Introduction to Machine Learning History

Electronic Brain


- Adjusted Weights
- Weights are not learned
- Learnable Weights and Threshold
- XOR Problem
- Solution to non-linearly separable problems
- Big computation, local optima and overfitting
- Limitations of learning prior knowledge
- Kernel function: Human intervention
- Hierarchical feature learning

ADALINE  XOR Problem  Multi-layered Perceptron (Backpropagation)  SVM  Deep Neural Network (Pretraining)

Golden Age  (“AI Winter”)
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


0 5 10 15 20 25
Figure 5 of paper "Learning Transferable Architectures for Scalable Image Recognition", https://arxiv.org/abs/1707.07012.
Neural Network Architecture of the '80s

- **Input layer**
  - $x_1$
  - $x_2$
  - $x_3$
  - $x_4$

- **Hidden layer**
  - $h_1$
  - $h_2$
  - $h_3$
  - $h_4$

- **Output layer**
  - $y_1$
  - $y_2$
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:

$$h = f(Wx)$$
Neural Network Activation Functions

Output Layers

- none (linear regression if there are no hidden layers)
- $\sigma$ (sigmoid; logistic regression if there are no hidden layers)

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

- softmax (maximum entropy model if there are no hidden layers)

$$\text{softmax}(\mathbf{x}) \propto e^\mathbf{x}$$

$$\text{softmax}(\mathbf{x})_i \overset{\text{def}}{=} \frac{e^{x_i}}{\sum_j e^{x_j}}$$
Hidden Layers

- none (does not help, composition of linear mapping is a linear mapping)
- $\sigma$ (but works badly – nonsymmetrical, $\frac{d\sigma}{dx}(0) = 1/4$)
- tanh
  - result of making $\sigma$ symmetrical and making derivation in zero 1
  - $\tanh(x) = 2\sigma(2x) - 1$
- ReLU
  - $\max(0, x)$
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 $N \in \mathbb{N}, v_i \in \mathbb{R}, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^m$, such that if we denote

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

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

$$|F(x) - f(x)| < \varepsilon.$$
Evolving ReLU Approximation
Evolving ReLU Approximation
Evolving ReLU Approximation
Evolving ReLU Approximation
Evolving ReLU Approximation

The graph illustrates the evolving ReLU approximation over a range of values. The x-axis represents the range from -1 to 1, while the y-axis shows the values from 0.1 to 0.1. The approximation curves are depicted in various colors, showing the progression and accuracy of the ReLU function as a model evolves with training data.
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 $f(x; \theta)$ using parameters $\theta$, the mean square error is computed as

$$\sum_i \left( f(x^{(i)}; \theta) - y^{(i)} \right)^2.$$
A model is usually trained in order to minimize the *loss* on the training data.

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

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

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

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

$$
\theta_{ML} = \arg \max_{\theta} p_{model}(\mathbf{X}; \theta)
= \arg \max_{\theta} \prod_{i=1}^{m} p_{model}(\mathbf{x}^{(i)}; \theta)
= \arg \min_{\theta} \sum_{i=1}^{m} - \log p_{model}(\mathbf{x}^{(i)}; \theta)
= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{data}} [- \log p_{model}(\mathbf{x}; \theta)]
= \arg \min_{\theta} H(\hat{p}_{data}, p_{model}(\mathbf{x}; \theta))
= \arg \min_{\theta} D_{KL}(\hat{p}_{data} || p_{model}(\mathbf{x}; \theta)) + H(\hat{p}_{data})
$$
Maximum Likelihood Estimation

Easily generalized to situations where our goal is predict $y$ given $x$.

$$\theta_{ML} = \arg \max_{\theta} p_{\text{model}}(Y|X; \theta)$$

$$= \arg \max_{\theta} \prod_{i=1}^{m} p_{\text{model}}(y^{(i)}|x^{(i)}; \theta)$$

$$= \arg \min_{\theta} \sum_{i=1}^{m} -\log p_{\text{model}}(y^{(i)}|x^{(i)}; \theta)$$

The resulting loss function is called negative log likelihood, or cross-entropy or Kullback-Leibler divergence.
Gradient Descent

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

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

we may use gradient descent:

\[
\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)
\]
Gradient Descent

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

Online (or Stochastic) Gradient Descent

We estimate the expectation in $J(\theta)$ 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 $J(\theta)$ 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.
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 \( g \)

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 \( \frac{\partial f(x)}{\partial x} \) is a Jacobian. However, the backpropagation algorithm is analogous.
Stochastic Gradient Descent (SGD) Algorithm

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

**Inputs:** Learning rate $\alpha$.

**Outputs:** Updated parameters $\theta$.

- Repeat until stopping criterion is met:
  - Sample a minibatch of $m$ training examples $(x^{(i)}, y^{(i)})$
  - $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
  - $\theta \leftarrow \theta - \alpha g$
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/VPmICjiKEKI/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/VPmIBxwGlsl/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
<table>
<thead>
<tr>
<th></th>
<th>Classical ('90s)</th>
<th>Deep Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>🟢 🟢 🟢 🟢</td>
<td>🟢 🟢 🟢 🟢 🟢 CNN, RNN, VAE, GAN, ...</td>
</tr>
<tr>
<td>Activation func.</td>
<td>tanh, σ</td>
<td>tanh, ReLU, PReLU, ELU, SELU, Swish, ...</td>
</tr>
<tr>
<td>Output function</td>
<td>none, σ</td>
<td>none, σ, softmax</td>
</tr>
<tr>
<td>Loss function</td>
<td>MSE</td>
<td>NLL (or cross-entropy or KL-divergence)</td>
</tr>
<tr>
<td>Optimization</td>
<td>SGD, momentum</td>
<td>SGD, RMSProp, Adam, ...</td>
</tr>
<tr>
<td>Regularization</td>
<td>L2, L1</td>
<td>L2, Dropout, BatchNorm, LayerNorm, ...</td>
</tr>
</tbody>
</table>
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.
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 $1/(1 - p)$. 
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
Convolutional Networks

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.
Similarities in V1 and CNNs

The primary visual cortex recognizes Gabor functions.

Figure 9.18, page 370 of Deep Learning Book, http://deeplearningbook.org
Similarities in V1 and CNNs

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

Figure 9.19, page 371 of Deep Learning Book, http://deeplearningbook.org
CNNs as Regularizers – Deep Prior

(a) Ground truth

(b) SRRResNet [18], Trained

(c) Bicubic, Not trained

(d) Deep prior, Not trained

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
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 Deep Neural Networks”
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
Beyond Image Classification

Object detection (including location)

Image segmentation

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

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

Object detection (including location)

Image segmentation

Human pose estimation

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

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

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

Single RNN cell
Recurrent Neural Networks

Single RNN cell

Unrolled RNN cells
Figure 1 of paper "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473.
Figure 5 of "Show and Tell: Lessons learned from the 2015 MSCOCO...", https://arxiv.org/abs/1609.06647.
Visual Question Answering

Figure 6 of “Multimodal Compact Bilinear Pooling for VQA and Visual Grounding”, https://arxiv.org/abs/1606.01847.
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.*
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.
<table>
<thead>
<tr>
<th>Method</th>
<th>SNR</th>
<th>CER</th>
<th>WER</th>
<th>BLEU†</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lips only</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional‡</td>
<td>-</td>
<td>58.7%</td>
<td>73.8%</td>
<td>23.8</td>
</tr>
<tr>
<td>WAS</td>
<td>-</td>
<td>59.9%</td>
<td>76.5%</td>
<td>35.6</td>
</tr>
<tr>
<td>WAS+CL</td>
<td>-</td>
<td>47.1%</td>
<td>61.1%</td>
<td>46.9</td>
</tr>
<tr>
<td>WAS+CL+SS</td>
<td>-</td>
<td>42.4%</td>
<td>58.1%</td>
<td>50.0</td>
</tr>
<tr>
<td>WAS+CL+SS+BS</td>
<td>-</td>
<td>39.5%</td>
<td>50.2%</td>
<td>54.9</td>
</tr>
<tr>
<td><strong>Audio only</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Google Speech API</td>
<td>clean</td>
<td>17.6%</td>
<td>22.6%</td>
<td>78.4</td>
</tr>
<tr>
<td>Kaldi SGMM+MMI*</td>
<td>clean</td>
<td>9.7%</td>
<td>16.8%</td>
<td>83.6</td>
</tr>
<tr>
<td>LAS+CL+SS+BS</td>
<td>clean</td>
<td>10.4%</td>
<td>17.7%</td>
<td>84.0</td>
</tr>
<tr>
<td>LAS+CL+SS+BS 10dB</td>
<td>10dB</td>
<td>26.2%</td>
<td>37.6%</td>
<td>66.4</td>
</tr>
<tr>
<td>LAS+CL+SS+BS 0dB</td>
<td>0dB</td>
<td>50.3%</td>
<td>62.9%</td>
<td>44.6</td>
</tr>
<tr>
<td><strong>Audio and lips</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLAS+CL+SS+BS clean</td>
<td>clean</td>
<td>7.9%</td>
<td>13.9%</td>
<td>87.4</td>
</tr>
<tr>
<td>WLAS+CL+SS+BS 10dB</td>
<td>10dB</td>
<td>17.6%</td>
<td>27.6%</td>
<td>75.3</td>
</tr>
<tr>
<td>WLAS+CL+SS+BS 0dB</td>
<td>0dB</td>
<td>29.8%</td>
<td>42.0%</td>
<td>63.1</td>
</tr>
</tbody>
</table>

## Lip Reading

<table>
<thead>
<tr>
<th>GT</th>
<th>IT WILL BE THE CONSUMERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>IN WILL BE THE CONSUMERS</td>
</tr>
<tr>
<td>L</td>
<td>IT WILL BE IN THE CONSUMERS</td>
</tr>
<tr>
<td>AV</td>
<td>IT WILL BE THE CONSUMERS</td>
</tr>
<tr>
<td>GT</td>
<td>CHILDREN IN EDINBURGH</td>
</tr>
<tr>
<td>A</td>
<td>CHILDREN AND EDINBURGH</td>
</tr>
<tr>
<td>L</td>
<td>CHILDREN AND HANDED BROKE</td>
</tr>
<tr>
<td>AV</td>
<td>CHILDREN IN EDINBURGH</td>
</tr>
<tr>
<td>GT</td>
<td>JUSTICE AND EVERYTHING ELSE</td>
</tr>
<tr>
<td>A</td>
<td>JUST GETTING EVERYTHING ELSE</td>
</tr>
<tr>
<td>L</td>
<td>CHINESES AND EVERYTHING ELSE</td>
</tr>
<tr>
<td>AV</td>
<td>JUSTICE AND EVERYTHING ELSE</td>
</tr>
</tbody>
</table>

Lip Reading

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Unseen Speakers</th>
<th>Overlapped Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CER</td>
<td>WER</td>
</tr>
<tr>
<td>Hearing-Impaired Person (avg)</td>
<td>–</td>
<td>47.7%</td>
</tr>
<tr>
<td>Baseline-LSTM</td>
<td>38.4%</td>
<td>52.8%</td>
</tr>
<tr>
<td>Baseline-2D</td>
<td>16.2%</td>
<td>26.7%</td>
</tr>
<tr>
<td>Baseline-NoLM</td>
<td>6.7%</td>
<td>13.6%</td>
</tr>
<tr>
<td>LipNet</td>
<td>6.4%</td>
<td>11.4%</td>
</tr>
</tbody>
</table>

*Figure 1 of “LipNet: End-to-end Sentence-level Lipreading”, [https://arxiv.org/abs/1611.01599](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

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

<table>
<thead>
<tr>
<th>Agent</th>
<th>no-ops</th>
<th>human starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>79%</td>
<td>68%</td>
</tr>
<tr>
<td>DDQN (*)</td>
<td>117%</td>
<td>110%</td>
</tr>
<tr>
<td>Prioritized DDQN (*)</td>
<td>140%</td>
<td>128%</td>
</tr>
<tr>
<td>Dueling DDQN (*)</td>
<td>151%</td>
<td>117%</td>
</tr>
<tr>
<td>A3C (*)</td>
<td>-</td>
<td>116%</td>
</tr>
<tr>
<td>Noisy DQN</td>
<td>118%</td>
<td>102%</td>
</tr>
<tr>
<td>Distributional DQN</td>
<td>164%</td>
<td>125%</td>
</tr>
<tr>
<td>Rainbow</td>
<td>223%</td>
<td>153%</td>
</tr>
</tbody>
</table>

Table showing performance comparison of different agents in terms of median human-normalized score.
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.
## AlphaZero – Ablations

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

<table>
<thead>
<tr>
<th>B</th>
<th>Chess</th>
<th>Shogi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/100 time</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>1/30 time</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>1/10 time</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>1/3 time</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>same time</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>Latest Stockfish</th>
<th>Aperyphapaq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opening Book</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>Human openings</th>
<th>TCEC openings</th>
</tr>
</thead>
</table>

- **AlphaZero wins**: Green
- **AlphaZero draws**: Grey
- **AlphaZero loses**: Pink
- **AlphaZero white**: ○
- **AlphaZero black**: ●

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

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

(a) FTW Agent Architecture

Game points $p_t$

Winning signal

Internal reward

Action

Policy

Sampled latent variable

Slow RNN

Fast RNN

Observation $x_t$

(b) Progression During Training

Agent Elo

FTW

Strong Human

Self-play + RS

Average Human

Self-play

Random agent

Learning Rate

KL Weighting

Internal Timescale

Figure 2 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.
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
For the Win agent for Capture The Flag

Phase 1 Learning the basics of the game
- Single Neuron Response
- Knowledge
- Relative Internal Reward Magnitude
- Agent Strength
- Behaviour Probability
- Games Played

Phase 2 Increasing navigation, tagging, and coordination skills
- Visitation Map
- Top Memory Read Locations

Phase 3 Perfecting strategy and memory
- Visitation Map
- Top Memory Read Locations

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