RL Algorithms Revisited

Q-Learning Algorithm

$$Q^*(s,a) = \quad E(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a)$$

- Initialize the Q-table

- Until convergence:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \Bigg)}^{\text{temporal difference}}$$

Q-Learning Pseudocode

Algorithm 1: Epsilon-Greedy Q-Learning Algorithm

```
Data: α: learning rate, γ: discount factor, ε: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated
        optimal policy π*
/* Initialization                                              */
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) ← 0;
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                              */
for each episode do
    /* Initialize state S, usually by resetting the
       environment                                             */
    Initialize state S;
    for each step in episode do
        do
            /* Choose action A from S using epsilon-greedy
               policy derived from Q                           */
            A ← SELECT-ACTION(Q, S, ε);
            Take action A, then observe reward R and next state S';
            Q(S, A) ← Q(S, A) + α [ R + γ max_a Q(S', a) - Q(S, A)];
            S ← S';
        while S is not terminal;
    end
end
```

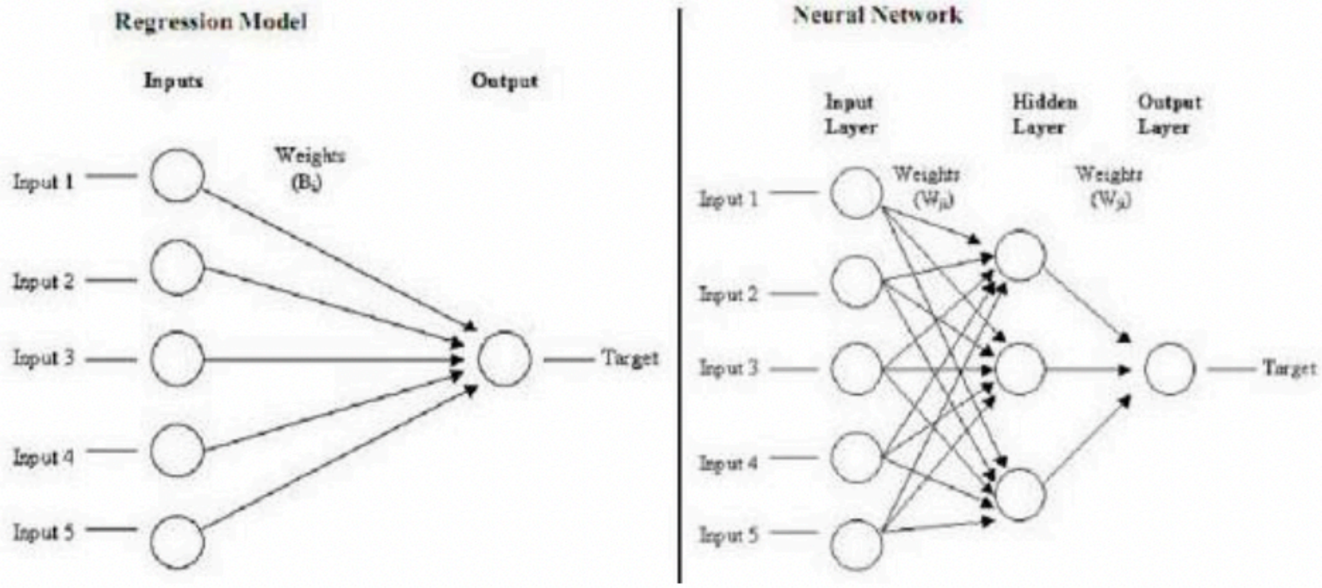
Algorithm 2: Epsilon-Greedy Action Selection

```
Data: Q: Q-table generated so far, : a small number, S: current
      state
Result: Selected action
Function SELECT-ACTION(Q, S, ε) is
    n ← uniform random number between 0 and 1;
    if n < ε then
        A ← random action from the action space;
    else
        A ← maxQ(S,.);
    end
    return selected action A;
end
```

Notebook example

Neural Networks

Neural Networks (NNs) and Forecasting



- Advantage:
  - By universal approximation theorem, can theoretically approximate almost any function!
  - Can capture complex non-linear dependencies
  - Scale favorably with large amounts of data
  - Very effective for language and image processing
- Disadvantages:
  - Expensive to train
  - Produces "black box" solution
  - Overfitting issues
- When it comes to time series, can be very effective too, but compare to simpler baseline

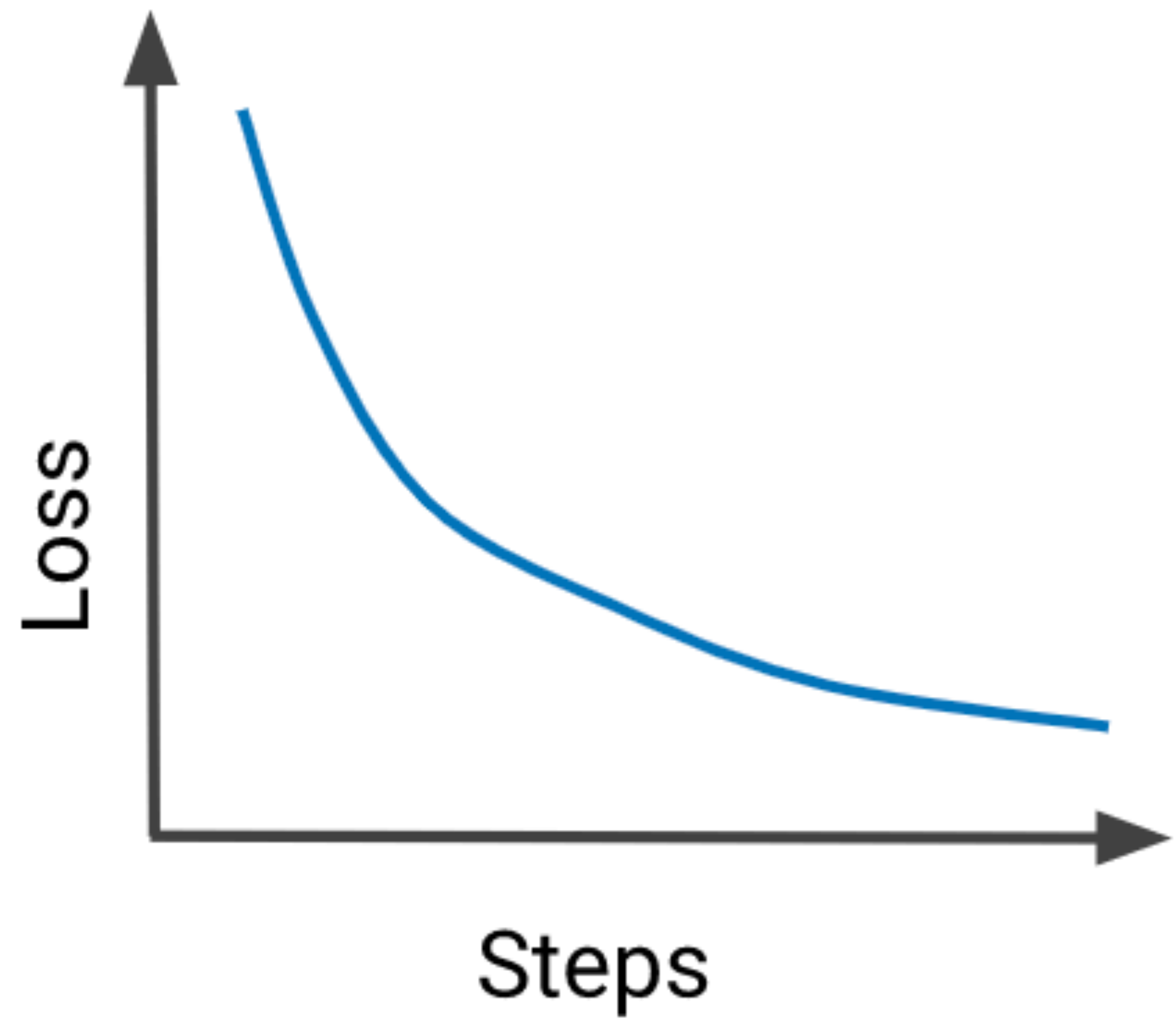
Packages to train neural networks

- pytorch, tensorflow
- http://playground.tensorflow.org/

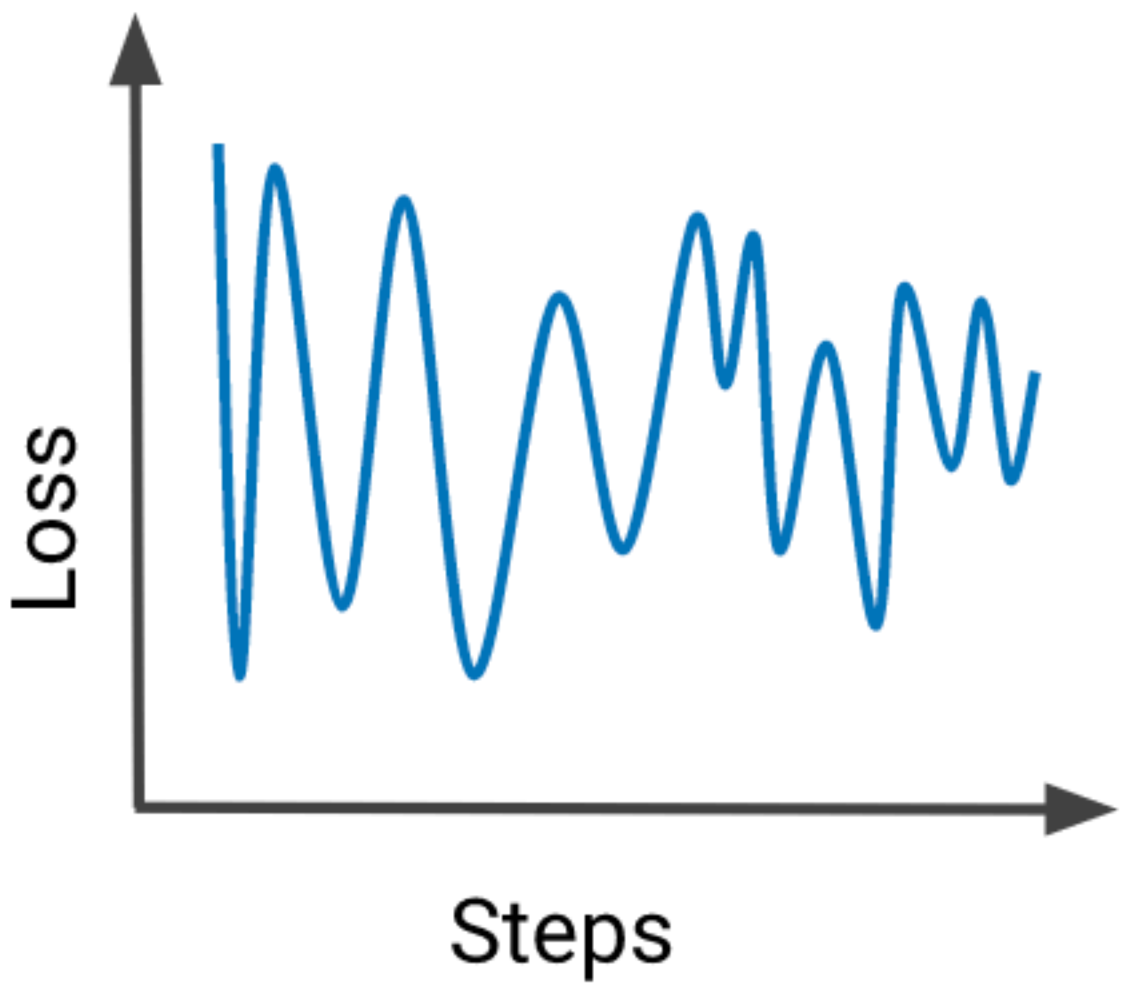
Build a `tf.keras.Sequential` model by stacking layers.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

Analyzing Training Loss

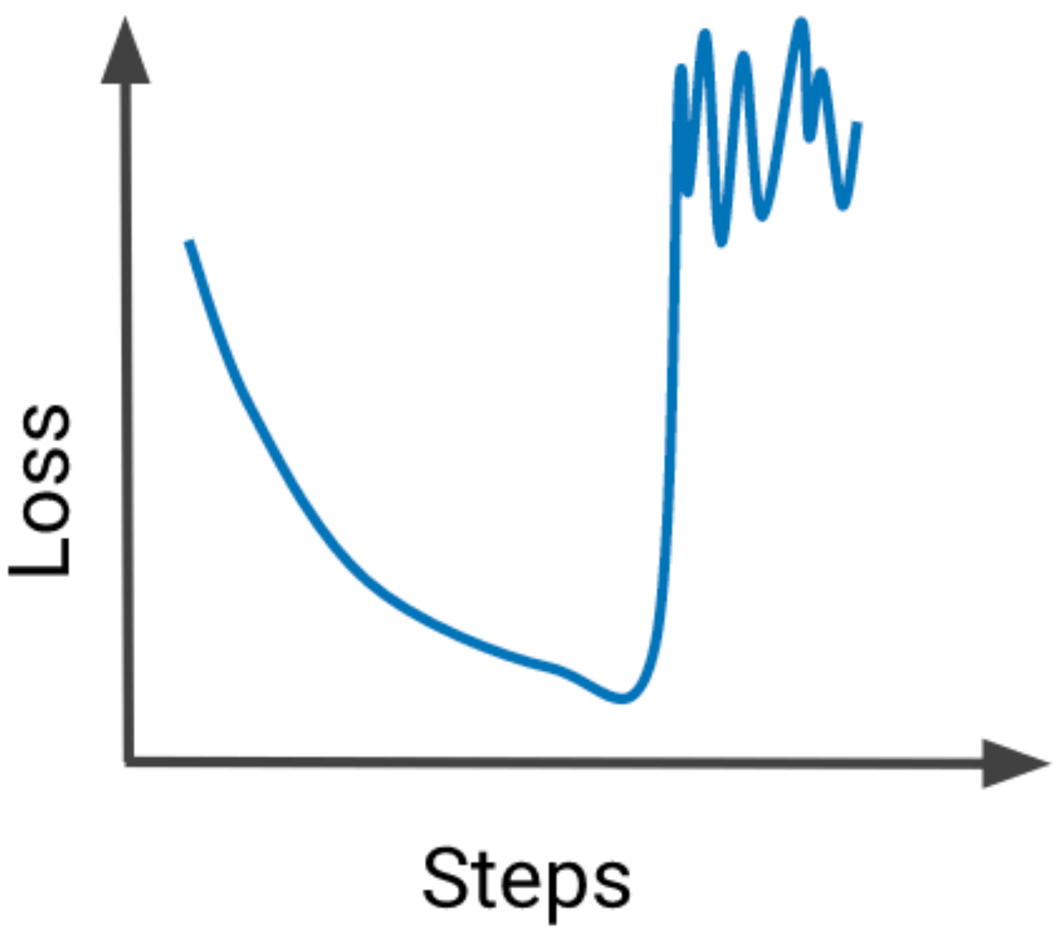


Training Loss Troubleshooting



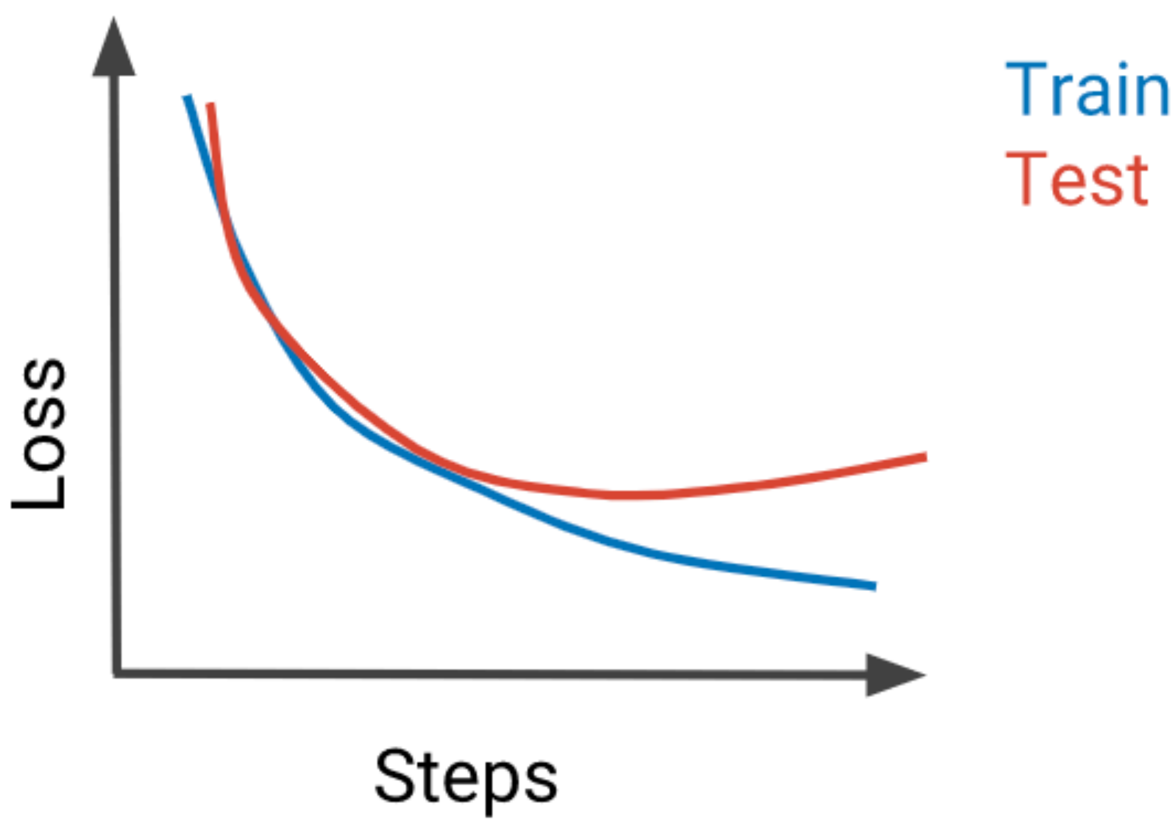
- Check that features can predict the labels
- Look for data outliers
- Reduce the **learning rate**
- Simplify the data to smaller dataset prediction for which you understand
  - Stabilize model on small dataset, then proceed the the bigger one

Training Loss Troubleshooting



- NaN in input data
- Gradients explode due to anomalies in input data

Training Loss Troubleshooting



- The model is overfitting!
- Reduce model capacity
- Reduce the number of input features
- Add regularization