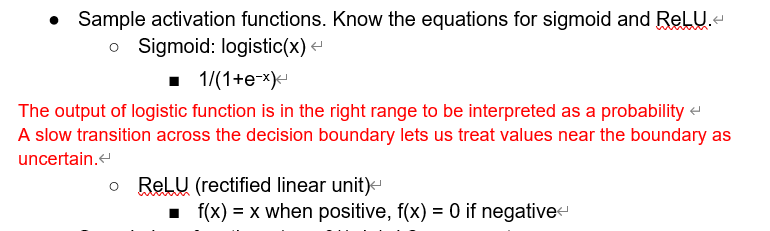
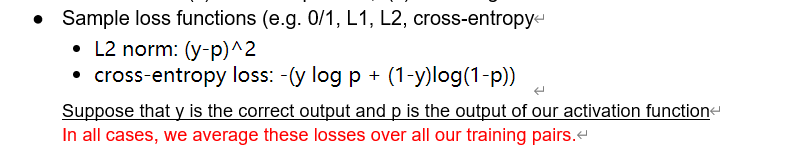
**Linear Classifiers**

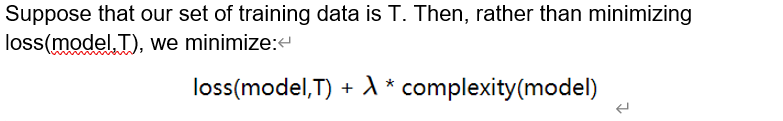




The high-level idea of cross-entropy is that we have built a model of probabilities (e.g. using some training data) and then use that model to compress a different set of data. Cross-entropy measures how well our model succeeds on this new data. This is one way to measure the difference between the model and the new data.

Update rule: 

Regularization: overcome overfitting

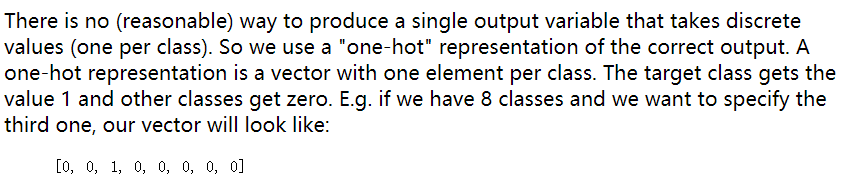


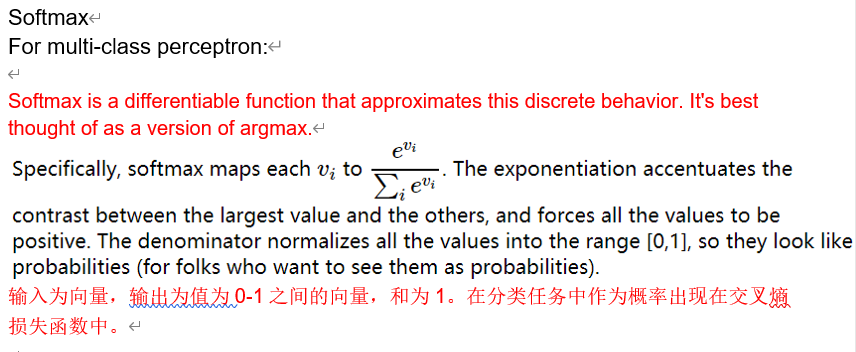
Small lambda means focus on the data: prone to overfitting

Large lambda means not: not fitting

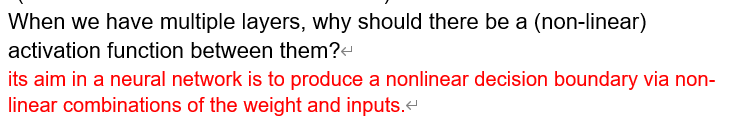
Use to measure the complexity of the model, pay attention to the wrong input, so make the classifier less likely to use them

One-hot representation: For multi-class perceptron, one element per class

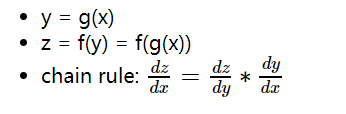




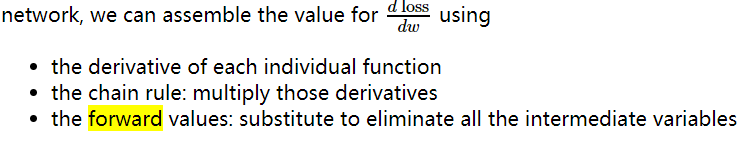
**Neural Nets**

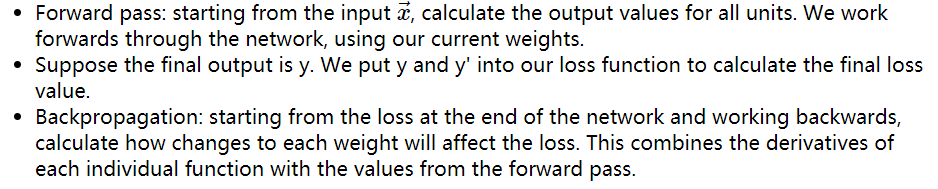


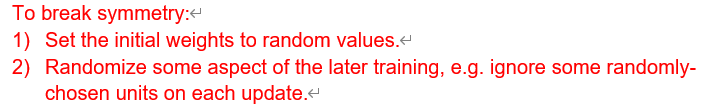
**Chain rule:**

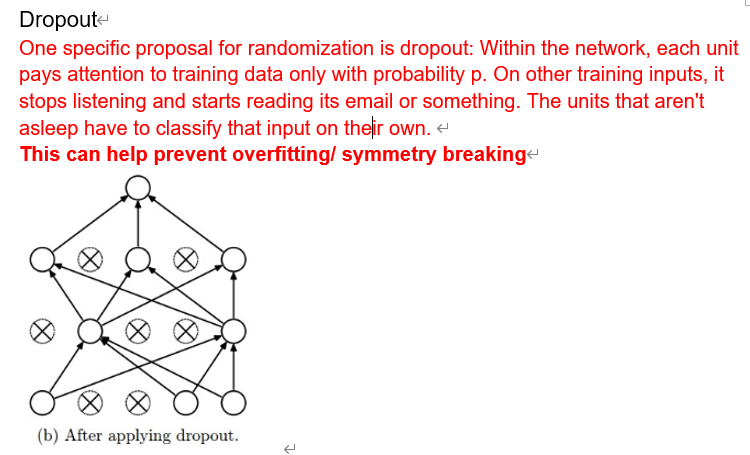


**Forward value**





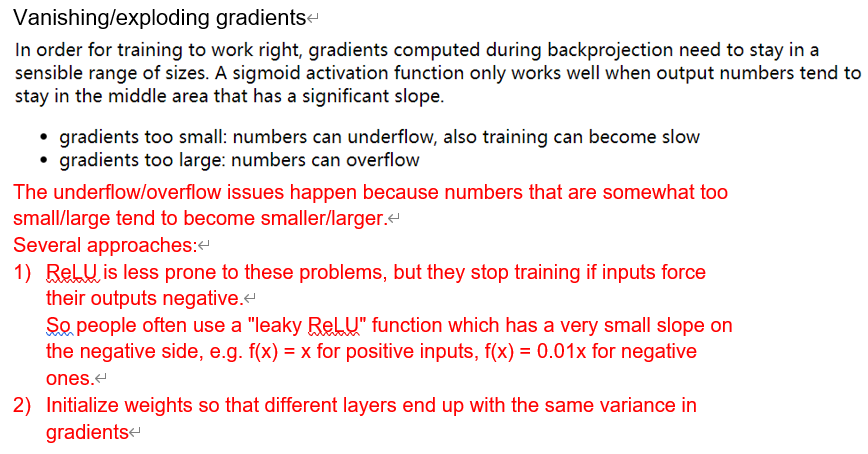


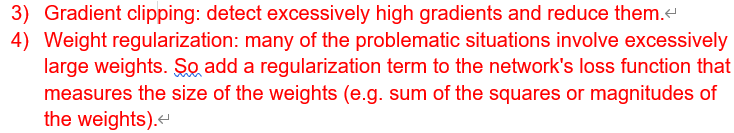


**Overfittings**:

The **dropout** technique will reduce this problem. Another method is Data augmentation.

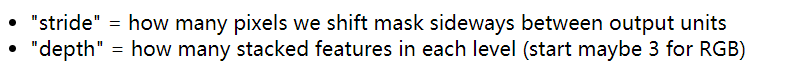
**Data augmentation** tackles the fact that **training data is always very sparse**, but we have additional domain knowledge that can **help fill in the gaps.** We can **make more training examples by perturbing existing ones** in ways that shouldn't (ideally) change the network's output. For example, if you have one picture of a cat, **make more by translating or rotating the cat.**



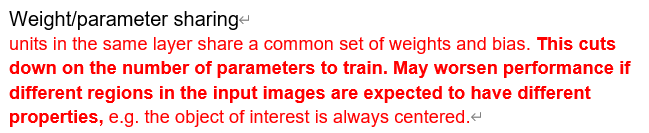


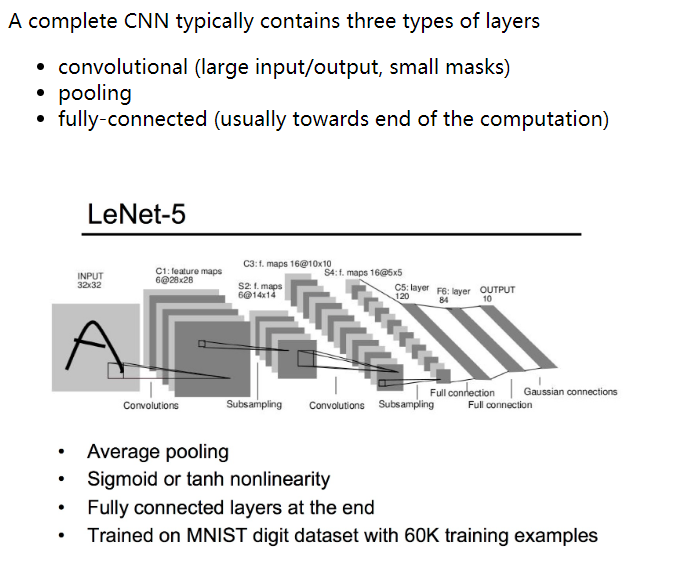
**How does convolution layer work:**

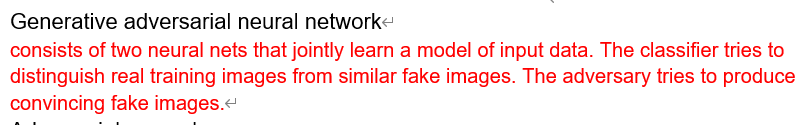
1. each unit computes a weighted sum of the values in that local region. In signal processing, this is known as "convolution" and the set of weights is known as a "mask."
2. each network layer has a significant thickness, i.e. a number of different values at each (x,y) location.











it is possible to cook up patterns that are fairly close to random noise but push the network's values towards or away from a particular output classification.

Recurrent neural networks:

are neural nets that have connections that loop back from a layer to the same layer.

The intent of the feedback loop in the picture is that each unit is connected to **all the other** units in the layer.

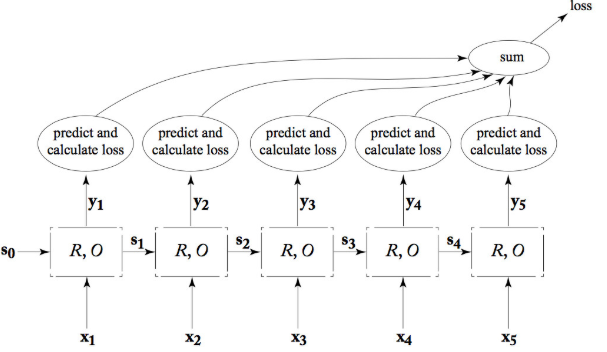
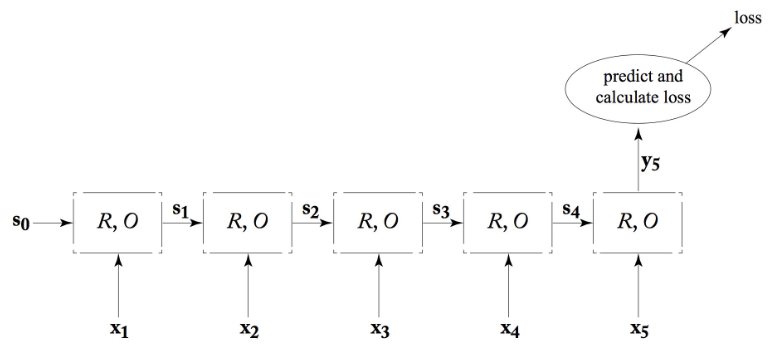
When unroll, all copies of the unit share the same parameter values.

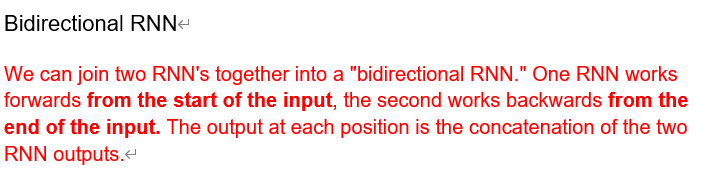
An RNN can be used as a classifier. That is, the system using the RNN only cares about the output from the last timestep

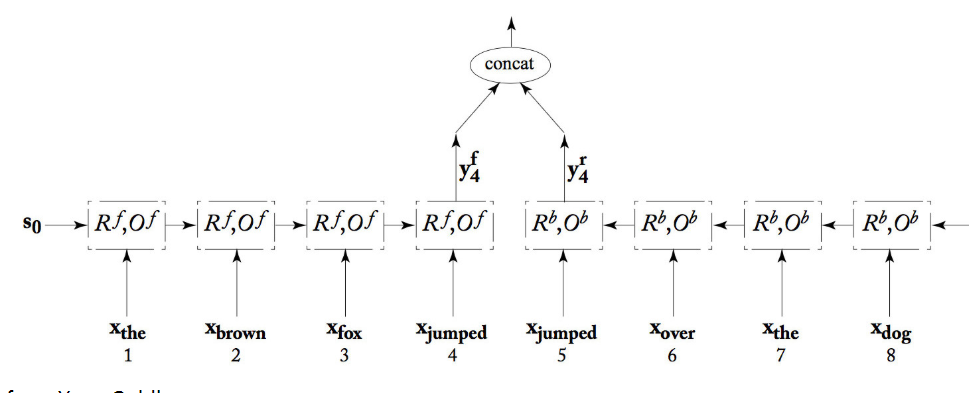
When would we compute loss from last unit vs. summed over all units?

When the RNN only cares about the output from the last timestep: we compute loss from the last unit.

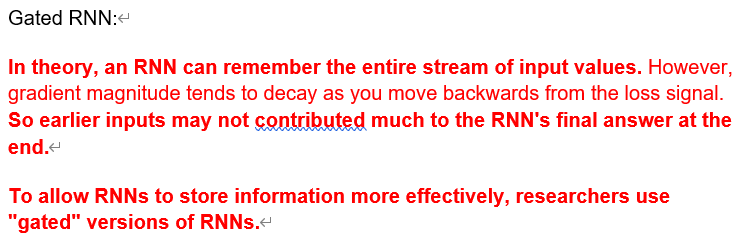
When care about the correctness at each timestep, we sum.



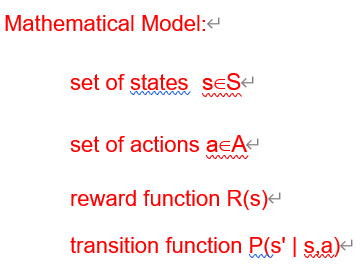




"deep" RNN's, which have more than one processing layer between the input and output streams



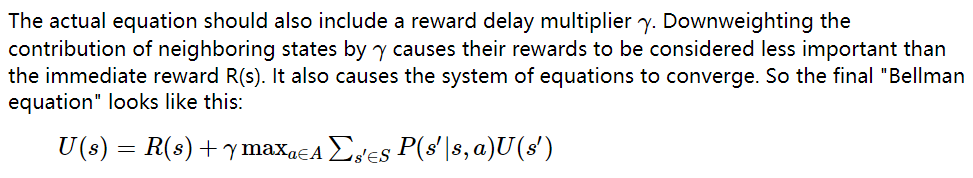
**Markov Decision Processes and Reinforcement Learning**



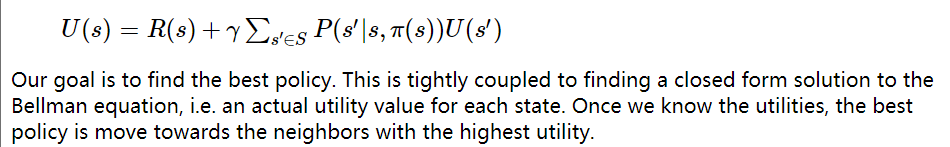
**we make the assumption that rewards are better if they occur sooner.**

That is, being near a big reward is almost as good as being at the big reward.

Bellman equation **Optimal Policy**

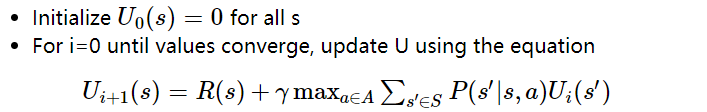


Bellman equation **Fixed Policy**

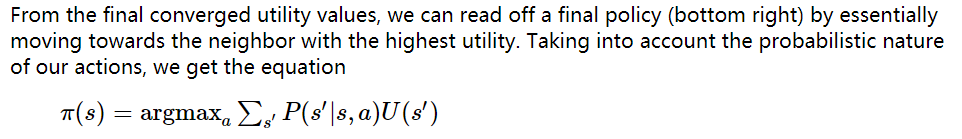


● Methods of solving the Bellman equation

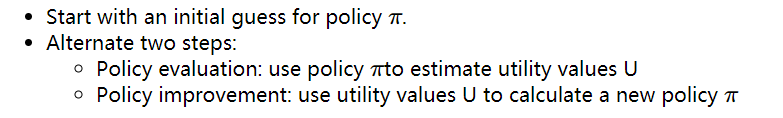
○ **Value iteration:** repeatedly applies the Bellman equation to update utility values for each state.



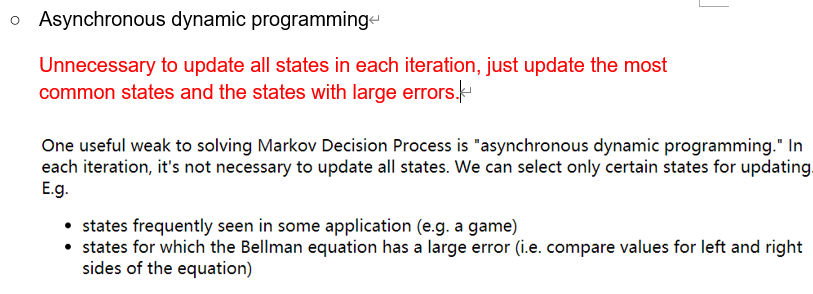
Until converged

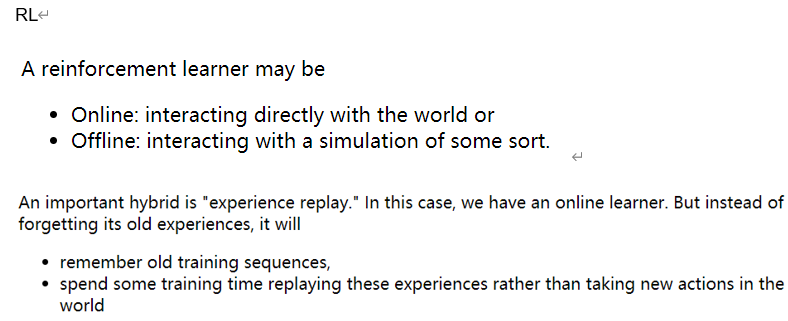


○ Policy iteration: Policy iteration produces the same solution, but faster.

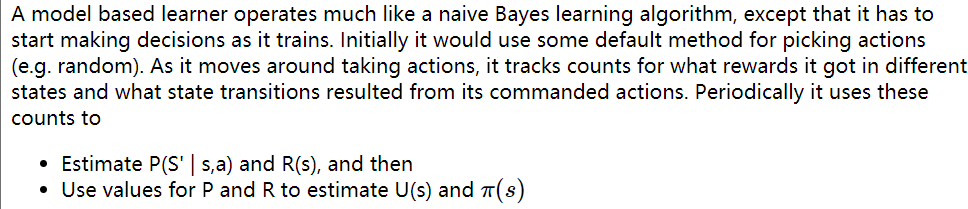


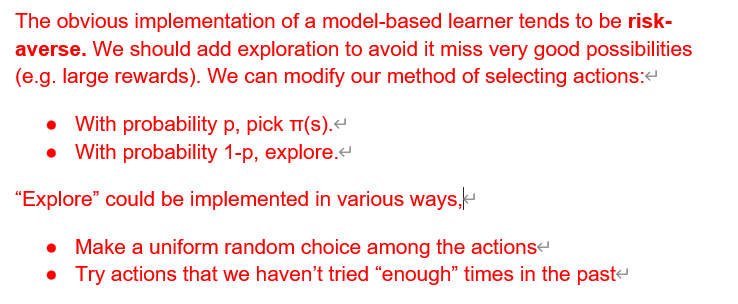
We don’t need a fully converged solution because we’ll refine policy in each iteration.





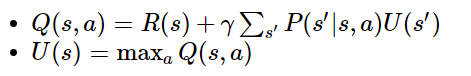
Model based RL:





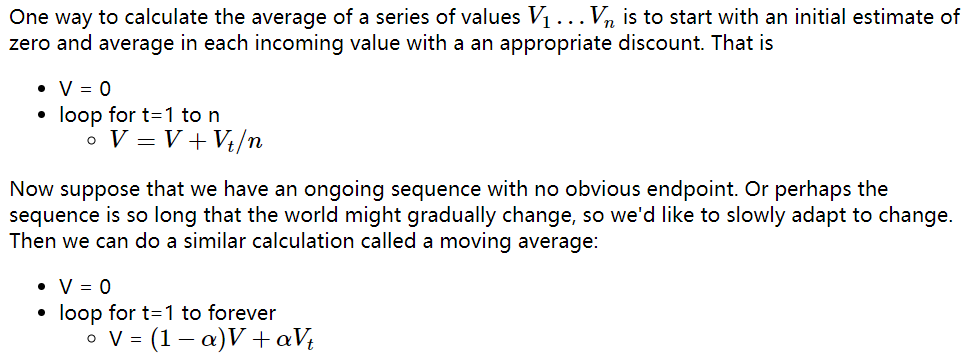
Model free RL (Q-learning)

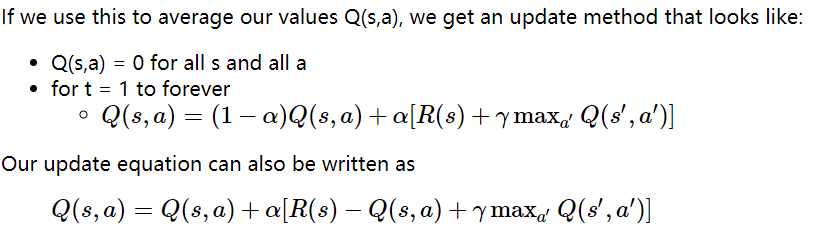
**Q(s,a) tells us the value of commanding action a when the agent is in state s.**



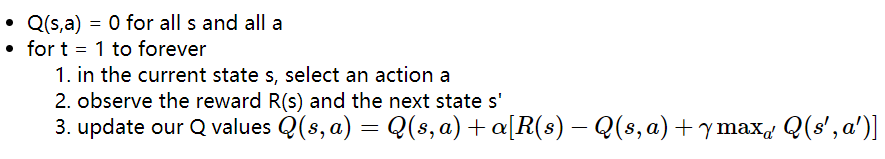


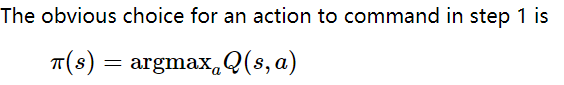
TD update:

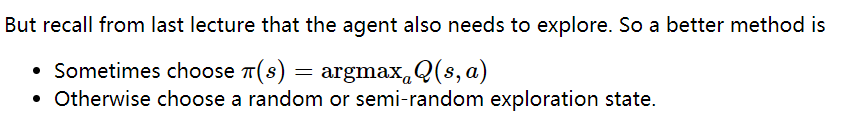




Alpha is because there is no obvious endpoint/ world may change



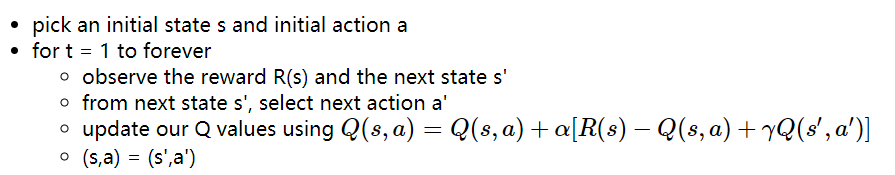


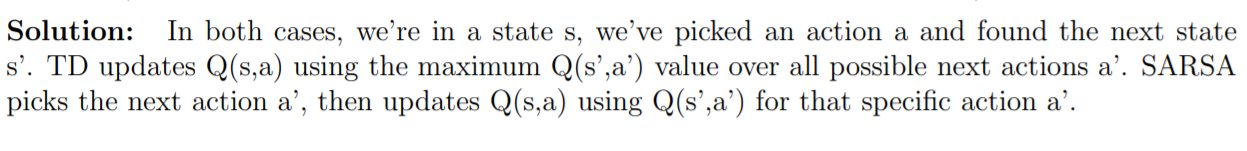


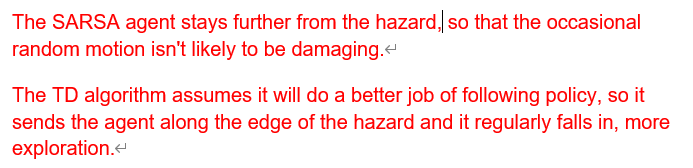
**Create inconsistency!**

the update in step 3 is based on a different action from the one actually chosen by the agent.

SARSA update algorithm: Adjusts TD update algorithm to align the update with the actual choice of action







**TF-IDF: word 2 document**

TF: normalized word count

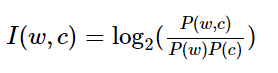
**TF = 1+log10(c)**, c is the count of the word

IDF: inversed document frequency

**IDF = log10(1+N/df)** N is # of documents, df is # of documents where word occurs

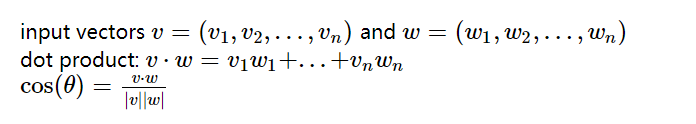
**TF-IDF = TF\*IDF**

**PMI: word 2 word**



**PPMI:**





When building vector space, try to Maximize !!!!dot product!!!!

