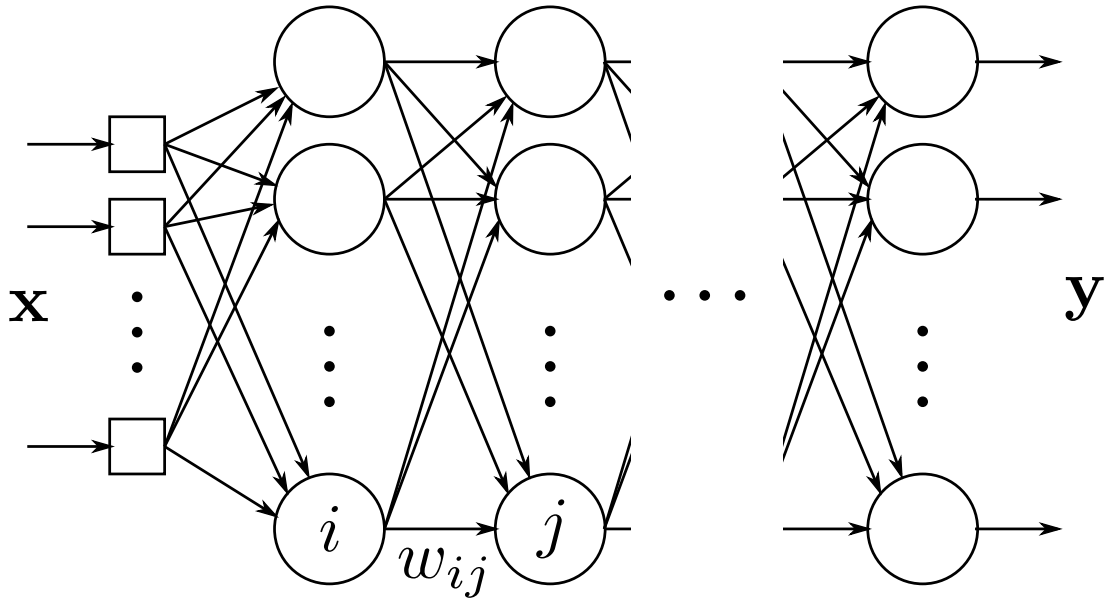
Linear Separability Problem



- Perceptrons criticized in 1969[9]
- Discouraged researchers from NNs for a decade

[9] M. Minsky and S. Papert, *Perceptrons*. Cambridge, MA: MIT Press, 1969.

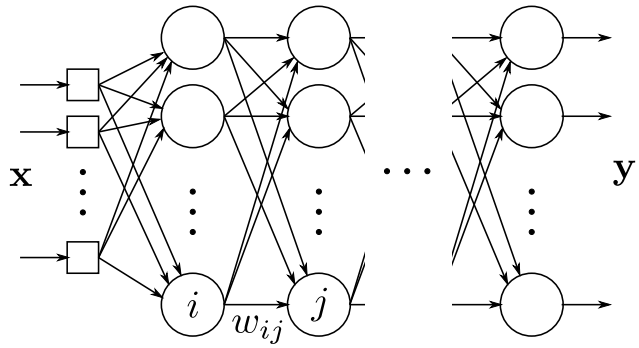
Multi Layer Perceptron



- Multiple layer can solve non-linearly separable problems[10]
- How?

[10] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, 1981, pp. 762–770.

Supervised Learning



- Each connection has a weight w
- Training set consist of N pairs: input $\mathbf{x}_n$, output $\mathbf{z}_n$
- MLP is a non-linear transform function of $F(\mathbf{x}_n) \rightarrow \mathbf{y}_n$
- Training changes F in order to match all $\mathbf{y}_n$ to $\mathbf{z}_n$ with a minimal error E

- Optimization problem
- Gradient is defined (how?)
- Perform gradient descent

Gradient Descent

- Define objective function:
  $Q(w) = \sum_{i=1}^{N} Q_i(w)$, for N training examples
- Optimize weights by gradient descent:
  $$w(t+1) = w(t) - \alpha \nabla Q(w(t)) = w(t) - \alpha \sum_{i=1}^{N} \nabla Q_i(w(t))$$
- Stochastic gradient descent (SGD):
  $w(t+1) = w(t) - \alpha \nabla Q_i(w(t))$
- Alternatively, use smaller batch:
  $$w(t+1) = w(t) - \alpha \sum_{i=1}^{B} \nabla Q_i(w(t)),\ B < N$$

Gradient Descent

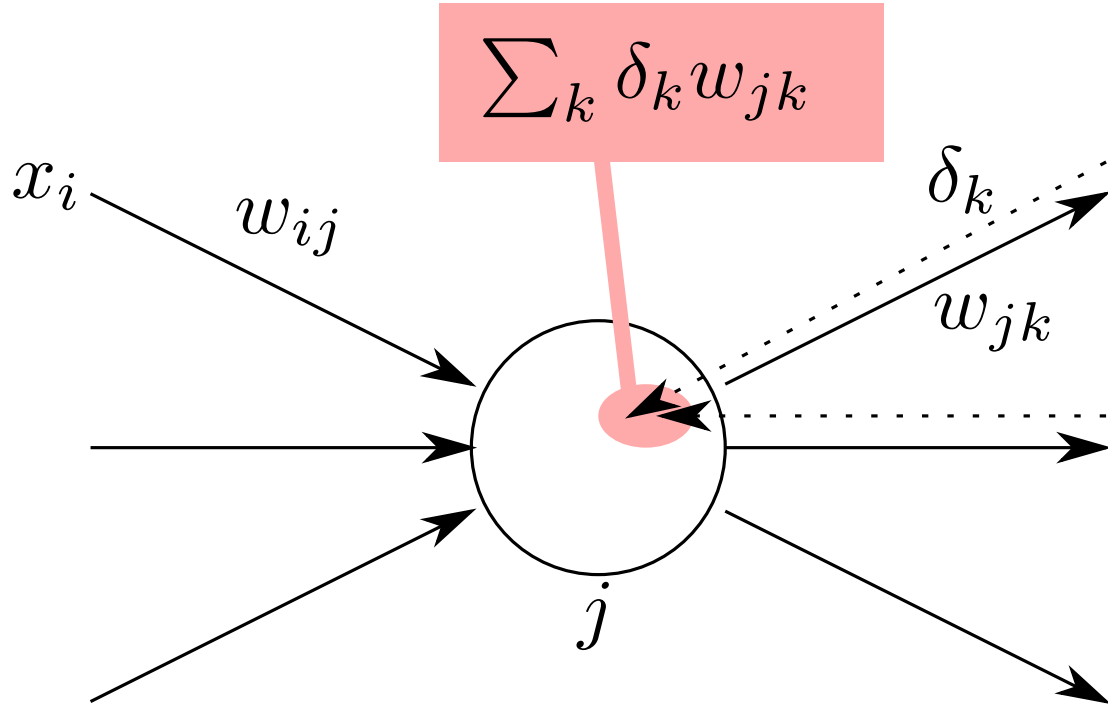
- Minimize error of N output neurons: $E = \sum_j \frac{1}{2}(z_j - y_z)^2$
- Recall $y = f(h)$, $h = \mathbf{wx}$
- Calculate partial with respect to each weight $\frac{\partial E}{\partial w_{ij}}$
- For a single output j and input weight w_i:
  $\frac{\partial E}{\partial w_i} = \frac{\partial(\frac{1}{2}(z-y)^2)}{\partial w_i}$
- Use *chain rule*: $= \frac{\partial(\frac{1}{2}(z-y)^2)}{\partial y}\frac{\partial y}{\partial w_i}$

$$= -(z-y)\frac{\partial y}{\partial h}\frac{\partial h}{\partial w_i}$$
$$= -(z-y)f'(h)\frac{\partial \mathbf{xw}}{\partial w_i}$$
$$= -(z-y)f'(h)x_i$$

- Error gradient: $\frac{\partial E}{\partial w_i} = -(z - y) f'(h) x_i$
- Change weight by a learning rate α: $\Delta wi = \alpha \underbrace{(z - y)}_{\delta} f'(h) x_i$
- called Delta Rule[11]

[11] B. Widrow and M. Hoff, "Associative storage and retrieval of digital information in networks of adaptive neurons," *Biological Prototypes and Synthetic Systems*, vol. 1, p. 160, 1962.

Multiple Layers



- Each connection weight participates proportionally on the error
- Propagate and sum δ backwards from output nodes k
- $\Delta w_{ij} = \alpha x_i \delta_j = \alpha x_i f'(h_j) \sum_k \delta_k w_{jk}$

MLP Error Backpropagation (BP)

1. Forward pass: compute $\mathbf{h}_{\mathscr{L}} = \mathbf{w}_{\mathscr{L}}\mathbf{x}_{\mathscr{L}}$ and $\mathbf{y}_{\mathscr{L}} = f(\mathbf{h}_{\mathscr{L}})$ for each layer $\mathscr{L} = 1, 2, 3, \ldots, L$
2. Inject top layer errors $(\boldsymbol{\delta}_L)$
3. Backward pass: compute $\boldsymbol{\delta}_{\mathscr{L}} = \mathbf{w}_{\mathscr{L}}^{\mathsf{T}}\boldsymbol{\delta}_{\mathscr{L}}$ for each layer $\mathscr{L} = L, L-1, L-2, \ldots, \mathscr{L}$
4. Update weights: $\mathbf{w}_{\mathscr{L}}(t+1) = \mathbf{w}_{\mathscr{L}} + \alpha(\mathbf{x}_{\mathscr{L}-\mathbf{1}} \otimes \boldsymbol{\delta}_{\mathscr{L}} f'(\mathbf{h}_{\mathscr{L}}))$ of all layers $\mathscr{L}, \ldots, L$

Note: biases omitted for clarity.

Backpropagation in Neural Networks

- Euler-LaGrange calculus + dynamic programming (1962)[12]
- Simplified version using chain rule (1962)[13]
- BP in sparse, discrete graphs (1970)[14], first MLP-like BP
- Automatic differentiation (1980)[15]
- BP for MLP (1981)[16,17]
- BP for *recurrent* NNs (1986)[18]

---

[12] A. E. Bryson Jr., "A steepest ascent method for solving optimum programming problems," *Journal of Applied Mechanics*, vol. 29, no. 2, p. 247, 1962.

[13] S. E. Dreyfus, "The numerical solution of variational problems," *Journal of Mathematical Analysis and Applications*, vol. 5(1), pp. 30–45, 1962.

[14] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors," Master's thesis, Univ. Helsinki, 1970.

[15] B. Speelpenning, "Compiling fast partial derivatives of functions given by algorithms," PhD thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, Jan. 1980.

[16] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, 1981, pp. 762–770.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds., vol. 1, MIT Press, 1986, pp. 318–362.

[18] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," in *Parallel Distributed Processing*, MIT Press, 1986, pp. 151–193.

How Many Layers?

- Recall: a neural network is a non-linear function F of X variables
- Hilbert's 13^{th} problem:
  $x^7 + ax^3 + bx^2 + cx + 1 = 0$
- "Its solution x (function of variables a, b, c) cannot be expressed as a superposition of functions of two variables?"
- Disprooved by Arnold and Kolmogorov (1954)
- More general and NN-relevant result from 1957:
- Kolmogorov Superposition Theorem[19]

[19] A. K. Kolmogorov, "On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition," *Doklady Akademii Nauk SSSR*, vol. 114, pp. 369–373, 1957.

Kolmogorov Superposition Theorem

For each $n \geq 2$ there exist continuous functions $\phi_q : [0,1] \to \mathbb{R}$, $q = 0, \ldots, 2n$ and constants $\lambda_p \in \mathbb{R}$, $p = 1, \ldots, n$ such that the following holds true: for each continuous function $f : [0,1]^n \to \mathbb{R}$ there exists a continuous function $g : [0,1] \to \mathbb{R}$ such that

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} g \left(\sum_{p=1}^{n} \lambda_p \phi_q(x_p) \right)$$

- Relation to NN criticized, because the functions g and ϕ are fractal and the number grows exponentially[20]
- Polynomial bounds derived[21]

---

[20] F. Girosi and T. A. Poggio, "Representation properties of networks: kolmogorov's theorem is irrelevant," *Neural Computation*, vol. 1, no. 4, pp. 465–469, 1989. DOI: 10.1162/neco.1989.1.4.465. [Online]. Available: http://dx.doi.org/10.1162/neco.1989.1.4.465.

[21] V. Kurková, "Kolmogorov's theorem is relevant," *Neural Computation*, vol. 3, no. 4, pp. 617–622, 1991. DOI: 10.1162/neco.1991.3.4.617. [Online]. Available: http://dx.doi.org/10.1162/neco.1991.3.4.617.

Universal Approximation Theorem

"A feedforward network with a single hidden layer containing finite number of neurons can approximate continuous functions on a compact subset of $\mathbb{R}^n$"

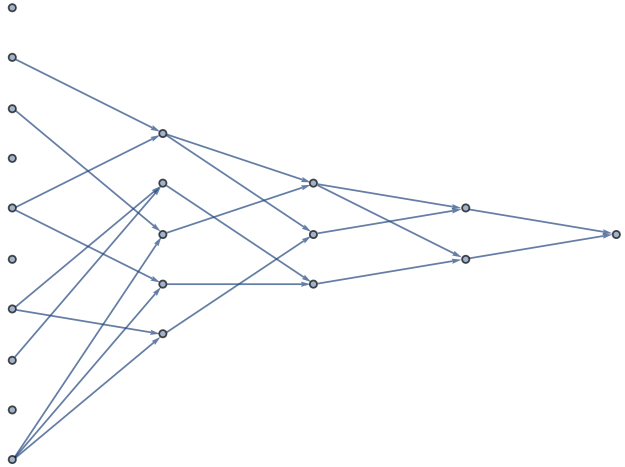
- Proof by Cybenko for sigmoid activation functions[22]
- Sigmoid activations are not required[23]

[22] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, 1989.

[23] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. DOI: 10.1016/0893-6080(91)90009-T. [Online]. Available: http://dx.doi.org/10.1016/0893-6080(91)90009-T.

Why Deep Learning then?

First Deep Learning



- Group Method of Data Handling (GMDH)[24,25]
- Deep multilayer networks of polynomial functions
- Layer-wise inductive training
- Learning regression models
- Sparse connections, pruning
- 8 layers already back in 1971[26]

[24] A. G. Ivakhnenko and V. G. Lapa, *Cybernetic Predicting Devices*. CCM Information Corporation, 1965.

[25] A. G. Ivakhnenko, "The group method of data handling – a rival of the method of stochastic approximation," *Soviet Automatic Control*, vol. 13, no. 3, pp. 43–55, 1968.

[26] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Transactions on Systems, Man and Cybernetics*, no. 4, pp. 364–378, 1971.

Modern Deep Learning

- Inspired by simple and complex cells in visual cortex[27]
- Neocognitron: Alternating weight-replicating convolutional layers and subsampling layers[28]
- Max-pooling sub-sampling (1992)[29]
- Backpropagation for convolutional NNs (1989)[30]
- Backpropagation for pooling Convolutional NNs[31]
- Max-pooling Conv. NNs on GPUs[32]

---

[27] D. H. Wiesel and T. N. Hubel, "Receptive fields of single neurones in the cat's striate cortex," *J. Physiol.*, vol. 148, pp. 574–591, 1959.

[28] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron," *Trans. IECE*, vol. J62-A(10), pp. 658–665, 1979.

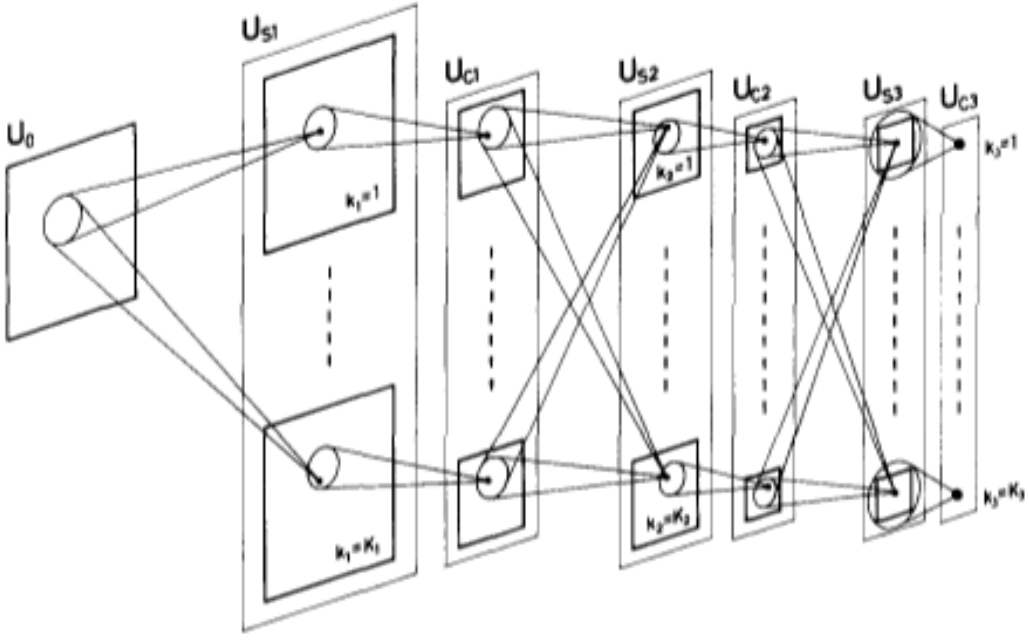
[29] J. Weng, N. Ahuja, and T. S. Huang, "Cresceptron: a self-organizing neural network which grows adaptively," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, vol. 1, 1992, pp. 576–581.

[30] Y. LeCun, L. Bottou, Y. Bengio, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] M. Ranzato, F. J. Huang, Y.-L. Boureau, *et al.*, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383157.

[32] D. C. Ciresan, U. Meier, L. M. Gambardella, *et al.*, "Deep big simple neural nets for handwritten digit recogntion," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

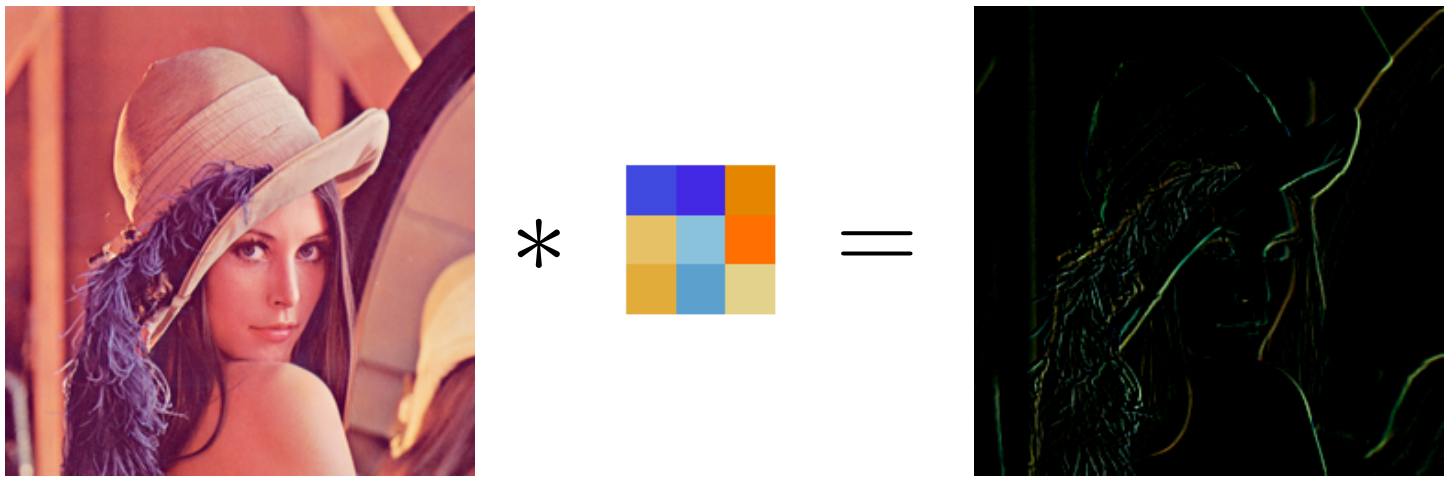
Neocognitron



- Introduced modern Convolutional Neural Networks (CNNs)
- A filter shifted across input map
- Massive weight replication $\rightarrow$ convolutions
- Non-trainable sub-sampling connections
- Trained with local Winner Take All unsupervised rules

Discrete Convolution

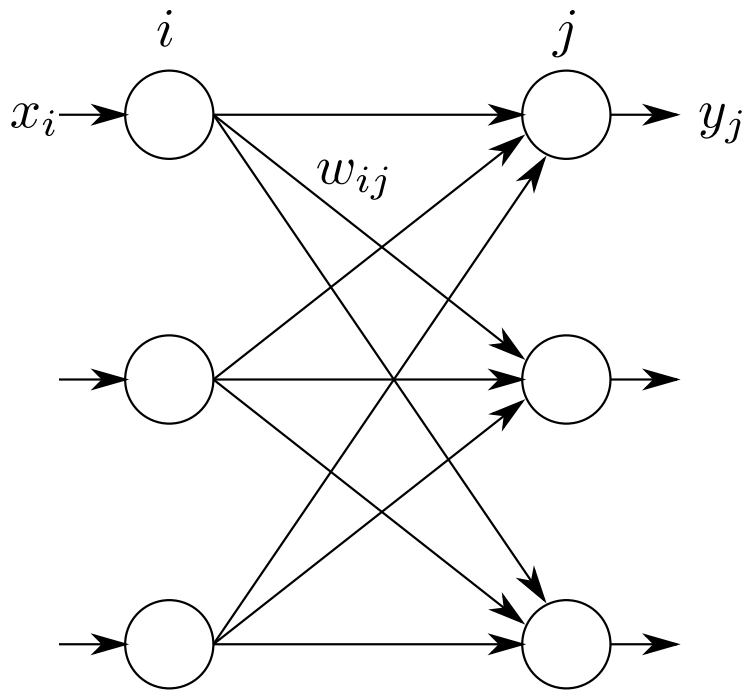
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$



$$\begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} * \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} =$$

$$= \begin{pmatrix} s_{11}k_{22} + s_{12}k_{21} + s_{21}k_{12} + s_{22}k_{11} & s_{12}k_{22} + s_{13}k_{21} + s_{22}k_{12} + s_{23}k_{11} \\ s_{21}k_{22} + s_{22}k_{21} + s_{31}k_{12} + s_{32}k_{11} & s_{22}k_{22} + s_{23}k_{21} + s_{32}k_{12} + s_{33}k_{11} \end{pmatrix}$$

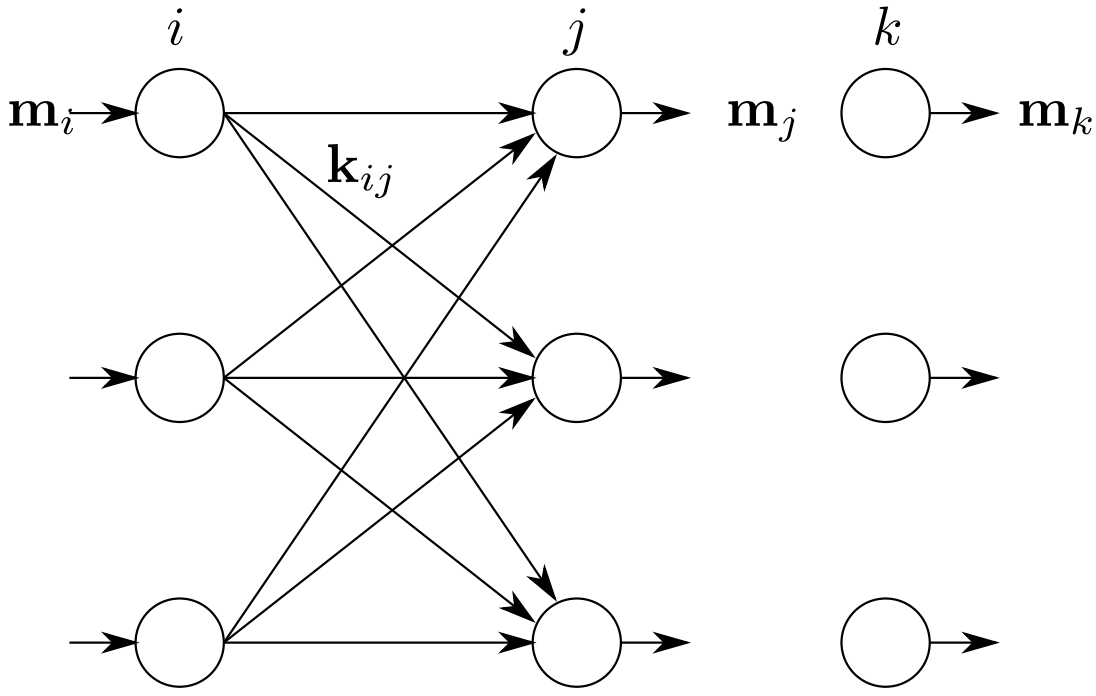
Convolutional Neural Networks



- MLP: $y_j = f(\sum_i w_{ij} x_i)$
- CNN: $\mathbf{m}_j = f(\sum_i \mathbf{k}_{ij} * \mathbf{m}_i)$

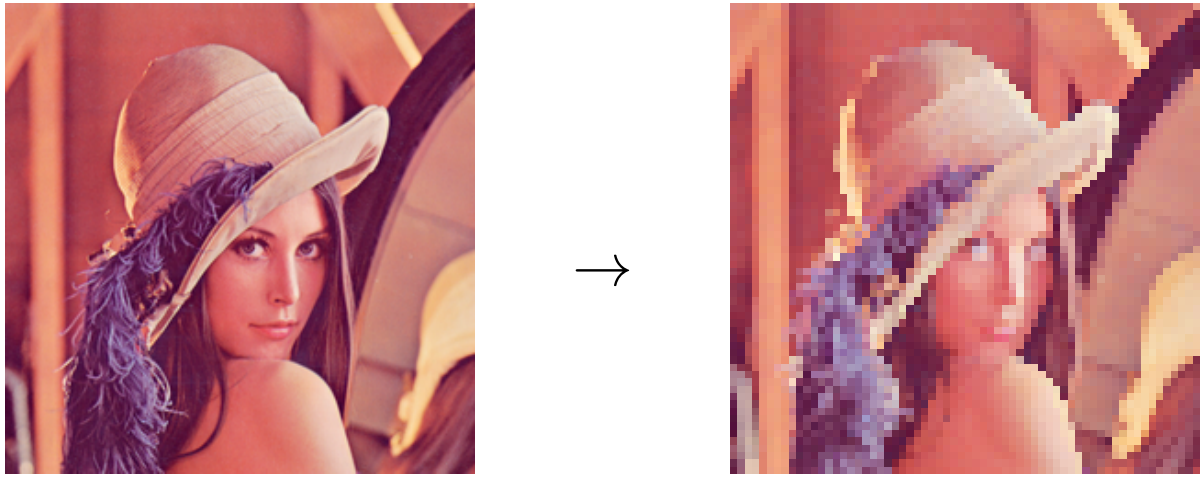
  $\mathbf{m}_i$ and $\mathbf{m}_j$ are pixel maps – whole images
- Matrix $\mathbf{w}$ becomes an array of (2D) filters $\mathbf{k}$

Sub-sampling



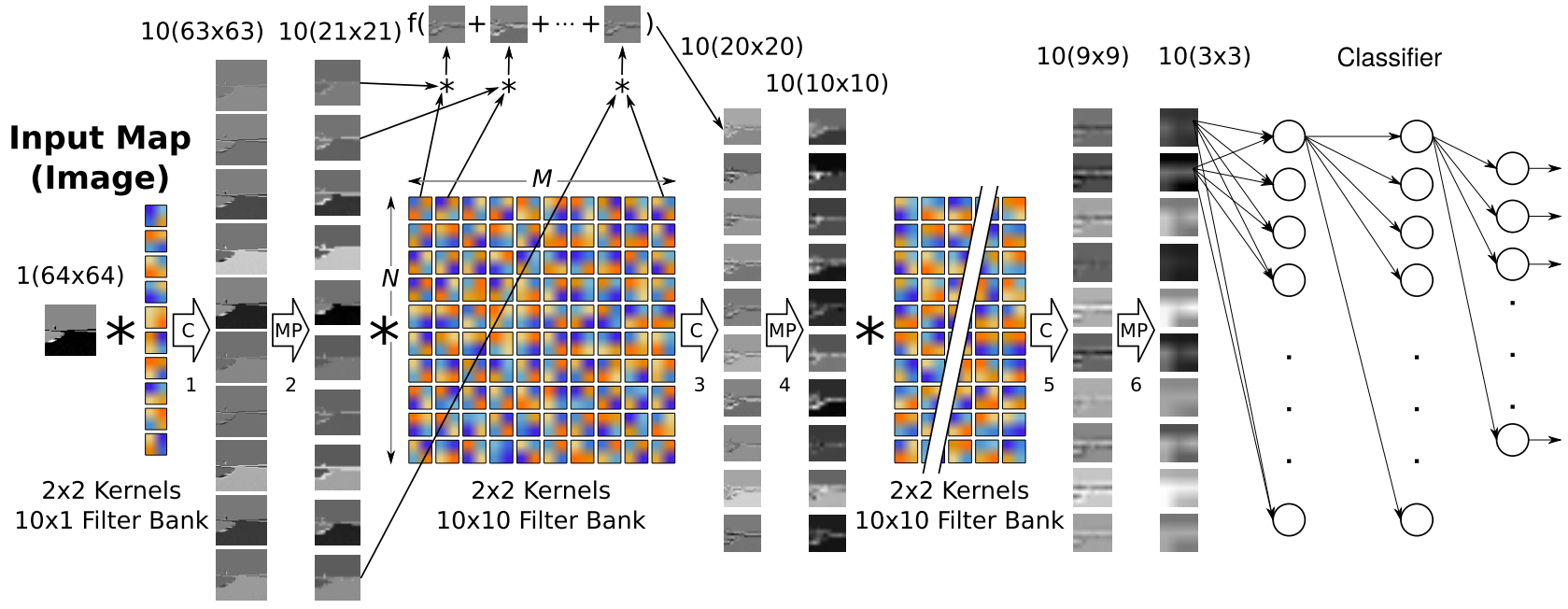
- Sub-sampling (pooling): $\mathbf{m}_k = \mathrm{pool}(\mathbf{m}_j)$
- Max-pooling

Max-Pooling



- Partition image (map) into non-overlapping blocks
- Compute maximum
- Speedup: the gradient propagates to argmax nodes only

Max-Pooling Convolutional Neural Network (MPCNN)



- Alternating Convolutional and Max-Pooling layers produce features
- Classifier (e.g. MLP) classify the features
- Trained with backpropagation

Applications

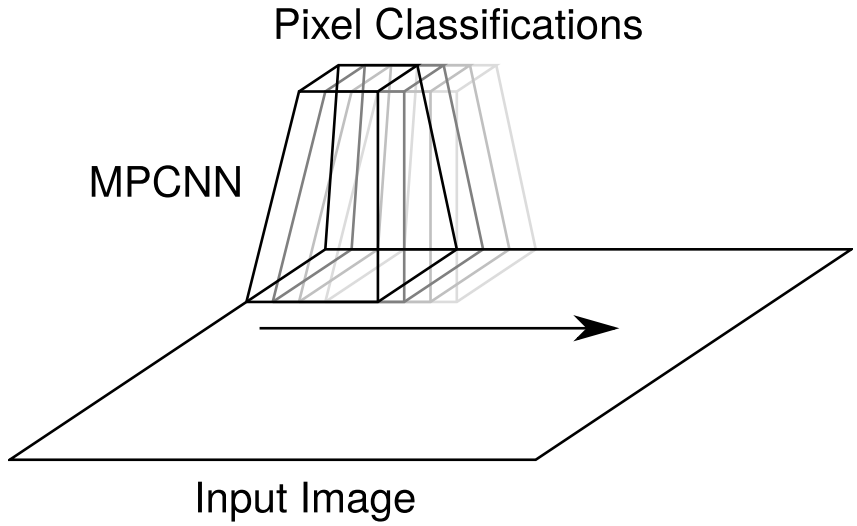
- 1% character error rate in offline Chinese handwriting (3700 classes)[33]

[33] D. C. Ciresan and J. Schmidhuber, "Multi-column deep neural networks for offline handwritten Chinese character classification," IDSIA, Tech. Rep., 2013, arXiv:1309.0261.

- LSVRC-2010 ImageNet classification[34]
- 1.3 million images
- 1000 classes
- 60 million parameters NN
- 5 convolution layers, some max-pooling layers
- 2 fully-connected layers
- Top-5 error of 18.9%

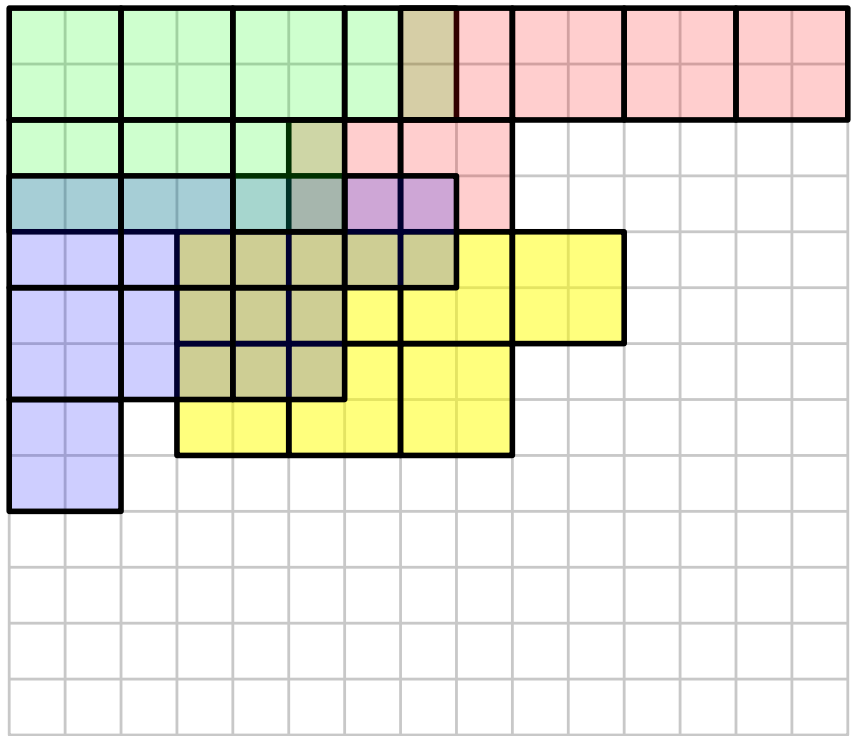
[34] A. Krizhevsky, I Sutskever, and G. E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS 2012)*, 2012, p. 4.

Fast Scanning MPCNN



- How to segment an image?
- Evaluate trained MPCNN for shifted receptive fields (slow)
- Share already computed convolutions
- Problem of pooling layers

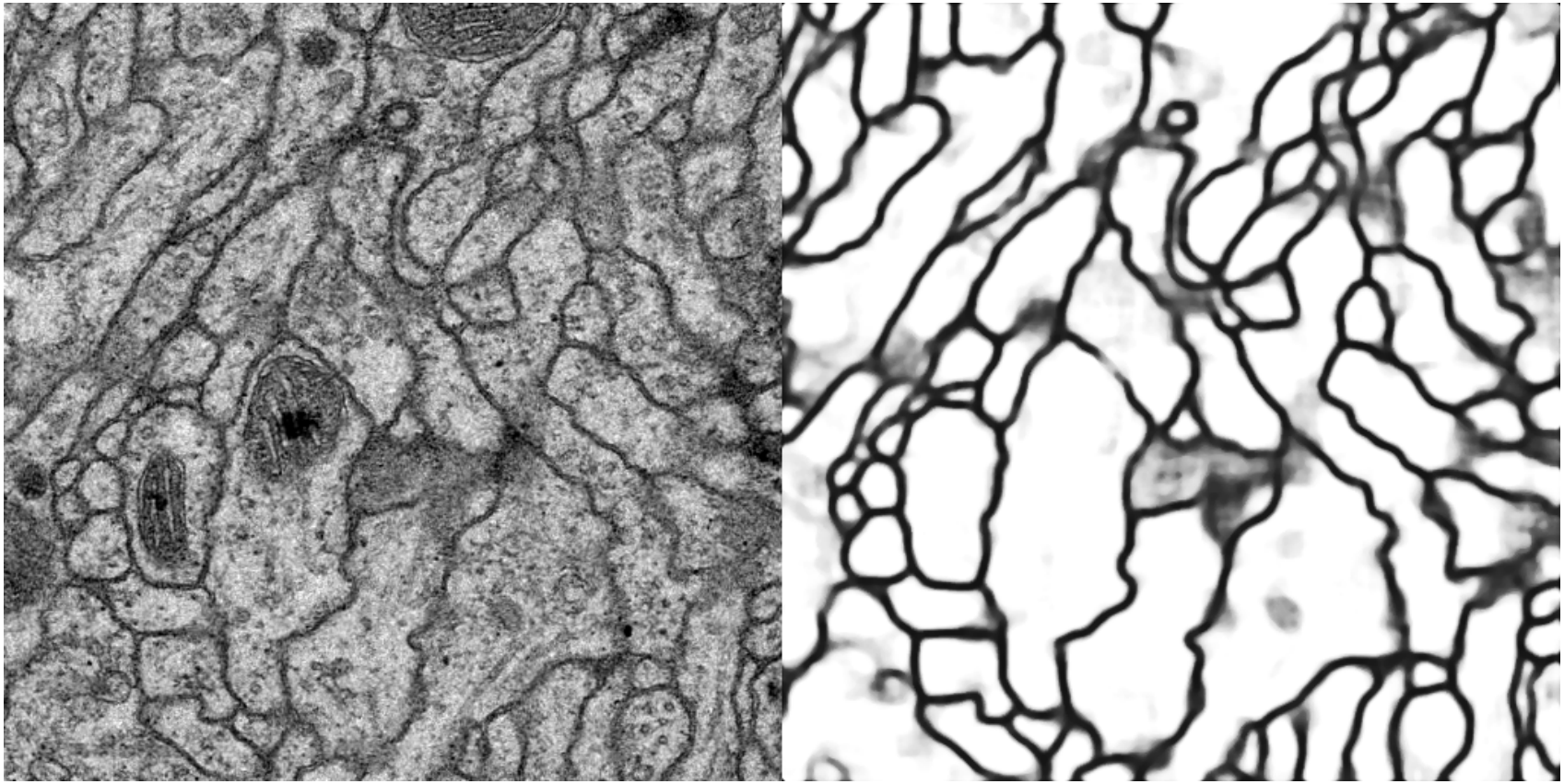
Fast Scanning MPCNN



- For each $m \times n$ pooling block mn fragments are spawned[35]
- Each pooling layer branches the fragments

[35] A. Giusti, D. C. Ciresan, J. Masci, *et al.*, "Fast image scanning with deep max-pooling convolutional neural networks," *CoRR*, vol. abs/1302.1700, 2013.

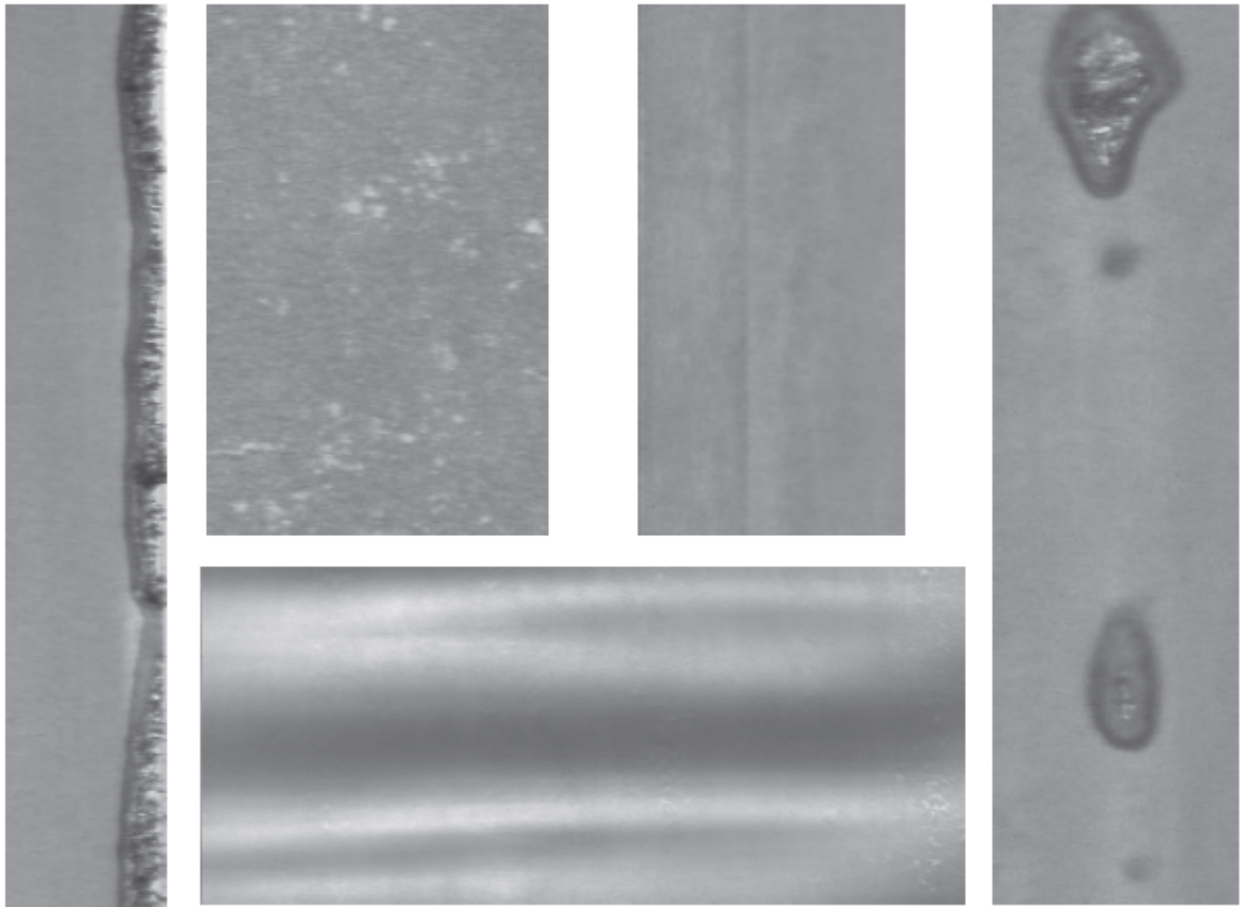
Microscopy Image Segmentation



- ISBI 2012 Segmentation Challenge[36]
- 780× speedup on 512 × 512 images over shifting MPCNN

[36] D. C. Ciresan, A. Giusti, L. M. Gambardella, *et al.*, "Neural networks for segmenting neuronal structures in EM stacks," in *ISBI Segmentation Challenge Competition: Abstracts*, 2012.

Steel Defect Segmentation

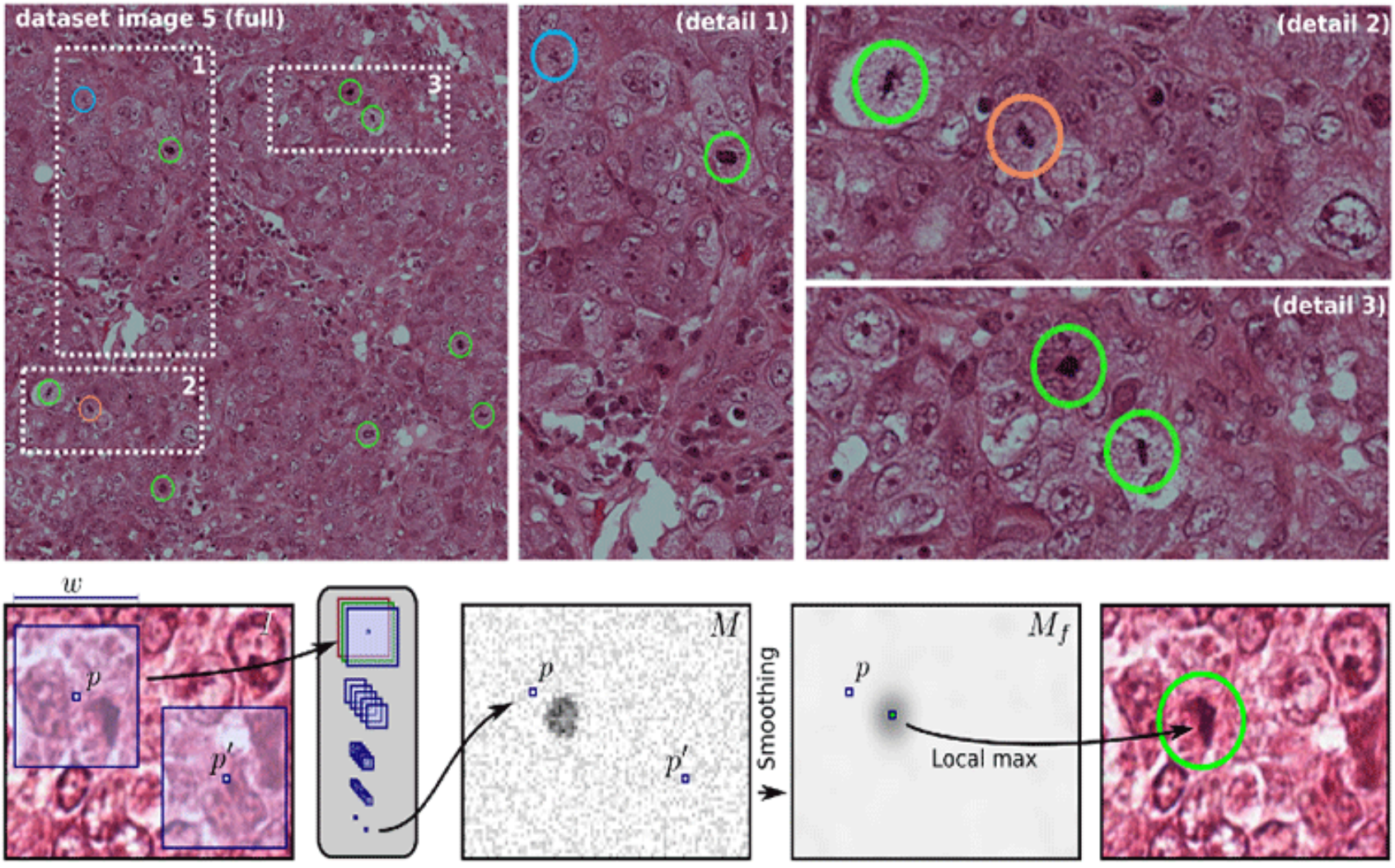


- First industrial scanning MPCNN application[37,38]

[37] J. Masci, U. Meier, D. C. Ciresan, *et al.*, "Steel defect classification with max-pooling convolutional neural networks," in *International Joint Conference on Neural Networks IJCNN*, 2012.

[38] J. Masci, U. Meier, F. G., *et al.*, "Multi-scale pyramidal pooling network for generic steel defect classification," in *International Joint Conference on Neural Networks (IJCNN2013)*, 2013, pp. 1–8. DOI: 10.1109/IJCNN.2013.6706920.

Mitosis Cell Detection

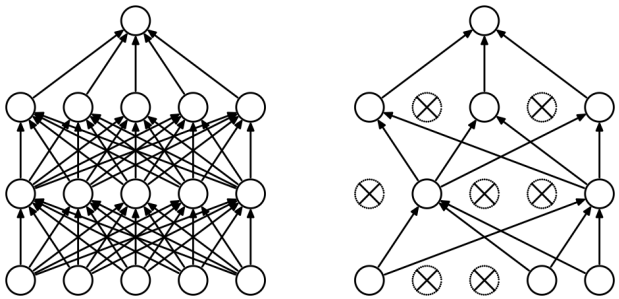


- MICCAI Grand Challenge on mitosis detection[39]

[39] D. C. Ciresan, A. Giusti, L. M. Gambardella, *et al.*, "Mitosis detection in breast cancer histology images with deep neural networks," in *Proc. MICCAI*, vol. 2, 2013, pp. 411–418.

Tweaking NNs

- Training with dropout[40,41]



- Rectified Linear units (ReLu)[42]
- Leave out pooling layers[43]
- Use convolution theorem[44]

---

[40] G. E. Hinton, N. Srivastava, A. Krizhevsky, *et al.*, *Improving neural networks by preventing co-adaptation of feature detectors*, Technical Report arXiv:1207.0580, 2012.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[42] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *International Conference on Machine Learning (ICML)*, 2010. [Online]. Available: https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf.

[43] B. Graham, "Spatially-sparse convolutional neural networks," *CoRR*, vol. abs/1409.6070, 2014. [Online]. Available: http://arxiv.org/abs/1409.6070.

[44] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *CoRR*, vol. abs/1312.5851, 2013. [Online]. Available: http://arxiv.org/abs/1312.5851.

Implementations

- Caffe[45]
  - Berkeley Vision Group, opensource, GPU/CPU Support, large team (1000+ developers), C++
- Theano[46]
  - Arbitrary topologies (graphs), automated gradient calculation, C++/Python, GPU/CPU Support
- Nvidia cuDNN[47]
  - Backend, supports deep network primitives
- TensorFlow[48]
- Keras[49]
  - Frontent to Theano or TensorFlow
- Brainstorm[50]
  - Python only library with cuDNN or CUDA backend, aimed to *RNNs* (what are they?)

[45] http://caffe.berkeleyvision.org/

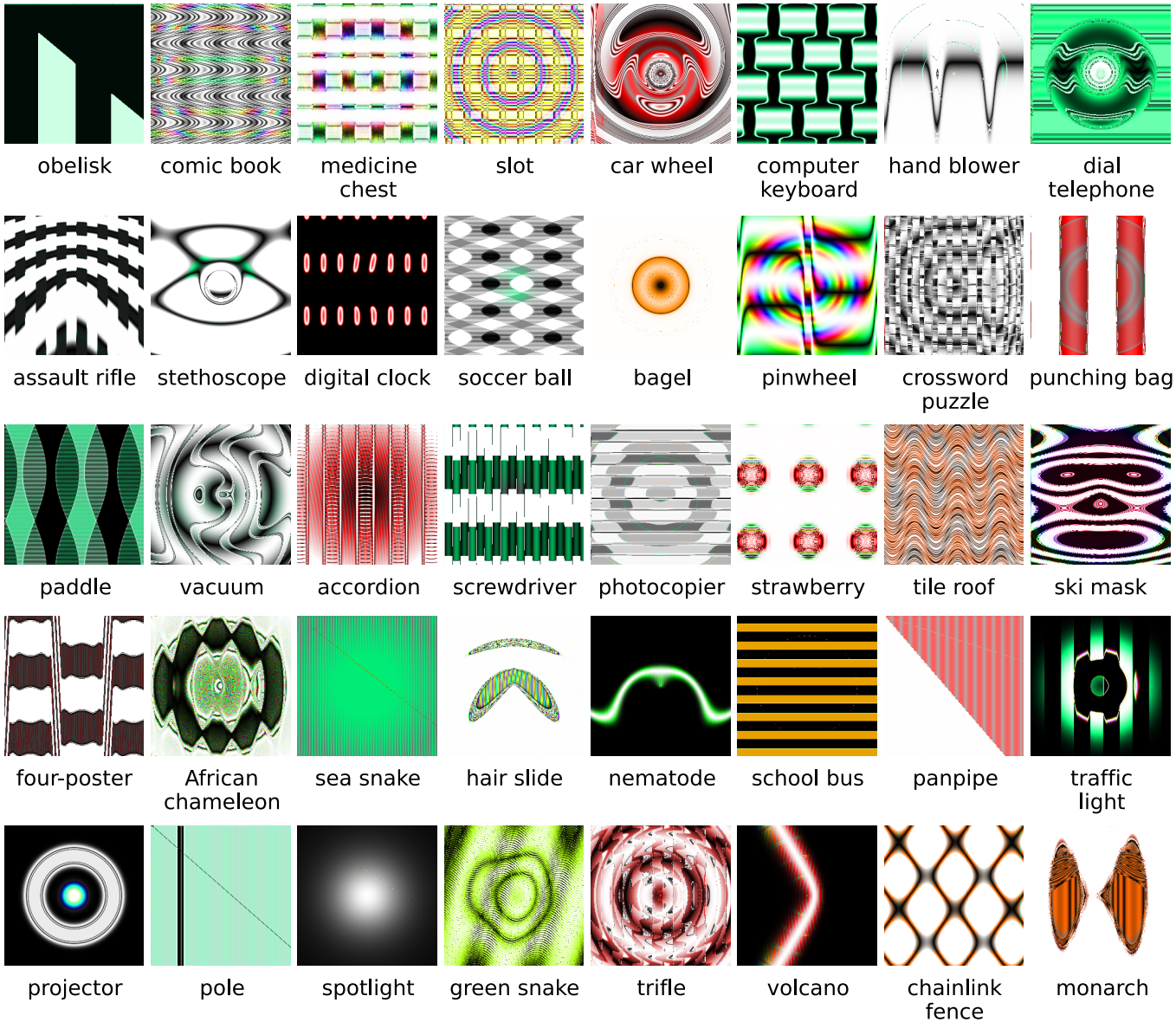
[46] http://deeplearning.net/software/theano

[47] https://developer.nvidia.com/cuDNN

[48] https://www.tensorflow.org/

[49] http://keras.io/

[50] https://github.com/IDSIA/brainstorm



- Evolving and training images that fool CNNs[51]

[51] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: high confidence predictions for unrecognizable images," *CoRR*, vol. abs/1412.1897, 2014. [Online]. Available: http://arxiv.org/abs/1412.1897.

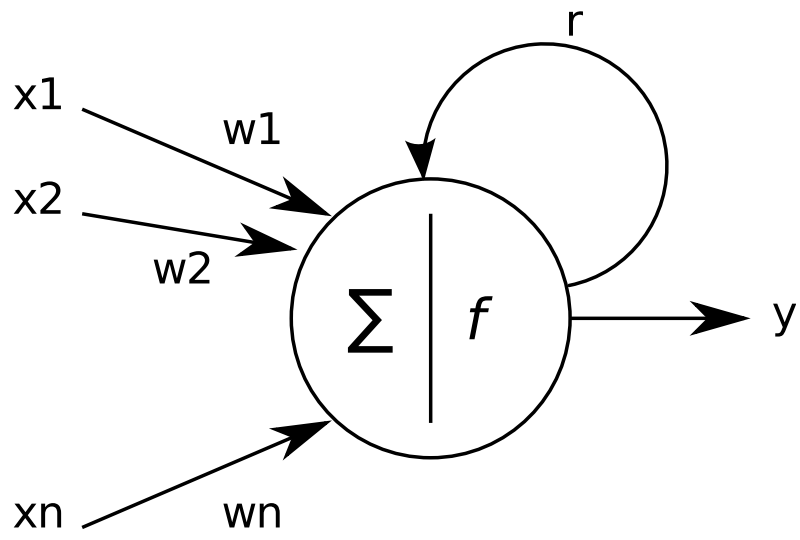
Deepest Learning

Recurrent Neural Networks

Sequence Learning

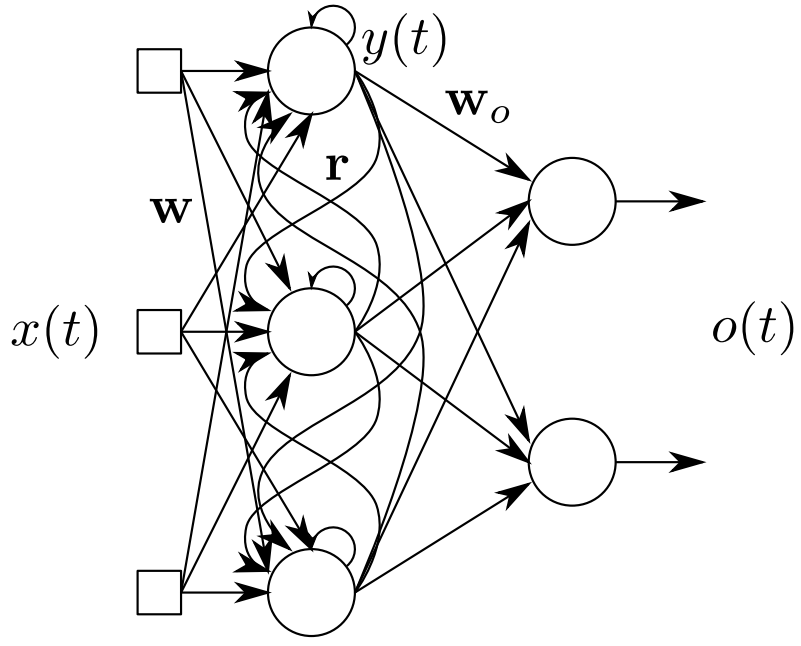
- Up to now, we looked at static mappings:
  $\mathbf{y}(t) = f(\mathbf{x}(t)), \forall t$
- Time t had no meaning
- Used only for the order of presenting inputs $\mathbf{x}$
- It does not matter when $\mathbf{x}$ was presented (except SGD)
- Output $\mathbf{y}$ depends on more than just immediate output
- $\mathbf{y} = f(\mathbf{s}(t)) = F(\mathbf{x}(t), \mathbf{x}(t-1), \ldots, \mathbf{x}(1))$
- where $\mathbf{s}$ is a state changing at every t:
  $\mathbf{s}(t) = g(\mathbf{s}(t-1), \mathbf{x}(t))$
- State is the memory
- Implemented by feedback, recurrent connections

Recurrent Neuron



- $y(t) = f\left(\sum_{i=1}^{I} w_i x_i + r_n y_n(t-1)\right) = f(\mathbf{wx} + ry(t-1))$

Simple Recurrent Network



- $\mathbf{y}(t) = f\big(\sum_{i=1}^{I} w_i x_i + \sum_{n=1}^{N} r_n y_n(t-1)\big) = f(\mathbf{wx} + \mathbf{ry})$
- $\mathbf{o}(t) = f(\mathbf{w_o y}(t))$

Why study sequences?

Natural data is inherently sequential in many cases:

- Speech
- Video
- Natural language
- DNA

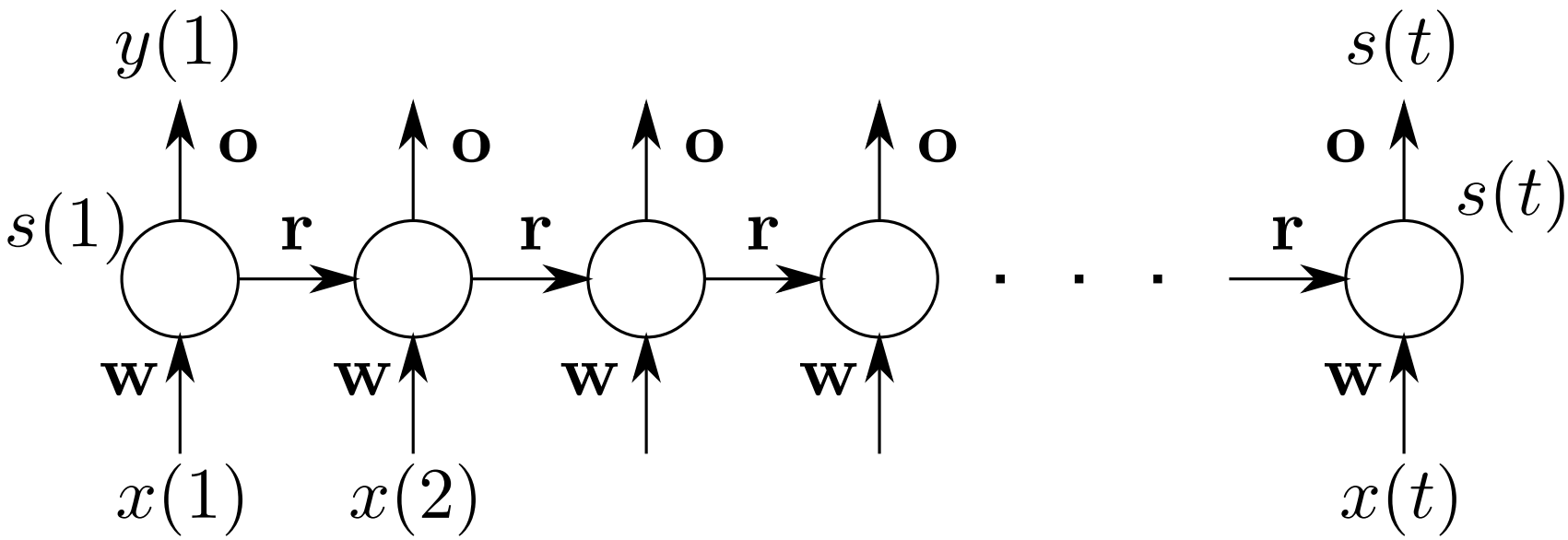
In many cases the memory in important, e.g. for sequential decision making.

- RNNs are general computers[52,53]

[52] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.

[53] J. Schmidhuber, *Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem.* (Dynamic neural nets and the fundamental spatio-temporal credit assignment problem.) *Dissertation, Inst. f. Inf., Tech. Univ. Munich*, 1990.

Backpropagation Through Time



- Unfolding in time[54,55,56]
- Deep Feed-forward network with shared weights across layers

---

[54] R. J. Williams, "Complexity of exact gradient computation algorithms for recurrent neural networks," Boston: Northeastern University, College of Computer Science, Tech. Rep. Technical Report NU-CCS-89-27, 1989.

[55] A. J. Robinson and F. Fallside, "The utility driven dynamic error propagation network," Cambridge University Engineering Department, Tech. Rep. CUED/F-INFENG/TR.1, 1987.

[56] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, 1988.

Vanishing Gradient Problem

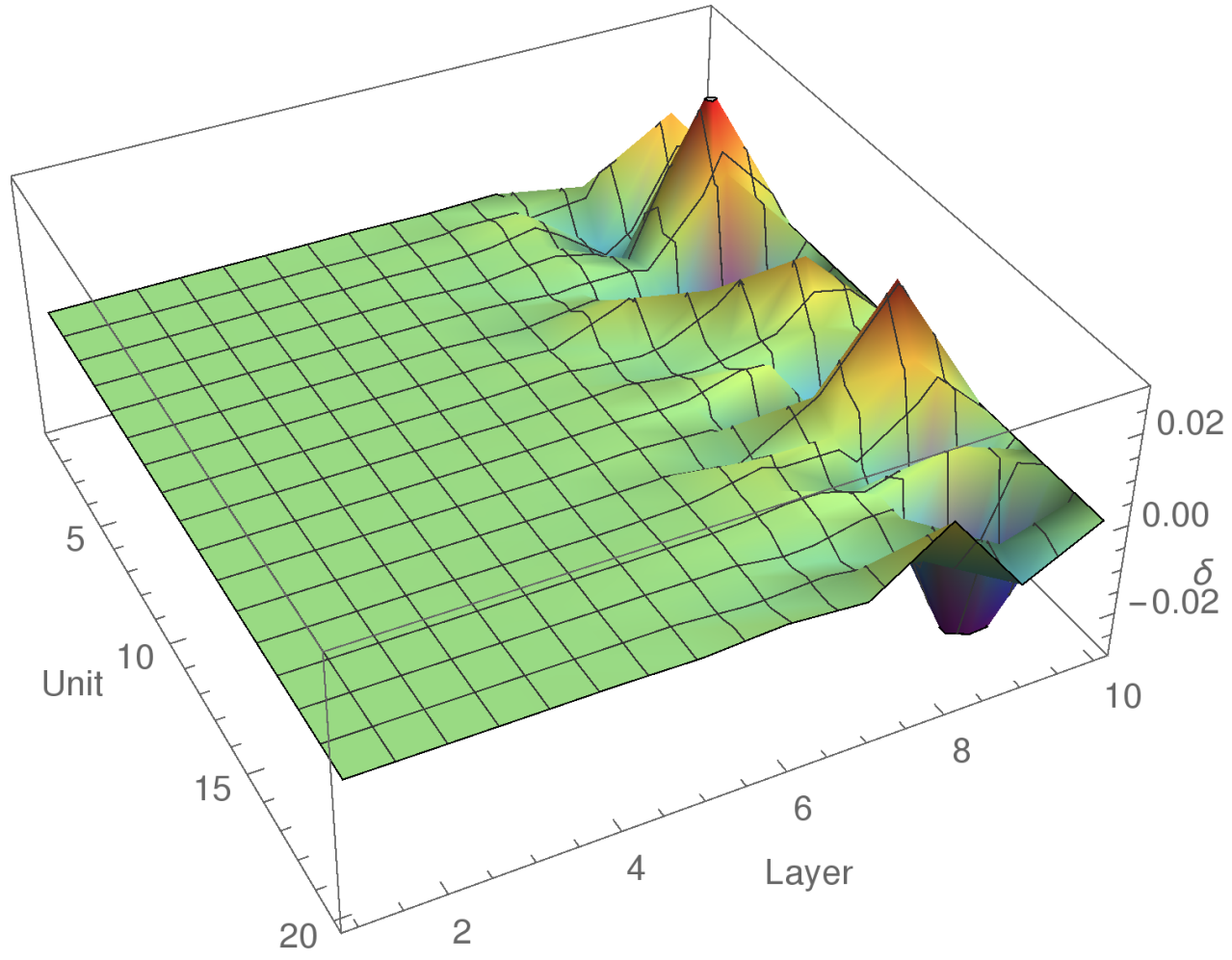
- Training RNNs suffers from dimensionality
- Sequences that span across more than 10 time steps become difficult to train[57,58,59]
- Gradient propagated through non-linearities either vanishes or explodes
- Weights cannot be adjusted to events from the distant past

[57] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen netzen. diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen*, Advisor: J. Schmidhuber, 1991.

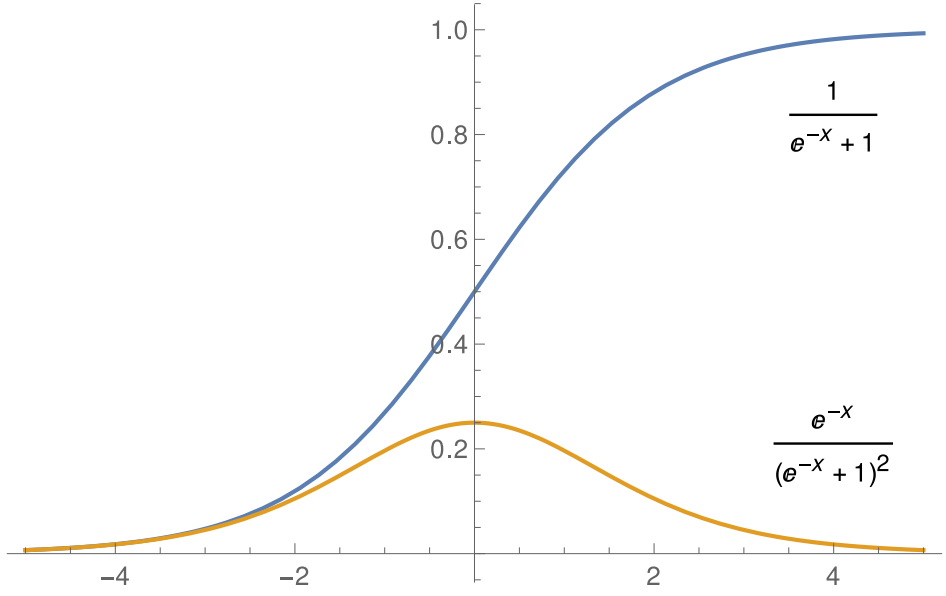
[58] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[59] S. Hochreiter, Y. Bengio, P. Frasconi, *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, Eds., IEEE Press, 2001.

Vanishing Gradient Problem



Vanishing Gradient Problem



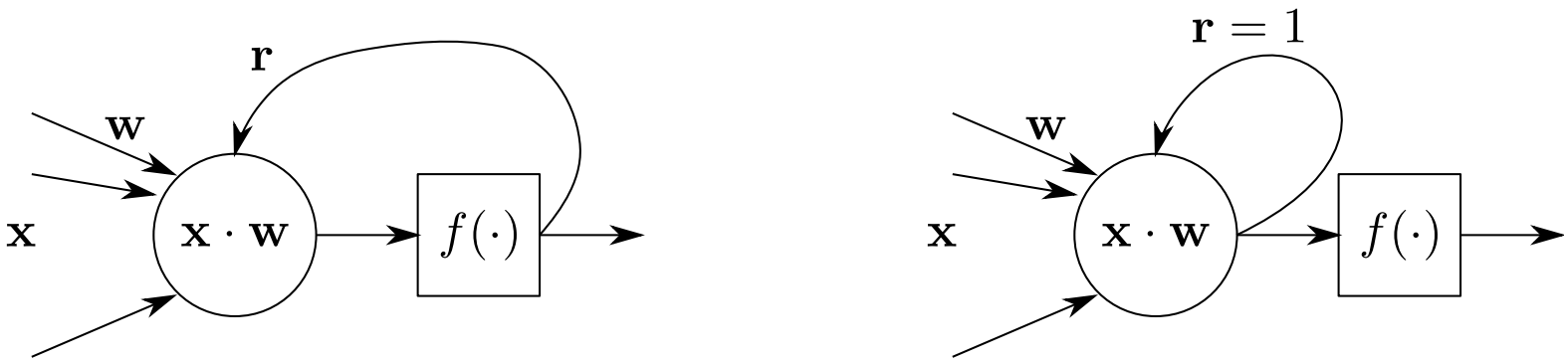
- Error blows up or decays exponentially
- Error tends to vanish when absolute weight values < 4.0
- Large weights do not help (sigmoid derivative goes to zero faster than the weights can grow)
- Increasing number of units does not increase the error flow
- The error decreases even with linear (non-sigmoid) units

Vanishing Gradient Problem

- Recurrent weights $\rightarrow$ matrix power
- If not identity matrix (or a matrix with spectral radius of 1), then
- It either explodes or vanishes
- Simple solution $\rightarrow$ initialize recurrent matrices with identity[60]
- Not as good as LSTM

[60] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *CoRR*, vol. abs/1504.00941, 2015. [Online]. Available: http://arxiv.org/abs/1504.00941.

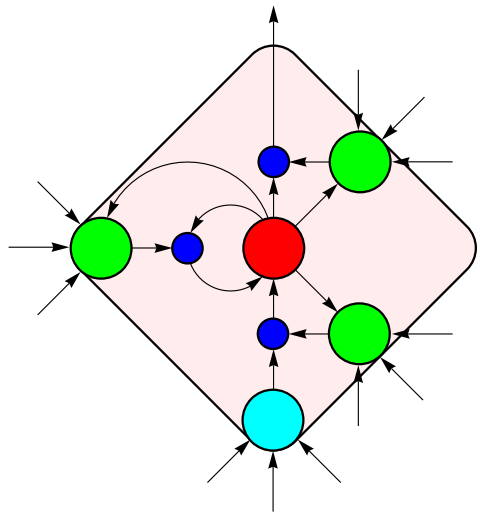
Long Short-Term Memory



- Central idea: **place the recurrent connection before the non-linear function**
- Constant Error Carousel (CEC) - linear cell
- Surround with gating units (multiply the signal with the gate output)[61]

[61] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with LSTM," , IDSIA, Lugano, Lugano, CH, Tech. Rep. IDSIA-01-99, 1999.

Long Short-Term Memory



- Long Short-Term Memory (LSTM)[62,63]
- Architectural solution vanishing gradient problem
- Detailed analysis in the *Odyssey* paper[64]

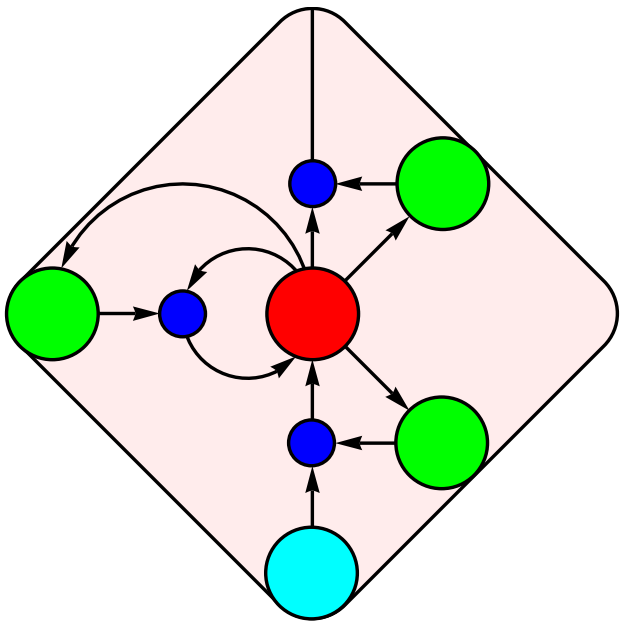
---

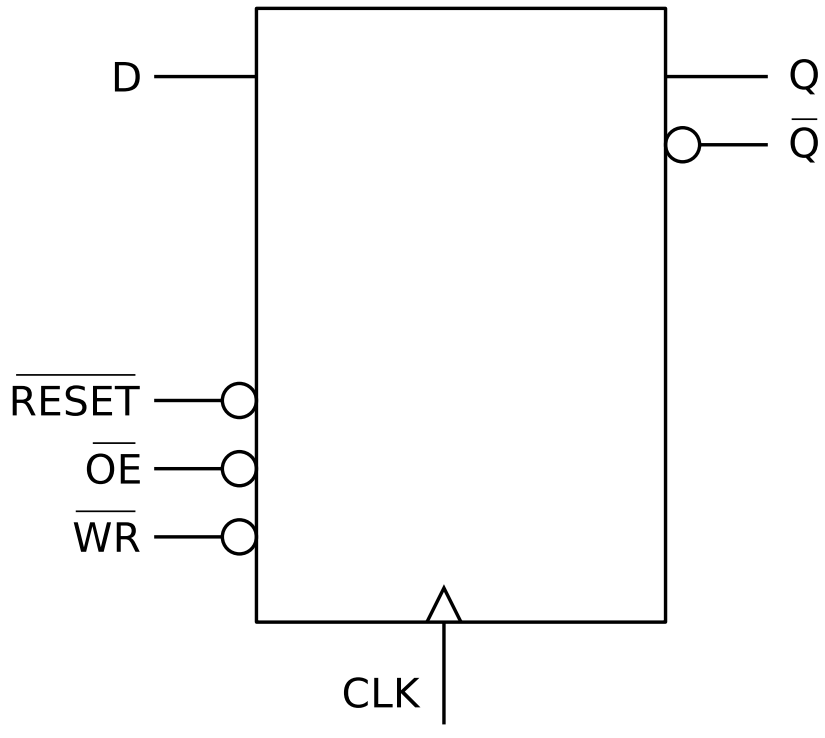
[62] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Fakultät für Informatik, Technische Universität München, Tech. Rep. FKI-207-95, 1995, Revised 1996 (see www.idsia.ch/~juergen, www7.informatik.tu-muenchen.de/~hochreit).

[63] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

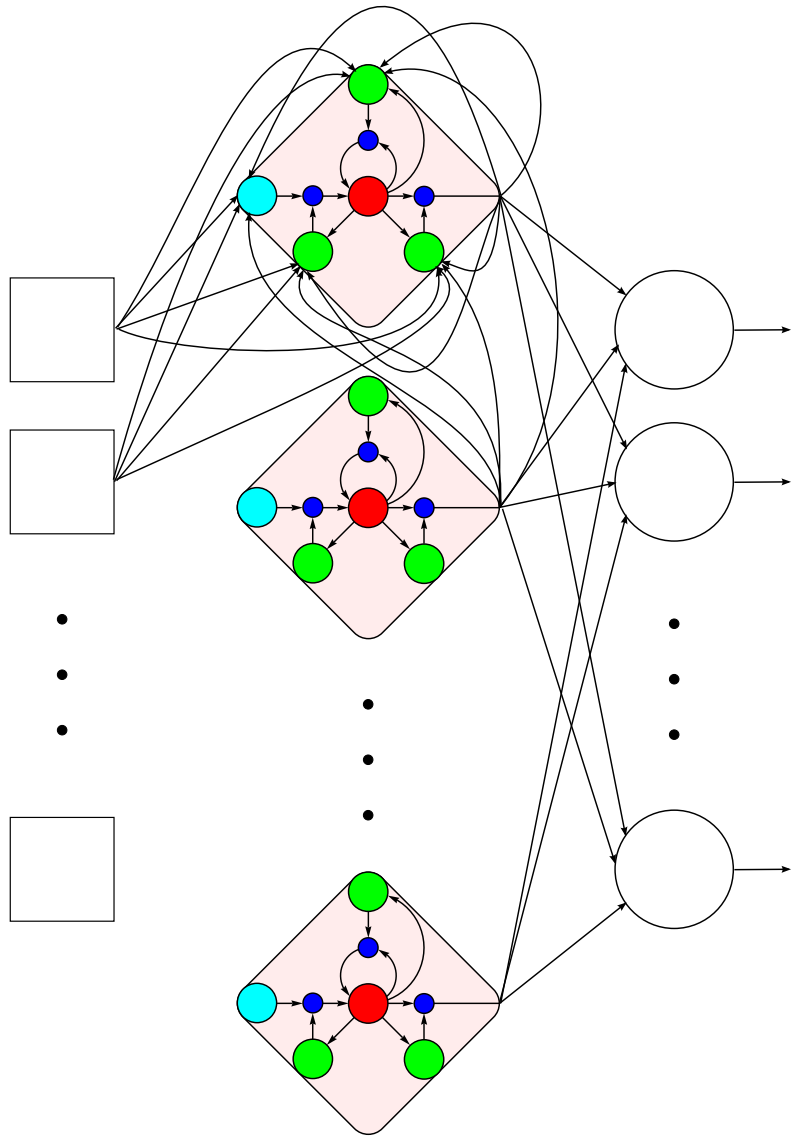
[64] K. Greff, R. K. Srivastava, J. Koutník, *et al.*, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015. [Online]. Available: `http://arxiv.org/abs/1503.04069`.

Long Short-Term Memory

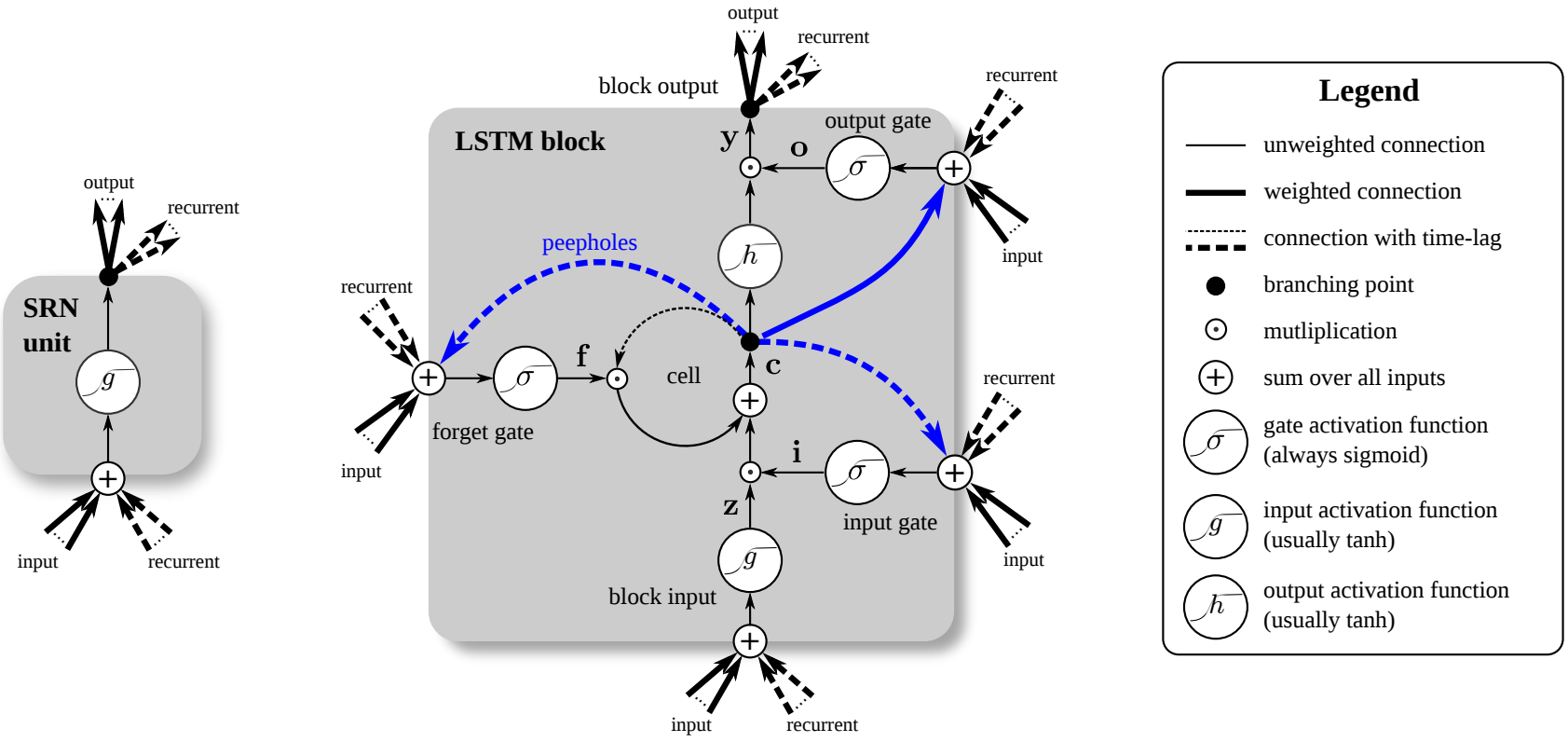


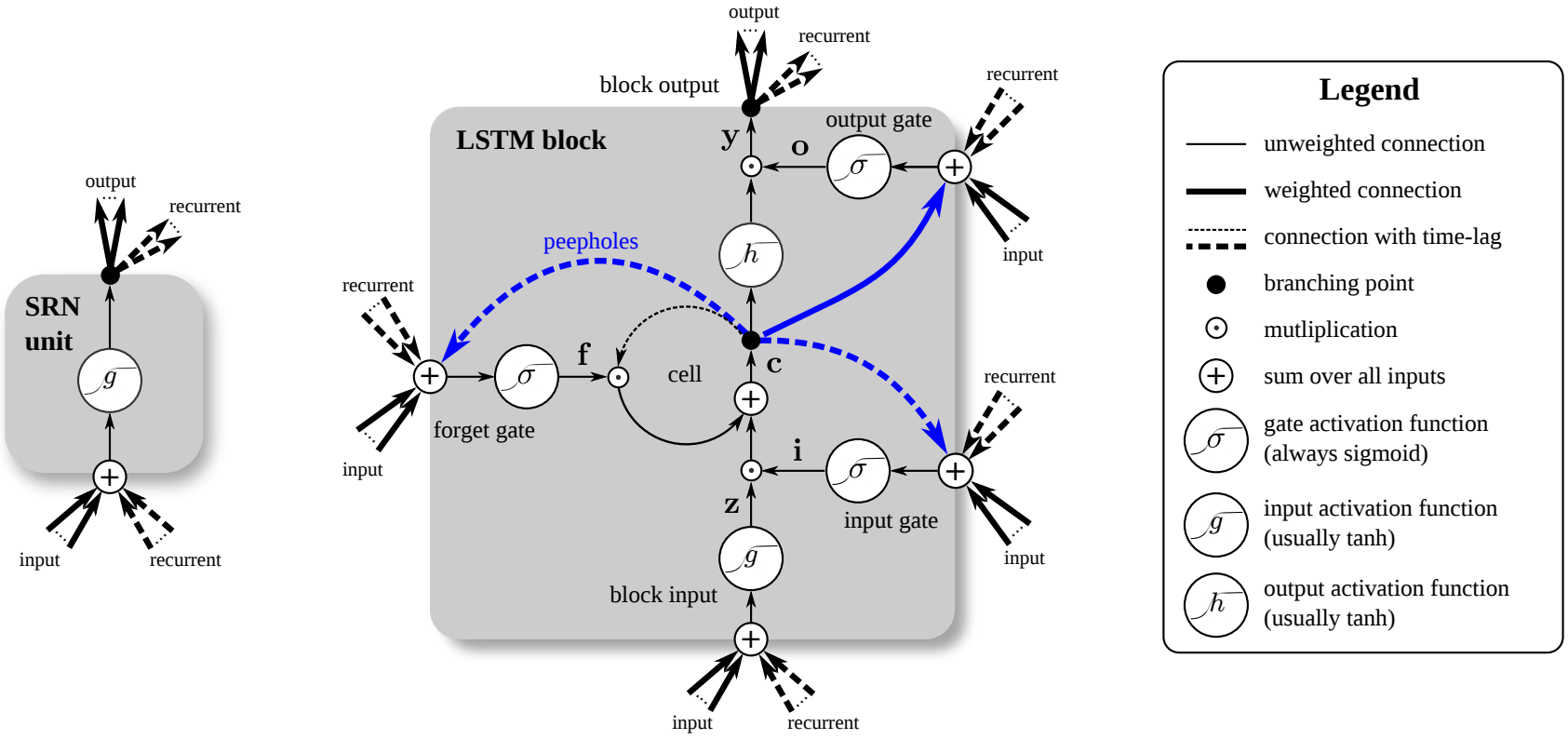


Long Short-Term Memory



LSTM Forward Pass





$$\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z) \qquad \textit{block input}$$

$$\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) \qquad \textit{input gate}$$

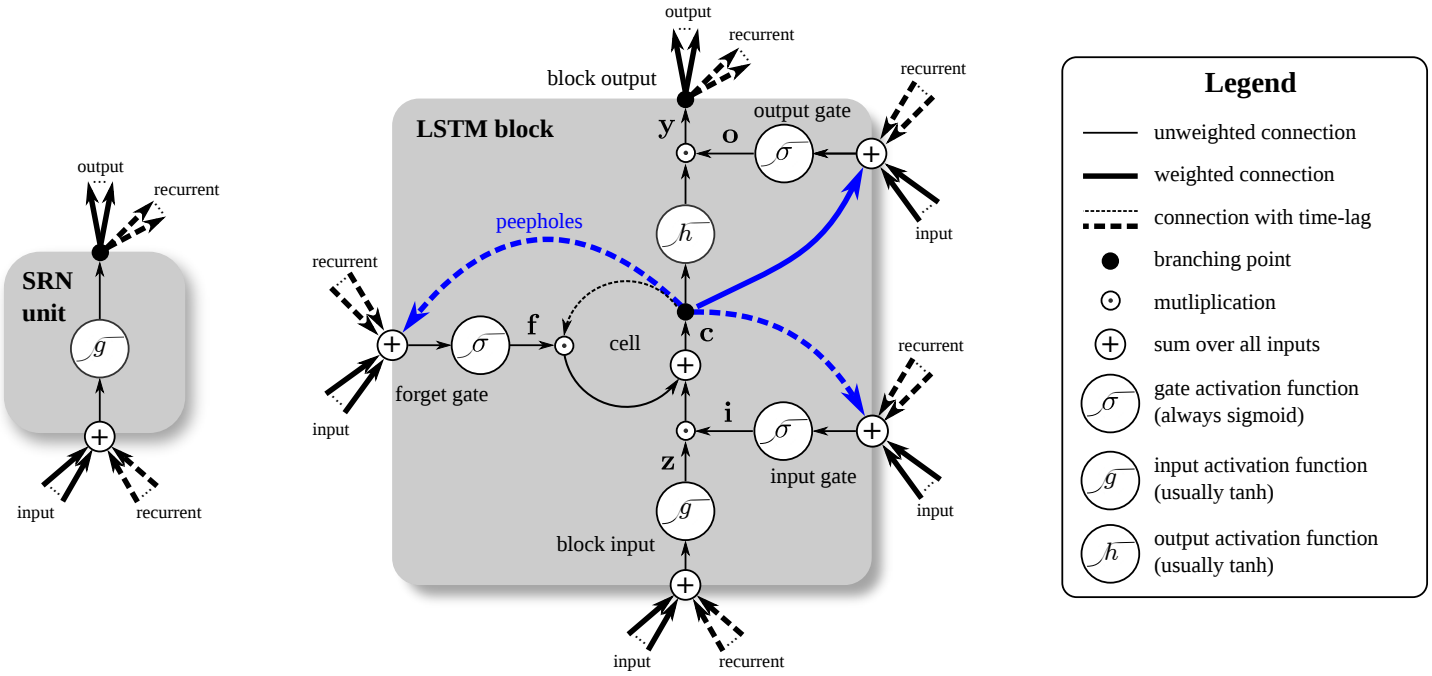
$$\mathbf{f}^t = \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) \qquad \textit{forget gate}$$

$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \qquad \textit{cell state}$$

$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) \qquad \textit{output gate}$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h(\mathbf{c}^t) \qquad \textit{block output}$$

LSTM Backward Pass



$$\delta \mathbf{y}^t = \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1}$$

$$\delta \mathbf{o}^t = \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t)$$

$$\delta \mathbf{c}^t = \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \mathbf{o}^t + \mathbf{p}_i \odot \delta \mathbf{i}^{t+1} + \mathbf{p}_f \odot \delta \mathbf{f}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1}$$

$$\delta \mathbf{f}^t = \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t)$$

$$\delta \mathbf{i}^t = \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t)$$

$$\delta \mathbf{z}^t = \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t)$$

Gradient Calculation

$$\delta \mathbf{W}_\star = \sum_{t=0}^{T} \langle \delta\star^t, \mathbf{x}^t \rangle \qquad \delta \mathbf{p}_i = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \mathbf{i}^{t+1}$$

$$\delta \mathbf{R}_\star = \sum_{t=0}^{T-1} \langle \delta\star^{t+1}, \mathbf{y}^t \rangle \qquad \delta \mathbf{p}_f = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \mathbf{f}^{t+1}$$

$$\delta \mathbf{b}_\star = \sum_{t=0}^{T} \delta\star^t \qquad \delta \mathbf{p}_o = \sum_{t=0}^{T} \mathbf{c}^t \odot \delta \mathbf{o}^t$$

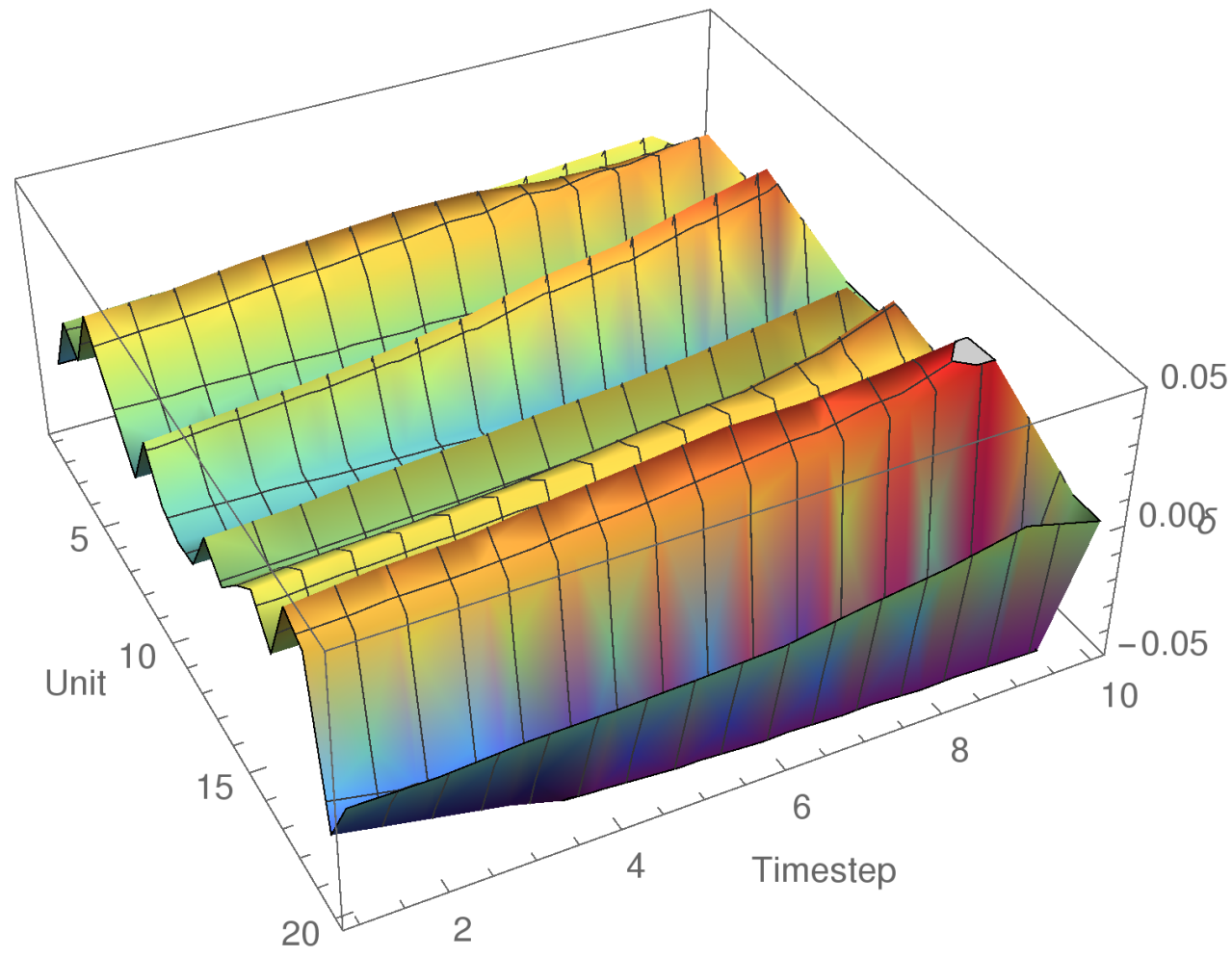
LSTM Properties

- Learn order from separated events in noisy environments
- Robust storage of high precision numbers across time intervals
- Recall information separated by long temporal lags
- Can learn regular and context free languages[65]
- Biologically plausible[66]

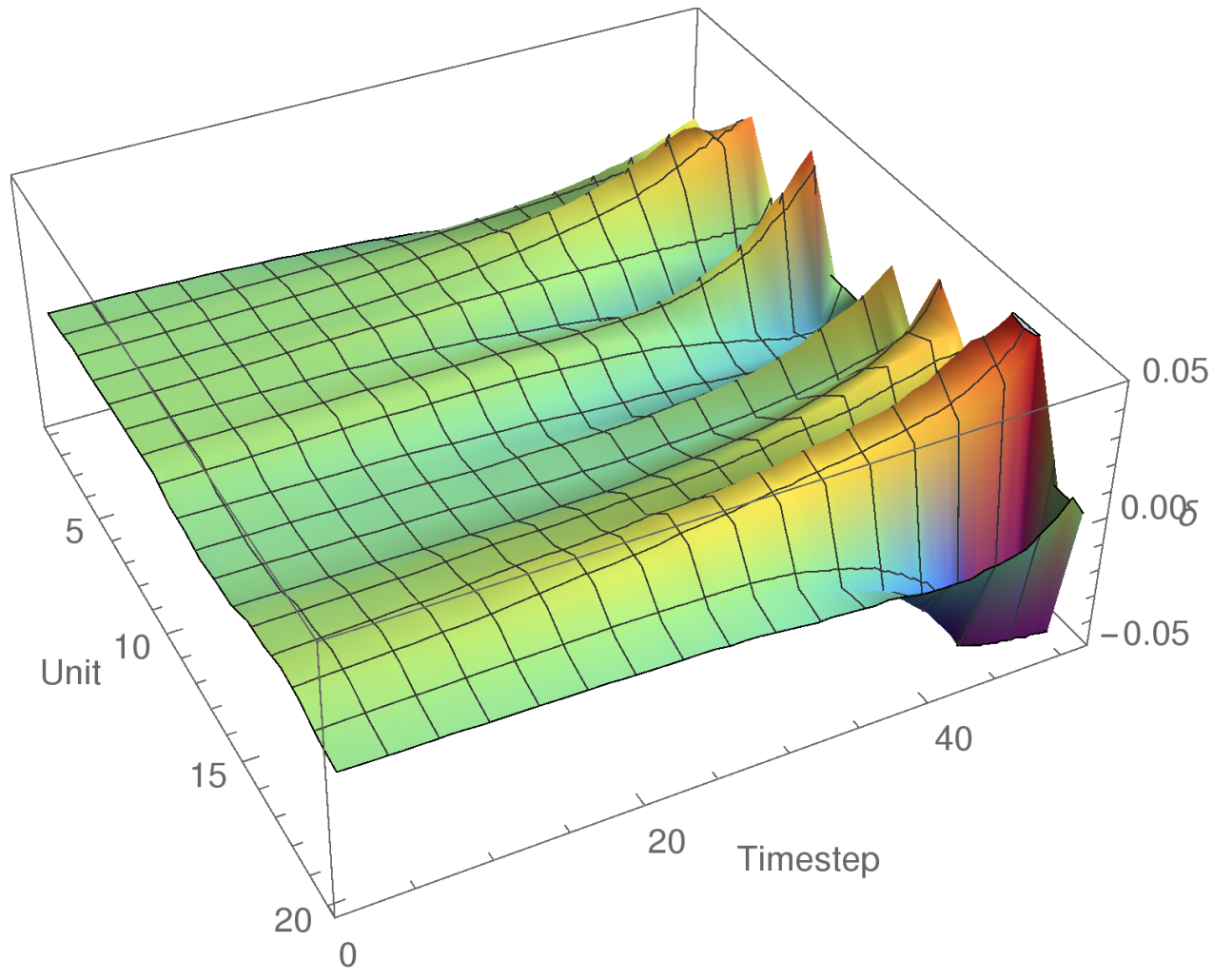
[65] F. A. Gers and J. Schmidhuber, "Long Short-Term Memory learns context free and context sensitive languages," in *Proceedings of the ICANNGA 2001 Conference*, K. et./ al./, Ed., vol. 1, Wien, NY: Springer, 2001, pp. 134–137.

[66] R. O'Reilly, "Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia," ICS, Tech. Rep. ICS-03-03, 2003.

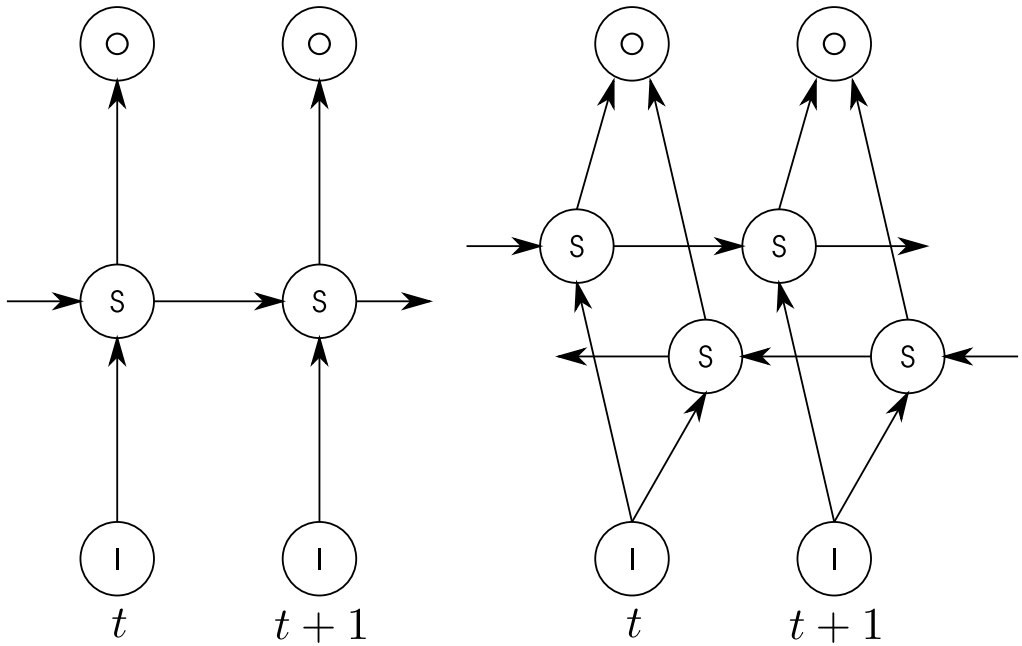
Gradient Flow in LSTM Cells



Gradient Flow in LSTM Cells



Bidirectional RNNs

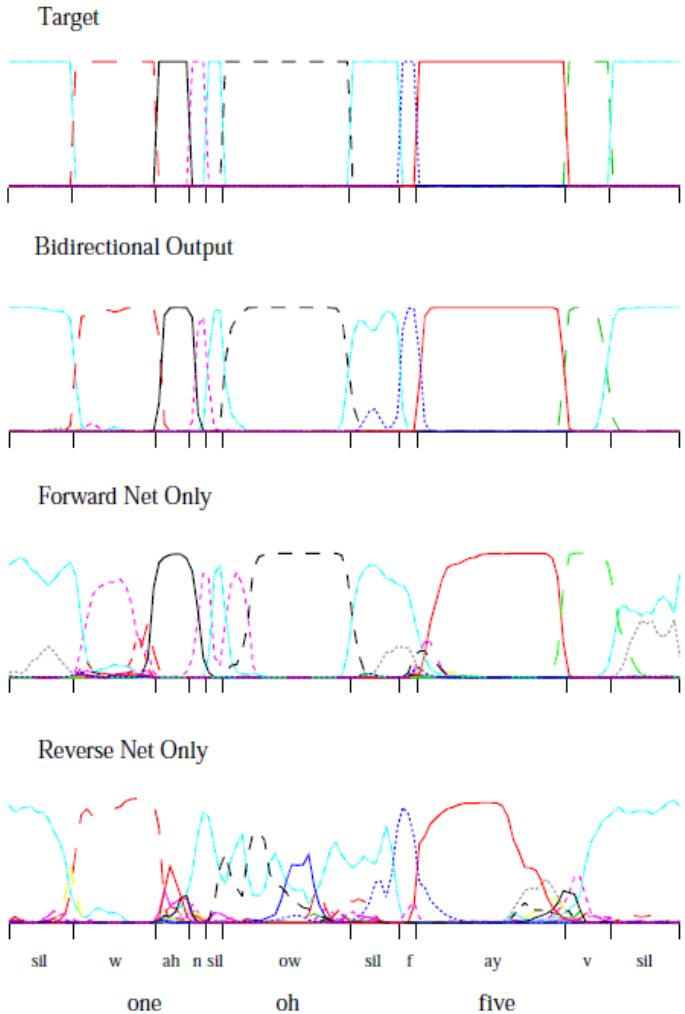


- Time flows in both directions[67,68]
- At each t context from past and future is available

[67] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, 1997.

[68] P. Baldi, S. Brunak, P. Frasconi, *et al.*, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics*, vol. 15, pp. 937–946, 1999.

Bidirectional LSTM



- Phoneme classification in speech data[69]

[69] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

Sequences of Different Lengths

- Connectionist Temporal Classification (CTC)[70]
- Encoder—Decoder[71,72,73]
- Grid LSTM[74]

---

[70] A. Graves, S. Fernandez, F. J. Gomez, *et al.*, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural nets," in *ICML'06: Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 369–376.

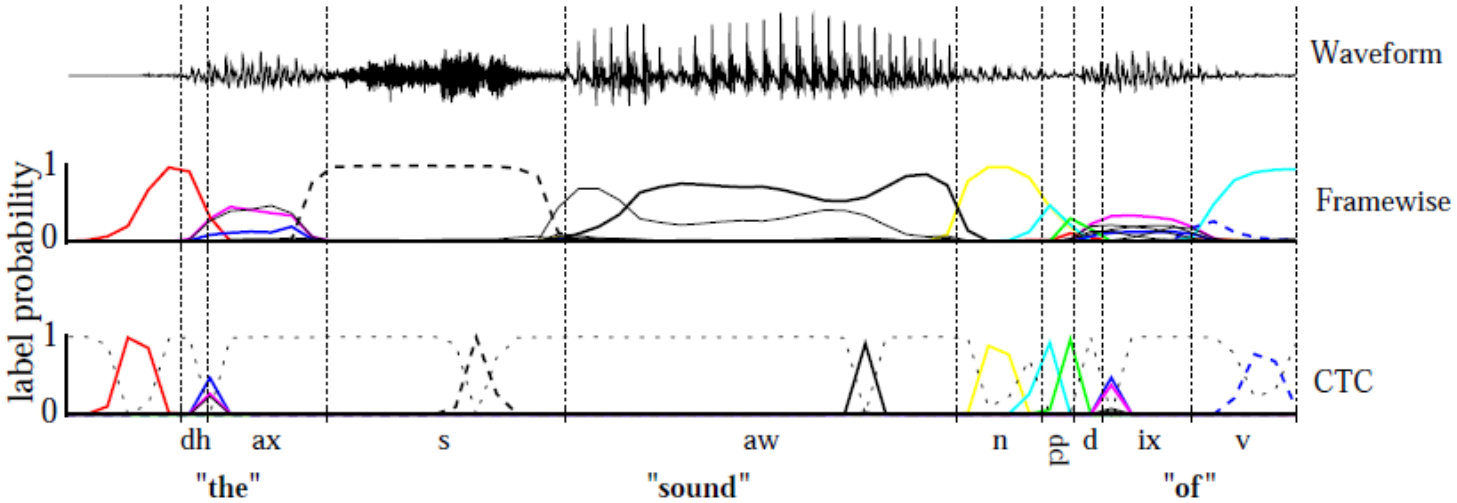
[71] K. Cho, B. van Merrienboer, D. Bahdanau, *et al.*, "On the properties of neural machine translation: encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: http://arxiv.org/abs/1409.1259.

[72] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215.

[73] K. Cho, B. van Merrienboer, Ç. Gülçehre, *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078.

[74] N. Kalchbrenner, I. Danihelka, and A. Graves, "Grid long short-term memory," *CoRR*, vol. abs/1507.01526, 2015. [Online]. Available: http://arxiv.org/abs/1507.01526.

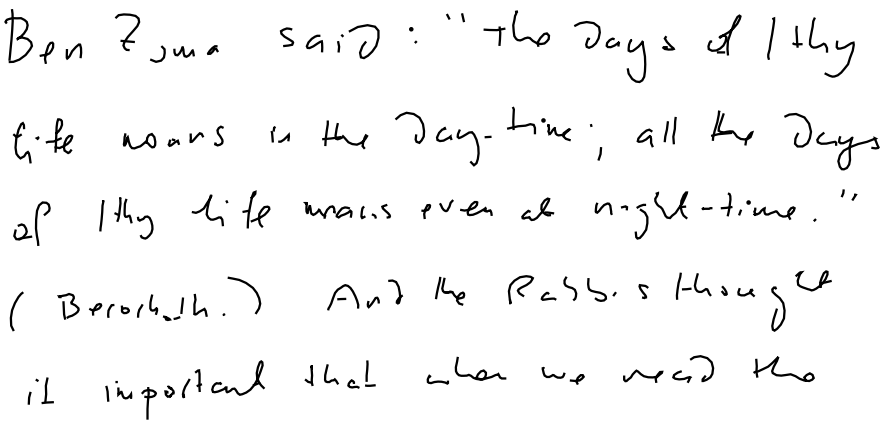
Connectionist Temporal Classification



- Connectionist Temporal Classification (CTC)[75]
- Automatically transforms frame labels to classification segments
- Provides labels for all time steps
- Allows to back-propagate the error from sparse labels

[75] A. Graves, S. Fernandez, F. J. Gomez, *et al.*, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural nets," in *ICML'06: Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 369–376.

Bidirectional LSTM



Ben Zoma said: "The days of 1thy
life means in the day-time; all the days
of 1thy life means even at night-time ."
(Berochoth .) And the Rabbis thought
it important that when we read the

- Online handwriting[76]
- 11.5% Character error rate with extensive preprocessing in 2009[77]
- State-of-the-art results in 2015 (9.26%)[78]

---

[76] M. Liwicki and H. Bunke, "IAM-OnDB – an on-line English sentence database acquired from handwritten text on a whiteboard," in *Proc. 8th Int. Conf. on Document Analysis and Recognition*, vol. 2, 2005, pp. 956–961.

[77] A. Graves, S. Fernandez, M. Liwicki, *et al.*, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in Neural Information Processing Systems (NIPS) 20*, J. Platt, D. Koller, Y. Singer, *et al.*, Eds., Cambridge, MA: MIT Press, 2008, pp. 577–584.

[78] K. Greff, R. K. Srivastava, J. Koutník, *et al.*, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015. [Online]. Available: http://arxiv.org/abs/1503.04069.

2014 – The year of LSTM

Best results achieved using LSTM

1. Large vocabulary speech recognition (Sak et al., Google, Interspeech)
2. English to French translation (Sutskever et al., Google, NIPS)
3. Text-to-speech synthesis (Fan et al., Microsoft, Interspeech)
4. Prosody contour prediction (Fernandez et al., IBM, Interspeech)
5. Language identification (Gonzalez-Dominguez et al., Google, Interspeech)
6. Medium vocabulary speech recognition (Geiger et al., Interspeech)
7. Audio onset detection (Marchi et al., ICASSP)
8. Social signal classification (Brueckner & Schulter, ICASSP)
9. Arabic handwriting recognition (Bluche et al., DAS)
10. Image caption generation (Vinyals et al., Google)
11. Video to textual description (Donahue et al.)

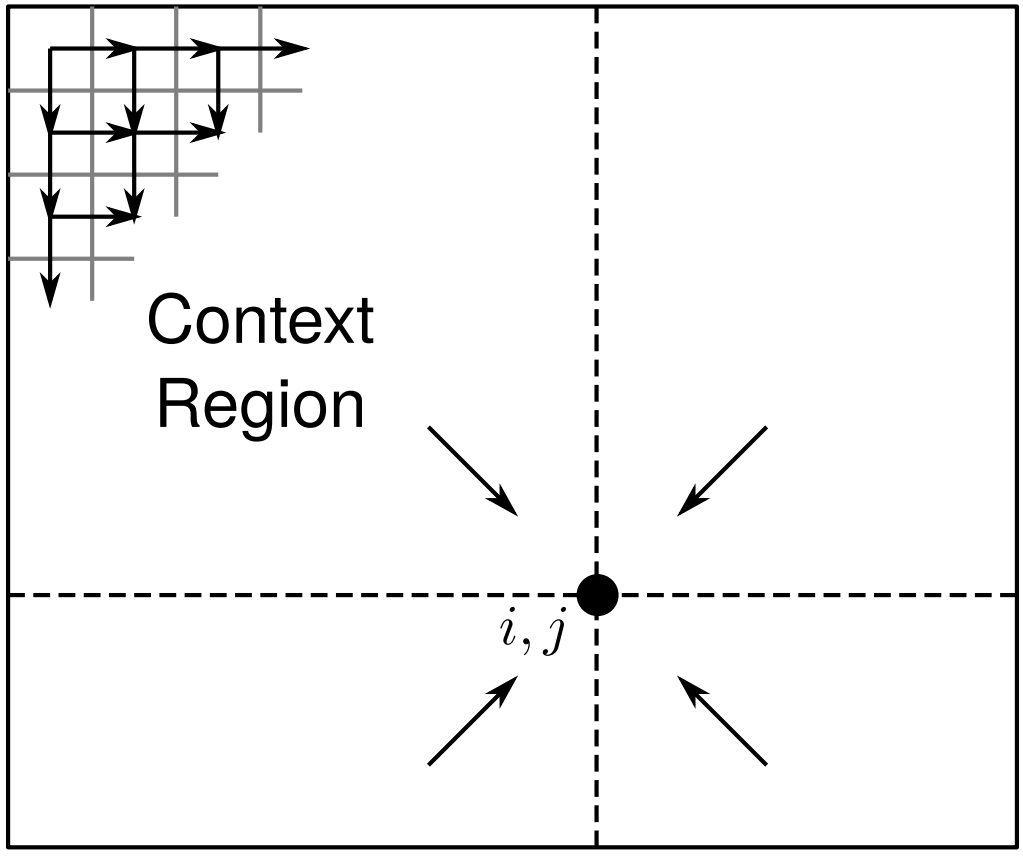
2015 was no Exception

1. Image caption generation (Xu et al.)
2. Scene labeling (Byeon et al., CVPR)
3. Subcellular localization of proteins (Sonderby et al.)
4. Language understanding (Peng et al.)
5. ...

- LSTM&CTC replaces HMMs in Google Voice[79] speech recognition on Android devices (1B+ installations), September 2015.

[79] http://googleresearch.blogspot.ch/2015/09/google-voice-search-faster-and-more.html

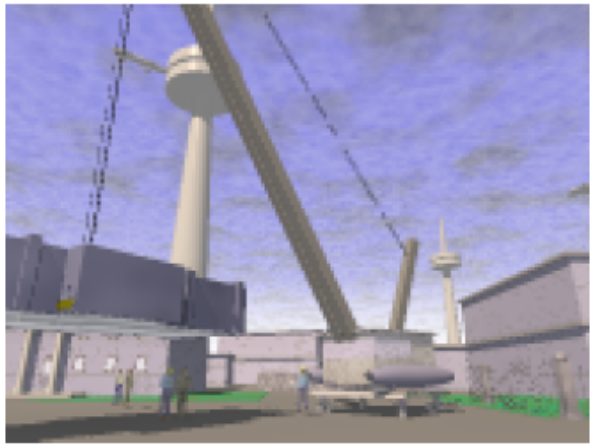
Multidimensional RNNs

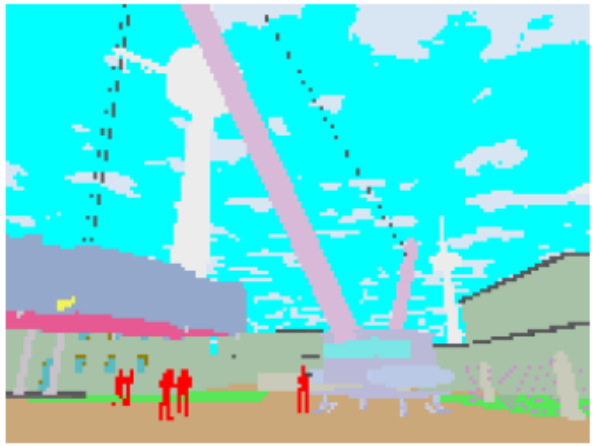


- Input array (image) treated as N-dimensional signal[80]
- 2^N contexts available at each *time* (point)

[80] A. Graves, S. Fernández, and J. Schmidhuber, "Multi-dimensional recurrent neural networks," in *Proceedings of the 17th International Conference on Artificial Neural Networks*, 2007.

Multidimensional LSTM





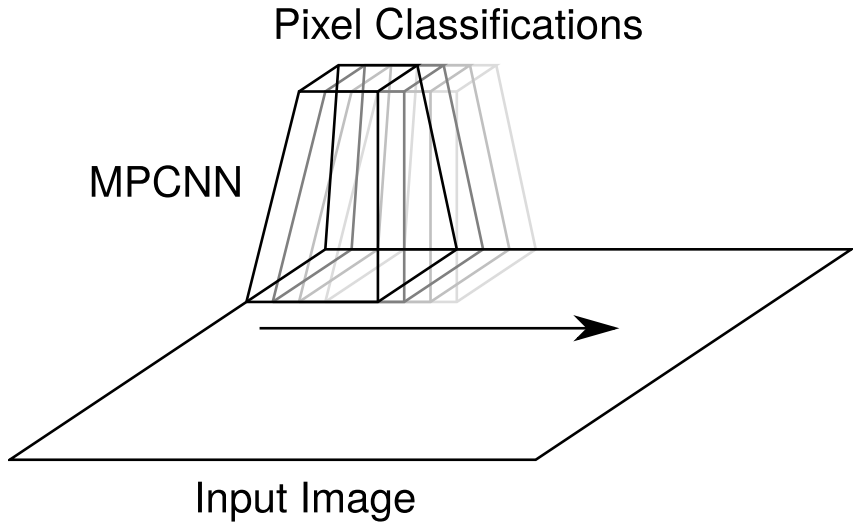
- Recurrent neuron replaced by LSTM unit
- Offline handwriting recognition[81]
- Image segmentation[82]
- Preliminary 2015 results – better than scanning MPCNN

---

[81] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems (NIPS) 21*, Cambridge, MA: MIT Press, 2009, pp. 545–552.

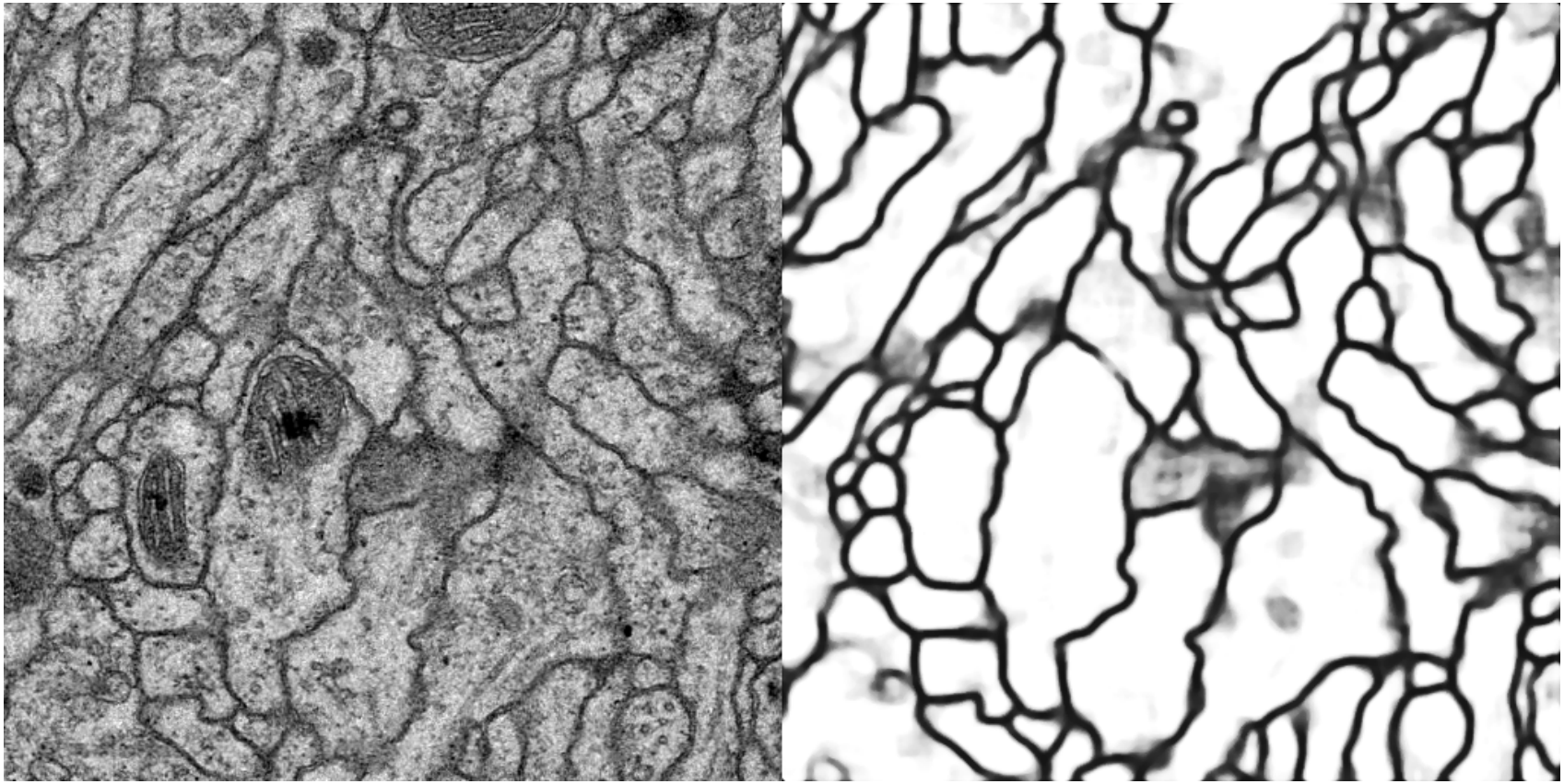
[82] A. Graves, "Supervised sequence labelling with recurrent neural networks," PhD thesis, Technische Universität München, 2008.

Reminder: Fast Scanning MPCNN



- How to segment an image?
- Evaluate trained MPCNN for shifted receptive fields (slow)
- Share already computed convolutions
- Problem of pooling layers

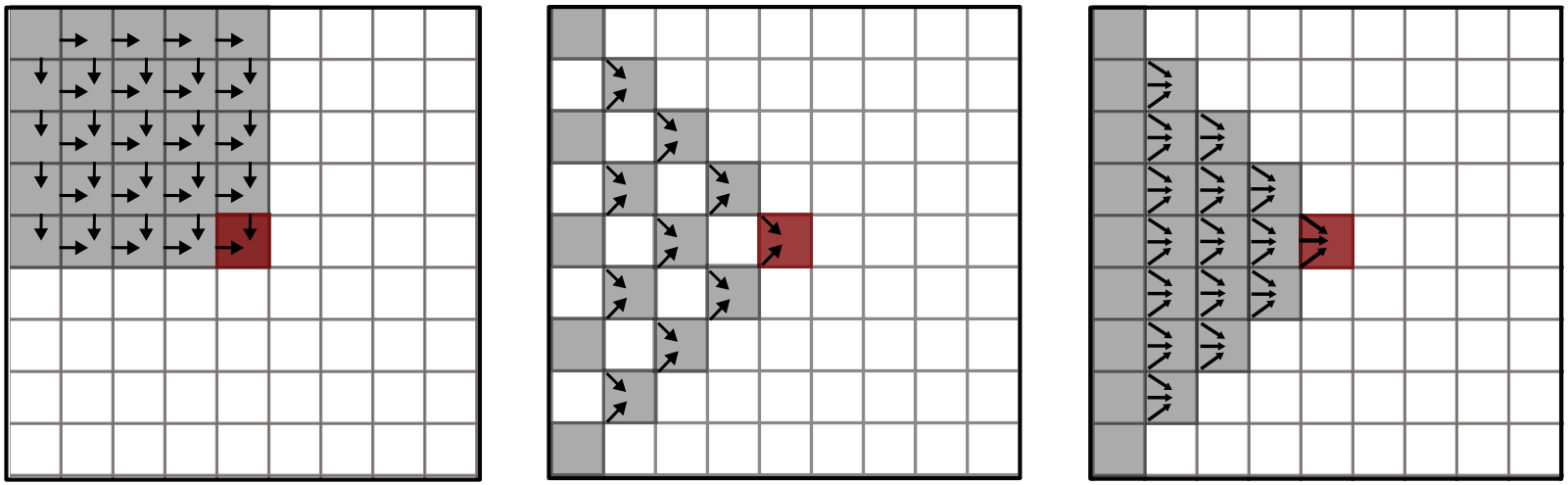
Microscopy Image Segmentation with MPCNN



- ISBI 2012 Segmentation Challenge[83]
- 780× speedup on 512 × 512 images over shifting MPCNN

[83] D. C. Ciresan, A. Giusti, L. M. Gambardella, *et al.*, "Neural networks for segmenting neuronal structures in EM stacks," in *ISBI Segmentation Challenge Competition: Abstracts*, 2012.

PyraMiD LSTM

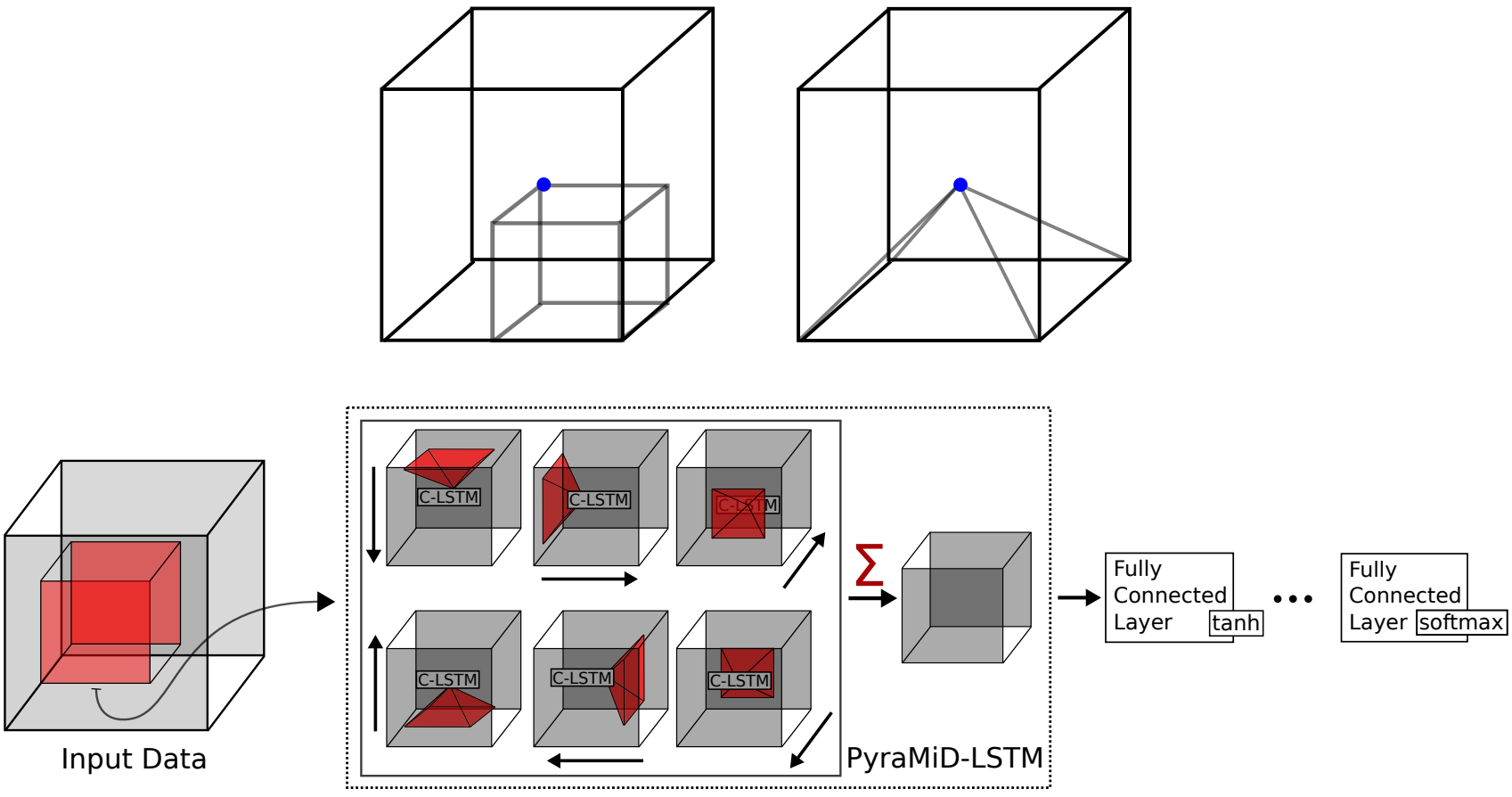


(a) Standard MD-LSTM (b) 'Turned' MD-LSTM (c) PyraMiD LSTM

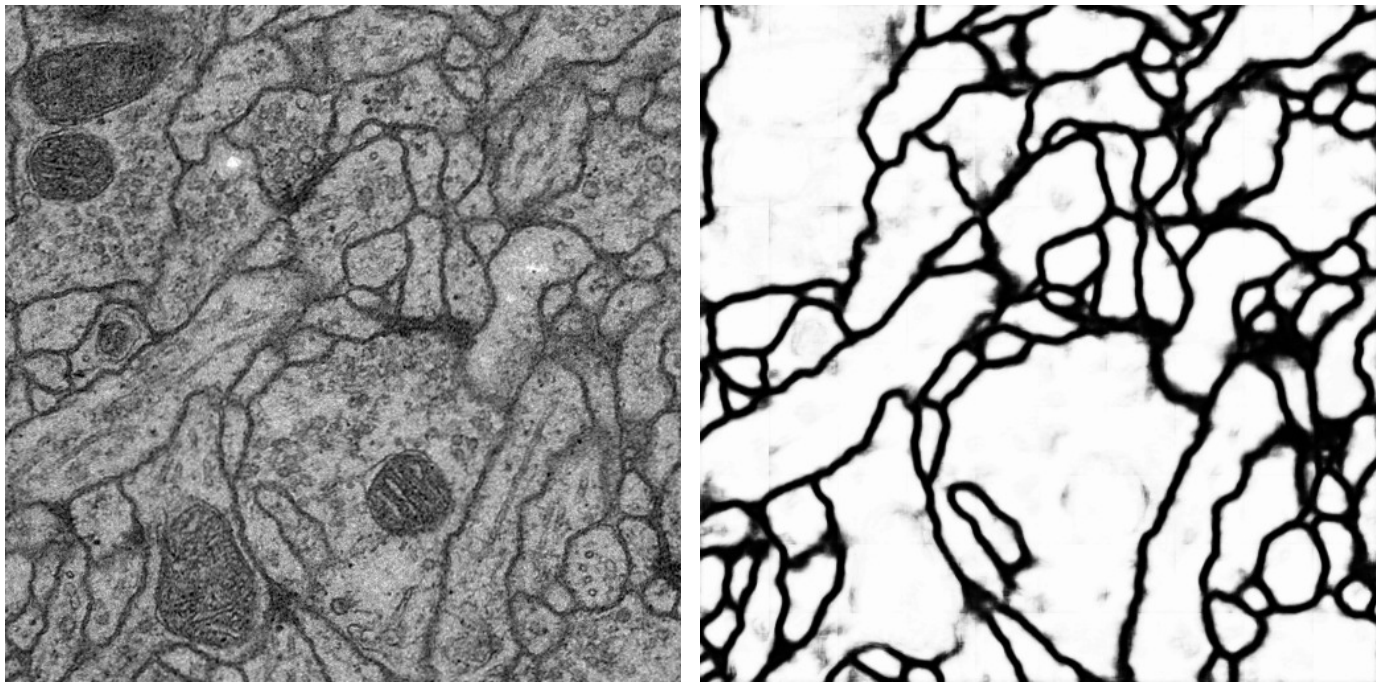
- PyraMiD LSTM[84]
- The context is rotated by 45 degrees
- Fast parallelization on GPUs

[84] M. Stollenga, W. Byeon, M. Liwicki, *et al.*, "Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation," *CoRR*, vol. abs/1506.07452, 2015. [Online]. Available: http://arxiv.org/abs/1506.07452.

PyraMiD LSTM



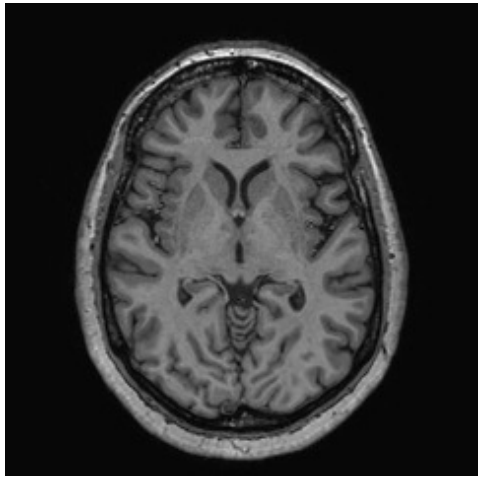
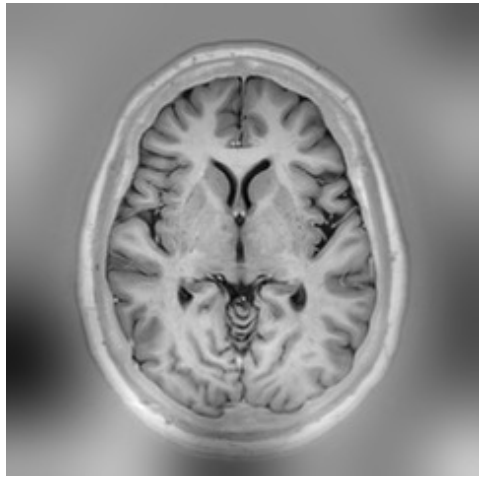
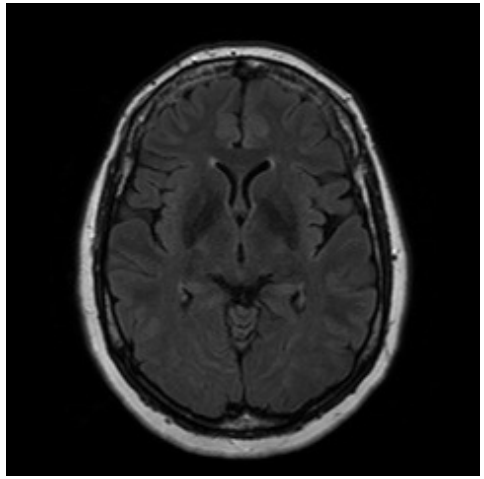
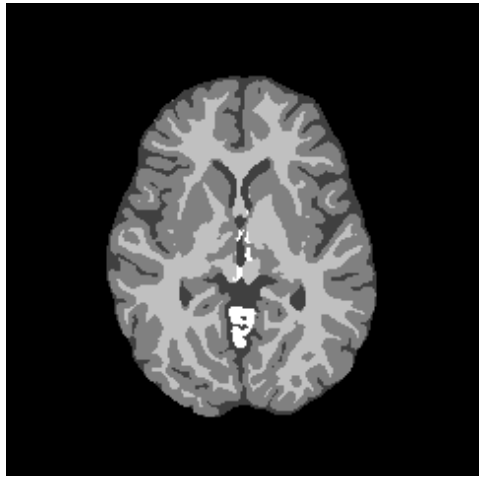
Segmentation with PyraMiD LSTM



- Electron microscopy (EM) dataset[85]

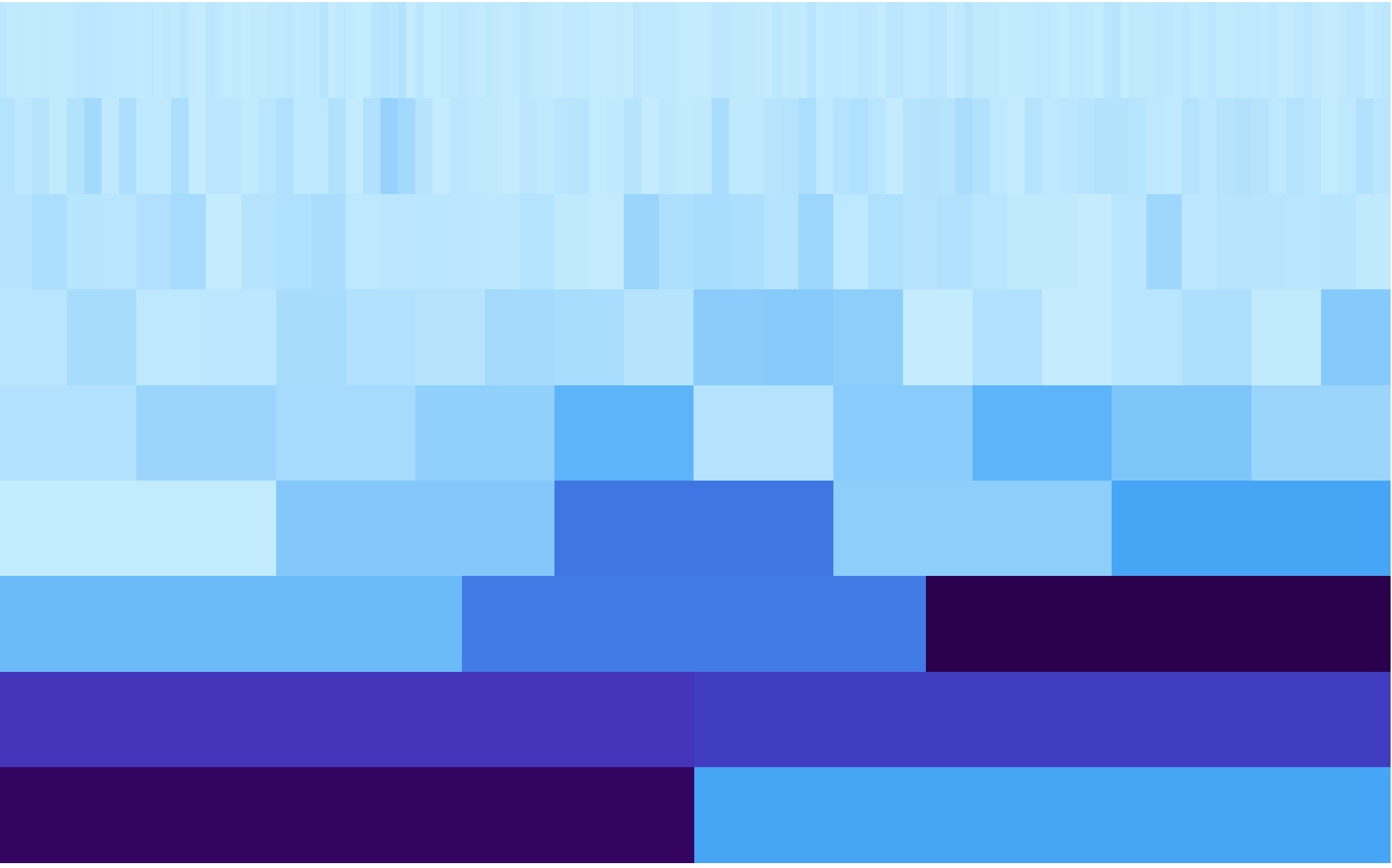
[85] A. Cardona, S. Saalfeld, S. Preibisch, *et al.*, "V.: an integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy," Tech. Rep., 2010.

Segmentation with PyraMiD LSTM

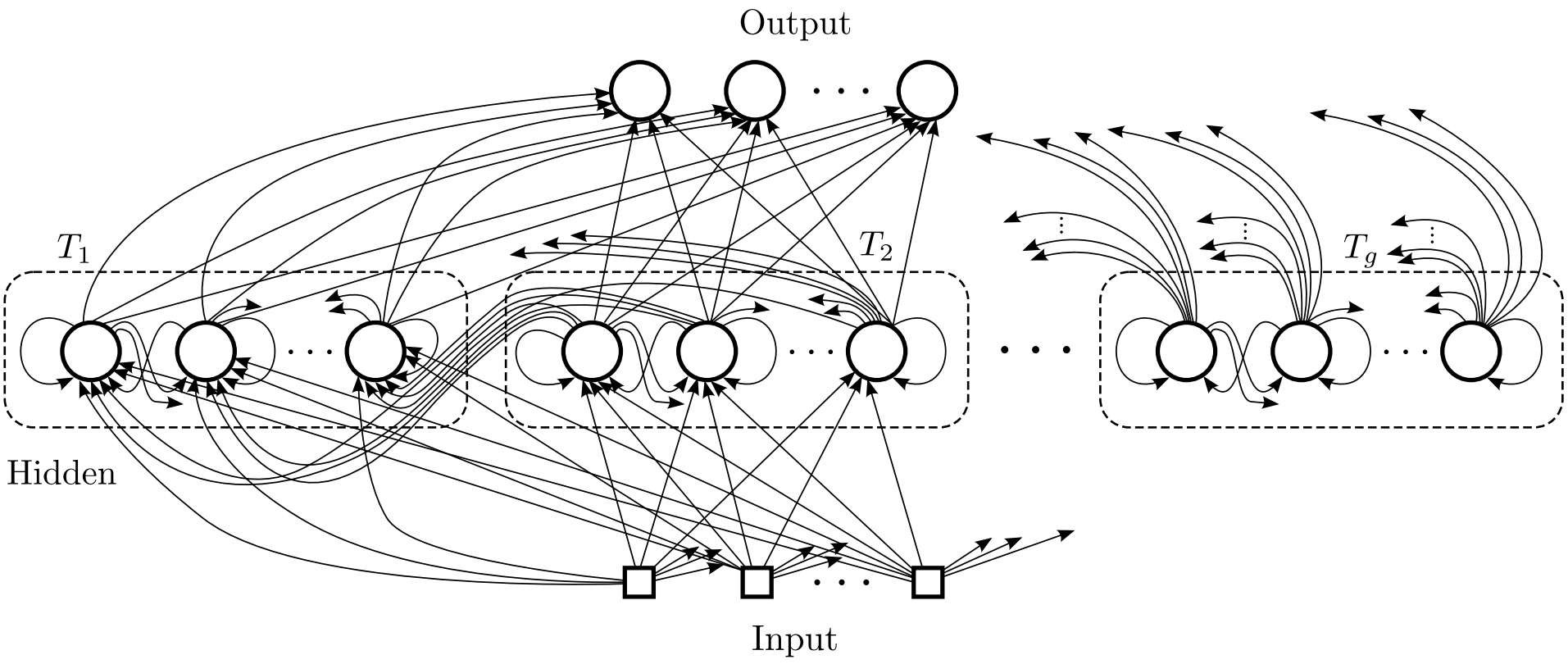





- MR dataset[86]

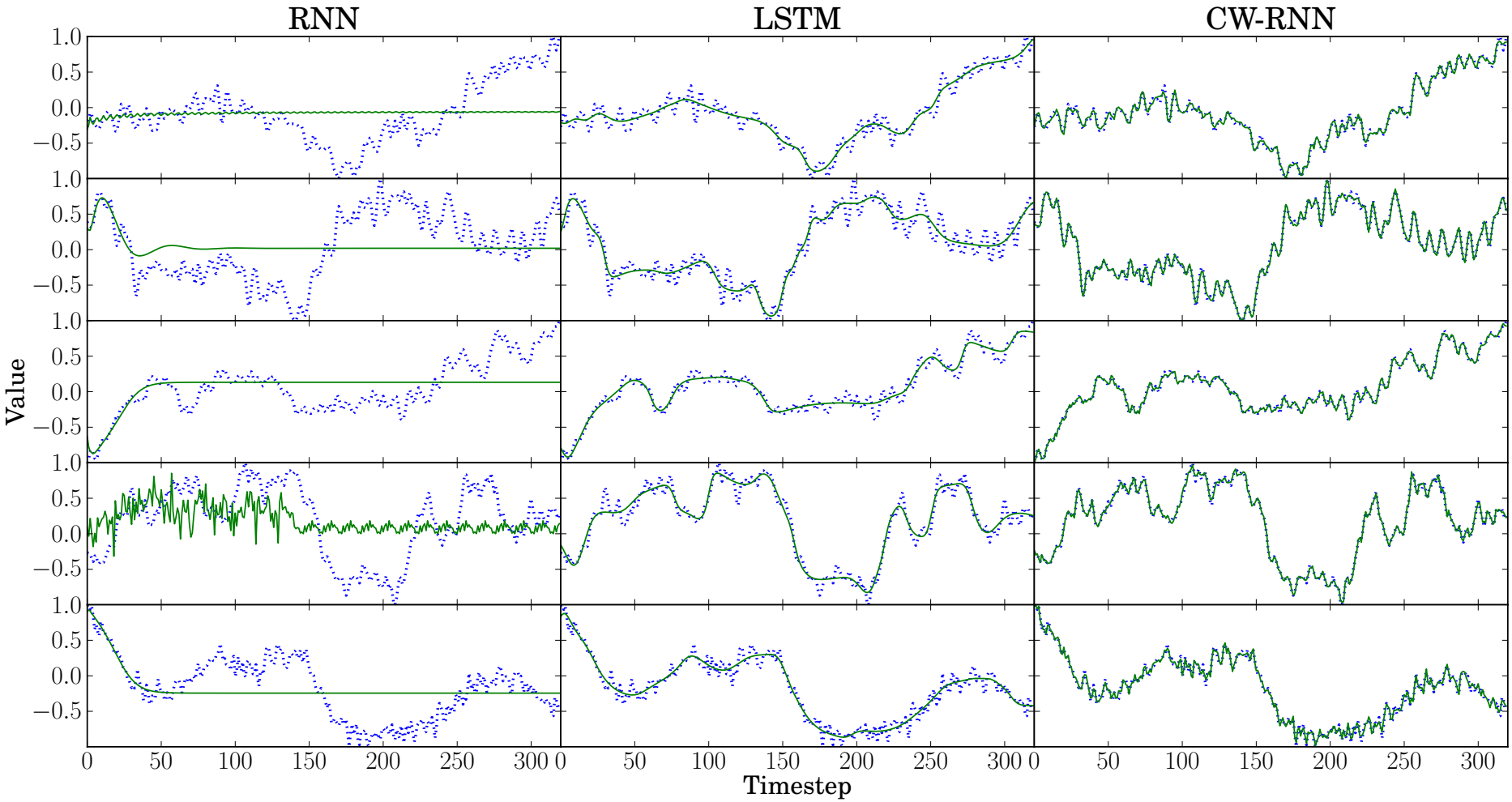
[86] A. Mendrik, K. Vincken, H. Kuijf, *et al.*, "MRBrainS Challenge: Online Evaluation Framework for Brain Image Segmentation in 3T MRI Scans," *Computational Intelligence and Neuroscience, special issue on Simulation and Validation in Brain Image Analysis*, vol. in press, 2015.



A Clockwork RNN



Sequence Generation with Clockwork RNN



- Better than LSTM in speech keyword recognition[87]
- Precise sequence generation

[87] J. Koutník, K. Greff, F. Gomez, *et al.*, "A Clockwork RNN," , in *Proceedings of the 31th International Conference on Machine Learning (ICML)*, arXiv:1402.3511 [cs.NE], vol. 32, 2014, pp. 1845–1853.

Wait, is the vanishing gradient problem for really deep feed-forward networks?

Highway Networks

- Vanishing gradient is relevant for feed-forward NNs[88,89]

[88] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *CoRR*, vol. abs/1505.00387, 2015. [Online]. Available: http://arxiv.org/abs/1505.00387.

[89] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," *CoRR*, vol. abs/1507.06228, 2015. [Online]. Available: http://arxiv.org/abs/1507.06228.

Highway Networks

- Standard feed-forward NN

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H})$$

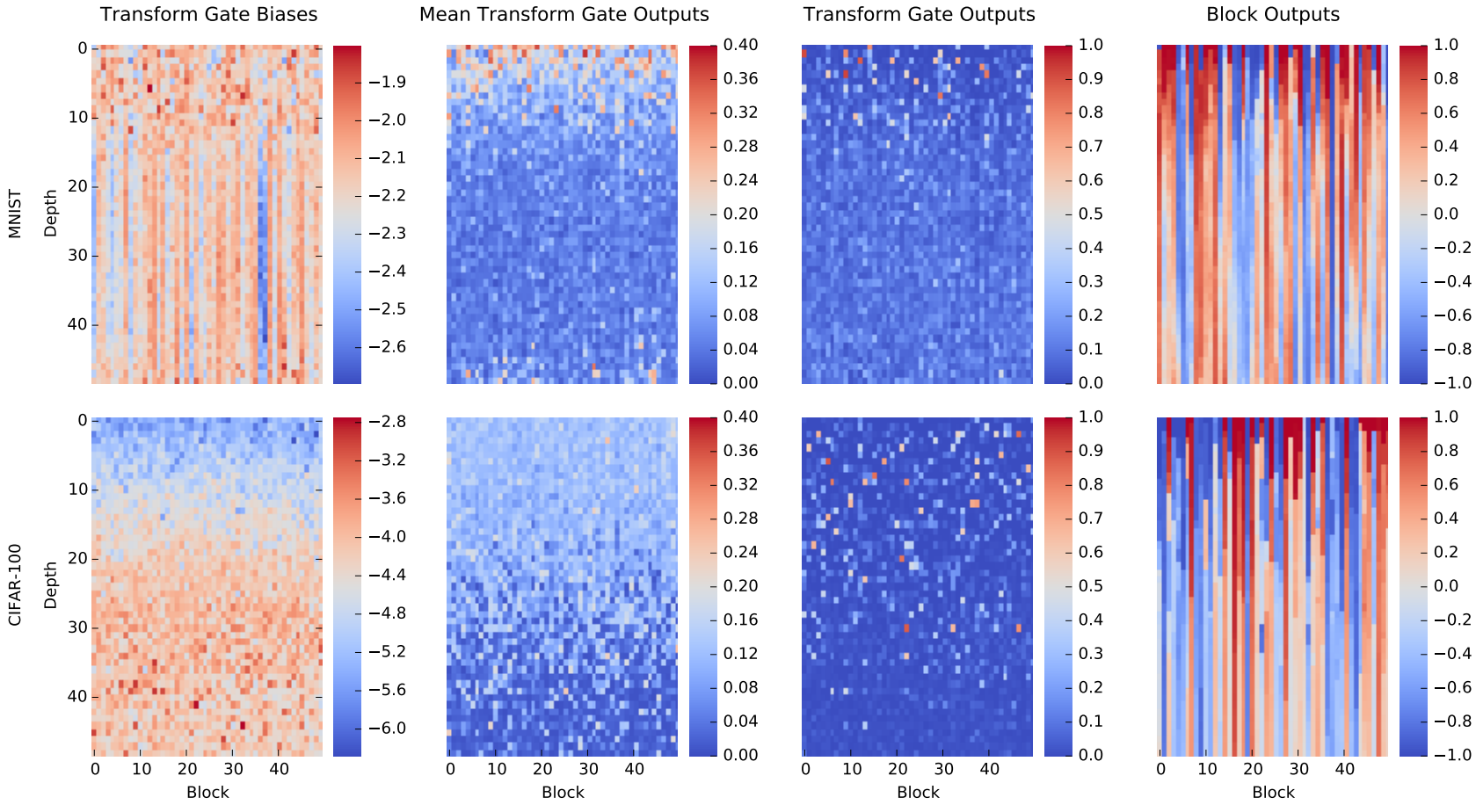
- Higway NN

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T}))$$

$$T(\mathbf{x}) = \sigma({\mathbf{W_T}}^T \mathbf{x} + \mathbf{b_T})$$

Highway Networks

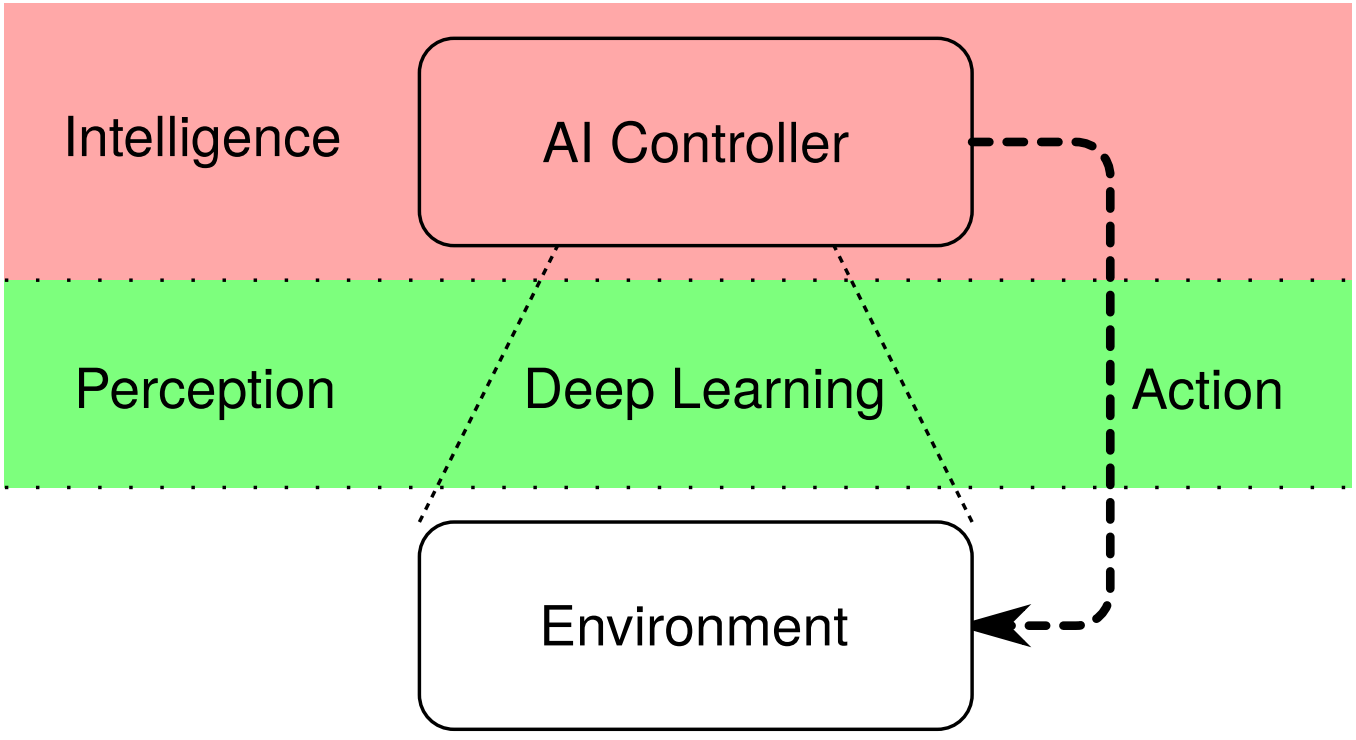


Conclusion

- Large static data set (e.g. images)
  $\rightarrow$ Large Convolutional Neural Network
- Sequential Dataset (e.g. Video, Speech, Text)
  $\rightarrow$ Long Short-Term Memory

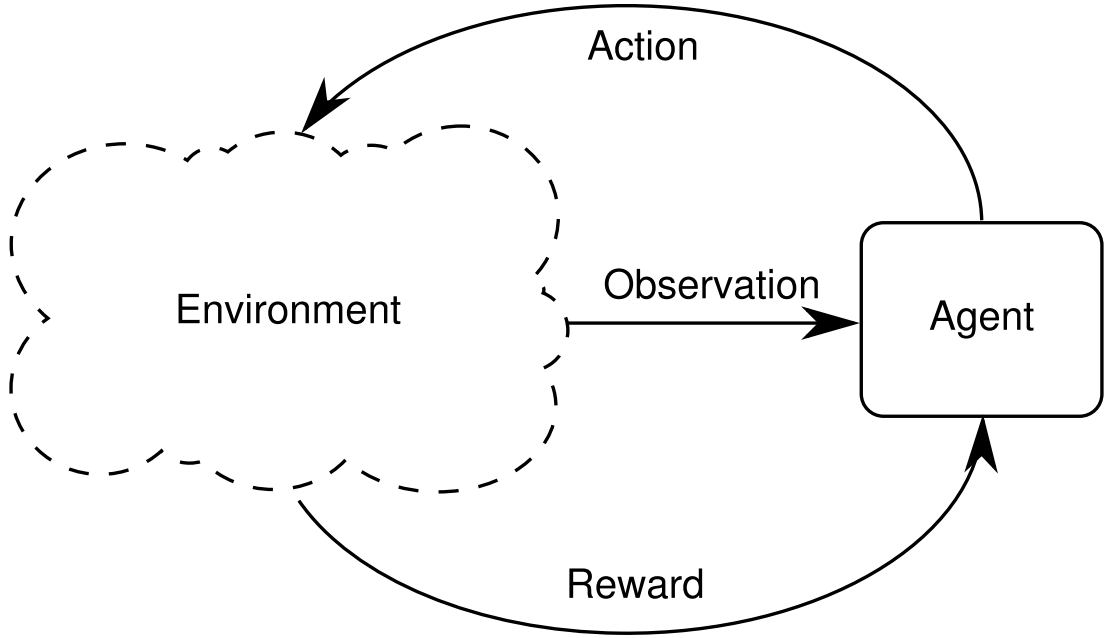
Reinforcement Learning

The Real Challenge



- Some see RL as an application of DL
- DL is a tool that RL can use to cope with the real world complexity
- The real *intelligence* needs to involve RL, not DL only

Reinforcement Learning



An agent maximizes cumulative expected reward through interactions with an environment

- **R**einforcement **L**earning – collection of classic methods[90]
- reinforcement learning – the problem

[90] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

reinforcement learning

Value Based Methods

- Each state s is assigned a value V
- How much reward can agent expect in the future, following policy π from state s

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}, \text{where}$$

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, 0 \leq \gamma \leq 1$$

Bellman Equation

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\
&= E_{\pi}\left\{ \sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s \right\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma E_{\pi}\left\{ \sum_{k=0}^{T} \gamma^k r_{t+k+2} | s_{t+1} = s \right\} \right] \\
&= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]
\end{aligned}
$$

Temporal Difference Methods

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- Value difference between the current and next visited state used for updates
- Does not need values of all states (only the visited ones)

TD-Control – Q-Learning

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- Q-value:[91] defined for (s_t, a_t) pair
- Uses *best* action in the next state
- Values for each action exists – can be used for online learning/control

[91] A. G. Barto, R. Sutton, and C. Watkins, "Learning and sequential decision making," University of Massachusetts, Amherst, Tech. Rep. COINS Technical Report 89-95, 1989.

Continuous State Space

- Approximate the Q-values
- Using e.g. a neural network
- Neural Fitted Q-iteration (NFQ)[92]
  - Input: (s_t, a_t)
  - Target: $r_{t+1} + \gamma max_a Q(s_{t+1}, a)$

[92] M. Riedmiller, "Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method," in *Proc. ECML-2005*, Springer-Verlag Berlin Heidelberg, 2005, pp. 317–328.

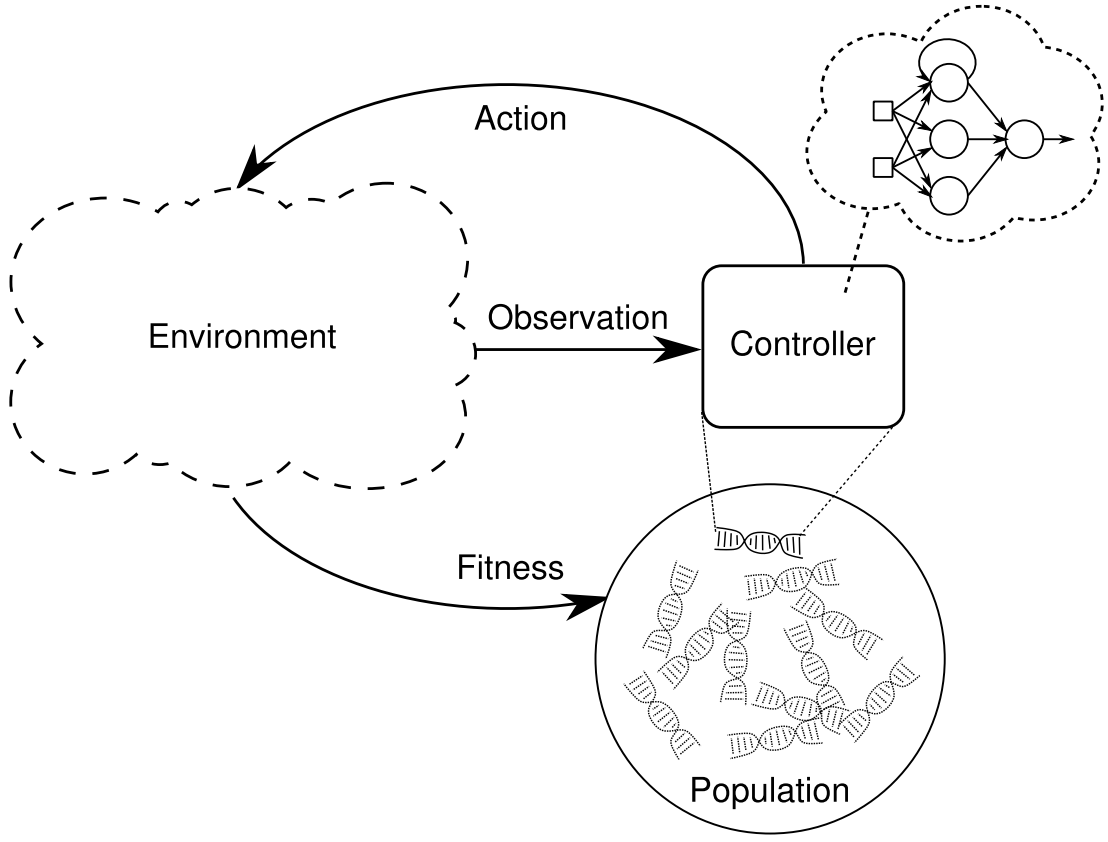
DQN

- DQN[93] = NFQ (2005) + Experience Replay[94] + Convolutional Networks with BP (1998)
- DQN Atari = DQN (2010) + Computing Power (2013)

[93] S. Lange and M. Riedmiller, "Deep learning of visual control policies," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2010.

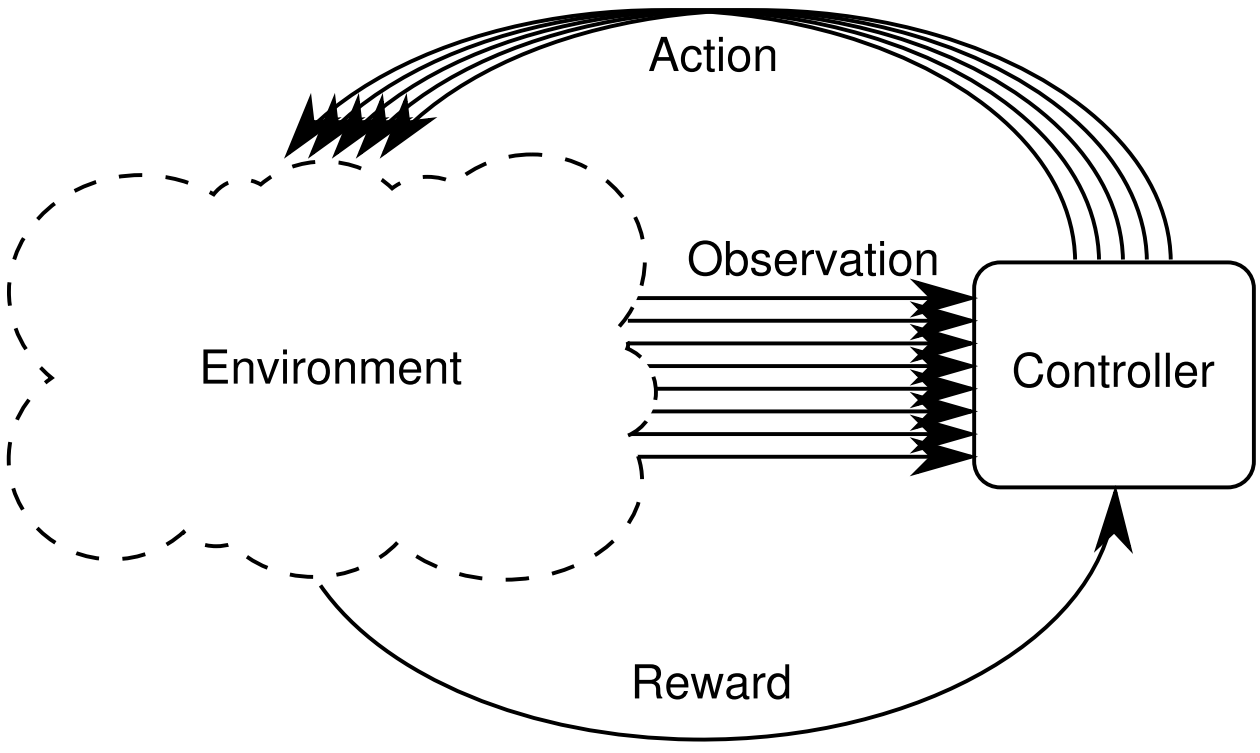
[94] L. Lin, "Self-improving reactive agents: case studies of reinforcement learning frameworks," in *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, J. A. Meyer and S. W. Wilson, Eds., MIT Press/Bradford Books, 1991, pp. 297–305.

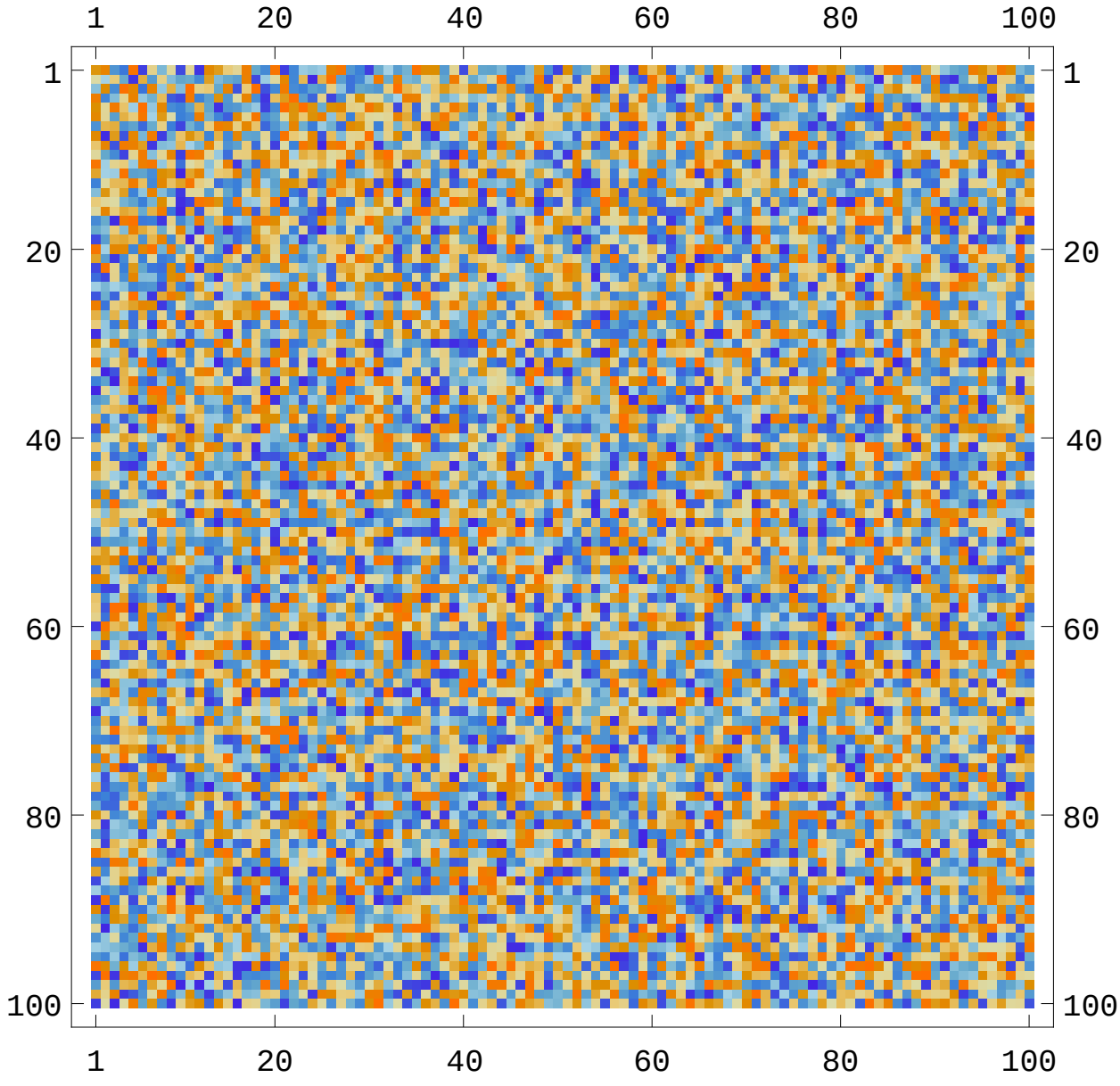
Neuroevolutionary RL

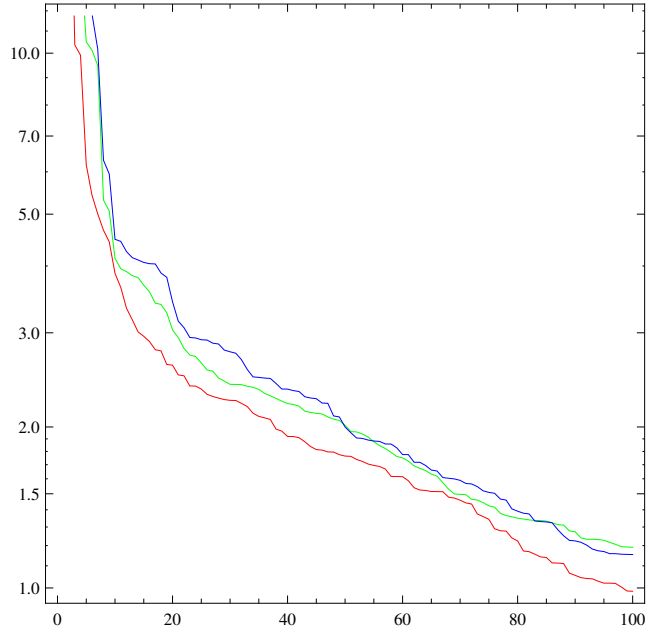


- Alternative Approach
- Naturally deals with continuous observation and **action** spaces
- Policy executed by a RNN (can have memory)
- Evolved (no need for a gradient)

Large Observation/Action Space

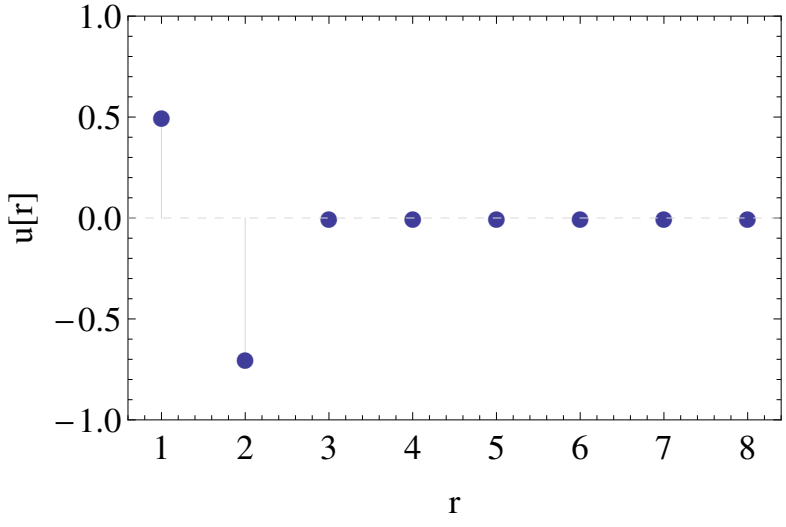


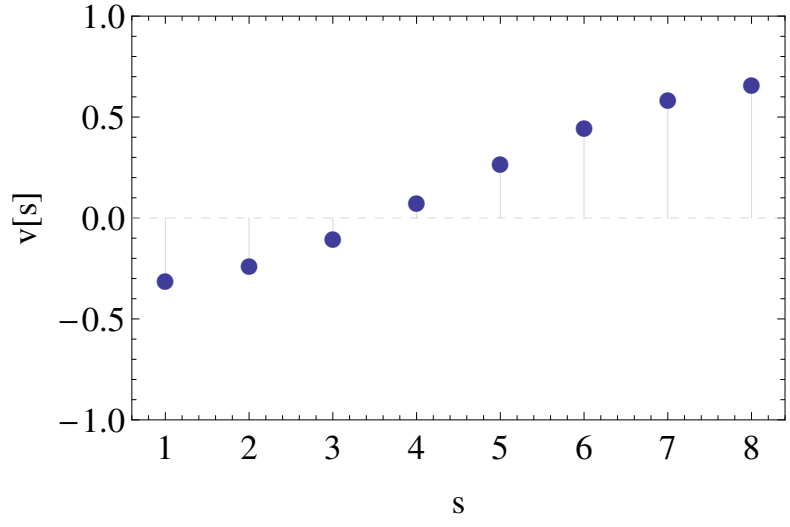

1
20
40
60
80
100
1
20
40
60
80
100
1
20
40
60
80
100
1
20
40
60
80
100

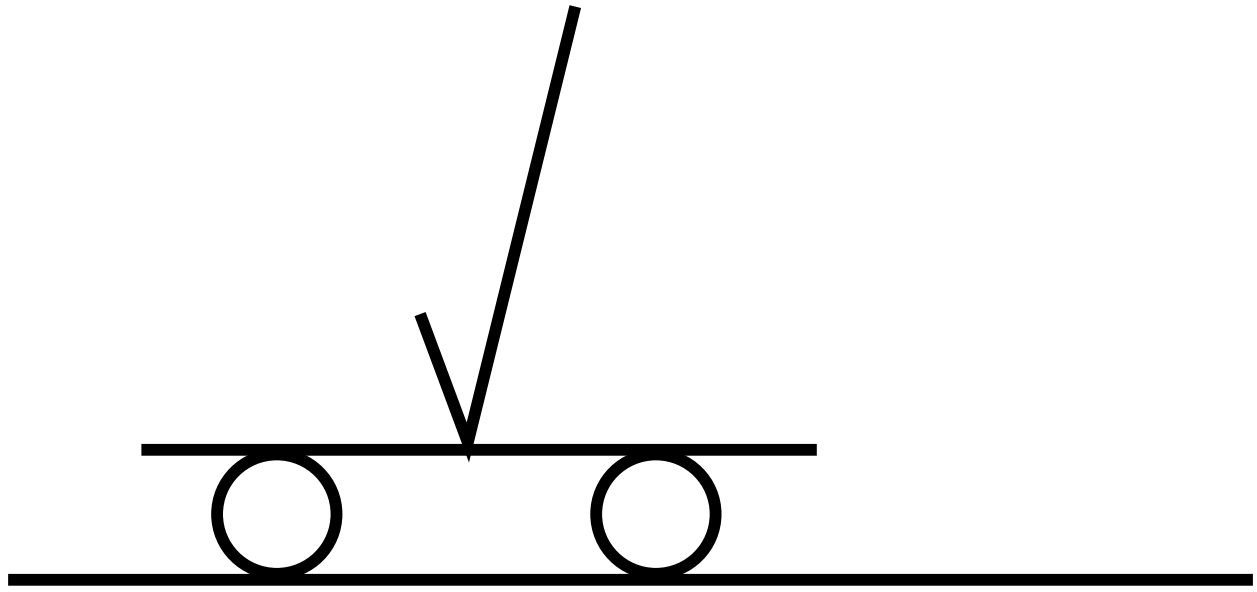

10.0
7.0
5.0
3.0
2.0
1.5
1.0
0
20
40
60
80
100

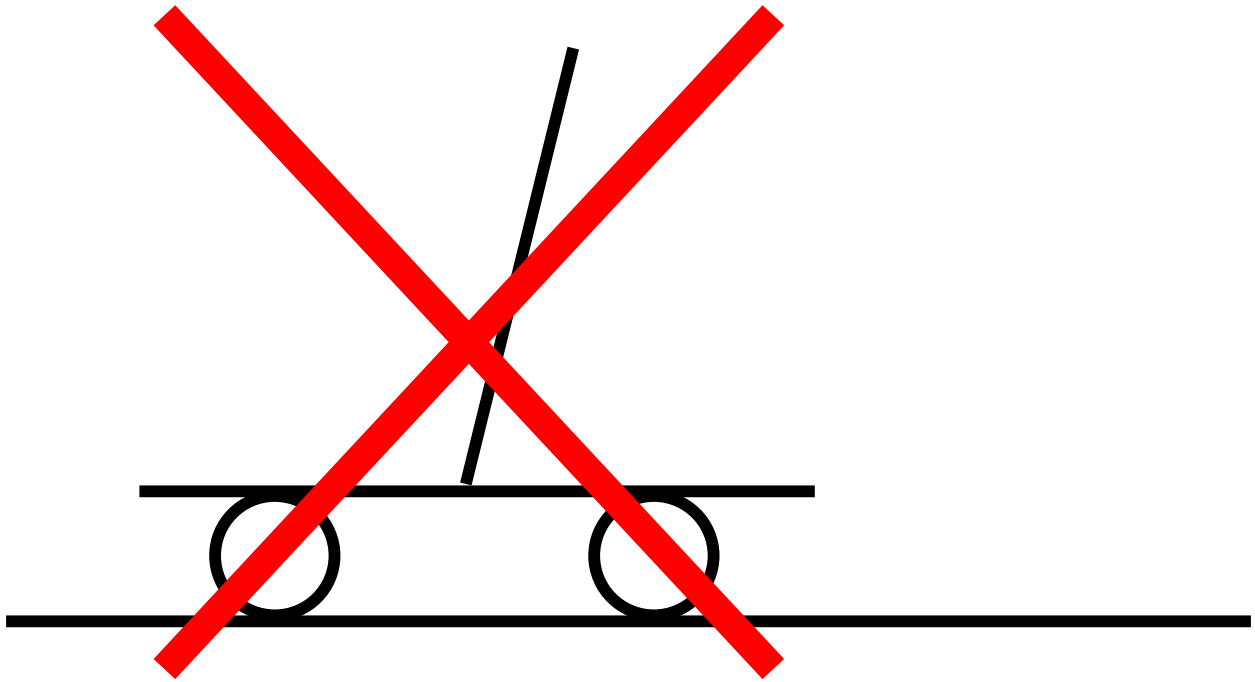
$$v[s] = \frac{1}{\sqrt{n}}\left[u[1] + 2\sum_{r=2}^{n} u[r]\cos\left(\frac{\pi}{n}(r-1)\left(s-\frac{1}{2}\right)\right)\right]$$

- FourierDCT[u,3]





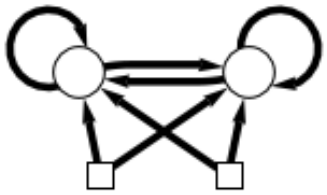


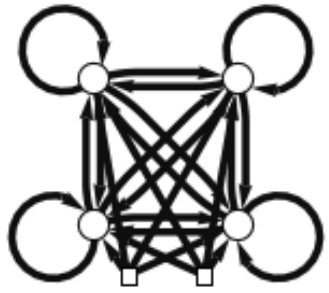


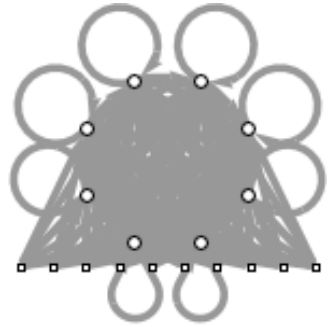
- Pole-balancing benchmark no longer exists[95]

[95] J. Koutník, F. Gomez, and J. Schmidhuber, "Searching for minimal neural networks in Fourier space," in *Proceedings of the 4th Annual Conference on Artificial General Intelligence (AGI)*, Atlantis Press, 2010.

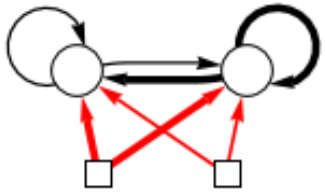
5

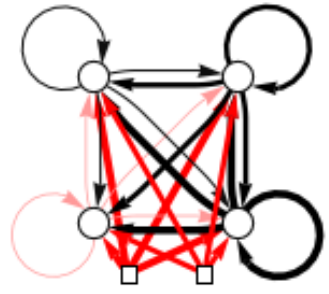




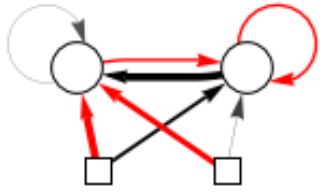


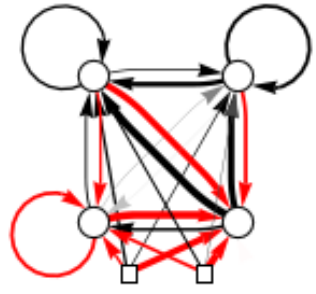
5	−8.2

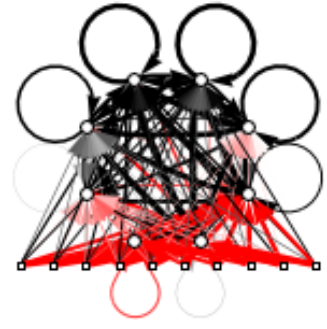


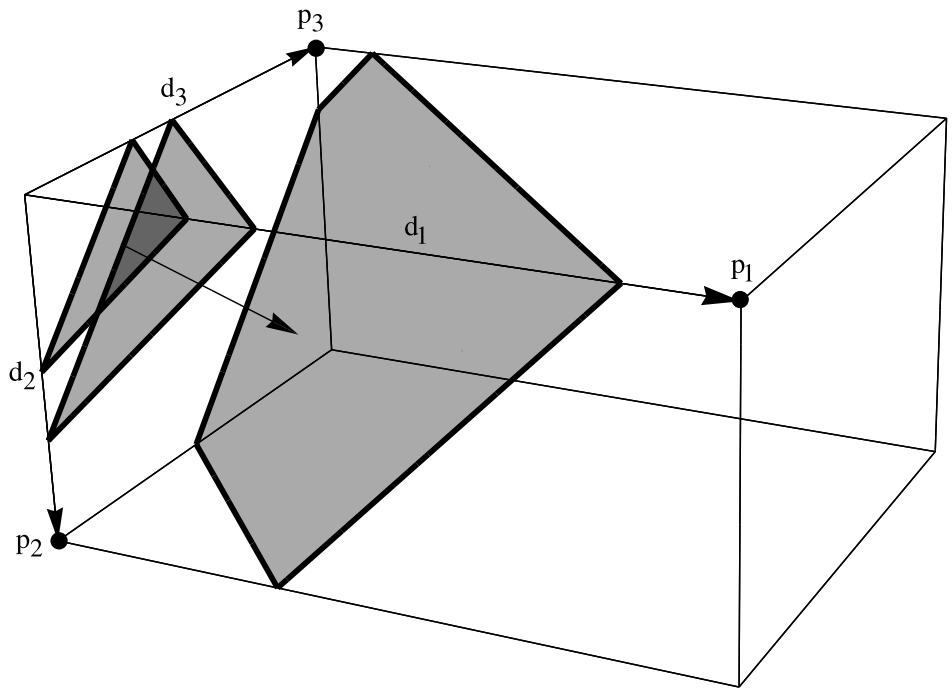


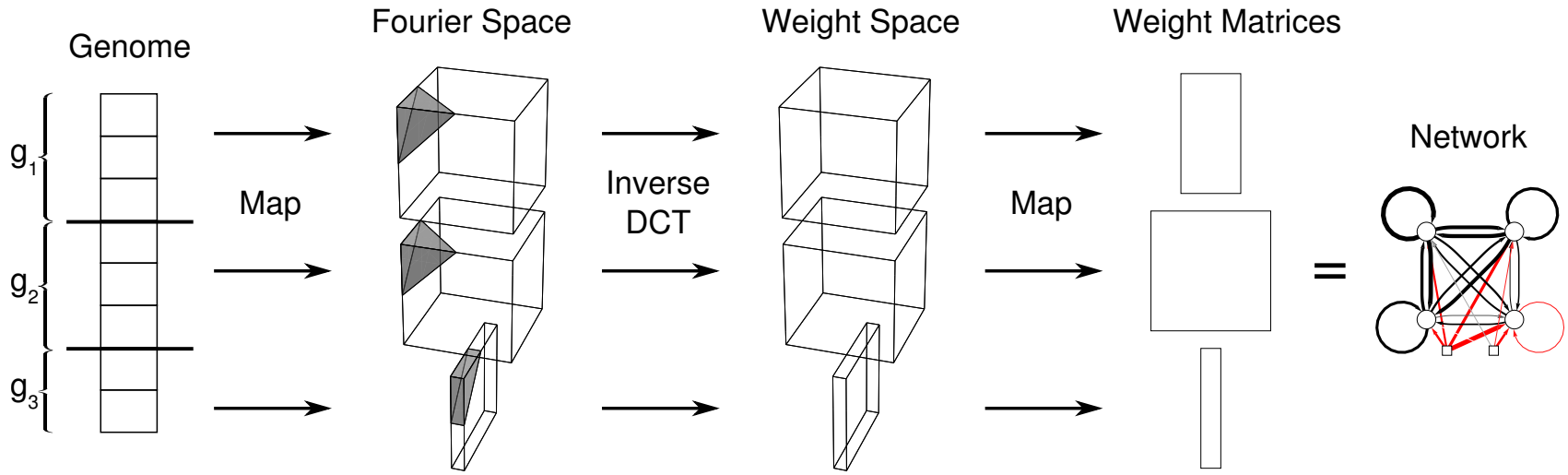
5	−8.2	3.8	4.1	−9








p3
d3
d1
p1
d2
p2



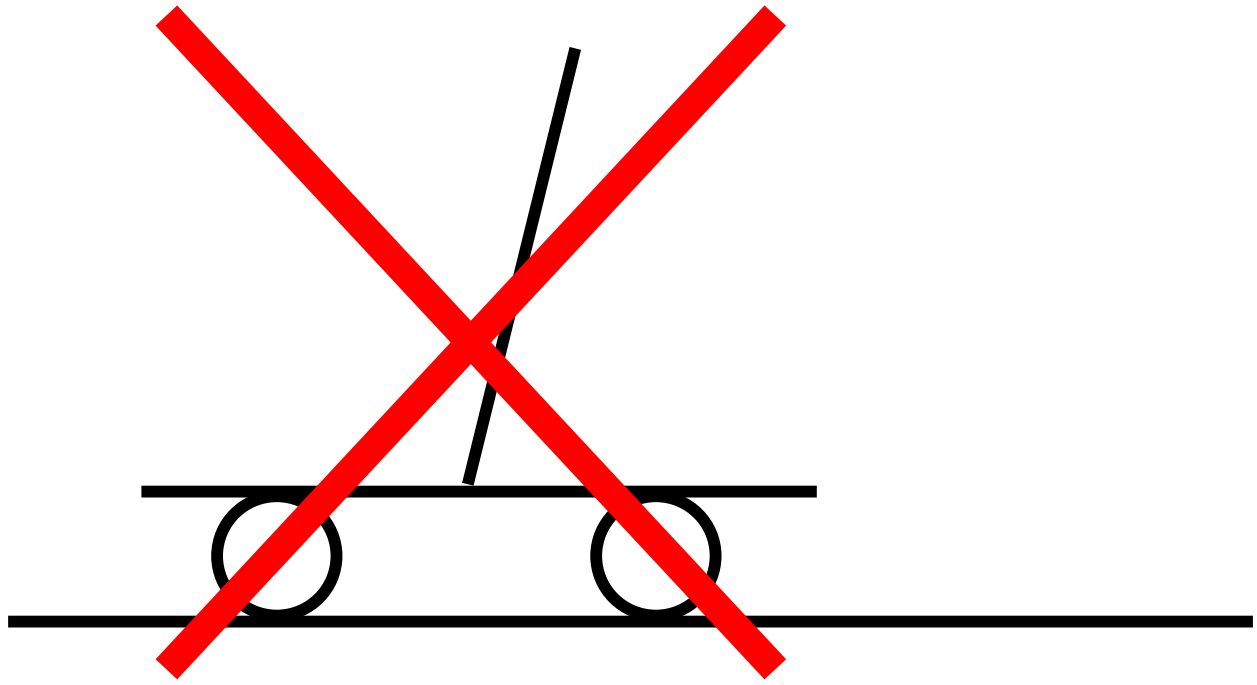
- *Compressed Neural Networks*[96]
  - Compression in weight precision
- Generalized Compressed NNs[97]
- Complexity search from Compressed NNs[98]

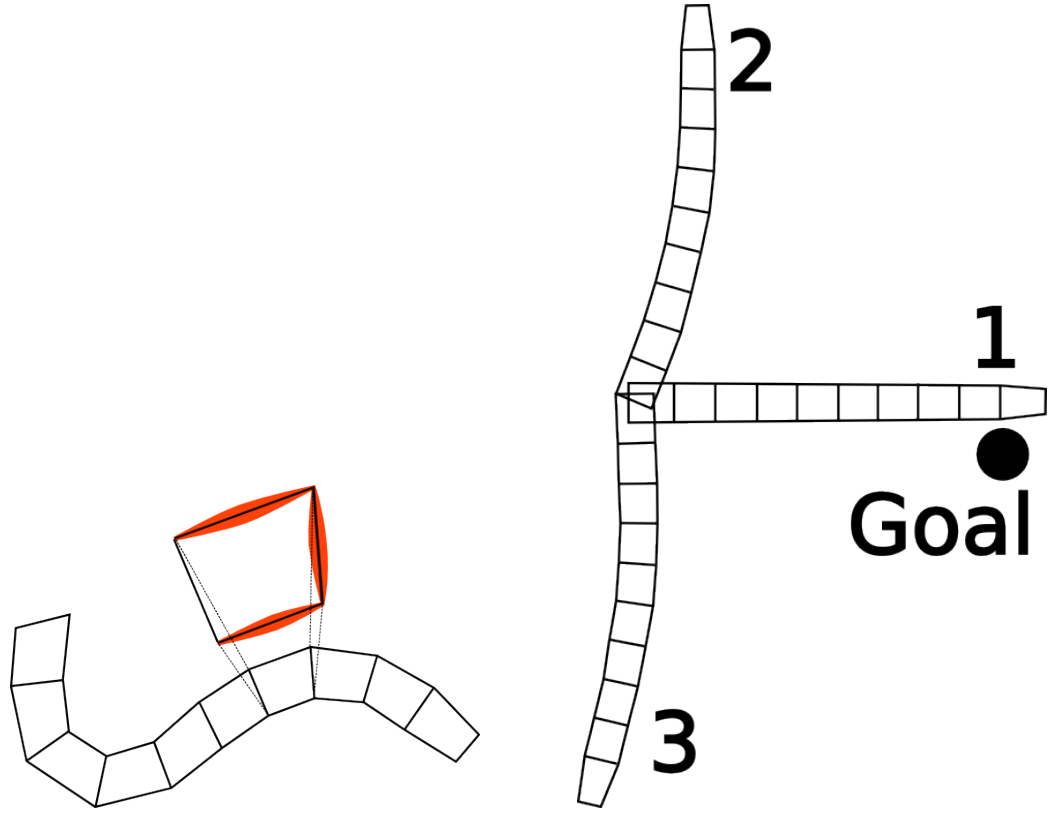
---

[96] J. Koutník, F. Gomez, and J. Schmidhuber, "Searching for minimal neural networks in Fourier space," in *Proceedings of the 4th Annual Conference on Artificial General Intelligence (AGI)*, Atlantis Press, 2010.

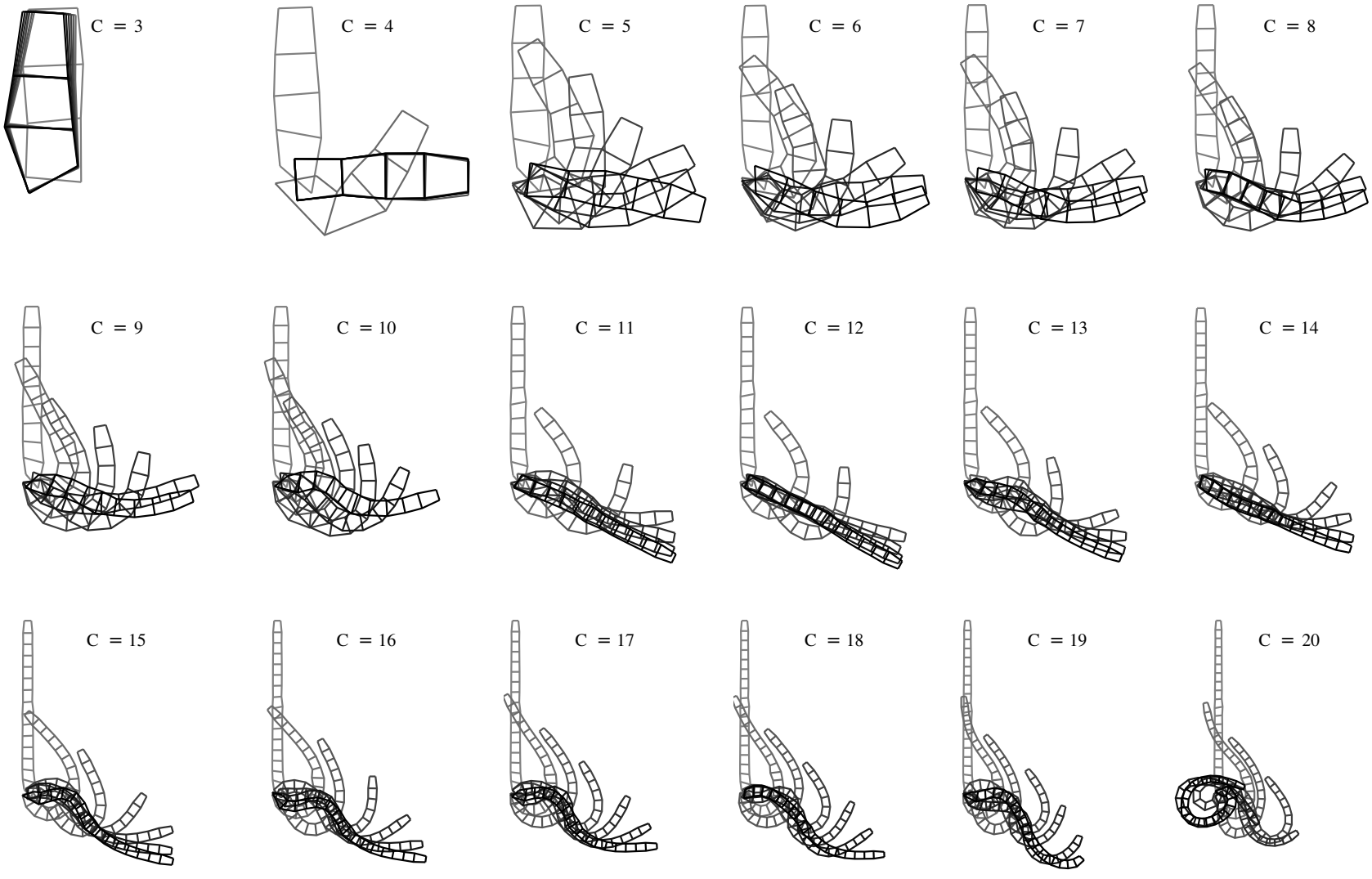
[97] R. K. Srivastava, J. Schmidhuber, and F. Gomez, "Generalized compressed network search," in *PPSN (1)*, 2012, pp. 337–346.

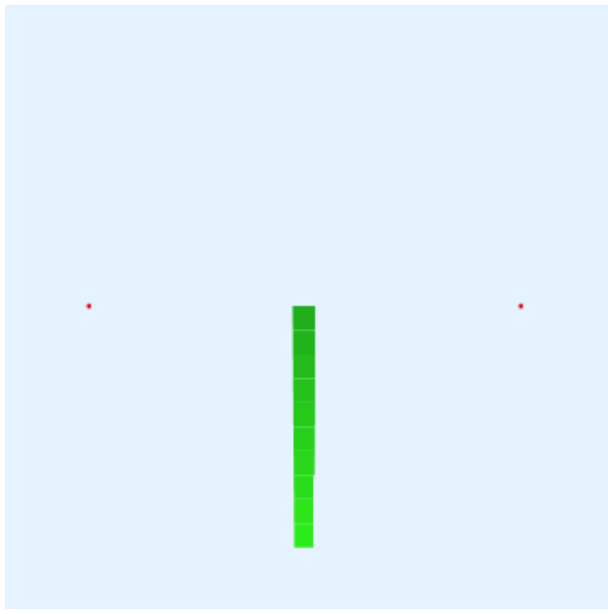
[98] F. Gomez, J. Koutník, and J. Schmidhuber, "Compressed networks complexity search," in *Parallel Problem Solving from Nature (PPSN)*, Springer, 2012, pp. 316–326.

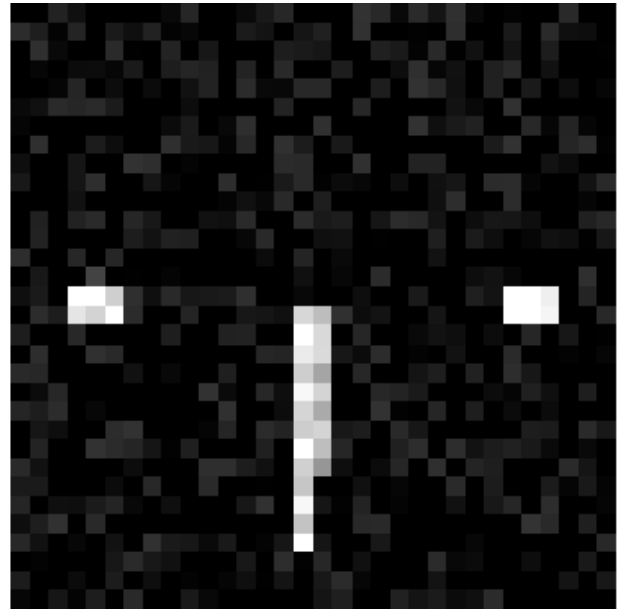




- RNN with 82 inputs and 8 meta-actions $\rightarrow$ 728 weights
- Compressed down to 20 DCT coefficients
- or 32 actions $\rightarrow$ 3680 weights (compression ratio 184:1)

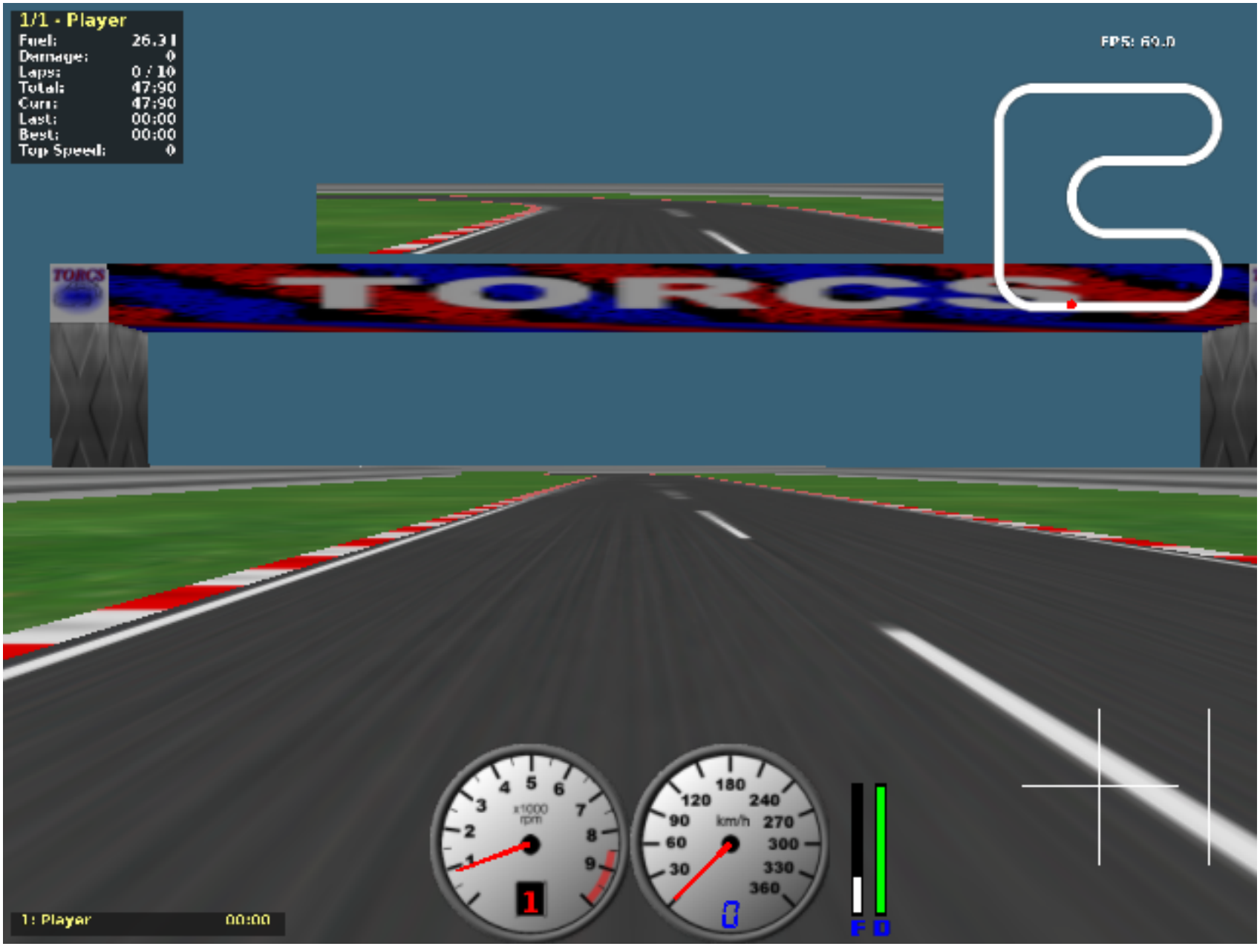

C = 3
C = 4
C = 5
C = 6
C = 7
C = 8
C = 9
C = 10
C = 11
C = 12
C = 13
C = 14
C = 15
C = 16
C = 17
C = 18
C = 19
C = 20





- RNN with 1024 inputs and 32 neurons $\rightarrow$ 33 824 weights
- Compressed down to 160 DCT coefficients (211:1)

1/1 - scr_server 1
Fuel: 93.5 l
Laps: 1 / 10
Best: 00:00 +00:00
Time: 25:61
<- --:--
-> --:--
FPS: 4.6
1 2 3 4 5 6 7 8 9
3
0 30 60 90 120 150 180 210 240 270 300 330 360
km/h
83
F D
1: scr_server 1 00:00

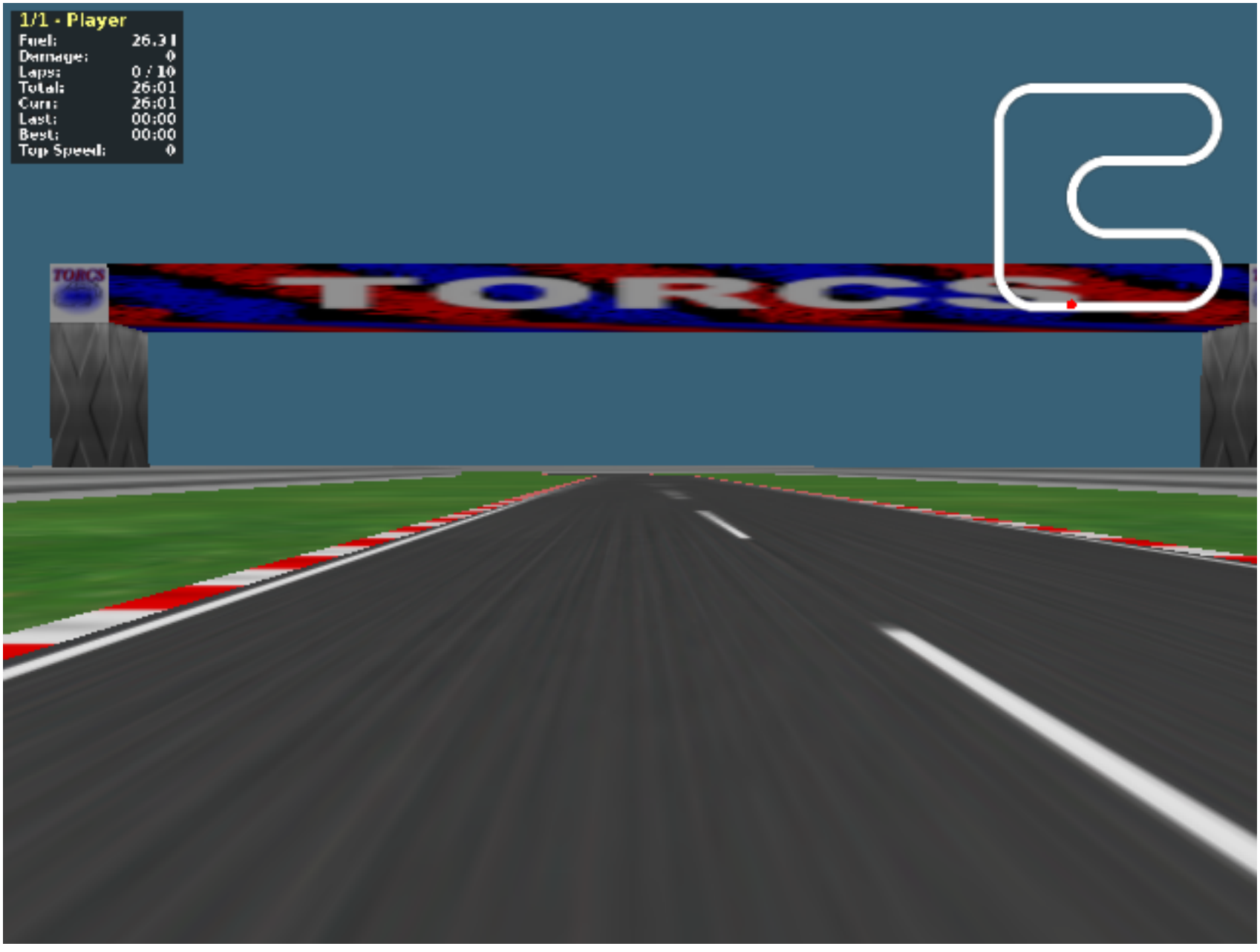

1/1 - Player
Fuel: 26.3 l
Damage: 0
Laps: 0 / 10
Total: 47:90
Curr: 47:90
Last: 00:00
Best: 00:00
Top Speed: 0
FPS: 60.0
TORCS
x1000 rpm
1
180
120
240
90
270
60
300
30
330
360
km/h
0
F
D
1: Player
00:00

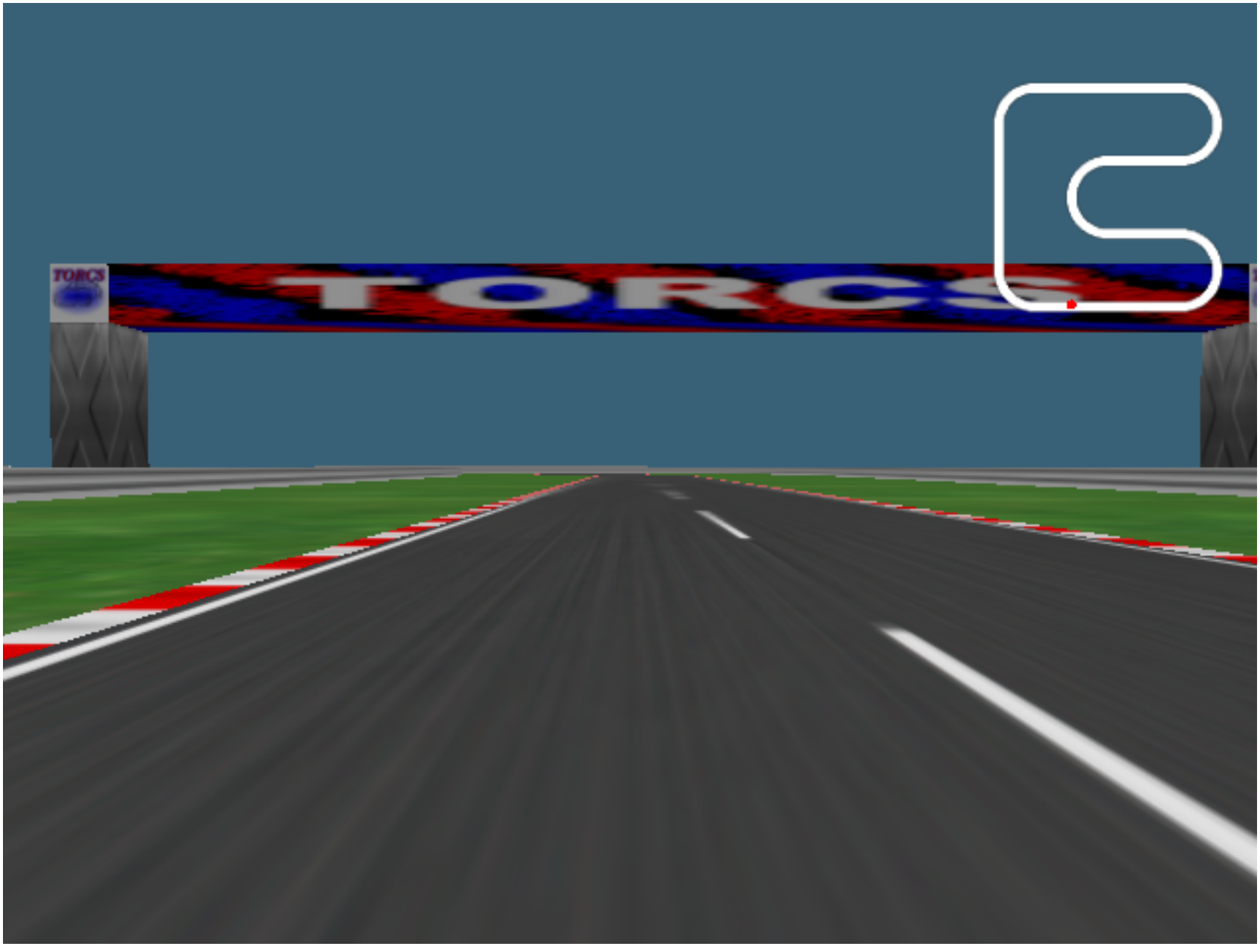
1/1 - Player
Fuel: 26.3 l
Damage: 0
Laps: 0 / 10
Total: 42:32
Curr: 42:32
Last: 00:00
Best: 00:00
Top Speed: 0
FPS: 49.9
TORCS
x1000 rpm
1 2 3 4 5 6 7 8 9
1
km/h
30 60 90 120 150 180 210 240 270 300 330 360
0
F D
1: Player 00:00

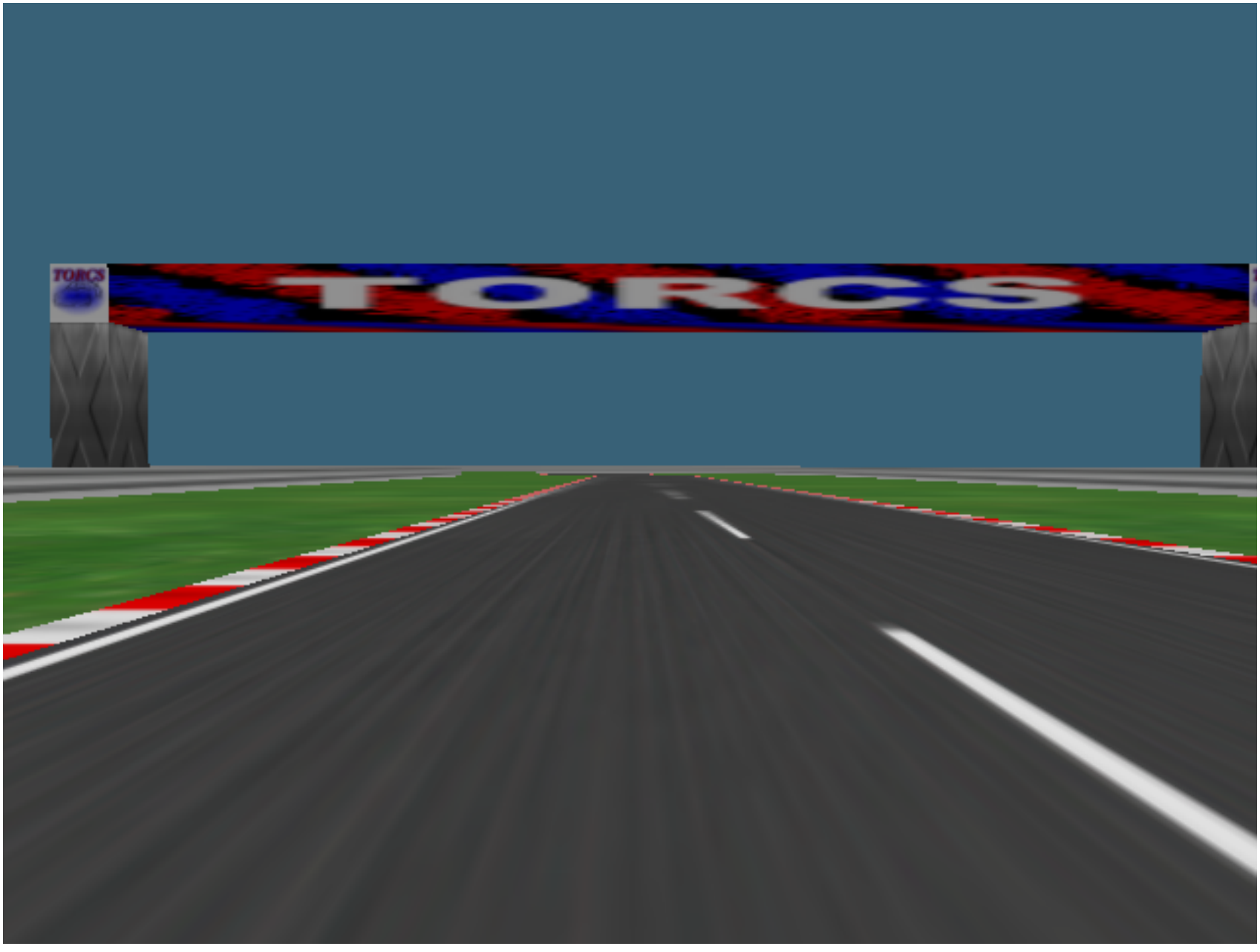
1/1 - Player
Fuel: 26.3 l
Damage: 0
Laps: 0 / 10
Total: 39:53
Curr: 39:53
Last: 00:00
Best: 00:00
Top Speed: 0
TORCS
x1000 rpm
1
km/h
0
F
D
1: Player 00:00

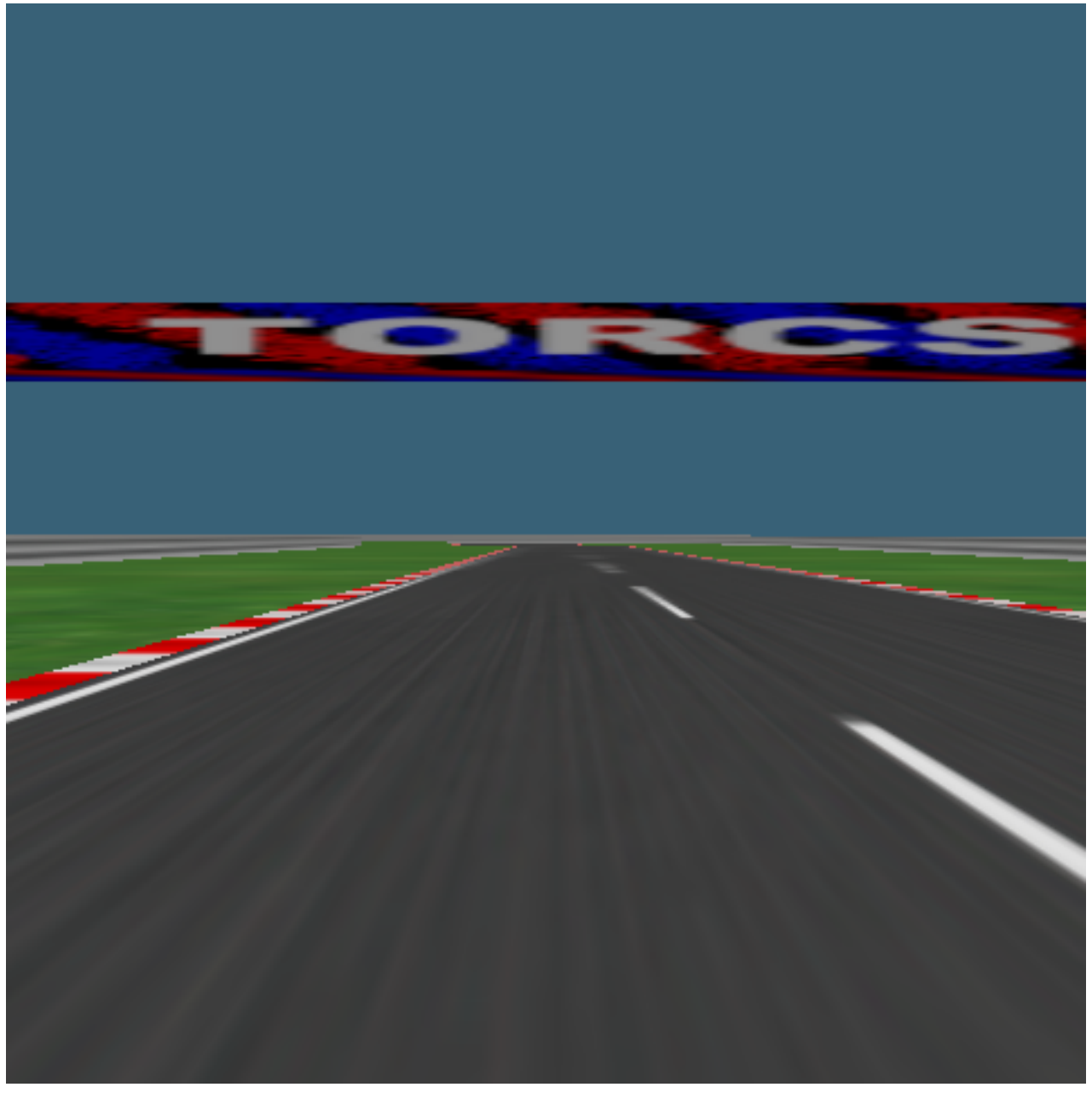
1/1 - Player
Fuel: 26.3 l
Damage: 0
Laps: 0 / 10
Total: 37:24
Curr: 37:24
Last: 00:00
Best: 00:00
Top Speed: 0
TORCS
x1000 rpm
1
km/h
0
F
D
1: Player 00:00

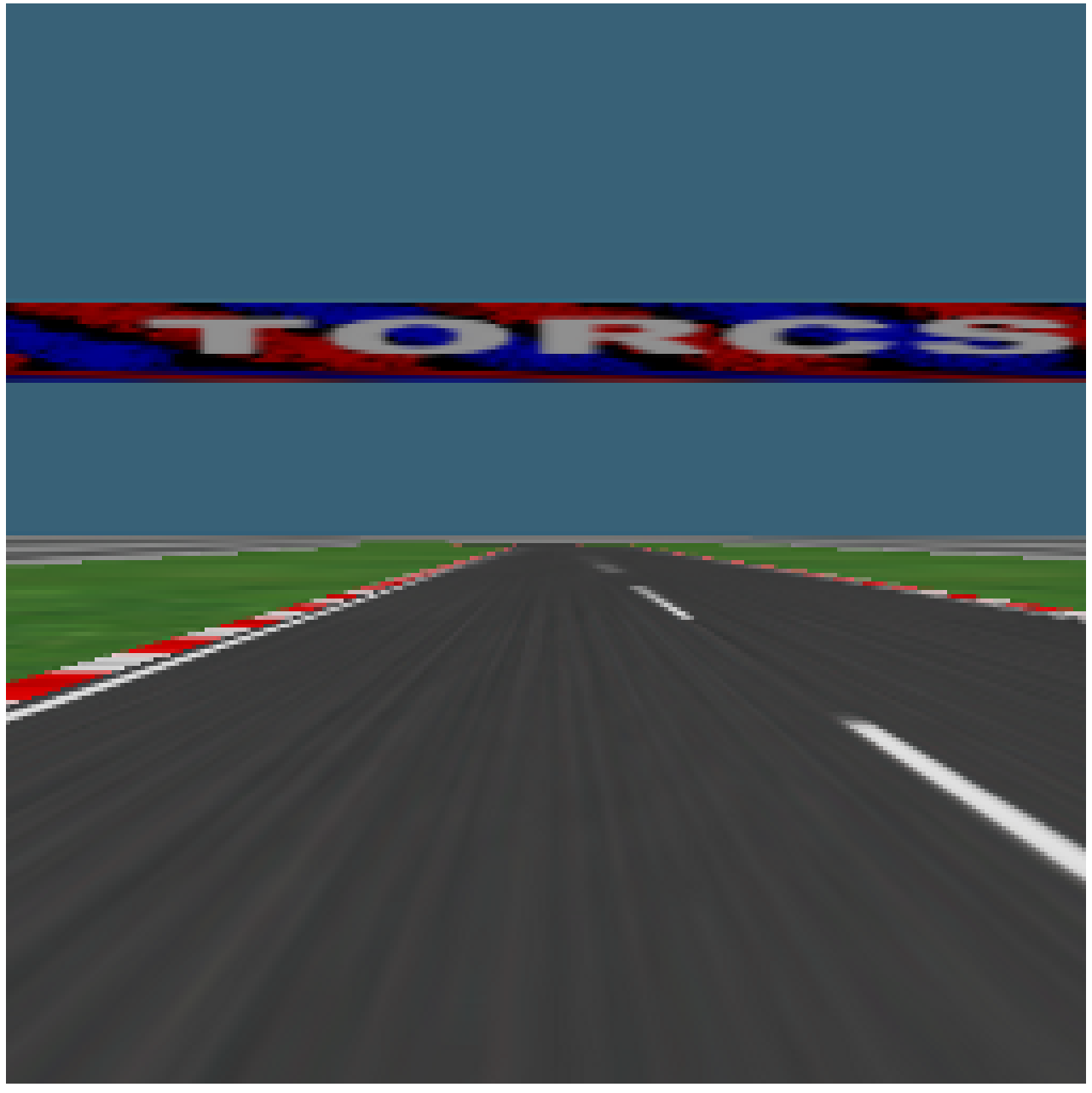
1/1 - Player
Fuel: 26.3 l
Damage: 0
Laps: 0 / 10
Total: 31:67
Curr: 31:67
Last: 00:00
Best: 00:00
Top Speed: 0
TORCS
x1000 rpm
1
2
3
4
5
6
7
8
9
km/h
30
60
90
120
180
240
270
300
330
360
0
F
D

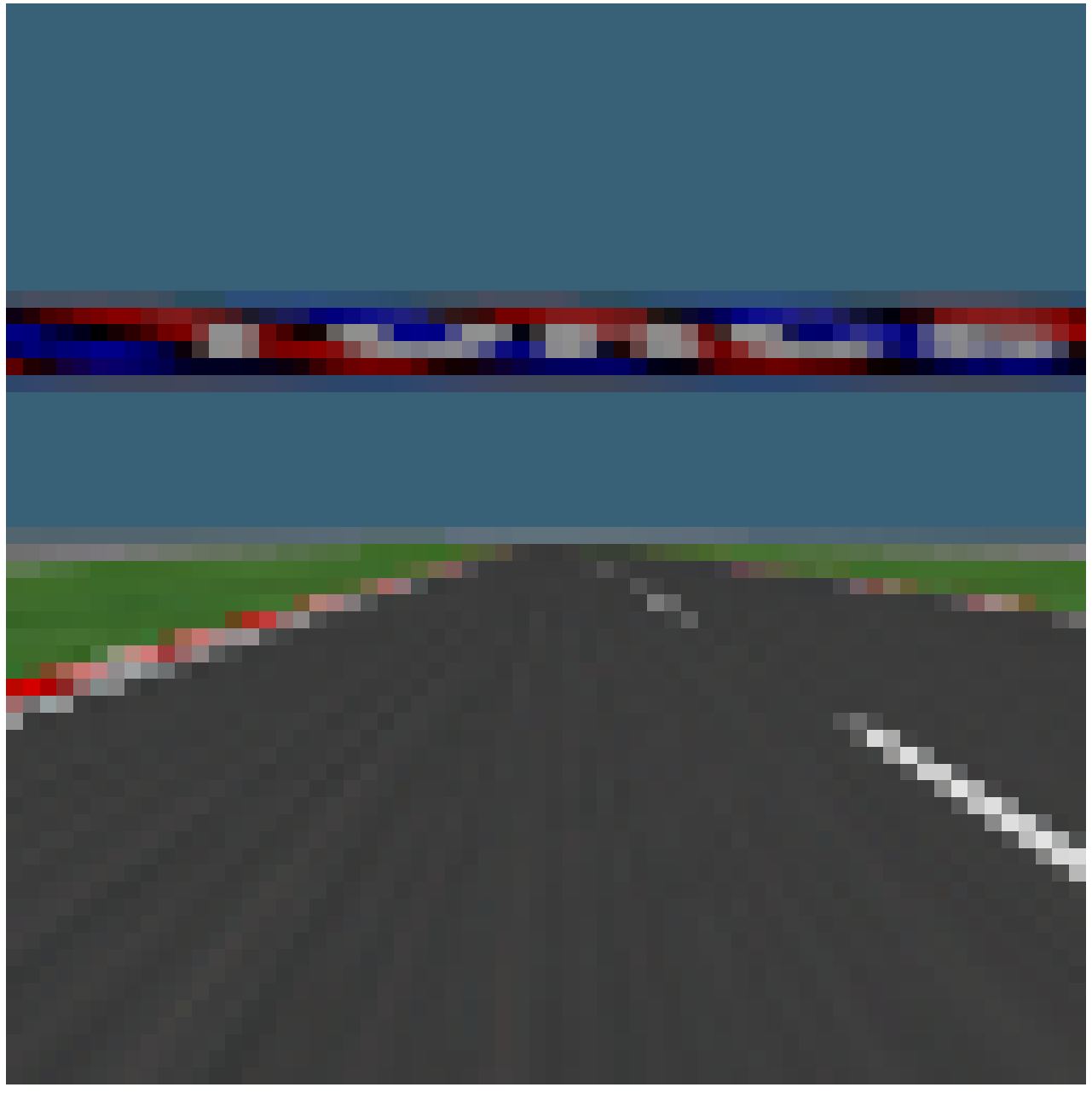

1/1 - Player
Fuel: 26.3 l
Damage: 0
Laps: 0 / 10
Total: 26:01
Curr: 26:01
Last: 00:00
Best: 00:00
Top Speed: 0
TORCS

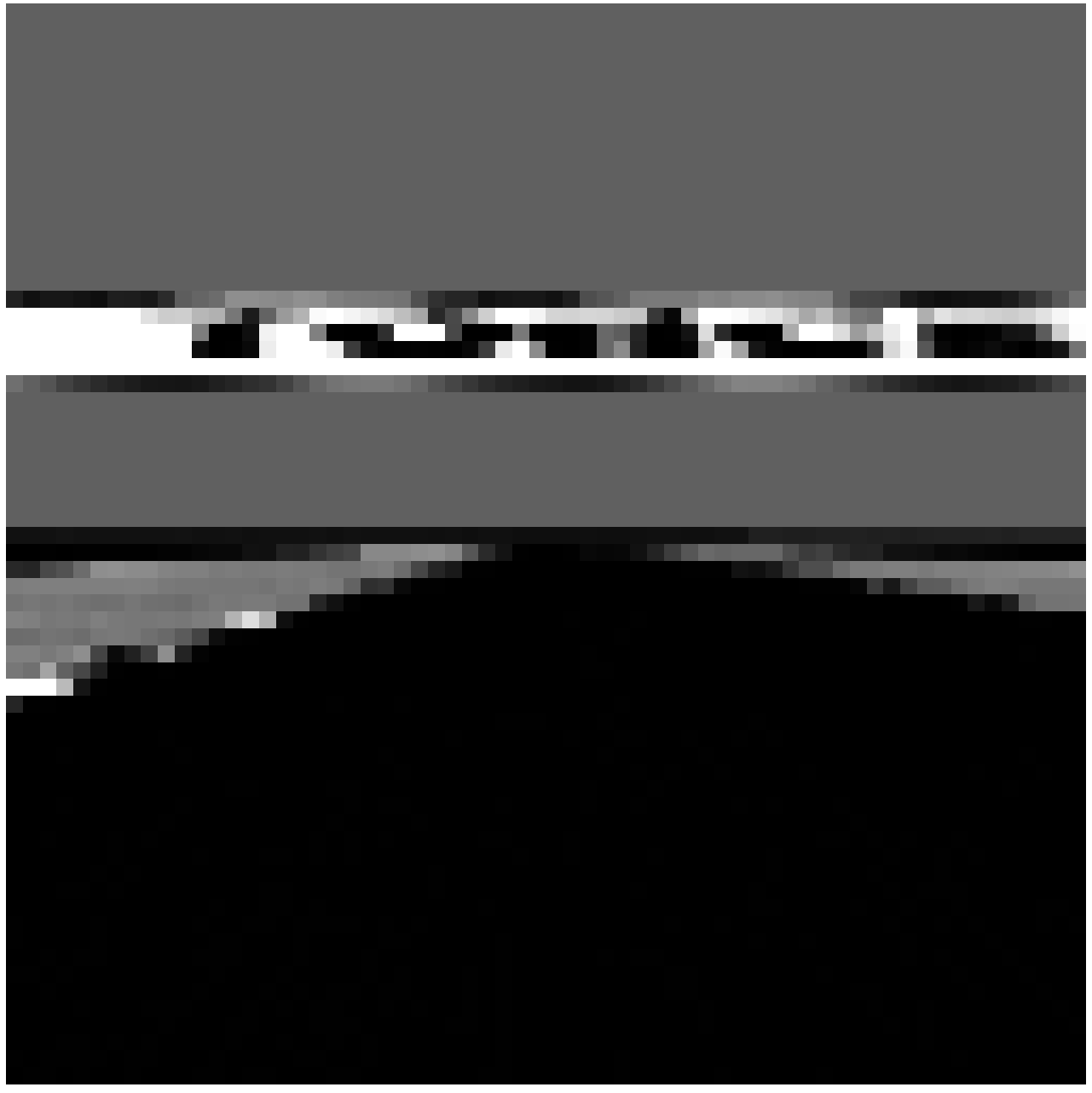

TORCS

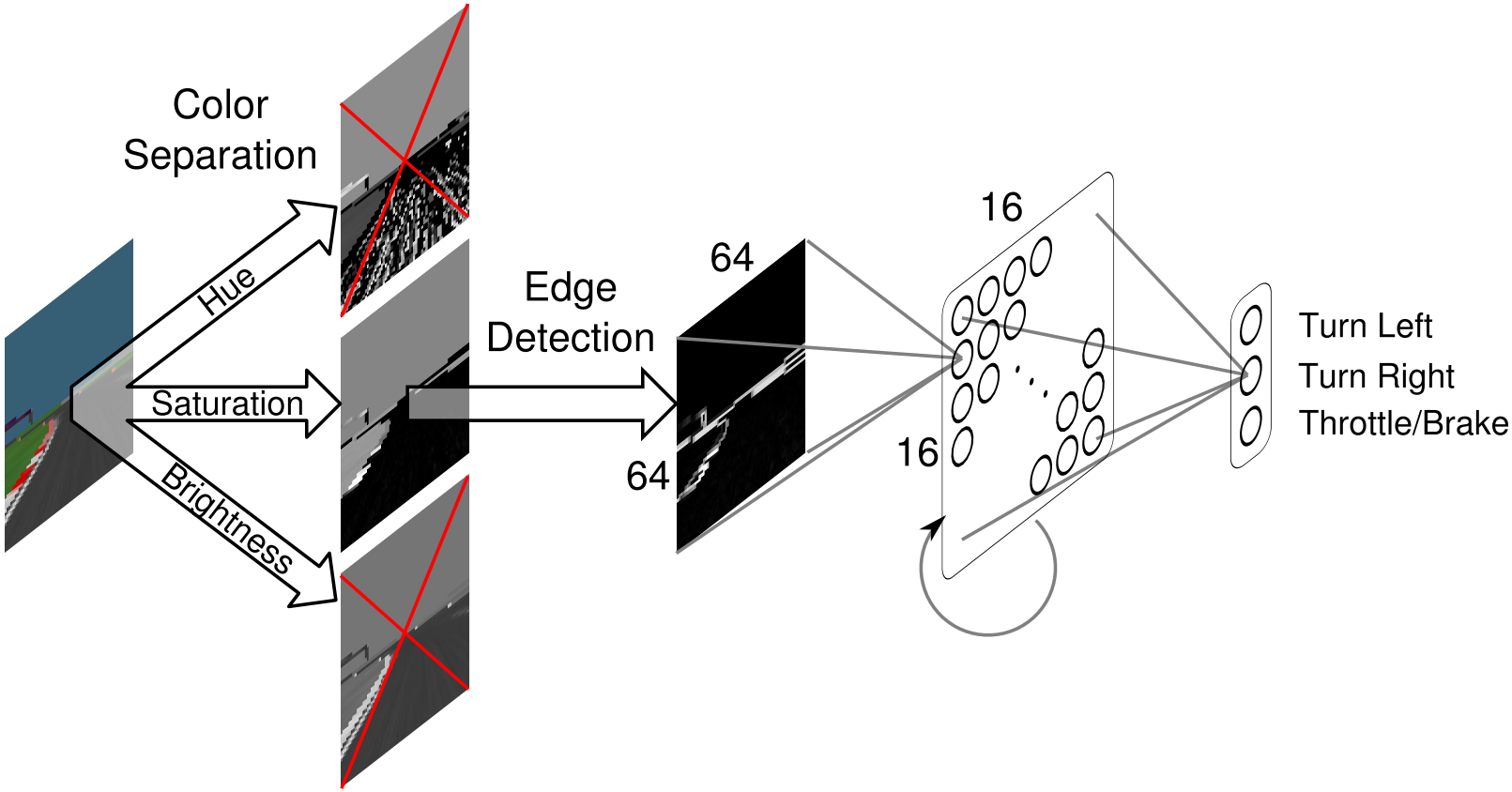

TORCS


TORCS


TORCS



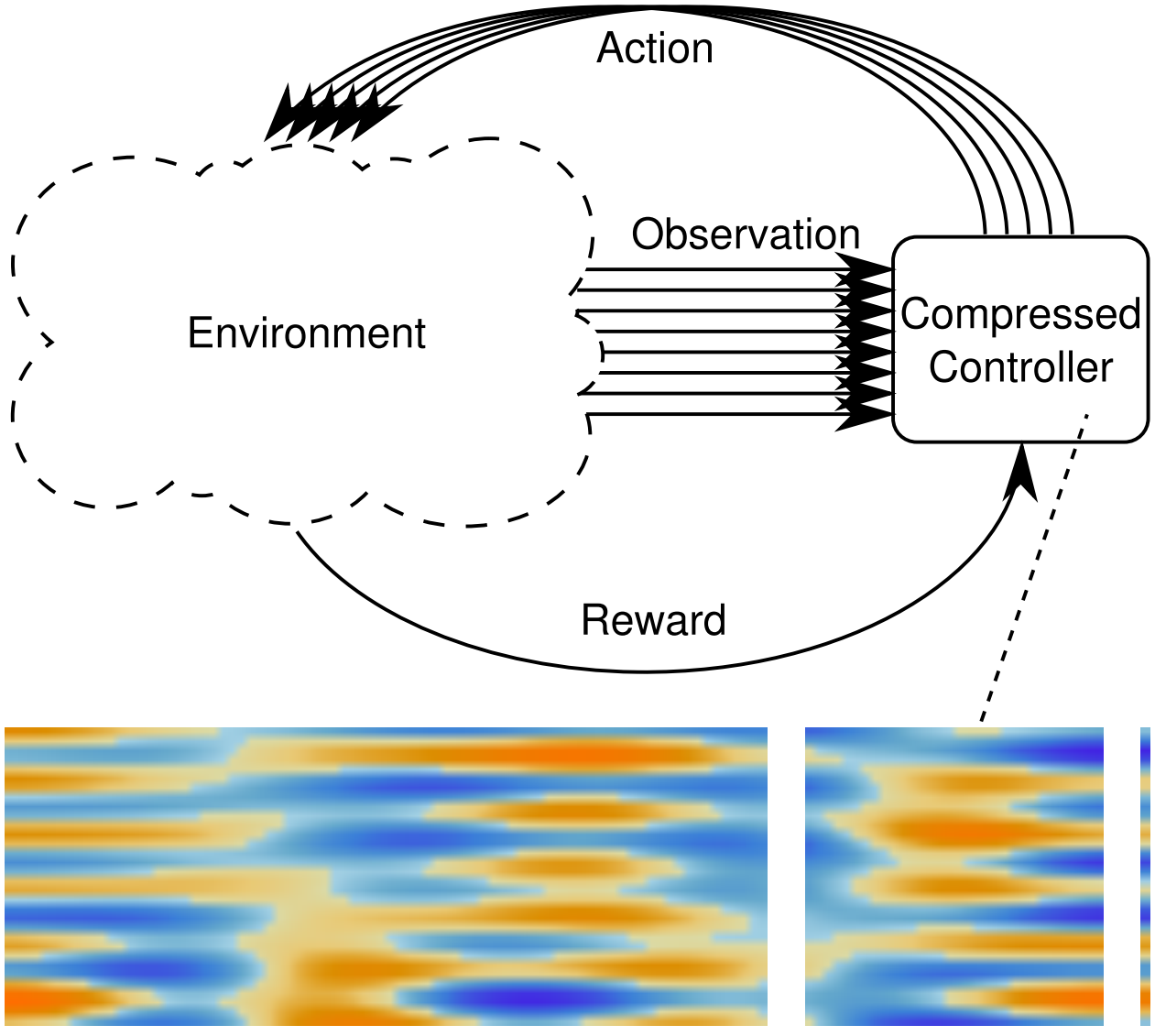


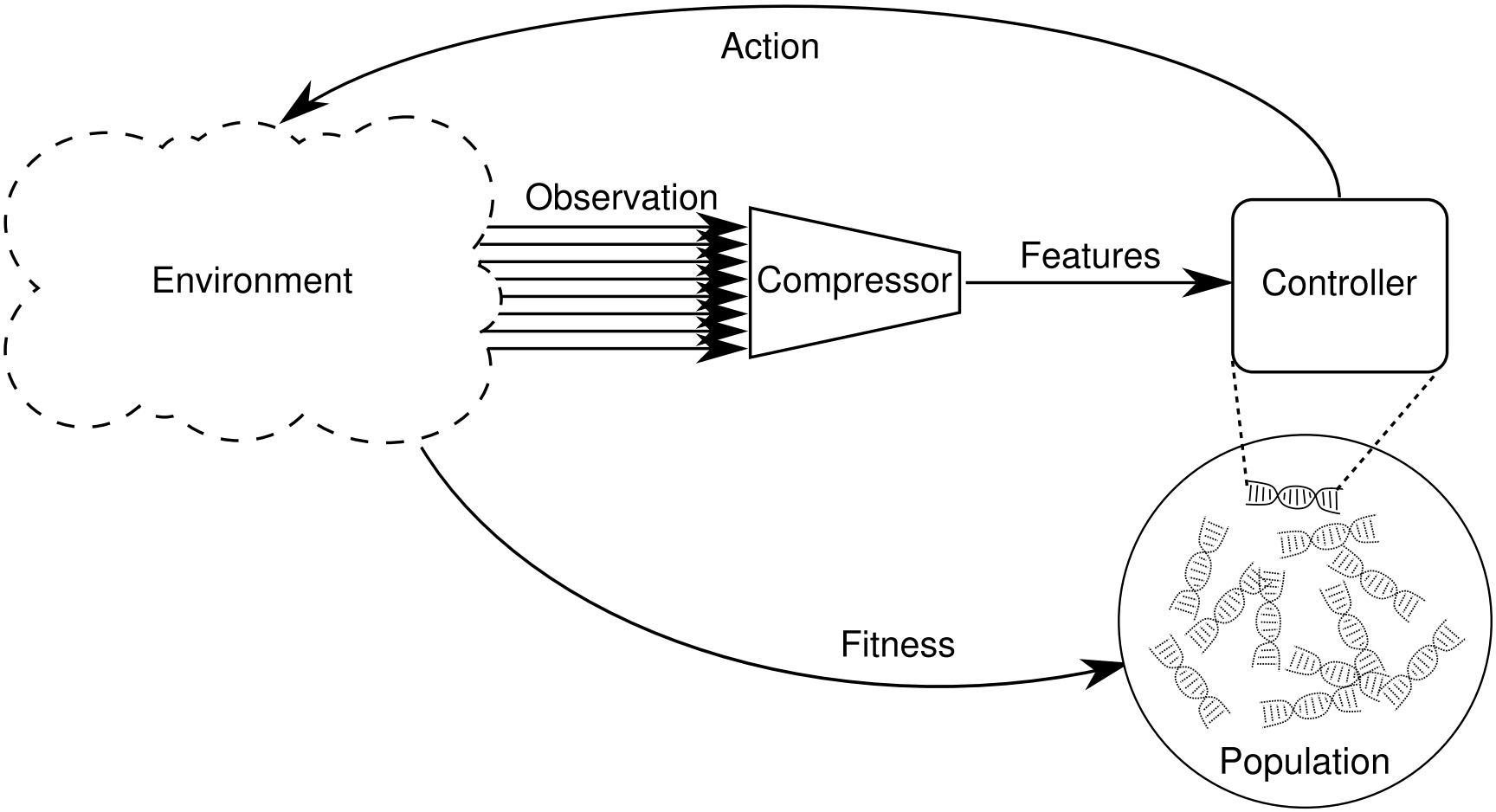


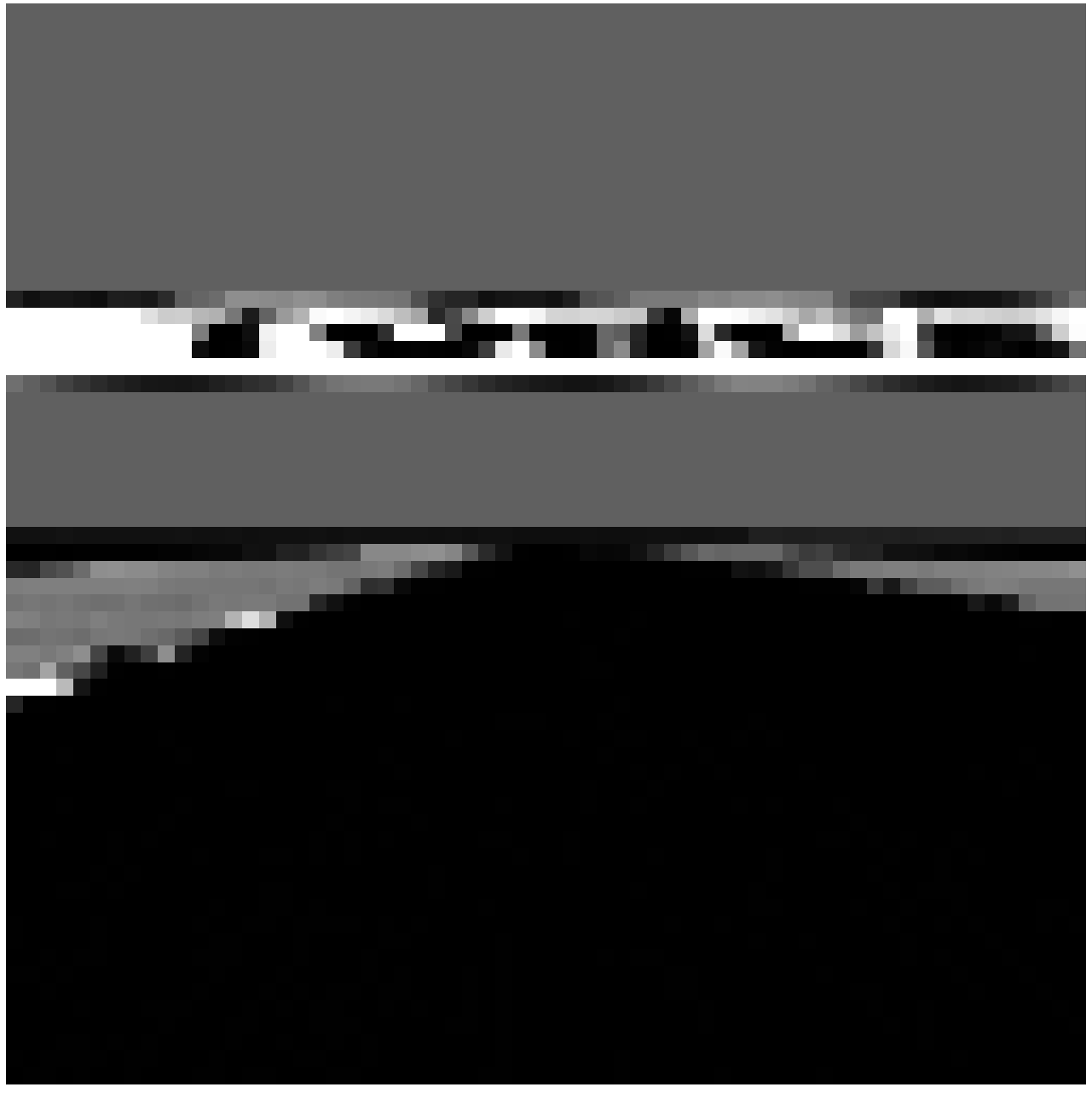
- RNN with 4096 (64×64) pixel inputs and 256 hidden neurons and 3 outputs → 1,115,139 weights
- Compressed to 200 coefficients (1:5575)
- Drives a simulated car along a track using vision only[99,100]

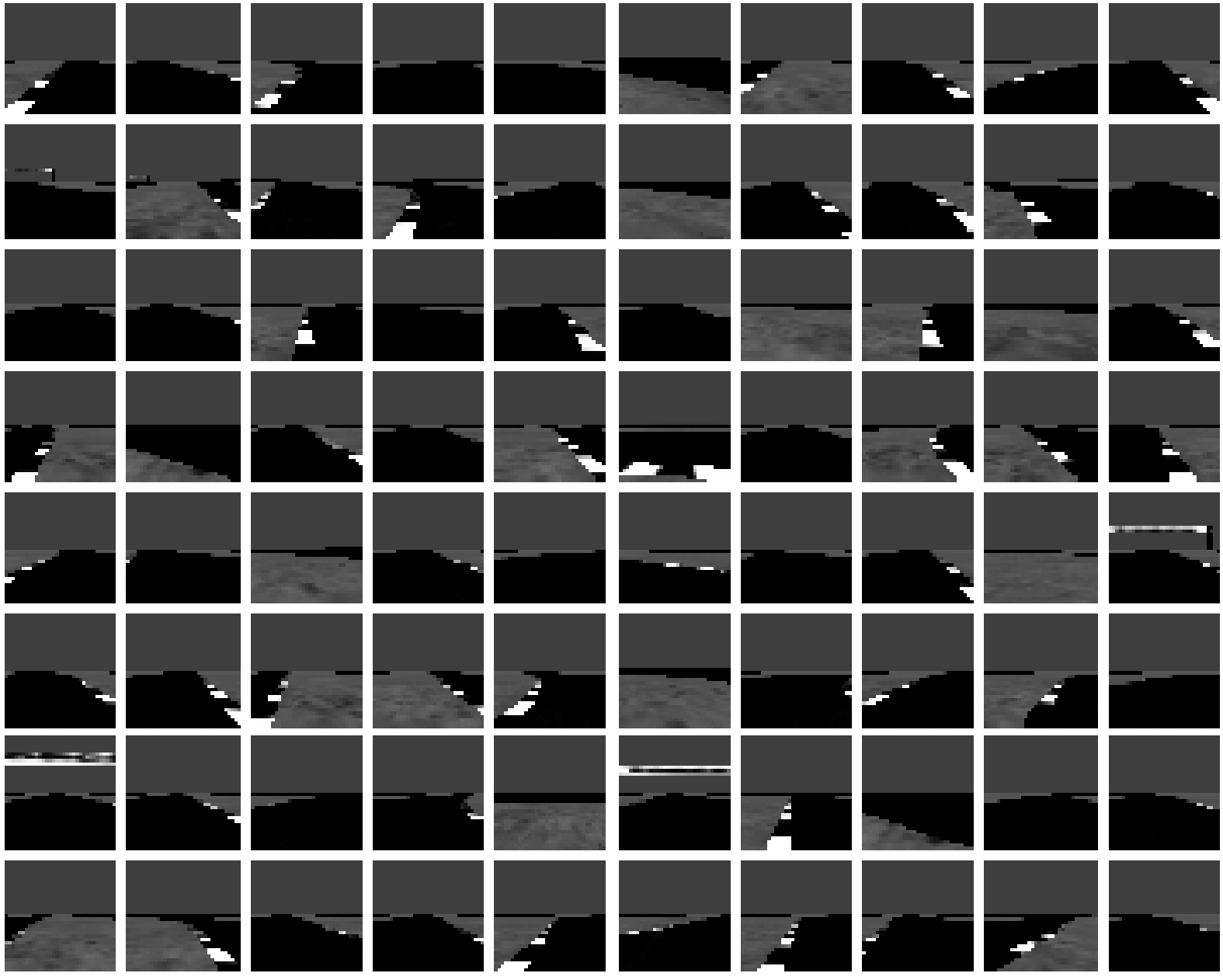
[99] J. Koutník, G. Cuccu, J. Schmidhuber, *et al.*, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Amsterdam: ACM, 2013, pp. 1061–1068, ISBN: ISBN 978-1-4503-1963-8.

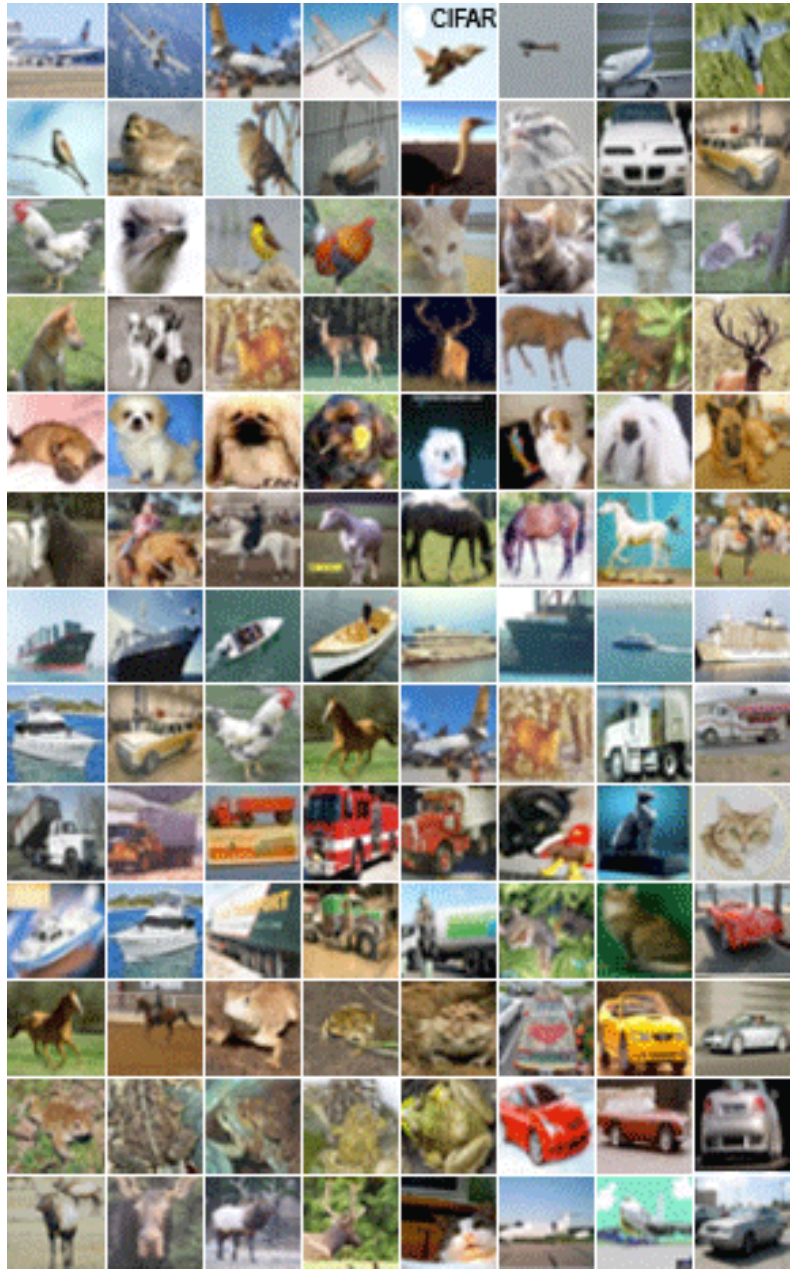
[100] J. Koutník, G. Cuccu, J. Schmidhuber, *et al.*, "Evolving large-scale neural networks for vision-based TORCS," , in *Foundations of Digital Games (FDG)*, Chania, Crete, GR, 2013, pp. 206–212, ISBN: ISBN 978-0-9913982-0-1.

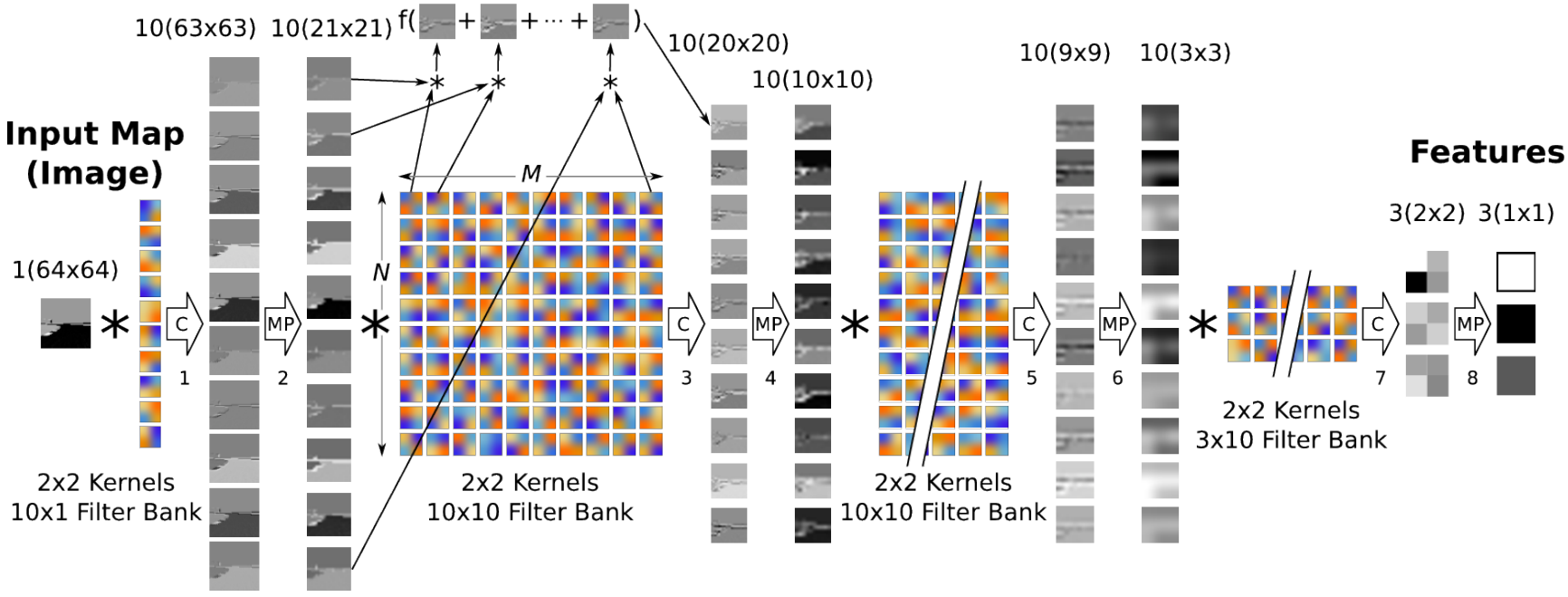

Action
Observation
Environment
Compressed
Controller
Reward

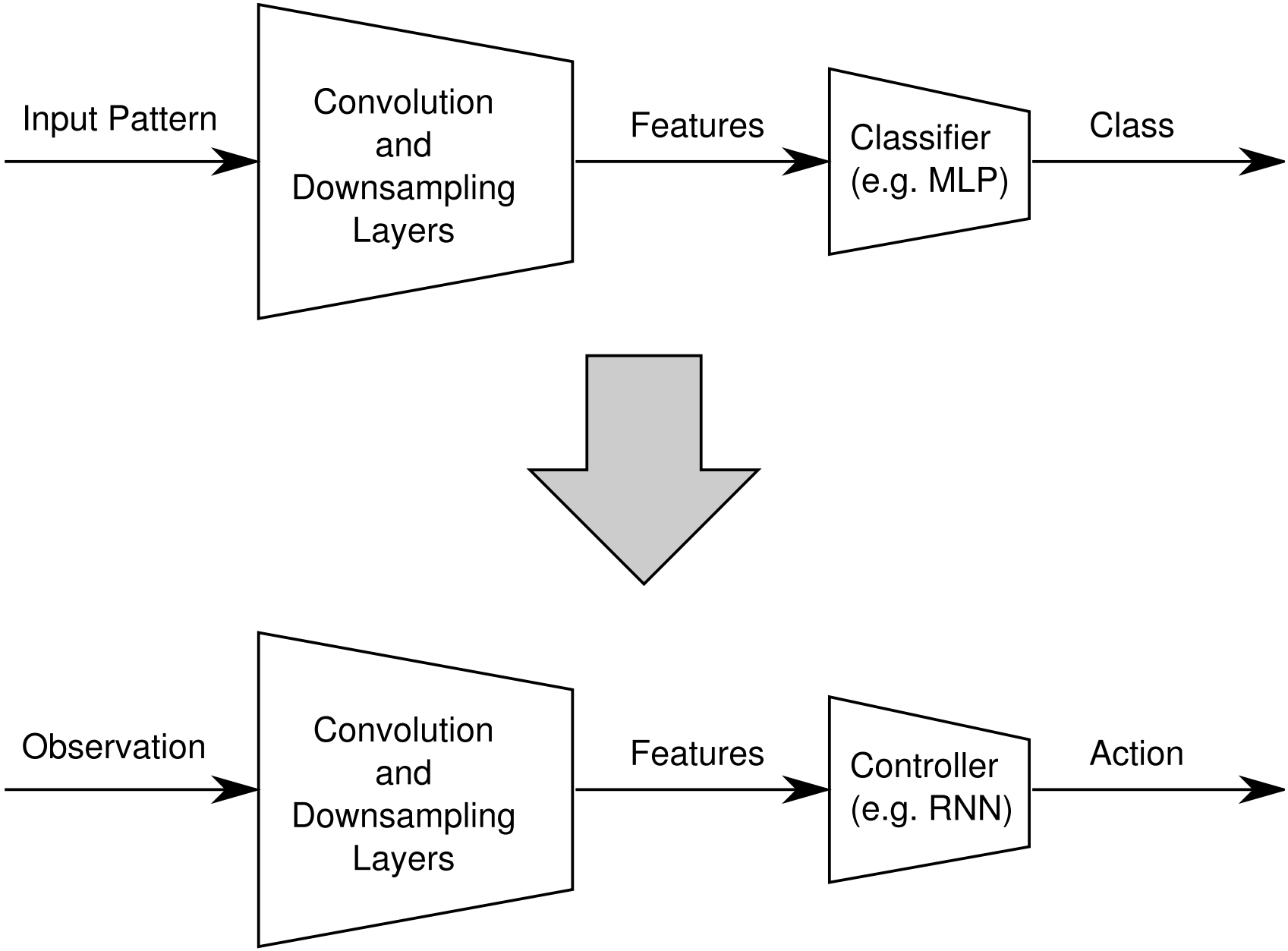

Action
Environment
Observation
Compressor
Features
Controller
Fitness
Population

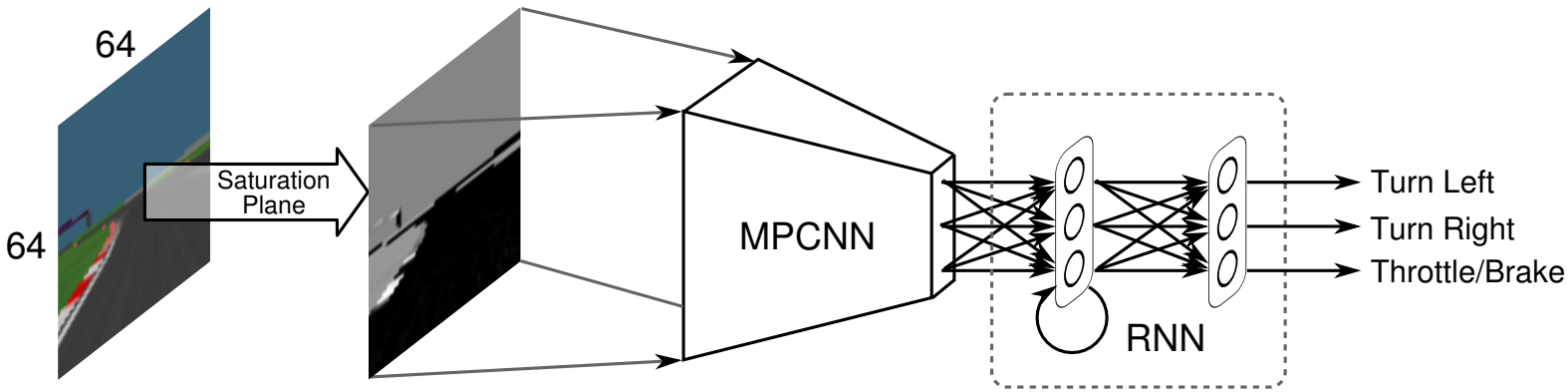


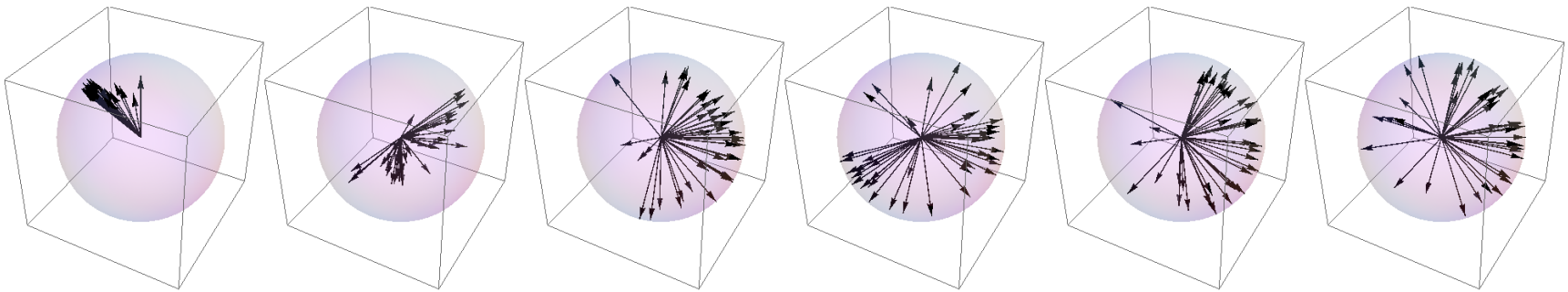


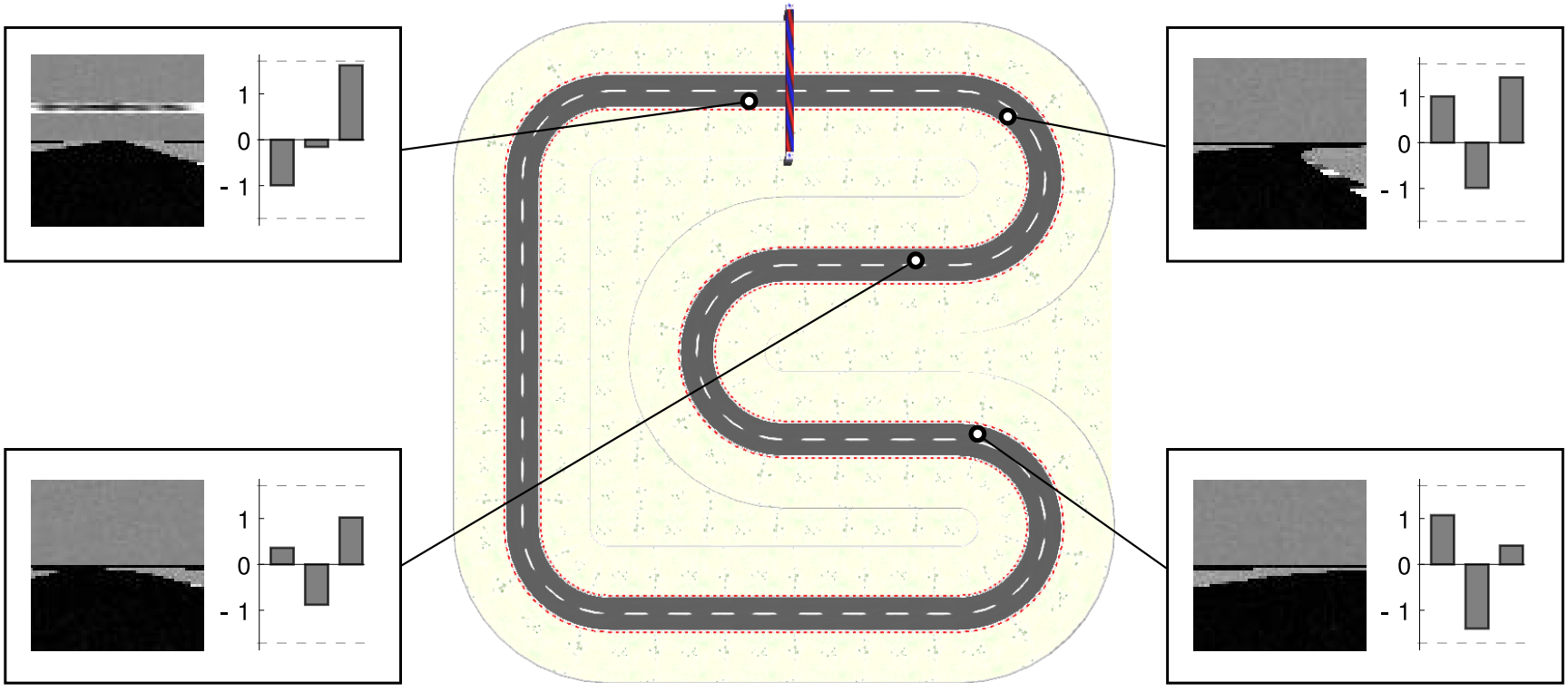

CIFAR

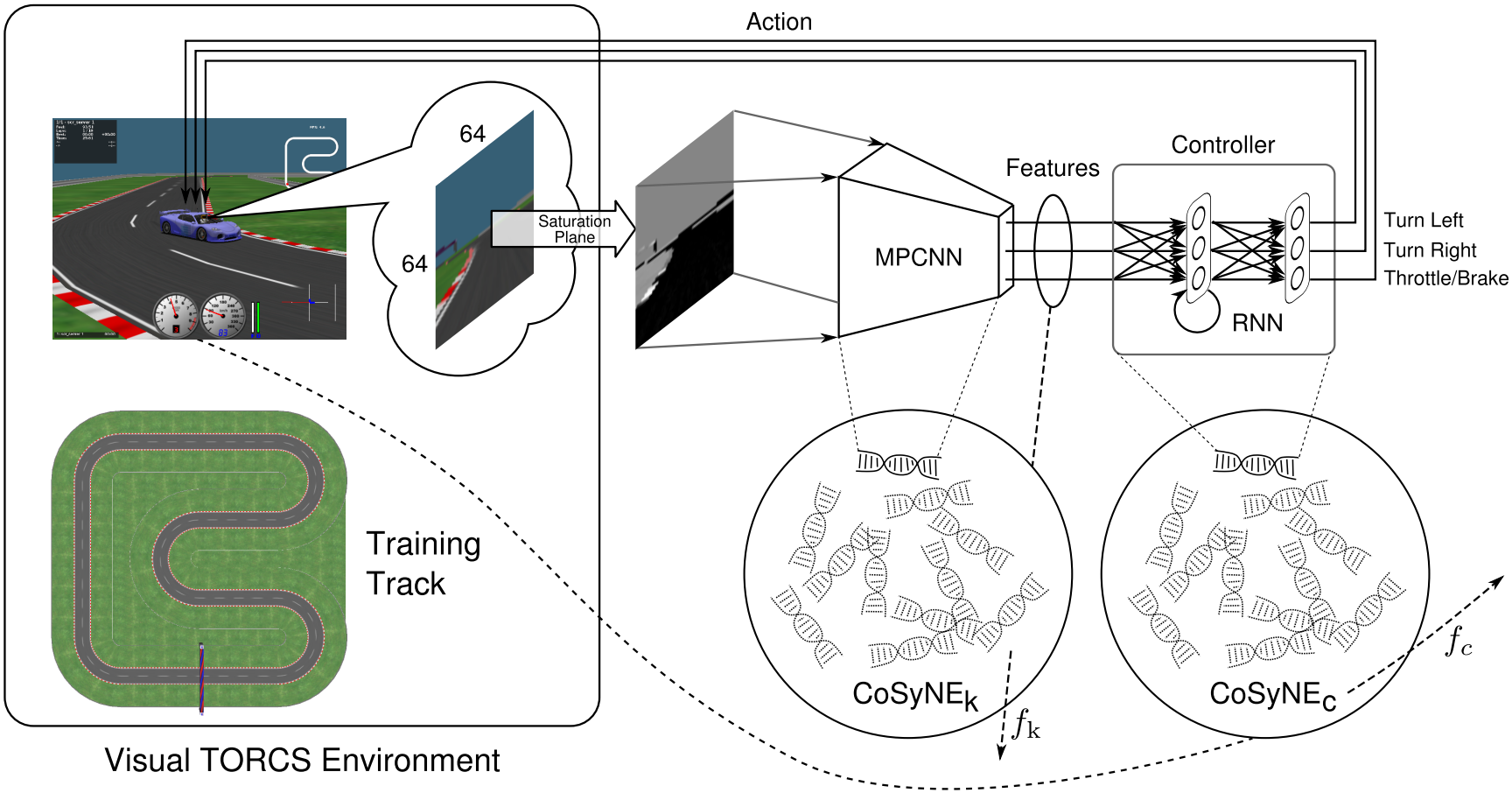

Input Map
(Image)
1(64x64)
2x2 Kernels
10x1 Filter Bank
C
1
10(63x63)
MP
2
10(21x21)
f(+ + ··· +)
M
N
2x2 Kernels
10x10 Filter Bank
C
3
10(20x20)
MP
4
10(10x10)
2x2 Kernels
10x10 Filter Bank
C
5
10(9x9)
MP
6
10(3x3)
2x2 Kernels
3x10 Filter Bank
C
7
3(2x2)
MP
8
3(1x1)
Features


Input Pattern
Convolution and Downsampling Layers
Features
Classifier (e.g. MLP)
Class
Observation
Convolution and Downsampling Layers
Features
Controller (e.g. RNN)
Action


64
64
Saturation
Plane
MPCNN
RNN
Turn Left
Turn Right
Throttle/Brake


0
1
2
4
16
32
Generation


1
0
- 1
1
0
- 1
1
0
- 1
1
0
- 1


Action
64
64
Saturation Plane
Features
Controller
MPCNN
RNN
Turn Left
Turn Right
Throttle/Brake
Training Track
CoSyNE$_k$
CoSyNE$_c$
f_k
f_c
Visual TORCS Environment

IDSIA is Hiring

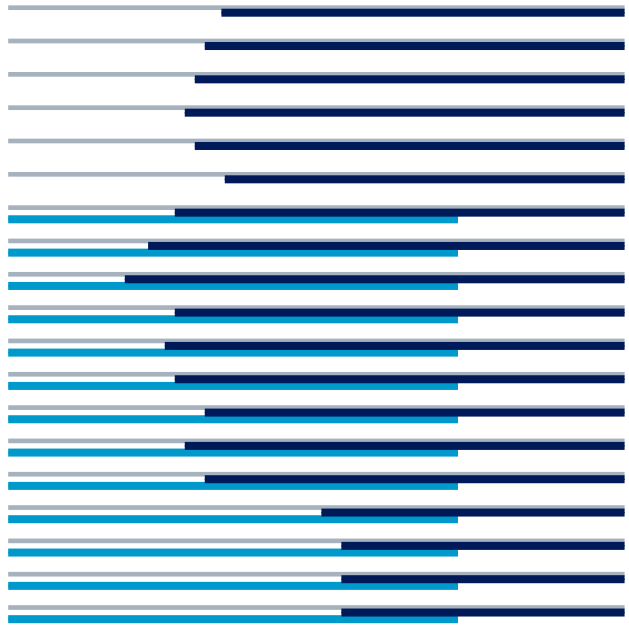
Learn to think[101] (at IDSIA)

- 1 post-doc
- 2 PhD students[102]

For the SNF project **RNNAIssance**

[101] J. Schmidhuber, "On learning to think: algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models," *CoRR*, vol. abs/1511.05249, 2015. [Online]. Available: http://arxiv.org/abs/1511.05249.

[102] http://people.idsia.ch/~juergen/rnnai2016.html



http://www.idsia.ch

Feed Forward Neural Network

Forward Pass Definition

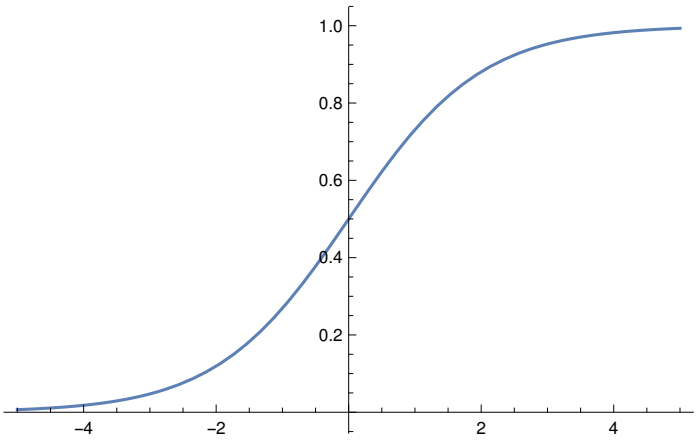
Let' s have the non-linearity to be the logistic sigmoid.

In[1]:=
```
sigmoid[x_] := 1 / (1 + Exp[-x])
```

In[2]:=
```
SetAttributes[sigmoid, Listable]
```

In[3]:=
```
Plot[sigmoid[x], {x, -5, 5}]
```

Out[3]=



Derivative of the sigmoid.

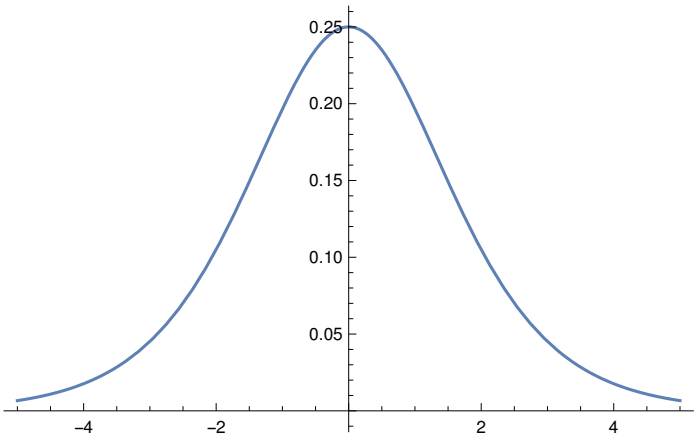
In[4]:=
```
D[sigmoid[x], x]
```

Out[4]=

$$\frac{e^{-x}}{(1+e^{-x})^2}$$

In[5]:=
```
Plot[Evaluate@D[sigmoid[x], x], {x, -5, 5}]
```

Out[5]=



Let' s have a feed - forward NN with 2 inputs and 2 layers with 2 neurons each.

In[6]:=
```
MatrixForm[wInput = {{w11, w12}, {w21, w22}}]
```

Out[6]//MatrixForm=

$$\begin{pmatrix} w11 & w12 \\ w21 & w22 \end{pmatrix}$$

In[7]:=
```
MatrixForm[wHidden = {{w31, w32}, {w41, w42}}]
```

Out[7]//MatrixForm=

$$\begin{pmatrix} w31 & w32 \\ w41 & w42 \end{pmatrix}$$

In[8]:=
```
MatrixForm[input = {x1, x2}]
```

Out[8]//MatrixForm=

$$\begin{pmatrix} x1 \\ x2 \end{pmatrix}$$

In[9]:=
```
net[x_, wI_, wH_] := sigmoid[wH.sigmoid[wI.x]]
```

In[10]:=
```
var = Flatten[{wInput, wHidden}]
```

Out[10]=
```
{w11, w12, w21, w22, w31, w32, w41, w42}
```

Forward evaluation of the network with 2 nodes returns a list of the length two. Here is the symbolic expression for the network evaluation.

In[11]:=
```
n = net[input, wInput, wHidden]
```

Out[11]=

$$\left\{\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}},\ \frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}\right\}$$

Loss (Error) Function

Squared error

In[12]:=
```
squaredError[y_, z_] := 1 / 2 Total[(z - y) ^ 2]
```

In[13]:=
```
squaredError[{y1, y2}, {z1, z2}]
```

Out[13]=

$$\frac{1}{2}\left((-y1+z1)^2+(-y2+z2)^2\right)$$

In[14]:=
```
D[squaredError[{y1, y2}, {z1, z2}], y1]
```

Out[14]=
```
y1 - z1
```

Error Gradient

In[15]:=
```
MatrixForm[target = {z1, z2}]
```

Out[15]//MatrixForm=

$$\begin{pmatrix} z1 \\ z2 \end{pmatrix}$$

The symbolic formula for obtaining the error from the forward pas. It is just the network forward pass combined with the squared error.

In[16]:=
```
squaredError[net[input, wInput, wHidden], target]
```

Out[16]=
$$\frac{1}{2}\left(\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)^2+\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)^2\right)$$

In[17]:=
```
gradient =
 Grad[squaredError[net[input, wInput, wHidden], target], Flatten[{wInput, wHidden}]]
```

Out[17]=
$$\left\{\frac{1}{2}\left(-\left(\left(2\,e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w31\,x1\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right)-\left(2\,e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w41\,x1\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right),$$

$$\frac{1}{2}\left(-\left(\left(2\,e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w31\,x2\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right)-\left(2\,e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w41\,x2\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right),$$

$$\frac{1}{2}\left(-\left(\left(2\,e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}-w21\,x1-w22\,x2}\,w32\,x1\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w21\,x1-w22\,x2}\right)^2\right)\right)-\left(2\,e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}-w21\,x1-w22\,x2}\,w42\,x1\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w21\,x1-w22\,x2}\right)^2\right)\right),$$

$$\frac{1}{2}\left(-\left(\left(2\,e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}-w21\,x1-w22\,x2}\,w32\,x2\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w21\,x1-w22\,x2}\right)^2\right)\right)-\left(2\,e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}-w21\,x1-w22\,x2}\,w42\,x2\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w21\,x1-w22\,x2}\right)^2\right)\right),$$

$$-\left(\left(e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)\right)\right),$$

$$-\left(\left(e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w21\,x1-w22\,x2}\right)\right)\right),$$

$$-\left(\left(e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)\right)\right),$$

$$-\left(\left(e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w21\,x1-w22\,x2}\right)\right)\right)\Bigg\}$$

The following is the first element of the gradient, which has 8 terms (the network has 8 weights).

In[18]:=
```
gradient[[1]]
```

Out[18]=
$$\frac{1}{2}\left(-\left(\left(2\,e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w31\,x1\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right)-\left(2\,e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w41\,x1\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right)$$

In[19]:=
```
D[squaredError[net[input, wInput, wHidden], target], w11]
```

Out[19]=
$$\frac{1}{2}\left(-\left(\left(2\,e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w31\,x1\left(-\frac{1}{1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}}+z1\right)\right)\Big/\left(\left(1+e^{-\frac{w31}{1+e^{-w11\,x1-w12\,x2}}-\frac{w32}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right)-\left(2\,e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}-w11\,x1-w12\,x2}\,w41\,x1\left(-\frac{1}{1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}}+z2\right)\right)\Big/\left(\left(1+e^{-\frac{w41}{1+e^{-w11\,x1-w12\,x2}}-\frac{w42}{1+e^{-w21\,x1-w22\,x2}}}\right)^2\left(1+e^{-w11\,x1-w12\,x2}\right)^2\right)\right)$$

Numerical Gradient

Let' s use the following numeric values for inputs (i), input weights (wi), hidden weights (wh) and target outputs (z).

In[20]:=
```
MatrixForm[i = {{-1, 1}, {1, -1}}]
```

Out[20]//MatrixForm=

$$\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

In[21]:=
```
MatrixForm[wi = RandomReal[{-.1, .1}, {2, 2}]]
```

Out[21]//MatrixForm=

$$\begin{pmatrix} 0.00960616 & -0.0283805 \\ 0.0278706 & -0.0362715 \end{pmatrix}$$

In[22]:=
```
MatrixForm[wh = RandomReal[{-.1, .1}, {2, 2}]]
```

Out[22]//MatrixForm=

$$\begin{pmatrix} -0.0702671 & 0.0345313 \\ -0.0775556 & -0.0354687 \end{pmatrix}$$

In[23]:=
```
MatrixForm[z = {{1, 0}, {0, 1}}]
```

Out[23]//MatrixForm=

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The makeRules function is a helper that generates rules that replace the symbolic values with numerical ones. One cannot do it together with the gradient derivation, because the symbolic function Grad needs symbols instead of numerical values and Mathematica would try to plug the real numbers earlier.

In[24]:=
```
makeRules[i_, wi_, wh_, z_] :=
 Flatten@MapThread[MapThread[Rule, {Flatten[#1], Flatten[#2]}] &,
   {{input, wInput, wHidden, target}, {i, wi, wh, z}}]
```

In[25]:=
```
makeRules[i[[1]], Flatten@wi, wh, z[[1]]]
```

Out[25]=
```
{x1 → -1, x2 → 1, w11 → 0.00960616, w12 → -0.0283805, w21 → 0.0278706, w22 → -0.0362715,
 w31 → -0.0702671, w32 → 0.0345313, w41 → -0.0775556, w42 → -0.0354687, z1 → 1, z2 → 0}
```

Calculate forward pass (just to know, (not) needed for the gradient calculation).

In[26]:=
```
net[#, wi, wh] & /@ i
```

Out[26]=
```
{{0.495562, 0.486202}, {0.495505, 0.48555}}
```

Numerical gradient for given weights, inputs and targets for both input/output pairs, calculated separately.

In[27]:=
```
gradient /. makeRules[i[[1]], wi, wh, z[[1]]]
```

Out[27]=
```
{0.000139719, -0.000139719, 0.00216336,
 -0.00216336, -0.0618525, -0.0610285, 0.0595756, 0.058782}
```

In[28]:=
```
gradient /. makeRules[i[[2]], wi, wh, z[[2]]]
```

Out[28]=
```
{0.00031553, -0.00031553, 0.00220652,
 -0.00220652, 0.0631093, 0.0639187, -0.0654728, -0.0663125}
```

The complete gradient is obtained by summing these two gradient parts.

```
In[29]:= g = Plus @@ MapThread[gradient /. makeRules[#1, wi, wh, #2] &, {i, z}]
Out[29]= {0.000455248, -0.000455248, 0.00436988,
 -0.00436988, 0.0012568, 0.00289021, -0.00589717, -0.00753055}
```

Let the learning rate α be 0.2.

```
In[30]:= α = 0.2
Out[30]= 0.2
```

The list of network parameters.

```
In[31]:= w = Flatten[{wi, wh}]
Out[31]= {0.00960616, -0.0283805, 0.0278706, -0.0362715,
 -0.0702671, 0.0345313, -0.0775556, -0.0354687}
```

The weights are updated subtracting the gradient (times the learning rate) from the current weights.

```
In[32]:= w = w - α g
Out[32]= {0.00951511, -0.0282894, 0.0269966, -0.0353975,
 -0.0705185, 0.0339532, -0.0763761, -0.0339626}
```

```
In[33]:= MatrixForm /@ ({wi, wh} = ArrayReshape[w, {2, 2, 2}])
Out[33]= {( 0.00951511  -0.0282894 )
          ( 0.0269966   -0.0353975 ), ( -0.0705185  0.0339532 )
                                      ( -0.0763761 -0.0339626 )}
```

Training

Forward pass with updated weights.

```
In[34]:= net[#, wi, wh] & /@ i
Out[34]= {{0.495464, 0.486524}, {0.495395, 0.485899}}
```

Run multiple epochs. The gradient update is put into a function.

```
In[35]:= update[wi_, wh_, α_] := ArrayReshape[Flatten[{wi, wh}] -
    α (Plus @@ MapThread[gradient /. makeRules[#1, wi, wh, #2] &, {i, z}]), {2, 2, 2}]
```

Run the function for 500 times.

```
In[36]:= MatrixForm /@ ({wi, wh} = Nest[update[#[[1]], #[[2]], α] &, {wi, wh}, 500])
Out[36]= {( 1.62456  -1.64333 )
          ( -1.44977  1.44137 ), ( -2.26984  2.18185 )
                                 ( 2.25568  -2.16908 )}
```

We can see that the trained network outputs numbers close to the desired ones.

```
In[37]:= MatrixForm[z]
Out[37]//MatrixForm=
(1  0)
(0  1)
```

In[38]:=
```
MatrixForm[Round[net[#, wi, wh] & /@ i, 0.001]]
```

Out[38]//MatrixForm=

$$\begin{pmatrix} 0.879 & 0.122 \\ 0.112 & 0.887 \end{pmatrix}$$