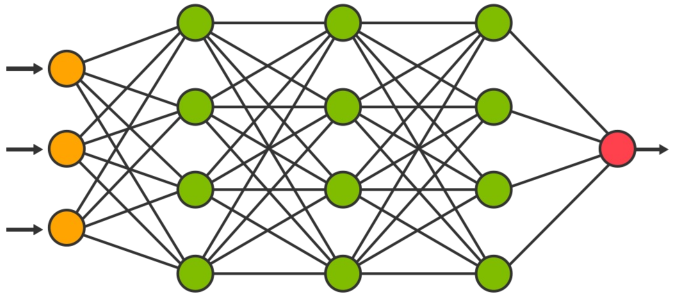
Large-Scale Retrieval for Reinforcement Learning

Peter C. Humphreys , Arthur Guez, Olivier Tieleman, Laurent Sifre, Théophane Weber, Timothy Lillicrap

Interpreted by:
John Tan Chong Min

Question to ponder

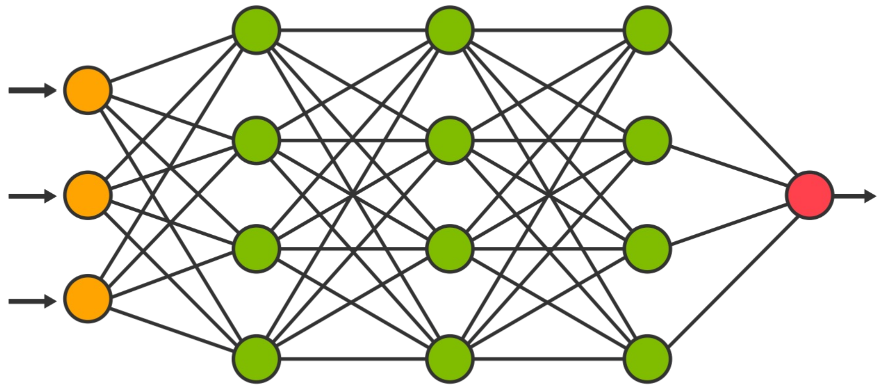
If you had a neural network, and you learnt something new about the world, how can you incorporate it into your memory?



Memory

- Weights of a neural network
- External memory

Types of memory



- Fast inference (one pass)
- Slow to learn (requires gradient descent)

- Slow inference unless using approximate techniques
- Can be quick to learn from experience (not done in this paper)

Memory to augment observations

- Model-based agent
- Predicts future policies and values conditioned on future actions in a given state
- Augment agent observation with nearest neighbor features

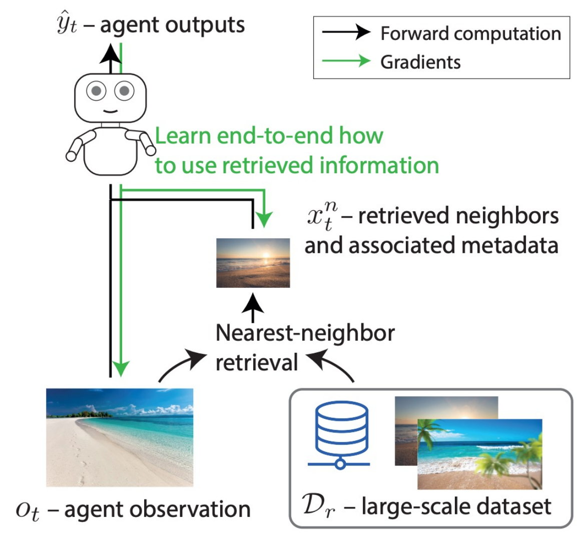

$\hat{y}_t$ – agent outputs
Forward computation
Gradients
Learn end-to-end how to use retrieved information
x_t^n – retrieved neighbors and associated metadata
Nearest-neighbor retrieval
o_t – agent observation
$\mathcal{D}_r$ – large-scale dataset

Figure 1: Retrieving information to support decision making. The agent observation o_t is used to generate a query to retrieve relevant information from a large-scale dataset $\mathcal{D}_r$. This information is used to inform network outputs $\hat{y}_t$. The agent is trained end-to-end to use this information to inform its decisions.

Potential concerns of using memory

- Need:

  1. A scalable way to select and retrieve relevant data

  2. Robust way of leveraging that data in the model

Part 1: Scalable Memory Retrieval

- Memory selection:
  - Inner product in an appropriate key-query space and select top *N* results
  - (Fixed) Embedding is generated using non-retrieval network embedding space
    - Embedding is the output of the near-final layer

- Uses the SCaNN architecture
  - Each entry mapped to a key vector
  - Retrieve entries with the lowest squared Euclidean distances between query and key

Neighbor Retrieval Procedure

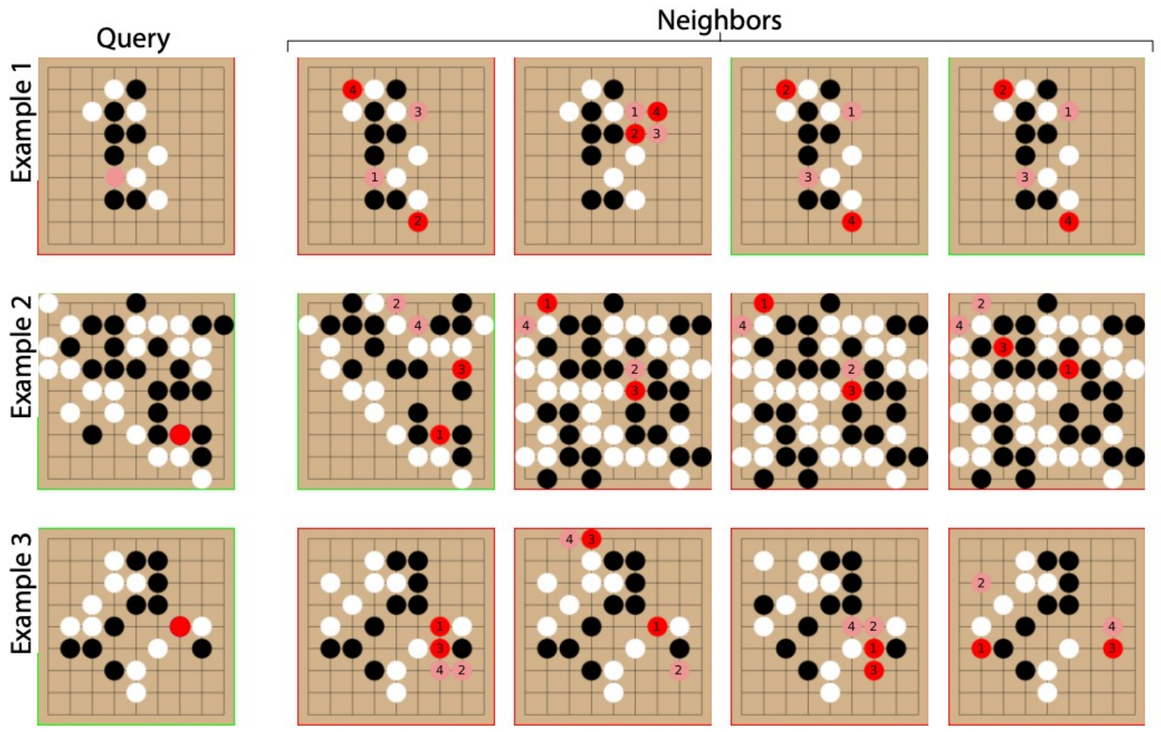
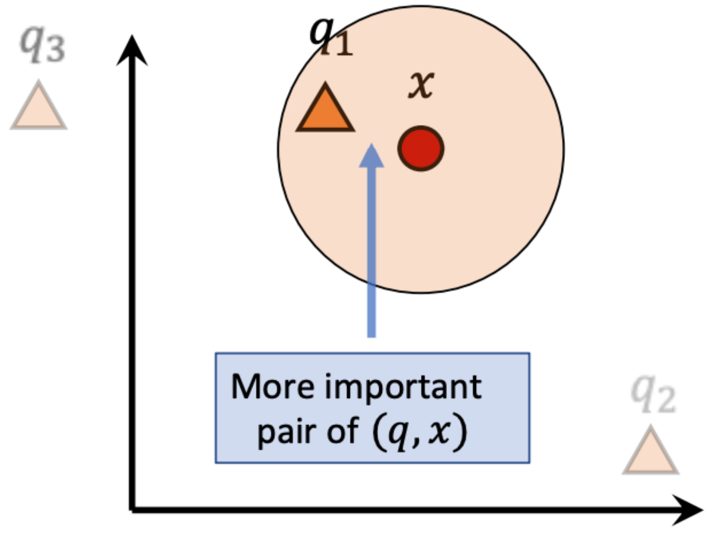


Figure 3: Visualisation of $N = 4$ approximate nearest-neighbors retrieved using a learned Go-specific distance function for 3 query positions (one per row). Red stone indicates the next action(s) – light red stones indicate white moves, dark red stones indicate black moves. The color of the board border indicates whether the current player to play (for each board) won the game.

SCaNN Architecture

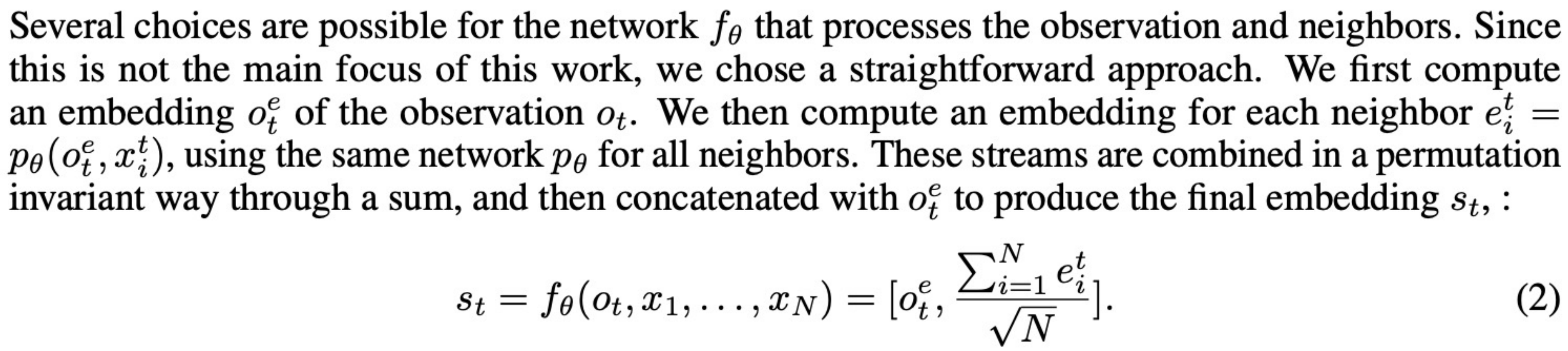


- Squared Euclidean distance is similar to reconstruction loss of the **query *q*** being placed at **position *x***

- Retrieved **memories *x*** with a lower squared Euclidean distance are more similar to the original **query *q***

Accelerating Large-Scale Inference with Anisotropic Vector Quantization. Ruiqi et al. 2019

Part 2: Robust Way of Leveraging Data



Several choices are possible for the network f_θ that processes the observation and neighbors. Since this is not the main focus of this work, we chose a straightforward approach. We first compute an embedding o_t^e of the observation o_t. We then compute an embedding for each neighbor $e_i^t = p_\theta(o_t^e, x_i^t)$, using the same network p_θ for all neighbors. These streams are combined in a permutation invariant way through a sum, and then concatenated with o_t^e to produce the final embedding s_t, :

$$s_t = f_\theta(o_t, x_1, \ldots, x_N) = [o_t^e, \frac{\sum_{i=1}^N e_i^t}{\sqrt{N}}]. \tag{2}$$

- Uses both the observation and the nearest neighbor data

- Embedding space is averaged out in this work

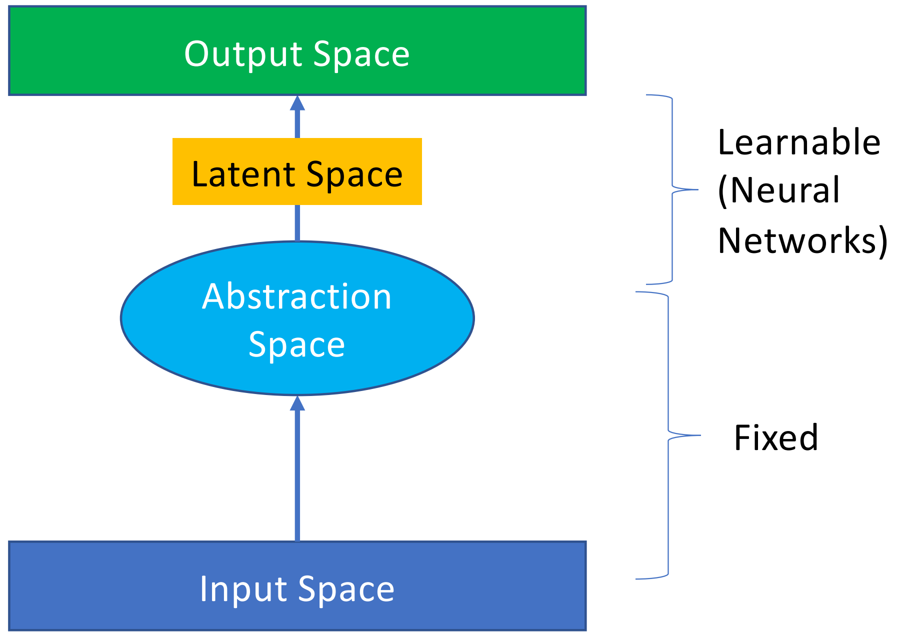
- Can also be concatenated

Recap:
Memories as abstraction
(my own views)

Abstraction as generalization

Abstraction

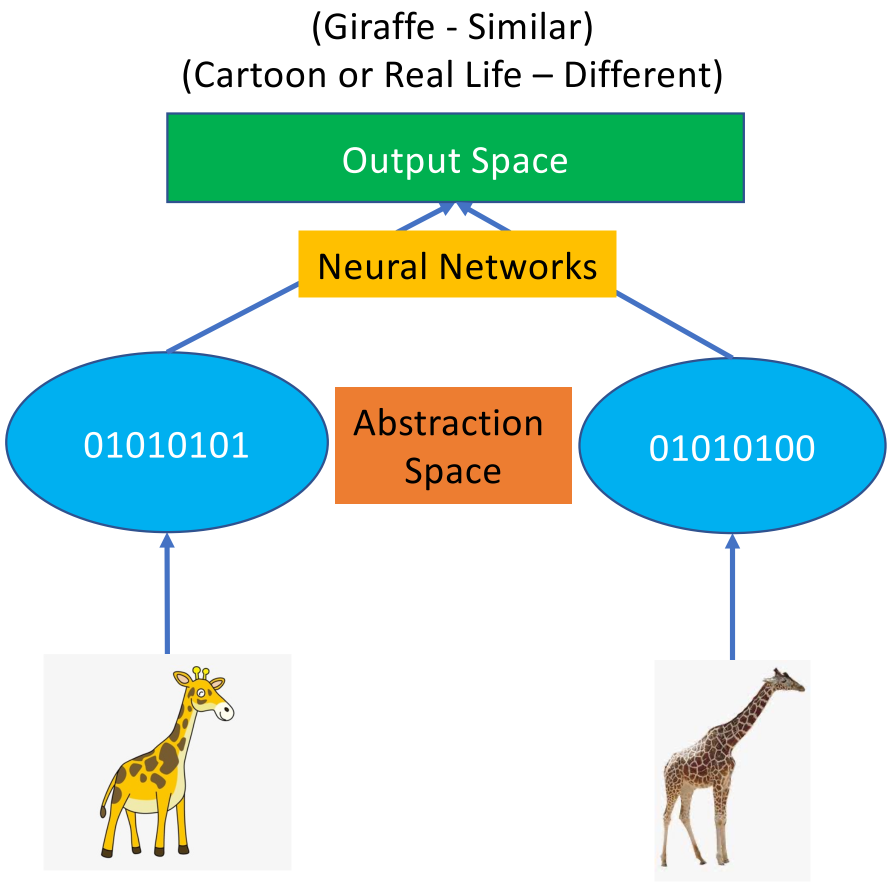
- Memories are stored in an abstract form
- The same abstraction can then referenced across contexts (generalization)
- This enables reuse of learnt memories in different contexts
- Fixed representation can be learnt over an initial training set of data



Abstraction

- **Hypothesis: Abstraction process is fixed and unlearnable, which stores new memories without the need of updating ALL previously stored memories the moment the abstraction mapping changes**

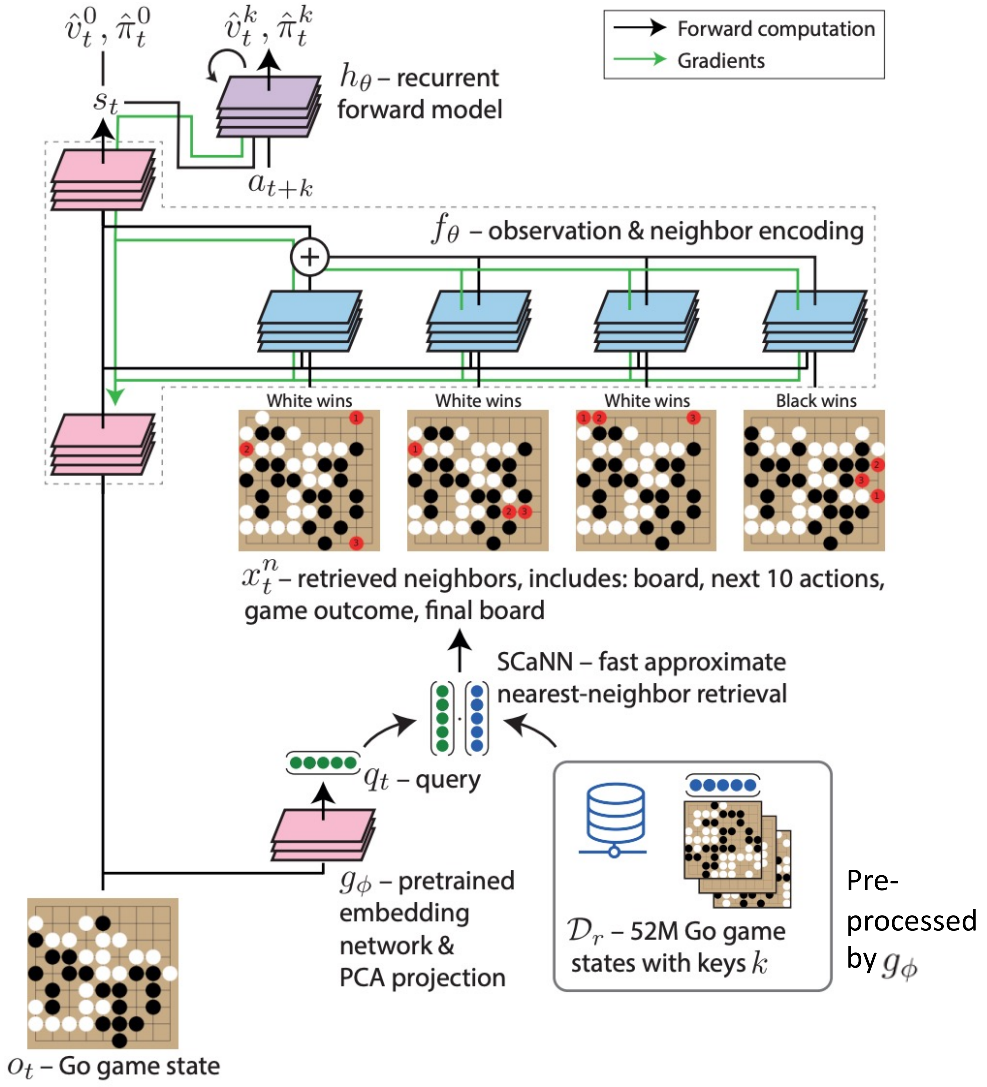
- With the abstraction space as input, neural networks can learn associations / dissociations between them for various tasks



Overall Model

Overall Model

- Current observation and memory bank is passed through *g, a* pre-embedding space and Principal Component Analysis (PCA) projection
- Match top *n* neighbors and process the network by concatenating their data to the observation
- Derive an associated value and policy (action)



Forward computation
Gradients
$\hat{v}_t^0, \hat{\pi}_t^0$
$\hat{v}_t^k, \hat{\pi}_t^k$
h_θ – recurrent forward model
s_t
a_{t+k}
f_θ – observation & neighbor encoding
White wins
White wins
White wins
Black wins
x_t^n – retrieved neighbors, includes: board, next 10 actions, game outcome, final board
SCaNN – fast approximate nearest-neighbor retrieval
q_t – query
g_ϕ – pretrained embedding network & PCA projection
$\mathcal{D}_r$ – 52M Go game states with keys k
Pre-processed by g_ϕ
o_t – Go game state

Overall Equation

$$\hat{y}_t = m_\theta(o_t, x_t^1, x_t^2, \ldots, x_t^N).$$

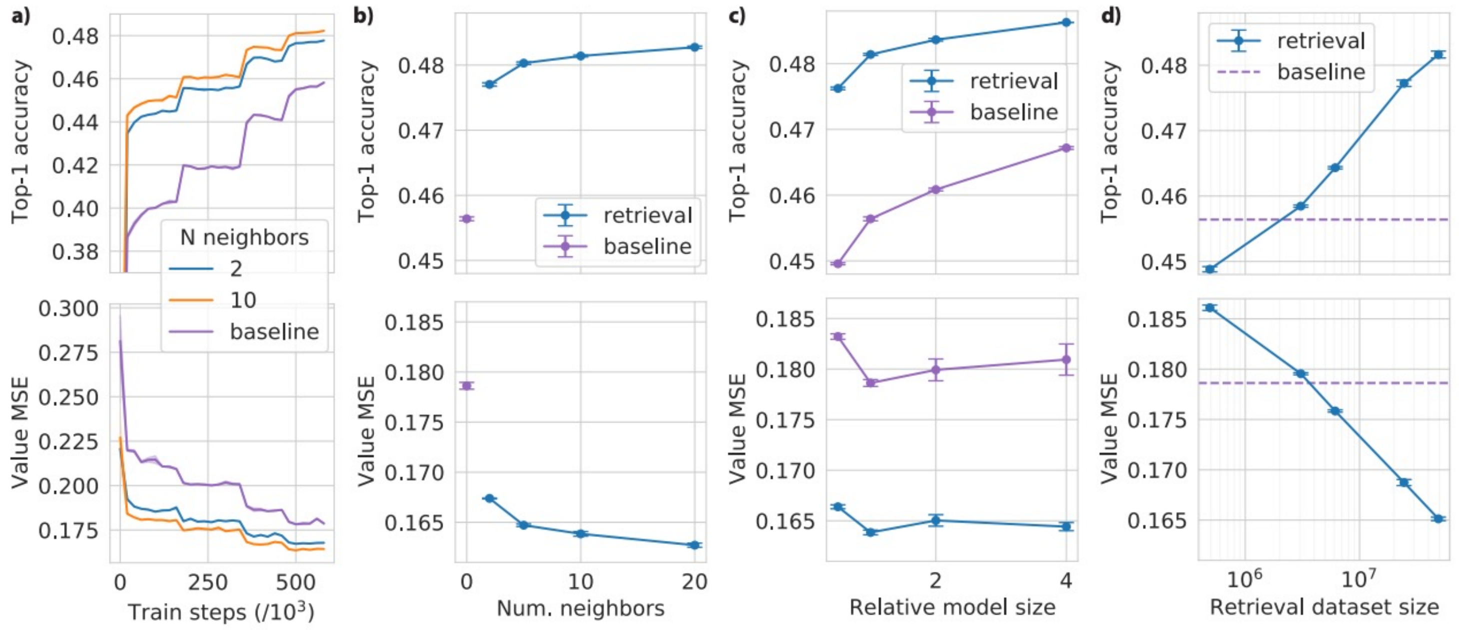
- *y* refers to model prediction (next action)
- *m* refers to model
- *o* refers to observation
- *x* refers to meta-data of nearest neighbors to observation from memory (action choices and their consequences)

Results

Experiment

- 3.5M expert 9x9 Go self-play games from AlphaZero-style agent
- Randomly subsampled 15% of the positions from these games as retrieval dataset
  - Proxy to avoid subsampling from same trajectory
- Training and retrieval datasets are the same: Use the entire training set to learn the mapping *g*
  - During testing/querying, the retrieval dataset is split into two parts and the part that does not contain the observation is queried
  - Prevent test data leakage

Results



- Better prediction with:
  - More neighbors
  - Larger model
  - Larger retrieval dataset

Win rate against Pachi

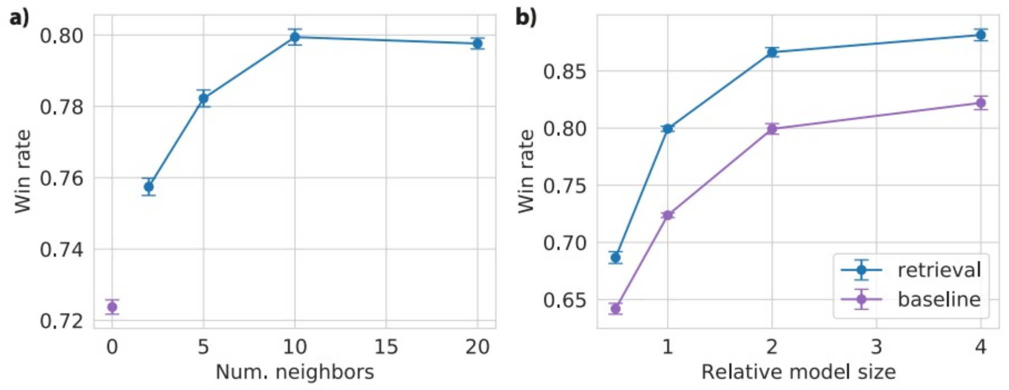


Figure 5: Win rate against a fixed reference opponent (Pachi) when playing using the MCTS policy π_s for (a) retrieval networks using varying numbers of retrieved neighbors and for a baseline non-retrieval network. (b) Win rate as a function of model size relative to the size used elsewhere in this study. Retrieval leads to a clear performance boost compared to non-retrieval baselines of the same capacity.

- Better win rate with increasing model size and neighbors
- Diminishing returns

Augmenting retrieval dataset improves performance

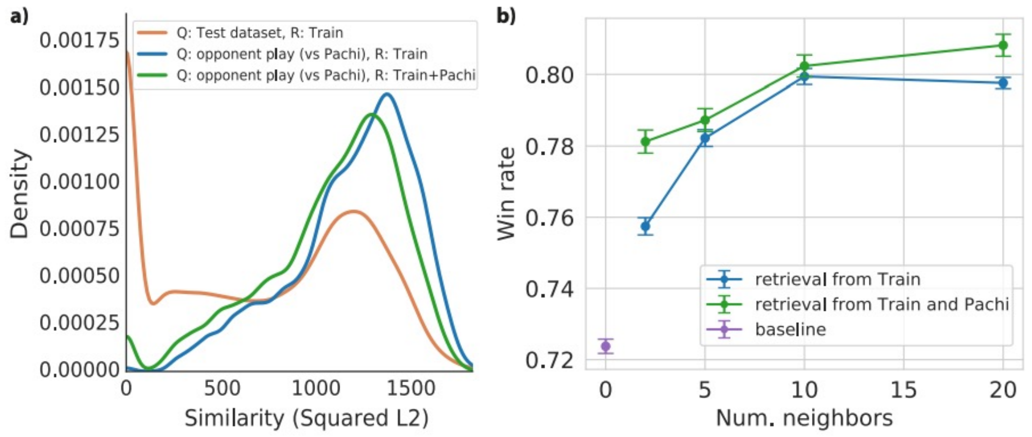
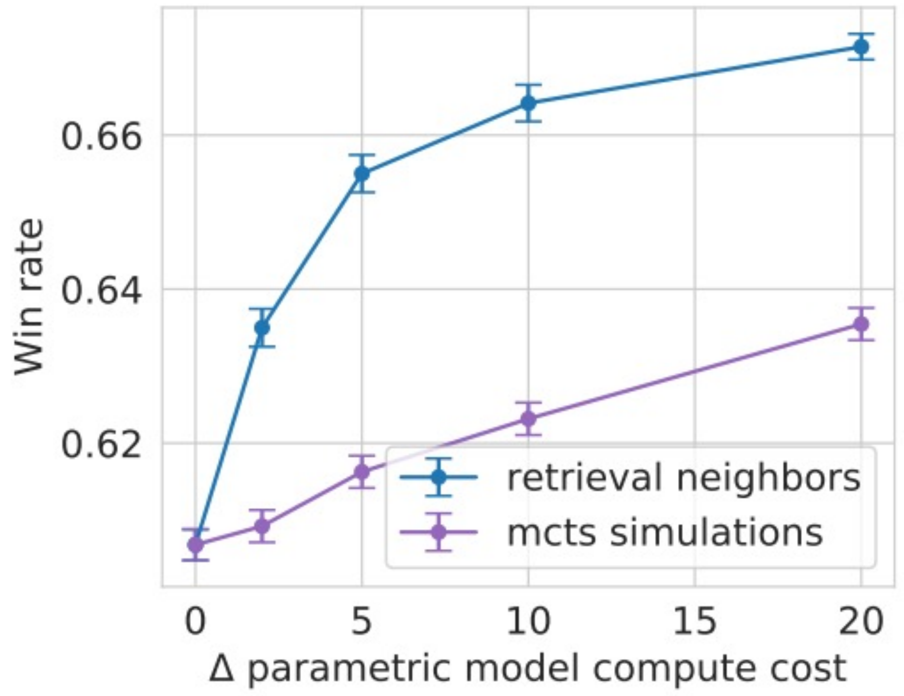


Figure 6: (a) Distribution of similarity distances from an encoded observation to its retrieved nearest neighbors. Retrieving from the original retrieval dataset using positions from the test dataset (orange) gives neighbors with a markedly different distance distribution to positions queried during play against a Pachi opponent (blue). Augmenting the retrieval dataset with ∼600k recorded agent-Pachi game states improves the similarity distribution (green). (b) For play with the MCTS policy π_s, this augmentation also leads to a consistent win-rate boost (green) over using the train retrieval dataset alone (blue).

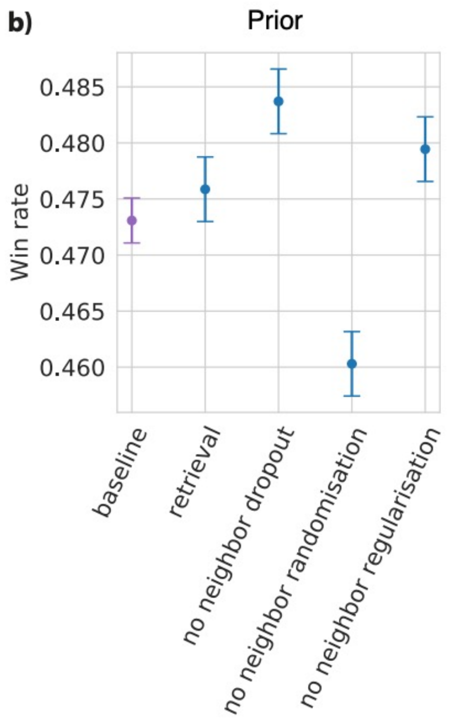
Retrieval better than more search!



Neighbor regularization

- Neighbor dropout: Drops out a random subset of retrieved neighbors
- Neighbor randomization: Replace subset of retrieved neighbors with neighbors of different observation within the mini-batch
- Neighbor regularization: Use loss to make embedding of neighbor retrieval similar to that of the original

Effect of neighbor regularization

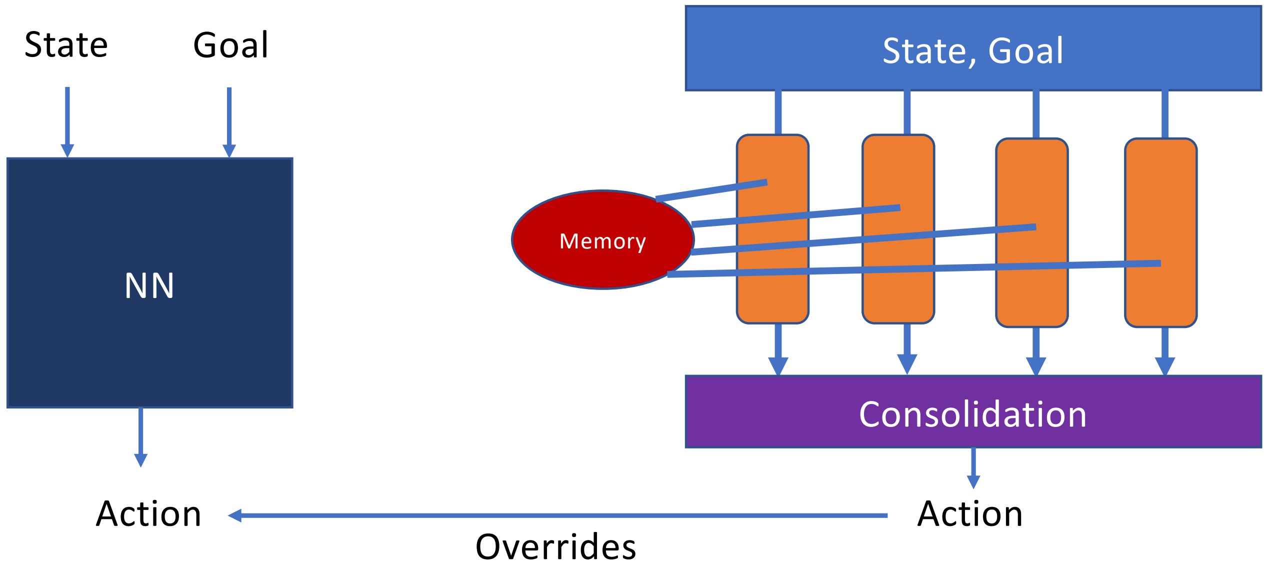


- Neighbor dropout: Seems to affect performance
- Neighbor randomization: Seems to be useful for performance – quite counterintuitive
- Neighbor regularization: Seems to affect performance

Questions to ponder

- Rather than just averaging out the embedding spaces of the nearest neighbors, why not use a weighted sum of them according to the Euclidean distance?
  - Can consider simply just using the Attention mechanism to get the final “value” as the embedding space
- Do we need to know future game trajectories, or just the next action / value?
- Can we do away with value and store the action?
- How can this mechanism learn without expert trajectories?

Follow-up work:
Two Networks – Fast and Slow



Learning, Fast and Slow - My own framework