Computation with uncertainty

Four broad hypotheses for how neuronal populations may encode and process uncertainty:
- Linear basis / kernel density estimators (Eliasmith and Anderson 2004)
- Densely Distributed population codes (Zemel et al. 1998; Sahani and Dayan 2003)
- Log-linear codes (Pitt et al. 2008)

Consistent computation requires a hand-crafted circuit.
- Supervised networks can learn to interpret noisy inputs, but do not fit a consistent representation of uncertainty (Orhan and Ma 2017).
- We seek a neural architecture and learning rule that automatically acquires consistent representations of and computations with uncertainty without explicit design.

The distributed distributional code (DDC)

- belief about r represented by linear projections of density p(y|x) on basis functions φi(r): VDC vector
- probability density function p(y|x)

(1) distributed population codes Zemel et al. 1998; Sahani and Dayan 2003

Representation

- Expectations (or moments) define an exponential family of beliefs by maximum entropy.
  - e.g. Gaussians are defined by linear and quadratic functions (first and second moments): where
  
  - arbitrary functions define more complicated families: such that
  
  \[ \mathbb{E}_r \psi(y) = r \]
  - The rates \( r \) are the mean parameters of the distribution.

- Can encode multiplicity and uncertainty:
  - multiplicity: left or right
  - multiplicity: mixed left and right

Learning

- Each neuron must estimate an expectation: easy to do with supervised learning.
- In fully observed models, probabilistic computations can be learned from training data:
  
  \[ \mathbb{E}_r \psi(y) = \sum_i \phi_i(r) \psi_i \]


Key question

- Can a network trained without explicit supervision of latents learn to represent probabilistic beliefs?
- Propagate uncertainty (message passing)?

A task that requires uncertainty computation

- Data: state-space model
  - Test
  - train
  - DDC
  - Bilinear RNN

\[ y_t \sim \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \] (normal)

\[ y_t \sim \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \] (normal)

\[ y_t \sim \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \] (normal)

\[ y_t \sim \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \] (normal)

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

Results

- The DDC network makes predictions closer to the particle filter than alternatives.

- Does a DDC representation arise automatically in the Bilinear RNN?

- Use radial/basis function (RBF) to learn functions on data history yielded lower APR.

- \( y_t \) found in the Bilinear network explains most of the variance in \( y_t \) (using simpler \( y_t \) (linear or quadratic) or learning functions on data history yielded lower APR.

- New belief is a bilinear function of memory \( y_t \) and observation \( y_t \).
- If we represent \( y_t \), \( y_t \) by a vector \( x_t \), we

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

- \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - Normal (y_t, \Sigma_t)
  - \( \mathcal{N}(\hat{y}_{t-1}, \Sigma_t) \) - normal

Conclusions

- DDC-based distributional code (DDC) represents distributions by nonlinear "moments".
  - A neural network model with bilinear architecture consistent with this hypothesis, trained to perform a task requiring inference (but without explicit probabilistic supervision);
  - even better than the alternative methods.
- The distributed representation provides a significantly lower mean-squared error.
  - Thus, the DDC offers a flexible, powerful and biologically plausible framework for representation, computation and learning.

- Questions
- How does the distributed code emerge in a bilinear form? How can we make a network learn an equivalent form? Does bilinear propagation occur in a short time window?
- How can the network learn sensory features and recognition in a natural environment?
- How can the network learn sensorimotor tasks? How can the network learn sensorimotor tasks? How can the network learn sensorimotor tasks? How can the network learn sensorimotor tasks? How can the network learn sensorimotor tasks? How can the network learn sensorimotor tasks? How can the network learn sensorimotor tasks?