Compiled Notes

Joseph Marino

August 23, 2017

Contents

1 Intelligence **1**
- 1.1 Intelligence . . . 1
- 1.2 Creating Intelligence . . . 1

2 Probability & Statistics **2**
- 2.1 Probability Basics . . . 2
  - 2.1.1 Definitions . . . 2
  - 2.1.2 Distributions . . . 2

3 Information Theory **3**
- 3.1 Basic Concepts . . . 3

4 Neuroscience **5**
- 4.1 Neurons . . . 5
- 4.2 Neural Circuits . . . 5
- 4.3 Brain Structures . . . 5

5 Neural Networks **6**
- 5.1 Basics . . . 6
  - 5.1.1 Linear Threshold Units . . . 6
  - 5.1.2 Multi-Layer Perceptrons . . . 6
  - 5.1.3 Backpropagation . . . 6

6 Graphical Models **7**
- 6.1 Basics . . . 7
- 6.2 Variational Inference . . . 7
  - 6.2.1 Inference as Optimization: Deriving the ELBO . . . 7
  - 6.2.2 Normalizing Flows . . . 8
- 6.3 Latent Variable Models . . . 9
  - 6.3.1 Temporal Latent Variable Models . . . 9

7 Representations & Representation Learning **11**
- 7.1 Motivation & Definitions . . . 11
- 7.2 Disentangled Representations . . . 11

8 Reinforcement Learning **13**
- 8.1 The RL Problem Setting . . . 13
  - 8.1.1 The Basics . . . 13
  - 8.1.2 Summary of Notation . . . 14

8.2 Value Based Methods . . . 14
8.2.1 Q Learning . . . 14
8.2.2 TD Learning . . . 14
8.3 Policy Gradient Methods . . . 14
8.4 Intrinsically Motivated RL . . . 14

9 Predictive Coding **15**
9.1 Introduction . . . 15
9.1.1 Literature Summary . . . 15
9.2 The Static Setting . . . 16
9.2.1 MAP State Estimation in a One–Level Model . . . 16
9.3 The Dynamic Setting . . . 18
9.4 Attention . . . 18
9.5 Active Inference . . . 18
9.6 Neural Implementation . . . 18
9.6.1 Correspondence with Cortical Mircrocircuits . . . 18

10 Experiment Best Practices **19**
10.1 GitHub . . . 19
10.2 Experiment Logging . . . 20
10.3 Plotting . . . 21

A Appendix 1 **24**

Chapter 1

Intelligence

1.1 Intelligence

Intelligence is often vaguely defined. In many situations, we use the term to refer to the ability to perceive and understand certain concepts, such as math, art, politics, business, etc., but this narrow definition fails to capture the spectrum that intelligence occupies. Instead, I use the following definition:

Intelligence is the ability of a system to interact meaningfully within an environment.

This definition has a few components: A **system** is some collection of matter: a molecular structure, single-celled organism, animal, machine, computer, human, society, etc. **Interact meaningfully**, in general terms, refers to non-random interactions[1] with the environment, i.e. the distribution of actions given the environmental state has low entropy. *todo: refine this definition.* These actions are typically associated with achieving some goal or reward, although this is not strictly necessary. Finally, **within an environment** emphasizes that intelligence is specific to an environment, also referred to as the data distribution or domain. Intelligence is not a characteristic of a system in isolation; it is always conditioned on a particular context that the system is suited to handle.

1.2 Creating Intelligence

There are two ways in which to create an intelligent system: through **design** and through **learning**. In design, one intelligent system constructs another intelligent system. In learning, a system gains intelligence through interaction with the environment.

[1] We could argue whether or not a passive system that can only *perceive* aspects of its environment is intelligent, but such a system has no practical purpose, since it has no way of communicating this intelligence to the environment.

Chapter 2

Probability & Statistics

2.1 Probability Basics

2.1.1 Definitions

Law of the unconscious statistician

2.1.2 Distributions

Chapter 3

Information Theory

3.1 Basic Concepts

Entropy is a measure of the uncertainty of a random variable. It tends to agree with the notion of information. Let X be a random variable, with alphabet $\mathcal{X}$ and probability mass function $p(x) = \Pr(X = x)$, with $x \in \mathcal{X}$. The entropy of X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x). \tag{3.1}$$

The units of entropy depend on the base of the logarithm. If the base is 2, then the units are in bits. If the base is e, then the units are in nats. To convert between different units of entropy, use $H_b(X) = (\log_b a) H_a(X)$. Note that entropy can be interpreted as the expectation of $-\log p(x)$. Also, entropy is non-negative: $H(X) \geq 0$.

The **joint entropy** of multiple random variables is defined similarly:

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x, y) \tag{3.2}$$

Likewise, the **conditional entropy** is defined as

$$\begin{aligned} H(Y|X) &= \sum_{x \in \mathcal{X}} p(x) H(Y|X = x) \\ &= -\sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x). \end{aligned}$$

Note that

$$H(X, Y) = H(X) + H(Y|X). \tag{3.3}$$

In words, joint entropy = individual entropy + conditional entropy. This also implies that

$$H(X, Y|Z) = H(X|Z) + H(Y|X, Z). \tag{3.4}$$

Because conditional entropy involves the conditional probability, we see that $H(Y|X) \neq H(X|Y)$. However,

$$H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{3.5}$$

Relative entropy is a measure of the distance between two distributions. Put differently, it is the inefficiency of assuming the distribution of the data is $q(x)$ when the true distribution is $p(x)$. This is also referred to as the **Kullback-Leibler distance** or **divergence**.

$$D_{KL}(p(x)||q(x)) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} = \mathbb{E}_{x \sim p(x)} \Big[\log \frac{p(x)}{q(x)} \Big] \tag{3.6}$$

Note that if there is a value $x \in \mathcal{X}$ such that $p(x) > 0$ and $q(x) = 0$, then $D_{KL} = \infty$. KL divergence is non-negative and is only zero when $p(x) = q(x)$. It is not symmetric, so it is not a proper distance measure.

Mutual Information is the amount of information that one random variable contains about another random variable. In other words, it is the reduction in the uncertainty of one random variable due to knowledge of the other. It is expressed as the relative entropy between the joint distribution $p(x, y)$ and the product of the marginals $p(x)p(y)$.

$$\begin{aligned} I(X;Y) &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \\ &= D_{KL}(p(x,y)||p(x)p(y)) \\ &= \mathbb{E}_{x,y \sim p(x,y)} \Big[\log \frac{p(x,y}{p(x)p(y)} \Big]. \end{aligned}$$

This can also be written as

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{3.7}$$

Thus, *X says as much about Y as Y says about X*. In words, mutual information is the amount of information in X minus the amount of information in X after observing Y (and vice versa). This corresponds to the amount of information in X that is explained by Y (and, again, vice versa). If Y perfectly explains X, then $H(X|Y) = 0$, so $I(X;Y) = H(X)$. At the other extreme, if Y says nothing about X, then $H(X|Y) = H(X)$, so $I(X;Y) = 0$. Also,

$$I(X;Y) = H(X) + H(Y) - H(X,Y), \tag{3.8}$$

and

$$I(X;X) = H(X) - H(X|X) = H(X). \tag{3.9}$$

Thus, entropy can be considered as the amount of self-information.

Chapter 4

Neuroscience

4.1 Neurons

4.2 Neural Circuits

4.3 Brain Structures

Chapter 5

Neural Networks

5.1 Basics

5.1.1 Linear Threshold Units

5.1.2 Multi-Layer Perceptrons

5.1.3 Backpropagation

Chapter 6

Graphical Models

6.1 Basics

6.2 Variational Inference

6.2.1 Inference as Optimization: Deriving the ELBO

Consider a model, $p(\mathbf{x}, \mathbf{z})$, that specifies a joint distribution over a latent variable $\mathbf{z}$ and observed variable $\mathbf{x}$. To infer the posterior distribution of $\mathbf{z}$ given $\mathbf{x}$, we can use the definition of conditional probability:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})}, \tag{6.1}$$

where the marginal likelihood (a.k.a. model evidence or partition function) $p(\mathbf{x})$ is calculated by marginalizing over the latent state space:

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \tag{6.2}$$

For large latent spaces and/or complicated models, performing the integration in eq. 6.2 is computationally intractable.

Variational inference transforms this intractable integration problem into an *optimization* problem by introducing an approximate posterior distribution, $q(\mathbf{z}|\mathbf{x})$, typically chosen from some simple family of distributions, such as independent Gaussians. We attempt to make $q(\mathbf{z}|\mathbf{x})$ as "close" as possible to $p(\mathbf{z}|\mathbf{x})$ by minimizing $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$. Notice that the word close is in quotations because KL divergence is not a true distance measure; it is not symmetric. When the (non-negative) KL divergence between these distributions is zero, we recover the true posterior, $p(\mathbf{z}|\mathbf{x})$. We cannot evaluate $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ directly, as it includes the true posterior, which we cannot tractably compute. Instead, we will maximize a lower bound on $p(\mathbf{x})$, which will have the effect of minimizing $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$, which we will now show. We start from the definition of KL divergence:

$$D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right], \tag{6.3}$$

$$D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{z}|\mathbf{x})\right]. \tag{6.4}$$

Plugging in the definition of conditional probability into the second term yields:

$$D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})}\right], \tag{6.5}$$

which can be expanded into

$$D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x},\mathbf{z})\right] + \log p(\mathbf{x}). \tag{6.6}$$

Now, we'll define the following quantity, which we refer to as the *evidence lower bound (ELBO)* or *variational lower bound* or *negative free energy*:

$$\boxed{\mathcal{L} \triangleq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})\right]} \tag{6.7}$$

Plugging this definition back into eq. 6.6, we get

$$D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})) = \log p(\mathbf{x}) - \mathcal{L}. \tag{6.8}$$

Rearranging terms, we see

$$\log p(\mathbf{x}) = \mathcal{L} + D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})). \tag{6.9}$$

Since $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ is non-negative, $\mathcal{L}$ is a lower bound on $\log p(\mathbf{x})$. Further, since $\log p(\mathbf{x})$ is not dependent on $q(\mathbf{z}|\mathbf{x})$, we see that maximizing $\mathcal{L}$ with respect to $q(\mathbf{z}|\mathbf{x})$ must minimize $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$. Thus, we can approximate the true posterior by optimizing $\mathcal{L}$ with respect to $q(\mathbf{z}|\mathbf{x})$.

Note that the ELBO can also be written in the following form:

$$\mathcal{L} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})\right] \tag{6.10}$$

$$\mathcal{L} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x})\right] \tag{6.11}$$

$$\mathcal{L} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z})\right] - D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{6.12}$$

This highlights that the ELBO specifies the optimal $q(\mathbf{z}|\mathbf{x})$ by trading off between representing the input, through the first term, and agreeing with the prior on the latent variables, through the second term. In other words, the first term attempts to fit the data, while the second term regularizes the representation.

6.2.2 Normalizing Flows

A common drawback of variational inference is that it is typically restricted to families of approximate posterior densities, e.g. factorized Gaussians. *Normalizing flows* [1] is a method for fitting flexible approximate posterior densities. It uses a sequence of invertible mappings, called a 'flow,' to transform simple posterior densities into arbitrary, flexible forms. If we start with a variable $\mathbf{z}$, drawn from a distribution $q(\mathbf{z})$, and apply an invertible, smooth mapping $f(\mathbf{z})$, then the change of variables formula allows us to write

$$q(\mathbf{z}') = q(\mathbf{z})\det\left|\frac{\partial f^{-1}}{\partial \mathbf{z}'}\right| = q(\mathbf{z})\det\left|\frac{\partial f}{\partial \mathbf{z}}\right|^{-1}. \tag{6.13}$$

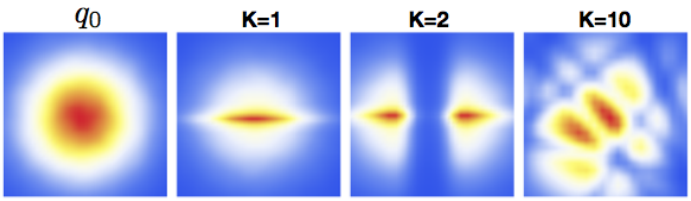

Figure 6.1: Normalizing flows applied to a 2D isotropic Gaussian, using transformations of the form given in eq. 6.15. Increasing the flow length K allows for more flexible approximate posterior distributions. Figure reproduced from [1].

Note that the mapping, f, can also depend on other inputs, such as observed variables $\mathbf{x}$ or other auxiliary inputs. By composing multiple mappings, we can construct arbitrarily complex densities. Thus, starting from an initial distribution of $q_0(\mathbf{z}_0)$, a flow of length K can be computed as

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \det \left| \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|. \tag{6.14}$$

This sequence can be interpreted as performing expansions and contractions on the initial density to construct a more flexible form.

To parameterize the mappings, [1] use transformations of the form

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\intercal \mathbf{z} + b) \tag{6.15}$$

where $\mathbf{u}$, $\mathbf{w}$, and b are learned parameters and $h(\cdot)$ is a non-linear function. The determinant of the Jacobian of this transformation is

$$\det \left| \frac{\partial f}{\partial \mathbf{z}} \right| = \left| \det \left(\mathbf{I} + \mathbf{u}(h'(\mathbf{w}^\intercal \mathbf{z} + b)\mathbf{w})^\intercal \right) \right| = \left| 1 + \mathbf{u}^\intercal h'(\mathbf{w}^\intercal \mathbf{z} + b)\mathbf{w} \right| \tag{6.16}$$

Inverse auto-regressive flow [2]

6.3 Latent Variable Models

6.3.1 Temporal Latent Variable Models

We have a sequence of observations $\mathbf{x}_{\leq T}$, which extend over T time-steps. Assume we also have a sequence of latent variables $\mathbf{z}_{\leq T}$. If we assume that time only moves in the forward direction, then with parameters θ, we can write the joint probability of this model as

$$p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}_t | \mathbf{x}_{<t}, \mathbf{z}_{\leq t}) p_\theta(\mathbf{z}_t | \mathbf{x}_{<t}, \mathbf{z}_{<t}). \tag{6.17}$$

The term $p_\theta(\mathbf{x}_t | \mathbf{x}_{<t}, \mathbf{z}_{\leq t})$ is the conditional likelihood or observation model, and the term $p_\theta(\mathbf{z}_t | \mathbf{x}_{<t}, \mathbf{z}_{<t})$ is the prior, transition, or dynamics model. The full graphical model is represented in figure 6.2. The conditional dependencies show that the computation in this model grows exponentially with the sequence length T. Later, we will enforce stricter assumptions on the temporal dependencies to make computation tractable.

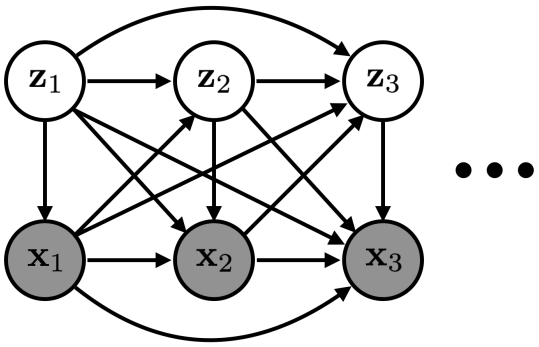

Figure 6.2: Simplified graphical model representation of temporal latent variable model with all connections present. Note that we're assuming the 'arrow of time' points forward; variables can only affect other variables at the current time step or at future time steps. The parameters θ have been omitted for clarity.

To train or perform inference with this model, we need to compute $p_\theta(\mathbf{x}_{\leq T})$, which involves marginalizing over all states $\mathbf{z}_{\leq T}$:

$$\log p_\theta(\mathbf{x}_{\leq T}) = \log \int p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) d\mathbf{z}_{\leq T} \tag{6.18}$$

Instead, we'll use an approximate posterior distribution, $q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})$, to compute a lower bound on this quantity:

$$\log p_\theta(\mathbf{x}_{\leq T}) = \log \mathbb{E}_{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \left[\frac{p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T})}{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \right] \tag{6.19}$$

$$\log p_\theta(\mathbf{x}_{\leq T}) \geq \mathbb{E}_{q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T})} \left[p_\theta(\mathbf{x}_{\leq T}, \mathbf{z}_{\leq T}) - q(\mathbf{z}_{\leq T}|\mathbf{x}_{\leq T}) \right] \tag{6.20}$$

Chapter 7

Representations & Representation Learning

7.1 Motivation & Definitions

[3] has a good review of different desirable qualities of representations.

Overview of types of representations. How do we learning representations? What training criteria/conditions result in different representations? Are certain representations better than others? In which cases and why? The quality of a representation can only fundamentally be judged by its usefulness in helping to perform some task. That is, representations must be viewed in the context of a task.

7.2 Disentangled Representations

Define disentanglement. Why are disentangled representations helpful? Trade off between disentanglement and faithfully representing the data. How do we learn disentangled representations, while at the same time, respecting the structure of the latent space? Importance of priors.

Two random variables are **independent** if their joint probability can be expressed as the product of their marginals:

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y}), \tag{7.1}$$

and they are **conditionally independent** if their conditional joint probability can be expressed as

$$p(\mathbf{x}, \mathbf{y}|\mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{y}|\mathbf{z}). \tag{7.2}$$

Independence is related to covariance, but is a stronger property. Two variables that are independent have zero covariance and two variables that have non-zero covariance are dependent. Zero covariance implies that the variables have no *linear* dependence. Independence implies that the variables also have no *non-linear* dependence. That is, it is possible for two dependent variables to have zero covariance.

Notes:

[4] propose using a cross-covariance penalty term in the setting of semi-supervised learning of the form:

$$C(\hat{\mathbf{y}}^{1\ldots N}, \mathbf{z}^{1\ldots N}) = \frac{1}{2}\sum_{ij}\left[\frac{1}{N}\sum_{n}(\hat{y}_i^n - \bar{\hat{y}}_i)(z_j^n - \bar{z}_j)\right]^2, \tag{7.3}$$

where $\hat{\mathbf{y}}$ is a vector of (one-hot) inferred labels and $\mathbf{z}$ is a latent representation. N is the batch size.

[5] propose to use a regularization weight in a VAE setting to "upweight" the amount of regularization as a means of promoting **redundancy reduction and independence** among the latent representation. They also argue for the importance of **dense sampling of the (continuous) data manifold** for disentanglement. Sparse sampling of the data manifold results in ambiguity is the manifold interpretation, requiring additional supervision for disentanglement.

[6] use supervision to impose structure on part of the latent representation in a VAE.

Dropout [7] is a technique for preventing "fragile coadaptation" between units within a representation, effectively enforcing that they represent different (independent) quantities. This technique also results in redundancy within the representation.

Chapter 8

Reinforcement Learning

8.1 The RL Problem Setting

8.1.1 The Basics

Reinforcement learning involves an *agent* interacting within an *environment*. The agent selects actions, and the environment responds with new states and rewards. Sutton & Barto [8] identify four additional sub-elements to reinforcement learning systems:

- a *policy*,
- a *reward signal*,
- a *value function*,
- and, optionally, a *model* of the environment.

A *policy* defines a mapping from (perceived) states to output actions to be taken. A *reward signal* is a real-valued number that the agent receives from the environment. The agent tries to maximize its total reward, which is referred to as value. The *value function* specifies the expected value, starting from a particular state. Thus, it is ultimately value that we try to optimize. Finally, a *model* of the environment is the agent's simulation of the environment, allowing the agent to plan actions and predict outcomes (states, rewards, etc.).

The boundary between agent and environment is often not clearly defined. Sutton & Barto place the agent-environment boundary at the limit of the agent's control, not at the physical boundary. For instance, the limbs of the agent, its internal energy reserves, and even the reward mechanism are considered to be part of the environment, since the agent does not have absolute control over these aspects.

We assume that time unfolds in a series of discrete time-steps, $t = 1, 2, 3, \ldots$ At each time step, the agent is in some state $S_t \in \mathcal{S}$ and chooses some action $A_t \in \mathcal{A}(S_t)$. Here, $\mathcal{S}$ denotes the space of possible states and $\mathcal{A}(S_t)$ denotes the space of possible actions from state S_t. The action is chosen based on the agent's policy $\pi_t(a|s)$, which probabilistically maps states $S_t = s$ to actions $A_t = a$. At the next time step, the agent receives a real-valued reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and arrives in the next state, S_{t+1}.

The *return*, G_t, is the (discounted) sum of rewards starting at time t. For *episodic tasks*, in which the sequence has a finite length, T, this is defined as

$$G_t \triangleq R_{t+1} + R_{t+2} + \cdots + R_T = \sum_{k=0}^{T-t-1} R_{t+k+1}, \tag{8.1}$$

However, for *continuing tasks*, in which the time sequence length can be infinite, we include a *discount rate*, $\gamma \in [0, 1]$, to prevent the return from going to infinity:

$$G_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{8.2}$$

The *value function* or *state-value function*, $v_\pi(s)$, is defined as the expected return from being in state s when using policy π:

$$v_\pi(s) \triangleq \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]. \tag{8.3}$$

Note that here we are assuming that the task respects the Markov property; the current state, along with the policy, contains all of the information to determine the value function. We can also define an *action-value function*, $q_\pi(s, a)$, which specifies expected returns for each action a taken from state s using policy π:

$$q_\pi(s, a) \triangleq \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \tag{8.4}$$

Using the definition of the value function, we can derive a recursive relationship known as the Bellman equation:

8.1.2 Summary of Notation

8.2 Value Based Methods

8.2.1 Q Learning

8.2.2 TD Learning

8.3 Policy Gradient Methods

8.4 Intrinsically Motivated RL

[9] use internal motivation, along with an option framework, to learn a collection of hierarchical skills.

[10] use empowerment.

Chapter 9

Predictive Coding

9.1 Introduction

Predictive coding, in its current form, was formalized by [11]. This theoretical neuroscience model posited that sensory processing is a filtering problem, in which latent states underlying the environmental stimuli are estimated according to their agreement with the observed input as well as prior beliefs about these quantities. This offered an explanation for "extra-classical" receptive field effects in visual processing, in which stimuli outside the receptive field of a particular cortical neuron are able to affect that neuron's activity. This was explained as the result top-down prior beliefs.

However, the notion of perception as a generative process dates back to at least [12]. *TODO: mention other early work before Rao and Ballard with similar ideas*

Predictive coding claims that sensory processing, i.e. perception, is fundamentally about constructing a generative model of the input sensory signal. To perform *inference* in this model (to perceive), the model uses its current estimate of the latent variables underlying the environment to generate reconstructions or predictions of the input. Using the residual (error) from this reconstruction or prediction, along with residuals from prior beliefs, the model updates its estimate of these latent variables. *Learning* corresponds to updating the parameters of the generative model to improve these overall residuals.

9.1.1 Literature Summary

[13, 14, 15] presents a free energy formulation of predictive coding that uses expectation maximization. Primarily focuses on the static setting. Puts forth a theory of cortical processing as carrying out predictive coding.

[16] presents *dynamic causal modeling.*

[17] presents *variational filtering.*

[18] presents *dynamic expectation maximization* (DEM).

[19] presents *generalized filtering*, which does not make the mean-field assumption and instead treats all unknown variables as conditionally dependent. This differs from variational filtering [17] in that it assumes the parameters and precisions also change with time, with

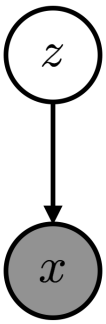

Figure 9.1: A graphical model denoting a latent variable model in which a single latent variable z generates a single observed variable x. In predictive coding, we typically assume that $p(z)$ and $p(x|z)$ are Gaussian densities, and the generative mapping from z to x is a non-linear function.

a prior on smoothness of these changes.

[20] claims that attention can be formulated as the adjustment of prior precisions in the predictive coding model.

[21], [22] , [23] [24], [25] discuss *active inference*.

9.2 The Static Setting

9.2.1 MAP State Estimation in a One–Level Model

Consider a situation in which the value of a single latent variable z must be inferred from a single observed variable x. This is represented by the graphical model in figure 9.1. Let g denote a non-linear function defining how z generates x. Assume that the generated output takes the form of a normal distribution with mean $\mu_x = g(z)$ and constant variance σ_x^2:

$$p(x|z) = \mathcal{N}(x; g(z), \sigma_x^2). \tag{9.1}$$

Recall that in one dimension, a normal distribution takes the form

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{9.2}$$

We will place a prior on z, which we also assume is a normal distribution with constant mean μ_p and constant variance σ_p^2:

$$p(z) = \mathcal{N}(z; \mu_p, \sigma_p^2). \tag{9.3}$$

To infer the posterior distribution of z, we can use Bayes' rule to invert the generative model:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}, \tag{9.4}$$

where the marginal distribution in the denominator is given as

$$p(x) = \int p(x|z)p(z)dz. \tag{9.5}$$

In general, computing the exact posterior distribution is computationally intractable due to the integration over z in eq. 9.5. Instead, we'll resort to variational inference[1] to find the maximum (mode) of the posterior, otherwise known as the maximum a posteriori or MAP estimate. This is the *most likely* estimate of the value of z. We will define our approximate distribution, a point mass estimate at μ_q, as $q(z|x) = \delta(z = \mu_q)$, with the MAP estimate denoted as $\hat{\mu}_q$.

We want to maximize the evidence lower bound (ELBO), $\mathcal{L}$:

$$\mathcal{L} = \mathbb{E}_{z\sim q(z|x)}\left[\log p(x,z) - \log q(z|x)\right] = \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z) + \log p(z) - \log q(z|x)\right] \tag{9.6}$$

$$\mathcal{L} = \log\left(\frac{1}{\sqrt{2\pi\sigma_x^2}}e^{-\frac{(x-g(\mu_q))^2}{2\sigma_x^2}}\right) + \log\left(\frac{1}{\sqrt{2\pi\sigma_p^2}}e^{-\frac{(\mu_q-\mu_p)^2}{2\sigma_p^2}}\right) \tag{9.7}$$

$$\mathcal{L} = -\frac{1}{2}\log(2\pi\sigma_x^2) - \frac{(x-g(\mu_q))^2}{2\sigma_x^2} - \frac{1}{2}\log(2\pi\sigma_p^2) - \frac{(\mu_q-\mu_p)^2}{2\sigma_p^2} \tag{9.8}$$

$$\mathcal{L} = \frac{1}{2}\left(-\log(\sigma_x^2) - \frac{(x-g(\mu_q))^2}{\sigma_x^2} - \log(\sigma_p^2) - \frac{(\mu_q-\mu_p)^2}{\sigma_p^2}\right) + \text{const.} \tag{9.9}$$

In going from eq. 9.6 to eq. 9.7, we used the fact that $q(z|x)$ is a delta function, so the expectation becomes an evaluation at a single point, $z = \mu_q$. The expectation of $q(z|x)$ at this point is 1, so $\log q(z|x)$ evaluates to 0. To find the MAP estimate, we must solve the following optimization problem:

$$\hat{\mu}_q = \text{argmax}_{\mu_q}\mathcal{L} = \text{argmax}_{\mu_q}\frac{1}{2}\left(-\log(\sigma_x^2) - \frac{(x-g(\mu_q))^2}{\sigma_x^2} - \log(\sigma_p^2) - \frac{(\mu_q-\mu_p)^2}{\sigma_p^2}\right). \tag{9.10}$$

To use first-order optimization methods, we must find the partial derivative of $\mathcal{L}$ w.r.t. μ_q:

$$\frac{\partial\mathcal{L}}{\partial\mu_q} = \frac{x-g(\mu_q)}{\sigma_x^2}\frac{dg}{d\mu_q} + \frac{\mu_q-\mu_p}{\sigma_p^2} \tag{9.11}$$

We see that we have two terms: **the first term moves the estimate toward agreement with the observation**, and **the second term moves the estimate toward agreement with the prior**. Each of these terms are weighted by their corresponding variances. In other words, inference (i.e. finding the optimal estimate of μ_q) involves a weighted combination of *bottom-up* and *top-down* information. By repeatedly moving our estimate μ_q along this gradient, we can hopefully arrive at the MAP estimate $\hat{\mu}_q$. Note that we may not find the true value $\hat{\mu}_q$ if the optimization surface is non-convex.

[1] We are not actually using variational inference here, since the maximum of $p(z|x)$ must also be the maximum of $p(x,z)$ through the definition of conditional probability: $p(z|x) = \frac{p(x,z)}{p(x)}$, since $p(x)$ does not depend on the value of z.

We would like to extend this single dimensional model to handle latent and observed variables of arbitrary size. In this setting, $\mathbf{x}$ and $\mathbf{z}$ are now vectors. The conditional likelihood, $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{\mathbf{x}} = g(\mathbf{z}), \boldsymbol{\Sigma}_{\mathbf{x}})$ is now a multi-variate Gaussian distribution. The general form for this distribution is

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \tag{9.12}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix, $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$, and n is the dimensionality of the vector $\mathbf{x}$. The prior on $\mathbf{z}$ is also a multi-variate Gaussian: $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$. The MAP estimate is now a vector, $\hat{\boldsymbol{\mu}}_q$, of the most likely estimate of $\boldsymbol{\mu}_q$. To find this estimate, we must again optimize $\mathcal{L}$ w.r.t. $\boldsymbol{\mu}_q$. Repeating the steps above, the gradient, $\nabla_{\boldsymbol{\mu}_q}\mathcal{L}$, is given as:

$$\nabla_{\boldsymbol{\mu}_q}\mathcal{L} = \left(\frac{dg}{d\boldsymbol{\mu}_q}\right)^\intercal \boldsymbol{\Sigma}_{\mathbf{x}}^{-1}(\mathbf{x} - g(\boldsymbol{\mu}_q)) + \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p). \tag{9.13}$$

9.3 The Dynamic Setting

9.4 Attention

9.5 Active Inference

9.6 Neural Implementation

9.6.1 Correspondence with Cortical Mircrocircuits

[26] outlines ideas about correspondence between predictive coding and cortical microcircuits.

[27] proposes that the pulvinar region of the thalamus is involved in setting precisions of predictions.

[28] gives an introductory tutorial to predictive coding with ideas about implementing predictive coding in neocortex.

[29] draws comparisons between backprop and predictive coding.

Chapter 10

Experiment Best Practices

This chapter contains some helpful practices for carrying out (machine learning) research projects.

10.1 GitHub

GitHub is an online platform for project source control. It is essential when collaborating on projects, as it allows you to meticulously track changes and manage conflicts between multiple versions of the code. But GitHub is also helpful when working alone on a project across multiple machines or if you just want to open source your code. The following are the basic commands for using GitHub:

Clone a repository:

```
git clone <repository address>
```

Add a file to the repository:

```
git add <filename>
```

Or to add everything to the repository:

```
git add .
```

or

```
git add -A
```

Commit changes to the repository:

```
git commit -m "<message>"
```

or

```
git commit
```

Note that if you run the second command, you will enter a vi interface where you can enter a multi-line message. To exit, type esc :wq

Push all committed changes to the repository:

```
git push
```

Pull the repository:

```
git pull
```

To create a new branch:

```
git branch <branch name>
```

Switch current branch:

```
git checkout <branch name>
```

Merge branch changes back into master (while currently checking out other branch):

```
git merge master
```

You can send pull requests on GitHub.com. After the branch is merged, you can delete the branch on the website.

To conclude, a good workflow practice is to (1) pull the repository at the beginning of each work session, (2) push any incremental changes at regular intervals, and (3) create new branches for any major changes to be implemented.

10.2 Experiment Logging

Maintaining proper records of experiments is an essential part of conducting good research. Scientific notebooks (preferably in a digital format) are often helpful for keeping high-level notes about experiments and methodologies, but one must keep experimental logs to track the different experimental set-ups that have been tested and the results. These are vital for surveying your project (what set-ups work? what set-ups should we consider trying?) and eventually communicating your results. Implementing and keeping track of these logs can seem like a hassle, but they are just as important as the experimental code itself.

A basic experiment logging pipeline looks something like this:

1. At the beginning of the experiment, **create a log directory dedicated to this experiment**. This log directory should have a unique name. An easy choice is the date and time of the start of the experiment, which allows one to sort all of the log directories in chronological order.

2. **Copy the source code into a subdirectory in the log directory**. This is absolutely vital, as it allows one to maintain the code that led to the result of that experiment, ensuring repeatability. If you are modifying the source code while running experiments, reproducing earlier experimental conditions can be a nightmare.

   The following code creates a new log folder for the experiment and saves the files from the current directory into a subdirectory called “source.”

```
import os
from time import strftime

def init_log(log_root):
    log_dir = strftime("%b_%d_%Y_%H_%M_%S") + '/'
    os.makedirs(log_path)
    os.system("rsync -au --include '*/' --include '*.py' --
        exclude '*' . " + log_path + "source")
```

3. In addition to maintaining source code of the experiment, it's important to **log the experimental results**. At the bare minimum, one should log train and validation performance metrics. Unless these metrics are logged, you have no means of comparing different experiments after they have run. There is an unending list of other results that can be logged, such as: outputs of the model, activations within the model, distributions of different metrics and parameters as they evolve during training, etc. The downside of logging this information is that it can slow down training and take up disc space. Often, it's best to set certain logging or visualization flags within your code. This allows you to log and understand what is happening within your model during the early experimental phase/debugging, then turn off this functionality during extended training sessions.

4. **Save the model weights**. This one seems obvious, but it is easy to be lazy and write code in which you cannot save and load previous models. If you want to keep training your model, perform further evaluation, or just run and demonstrate it's capabilities, it is essential that you build this functionality into your code. Many basic libraries will make this easy, but it's up to you to add this to your experimental code to make this seamless.

There are some upfront costs associated with setting up these practices within your code, but much of this code base can also be reused for each project. You have no excuse for not keeping proper records of your experiments!

10.3 Plotting

Bibliography

[1] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[2] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," 2016.

[3] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[4] B. Cheung, J. A. Livezey, A. K. Bansal, and B. A. Olshausen, "Discovering hidden factors of variation in deep networks," *arXiv preprint arXiv:1412.6583*, 2014.

[5] I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner, "Early visual concept learning with unsupervised deep learning," *arXiv preprint arXiv:1606.05579*, 2016.

[6] N. Siddharth, B. Paige, A. Desmaison, V. de Meent, F. Wood, N. D. Goodman, P. Kohli, P. H. Torr, *et al.*, "Inducing interpretable representations with variational autoencoders," *arXiv preprint arXiv:1611.07492*, 2016.

[7] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[9] A. G. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *Proceedings of the 3rd International Conference on Development and Learning*, pp. 112–19, 2004.

[10] S. Mohamed and D. J. Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, pp. 2125–2133, 2015.

[11] R. P. Rao and D. H. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects," *Nature neuroscience*, vol. 2, no. 1, pp. 79–87, 1999.

[12] H. Von Helmholtz, *Handbuch der physiologischen Optik*, vol. 9. Voss, 1867.

[13] K. Friston, "Functional integration and inference in the brain," *Progress in neurobiology*, vol. 68, no. 2, pp. 113–143, 2002.

[14] K. Friston, "Learning and inference in the brain," *Neural Networks*, vol. 16, no. 9, pp. 1325–1352, 2003.

[15] K. Friston, "A theory of cortical responses," *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, vol. 360, no. 1456, pp. 815–836, 2005.

[16] K. J. Friston, L. Harrison, and W. Penny, "Dynamic causal modelling," *Neuroimage*, vol. 19, no. 4, pp. 1273–1302, 2003.

[17] K. J. Friston, "Variational filtering," *NeuroImage*, vol. 41, no. 3, pp. 747–766, 2008.

[18] K. J. Friston, N. Trujillo-Barreto, and J. Daunizeau, "Dem: a variational treatment of dynamic systems," *Neuroimage*, vol. 41, no. 3, pp. 849–885, 2008.

[19] K. Friston, K. Stephan, B. Li, and J. Daunizeau, "Generalised filtering," *Mathematical Problems in Engineering*, vol. 2010, 2010.

[20] H. Feldman and K. J. Friston, "Attention, uncertainty, and free-energy," *Frontiers in human neuroscience*, vol. 4, 2010.

[21] K. Friston, J. Mattout, and J. Kilner, "Action understanding and active inference," *Biological cybernetics*, vol. 104, no. 1, pp. 137–160, 2011.

[22] K. Friston, P. Schwartenbeck, T. FitzGerald, M. Moutoussis, T. Behrens, and R. J. Dolan, "The anatomy of choice: active inference and agency," *Frontiers in human neuroscience*, vol. 7, 2013.

[23] K. Friston, F. Rigoli, D. Ognibene, C. Mathys, T. Fitzgerald, and G. Pezzulo, "Active inference and epistemic value," *Cognitive neuroscience*, vol. 6, no. 4, pp. 187–214, 2015.

[24] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, and G. Pezzulo, "Active inference: A process theory," *Neural computation*, 2016.

[25] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, G. Pezzulo, *et al.*, "Active inference and learning," *Neuroscience & Biobehavioral Reviews*, vol. 68, pp. 862–879, 2016.

[26] A. M. Bastos, W. M. Usrey, R. A. Adams, G. R. Mangun, P. Fries, and K. J. Friston, "Canonical microcircuits for predictive coding," *Neuron*, vol. 76, no. 4, pp. 695–711, 2012.

[27] R. Kanai, Y. Komura, S. Shipp, and K. Friston, "Cerebral hierarchies: predictive processing, precision and the pulvinar," *Phil. Trans. R. Soc. B*, vol. 370, no. 1668, p. 20140169, 2015.

[28] R. Bogacz, "A tutorial on the free-energy framework for modelling perception and learning," *Journal of Mathematical Psychology*, vol. 76, pp. 198–211, 2017.

[29] J. C. Whittington and R. Bogacz, "An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity," *Neural computation*, 2017.

Appendix A

Appendix 1