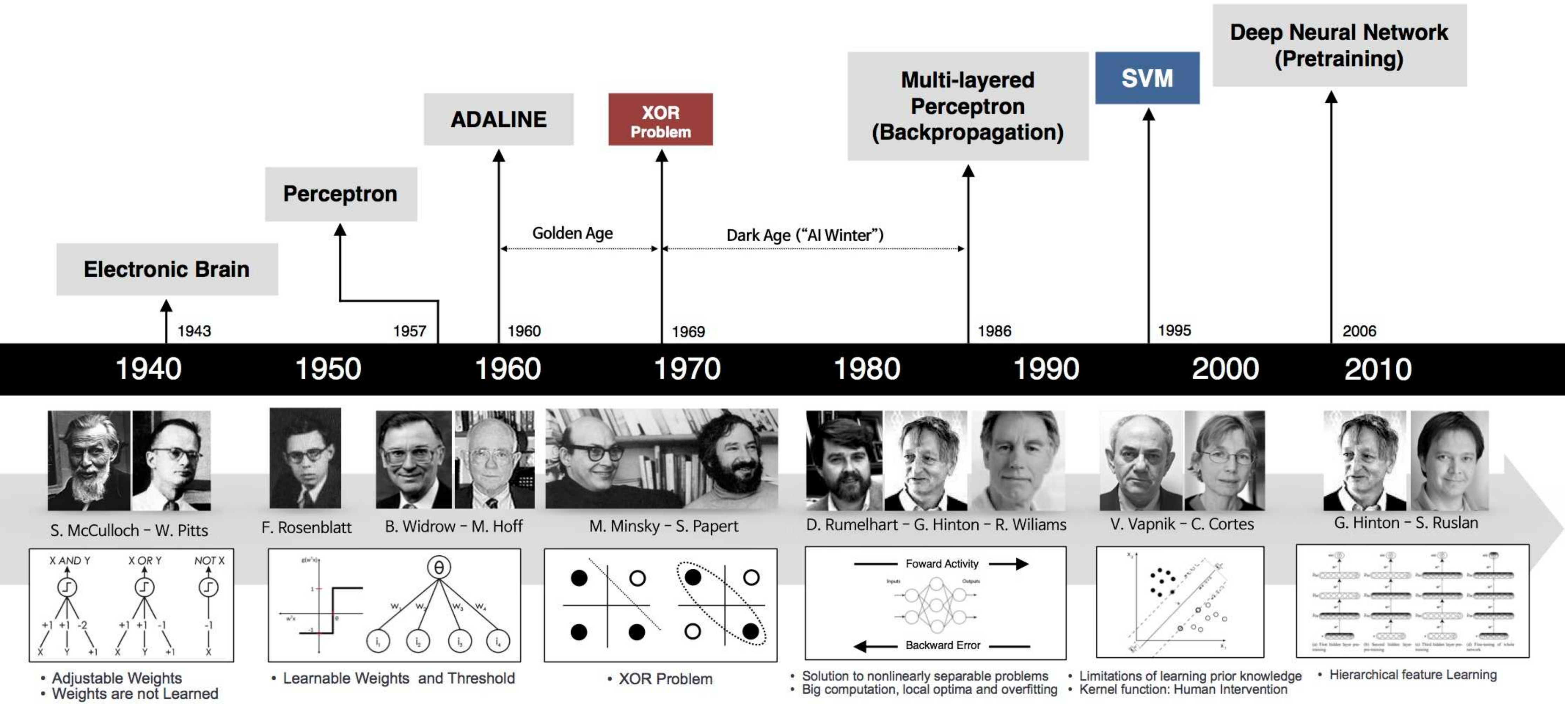
Deep Learning
An Introduction

Milan Straka

January 10, 2019

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Introduction to Machine Learning History



https://www.slideshare.net/deview/251-implementing-deep-learning-using-cu-dnn/4

Curse of Dimensionality

Figure 5.9, page 156 of Deep Learning Book, http://deeplearningbook.org.

Machine and Representation Learning

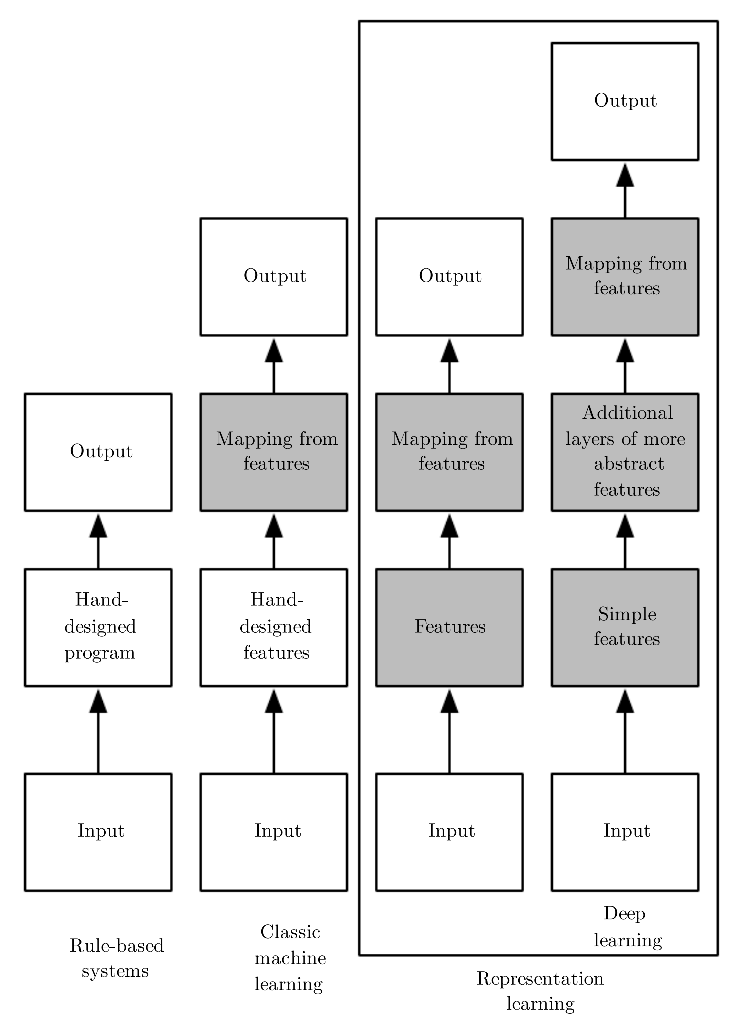
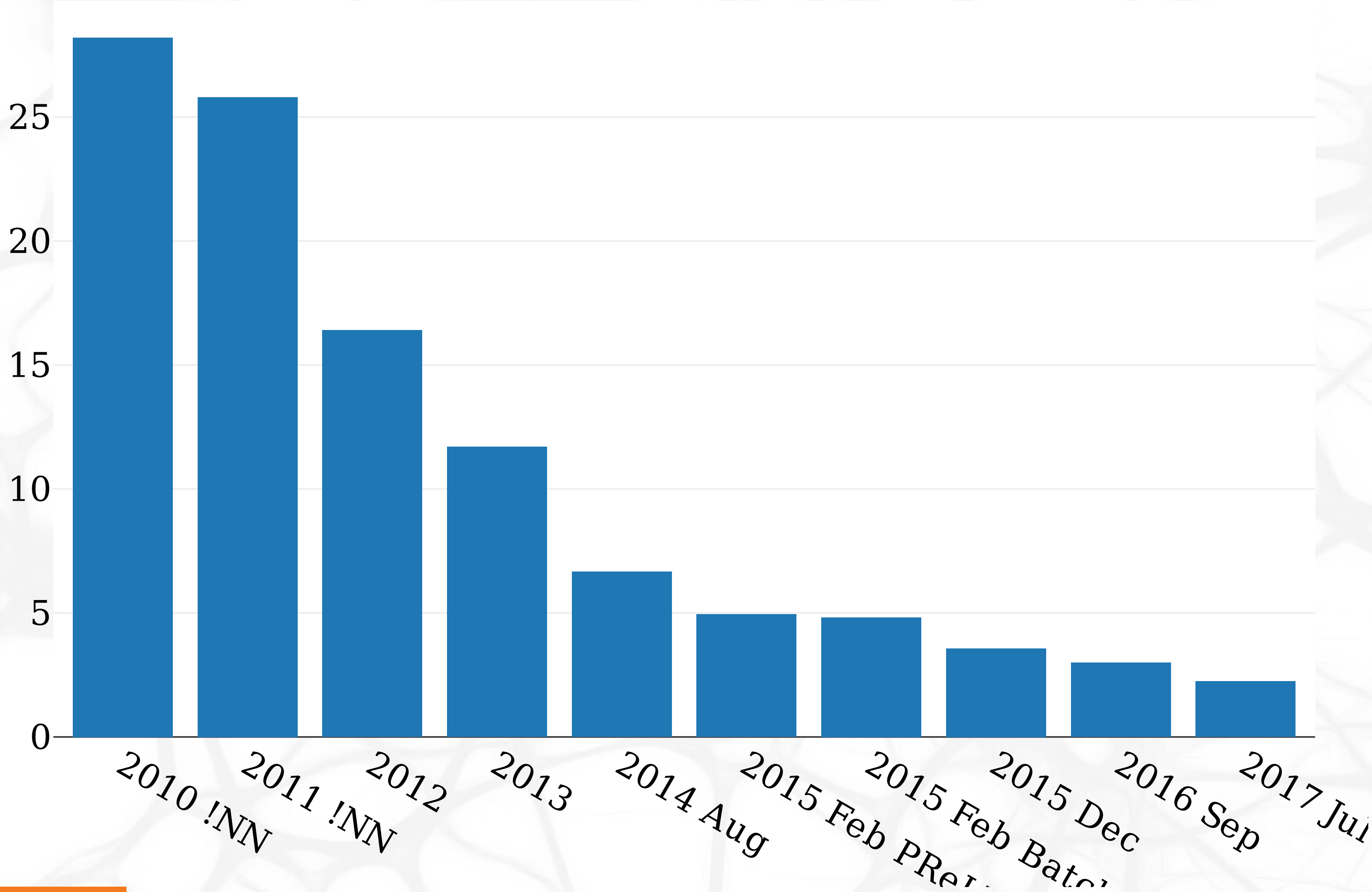
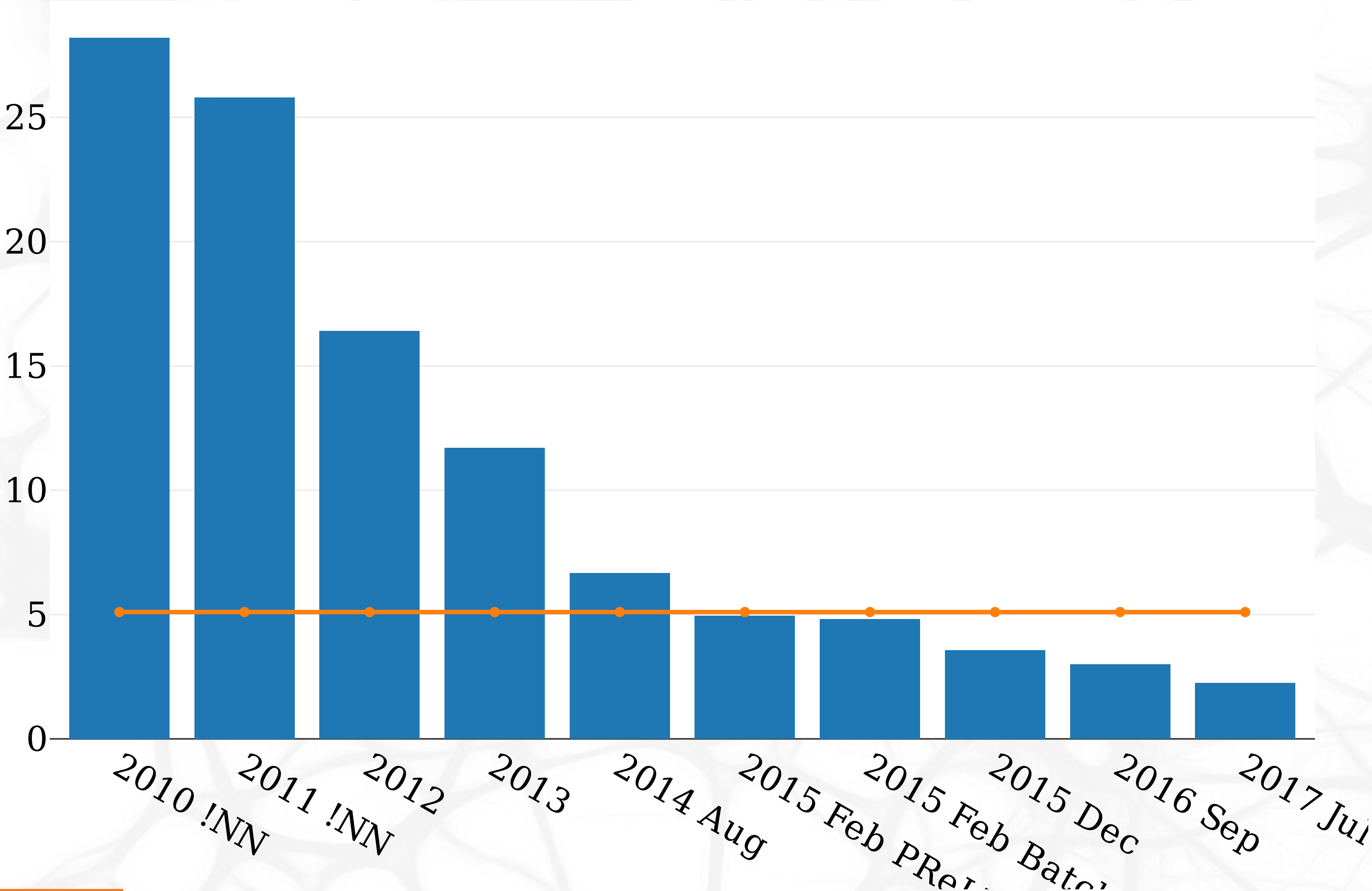


Figure 1.5, page 10 of Deep Learning Book, http://deeplearningbook.org.

ILSVRC Image Recognition Error Rates



ILSVRC Image Recognition Error Rates



ILSVRC Image Recognition Error Rates

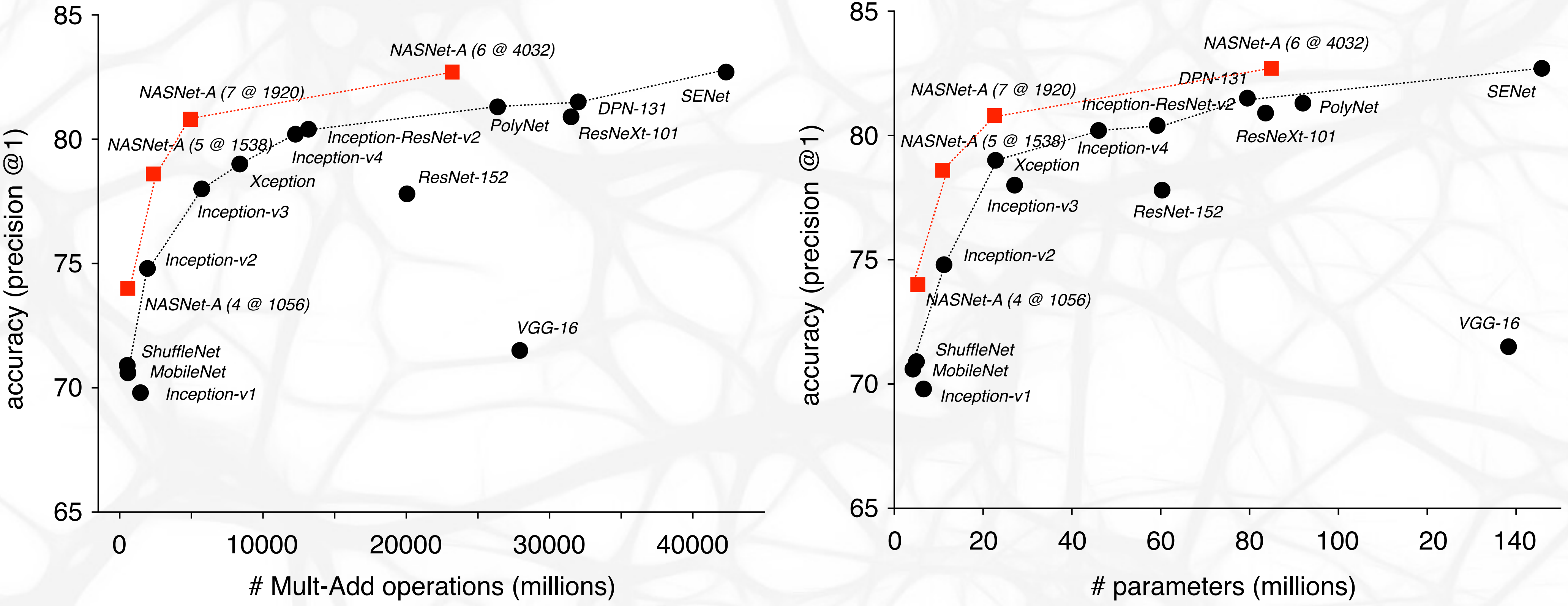
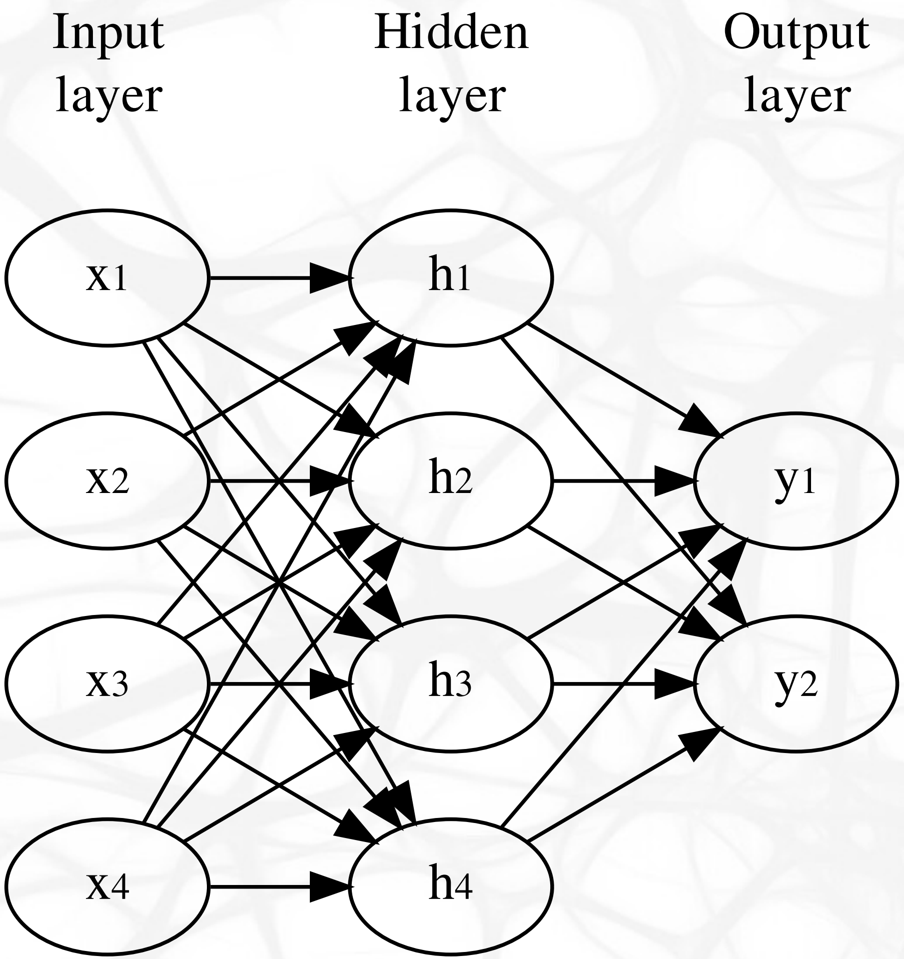


Figure 5 of paper "Learning Transferable Architectures for Scalable Image Recognition", https://arxiv.org/abs/1707.07012.

Neural Network Architecture of the '80s



Neural Network Architecture

There is a weight on each edge, and an activation function f is performed on the hidden layers, and optionally also on the output layer.

$$h_i = f\left(\sum_j w_{i,j} x_j\right)$$

If the network is composed of layers, we can use matrix notation and write:

$$\boldsymbol{h} = f\left(\boldsymbol{W}\boldsymbol{x}\right)$$

Output Layers

- none (linear regression if there are no hidden layers)
- σ (sigmoid; logistic regression if there are no hidden layers)

$$\sigma(x) \stackrel{\text{def}}{=} \frac{1}{1+e^{-x}}$$

- $\operatorname{softmax}$ (maximum entropy model if there are no hidden layers)

$$\operatorname{softmax}(\boldsymbol{x}) \propto e^{\boldsymbol{x}}$$

$$\operatorname{softmax}(\boldsymbol{x})_i \stackrel{\text{def}}{=} \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Hidden Layers

- none (does not help, composition of linear mapping is a linear mapping)
- σ (but works badly – nonsymmetrical, $\frac{d\sigma}{dx}(0) = 1/4$)
- $\tanh$
  - result of making σ symmetrical and making derivation in zero 1
  - $\tanh(x) = 2\sigma(2x) - 1$
- ReLU
  - $\max(0, x)$

Universal Approximation Theorem '89

Let $\varphi(x)$ be a nonconstant, bounded and monotonically-increasing continuous function.

Then for any $\varepsilon > 0$ and any continuous function f on $[0, 1]^m$ there exists an $N \in \mathbb{N}, v_i \in \mathbb{R}, \boldsymbol{b_i} \in \mathbb{R}$ and $\boldsymbol{w}_i \in \mathbb{R}^m$, such that if we denote

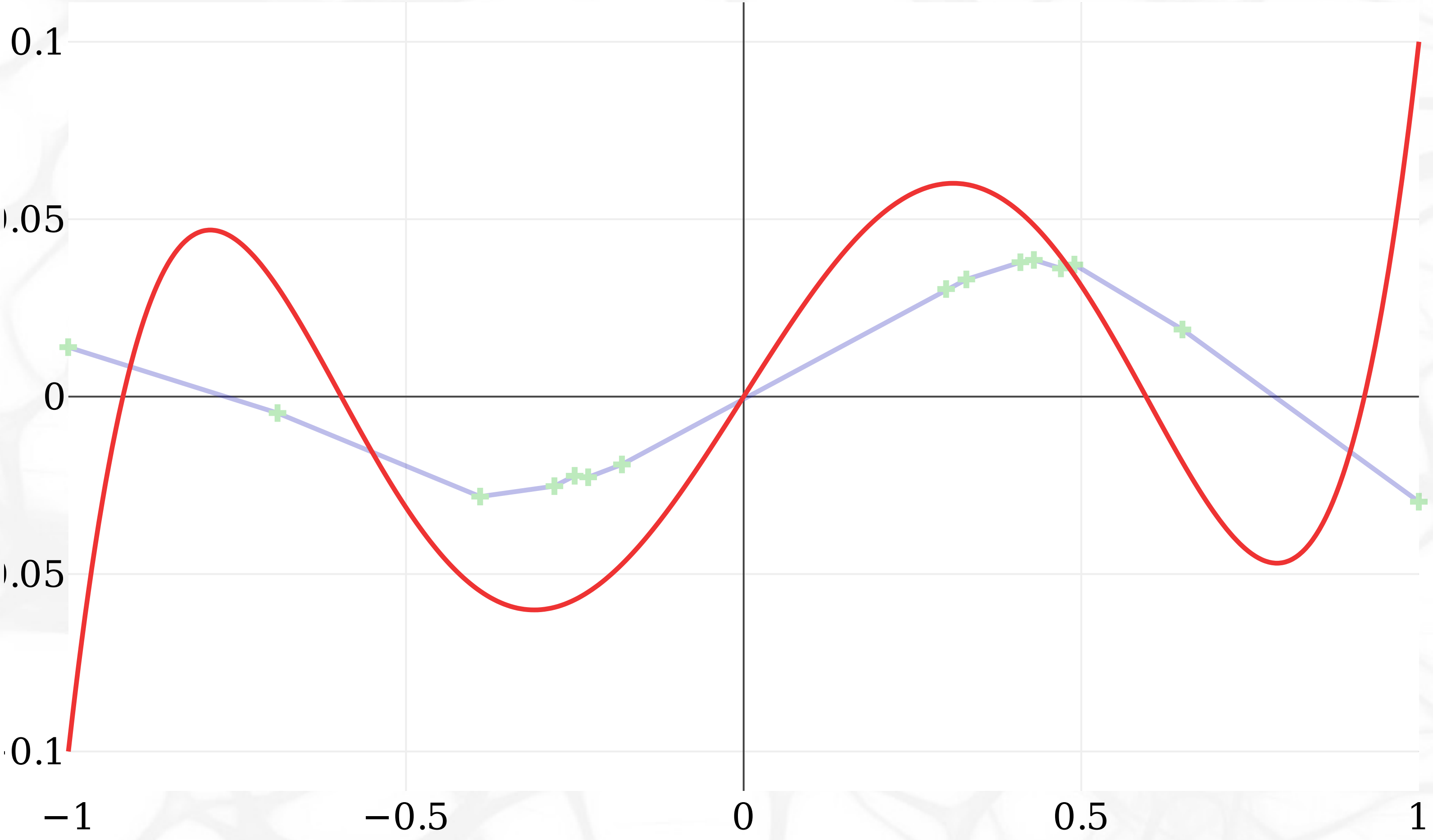
$$F(\boldsymbol{x}) = \sum_{i=1}^{N} v_i \varphi(\boldsymbol{w_i}^T \boldsymbol{x} + b_i)$$

then for all $x \in [0, 1]^m$

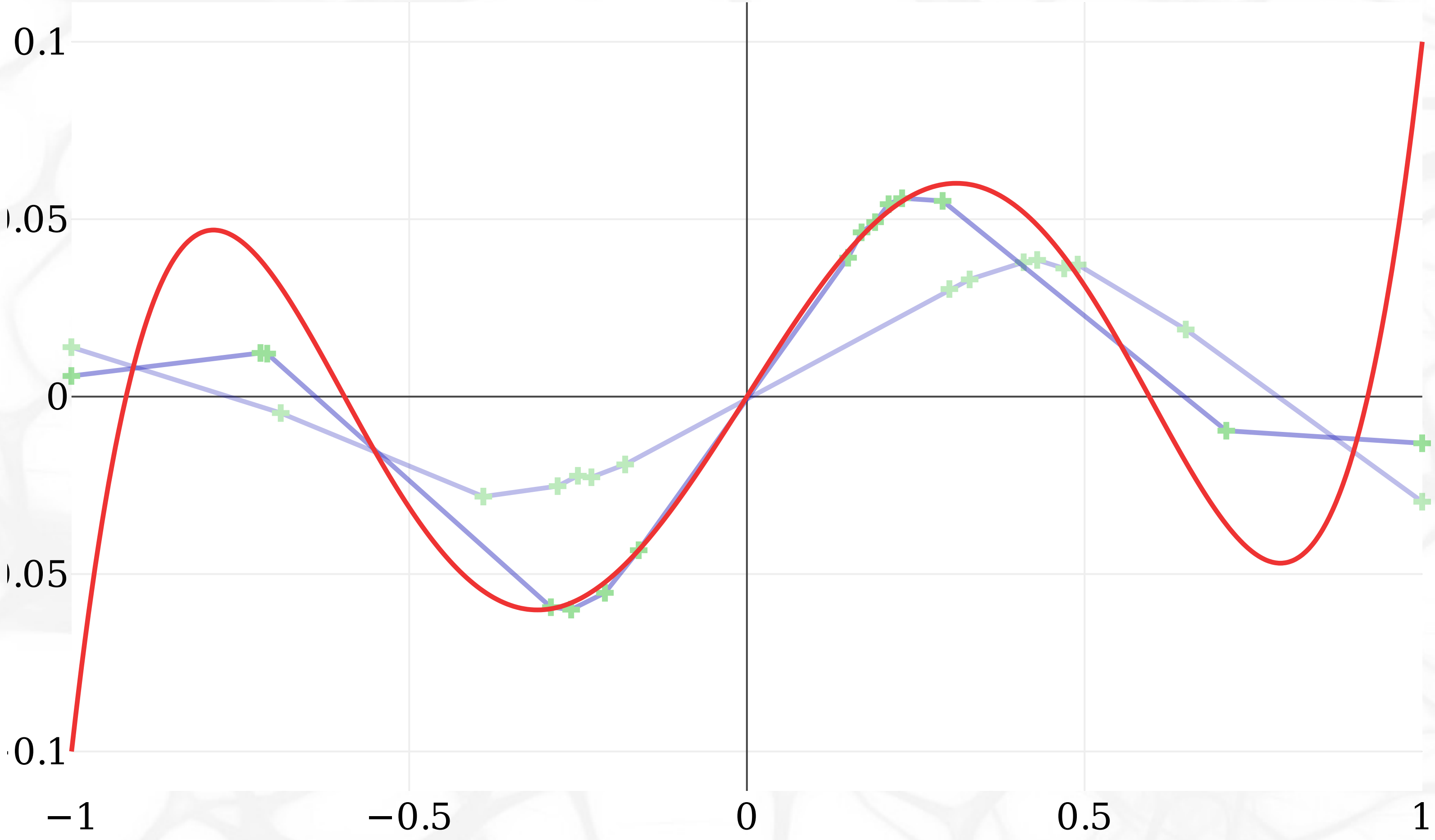
$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \varepsilon.$$

Evolving ReLU Approximation

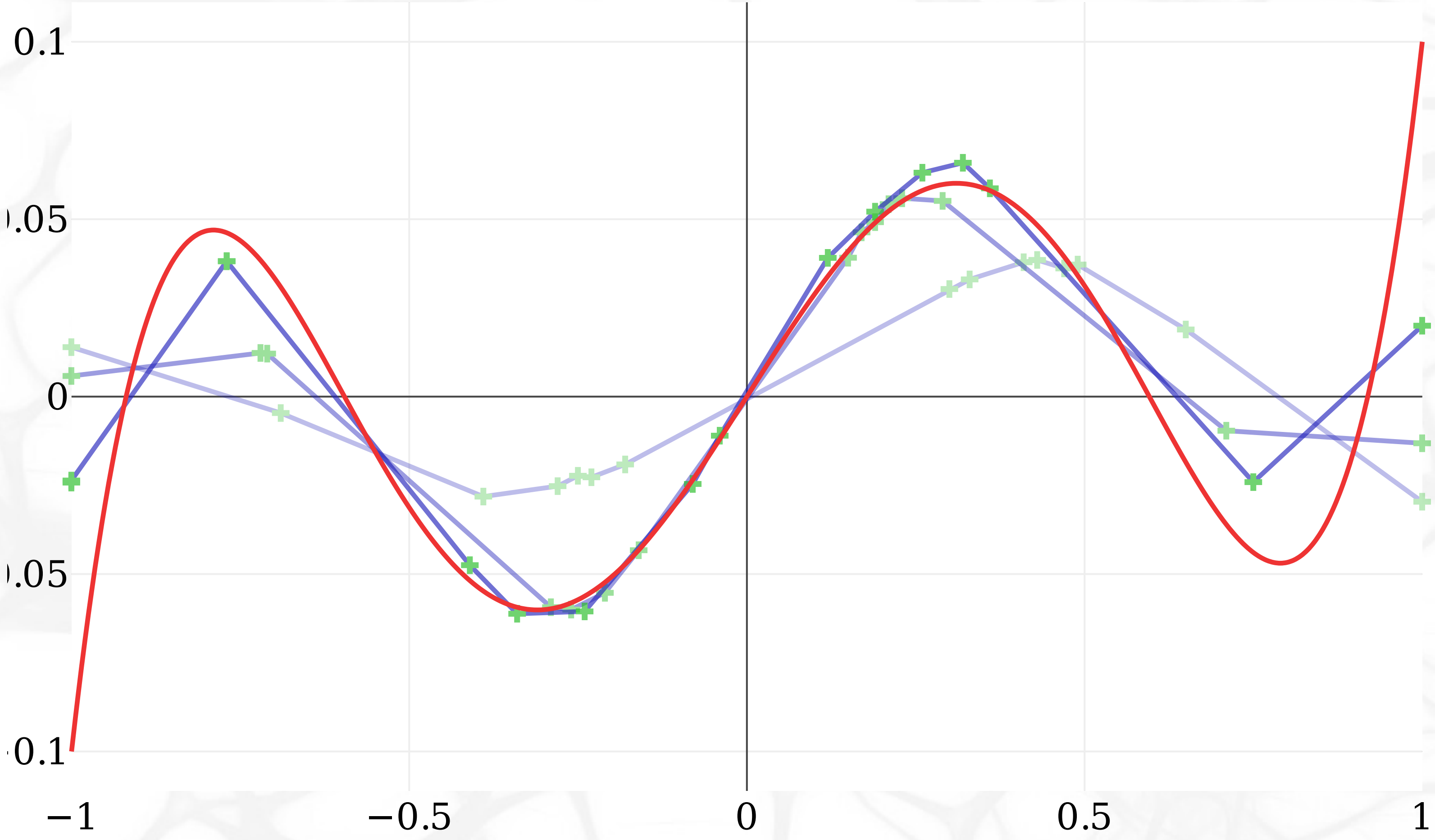
Evolving ReLU Approximation



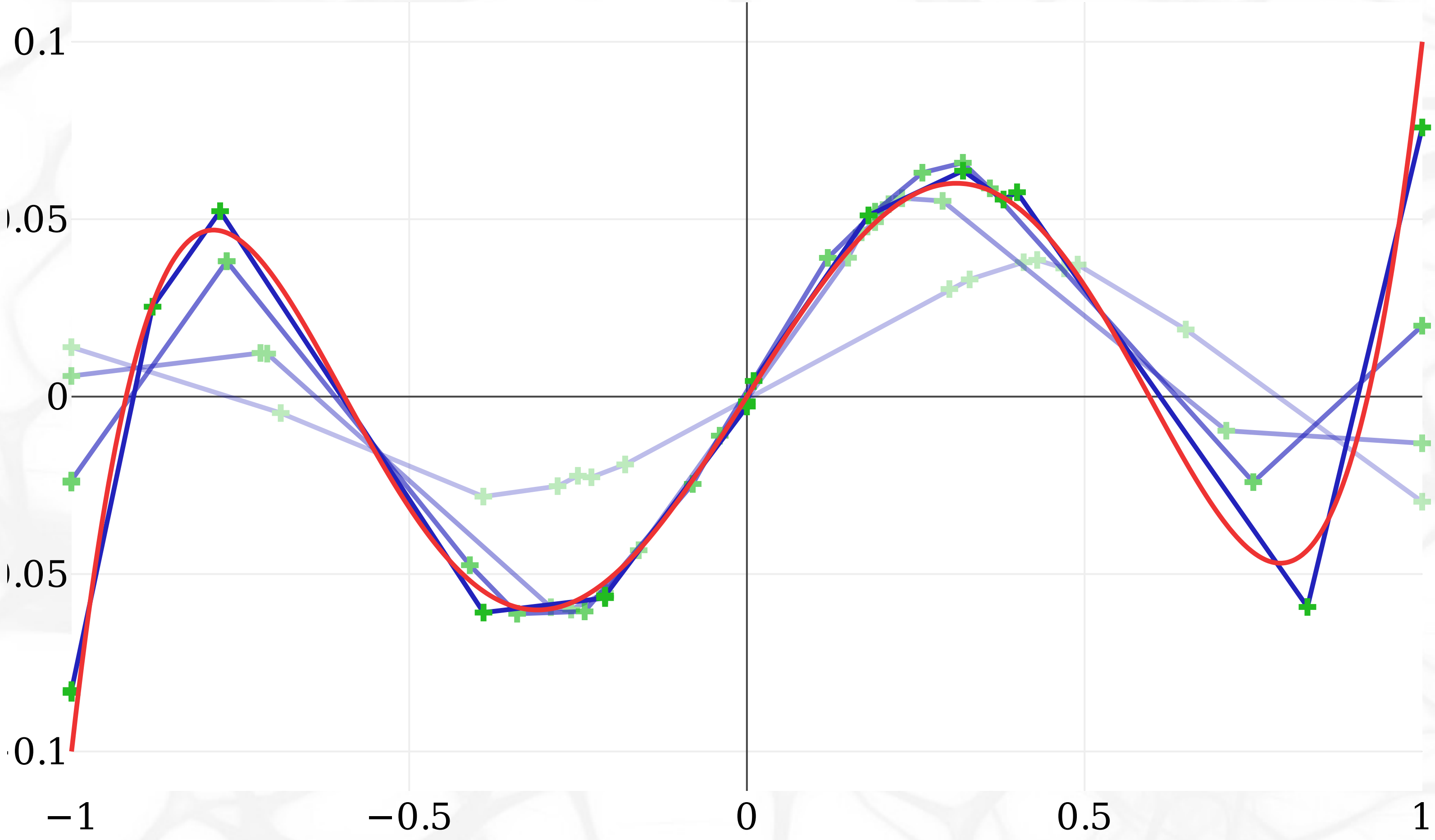
Evolving ReLU Approximation



Evolving ReLU Approximation



Evolving ReLU Approximation



Loss Function

A model is usually trained in order to minimize the *loss* on the training data.

Loss Function

A model is usually trained in order to minimize the *loss* on the training data.

Assuming that a model computes $f(\boldsymbol{x}; \boldsymbol{\theta})$ using parameters $\boldsymbol{\theta}$, the *mean square error* is computed as

$$\sum_i \left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)} \right)^2.$$

Loss Function

A model is usually trained in order to minimize the *loss* on the training data.

Assuming that a model computes $f(\boldsymbol{x}; \boldsymbol{\theta})$ using parameters $\boldsymbol{\theta}$, the *mean square error* is computed as

$$\sum_i \left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)} \right)^2.$$

A common principle used to design loss functions is the *maximum likelihood principle*.

Maximum Likelihood Estimation

Let $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ be training data drawn independently from the data-generating distribution p_data. We denote the empirical data distribution as $\hat{p}_\text{data}$. Let $p_\text{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a family of distributions. The *maximum likelihood estimation* of parameters $\boldsymbol{\theta}$ is:

$$
\begin{aligned}
\boldsymbol{\theta}_\text{ML} &= \arg\max_{\boldsymbol{\theta}} p_\text{model}(\mathbb{X}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \prod\nolimits_{i=1}^m p_\text{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \sum\nolimits_{i=1}^m -\log p_\text{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_\text{data}} [-\log p_\text{model}(\boldsymbol{x}; \boldsymbol{\theta})] \\
&= \arg\min_{\boldsymbol{\theta}} H(\hat{p}_\text{data}, p_\text{model}(\boldsymbol{x}; \boldsymbol{\theta})) \\
&= \arg\min_{\boldsymbol{\theta}} D_\text{KL}(\hat{p}_\text{data} || p_\text{model}(\boldsymbol{x}; \boldsymbol{\theta})) + H(\hat{p}_\text{data})
\end{aligned}
$$

Maximum Likelihood Estimation

Easily generalized to situations where our goal is predict y given $\boldsymbol{x}$.

$$\begin{aligned}\boldsymbol{\theta}_{\mathrm{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{Y}|\mathbb{X};\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod\nolimits_{i=1}^{m} p_{\mathrm{model}}(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\ &= \arg\min_{\boldsymbol{\theta}} \sum\nolimits_{i=1}^{m} -\log p_{\mathrm{model}}(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta})\end{aligned}$$

The resulting *loss function* is called *negative log likelihood*, or *cross-entropy* or *Kullback-Leibler divegence*.

Gradient Descent

Let a model compute $f(\boldsymbol{x}; \boldsymbol{\theta})$ using parameters $\boldsymbol{\theta}$. In order to compute

$$J(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_\text{data}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y),$$

we may use *gradient descent*:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

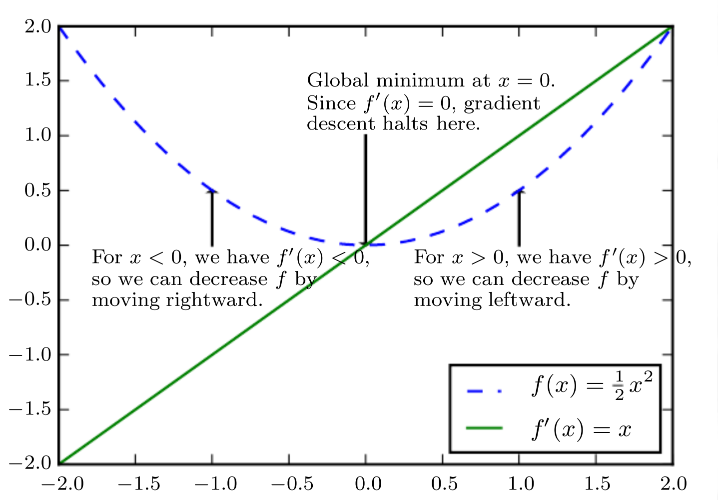


Figure 4.1, page 83 of Deep Learning Book, http://deeplearningbook.org

Gradient Descent

We use all training data to compute $J(\boldsymbol{\theta})$.

Online (or Stochastic) Gradient Descent

We estimate the expectation in $J(\boldsymbol{\theta})$ using a single randomly sampled example from the training data. Such an estimate is unbiased, but very noisy.

Minibatch SGD

The minibatch SGD is a trade-off between gradient descent and SGD – the expectation in $J(\boldsymbol{\theta})$ is estimated using m random independent examples from the training data.

Gradient Descent

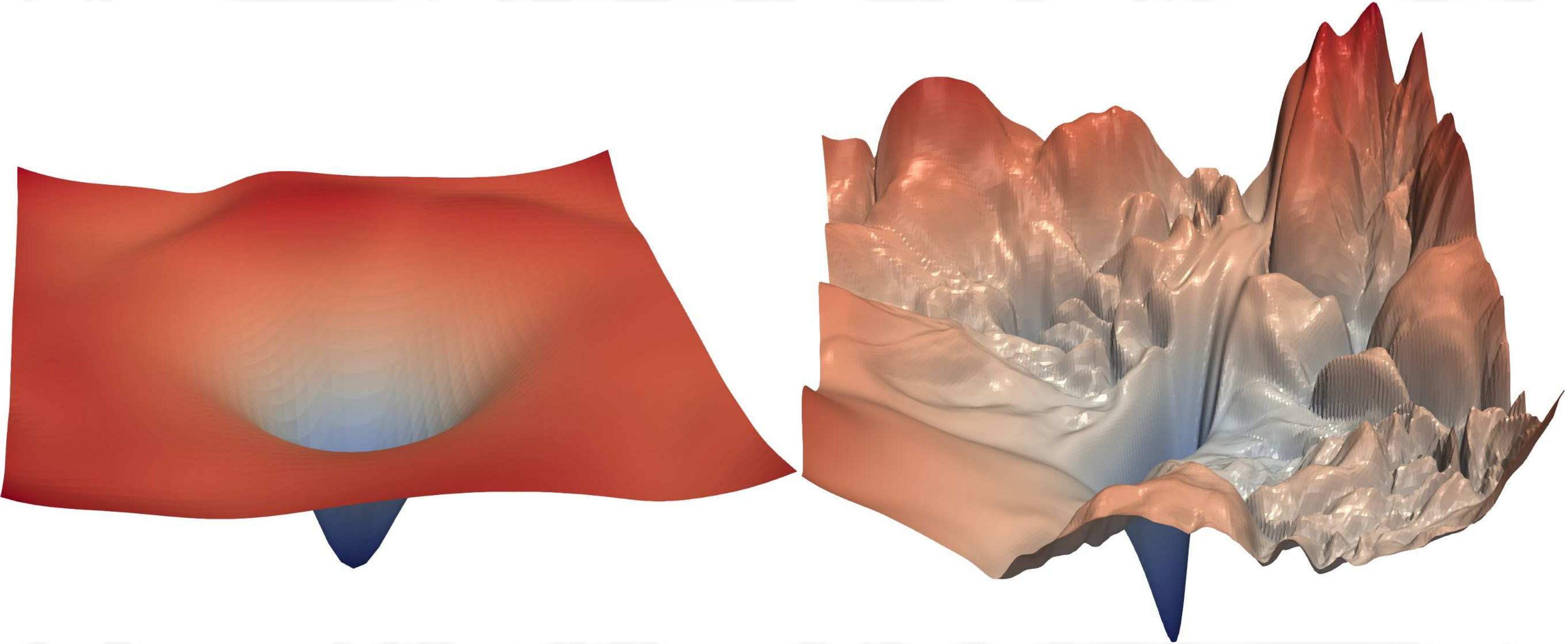
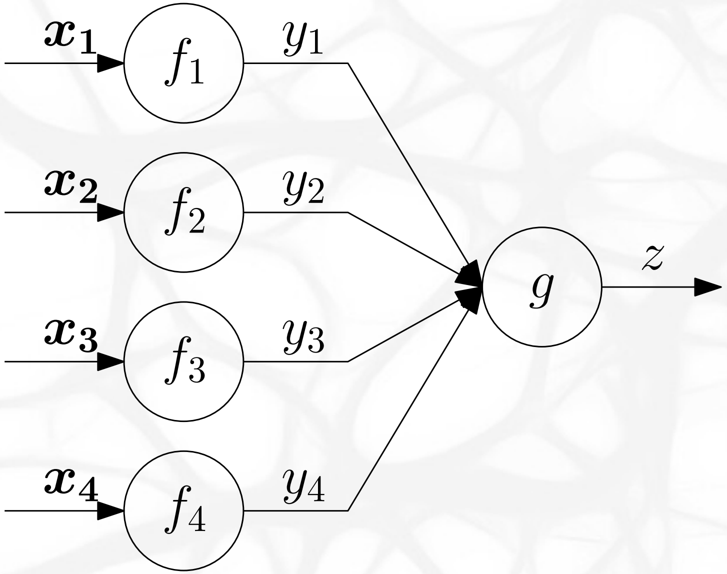


Figure 1 of paper "Visualizing the Loss Landscape of Neural Nets", https://arxiv.org/abs/1712.09913.

Backpropagation

Assume we want to compute partial derivatives of a given loss function J and let $\frac{\partial J}{\partial z}$ be known.



$$\frac{\partial J}{\partial y_i} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial y_i} = \frac{\partial J}{\partial z}\frac{\partial g(\boldsymbol{y})}{\partial y_i}$$

$$\frac{\partial J}{\partial \boldsymbol{x}_i} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial \boldsymbol{x}_i} = \frac{\partial J}{\partial z}\frac{\partial g(\boldsymbol{y})}{\partial y_i}\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i}$$

Backpropagation Algorithm

Simple Variant of Backpropagation

Inputs: The network as in the Forward propagation algorithm.

Outputs: Partial derivatives $g^{(i)} = \frac{\partial u^{(n)}}{\partial u^{(i)}}$ of $u^{(n)}$ with respect to all $u^{(i)}$.

- Run forward propagation to compute all $u^{(i)}$
- $g^{(n)} = 1$
- For $i = n - 1, \ldots, 1$:
  - $g^{(i)} \leftarrow \sum_{j: i \in P(u^{(j)})} g^{(j)} \frac{\partial u^{(j)}}{\partial u^{(i)}}$
- Return $\boldsymbol{g}$

In practice, we do not usually represent networks as collections of scalar nodes; instead we represent them as collections of tensor functions – most usually functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Then $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is a Jacobian. However, the backpropagation algorithm is analogous.

Stochastic Gradient Descent (SGD) Algorithm

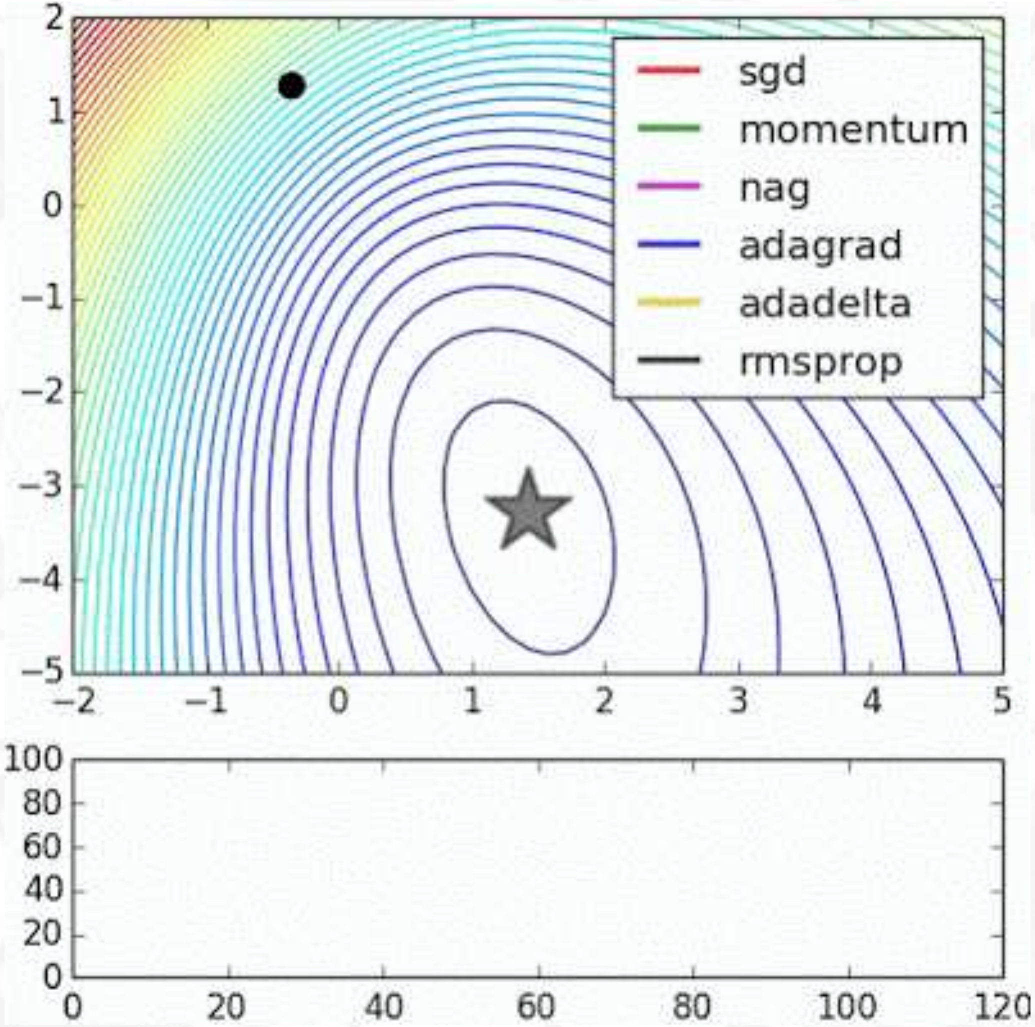
Inputs: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

Inputs: Learning rate α.

Outputs: Updated parameters $\boldsymbol{\theta}$.

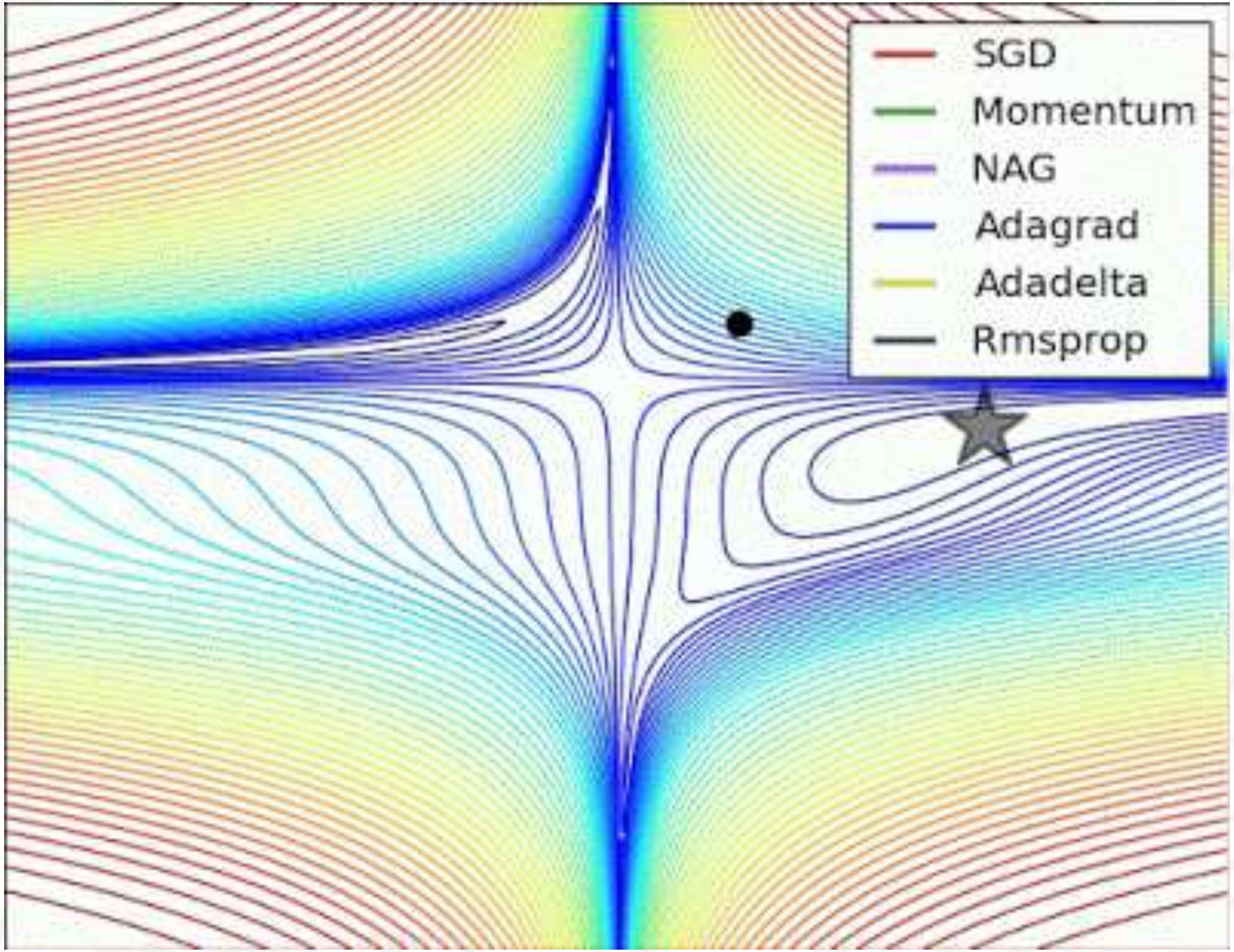
- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{g}$

Adaptive Optimizers Animations



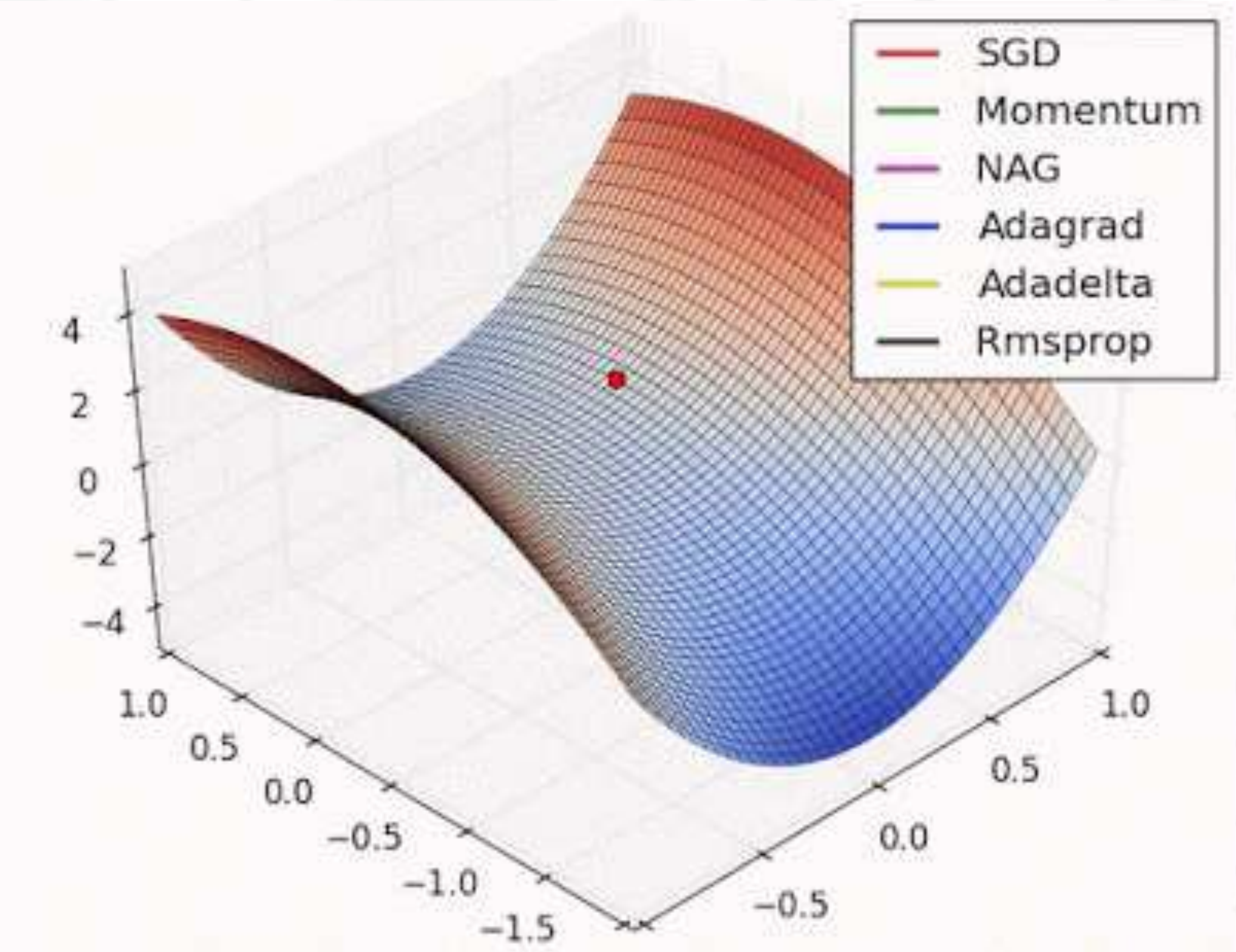
http://2.bp.blogspot.com/-q6l20Vs4P_w/VPmIC7sEhnI/AAAAAAAACC4/g3UOUX2r_yA/s400/ found at http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Adaptive Optimizers Animations



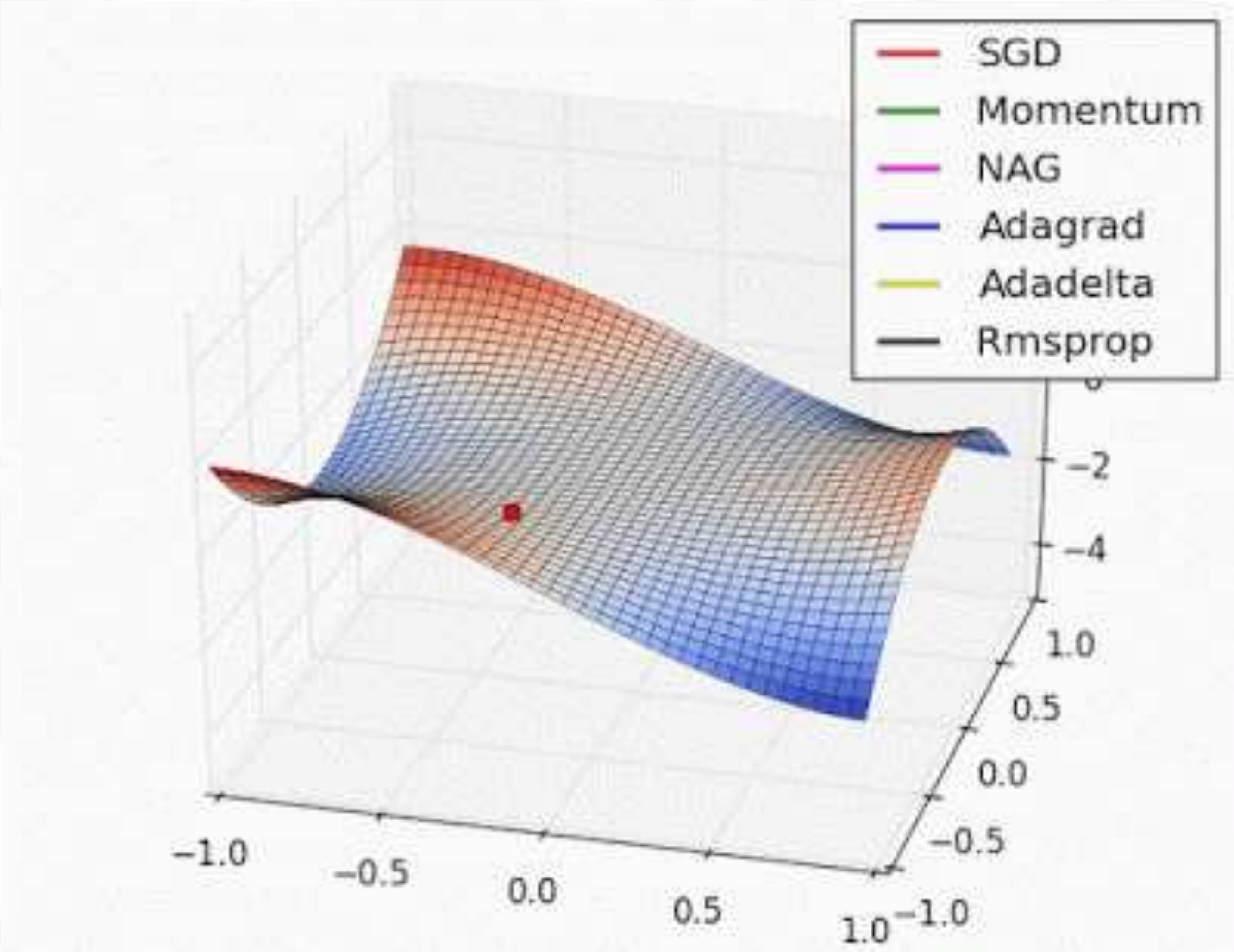
http://2.bp.blogspot.com/-L98w-SBmF58/VPmICIjKEKI/AAAAAAAACCs/rrFz3VetYmM/s400/ found at http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Adaptive Optimizers Animations



http://3.bp.blogspot.com/-nrtJPrdBWuE/VPmIB46F2aI/AAAAAAAACCw/vaE_B0SVy5k/s400/ found at http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Adaptive Optimizers Animations



http://1.bp.blogspot.com/-K_X-yud8nj8/VPmIBxwGlsI/AAAAAAAACC0/JS-h1fa09EQ/s400/ found at http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Neural Networks Demos

- TensorFlow Playground
- TensorFlow.js
- Sketch RNN Demo
- MetaCar

High Level Overview

	Classical ('90s)	Deep Learning
Architecture	$\vdots\ \vdots\ \vdots$	$\vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots$ CNN, RNN, VAE, GAN, …
Activation func.	$\tanh, \sigma$	$\tanh$, ReLU, PReLU, ELU, SELU, Swish, …
Output function	none, σ	none, σ, $\mathrm{softmax}$
Loss function	MSE	NLL (or cross-entropy or KL-divergence)
Optimalization	SGD, momentum	SGD, RMSProp, Adam, …
Regularization	L2, L1	L2, Dropout, BatchNorm, LayerNorm, …

Regularization – Dropout

How to design good universal features?

- In reproduction, evolution is achieved using gene swapping. The genes must not be just good with combination with other genes, they need to be universally good.

Regularization – Dropout

How to design good universal features?

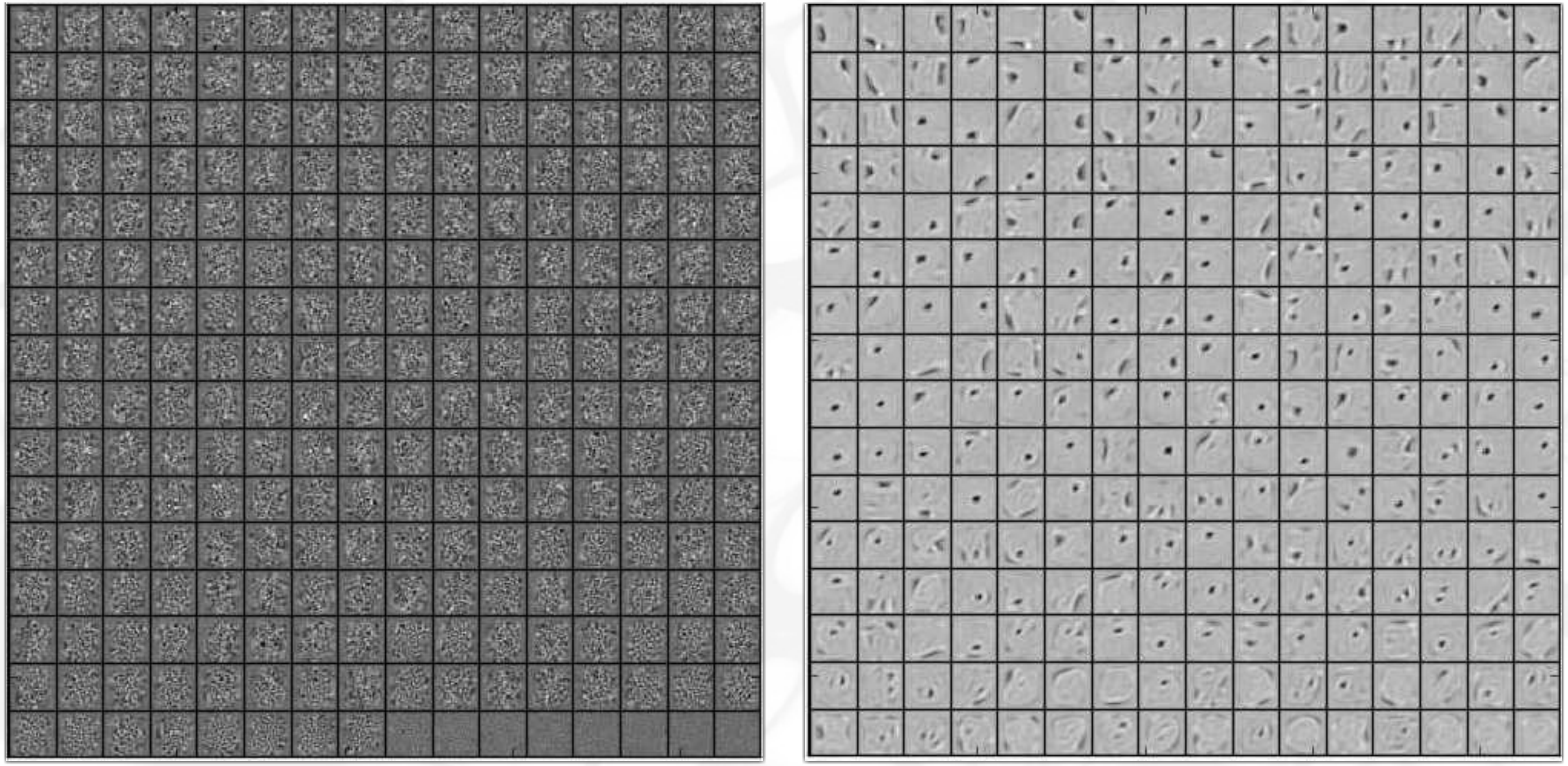
- In reproduction, evolution is achieved using gene swapping. The genes must not be just good with combination with other genes, they need to be universally good.

Idea of *dropout* by (Srivastava et al., 2014), in preprint since 2012.

When applying dropout to a layer, we drop each neuron independently with a probability of p (usually called *dropout rate*). To the rest of the network, the dropped neurons have value of zero.

Dropout is performed only when training, during inference no nodes are dropped. However, in that case we need to *scale the activations down* by a factor of $1 - p$ to account for more neurons than usual.

Alternatively, we might *scale the activations up* during training by a factor of $1/(1 - p)$.



(a) Without dropout (b) Dropout with $p = 0.5$.

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

Figure 7 of paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Consider data with some structure (temporal data, speech, images, ...).

Unlike densely connected layers, we might want:

- Sparse (local) interactions
- Parameter sharing (equal response everywhere)
- Shift invariance

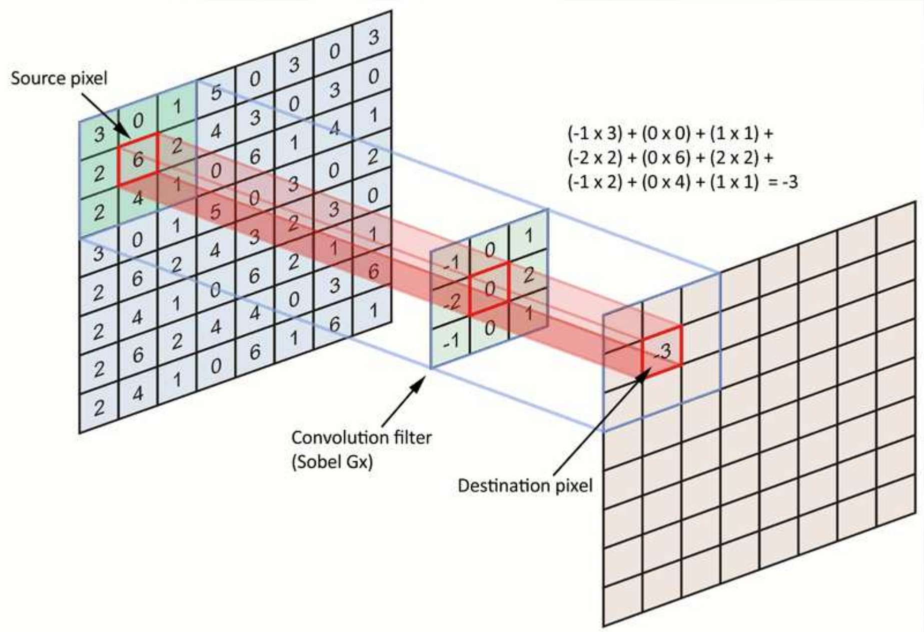
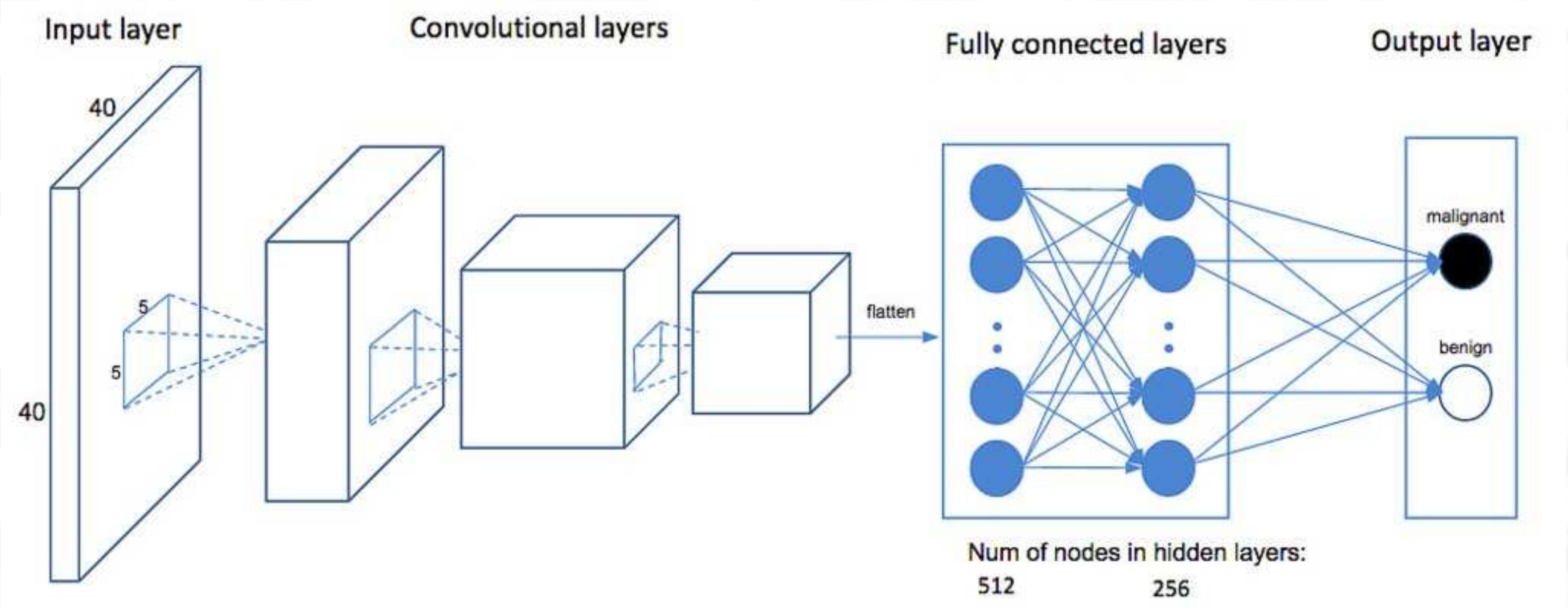


Image from https://i.stack.imgur.com/YDusp.png.

High-level CNN Architecture

We repeatedly use the following block:

1. Convolution operation
2. Non-linear activation (usually ReLU)
3. Pooling



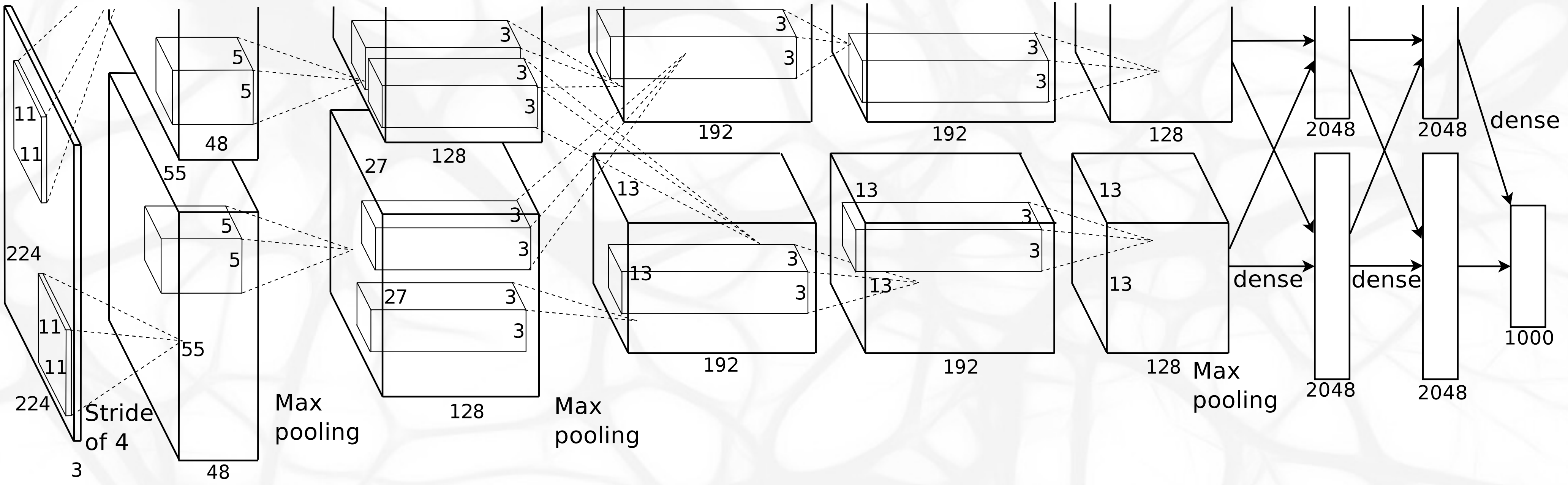


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253, 40–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure 2 of paper "ImageNet Classification with Deep Convolutional Neural Networks", https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Similarities in V1 and CNNs

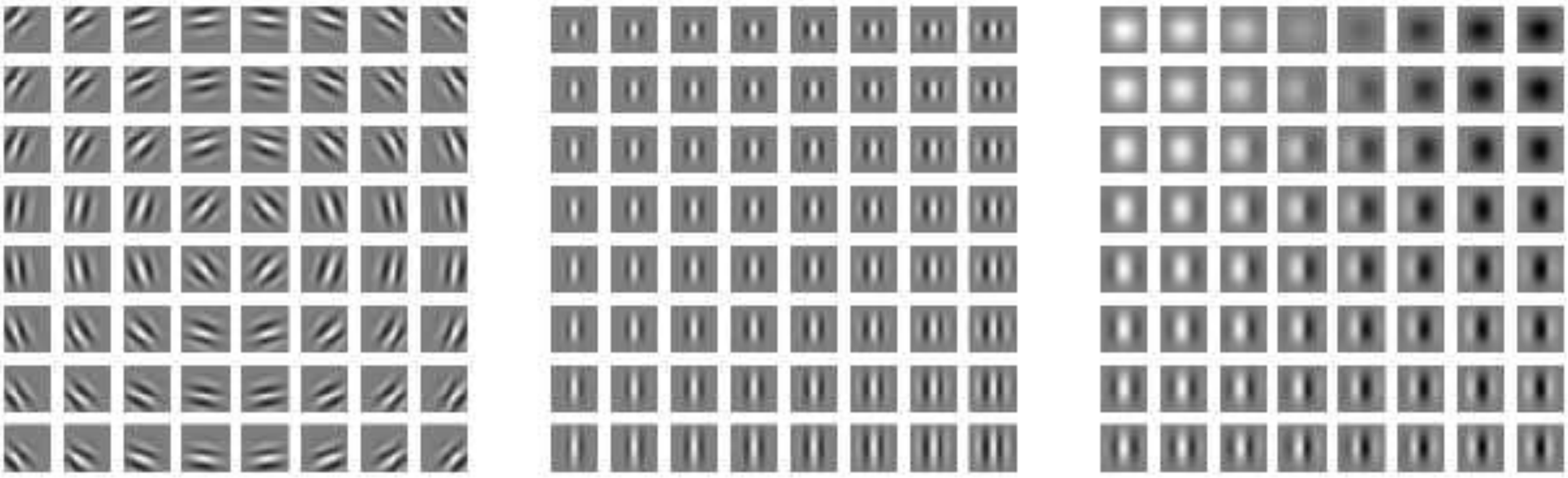


Figure 9.18, page 370 of Deep Learning Book, http://deeplearningbook.org

The primary visual cortex recognizes Gabor functions.

Similarities in V1 and CNNs

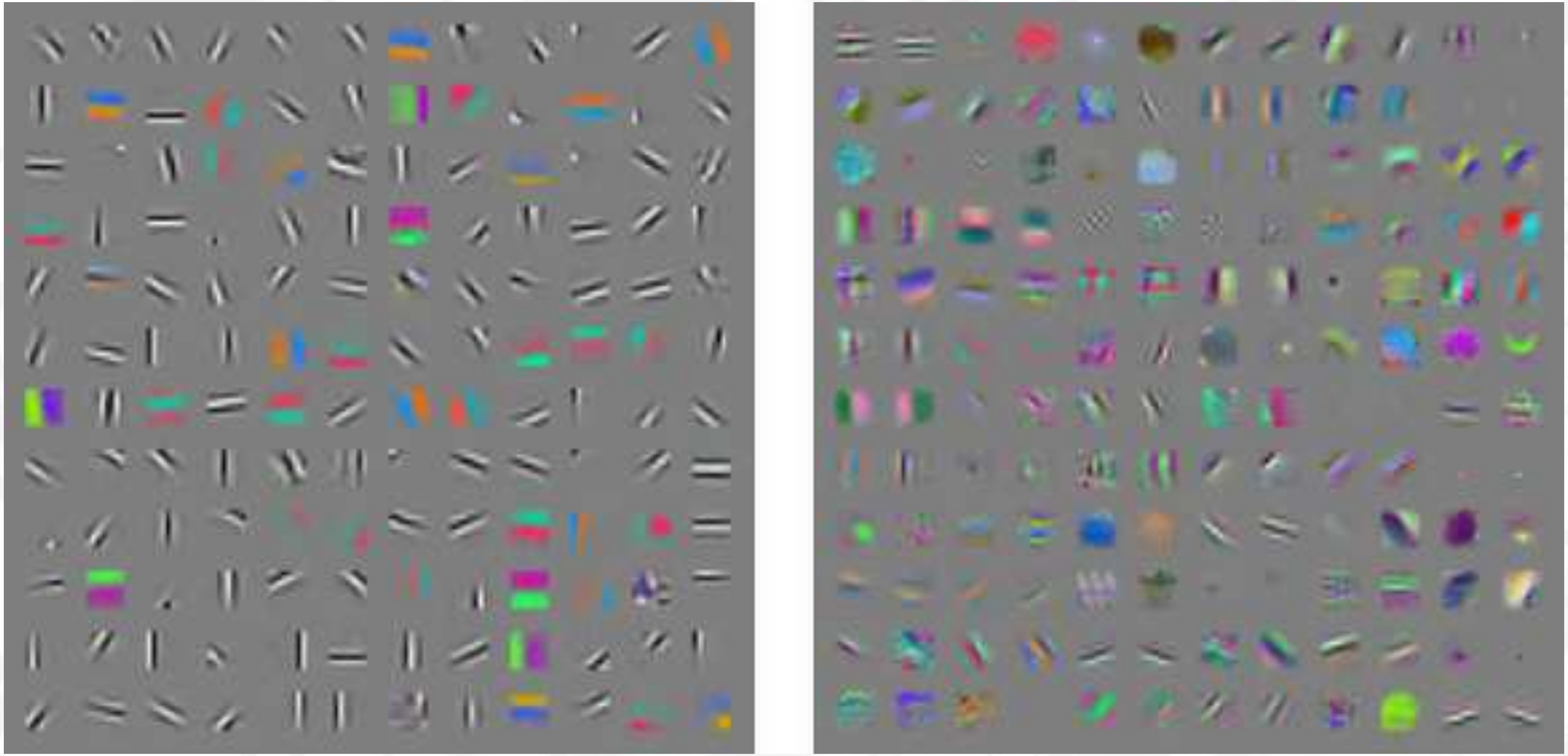


Figure 9.19, page 371 of Deep Learning Book, http://deeplearningbook.org

Similar functions are recognized in the first layer of a CNN.

CNNs as Regularizers – Deep Prior

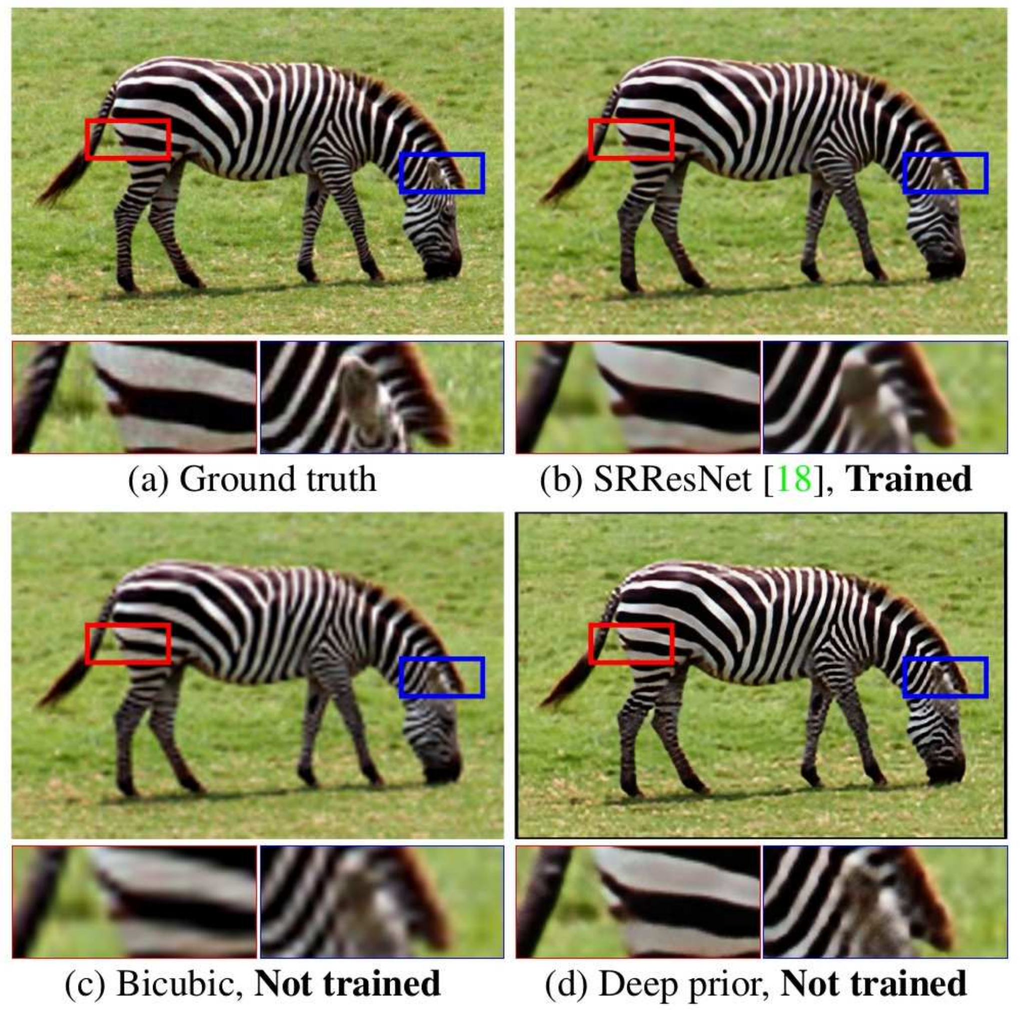


Figure 1 of paper "Deep Prior", https://arxiv.org/abs/1712.05016

CNNs as Regularizers – Deep Prior

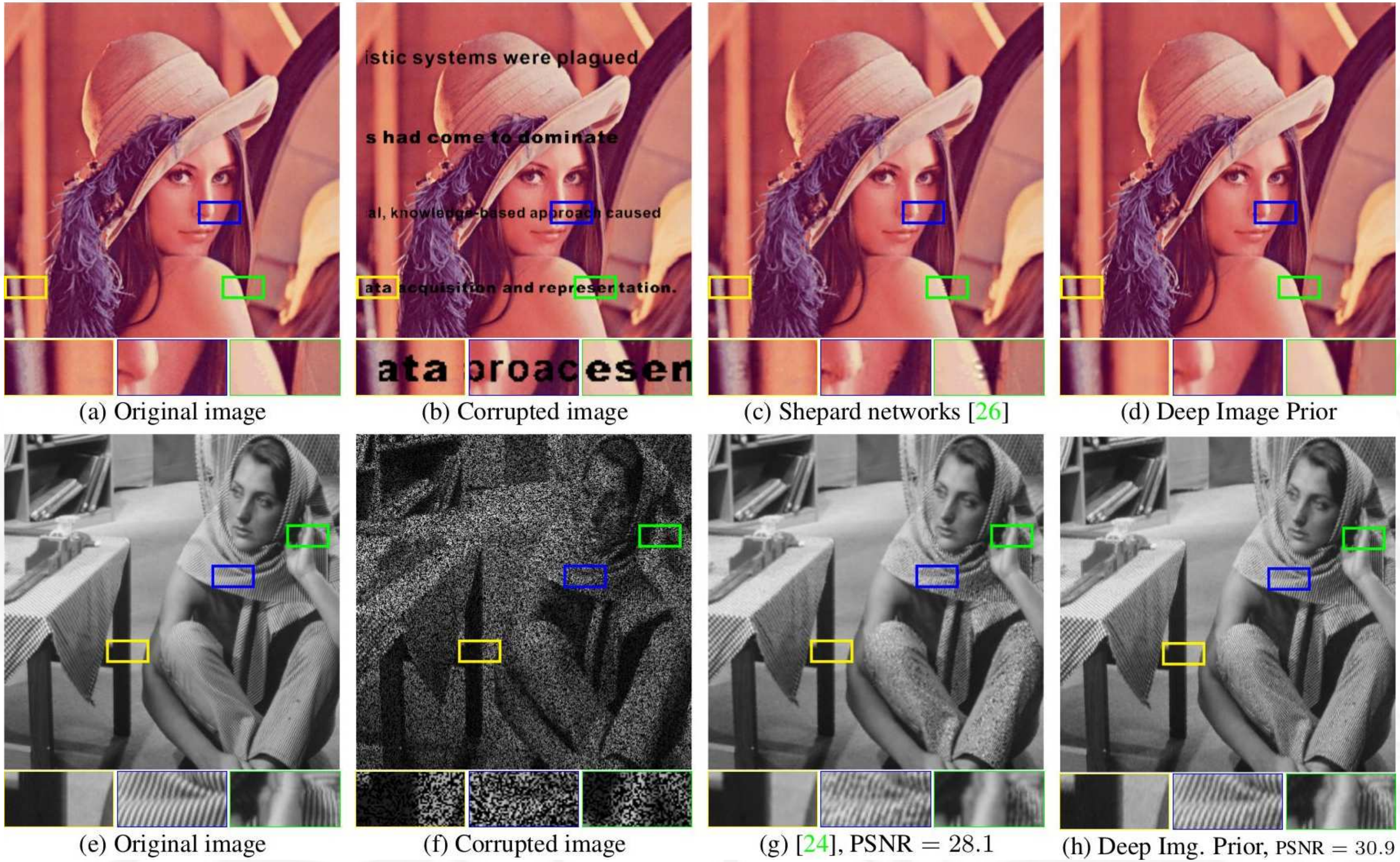


Figure 7 of paper "Deep Prior", https://arxiv.org/abs/1712.05016

CNNs as Regularizers – Deep Prior

Figure 5: **Inpainting diversity.** Left: original image (black pixels indicate holes). The remaining four images show results obtained using deep prior corresponding to different input vector z.

Figure 5 of supplementary materials of paper "Deep Prior", https://arxiv.org/abs/1712.05016

Inception (GoogLeNet) – 2014 (6.7% error)

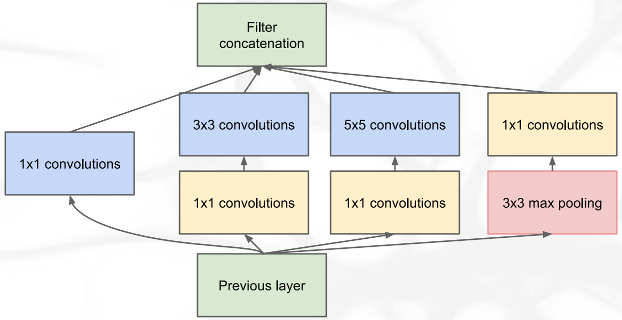


Figure 2 of paper "Going Deeper with Convolutions", https://arxiv.org/abs/1409.4842.

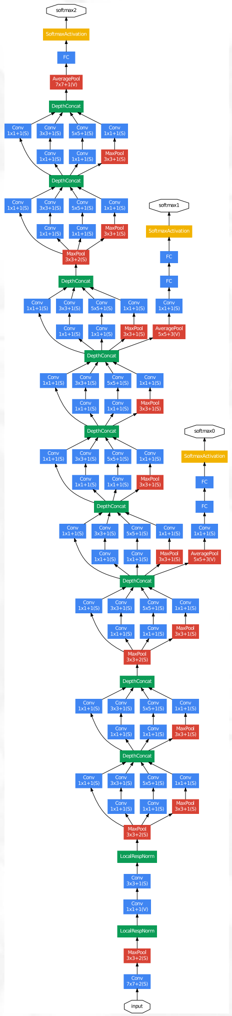


Figure 3 of paper "Going Deeper with

ResNet – 2015 (3.6% error)

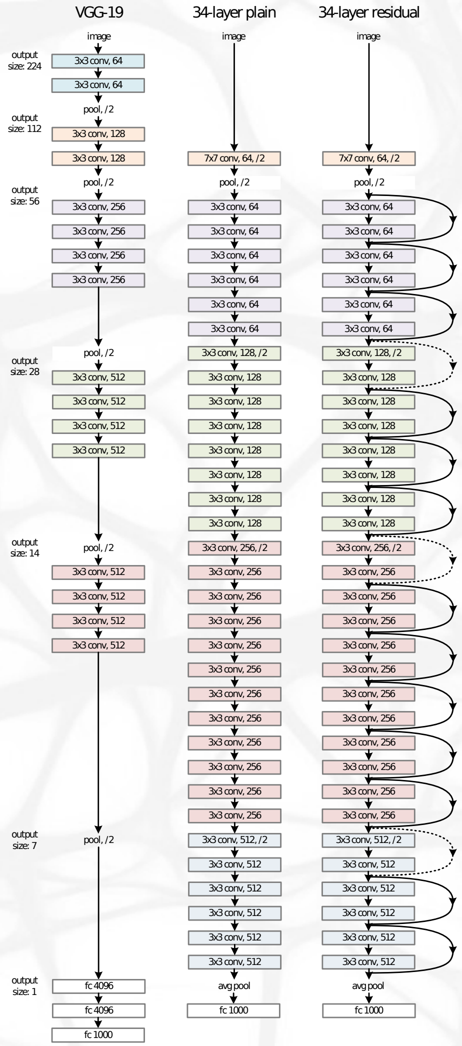


Figure 3 of paper "Deep Residual Learning for Image Recognition", https://arxiv.org/abs/1512.03385.

Beyond Image Classification

Object detection (including location)

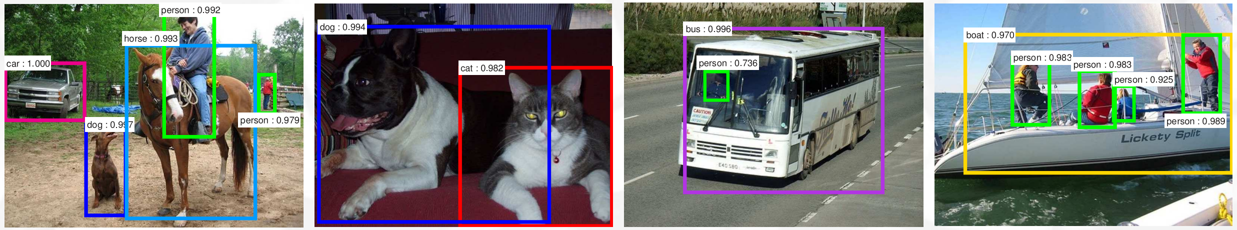


Figure 3 of paper "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

Object detection (including location)

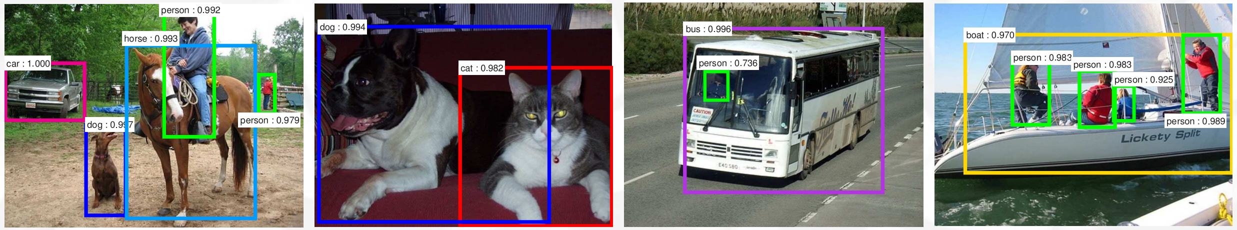


Figure 3 of paper "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

Image segmentation

Figure 2 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

Object detection (including location)

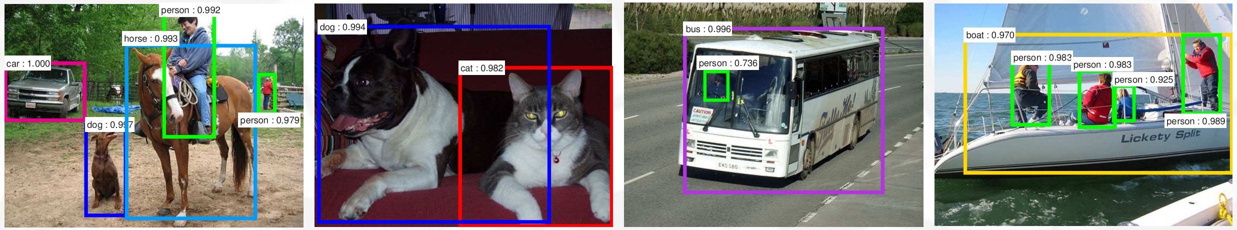


Figure 3 of paper "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", https://arxiv.org/abs/1506.01497

Image segmentation

Figure 2 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

Human pose estimation

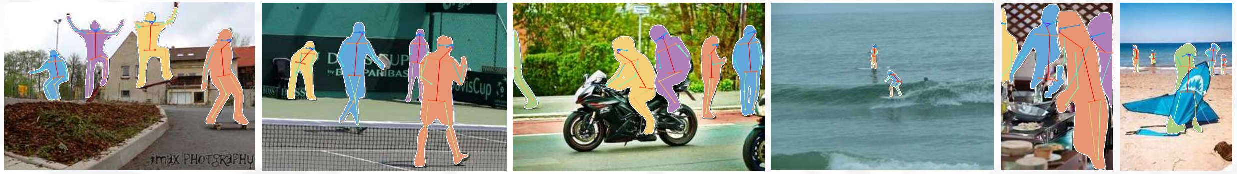
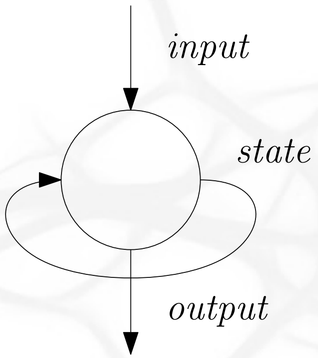
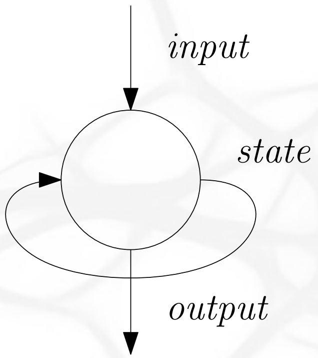


Figure 7 of paper "Mask R-CNN", https://arxiv.org/abs/1703.06870.

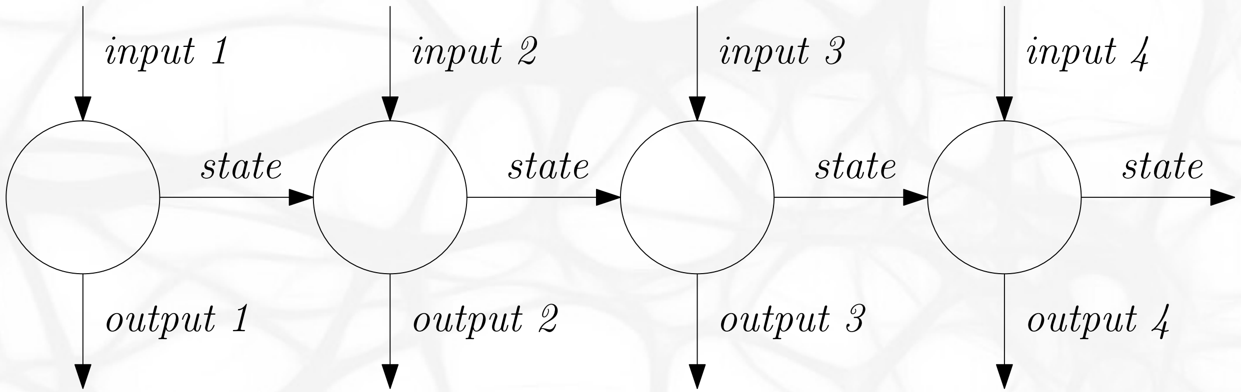
Single RNN cell



Single RNN cell



Unrolled RNN cells



Sequence-to-Sequence Architecture

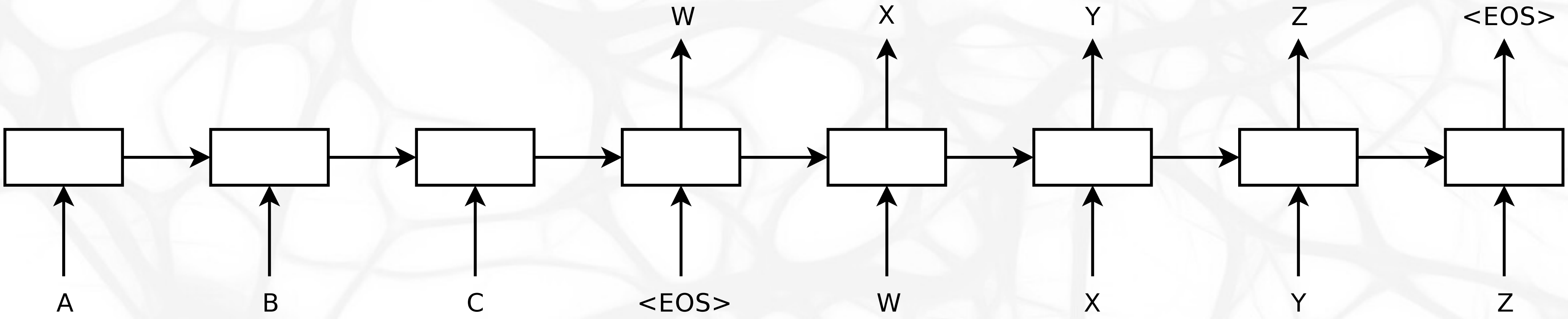


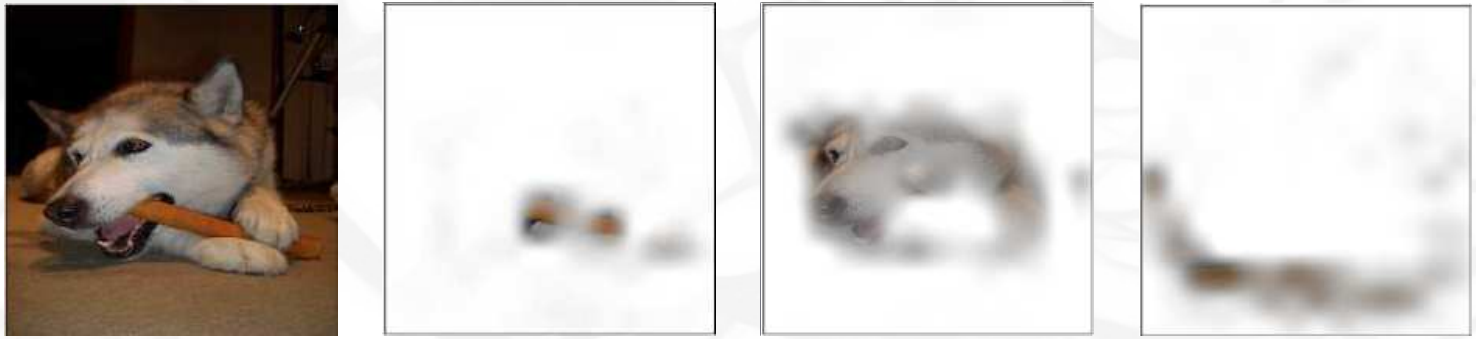
Figure 1 of paper "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473.

Image Labeling

Fig. 5. A selection of evaluation results, grouped by human rating.

Figure 5 of "Show and Tell: Lessons learned from the 2015 MSCOCO...", https://arxiv.org/abs/1609.06647.

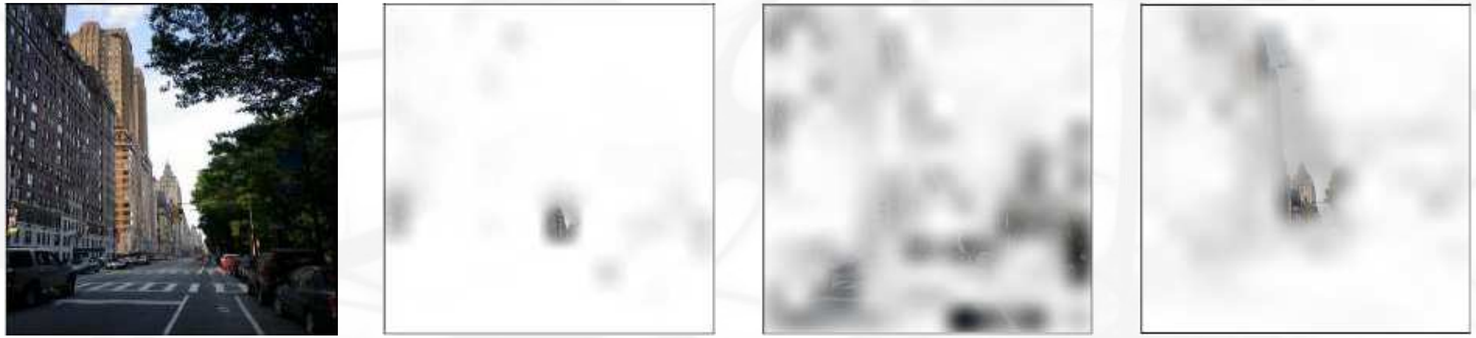
Visual Question Answering



What vegetable is the dog chewing on?
MCB: carrot
GT: carrot

What kind of dog is this?
MCB: husky
GT: husky

What kind of flooring does the room have?
MCB: carpet
GT: carpet



What color is the traffic light?
MCB: green
GT: green

Is this an urban area?
MCB: yes
GT: yes

Where are the buildings?
MCB: in background
GT: on left

Figure 6 of "Multimodal Compact Bilinear Pooling for VQA and Visual Grounding", https://arxiv.org/abs/1606.01847.

Lip Reading

Figure 3. **Top:** Original still images from the BBC lip reading dataset – News, Question Time, Breakfast, Newsnight (from left to right). **Bottom:** The mouth motions for 'afternoon' from two different speakers. The network sees the areas inside the red squares.

Figure 3 of "Lip Reading Sentences in the Wild", https://arxiv.org/abs/1611.05358.

Lip Reading

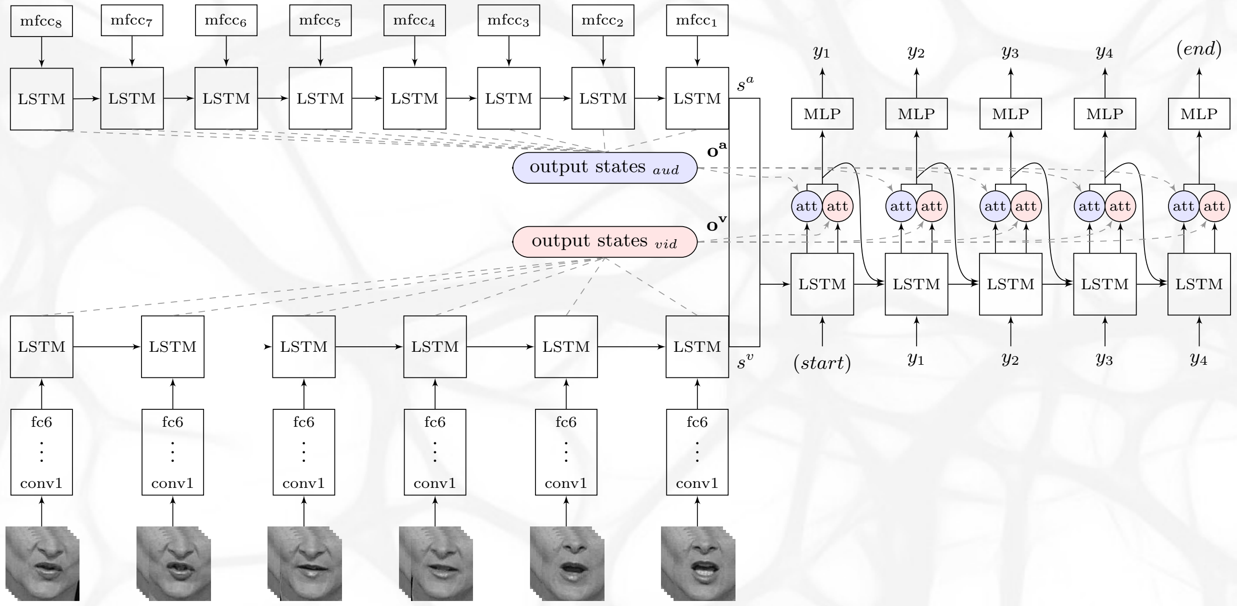


Figure 1 of "Lip Reading Sentences in the Wild", https://arxiv.org/abs/1611.05358.

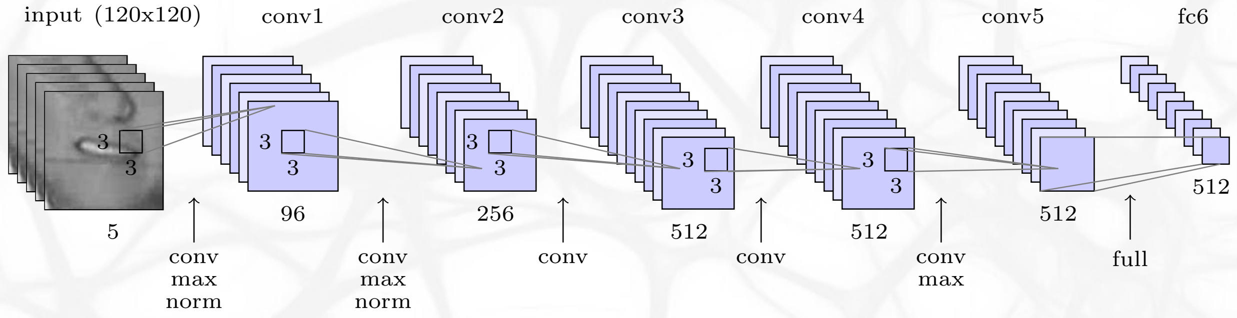


Figure 2 of "Lip Reading Sentences in the Wild", https://arxiv.org/abs/1611.05358.

Lip Reading

Method	SNR	CER	WER	BLEU[†]
Lips only				
Professional[‡]	-	58.7%	73.8%	23.8
WAS	-	59.9%	76.5%	35.6
WAS+CL	-	47.1%	61.1%	46.9
WAS+CL+SS	-	42.4%	58.1%	50.0
WAS+CL+SS+BS	-	39.5%	50.2%	54.9
Audio only				
Google Speech API	clean	17.6%	22.6%	78.4
Kaldi SGMM+MMI[⋆]	clean	9.7%	16.8%	83.6
LAS+CL+SS+BS	clean	10.4%	17.7%	84.0
LAS+CL+SS+BS	10dB	26.2%	37.6%	66.4
LAS+CL+SS+BS	0dB	50.3%	62.9%	44.6
Audio and lips				
WLAS+CL+SS+BS	clean	7.9%	13.9%	87.4
WLAS+CL+SS+BS	10dB	17.6%	27.6%	75.3
WLAS+CL+SS+BS	0dB	29.8%	42.0%	63.1

Table 5 of "Lip Reading Sentences in the Wild", https://arxiv.org/abs/1611.05358.

Lip Reading

GT	IT WILL BE THE CONSUMERS
A	IN WILL BE THE CONSUMERS
L	IT WILL BE IN THE CONSUMERS
AV	IT WILL BE THE CONSUMERS
GT	CHILDREN IN EDINBURGH
A	CHILDREN AND EDINBURGH
L	CHILDREN AND HANDED BROKE
AV	CHILDREN IN EDINBURGH
GT	JUSTICE AND EVERYTHING ELSE
A	JUST GETTING EVERYTHING ELSE
L	CHINESES AND EVERYTHING ELSE
AV	JUSTICE AND EVERYTHING ELSE

Table 7 of "Lip Reading Sentences in the Wild", https://arxiv.org/abs/1611.05358.

Lip Reading

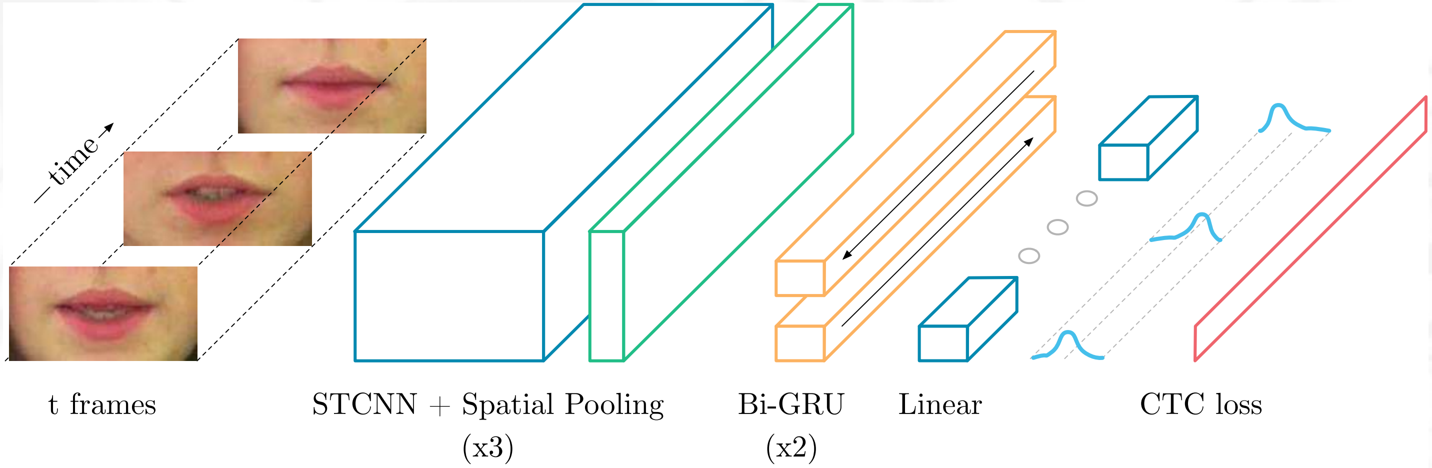


Figure 1 of "LipNet: End-to-end Sentence-level Lipreading", https://arxiv.org/abs/1611.01599.

Lip Reading

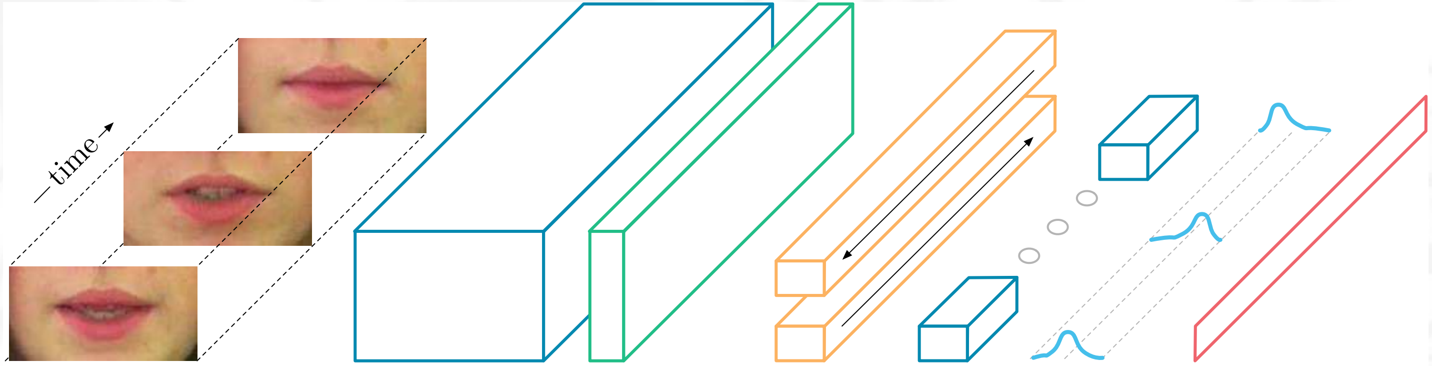


Figure 1 of "LipNet: End-to-end Sentence-level Lipreading", https://arxiv.org/abs/1611.01599.

Method	Unseen Speakers CER	Unseen Speakers WER	Overlapped Speakers CER	Overlapped Speakers WER
Hearing-Impaired Person (avg)	–	47.7%	–	–
Baseline-LSTM	38.4%	52.8%	15.2%	26.3%
Baseline-2D	16.2%	26.7%	4.3%	11.6%
Baseline-NoLM	6.7%	13.6%	2.0%	5.6%
LipNet	**6.4%**	**11.4%**	**1.9%**	**4.8%**

Table 2 of "LipNet: End-to-end Sentence-level Lipreading", https://arxiv.org/abs/1611.01599.

Deep Q Network

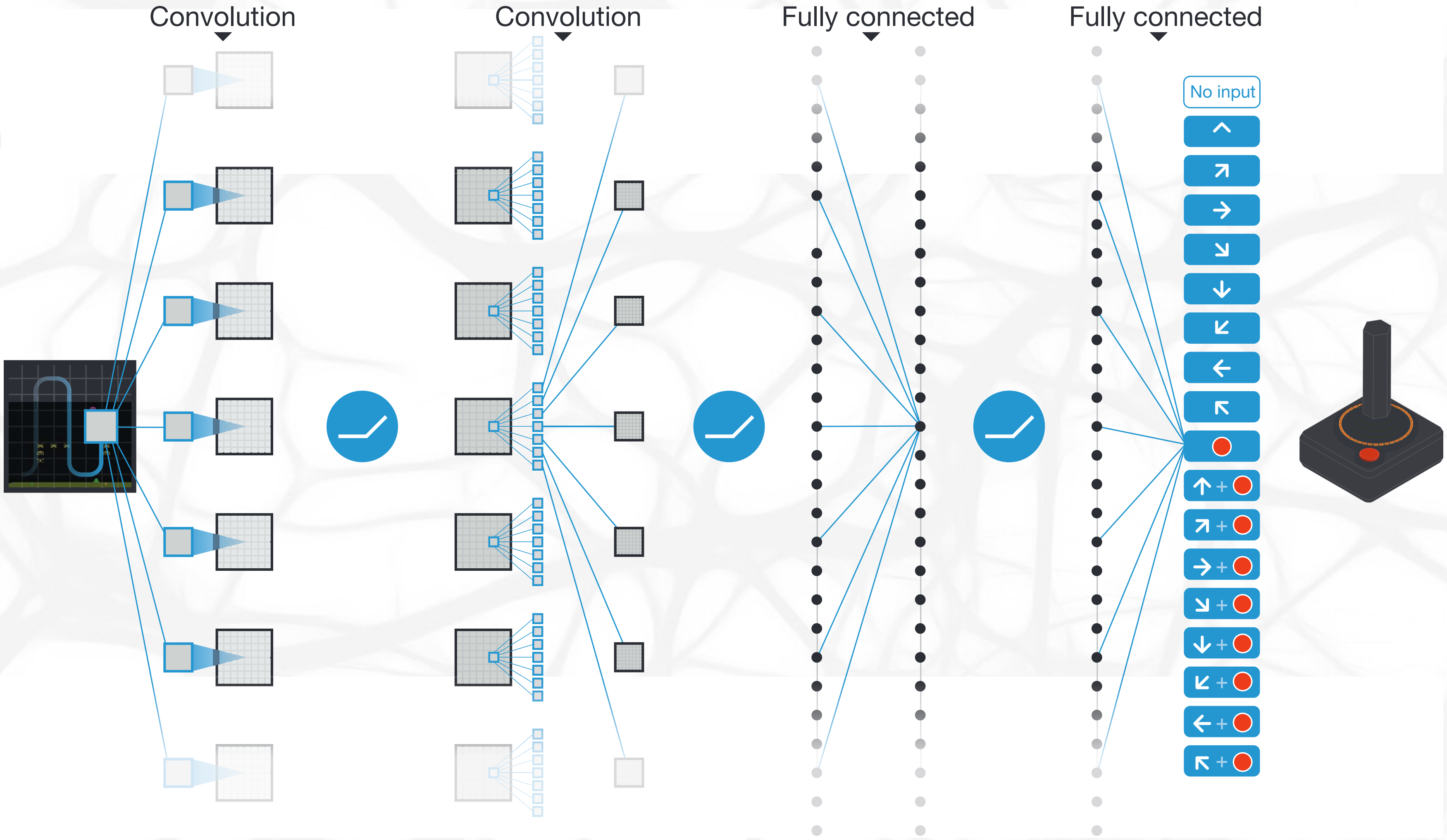
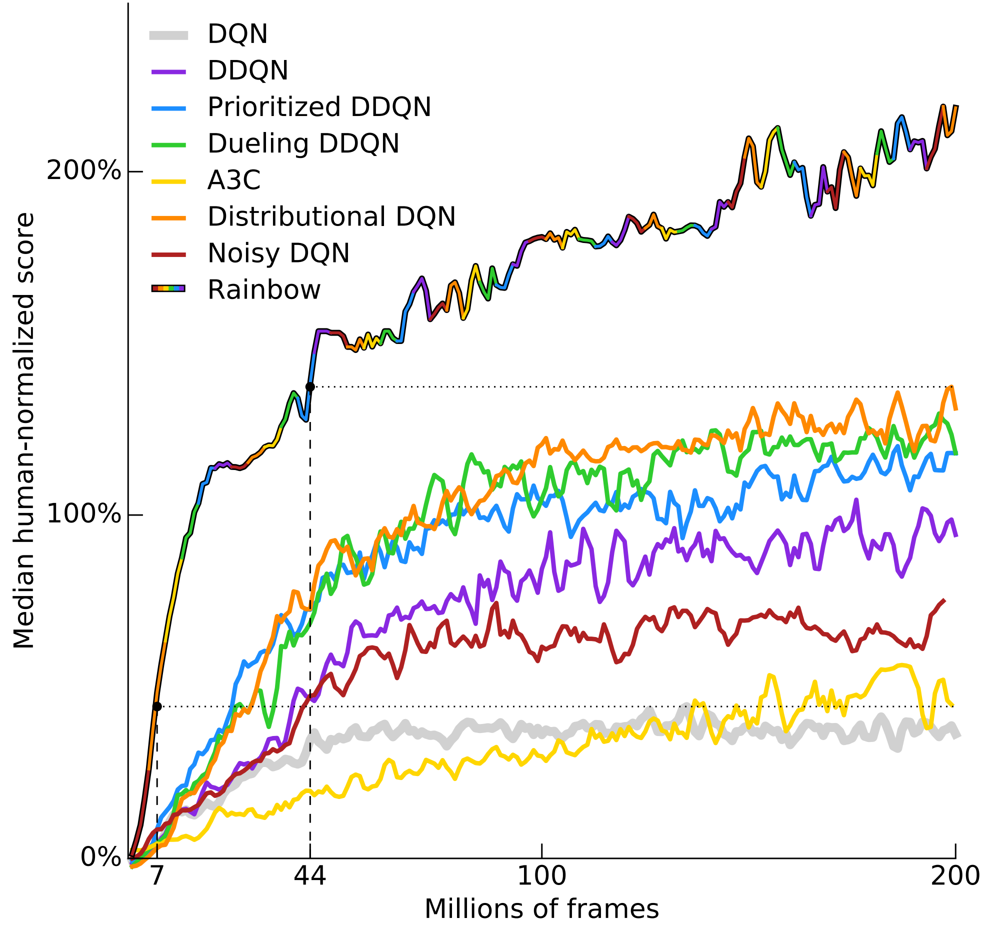


Figure 1 of the paper "Human-level control through deep reinforcement learning" by Volodymyr Mnih et al.

Rainbow



Agent	no-ops	human starts
DQN	79%	68%
DDQN (*)	117%	110%
Prioritized DDQN (*)	140%	128%
Dueling DDQN (*)	151%	117%
A3C (*)	-	116%
Noisy DQN	118%	102%
Distributional DQN	164%	125%
Rainbow	223%	153%

Table 2 of the paper "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

On 7 December 2018, the AlphaZero paper came out in Science journal. It demonstrates learning chess, shogi and go, *tabula rasa* – without any domain-specific human knowledge or data, only using self-play. The evaluation is performed against strongest programs available.

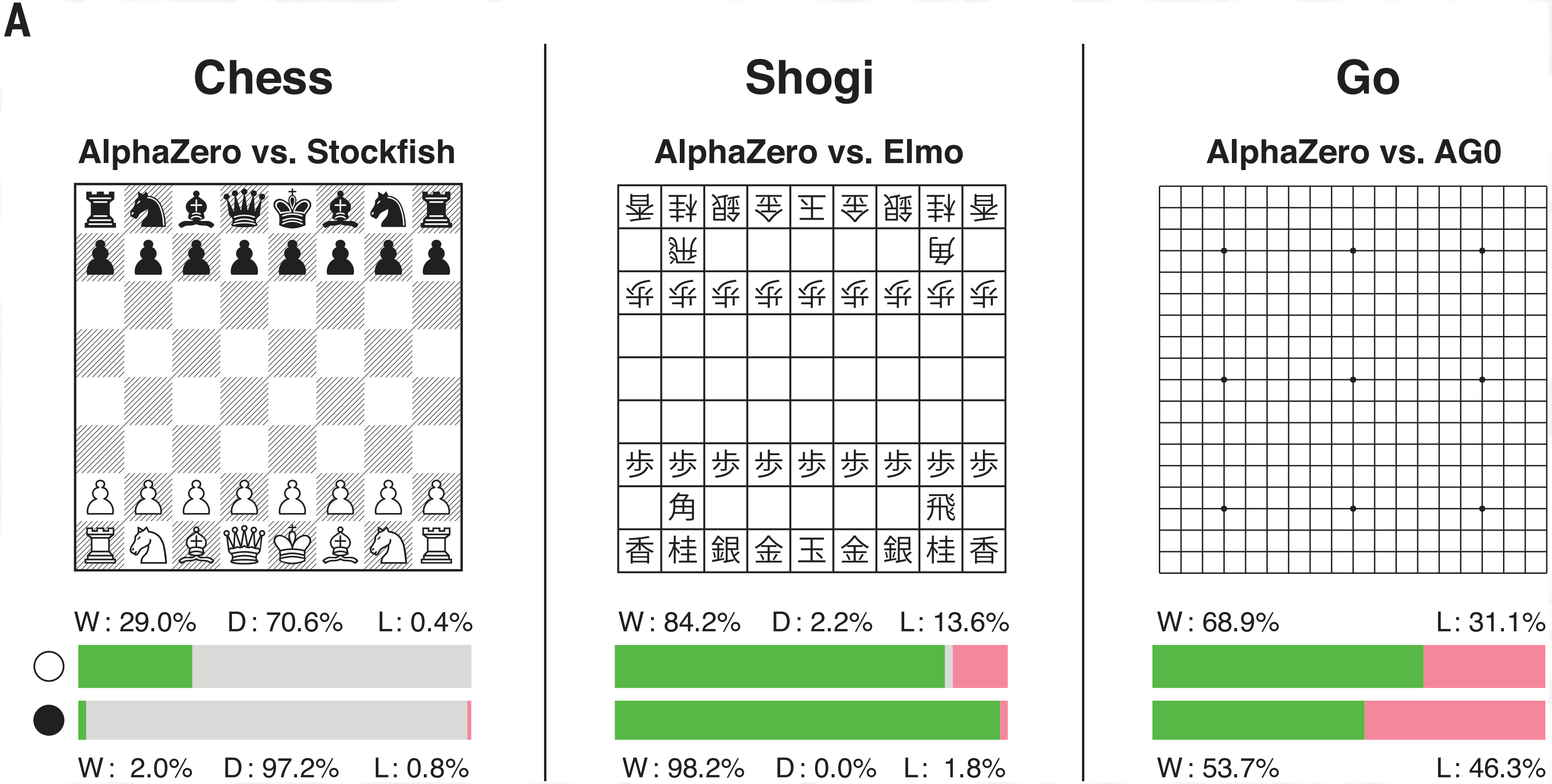


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

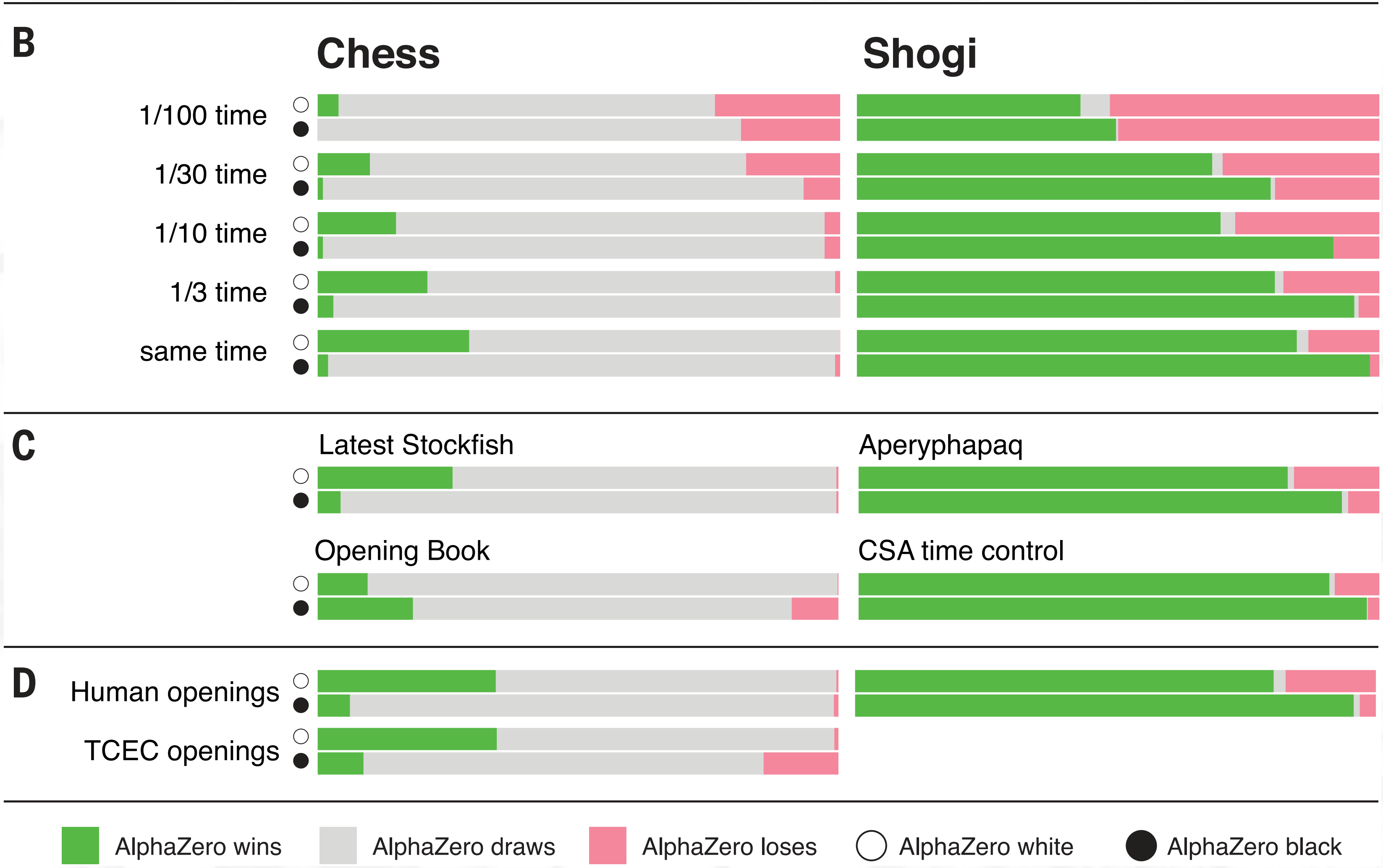


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Training

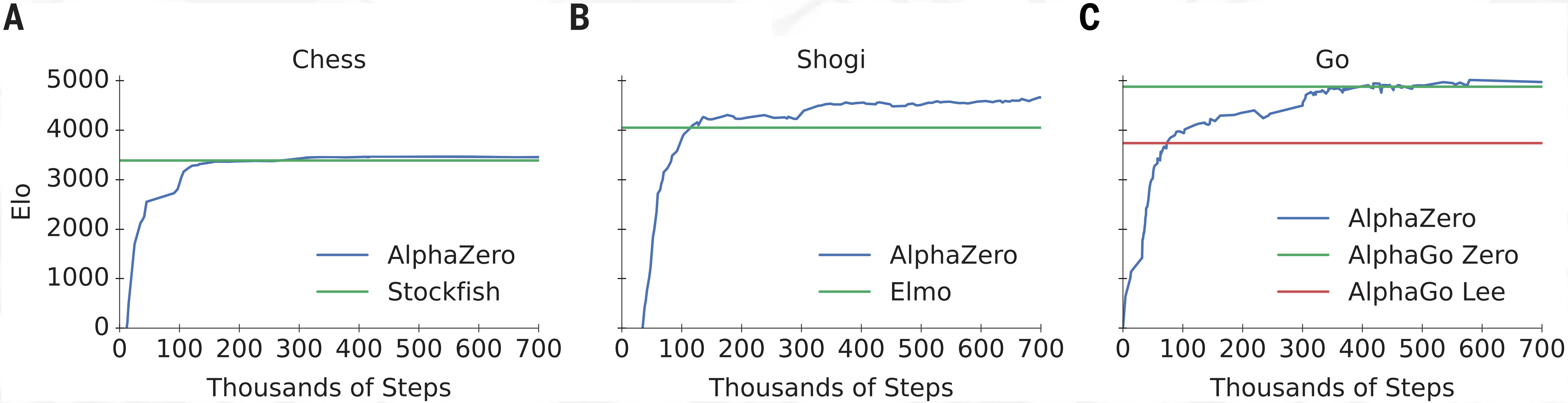


Figure 1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

	Chess	Shogi	Go
Mini-batches	700k	700k	700k
Training Time	9h	12h	13d
Training Games	44 million	24 million	140 million
Thinking Time	800 sims	800 sims	800 sims
	~ 40 ms	~ 80 ms	~ 200 ms

Table S3 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

For the Win agent for Capture The Flag

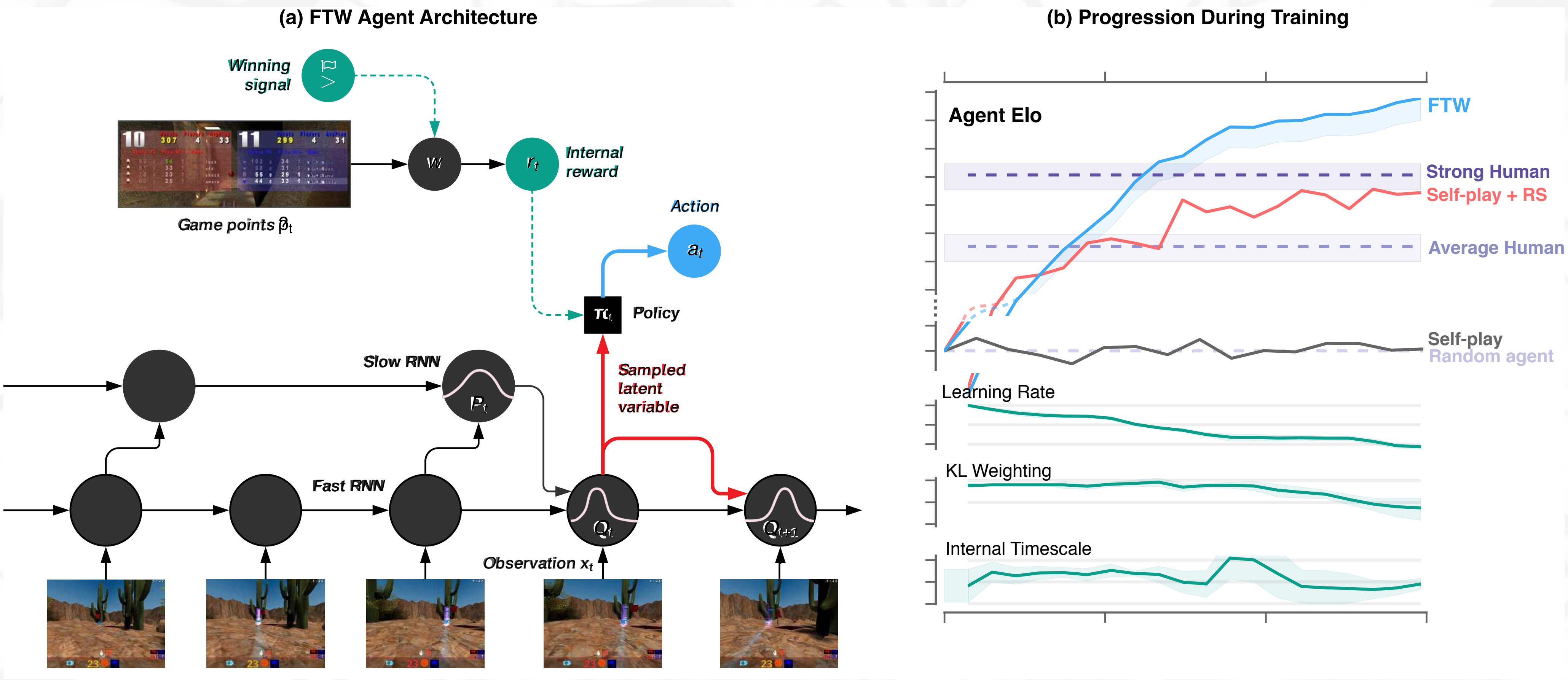


Figure 2 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

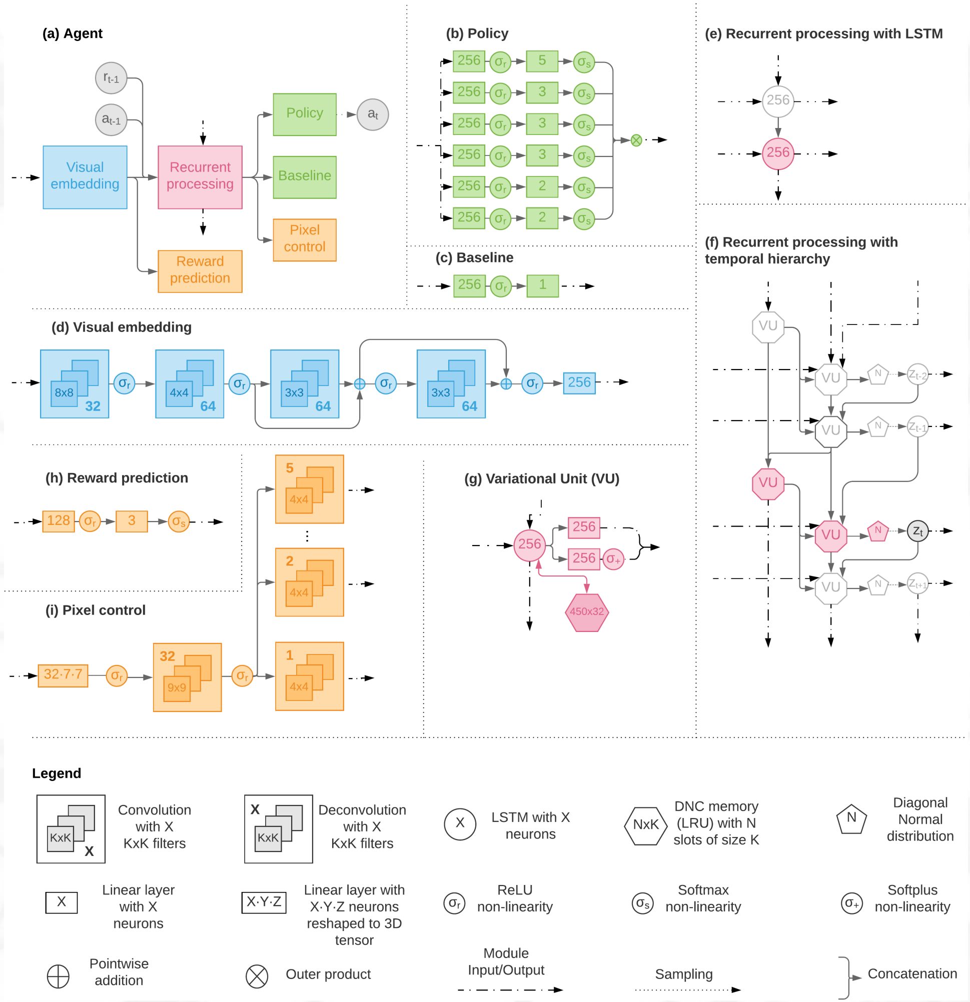


Figure S10 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

For the Win agent for Capture The Flag

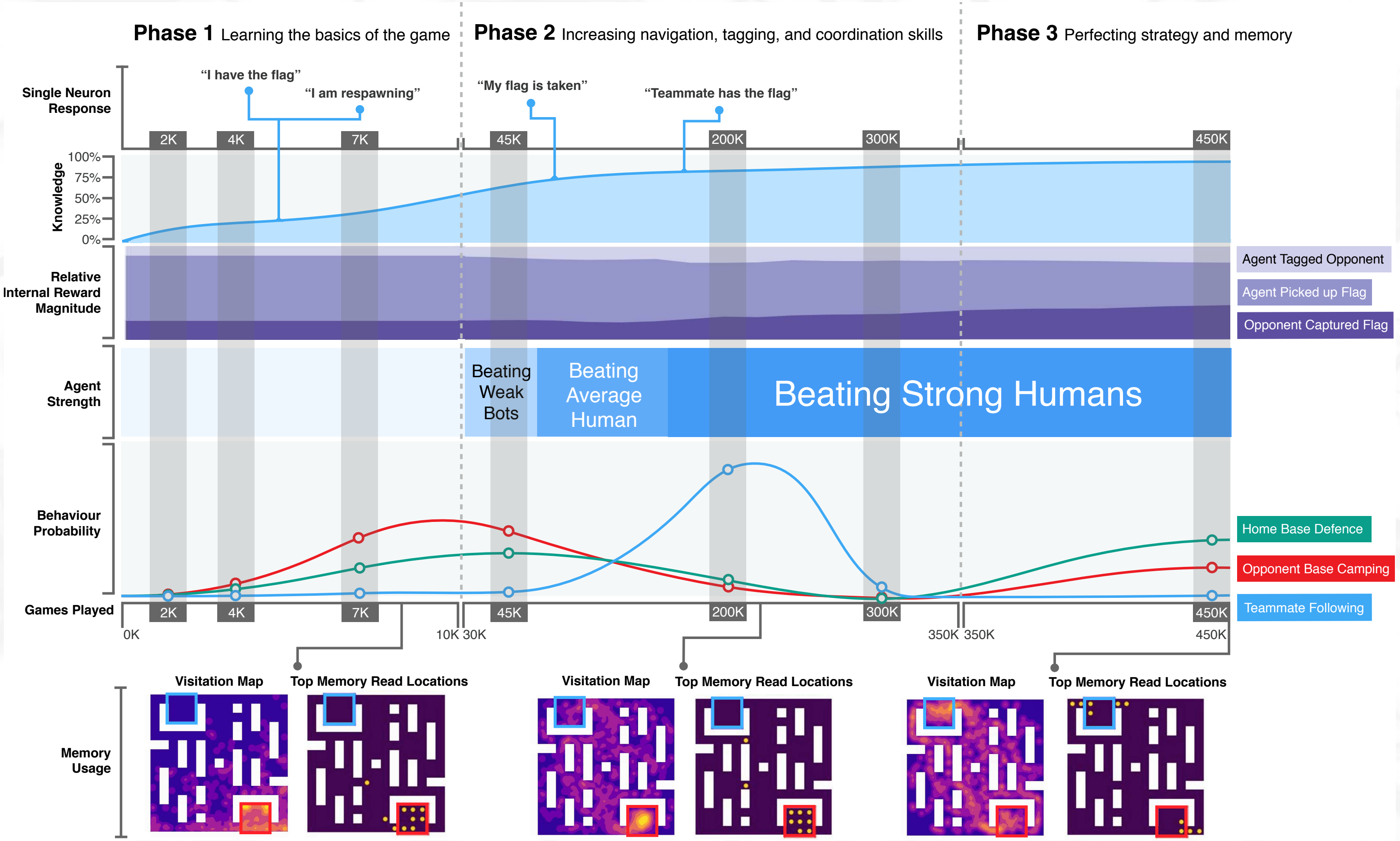


Figure 4 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.