IE 534/CS 598 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2018

Lecture 12

Topics we will cover this week:

- Actor-critic methods.
- Deep exploration.
- AlphaGo algorithm.

- How to efficiently explore?
- Exploration versus greedy strategy.
- Bayesian approach can be efficient in simple cases such as multi-armed bandits (commonly referred to as “Thompson sampling”).

- Let $Q(x, a; \theta)$ be the approximation for the value function.
- Suppose we have a prior $p(\theta)$.
- We observe a sequence $(A_t, X_t, R_t)_t$.
- Bayesian posterior $p_t(\theta)$ given the data $(A_\tau, X_\tau, R_\tau)_{\tau=1:t}$.

Thompson algorithm:

- Sample θ_t from $p_{t-1}(\theta)$.
- Take action

$$A_t = \arg\max_{a \in \mathcal{A}} Q(X_t, a; \theta_t). \tag{1}$$

- Observe reward R_t.
- Update posterior distribution to produce $p_t(\theta)$.

Given the prior distribution $p_{t-1}(\theta)$ and the observations $H_t = (A_\tau, X_\tau, R_\tau)_{\tau=1:t}$, use Bayes' theorem to compute the posterior distribution

$$p_t(\theta) = \frac{\mathbb{P}[H_t|\theta]p_{t-1}(\theta)}{\int_\Theta \mathbb{P}[H_t|\theta']p_{t-1}(\theta')d\theta'}, \tag{2}$$

where $\theta \in \Theta$.

Given the prior distribution $p_{t-1}(\theta)$ and the observations $H_t = (A_\tau, X_\tau, R_\tau)_{\tau=1:t}$, use Bayes' theorem to compute the posterior distribution

$$p_t(\theta) = \frac{\mathbb{P}[H_t|\theta]p_{t-1}(\theta)}{\int_\Theta \mathbb{P}[H_t|\theta']p_{t-1}(\theta')d\theta'}, \tag{3}$$

where $\theta \in \Theta$.

- Note that θ is high-dimensional, so we have to compute a high-dimensional integral.
- $\mathbb{P}[H_t|\theta]$ may also be challenging to compute.
- Only feasible in very simple settings (e.g., linear models, multi-armed bandits).

Example.

- Suppose we have observations $y_i \sim \mathcal{N}(\mu^*, \sigma^2)$.
- The prior is $p(\mu^*) = \mathcal{N}(\mu_0, \sigma_0^2)$.
- The posterior after N observations is

$$p(\mu^*|y_1, \ldots, y_N) = \mathcal{N}\left(\mu_N = \frac{\frac{\mu_0}{\sigma_0^2} + \sum_{i=1}^{N} \frac{y_i}{\sigma^2}}{\frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}}, \sigma_N = \frac{1}{\frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}}\right). \quad (4)$$

- In general, Bayesian estimation for neural networks is computationally infeasible.
- Various approximations have been proposed for the Bayesian posterior update, but don't necessarily perform better than SGD.
- There is also a claim that dropout produces a Bayesian posterior distribution (which is incorrect!). See "Risk versus Uncertainty in Deep Learning: Bayes, Bootstrap and the Dangers of Dropout" by Osband.

Thompson algorithm:

- Very good at exploration.
- Computationally efficient and has optimal mathematical properties in simple settings.
- Computationally impractical for more complex settings (such as RL) and models (such as deep RL).

- Develop a "bootstrap deep Q-learning network" which is similar in spirit to Thompson sampling.
- Ensemble of neural networks $\theta^1, \ldots, \theta^K$.
- Each network has a different random initialization.
- Each model corresponds to a different value function $Q(x, a; \theta^k)$.
- Therefore, a variety of strategies will be explored.

Bootstrap deep Q-learning network:

- Select k at random from $\{1, 2, \ldots, K\}$.
- Take action

$$A_t = \arg\max_{a \in \mathcal{A}} Q(X_t, a; \theta_t^k). \tag{5}$$

- Observe reward R_t and X_{t+1}.
- Sample bootstrap mask m_t (a binary mask of length K)
- Save $(X_t, A_t, R_t, X_{t+1}, m_t)$ in replay buffer.
- Update models $\theta_t^1, \ldots, \theta_t^K$.

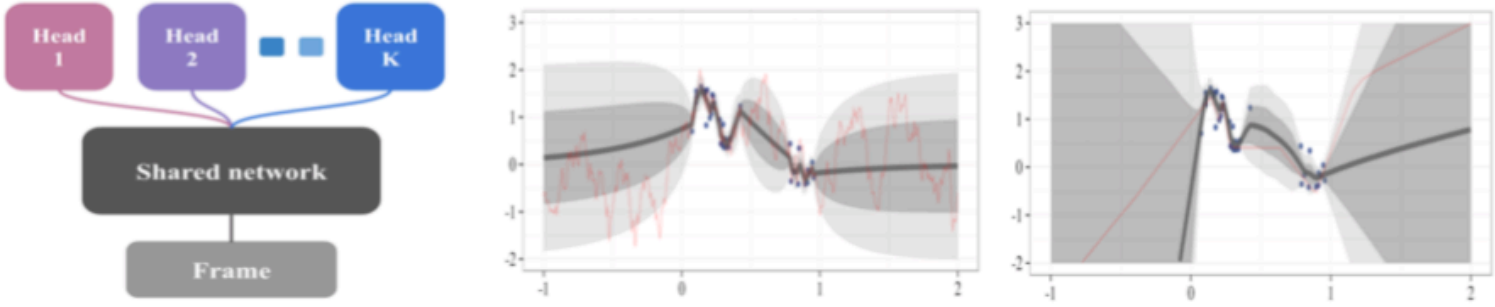
Suppose we sample $(X_\tau, A_\tau, R_\tau, X_{\tau+1}, m_\tau)$ from the replay buffer. The update to the k-th model is

$$\begin{aligned} \theta_t^k &= \theta_t^k + g_\tau^k, \\ g_\tau^k &= m_\tau^k \Big(y_\tau - Q(X_\tau, A_\tau; \theta_t^k) \Big) \nabla_\theta Q(X_\tau, A_\tau; \theta_t^k), \\ y_\tau &= R_\tau + \gamma \max_{a \in \mathcal{A}} Q(X_{\tau+1}, a; \theta_t^k). \end{aligned} \tag{6}$$

Of course, in practice we sample a mini-batch from the replay buffer.

Some important implementation details:

- Replay buffer
- Shared network between the different models $1, 2, \ldots, M$.
- Gradient normalization
- Replace objective function (6) with Double Q-learning.
- See Appendix of NIPS paper.



(a) Shared network architecture (b) Gaussian process posterior (c) Bootstrapped neural nets

Figure 1: Bootstrapped neural nets can produce reasonable posterior estimates for regression.

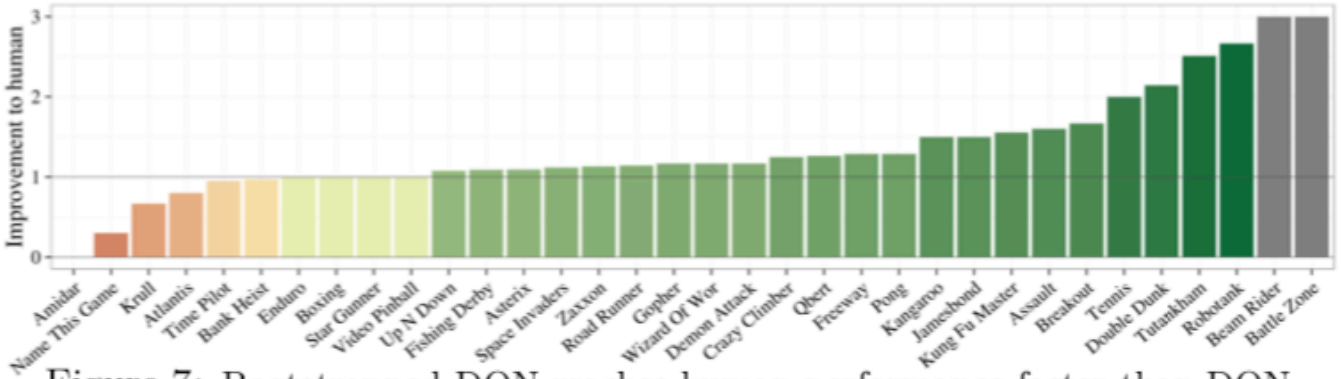


Figure 7: Bootstrapped DQN reaches human performance faster than DQN.

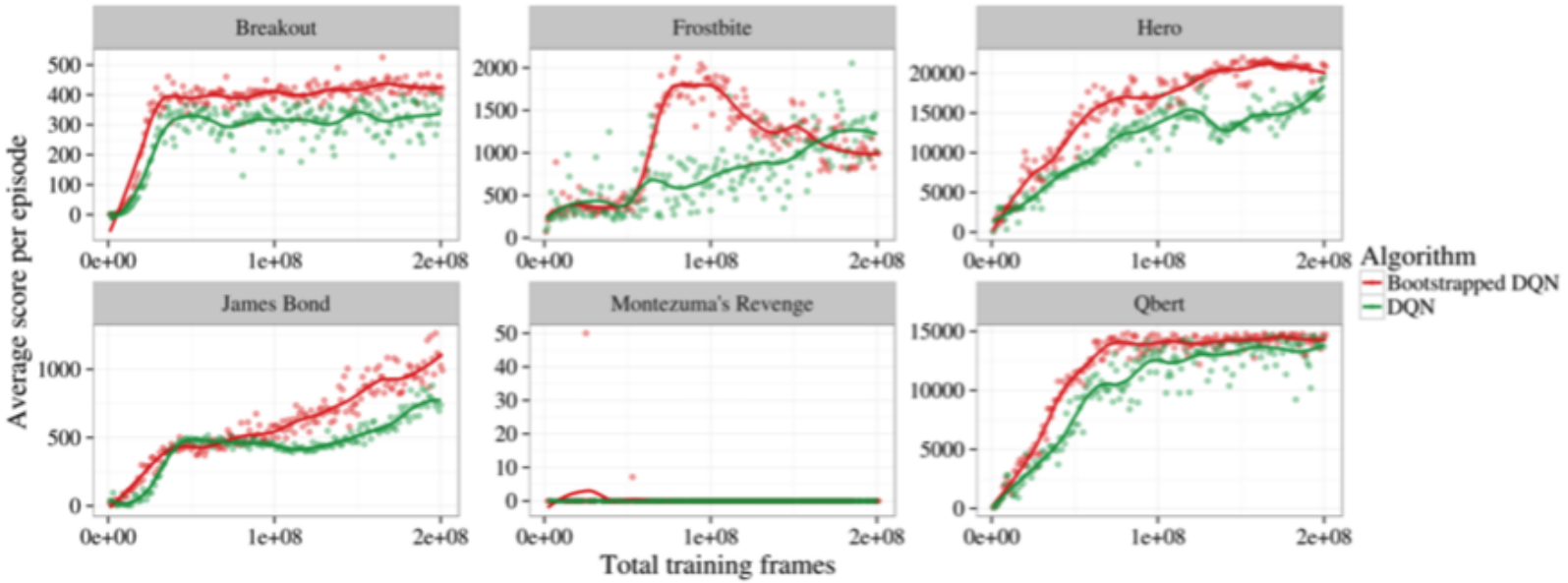


Figure 6: Bootstrapped DQN drives more efficient exploration.

Source: "Deep Exploration via Bootstrapped DQN" by Osband et al. (2015).

Interchange derivative and expectation.

Recall that the SGD step $\nabla_\theta \rho(f(X;\theta), Y)$ is an unbiased estimate of the gradient descent direction:

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}\Big[\rho\big(f(X;\theta), Y\big)\Big], \\ \mathbb{E}\Big[\nabla_\theta \rho\big(f(X;\theta), Y\big)\Big] &= \nabla_\theta \mathbb{E}\Big[\rho\big(f(X;\theta), Y\big)\Big] = \nabla_\theta \mathcal{L}(\theta). \quad (7) \end{aligned}$$

Technical requirements allowing us to interchange derivative and expectation:

- $\mathcal{L}(\theta) < \infty$ for each θ.
- $\nabla_\theta \rho\big(f(x;\theta), y\big)$ exists for every (x, y, θ).
- There is an integrable function $g(x, y)$ such that $\big|\nabla_\theta \rho\big(f(x;\theta), y\big)\big| \leq g(x, y)$ for every (x, y, θ).

Dropout as a Bayesian approximation?

- Train a neural network $f(x, R; \theta)$ with dropout algorithm.
- The distribution of $f(x, R; \theta)$ is claimed to be approximately the Bayesian posterior for θ.
- "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning" (500 citations)
- "Risk versus Uncertainty in Deep Learning: Bayes, Bootstrap and the Dangers of Dropout" (Osband, 2016).

Example.

- Suppose we have observations $y_i \sim \mathcal{N}(\mu^*, \sigma^2)$.
- The prior is $p(\mu^*) = \mathcal{N}(\mu_0, \sigma_0^2)$.
- The posterior after N observations is

$$p(\mu^*|y_1, \ldots, y_N) = \mathcal{N}\left(\mu_N = \frac{\frac{\mu_0}{\sigma_0^2} + \sum_{i=1}^{N} \frac{y_i}{\sigma^2}}{\frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}}, \sigma_N = \frac{1}{\frac{1}{\sigma_0^2} + \frac{N}{\sigma^2}}\right). \quad (8)$$

Note that $p(\mu^*|y_1, \ldots, y_N) \to \delta(\mu - \mu^*)$ as $N \to \infty$. That is, $\sigma_N \to 0$ and $\mu_N \to \mu^*$.

- Let θ^N be the estimated parameters after observing N data samples.
- Note that $\mathbb{P}[f(x, R; \theta^N) \in dy]$ does not converge to a delta function as $N \to \infty$.
- That is, the "dropout posterior" does not "concentrate as more data becomes available.
- Dropout is not a Bayesian posterior..

Example.

- Let $f(\theta, R) = R\theta$ with dropout $R \sim \text{Bernoulli}(p)$.
- The SGD algorithm is

$$\theta = \theta + \alpha\big(Y - f(\theta, R)\big)R. \tag{9}$$

- The algorithm will converge to a θ such that

$$\mathbb{E}\Big[\big(Y - f(\theta, R)\big)R\Big] = 0. \tag{10}$$

- This corresponds to the (mean-squared error) objective function

$$\mathbb{E}\Big[\big(Y - f(\theta, R)\big)^2\Big]. \tag{11}$$

Suppose we have N observations, the objective function is then

$$\mathcal{L}^N(\theta) = \sum_{i=1}^{N} \mathbb{E}_R\Big[\big(y_i - f(\theta, R)\big)^2\Big]. \tag{12}$$

The global minimum satisfies

$$\sum_{i=1}^{N} \mathbb{E}_R\Big[\big(y_i - \theta R\big)R\Big] = 0. \tag{13}$$

Rearranging,

$$\begin{aligned} \sum_{i=1}^{N} y_i p &= N\theta \mathbb{E}_R\big[R^2\big] \\ &= N\theta p. \end{aligned} \tag{14}$$

Therefore,

$$\theta = \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{15}$$

Of course,

$$f(\theta, R) = \frac{R}{N} \sum_{i=1}^{N} y_i. \tag{16}$$

Let $\mu = \mathbb{E}[Y]$. Then, as $N \to \infty$, we have that

$$f(\theta, R) = \mu R. \tag{17}$$

The distribution of $f(\theta, R)$ does not concentrate..

$$\lim_{N \to \infty} \mathbb{E}\left[f(\theta^N, R) \right] = \mu p, \tag{18}$$

and

$$\lim_{N \to \infty} \mathrm{Var}\left[f(\theta^N, R) \right] = \mu^2 p(1-p). \tag{19}$$

The distribution of $f(\theta, R)$ does not concentrate..

$$\lim_{N\to\infty} \mathbb{E}\Big[f(\theta^N, R)\Big] = \mu p, \tag{20}$$

and

$$\lim_{N\to\infty} \mathrm{Var}\Big[f(\theta^N, R)\Big] = \mu^2 p(1-p). \tag{21}$$

The variance also does not depend upon N! (We should become more confident as the number of data samples increases..)

Note that the ensemble estimate $\mathbb{E}\big[f(\theta, R)\big] = \mu p$ is biased. Recall that

$$\begin{aligned} \text{Bias} &= \Big|\mathbb{E}\big[f(\theta, R)\big] - \mathbb{E}\big[Y\big]\Big| \\ &= \Big|\mathbb{E}\big[f(\theta, R)\big] - \mu\Big|. \end{aligned} \tag{22}$$

Note that the ensemble estimate $\mathbb{E}\big[f(\theta, R)\big] = \mu p$ is biased.

- Let $f(\theta, R) = \sum_{i=1}^{K} R_i \theta_i$ where R_i are independent $\mathrm{Bernoulli}(p)$ RVs.
- Then, the global minimizer after observing N data samples satisfies

$$\sum_{i=1}^{N} \mathbb{E}_R \Big[\big(y_i - \theta \cdot R\big) R \Big] = 0. \tag{23}$$

This yields the system of equations

$$\frac{1}{N} \sum_{i=1}^{N} y_i p = \sum_{i=1}^{K} \theta_i \big(\mathbf{1}_{i=j} p + \mathbf{1}_{i \neq j} p^2 \big), \tag{24}$$

for $j = 1, \ldots, K$.

This produces the system of equations

$$\frac{1}{N}\sum_{i=1}^{N} y_i p = \theta_j p + \sum_{i \neq j} \theta_i \mathbf{1}_{i \neq j} p^2. \tag{25}$$

for $j = 1, \ldots, K$. Note that this equation is symmetric with respect to θ_i, therefore $\theta_i = \theta$.

$$\frac{1}{N}\sum_{i=1}^{N} y_i p = \theta p + (K-1)\theta p^2. \tag{26}$$

Therefore, the global minimum is

$$\theta = \frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{1 + (K-1)p}. \tag{27}$$

$$\theta^N = \frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{1+(K-1)p}. \tag{28}$$

The ensemble estimate is

$$\begin{aligned}
\mathbb{E}\Big[f(\theta^N, R)\Big] &= \mathbb{E}\Big[\sum_{k=1}^{K} \frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{1+(K-1)p} R_k\Big] \\
&= Kp\frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{1+(K-1)p}.
\end{aligned} \tag{29}$$

As K becomes large, $\mathbb{E}\Big[f(\theta^N, R)\Big] \to \frac{1}{N}\sum_{i=1}^{N} y_i$.

Similarly,

$$
\begin{aligned}
\mathrm{Var}\left[f(\theta^N, R)\right] &= \mathrm{Var}\left[\sum_{k=1}^{K} \frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{1+(K-1)p} R_k\right] \\
&= \left(\frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{1+(K-1)p}\right)^2 Kp(1-p) \\
&= \left(\frac{\frac{1}{N}\sum_{i=1}^{N} y_i}{\frac{1}{K}+\frac{K-1}{K}p}\right)^2 \frac{p(1-p)}{K} \\
&= \left(\frac{1}{N}\sum_{i=1}^{N} y_i\right)^2 \frac{1-p}{Kp} \qquad (30)
\end{aligned}
$$

Confidence Intervals.

- Suppose that $Y = f(X; \theta^*) + \epsilon$.
- We estimate the model $f(x; \theta)$ from the dataset $\mathcal{D} = (x^i, y^i)_{i=1}^N$.
- Let's call our estimate $\theta(\mathcal{D})$.
- A confidence interval is an interval $\mathcal{I} = [L(\mathcal{D}), U(\mathcal{D})]$ such that

$$\mathbb{P}[\theta^* \in \mathcal{I}] = \lambda. \tag{31}$$

Can we use bootstrap methods to estimate the confidence interval? Why or why not?

Theoretical properties of neural networks:

- Universal approximation theorem (consistency)
- Central limit theorem for global minima (can be used to estimate confidence intervals). Why does this not work in practice?
- Central limit theorem for neural networks trained with SGD (see Sirignano and Spiliopoulos 2018). This does provide an estimate on the "training uncertainty" of neural networks, but not confidence intervals in the traditional sense.

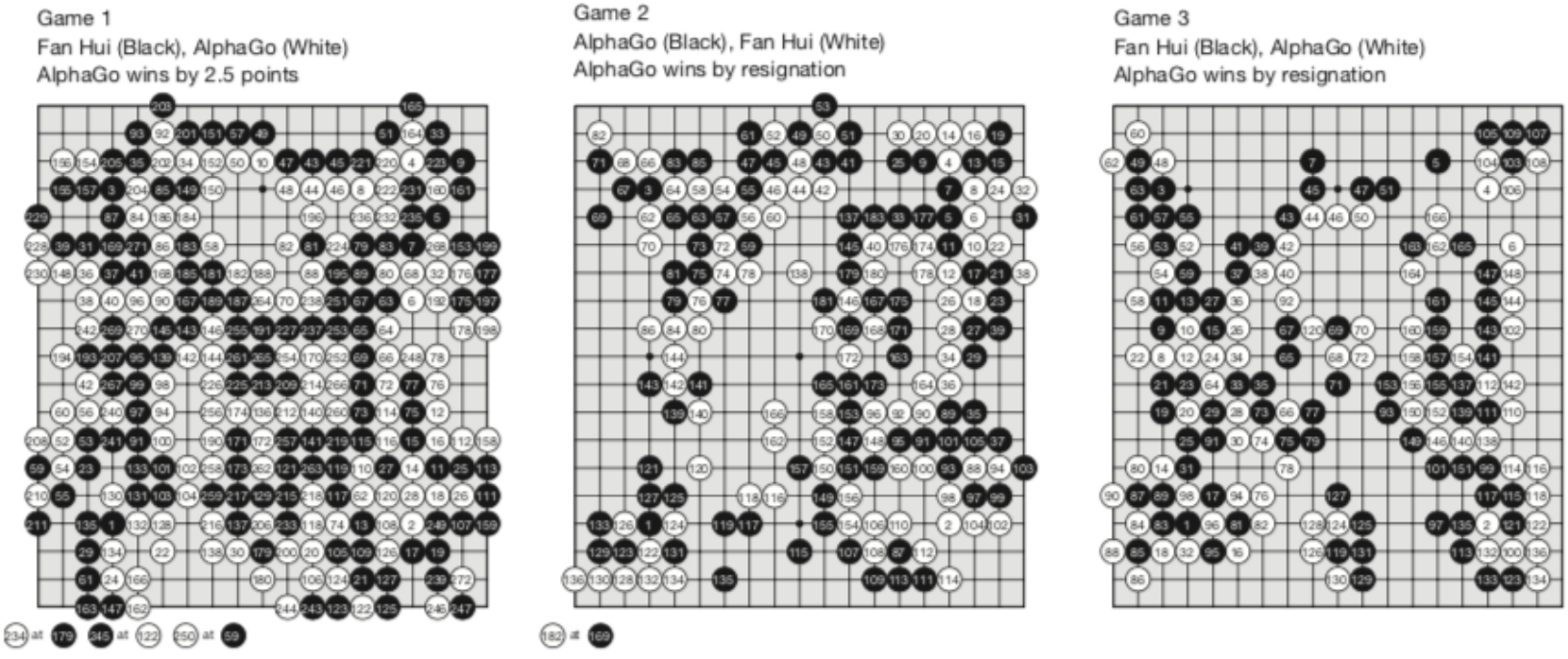
For single-layer neural networks trained with SGD:

- The empirical distribution of the neural network will be approximately Gaussian for large d_H.
- The prediction, conditional on the input x, will be approximately Gaussian for large d_H.

Uncertainty quantification.

- Traditional statistical methods do not necessarily hold.
- Best approach: estimate error distribution on test set.

Alpha Go.



Source: "Mastering the game of Go with deep neural networks and tree search" by Silver et al.

Alpha Go.

- Train a network $p_\sigma(a|s)$ on 30 million expert moves.

$$\rho_{n+1} = \rho_n + \alpha_n \nabla_\rho \log p_{\rho_n}(a_t|s_t). \tag{32}$$

- Supervised learning.
- 57% accuracy once trained.
- SL (supervised learning) policy network: 13 layers, convolution layers, fully connected ReLU layers, and softmax layer.

- Fast rollout policy $p_\pi(a|s)$.
- Logistic regression with nonlinear, custom-designed features.
- 24.2 % accuracy
- Evaluation time for SL network: 23 milliseconds.
- Evaluation time for Fast rollout network: 2 microseconds.

Alpha Go.

- Initialize $\rho_0 = \sigma$.
- Train $p_\rho(a|s)$ using policy gradient descent.
- Let R_t be the terminal reward at the end of the game (± 1)

$$\rho_{n+1} = \rho_n + \alpha_n \nabla_\rho \log p_{\rho_n}(a_t|s_t) R_t. \tag{33}$$

- p_{ρ_n} is trained against a previous iteration of the model, i.e. p_{ρ_k} for $k < n$.
- RL policy network wins 85% of games against best open-source Go program!
- Previous state-of-the-art (a supervised convolution network) only won 11% of its games.

- Estimate value $Q_\theta(s)$ under policy $p_\rho(a|s)$ by minimizing the objective function

$$\sum_{t,g} \Big(R_{t,g} - Q_\theta(S_{t,g})) \Big)^2. \tag{34}$$

- That is, take SGD steps

$$\theta_{n+1} = \theta_n - \alpha \Big(R_{t,g} - Q_\theta(S_{t,g}) \Big) \nabla_\theta Q_\theta(S_{t,g}). \tag{35}$$

- Consecutive positions are very correlated.
- Sample a single position for each of 30 million games (played by p_ρ against itself).
- Monte Carlo tree search (see paper).

Numerical Implementation of convolutions.

- Lower convolutions into a matrix multiplication
- "Unroll" filter and expand input data.
- Large matrix multiplication
- Reshape output

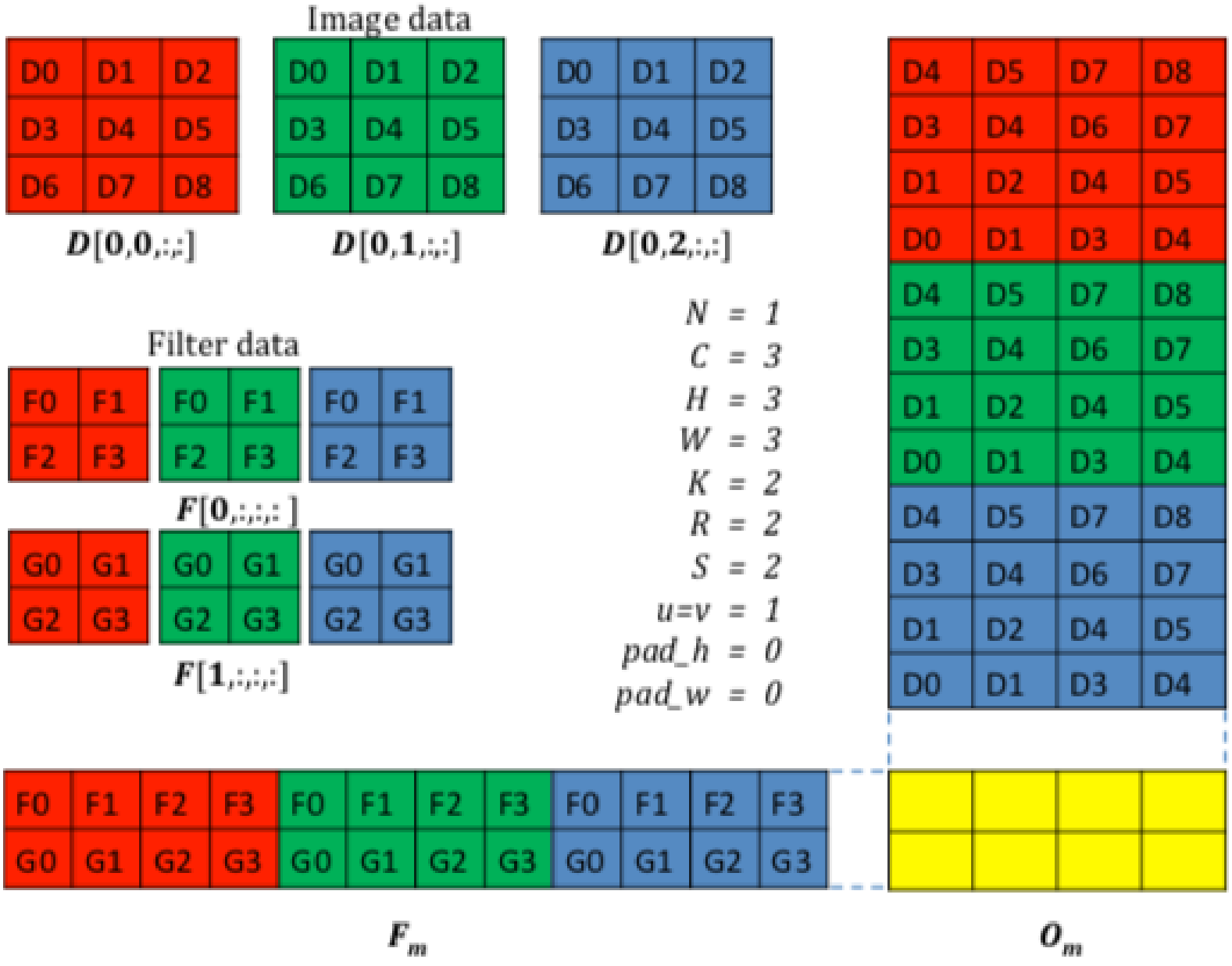
Parameter	Meaning
N	Number of images in mini-batch
C	Number of input feature maps
H	Height of input image
W	Width of input image
K	Number of output feature maps
R	Height of filter kernel
S	Width of filter kernel
u	Vertical stride
v	Horizontal stride
pad_h	Height of zero-padding
pad_w	Width of zero-padding

Table 1: Convolutional parameters

Source: "cuDNN: Efficient Primitives for Deep Learning" by Chetlur et al.

Reshape filter and input data:

- Filter tensor F has dimensions $K \times C \times R \times S$.
- Reshape as matrix F_m with dimensions $K \times CRS$.
- Input data D has dimensions $N \times C \times H \times W$.
- Reshape as matrix D_m with dimensions $CRS \times NPQ$.
- Let $O_m = F_m \times D_m$, then reshape the matrix O_m to produce the output O with dimensions $N \times K \times P \times Q$.



Source: "cuDNN: Efficient Primitives for Deep Learning" by Chetlur et al.

Advantages and disadvantages:

- Matrix multiplication is highly optimized.
- Efficiency increases as the size of the matrix increases.
- Constructing D_m requires repeating elements of D, so there is extra memory cost.

Fast Fourier Transform (FFT):

- Filter must be padded to equal the size of the input data.
- Computationally expensive to compute for strided convolutions.
- FFT is useful when filter size is large relative to the input data size.
- However, in deep learning, typically the filter size is small relative to the input data size.

Applications outside of finance:

- Medicine
- Engineering
- Finance