CS545: TD Value Updates as Gradient Descent

Chuck Anderson

Department of Computer Science
Colorado State University

Fall, 2009
Outline

Mean Squared Error Between Return and Model Output
Reinforcement Learning Objective
Gradient of Objective

Gradient for Q function as Table

Gradient for Q function as Neural Network

Training the Q Neural Network
Reinforcement Learning Objective

- The objective for any reinforcement learning problem is to find the sequence of actions that maximizes (or minimizes) the sum of reinforcements along the sequence.

\[
Q(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1}
\]

Usually formulated as a least squares objective

\[
\text{Minimize } \mathbb{E} \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2
\]
Reinforcement Learning Objective

- The objective for any reinforcement learning problem is to find the sequence of actions that maximizes (or minimizes) the sum of reinforcements along the sequence.
- Reduced to the objective of acquiring the Q function the predicts the expected sum of future reinforcements, because Q determines optimal next action.

\[ Q(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1} \]

Usually formulated as a least squares objective

\[ \min_{Q} E \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2 \]
The objective for any reinforcement learning problem is to find the sequence of actions that maximizes (or minimizes) the sum of reinforcements along the sequence.

Reduced to the objective of acquiring the Q function the predicts the expected sum of future reinforcements, because Q determines optimal next action.

So, the RL objective is to make this approximation as accurate as possible

\[
Q(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1}
\]
Reinforcement Learning Objective

- The objective for any reinforcement learning problem is to find the sequence of actions that maximizes (or minimizes) the sum of reinforcements along the sequence.
- Reduced to the objective of acquiring the Q function the predicts the expected sum of future reinforcements, because Q determines optimal next action.
- So, the RL objective is to make this approximation as accurate as possible

\[
Q(s_t, a_t) \approx \sum_{k=0}^{\infty} r_{t+k+1}
\]

- Usually formulated as a least squares objective

\[
\text{Minimize } E \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2
\]
What is the expectation over?

$$\text{Minimize } E \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2$$
What is the expectation over?

Minimize \( E \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2 \)

Sources of randomness
What is the expectation over?

\[
\text{Minimize } \mathbb{E} \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2
\]

Sources of randomness
- action selection

Sources of randomness
- action selection
What is the expectation over?

$$\text{Minimize } \mathbb{E}\left(\sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t)\right)^2$$

Sources of randomness
- action selection
- state transitions

What is the expectation over?
What is the expectation over?

Minimize $E \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2$

Sources of randomness
- action selection
- state transitions
- reinforcement received
What is the expectation over?

$$\text{Minimize } \mathbb{E} \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2$$

Sources of randomness
- action selection
- state transitions
- reinforcement received

Major problem.....We don’t know $$\sum_{k=0}^{\infty} r_{t+k+1}$$. What do we do?
What is the expectation over?

$$\text{Minimize } \mathbb{E}\left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t) \right)^2$$

Sources of randomness
- action selection
- state transitions
- reinforcement received

Major problem.....We don’t know $\sum_{k=0}^{\infty} r_{t+k+1}$. What do we do?

Approximate it by

$$\sum_{k=0}^{\infty} r_{t+k+1} = r_{t+1} + \sum_{k=1}^{\infty} r_{t+k+1}$$

$$= r_{t+1} + \sum_{k=0}^{\infty} r_{t+1+k+1}$$

$$\approx r_{t+1} + Q(s_{t+1}, a_{t+1})$$
Gradient of Reinforcement Learning Objective

- Now our minimization problem is the following. Let $Q$ be a parameterized function, with parameters $w$. How do we try to minimize it?

\[
\text{Minimize } J(w) = \mathbb{E} \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t; w) \right)^2
\]

\[= \mathbb{E} \left( r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w) \right)^2\]
Gradient of Reinforcement Learning Objective

Now our minimization problem is the following. Let $Q$ be a parameterized function, with parameters $w$. How do we try to minimize it?

Minimize $J(w) = E \left( \sum_{k=0}^{\infty} r_{t+k+1} - Q(s_t, a_t; w) \right)^2$

$= E \left( r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w) \right)^2$

Right! Take the gradient of $J(w)$ with respect to $w$ and do gradient descent. $Q$ appears twice in this expression. A very common approach is to only take the gradient with respect to $Q(s_t, a_t; w)$ and treat $Q(s_{t+1}, a_{t+1}; w)$ as constant. Not correct, but works well and simplifies the resulting algorithm.
Recall that the expectation operator is a sum of possible values, weighted by their probabilities. If $d$ is the value on the top of a (fair) dice,

$$E(d) = \frac{1}{6} 1 + \frac{1}{6} 2 + \frac{1}{6} 3 + \frac{1}{6} 4 + \frac{1}{6} 5 + \frac{1}{6} 6$$
Recall that the expectation operator is a sum of possible values, weighted by their probabilities. If $d$ is the value on the top of a (fair) dice,

$$E(d) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6}$$

So the gradient of an expected value is the expected value of the gradients.

$$\nabla_w J(w) = 2E\left(r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w)\right)(-1)\nabla_w Q(s_t, a_t; w)$$
Recall that the expectation operator is a sum of possible values, weighted by their probabilities. If $d$ is the value on the top of a (fair) dice,

$$E(d) = \frac{1}{6}1 + \frac{1}{6}2 + \frac{1}{6}3 + \frac{1}{6}4 + \frac{1}{6}5 + \frac{1}{6}6$$

So the gradient of an expected value is the expected value of the gradients.

$$\nabla_w J(w) =$$

$$2E\left(r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w)\right)(-1)\nabla_w Q(s_t, a_t; w)$$

The expectation operator requires knowledge of the probabilities of all of those random effects. Instead, we will just sample the world.
Recall that the expectation operator is a sum of possible values, weighted by their probabilities. If \( d \) is the value on the top of a (fair) dice,

\[ E(d) = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 5 + \frac{1}{6} \cdot 6 \]

So the gradient of an expected value is the expected value of the gradients.

\[ \nabla_w J(w) = 2E \left( r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w) \right) (-1) \nabla_w Q(s_t, a_t; w) \]

The expectation operator requires knowledge of the probabilities of all of those random effects. Instead, we will just sample the world.

More probable events will take place more often. If we just add updates to \( w \) for each sample, we effectively have done the expectation.
Outline

Mean Squared Error Between Return and Model Output
Reinforcement Learning Objective
Gradient of Objective

Gradient for Q function as Table

Gradient for Q function as Neural Network

Training the Q Neural Network
Gradient for Q Table

- What is the gradient with Q is a table, like the maze and the tic-tac-toe problem?
Gradient for Q Table

- What is the gradient with $Q$ is a table, like the maze and the tic-tac-toe problem?
- Must decide what the $w$ parameters are for a table.
Gradient for Q Table

- What is the gradient with $Q$ is a table, like the maze and the tic-tac-toe problem?
- Must decide what the $w$ parameters are for a table.
- They are just the entries in the table. Can formulate the selection of the correct parameter in the table as a dot product.

Let $x_t^T$ be a column vector of length equal to the number of table cells and with values that are all 0's and a single 1. The 1 designates the cell corresponding to $s_t$ and $a_t$. Then $Q(s_t, a_t; w) = x_t^T w$.

$\nabla_w Q(s_t, a_t; w) = \nabla_w (x_t^T w)$.
Gradient for Q Table

- What is the gradient with $Q$ is a table, like the maze and the tic-tac-toe problem?
- Must decide what the $w$ parameters are for a table.
- They are just the entries in the table. Can formulate the selection of the correct parameter in the table as a dot product.
- Let $x_t$ be a column vector of length equal to the number of table cells and with values that are all 0’s and a single 1. The 1 designates the cell corresponding to $s_t$ and $a_t$. Then

\[ Q(s_t, a_t; w) = x_t^T w \]
Gradient for Q Table

- What is the gradient with $Q$ is a table, like the maze and the tic-tac-toe problem?
- Must decide what the $w$ parameters are for a table.
- They are just the entries in the table. Can formulate the selection of the correct parameter in the table as a dot product.
- Let $x_t$ be a column vector of length equal to the number of table cells and with values that are all 0’s and a single 1. The 1 designates the cell corresponding to $s_t$ and $a_t$. Then

$$Q(s_t, a_t; w) = x_t^T w$$

- So,

$$\nabla_w Q(s_t, a_t; w) = \nabla_w (x_t^T w) = x_t$$
Let's define the temporal-difference error to be $\delta_t$

$$\delta_t = r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w)$$
Let’s define the temporal-difference error to be $\delta_t$

$$\delta_t = r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w)$$

Now

$$\nabla_w J(w) = 2\mathbb{E}\left(\delta_t(-1)\nabla_w Q(s_t, a_t; w)\right)$$

$$= -2\mathbb{E}(\delta_t x_t)$$

$$= \begin{cases} 
-2\mathbb{E}(\delta_t), & \text{for cell for } s_t, a_t; \\
0, & \text{for all other cells}
\end{cases}$$
Let’s define the temporal-difference error to be $\delta_t$

$$\delta_t = r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w)$$

Now

$$\nabla_w J(w) = 2E\left(\delta_t (-1) \nabla_w Q(s_t, a_t; w)\right)$$

$$= -2E(\delta_t x_t)$$

$$= \begin{cases} -2E(\delta_t), & \text{for cell for } s_t, a_t; \\ 0, & \text{for all other cells} \end{cases}$$

Replacing the expectation with samples, we get the TD update rule for Q tables

$$w \leftarrow w - \nabla_w J(w)$$

$$\leftarrow w + \rho \delta_t x_t$$

which is really the same as what we have seen before,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \rho \delta_t$$

where the update to just the $s_t, a_t$ cell of Q was implicit.
Outline

Mean Squared Error Between Return and Model Output
Reinforcement Learning Objective
Gradient of Objective

Gradient for Q function as Table

Gradient for Q function as Neural Network

Training the Q Neural Network
Other Function Approximators for $Q$

- Using a table for $Q$ has limited use.
Other Function Approximators for Q

- Using a table for Q has limited use.
  - Size of table might be too big.
Other Function Approximators for Q

- Using a table for Q has limited use.
  - Size of table might be too big.
  - Learning could be slow. Experience in one cell does not transfer to other cells.
Using a table for Q has limited use.
- Size of table might be too big.
- Learning could be slow. Experience in one cell does not transfer to other cells.

Let’s use a different function approximator. Uhhh...what shall we use?
Other Function Approximators for Q

- Using a table for Q has limited use.
  - Size of table might be too big.
  - Learning could be slow. Experience in one cell does not transfer to other cells.
- Let’s use a different function approximator. Uhhh...what shall we use?
- Brilliant! A neural network.
Other Function Approximators for Q

- Using a table for Q has limited use.
  - Size of table might be too big.
  - Learning could be slow. Experience in one cell does not transfer to other cells.
- Let’s use a different function approximator. Uhhh...what shall we use?
- Brilliant! A neural network.
- What has to change in our derivation of the gradient?
Q Neural Network

Now \( Q(s_t, a_t; \mathbf{w}) \) is approximated as a neural network. The values of \( \mathbf{w} \) are the \( V \) and \( W \) weights.
Q Neural Network

- Now $Q(s_t, a_t; \mathbf{w})$ is approximated as a neural network. The values of $\mathbf{w}$ are the $V$ and $W$ weights.

- Looks like the neural network will have a single output value and inputs for $s_t$ and $a_t$. To find best action, we would input $s_t$ and try all possible actions $a_t$, calculating the output of the network for each.
Q Neural Network

- Now \( Q(s_t, a_t; \mathbf{w}) \) is approximated as a neural network. The values of \( \mathbf{w} \) are the \( V \) and \( W \) weights.
- Looks like the neural network will have a single output value and inputs for \( s_t \) and \( a_t \). To find best action, we would input \( s_t \) and try all possible actions \( a_t \), calculating the output of the network for each.
- Pick the action that produces the best neural network output.
Q Neural Network

- Now \( Q(s_t, a_t; w) \) is approximated as a neural network. The values of \( w \) are the \( V \) and \( W \) weights.
- Looks like the neural network will have a single output value and inputs for \( s_t \) and \( a_t \). To find best action, we would input \( s_t \) and try all possible actions \( a_t \), calculating the output of the network for each.
- Pick the action that produces the best neural network output.
- With small number of possible actions, we have an alternative. Input is just \( s_t \) and we have multiple outputs, one for each action. Then just do one forward pass through the network for a given \( s_t \) and compare the outputs to determine best action.
Here is the general gradient.

\[ \delta_t = r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w) \]

\[ \nabla_w J(w) = -E \left( \delta_t \nabla_w Q(s_t, a_t; w) \right) \]
Gradient of Q Neural Network

- Here is the general gradient.

\[ \delta_t = r_{t+1} + Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a_t; w) \]

\[ \nabla_w J(w) = -E\left( \delta_t \nabla_w Q(s_t, a_t; w) \right) \]

- How is this specialized for a neural network? Must only deal with \( \nabla_w Q(s_t, a_t; w) \).
Must only back-propagate the temporal-difference error for the action selected at time $t$.
Must only back-propagate the temporal-difference error for the action selected at time $t$

or, in matheze, let chosen action be $k$,

$$ \nabla_w Q(s_t, a_t; w) = \nabla_w y_k $$
Must only back-propagate the temporal-difference error for the action selected at time $t$

or, in matheze, let chosen action be $k$,

$$\nabla_w Q(s_t, a_t; w) = \nabla_w y_k$$

This is exactly what we have done before when using neural networks for nonlinear regression, except we only deal with one of the outputs.
Here are the gradient descent updates we had for neural networks

\[
\begin{align*}
Z &= h(\tilde{X}V) \\
Y &= \tilde{Z}W \\
V &\leftarrow V + \rho h \frac{1}{N} \frac{1}{K} \tilde{X}^T (T - Y)\hat{W}^T \cdot (1 - Z^2) \\
W &\leftarrow W + \rho o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y)
\end{align*}
\]
Here are the gradient descent updates we had for neural networks

\[ Z = h(\tilde{X}V) \]
\[ Y = \tilde{Z}W \]
\[ V \leftarrow V + \rho h \frac{1}{N} \frac{1}{K} \tilde{X}^T (T - Y) \hat{W}^T \cdot (1 - Z^2) \]
\[ W \leftarrow W + \rho_o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y) \]

The key change we need to make is in the error. Our targets are now given by a reinforcement plus the next step predicted Q values. Must assemble a bunch of samples into matrices.
Here are the gradient descent updates we had for neural networks:

\[
Z = h(\tilde{X}V)
\]
\[
Y = \tilde{Z}W
\]
\[
V \leftarrow V + \rho_h \frac{1}{N} \frac{1}{K} \tilde{X}^T \left( (T - Y)\hat{W}^T \cdot (1 - Z^2) \right)
\]
\[
W \leftarrow W + \rho_o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y)
\]

The key change we need to make is in the error. Our targets are now given by a reinforcement plus the next step predicted Q values. Must assemble a bunch of samples into matrices.

Multiple \( s_t \)'s will be collected as rows in \( X \), which, when passed through the neural network, produce \( Z \) and \( Y \).
Here are the gradient descent updates we had for neural networks

\[ Z = h(\tilde{X}V) \]
\[ Y = \tilde{Z}W \]
\[ V \leftarrow V + \rho_h \frac{1}{N} \frac{1}{K} \tilde{X}^T \left( (T - Y)\hat{W}^T \cdot (1 - Z^2) \right) \]
\[ W \leftarrow W + \rho_o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y) \]

The key change we need to make is in the error. Our targets are now given by a reinforcement plus the next step predicted Q values. Must assemble a bunch of samples into matrices.

Multiple \( s_t \)'s will be collected as rows in \( X \), which, when passed through the neural network, produce \( Z \) and \( Y \).

Reinforcements received, \( r_{t+1} \)'s, will be collected as rows of \( R \).
Here are the gradient descent updates we had for neural networks

\[ Z = h(\tilde{X}V) \]
\[ Y = \tilde{Z}W \]
\[ V \leftarrow V + \rho_h \frac{1}{N} \frac{1}{K} \tilde{X}^T \left( (T - Y)\hat{W}^T \cdot (1 - Z^2) \right) \]
\[ W \leftarrow W + \rho_o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y) \]

The key change we need to make is in the error. Our targets are now given by a reinforcement plus the next step predicted Q values. Must assemble a bunch of samples into matrices.

Multiple \( s_t \)'s will be collected as rows in \( X \), which, when passed through the neural network, produce \( Z \) and \( Y \).

Reinforcements received, \( r_{t+1} \)'s, will be collected as rows of \( R \).

Now just need \( Q(s_{t+1}, a_{t+1}) \)'s. These are really like rows of \( Y \) shifted up by one row. But need a way to designate which actions were chosen each step.
Can make use of the same neural network gradient descent updates if we structure our error matrix correctly.
Can make use of the same neural network gradient descent updates if we structure our error matrix correctly.

Define the temporal difference error to be \((R + Q - Y)\) where \(Q\) represents the \(Q(s_{t+1}, a_{t+1})\)'s. We want this to produce zero's all but the actions chosen at each step.

\[ R = \begin{bmatrix} 0 & 0 & r_1 & 0 & 0 & r_2 & 0 & 0 & r_3 & 0 & 0 & r_4 \end{bmatrix} \]

\(R\) is \(N\times K\).
Can make use of the same neural network gradient descent updates if we structure our error matrix correctly.

Define the temporal difference error to be \((R + Q - Y)\) where \(Q\) represents the \(Q(s_{t+1}, a_{t+1})\)'s. We want this to produce zero's all but the actions chosen at each step.

Make all matrices have \(K\) columns, but so that the total error is zero for actions not taken.
- Can make use of the same neural network gradient descent updates if we structure our error matrix correctly.
- Define the temporal difference error to be \((R + Q - Y)\) where \(Q\) represents the \(Q(s_{t+1}, a_{t+1})\)'s. We want this to produce zero's all but the actions chosen at each step.
- Make all matrices have \(K\) columns, but so that the total error is zero for actions not taken.
- Start with \(R\). Say we have three actions, 1, 2, and 3. For four samples at times \(t = 1, 2, 3, 4\), we see actions \(a_1 = 3, a_2 = 1, a_3 = 2,\) and \(a_4 = 1\) and we receive reinforcements \(r_1, r_2, r_3,\) and \(r_4\). Build \(R\) to look like

\[
R = \begin{bmatrix}
0 & 0 & r_1 \\
0 & r_2 & 0 \\
r_2 & 0 & 0 \\
0 & 0 & r_3 \\
r_3 & 0 & 0 \\
r_4 & 0 & 0
\end{bmatrix}
\]

\(R\) is \(N \times K\).
We can build \( \mathbf{R} \) by first making a matrix \( \mathbf{A} \) designating actions selected

\[
\mathbf{A} = \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{bmatrix}
\]

and a column matrix of reinforcements \( \mathbf{r} \). Let’s call it \( \mathbf{\bar{R}} \), because the accent looks like half of a zero.
We can build $\mathbf{R}$ by first making a matrix $\mathbf{A}$ designating actions selected

$$
\mathbf{A} = \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
$$

and a column matrix of reinforcements $\mathbf{r}$. Let’s call it $\tilde{\mathbf{R}}$, because the accent looks like half of a zero.

Then

$$
\tilde{\mathbf{R}} = \mathbf{r}^T \mathbf{A}
$$
We can build $\mathbf{R}$ by first making a matrix $\mathbf{A}$ designating actions selected

$$
\mathbf{A} = \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
$$

and a column matrix of reinforcements $\mathbf{r}$. Let’s call it $\check{\mathbf{R}}$, because the accent looks like half of a zero.

Then

$$
\check{\mathbf{R}} = \mathbf{r}^T \mathbf{A}
$$

We can use $\mathbf{A}$ to build similar matrices for $\mathbf{Q}$ and $\mathbf{Y}$.

$$
\check{\mathbf{Y}} = \mathbf{Y} \cdot \mathbf{A}
$$

$$
\check{\mathbf{Q}} = \mathbf{Q} \cdot \mathbf{A}
$$
Finally, we can rewrite the gradient descent updates. Here are the original ones again.

\[ Z = h(\tilde{X}V) \]
\[ Y = \tilde{Z}W \]
\[ V \leftarrow V + \rho_h \frac{1}{N} \frac{1}{K} \tilde{X}^T \left( (T - Y)\hat{W}^T \cdot (1 - Z^2) \right) \]
\[ W \leftarrow W + \rho_o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y) \]
Finally, we can rewrite the gradient descent updates. Here are the original ones again.

\[ Z = h(\tilde{X}V) \]
\[ Y = \tilde{Z}W \]
\[ V \leftarrow V + \rho_h \frac{1}{N} \frac{1}{K} \tilde{X}^T \left( (T - Y)\hat{W}^T \cdot (1 - Z^2) \right) \]
\[ W \leftarrow W + \rho_o \frac{1}{N} \frac{1}{K} \tilde{Z}^T (T - Y) \]

We change the updates to

\[ V \leftarrow V + \rho_h \frac{1}{N} \tilde{X}^T \left( (\check{R} + \check{Q} - \check{Y})\hat{W}^T \cdot (1 - Z^2) \right) \]
\[ W \leftarrow W + \rho_o \frac{1}{N} \tilde{Z}^T (\check{R} + \check{Q} - \check{Y}) \]
Must do more than one batch update

- These are batch updates, remember? Before, we had all input and target samples in $X$ and $T$. 
Must do more than one batch update

- These are batch updates, remember? Before, we had all input and target samples in $X$ and $T$.
- Now one set of samples are collected for one set of weight values, meaning for one Q function.
Must do more than one batch update

- These are batch updates, remember? Before, we had all input and target samples in \( X \) and \( T \).
- Now one set of samples are collected for one set of weight values, meaning for one Q function.
- After updates, we have new Q function, which will produce different actions. So?
Must do more than one batch update

- These are batch updates, remember? Before, we had all input and target samples in $X$ and $T$.
- Now one set of samples are collected for one set of weight values, meaning for one Q function.
- After updates, we have new Q function, which will produce different actions. So?
- Must repeat these steps
Must do more than one batch update

- These are batch updates, remember? Before, we had all input and target samples in $X$ and $T$.
- Now one set of samples are collected for one set of weight values, meaning for one $Q$ function.
- After updates, we have new $Q$ function, which will produce different actions. So?
- Must repeat these steps
  - Generate $N$ samples.
Must do more than one batch update

- These are batch updates, remember? Before, we had all input and target samples in \( X \) and \( T \).
- Now one set of samples are collected for one set of weight values, meaning for one Q function.
- After updates, we have new Q function, which will produce different actions. So?
- Must repeat these steps
  - Generate \( N \) samples.
  - Update \( V \) and \( W \).
Outline

Mean Squared Error Between Return and Model Output
Reinforcement Learning Objective
Gradient of Objective

Gradient for Q function as Table

Gradient for Q function as Neural Network

Training the Q Neural Network
Training the Q Neural Network

- Now let’s see if it works......