#ml-papers November 2019

Learning to Predict by the Methods of Temporal Differences

Section 1: Introduction - What are TD Methods?

Temporal difference methods are driven by a **change in prediction**

Conventional prediction learning is driven by the **difference between the prediction and actuals**.

Advantages of TD over conventional methods

- Easier to compute (more incremental)
- More efficient use of experience (converge faster / better predictions in the same time)

Examples:

- Weatherman who updates daily predictions
- Checkers playing program (Samuel 1959)

> Samuel implemented what is now called **alpha-beta pruning**.
>
> Instead of searching each path until it came to the game's conclusion, Samuel developed a scoring function based on the position of the board at any given time.
>
> This function tried to measure the chance of winning for each side at the given position.

Section 2

Single Step vs. Multi Step Predictions

- In a single step prediction, TD and prediction learning methods are the same.
- This paper will focus only on Multi Step Predictions

- **Single Step Prediction:** Number of subscribers tomorrow, given today's information

  Multi Step Prediction: Forecasted Number of subscribers at end of month, updated daily as new information arises.

Relating TD to Gradient Descent (Widrow-Hoff rule)

- **Proof:** TD produces the same weight changes as Gradient Descent

  $x_1, x_2, x_3, \ldots, x_m, z,$
  observations x, with prediction target z

Traditional Learning

$$\Delta w_t = \alpha(z - w^T x_t)x_t.$$

Learning Rate · error · observation vector

Temporal Difference Learning

$$z - P_t = \sum_{k=t}^{m}(P_{k+1} - P_k) \quad \text{where} \quad P_{m+1} \stackrel{\text{def}}{=} z.$$

error · sum of differences in predictions

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t.$$

$$\Delta w_t = \alpha(z - P_t)\nabla_w P_t,$$

$$w \leftarrow w + \sum_{t=1}^{m} \alpha(z - P_t)\nabla_w P_t$$

$$w + \sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k.$$

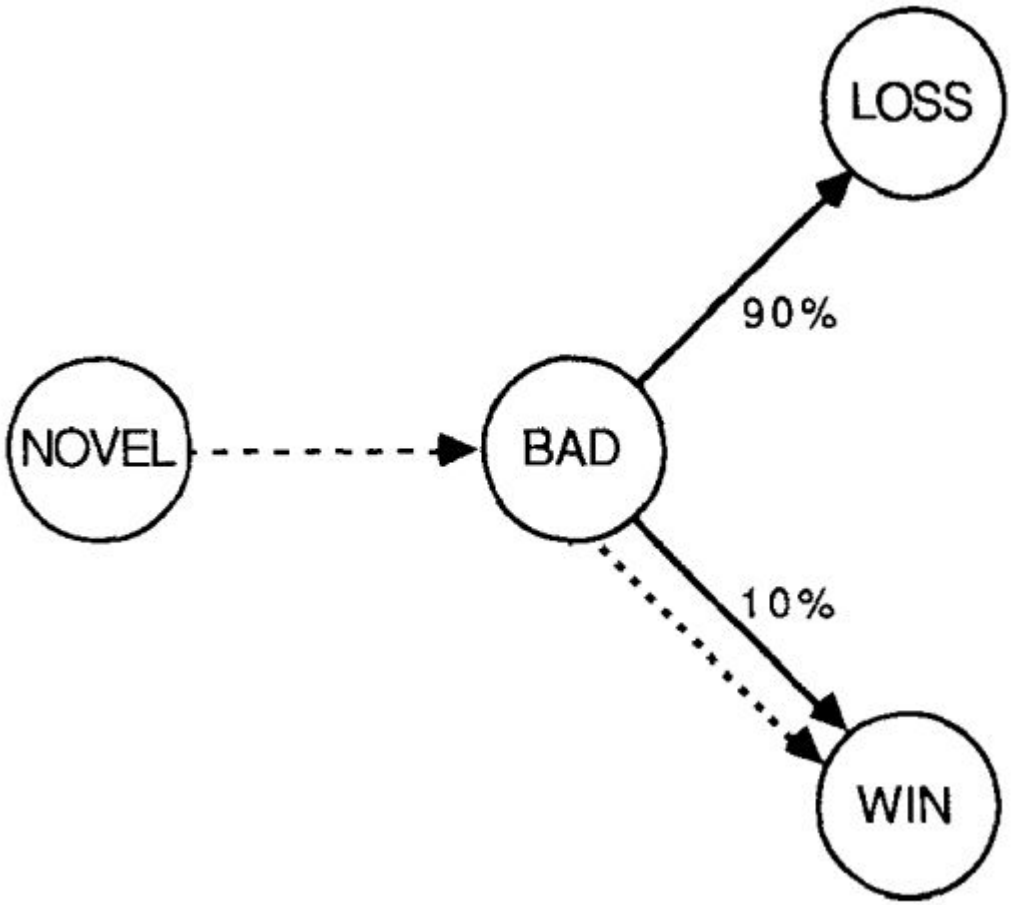
$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k.$$

Section 3 - Examples of faster learning with TD methods

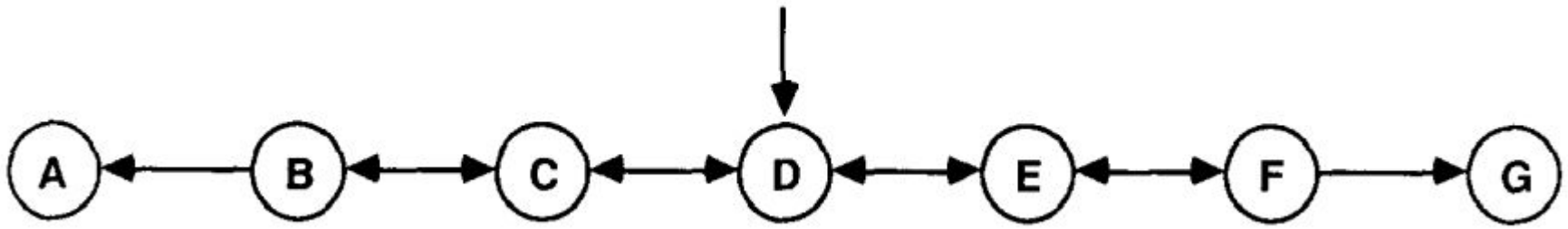
Game play thought experiment:

- circles / states represent positions in a 2 player board game
- we know with high confidence that the "bad" position leads to game loss 90% of the time from an abundance of past experience
- The first time we encounter the "novel" position it evolves through the dotted line (leads to "bad" and then to "win")

What evaluation (likelihood of winning) should we give the novel position?



Section 3 - Simulation Experiments



The next experiments all use the same simulated dataset, 100 train sets of 10 episodes of bounded random walk.

In each experiment we will be predicting the likelihood of ending in state **G** starting at the current state.

```
# single example
train_sets[0][0]
```

Out[1]:
```
[array([0, 0, 1, 0, 0]),
 array([0, 1, 0, 0, 0]),
 array([1, 0, 0, 0, 0]),
 array([0, 1, 0, 0, 0]),
 array([1, 0, 0, 0, 0]),
 0]
```

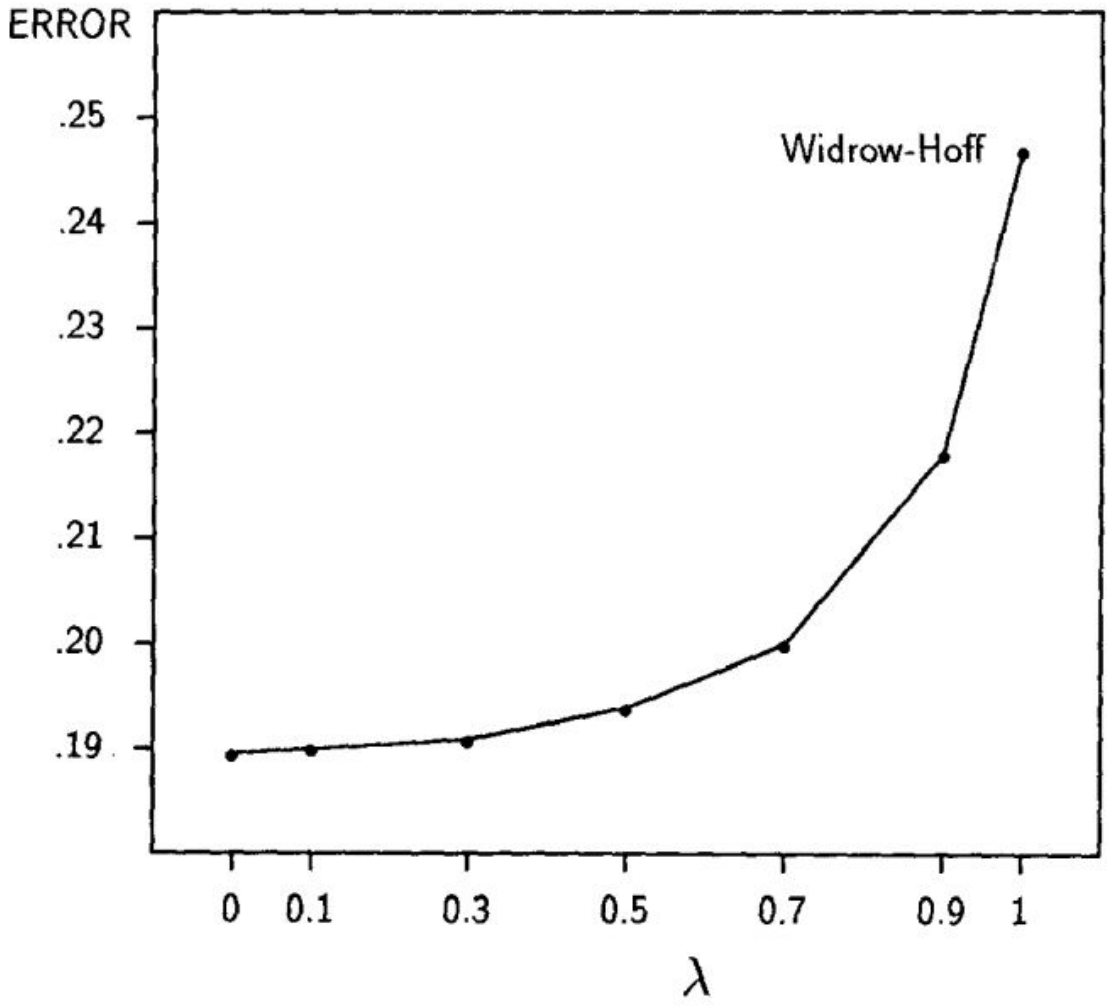
Section 3 - Simulation Experiment 1

First experiment:

- we show each train set (of 100) until we reach weight convergence (no definition here), once the weights converge we evaluate train set RMSE against the label and move on to the next train set.
- the models we are evaluating are [0.0 - 1.0] range of λ values combined with a non-specified range of alpha. we pick the best RMSE for each λ.

What do these results prove?

(btw this is the figure from the erratum, the original figure is wrong)



Section 3 - Simulation Experiment 1

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k. \quad \text{where} \quad P_{m+1} \stackrel{\text{def}}{=} z.$$

```
# compute next weight update
for train_example_i, train_example in enumerate(train_set):
    # compute predictions for each time step of this training example & append

    predictions = [np.dot(weights.T, state) for state in train_example[:-1]]
    predictions.append(train_example[-1])

    # compute weight updates for each time step of this episode
    this_train_example_delta_weights = []
    for t, state in enumerate(train_example[:-1]):
        td_error = alpha * (predictions[t+1] - predictions[t])
        td_gradients = np.zeros(5)
        for k in range(t + 1):
            td_gradients += train_example[k] * pow(lambda_, t - k)

        this_train_example_delta_weights.append(td_error * td_gradients)

    for e in this_train_example_delta_weights:
        accumulated_delta_weights += e
```

key points:

- lambda is < 1 and taken to the power of (# of time steps behind current step), so it decays the gradients further back you go
- derivative of P_k w.r.t w = X_t, which is a 1 hot vec
- the last time step prediction updates using label error

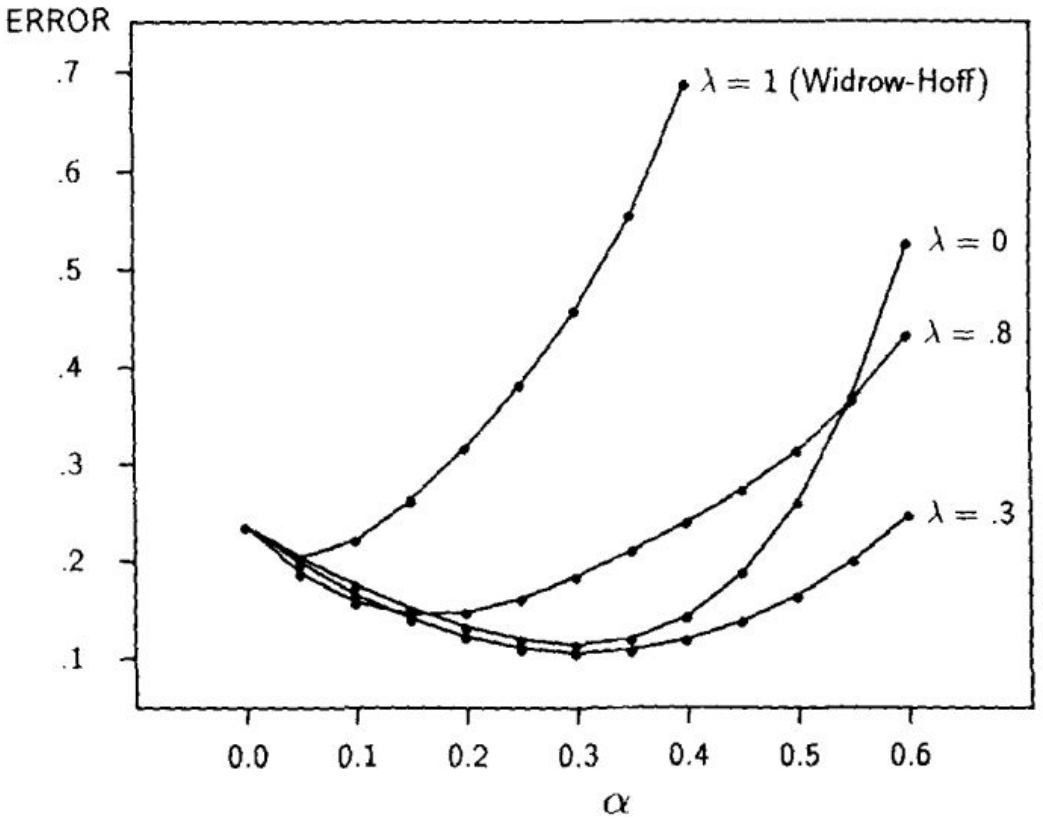
source

Section 3 - Simulation Experiment 2

Second experiment:

- we show each train set (of 100) once then evaluate train set RMSE against the label. **We reset learned weights to .5 before moving to next set.**
- the models we are evaluating are [0.0 - 1.0] range of λ values combined with alpha range [0.0 - 0.6].

What do these results prove?

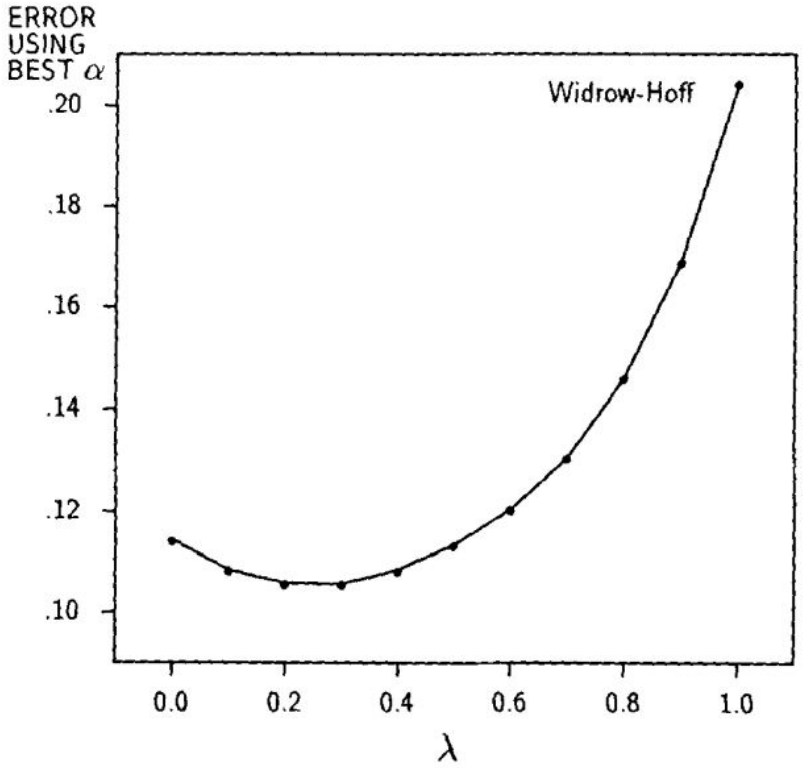


Section 3 - Simulation Experiment 3

Third experiment:

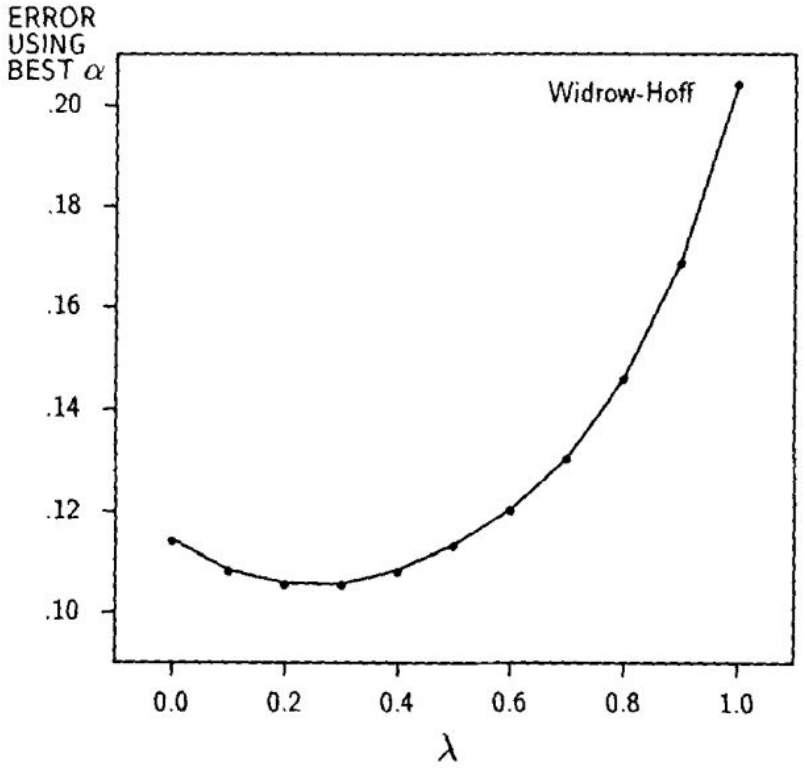
- Same setup as #2: we show each train set (of 100) once then evaluate train set RMSE against the label. **We reset learned weights to .5 before moving to next set.**
- the models we are evaluating are [0.0 - 1.0] range of λ values combined with an unspecified alpha range, this time we only plot the best alpha score @ each λ

What do these results prove?



Section 3 - Simulation Experiment 3

"One reason lambda = 0 is not optimal for this problem is that TD(0) is relatively slow at propagating prediction levels back along a sequence. For example, suppose states D, E, and F all start with the prediction value 0.5, and the sequence xD, XE, XF, 1 is experienced. TD(0) will change only F's prediction, whereas the other procedures will also change E's and D's"



Section 4.1

- Markov absorbing processes will converge to correct values with sufficient learning
  - "Markov" means that the subsequent state is only determined by current state and we have transition probabilities between these states
  - "Absorbing" means that cycling indefinitely through non terminal states is impossible
  - All together, we want to show formally that this type of dynamic comes closer and closer to the expected value of the terminal state; OR that there exists a limit for:

$$E\{z \mid i\} = \sum_{j \in T} p_{ij} \bar{z}_j + \sum_{j \in N} p_{ij} \sum_{k \in T} p_{jk} \bar{z}_k + \sum_{j \in N} p_{ij} \sum_{k \in N} p_{jk} \sum_{l \in T} p_{kl} \bar{z}_l + \cdots$$

for all transition probabilities and states

- Linear TD(0) updates weights (with a linear relationship) at each step:
- So, we need to show that this additional element still converges:

$$\begin{aligned} E\{w_{n+1} \mid w_n\} = w_n \quad &+ \sum_{i \in N} \sum_{j \in N} d_i p_{ij} \alpha \left(w_n^T \mathbf{x}_j - w_n^T \mathbf{x}_i\right) \mathbf{x}_i \\ &+ \sum_{i \in N} \sum_{j \in T} d_i p_{ij} \alpha \left(\bar{z}_j - w_n^T \mathbf{x}_i\right) \mathbf{x}_i, \end{aligned}$$

$$\begin{aligned} w_{n+1} &= w_n + \sum_{t=1}^{m} \alpha (P_{t+1} - P_t) \nabla_w P_t \qquad \text{where} \qquad P_{m+1} \stackrel{\text{def}}{=} z \\ &= w_n + \sum_{t=1}^{m-1} \alpha (P_{t+1} - P_t) \nabla_w P_t + \alpha (z - P_m) \nabla_w P_m \\ &= w_n + \sum_{t=1}^{m-1} \alpha (w_n^T \mathbf{x}_{q_{t+1}} - w_n^T \mathbf{x}_{q_t}) \mathbf{x}_{q_t} + \alpha (z - w_n^T \mathbf{x}_{q_m}) \mathbf{x}_{q_m}, \end{aligned}$$

(Optional slide) section 4.1

$\bar{w}_n$, is the expected value of the terminal state. Then since there is a linear dependency between states, we can rewrite it as:

$$\bar{w}_{n+1} = \bar{w}_n + \sum_{i \in N} \sum_{j \in N} d_i p_{ij} \alpha \left(\bar{z}_j - \bar{w}_n^T \mathbf{x}_i \right) \mathbf{x}_i + \sum_{i \in N} \sum_{j \in T} d_i p_{ij} \alpha \left(\bar{z}_j - \bar{w}_n^T \mathbf{x}_i \right) \mathbf{x}_i .$$

Learning to predict by the methods of temporal differences

(Optional Slide) Section 4.1

$$
\begin{aligned}
\bar{w}_{n+1} &= \bar{w}_n + \alpha \sum_{i \in N} d_i \mathbf{x}_i \left(\sum_{j \in T} p_{ij} \bar{z}_j + \sum_{j \in N} p_{ij} \bar{w}_n^T \mathbf{x}_j - \bar{w}_n^T \mathbf{x}_i \sum_{j \in N \cup T} p_{ij} \right) \\
&= \bar{w}_n + \alpha \sum_{i \in N} d_i \mathbf{x}_i \left([h]_i + \sum_{j \in N} p_{ij} \bar{w}_n^T \mathbf{x}_j - \bar{w}_n^T \mathbf{x}_i \right) \\
&= \bar{w}_n + \alpha X D \left(h + Q X^T \bar{w}_n - X^T \bar{w}_n \right);
\end{aligned}
$$

$$
\begin{aligned}
X^T \bar{w}_{n+1} &= X^T \bar{w}_n + \alpha X^T X D \left(h + Q X^T \bar{w}_n - X^T \bar{w}_n \right) \\
&= \alpha X^T X D h + (I - \alpha X^T X D (I - Q)) X^T \bar{w}_n \\
&= \alpha X^T X D h + (I - \alpha X^T X D (I - Q)) \alpha X^T X D h \\
&\qquad + (I - \alpha X^T X D (I - Q))^2 X^T \bar{w}_{n-1} \\
&\vdots \\
&= \sum_{k=0}^{n-1} (I - \alpha X^T X D (I - Q))^k \alpha X^T X D h \\
&\qquad + (I - \alpha X^T X D (I - Q))^n X^T w_0.
\end{aligned}
$$

$$\begin{aligned}
\lim_{n\to\infty} X^T \bar{w}_n &= \left(I - (I - \alpha X^T X D(I - Q))\right)^{-1} \alpha X^T X D h \\
&= (I - Q)^{-1} D^{-1} (X^T X)^{-1} \alpha^{-1} \alpha X^T X D h \\
&= (I - Q)^{-1} h;
\end{aligned}$$

$$\lim_{n\to\infty} E\left\{\mathbf{x}_i^T w_n\right\} = \left[(I - Q)^{-1} h\right]_i \qquad \forall i \in N,$$

(Optional Slide) Section 4.1

We still have to show that the limit of the last term in the last slide = 0. We do this by first showing that D(I-Q) is positive definite, and then that XTXD (I- Q) has a full set of eigenvalues all of whose real parts are positive. This will enable us to show that a can be chosen such that all eigenvalues of I-c~XTXD(I-Q) are less than 1 in modulus, which assures us that its powers converge.

$$S = D(I - Q) + (D(I - Q))^T$$

$$\begin{aligned}
\sum_j [S]_{ij} &= \sum_j \left([D(I-Q)]_{ij} + [(D(I-Q))^T]_{ij} \right) \\
&= \sum_j d_i [I-Q]_{ij} + \sum_j d_j [I-Q]_{ji} \\
&= d_i \sum_j [I-Q]_{ij} + [d^T(I-Q)]_i \\
&= d_i (1 - \sum_j p_{ij}) + [\mu^T (I-Q)^{-1}(I-Q)]_i \qquad \text{by (7)} \\
&= d_i (1 - \sum_j p_{ij}) + \mu_i \\
&\geq 0.
\end{aligned}$$

(Optional Slide) Slide 4.1

$$y^* D(I-Q)y = z^* X^T X D(I-Q)y = z^* \lambda y = \lambda z^* X^T X z = \lambda (Xz)^* Xz,$$

where "$*$" denotes the conjugate-transpose. This implies that

$$\mathrm{Re}\left(y^* D(I-Q)y\right) = \mathrm{Re}\left(\lambda (Xz)^* Xz\right);$$

$$a^T D(I-Q)a + b^T D(I-Q)b = (Xz)^* Xz \ \mathrm{Re}\ \lambda. \qquad \leftarrow \text{Both sides must be positive}$$

$$\begin{aligned} |1-\alpha\lambda| &= \sqrt{(1-\alpha a)^2 + (-\alpha b)^2} \\ &= \sqrt{1 - 2\alpha a + \alpha^2 a^2 + \alpha^2 b^2} \\ &= \sqrt{1 - 2\alpha a + \alpha^2 (a^2 + b^2)} \\ &< \sqrt{1 - 2\alpha a + \alpha \frac{2a}{a^2+b^2}(a^2+b^2)} = \sqrt{1 - 2\alpha a + 2\alpha a} = 1. \end{aligned}$$

← So we can find the eigenvalues this way. We need to show that we can find some that are less than 1 modulus. This exercise is actually simple. Pick alpha such that:

$$0 < \alpha < \frac{2a}{a^2+b^2},$$

Section 5
extension of TD method

1. Predict accumulated cost over the sequence

when each step has corresponding gain / cost and we want to predict net gain/lost (= preceding cost + remaining cost)

$$z_t - P_t = \sum_{k=t}^{m} c_{k+1} - P_t = \sum_{k=t}^{m} (c_{k+1} + P_{k+1} - P_k)$$

where $z_t = \sum_{k=t}^{m} c_{k+1}$: cumulative remaining cost

$-P_t = \sum_{k=1}^{m} (P_{k+1} - P_k)$: cumulative expected cost at each step

cumulative TD(λ) family:

$$\Delta w_t = \alpha(c_{t+1} + P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

Section 5
extension of TD method

2. Intra-sequence weight updating

- Previous TD method assumes w is updated only once for each complete output sequence thus does not change during a sequence;

- Extension:
instead of observation of a complete sequence, it's simpler to update the weight after each observation: w changes for different t (weight alternating)

$$w_{t+1} = w_t + \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k, \quad \text{where} \quad P_t \stackrel{\text{def}}{=} P(x_t, w_{t-1})$$

$$w_{t+1} = w_t + \alpha\Big(P(x_{t+1}, w_t) - P(x_t, w_t)\Big) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P(x_k, w_t)$$

Section 5
extension of TD method

3. Prediction by a fixed interval

Consider making a prediction for particular fixed window (Mon->Mon, Tue->Tue, etc.)
-- can't apply TD method because each prediction is a different event

- Extension:
at each day t, we form P_t^δ is an estimate of the probability of rain δ days later

this provides overlapping sequences of inter-related predictions
($\because P_t^\delta = P_{t+1}^{\delta-1}$, $\therefore P_t^7, P_{t+1}^6, P_{t+2}^5, \ldots, P_{t+6}^1$, all of the same event)

$$\Delta w^\delta = \alpha(P_{t+1}^{\delta-1} - P_t^\delta) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k^\delta$$

e.g. on Monday, make prediction for Tue, Wed, …, Mon;
update weight for Tue based on the difference of today's vs. yesterday's prediction