Outline

1. Deep Learning Vs Deep Reinforcement Learning
2. Theoretical Background
3. Deep Learning
4. Deep RL
5. Deep Inverse RL
Machine learning: Learn from data, make predictions/descisions

Reinforcement Learning: Learning optimal policy for sequential decision making problems (considering long term reward)

Deep learning: ML scheme for (un)supervised learning

Deep RL: Integration of DL with RL for function approximation
Theoretical Background

**Maths:** Linear Algebra, Probability and information theory, probabilistic models and inference

**ML problem:**

1. **Dataset:** training, validation, testing
2. **Cost/Loss function:** measure model performance
3. **Optimization Procedure:** minimize error in training data
4. **Model:** artifact created through the training process
**Training error**: error rate in training

**Test error**: generalization error (the expected value of the error on a new input)

**Hyperparameters**: settings to control the algorithms behavior learned during validation process ($k$-fold cross-validation)

**Estimators**: usually vector of parameter estimations (e.g. weights) giving the "best" prediction

**Bias**: expectation over data included in estimator

**MLE**: most common principle for parameter estimation

\[
\theta_{ML} = \arg \max \ p_{model}(X; \theta) \tag{1}
\]
Agent’s behavior: policy $\pi(\alpha_t \mid s_t)$ at each time step $t$

Value function: prediction of the expected future reward

Bellman Equation:

$$V_*(s) = \max_a \{ R(s) + \gamma \sum_{s'} p(s' \mid s) V_*(s') \}$$

Reward Function (linear):

$$R(s) = \alpha_1 \phi_1(s) + \alpha_2 \phi_2(s) + \ldots + \alpha_d \phi_d(s) +$$

Function Approximation: Construct an approximate of a function from generalized examples
Deep Learning - The main idea

- input layer + num of hidden layers + output layer

- Each unit’s input is the weighted sum of units from previous layers
- Weights are on links between layers
- Input representations: nonlinear transformation, activation functions (tanh, logistic, rectified linear unit)
- Compute error derivatives and backpropagate gradients towards the input layer
- Update weights from gradients and optimize loss function

why gradients? best set of parameters regarding cost is found in a function’s minima
Deep Feedforward neural network analysis

- Defines a mapping $y = f(x; \theta)$ and learn parameters $\theta$ to compute $f^*(x)$
- Chain structure $f(x) = f^{(n)}(f^{(n-1)}(\ldots(f^{(1)}(x))))$
- During training we drive $f(x)$ to match $f^*(x)$
Other NN’s

- **Convolutional**: neural networks that use convolution with small kernel in at least one of their layers, "shallow" parameter sharing (receptive fields), used especially in grid processing

- **Recurrent**: connections between units form a directed graph along a sequence, chain structure, output in each time step, parameter sharing, used especially for sequence of values
Deep RL [3]- Q function approximation

- Select random action $\alpha$ with prob $\epsilon$ else select $\alpha_t = \text{arg } \max_{\alpha} Q(\phi(s_t), \alpha; \theta)$
- Observe reward $r_t$ and image $x_{t+1}$
- Set $s_{t+1} = s_t, \alpha_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
- Store transition $(\phi_t, \alpha_t, r_t, \phi_{t+1})$ in $D$
- Sample random minibatch $(\phi_j, \alpha_j, r_j, \phi_{j+1})$ from $D$
- Set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{\alpha'} \hat{Q}(\phi_{j+1}, \alpha'; \theta^-) & \text{otherwise} \end{cases}$
- Perform gradient descent step on $(y_j - Q(\phi_j, \alpha_j; \theta))^2$ with respect to $\theta$
- Every $C$ steps $\hat{Q} = Q$
**Main Idea:** Input in DNN state features $x$, map them to state reward $r$.

MAP estimation: maximize the joint posterior distribution of $D$ under given $r, \theta, x$:

$$\log P(D, \theta | x) = \log P(D | r) + \log P(\theta)$$

$$\frac{\partial L}{\partial \theta} = \frac{\partial L_D}{\partial \theta} + \frac{\partial L_\theta}{\partial \theta} \Rightarrow \frac{\partial L_D}{\partial \theta} = \frac{\partial L_D}{\partial r} \cdot \frac{\partial r}{\partial \theta}$$

$$\frac{\partial L_D}{\partial \theta} = \frac{\partial L_D}{\partial r} \cdot \frac{\partial r}{\partial \theta} = (\mu_D - \mathbb{E}[\mu]) \cdot \frac{\partial}{\partial \theta} f(x, \theta)$$

where expert’s demonstrations $D = \{s_1, s_2, \ldots, s_N\}$ s.t.

$$s_i = \{(s_0, \alpha_0), (s_1, \alpha_1), \ldots, (s_k, \alpha_k)\}$$

reward function: $r = f(x, \theta) = f_1(f_2(\ldots f_n(x, \theta_n), \ldots), \theta_2), \theta_1$ and summation over possible trajectories: $\mathbb{E}[\mu] = \sum_{\zeta:s, \alpha \in \zeta} P(\zeta | r)$
For Further Reading

I. Goodfellow, Y. Bengio, A. Courville
*Deep Learning.*

Y. Li.
https://dblp.org/rec/bib/journals/corr/Li17b

V. Mnih et. al.
Human-level control through deep reinforcement learning

M. Wulfmeier, P. Ondruska, I. Posner
Deep inverse reinforcement learning
https://pdfs.semanticscholar.org/fde4/8677ba592ed5710b14ef2da7fb8c8144feda.pdf