Module VI

Deep Reinforcement Learning

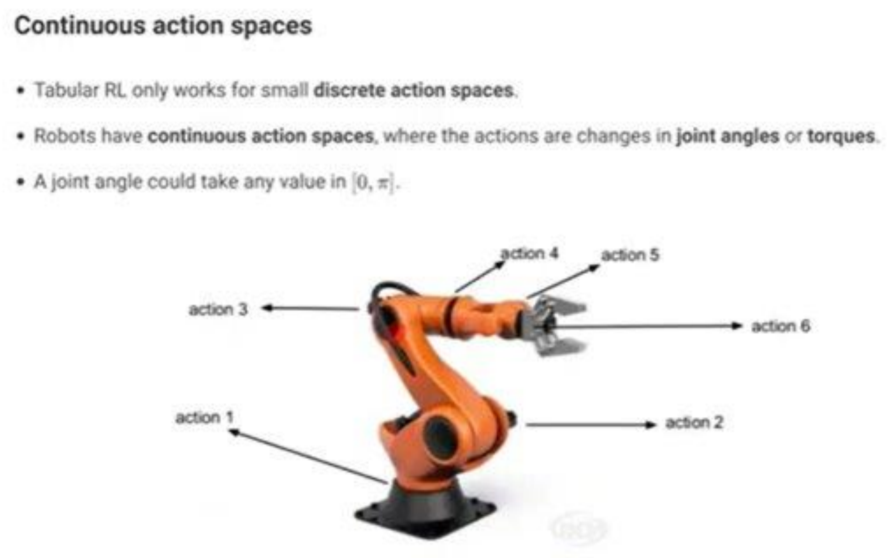
Contents

Function Approximation

- We have seen that our estimates of value functions are represented as a table with one entry for each state or for each state–action pair.
- This is a particularly clear and instructive case, but of course it is limited to problems with small numbers of states and actions.
- The problem is not just the memory needed for large tables, but the time and data needed to fill them accurately.
- In other words, the key issue is that of generalization.
- How can experience with a limited subset of the state space be usefully generalized to produce a good approximation over a much larger subset?

Function Approximation

- In many tasks to which we would like to apply reinforcement learning, most states encountered will never have been experienced exactly before.
- This will almost always be the case when the state or action spaces include continuous variables or complex sensations, such as a visual image.
- The only way to learn anything at all on these tasks is to generalize from previously experienced states to ones that have never been seen.
- we need to combine reinforcement learning methods with existing generalization methods.
- The kind of generalization we require is often called function approximation because it takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation of the entire function



Continuous action spaces

- Tabular RL only works for small **discrete action spaces**.
- Robots have **continuous action spaces**, where the actions are changes in **joint angles** or **torques**.
- A joint angle could take any value in $[0, \pi]$.

Function Approximation

- A function is just a mapping from inputs to outputs, and these can take many forms.
- Let's say you want to train a machine learning model that predicts a person's clothing size.
- The inputs are the person's height, weight, and age. The output is the size.
- What we are trying to do is produce a function that converts a person's height/weight/age combination (a triple of numbers) into a size (perhaps a continuous scalar value or a classification like XS, S, M, L, XL).

Function Approximation

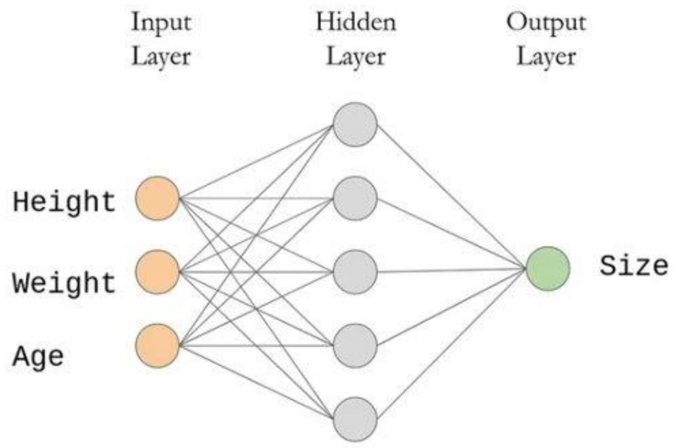
According to machine learning, we can do this by using the following steps:

- Gather data that is representative of the population (height/weight/age numbers for a large number of people, paired with their actual clothing size).
- Train a model to approximate the function that maps the inputs to the outputs of your training data.
- Test your model on unseen data: give it height/weight/age numbers for new people and hopefully it will produce an accurate clothing size!
- Training a model would be easy if the clothing size were just a linear combination of the input variables.
- A simple linear regression could get you good values for a, b, c, and d in the following equation.

```
size = a*height + b*weight + c*age + d
```

Function Approximation

- However, we **cannot** assume, in general, that an output is a linear combination of input variables.
- Conditions in real life are complicated. Rules have exceptions and special cases.
- Examples like handwriting recognition and image classification clearly require very complicated patterns to be learned from high-dimensional input data.
- Wouldn’t it be great if there were a way to approximate **any** function?
- According to the Universal Approximation Theorem, a neural network with a single hidden layer can do exactly that.



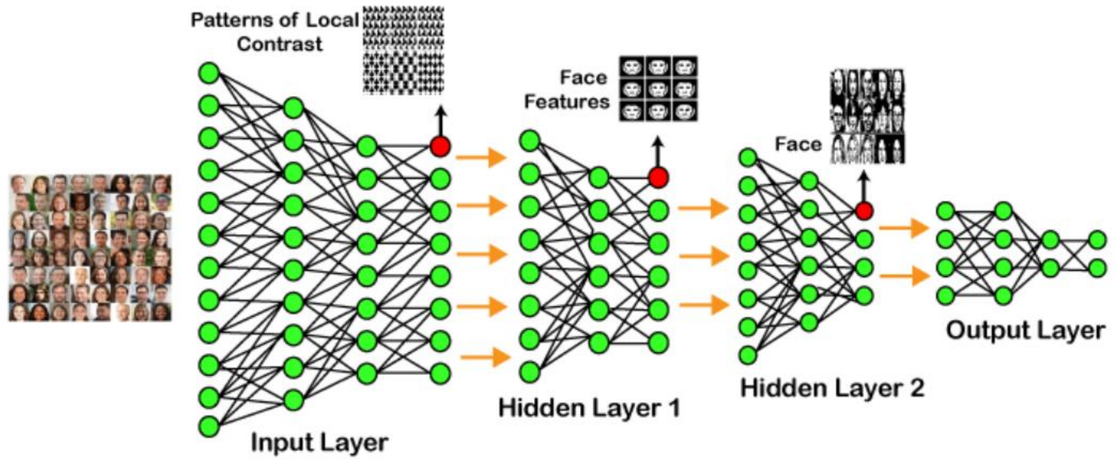
Function Approximators

There are many function approximators like:

1. Artificial Neural Networks
2. Decision Tree
3. Nearest Neighbor
4. Fourier/wavelet bases
5. Coarse coding

Artificial Neural Network as Function Approximator

- ANN, comprising many layers, drive deep learning.
- Deep Neural Networks (DNNs) are such types of networks where each layer can perform complex operations such as representation and abstraction that make sense of images, sound, and text.
- Considered the fastest-growing field in machine learning, deep learning represents a truly disruptive digital technology, and it is being used by increasingly more companies to create new business models.



Advantages of Function Approximation

- Deep reinforcement learning replaces tabular methods of estimating state values with function approximation
- Function approximation not only eliminates the need to store all state and value pairs in a table, it enables the agent to generalize the value of states it has never seen before, or has partial information about, by using the values of similar states.
- Much of the exciting advancements in deep reinforcement learning have come about because of the strong ability of neural networks to generalize across enormous state spaces.

Drawbacks of Tabular Implementation

- An important property of function-approximation methods is how well they generalize to unseen inputs.
- **For example, how good is a Q-function estimate for unvisited (s, a) pairs?** Linear (e.g., linear regression) and nonlinear (e.g., neural networks) function approximation methods are both capable of some generalization, whereas tabular methods are not.
- Let’s consider a tabular representation for Qπ(s, a). Suppose the state-action space is very large, with millions of (s, a) pairs, and at the beginning of training each cell representing a particular Qπ(s, a) is initialized to 0.
- During training, an agent visits (s, a) pairs and the table is updated, but the unvisited (s, a) pairs continue to have Qπ(s, a) = 0. Since the state-action space is large, many (s, a) pairs will remain unvisited and their Q-value estimates will remain at 0 even if (s, a) is desirable, with Qπ(s, a) 0.
- The main issue is that a tabular function representation does not learn anything about how different states and actions relate to each other.

Deep Learning and Reinforcement Learning

- **Deep Learning:** is essentially an autonomous, self-teaching system in which you use existing data to train algorithms to find patterns and then use that to make predictions about new data.
- **Reinforcement learning:** is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

Deep Learning Vs Reinforcement Learning

DL

- Deep learning is learning from a training set and then applying that learning to a new data set.
- DL is data driven
- Used in the areas of speech and picture recognition, the dimension reduction task and pretraining for deep networks
- Deep learning is focused mostly on recognition and has a weaker connection to interactive learning

RL

- Reinforcement learning is dynamically learning by adjusting actions based in continuous feedback to maximize a reward.
- It is goal driven
- Used in the fields of robotics, computer gaming, Telecommunications, Healthcare, elevator scheduling
- Reinforcement learning is a type of artificial intelligence that can be enhanced by the use of feedback, making it more comparable to the capabilities of the human brain than deep learning.

Deep Reinforcement Learning

- Deep reinforcement learning (deep RL) is machine learning that combines reinforcement learning (RL) and deep learning.
- RL considers the problem of a computational agent learning to make decisions by trial and error.
- **Deep RL** incorporates deep learning into the solution, allowing agents to make decisions from unstructured input data without manual engineering of the state space.
- Deep RL algorithms are able to take in very large inputs (e.g. every pixel rendered to the screen in a video game) and decide what actions to perform to optimize an objective (e.g. maximizing the game score).
- Deep reinforcement learning has been used for a diverse set of applications including but not limited to robotics, video games, natural language processing, computer vision, education, transportation, finance and healthcare.

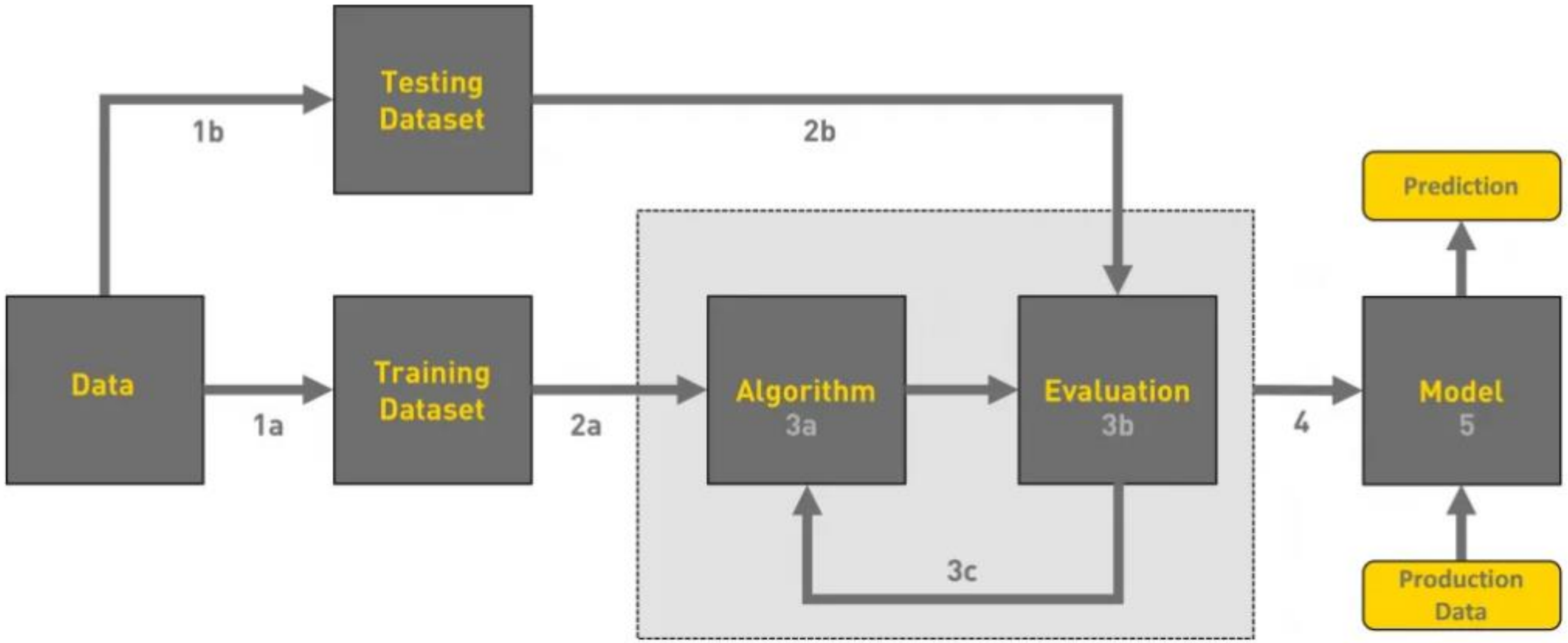
Deep RL Applications

- In 2022, AI agent named AlphaStar beat the world's best StarCraft II player – and this is particularly interesting because unlike games like Chess and Go, players in StarCraft don't know what their opponent is doing. Instead, players had to make an initial strategy then adapt as they found out what their opponent was planning.
- But how is that even possible? If a model has a neural network of more than five layers, it has the ability to cater to high dimensional data. Due to this, the model can learn to identify patterns on its own without having a human engineer curate and select the variables which should be input into the model to learn.

Deep RL Applications

- In open-ended scenarios, you can really see the beauty of deep reinforcement learning.
- Consider the example of booking a table at a restaurant or placing an order for an item—situations in which the agent has to respond to any input from the other end.
- Deep reinforcement learning may be used to train a conversational agent directly from the text or audio signal from the other end. When using an audio signal, the agent may also learn to pick up on subtle cues in the audio such as pauses, intonation, etc—this is the power of deep reinforcement learning.

Machine Learning Workflow

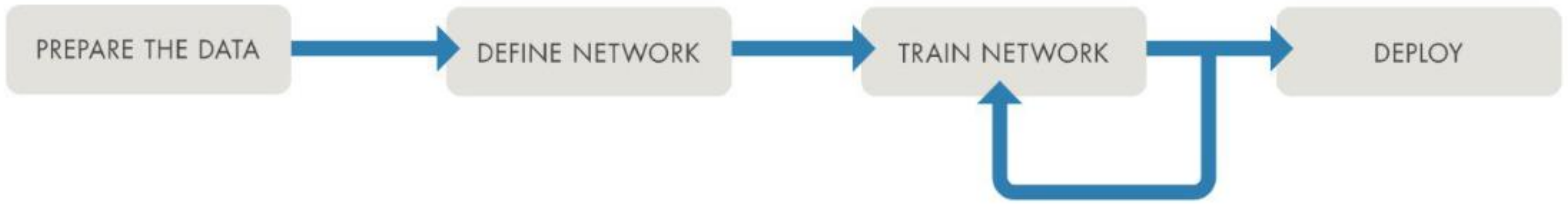


Workflow includes all the steps required to build the proper machine learning project from scratch.

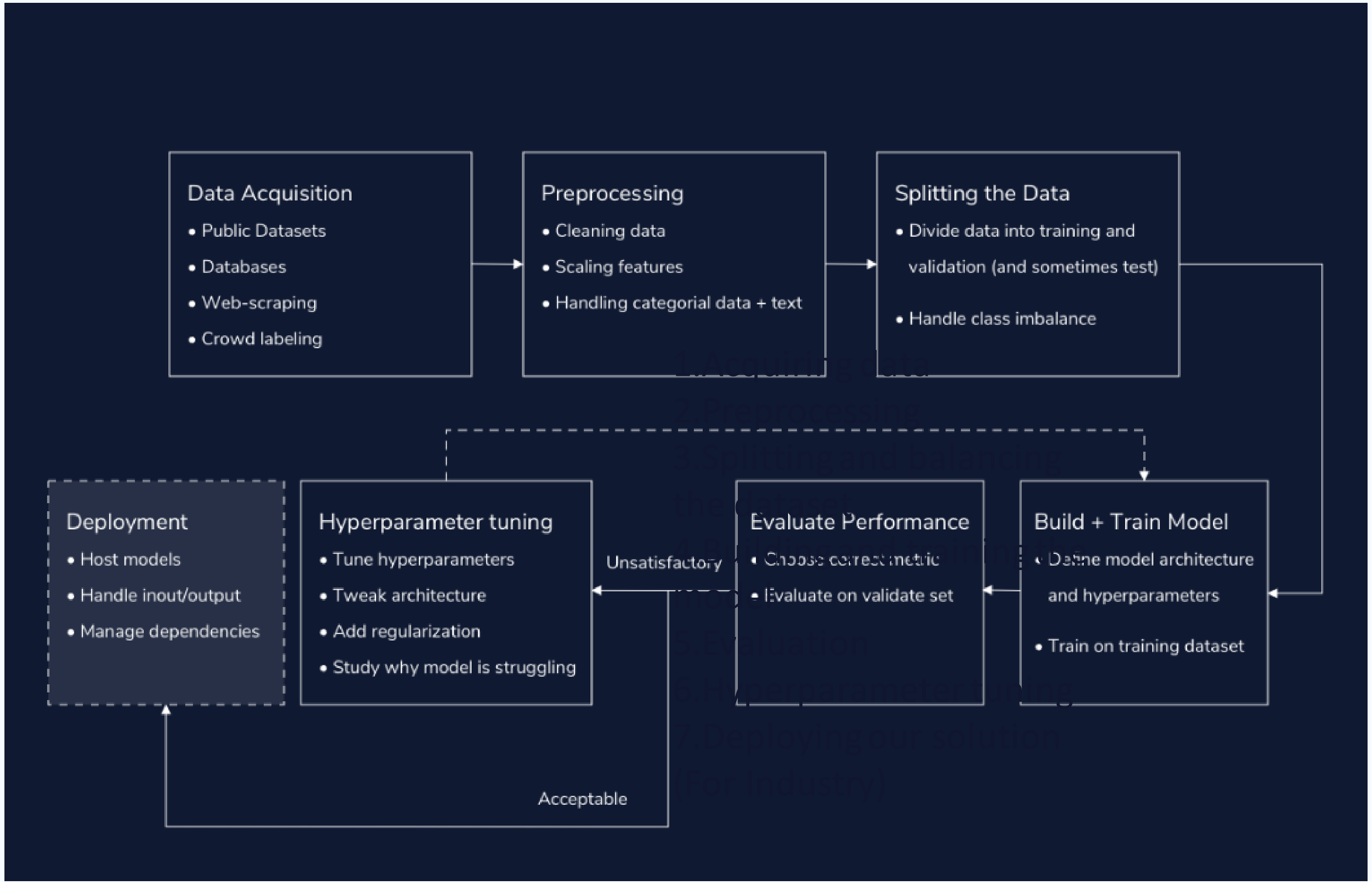
Deep Learning Workflow

A traditional deep learning workflow (Figure 1) consists of four main steps:

- Prepare the data
- Define the network
- Train the network
- Deploy the trained model



Deep Learning Workflow

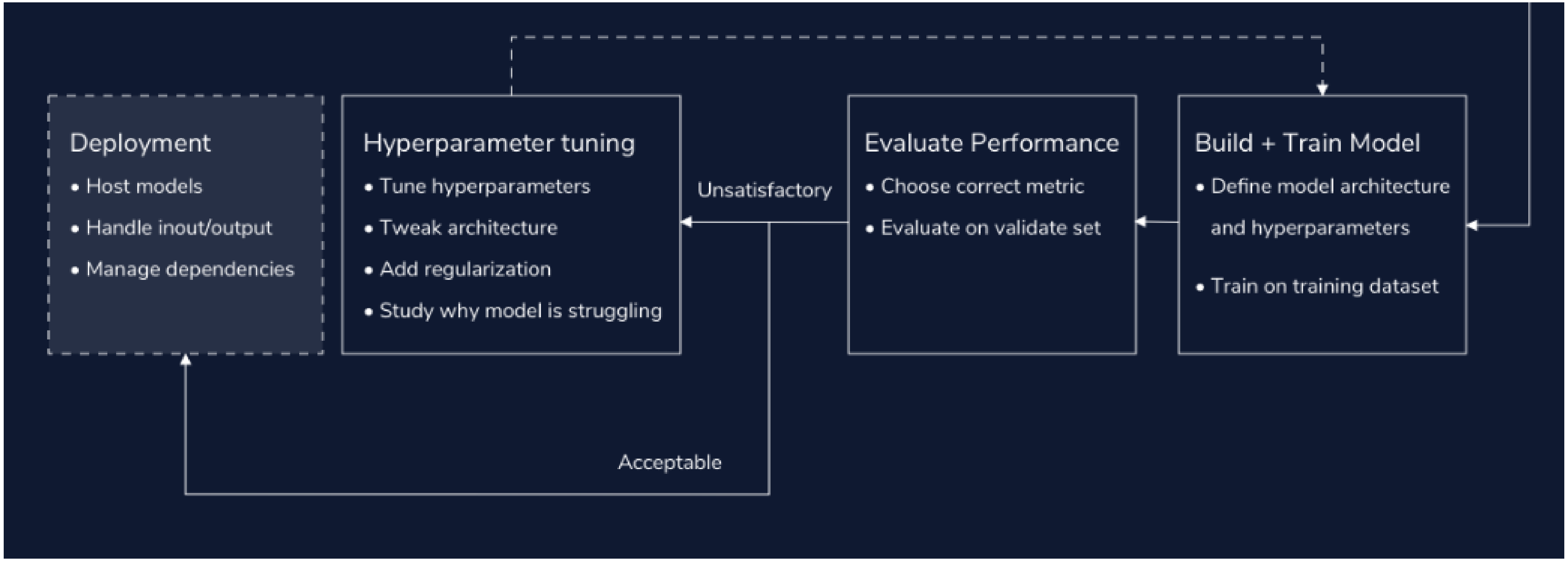


1. Acquiring data
2. Preprocessing
3. Splitting and balancing the dataset
4. Building and training the model
5. Evaluation
6. Hyperparameter tuning
7. Deploying our solution (For Industry)

Training in Deep Learning

- This is one of the most critical and time-consuming steps in the entire workflow.
- It consists of training your model and fine-tuning it to achieve a desired level of performance in the validation set.
- It includes a number of highly technical decisions, such as choice of optimizer and its hyperparameters (batch size, learning rate), cost function, number of epochs, to name just a few.
- At the end of this step you should have a trained model whose performance on the validation set suggests that it solves the original problem satisfying an agreed-upon metric of success.

Deep Learning Training Workflow



1. Build and Train Model
2. Evaluate Performance and Hyperparameter tuning if needed
3. Deployment

DL Training Workflow: Build and Train Model

- Once we have our dataset, it’s time to choose our loss function, and our layers.
- For each layer, we also need to select a reasonable number of hidden units. There is no absolute science to choosing the right size for each layer, nor the number of layers – it all depends on your specific data and architecture.
- It’s good practice to start with a few layers (2-6).
- Usually, we create each layer with between 32 and 512 hidden units.
- We also tend to decrease the size of hidden layers as we move upwards - through the model.
- We usually try SGD and Adam optimizers first.
- When setting an initial learning rate, a common practice is to default to `0.01`

DL Training Workflow: Performance Evaluation

- Each time we train the model, we evaluate its performance on our validation (Test) set.
- Our performance on the validation set gives us a sense for how our model will perform on new, unseen data.
- When considering performance, it’s important to choose the correct metric.
- If our data set is heavily imbalanced, accuracy will be less meaningful. In this case, we likely want to consider metrics like precision and recall.
- F1-score is another useful metric that combines both precision and recall.
- A confusion matrix can help visualize what data-points are misclassified etc.

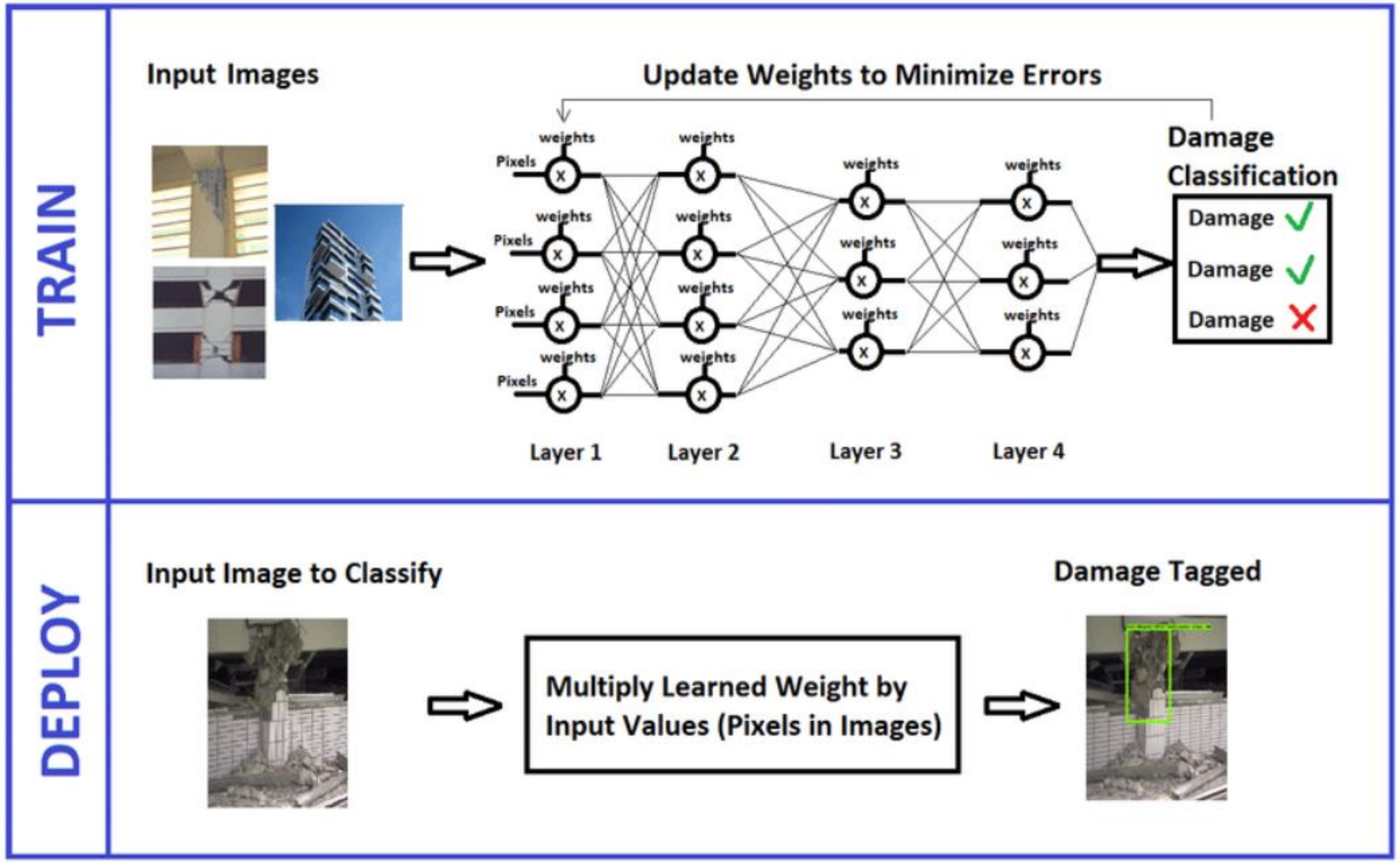
DL Training Workflow: Tuning Parameters

- We will almost always need to iterate upon our initial hyperparameters.
- When training and evaluating our model, we explore different learning rates, batch sizes, architectures, and regularization techniques.
- As we tune our parameters, we should watch our loss and metrics, and be on the lookout for clues as to why our model is struggling.
- Unstable learning means that we likely need to reduce our learning rate and or/increase our batch size. A disparity between performance on the training and evaluation sets means we are overfitting, and should reduce the size of our model
- Poor performance on both the training and the test set means that we are underfitting, and may need a larger model or a different learning rate.
- A common practice is to start with a smaller model and scale up our hyperparameters until we do see training and validation performance diverge, which means we have overfit to our data.

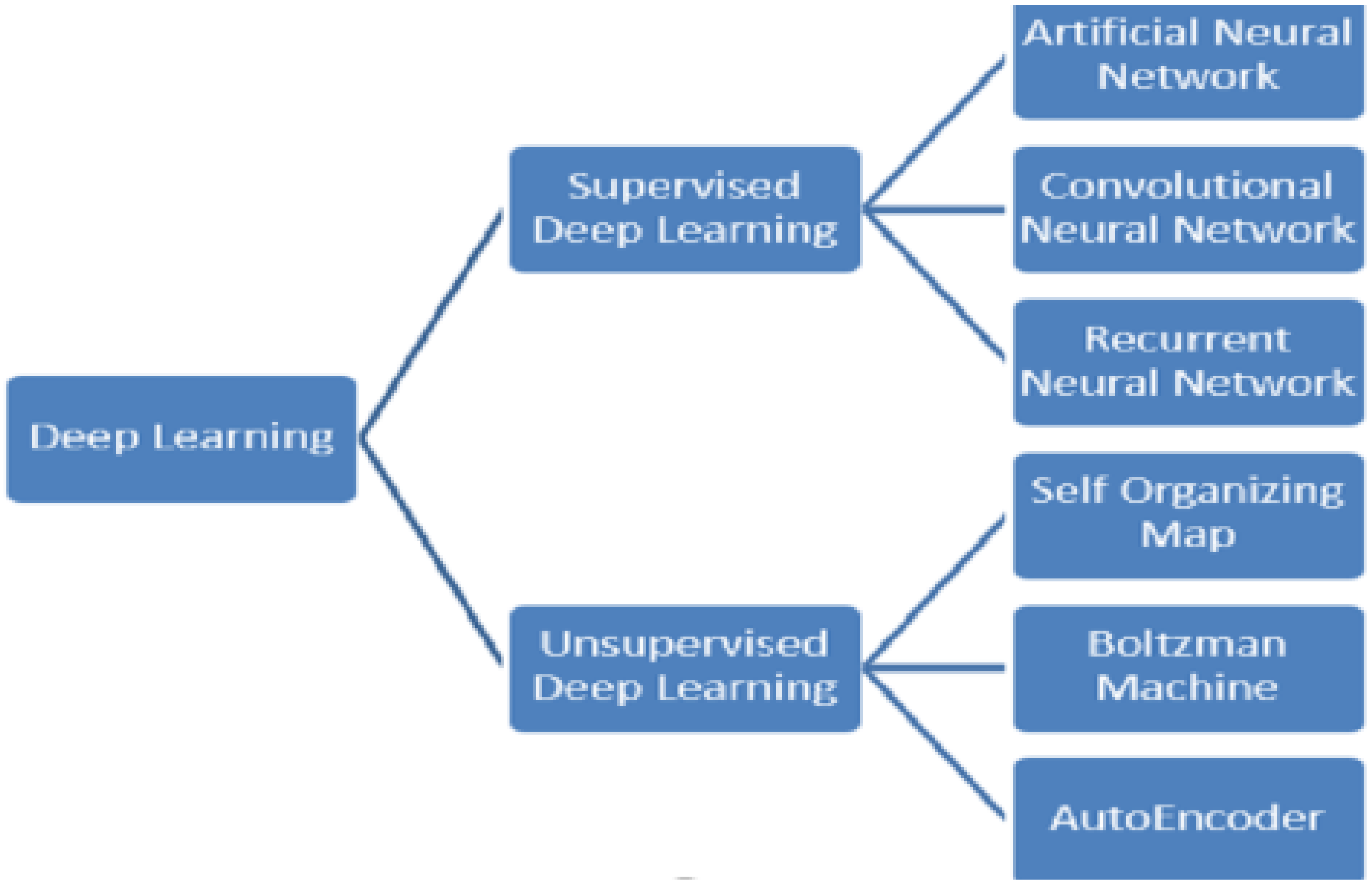
DL Training Workflow: Deployment

- Once you're confident (based on the performance evaluation on the validation set) that you have a model that meets the expectations and requirements set earlier, it is time to bring up the test set and evaluate the model's performance on the test set, which will provide a measure of the model's ability to generalize to previously unseen data.
- At the end of this step, you should have a model whose performance on the test set satisfies an agreed-upon metric of success and is ready for deployment.

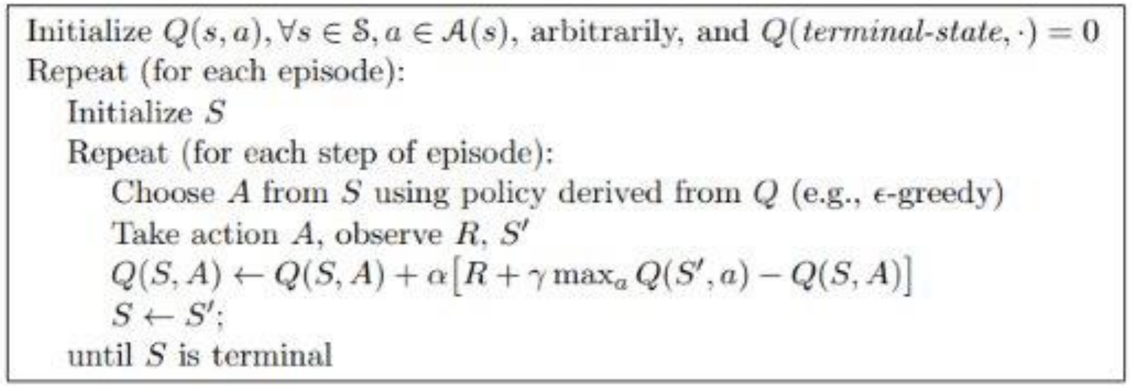
Example: Deep Learning



Categories of Deep Learning



Deep Q - Network



```
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

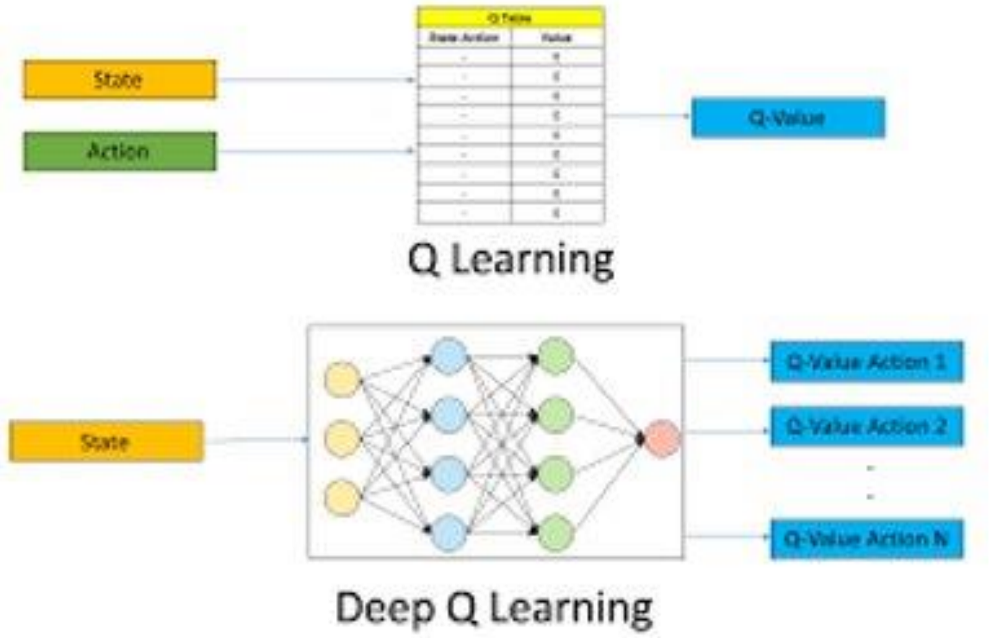
- We have a basic strategy – at any given state, perform the action that will eventually yield the highest cumulative reward (algorithms like this we call greedy)
- We Implement it by drawing a Q table to store all possible state-action combinations and use it to save Q Values. We can then update it using the Bellman Equation as an update rule.
- what happens when the number of states and actions becomes very large? This is actually not that rare – even a simple game such as Tic Tac Toe has hundreds of different states and we multiply this number by 9, which is the number of possible actions. So how will we solve really complex problems?

Deep Q- Network

- Cobine Q Learning and Deep Learning, which yields *Deep Q Networks*. The idea is simple: we'll replace the the Q Learning's table with a neural network that tries to approximate Q Values. It is usually referred to as the *approximator* or the *approximating function*, and denoted as *Q(s,a; θ)*, where *θ* represents the trainable weights of the network.
- Now it only makes sense to use the Bellman Equation as the cost function — but what exactly will we minimize? Let's take another look at it:

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

- The "=" sign marks *assignment*, but is there any condition which will also satisfy an *equality*? Well, yes — when the Q Value reached its converged and final value. And this is *exactly* our goal — so we can minimize the difference between the left-hand side and the right-hand side



Deep Q-Network

- Our cost function:

$$Cost = \left[Q(s, a; \theta) - \left(r(s, a) + \gamma \max_{a} Q(s', a; \theta) \right) \right]^2$$

DQN cost function

- Does this looks familiar? Probably – it’s the Mean Square Error function, where the current Q Value is the prediction (*y*), and the immediate and future rewards are the target (*y’*):

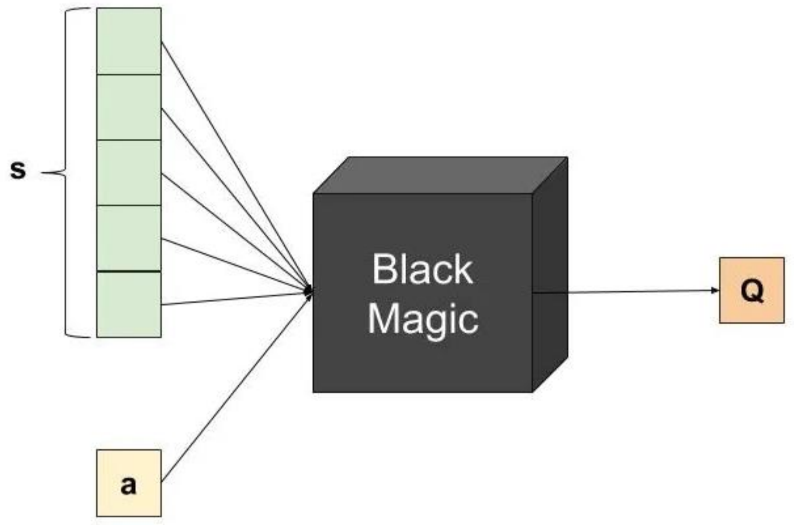
$$MSE = \frac{1}{n} \sum_{1}^{n} (y_i - y_i')^2$$

Mean square error function

- This is why *Q(s’,a; ϑ)* is usually referred to as *Q-target*.

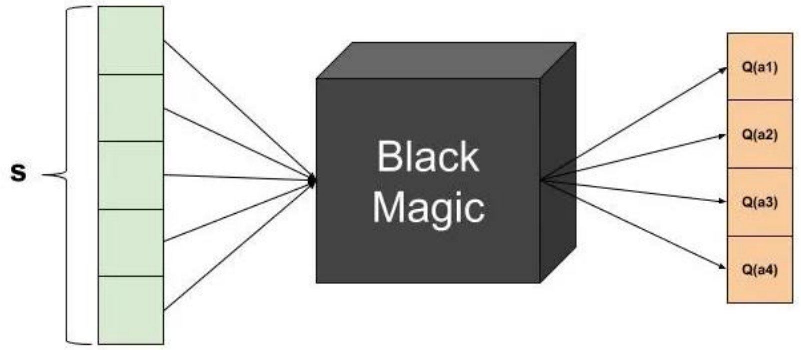
DEEP Q Network

- Things are getting real, so let's talk architecture: if imitating a table, the network should receive as input the state and action, and should output a Q Value:



DEEP Q Network

- Note that the cost function requires the *maximal* future Q Value, so we'll need several network predictions for a single cost calculation. So instead, we can use the following architecture:



Here we provide the network only the state *s* as input and receive Q Values for all possible actions at-once.

Challenges in Deep Q-Learning

- One of the key challenges in implementing Deep Q-Learning is that the Q-function is typically non-linear and can have many local minima.
- This can make it difficult for the neural network to converge to the correct Q-function.
- To address this, several techniques have been proposed, such as experience replay and target networks.

Challenges in Deep Q-Learning

- **Experience replay** is a technique where the agent stores a subset of its experiences (state, action, reward, next state) in a memory buffer and samples from this buffer to update the Q-function.
- This helps to decorrelate the data and make the learning process more stable.
- **Target networks**, on the other hand, are used to stabilize the Q-function updates.
- In this technique, a separate network is used to compute the target Q-values, which are then used to update the Q-function network.

References

- Reinforcement Learning 5: Function Approximation and Deep Reinforcement Learning - Bing video
- 6. Function approximation — Deep Reinforcement Learning (julien-vitay.net)
- Qrash Course: Reinforcement Learning 101 & Deep Q Networks in 10 Minutes | by Shaked Zychlinski | Towards Data Science
- https://www.forbes.com/sites/bernardmarr/2018/10/22/artificial-intelligence-whats-the-difference-between-deep-learning-and-reinforcement-learning/?sh=5644c4eb271e
- https://www.techopedia.com/reinforcement-learning-vs-deep-reinforcement-learning-whats-the-difference/2/34039
- https://www.codecademy.com/article/deep-learning-workflow
- https://www.kdnuggets.com/2020/09/mathworks-deep-learning-workflow.html