Data-driven and Learning-based Control

Deep Reinforcement Learning

Erica Salvato

Table of Contents

What we know so far?

1 A brief recap

We introduced Value-function methods, where the policy π is implicitly defined via $V(x^{(k)})$ or $Q(x^{(k)}, u^{(k)})$, and we assume to work with a discrete compact action set $\mathcal{U}$.

We can classify Value-function methods:

- Depending on the evaluation procedure in:
  - On-policy algorithms
    - SARSA
  - Off-policy algorithms
    - Q-learning
- Depending on the state space representation
  - Tabular
    - SARSA
    - Q-learning
  - Linear function approximation
    - SARSA
    - Q-learning

What we know so far?

In particular, we observe that in linear function approximation we assume that action-value-function is a weighted combination of a set of BFs, with weights θ, where each BF is a function of the state and the control input $\phi(x, u)$.

$$\phi(x, u)^\top \theta$$

It often works well given the right set of BFs.

How to choose BFs?

- experience and carefully hand-designing

What we know so far?

1 A brief recap

In particular, we observe that in linear function approximation we assume that action-value-function is a weighted combination of a set of BFs, with weights θ, where each BF is a function of the state and the control input $\phi(x, u)$.

$$\phi(x, u)^\top \theta$$

It often works well given the right set of BFs.

How to choose BFs?

- experience and carefully hand-designing

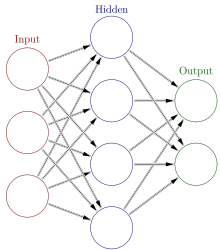
An alternative is to use a much richer function approximation class that is able to directly go from states without requiring an explicit specification of features

Table of Contents

Neural network

2 Neural networks

A **neural network** is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form.

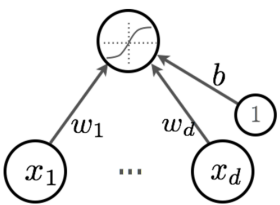


The concept of the artificial neural network (ANN) was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses.

Artificial Neuron

Therefore, the main element of an ANN is the **artificial neuron** which is defined by:

- $x \in \mathcal{X} \subseteq \mathbb{R}^d$ the input vector of a neuron
- $y \in \mathcal{Y} \subseteq \mathbb{R}$ the output vector of a neuron
- $w \in \mathcal{W} \subseteq \mathbb{R}^d$ the weights vector of a neuron
- $b \in \mathbb{R}$ the bias of a neuron
- $a : \mathcal{W} \times \mathbb{R} \times \mathcal{X} \to \mathcal{Y}$ output activation function



$$y = a\left(b + w^\top x\right)$$

Single Hidden Layer Neural Network

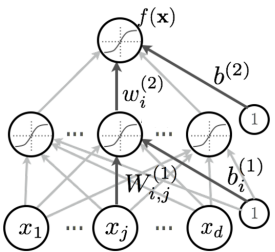
Another important element of ANNs is the hidden layer: the intermediary stage between input and output in a neural network.

- Hidden layer activation:

$$a\left(b^{(1)} + {W^{(1)}}^{\top} x\right)$$

- Output layer activation:

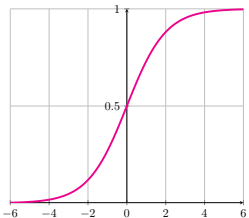
$$f(x) = o\left(b^{(2)} + {W^{(2)}}^{\top} a\left(b^{(1)} + {W^{(1)}}^{\top} x\right)\right)$$



Activation Function

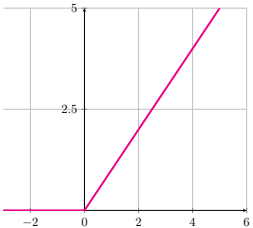
2 Neural networks

- **Sigmoid activation function**
  - Squashes the neuron's output between 0 and 1
  - Always positive
  - Bounded
  - Strictly Increasing



Activation Function

- **Rectified linear (ReLU) activation function:**
  - Bounded below by 0 (always non-negative)
  - Tends to produce units with sparse activities
  - Not upper bounded
  - Strictly Increasing



Multilayer Neural Network

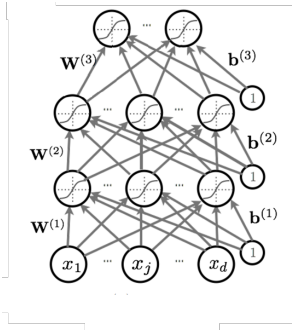
It is an ANN with L hidden layer

- Hidden layer activation:

$$a\left(b^{(1)} + W^{(1)^\top} x\right)$$

- Output layer activation:

$$f(x) = o\left(b^{(L+1)} + W^{(L+1)^\top} a\left(b^{(L)} + W^{(L)^\top} x\right)\right)$$



Multilayer Neural Network

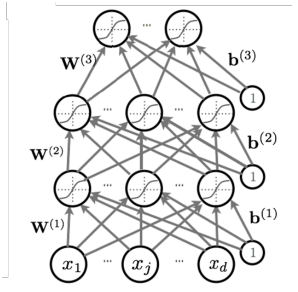
It is an ANN with L hidden layer

- Hidden layer activation:

$$a\left(b^{(1)} + W^{(1)^\top} x\right)$$

- Output layer activation:

$$f(x) = o\left(b^{(L+1)} + W^{(L+1)^\top} a\left(b^{(L)} + W^{(L)^\top} x\right)\right)$$



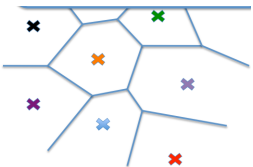
An ANN with many hidden layers is said **Deep Neural Network**

Power of ANNs

- ANNs provide distributed representations

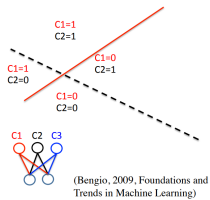
Classical function approximation:

- Parameters for each region, e.g. $c_i \in \mathbb{R}^d$, σ_i
- # of regions is linear with # of parameters $(N+1)^d$



ANNs:

- Each parameter affects many regions, not just local
- # of regions grows (roughly) exponentially in # of parameters.



(Bengio, 2009, Foundations and Trends in Machine Learning)

Power of ANNs

- ANNs are universal approximators
  - Universal Approximation Theorem (Hornik, 1991):

    a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units

  - This applies for many activation functions

Power of ANNs

- ANNs are universal approximators
  - Universal Approximation Theorem (Hornik, 1991):

    a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units

  - This applies for many activation functions

However, this does not mean that there is a learning algorithm that can find the necessary parameter values.

Training of ANNs

The **Empirical Risk Minimization** (ERM) principle is a learning paradigm that consists of selecting the model with minimal average error over the training set.

This so-called training error can be seen as an estimate of the risk.

$$\arg\min_{\theta} \frac{1}{T} \sum_{k} l\left(f\left(x^{(k)}, \theta\right), x^{(k)}\right)$$

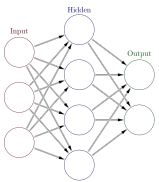
where:

- $l\left(f\left(x^{(k)}, \theta\right), x^{(k)}\right)$ is the loss function
- $\theta = \left[b^{(1)},\ W^{(1)},\ b^{(2)},\ W^{(2)},\ \ldots,\ b^{(L+1)},\ W^{(L+1)},\ \right]$ is the parameters that we have to learn.

Training of ANNs

Then given a training data-set composed of (x, y) pairs, for each pair we need to perform:

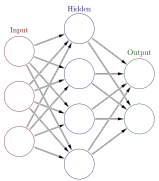
1. **Forward propagation:** it refers to the calculation and storage of intermediate variables for a neural network in order from the input layer to the output layer



Training of ANNs

Then given a training data-set composed of (x, y) pairs, for each pair we need to perform:

1. **Forward propagation:** it refers to the calculation and storage of intermediate variables for a neural network in order from the input layer to the output layer
2. **Backpropagation:** it refers to the method of calculating the gradient of neural network parameters. In short, the method traverses the network in reverse order, from the output to the input layer



ANNs in Reinforcement Learning

2 Neural networks

In Value-function RL ANNs are used to approximate the Q-function for the state-action pairs $Q(x, u)$.

The parameters vector θ is learned in order to minimize the loss-function defined as:

$$r^{(k+1)} + \gamma \max Q\left(x^{(k+1)}, u\right) - Q\left(x^{(k)}, u^{(k)}\right)$$

ANNs in Reinforcement Learning

2 Neural networks

In Value-function RL ANNs are used to approximate the Q-function for the state-action pairs $Q(x, u)$.

The parameters vector θ is learned in order to minimize the loss-function defined as:

$$r^{(k+1)} + \gamma \max Q\left(x^{(k+1)}, u\right) - Q\left(x^{(k)}, u^{(k)}\right)$$

temporal difference error

ANNs in Reinforcement Learning

However, when we tackle RL problems with function approximation we need to consider that:

- Data is sequential
  - Successive samples are correlated, non-iid (independent and identically distributed)
- Policy changes rapidly with slight changes to Q-values
  - Policy may oscillate
  - Distribution of data can swing from one extreme to another

ANNs in Reinforcement Learning

However, when we tackle RL problems with function approximation we need to consider that:

- Data is sequential
  - Successive samples are correlated, non-iid (independent and identically distributed)

↓

Correlations between samples

- Policy changes rapidly with slight changes to Q-values
  - Policy may oscillate
  - Distribution of data can swing from one extreme to another

ANNs in Reinforcement Learning

However, when we tackle RL problems with function approximation we need to consider that:

- Data is sequential
  - Successive samples are correlated, non-iid (independent and identically distributed)

↓

Correlations between samples

- Policy changes rapidly with slight changes to Q-values
  - Policy may oscillate
  - Distribution of data can swing from one extreme to another

↓

Non-stationary targets

Deep Neural Networks in Reinforcement Learning

2 Neural networks

Deep Q-learning (DQN) is a Value-function approximation algorithm based on Deep Neural Networks that addresses both

- Correlations between samples
- Non-stationary targets

by introducing the concepts of:

- **Experience replay**
- **Fixed Q-targets**

Experience Replay

2 Neural networks

To help remove correlations, the idea is to store a dataset, also called **replay buffer** D from prior experience:

$x^{(0)}, u^{(0)}, x^{(1)}, r^{(1)}$
$x^{(1)}, u^{(1)}, x^{(2)}, r^{(2)}$
$\vdots$
$x^{(k)}, u^{(k)}, x^{(k+1)}, r^{(k+1)}$

Experience replay, works by iteratively repeating the following:

- Sample an experience tuple from the dataset $\left(x^{(i)}, u^{(i)}, x^{(i+1)}, r^{(i+1)}\right)$
- Compute the target value for the sampled $x^{(i)}$: $r^{(i+1)} + \gamma \max_u \hat{Q}_\theta \left(x^{(i+1)}, u\right)$
- Use stochastic gradient descent to update the network parameters

$$\theta = \theta + \alpha \left[r^{(i+1)} + \gamma \max_u \hat{Q}_\theta \left(x^{(i+1)}, u\right) - \hat{Q}_\theta \left(x^{(i)}, u^{(i)}\right)\right] \nabla_\theta \hat{Q}_\theta \left(x^{(i)}, u^{(i)}\right)$$

Fixed Q-targets

To help improve stability, fix the target parameters used in the target calculation for multiple updates.
Let parameters θ^- be the set of parameters used in the target, and θ be the parameters that are being updated Fixed Q-targets with Experience replay works by iteratively repeating the following:

- Sample an experience tuple from the dataset $\left(x^{(i)}, u^{(i)}, x^{(i+1)}, r^{(i+1)}\right)$
- Compute the target value for the sampled $x^{(i)}$: $r^{(i+1)} + \gamma \max_u \hat{Q}_{\theta^-}\left(x^{(i+1)}, u\right)$
- Use stochastic gradient descent to update the network parameters

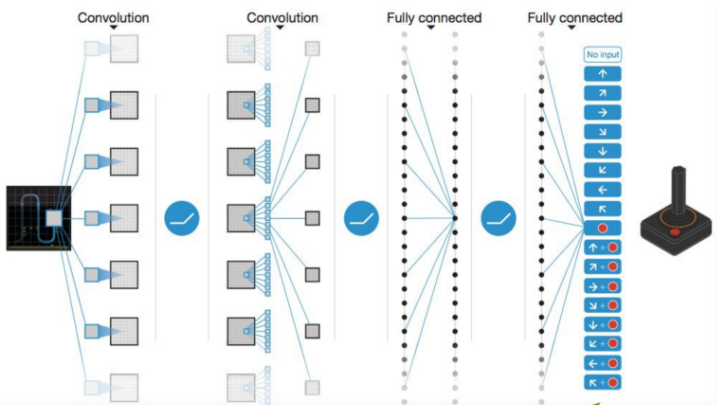
$$\theta = \theta + \alpha \left[r^{(i+1)} + \gamma \max_u \hat{Q}_{\theta^-}\left(x^{(i+1)}, u\right) - \hat{Q}_\theta\left(x^{(i)}, u^{(i)}\right)\right] \nabla_\theta \hat{Q}_\theta\left(x^{(i)}, u^{(i)}\right)$$

Deep Q-learning in summary

- DQN uses experience replay and fixed Q-targets
- Store transition $\left(x^{(k)}, u^{(k)}, x^{(k+1)}, r^{(k+1)}\right)$ in replay memory D
- Sample random mini-batch of transitions $\left(x^{(k)}, u^{(k)}, x^{(k+1)}, r^{(k+1)}\right)$ from D
- Compute Q-learning targets w.r.t. old, fixed parameters θ^{-}
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

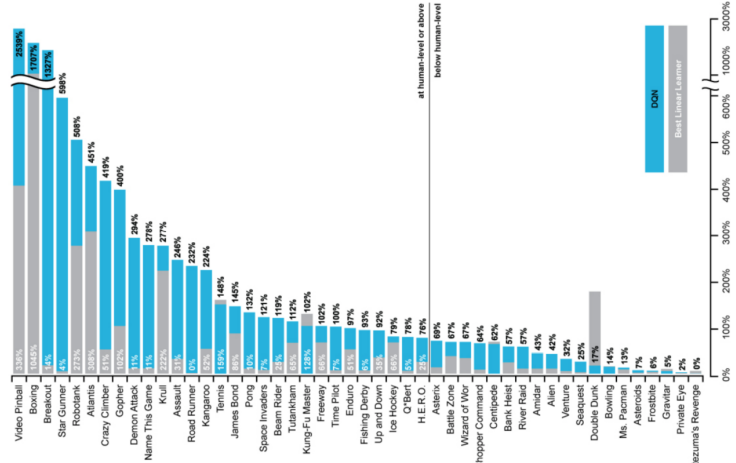
Success of DQN in Atari solutions

2 Neural networks



Success of DQN in Atari solutions

2 Neural networks



Success of DQN in Atari solutions

Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL.

Interesting readings:

- Double DQN (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
- Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
- Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)

Questions' time!

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE