NeoRL – Overview

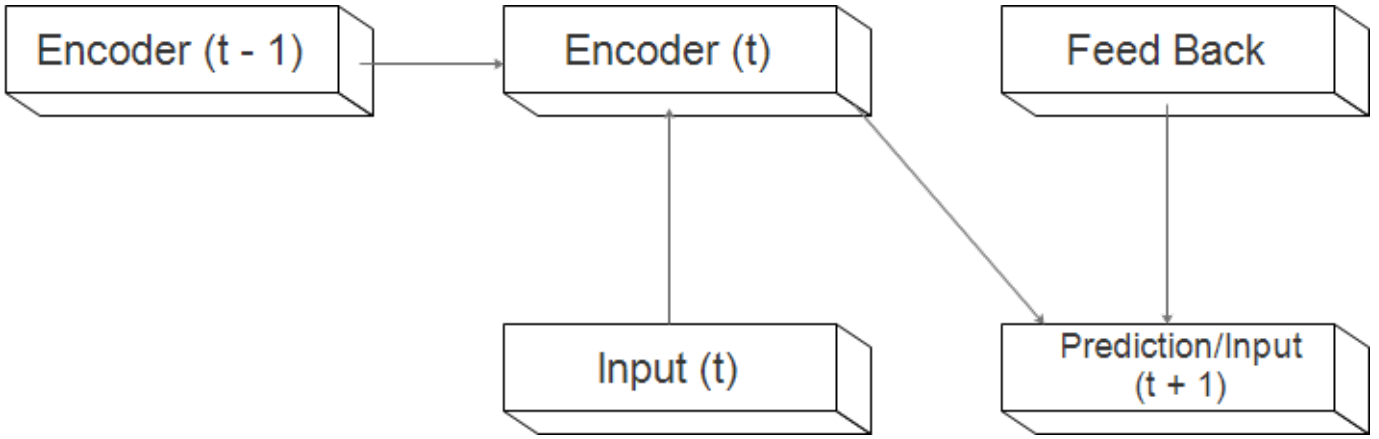
CireNeikual

What is NeoRL?

- Algorithmic neocortex-like simulation
- Online predictor of temporal data streams
- Reinforcement learner
- Unsupervised spatio-temporal feature extractor

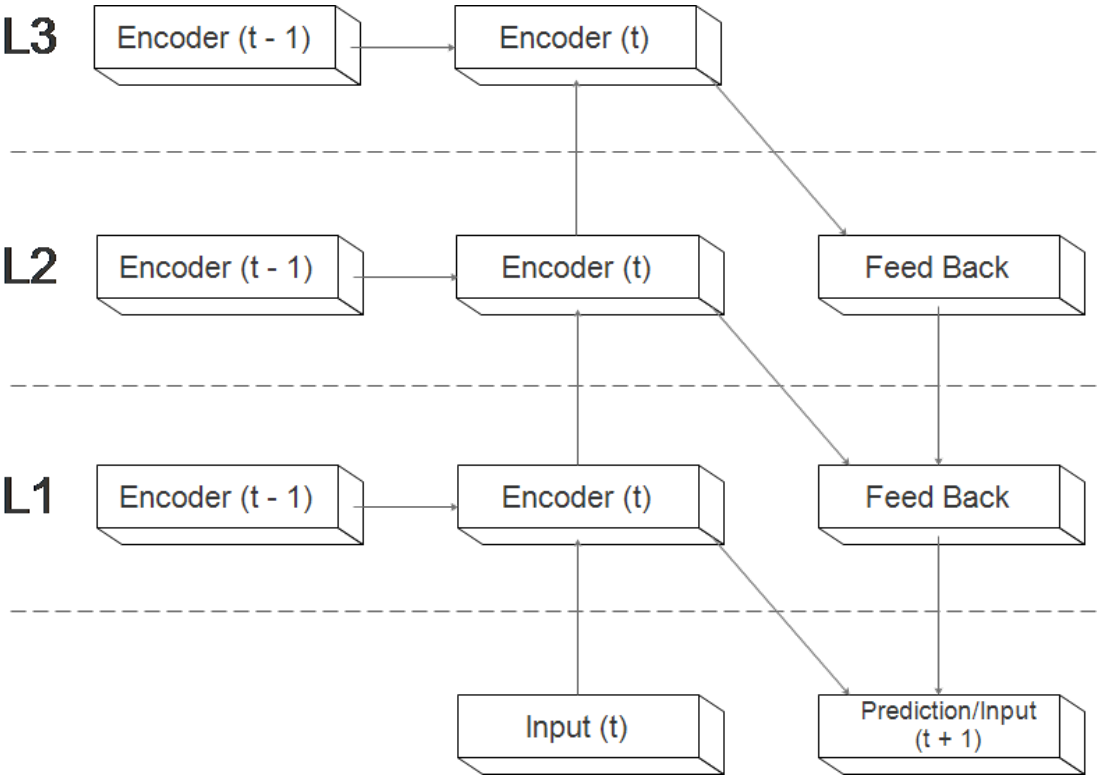
NeoRL's Architecture

- NeoRL can be seen as a deep recurrent neural network
- Each layer consists of an encoder and a decoder
- The encoder forms sparse distributed representations (SDRs) of the input
- The decoder uses these to predict the next timestep of input
- The encoder has recurrent connections to itself (t – 1)
- The decoder has feedback connections from higher layers



NeoRL's Architecture: Hierarchy

- Layers can be stacked to form a bidirectional hierarchy
- Processing happens in 2 passes: The up pass (feature extraction) and the down pass (prediction)
- Each layer's prediction is used as feed-back input to the next lower layer
- Recurrent connections are intra-layer only



Running NeoRL

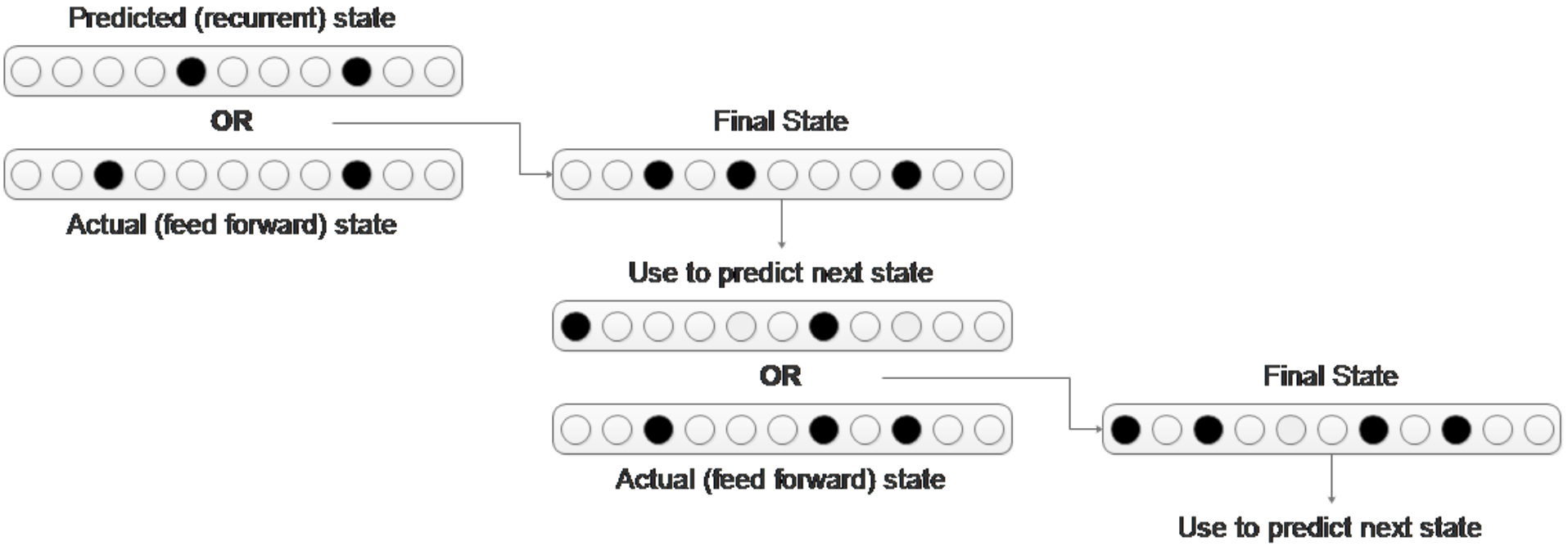
- The encoder and decoder form a sort of predictive autoencoder
- The hidden states of the encoder are kept sparse through a K-sparsity restriction
- This encoder-decoder pair is then trained with backpropagation (1-step). The decoder also trains feedback connections as an additional input layer
- But what about the recurrent connections? Could use backpropagation through time... but there is something more efficient

Predictive Coding

- NeoRL uses a form of predictive coding (PC) to organize its hidden state to allow for accurate predictions in partially observable (memory-requiring) tasks
- The predictive coding idea works as follows: Have the recurrent connections try to predict the next network state, then OR this prediction with the actual state to produce the final hidden state
- This means that when an incorrect prediction is made, more nodes will be active which can then be used for other predictions. So, as soon as an input cannot be predicted based off of the current timestep (reactive), it will start using additional nodes to bridge the gap
- Uses eligibility traces to do so without backpropagation through time

Predictive Coding Example

- If the next state cannot be predicted from the current state, this will produce a prediction error
- If this error is not resolved, it will propagate further and further back in time



Reinforcement Learning

- How can we use this predictive hierarchy for reinforcement learning?
- Good ol' TD (temporal difference) learning
- Basic idea: Learn to predict the reward ahead of time, then if you get more then expected you did well otherwise you did poorly (this is the temporal difference error)
- Reward is propagated back in time (credit assignment)
- Results in this Bellman equation (Q learning shown here, NeoRL uses SARSA which is similar):

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right)$$

Reinforcement Learning with NeoRL

- We must predict the cumulative reward at each timestep
- Set aside a certain set of predictions which will count as actions
- These actions are updated to predict whatever they are currently outputting (reinforcement) in the presence of a positive temporal difference error, and stay the same if the temporal difference error is negative
- NeoRL extends this further by using eligibility traces to propagate reward back in time efficiently

Feeding inputs to NeoRL

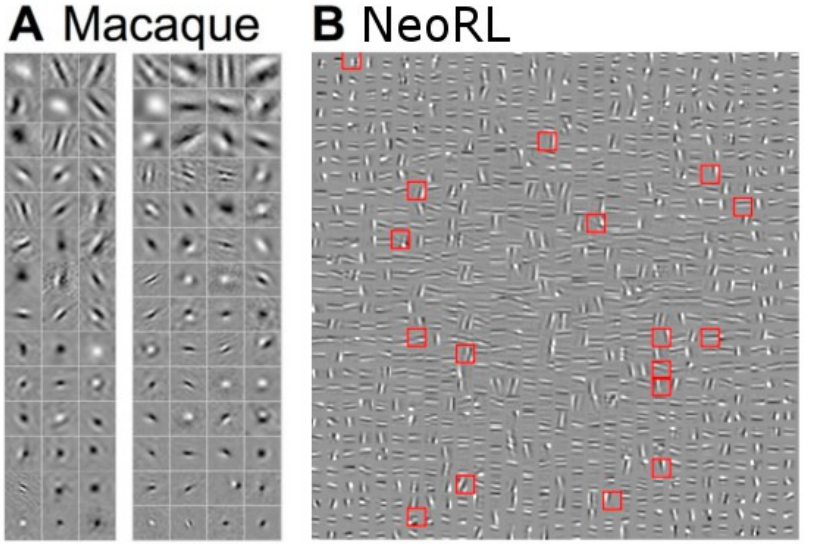
- NeoRL works best when inputs are pre-processed a bit
- In particular, a whitening transform helps with better distributing the SDRs across the input space
- Further, scalars are best encoded by performing several random transformations on them, again to better distribute the SDRs

NeoRL's Data Whitener

- NeoRL uses a simple local whitening transformation to process images
- This removes correlations in the image, good for input to a sparse coder that assumes decorrelated input
- Once in sparse form, input is already decorrelated, so no further whitening is necessary
- It works by examining pixels in a radius, measuring covariance with the center pixel, and then boosting/weakening the pixel based on this. It also has a normalizing effect, as the image gets transformed into the [-1, 1] range

Whitening Example

Filters learned:



Experiments with NeoRL

- Show them

The End

- NeoRL: https://github.com/222464/NeoRL
- MiniNeoRL: https://github.com/222464/MiniNeoRL