Chapter 12

Bayesian models and neural networks

Thomas L. Griffiths, Ishita Dasgupta, & Erin Grant

Artificial neural networks have a long history as models of human cognition (McClelland & Rumelhart, 1986), and have undergone a recent resurgence to become a dominant approach to machine learning (LeCun, Bengio, & Hinton, 2015). Given their history and current popularity, understanding how neural networks relate to probabilistic models of cognition is important both for contextualizing this work and developing an understanding of human cognition that draws upon both theoretical traditions.

There are two paths towards establishing connections between Bayesian inference and neural networks: using neural networks as systems for performing Bayesian inference, and thinking about neural networks as just another probabilistic model to which Bayesian inference can be applied. Each path offers distinctive insights that are relevant to understanding human cognition, helping us imagine how human brains could approximate Bayesian inference and how human learning could be guided by the equivalent of prior distributions without anything that looks explicitly like Bayesian inference taking place.

If we follow the first path, the way to integrate Bayesian models of cognition with neural networks is to think about them as operating at different levels of analysis: Bayesian models capture Marr's (1982) computational level, while neural networks address the algorithmic. Neural networks thus complement – and potentially implement – algorithms for approximate Bayesian inference such as the Monte Carlo and variational methods discussed in Chapters 6 and 11, and provide a source of hypotheses about some of the systematic deviations between rational action and human behavior mentioned in Chapter 7.

If we follow the second path, creating what are sometimes referred to as **Bayesian neural networks** (e.g., MacKay, 1995), the concepts that make Bayesian inference a powerful tool for understanding human cognition, such as priors that capture specific inductive biases, are used to understand and constrain neural networks. In doing so, we have the opportunity to explore methods for making neural networks that instantiate inductive biases that are better aligned with those of human learners.

Our goal in this chapter is to chart these two paths, summarizing some of the key ideas that serve as landmarks along the way. The literature on these topics is vast, with many contributions having been made in the last few years, and rather than providing a comprehensive review we focus on the conceptual connections that are valuable for linking these theoretical perspectives in the context of understanding human cognition. However, we also provide pointers to resources that can be used to explore this literature more deeply.

12.1 What is a neural network?

There are many different types of artificial neural networks, unified by the idea of defining a system of simple computational units that interact with one another via weighted connections. These various formalisms are all, to greater or lesser extent, inspired by brains – systems of neurons that interact with one another via synaptic connections. Some artificial neural networks are especially formulated as probabilistic models. For example, Boltzmann machines can be interpreted as a kind of undirected graphical model (see Chapter 4).

For simplicity, in this chapter we will primarily focus on multi-layer perceptrons (for more detailed treatments of these and other neural network architectures see McClelland & Rumelhart, 1986; Goodfellow, Bengio, & Courville, 2016). A multi-layer perception is a "feed-forward" neural network in which computational units – "nodes" – are organized into layers, and information flows from one layer to the next via weighted connections. The activation of the ith node in a given layer of the network, y_i, is determined by its input and an **activation function**. The input is the sum of the activations of all nodes in the previous layer of the network, weighted by the strength of their connections

$$\text{input}_i = \sum_j w_{ji} z_j \tag{12.1}$$

where z_j is the activation of the jth node in the previous layer and w_{ji} is the weight from that node to the ith node in the next layer. These weights can be collected into a **weight matrix W**. Nodes can also have a bias term, w_{0i}, that determines their default activation.

The activation function transforms the input into the activation of the node. This transformation is non-linear, inspired by the way that biological neurons accumulate input until it exceeds a threshold, then fire. This non-linearity also serves a practical purpose, as without it a multi-layer neural network could be expressed as a single linear function. A classic choice for the activation function is the sigmoid

$$g(\text{input}) = \frac{1}{1 + \exp\{-\text{input}\}}, \tag{12.2}$$

but contemporary applications of neural networks use other activation functions that better support learning in neural networks with many layers (e.g., rectified linear units; Nair & Hinton, 2010).

Learning is typically performed by stochastic gradient descent. Each output node y_i in a network has some target value it should produce. We can define a **loss function** $\mathcal{L}(\mathbf{W})$ that captures the difference between the outputs and the targets as a function of the weights $\mathbf{W}$. A standard loss function is the squared-error loss, which we can write for a single observation as

$$\begin{aligned} \mathcal{L}(\mathbf{W}) &= \sum_i (t_i - y_i)^2 & (12.3) \\ &= \sum_i (t_i - g(\sum_j w_{ji} z_j))^2. & (12.4) \end{aligned}$$

The gradient descent algorithm finds weights that reduce this loss by calculating the gradient of the loss – its derivative with respect to the weights – and then moving in the direction that reduces that loss.

If we differentiate this loss function with respect to the weight w_{ji} we obtain

$$\frac{d\mathcal{L}}{dw_{ji}} = -2(t_i - g(\sum_j w_{ji} z_j)) g'(\sum_j w_{ji} z_j) z_j \tag{12.5}$$

via the chain rule for derivatives. Since we want to minimize the loss, we update the weights by moving in the opposite direction of the gradient (ie. going in the direction where the loss goes down). In this case, that means setting w_{ji} to $w_{ji} - \eta \frac{d\mathcal{L}}{dw_{ji}}$, where η is a **learning rate**. Typically we want to minimize the loss over an entire dataset, which would require computing the gradient of the loss function across all of the observations in that dataset, but stochastic gradient descent approximates this by taking the gradient for single observation or a small number of observations at a time. If the learning rates are gradually decreased over time, this algorithm is guaranteed to converge to a local minimum of the loss.

In a multi-layer perceptron, we have many layers of nodes and weights. The network receives inputs x and produces outputs y, with the intermediate layers z being referred to as **hidden layers** as they do not correspond to variables that are observed in the dataset. Stochastic gradient descent can be used to update all of the weights in the network, using the chain rule to calculate the derivative of the last with respect to each weight. This derivative has terms that capture the contribution of each node to the overall loss which can be interpreted as propagating the loss back through the network, resulting in this algorithm being called **backpropagation** (Rumelhart, Hinton, & Wilson, 1986). Contemporary software for training neural networks automatically computes the required derivatives, making it easy to define and train neural networks with arbitrarily complex architectures (e.g., Abadi et al., 2015; Paszke et al., 2019).

12.2 Bayesian inference *by* neural networks

We will begin by considering how neural networks can be used to perform Bayesian inference. Our starting point is a classic observation of equivalence between a simple neural network and a simple Bayesian model. We then consider how other approximation schemes – such as the Monte Carlo and variational methods discussed in Chapter 6 – can be implemented in neural networks.

12.2.1 A simple neural network that performs Bayesian inference

Consider a simple classification problem: we have a set of objects that are drawn from two classes, with each object having d observed binary features $x_1, \ldots, x_d$. We want to define a Bayesian model that captures classification in this setting.

To simplify things, we assume that the features are independent.[1] Letting y denote the class, we have

$$P(x_1, \ldots, x_d|y) = \prod_j P(x_j|y) \tag{12.6}$$

for each class. Since there are only two hypotheses (call them $y = 1$ and $y = 0$) we can write the posterior in odds form,

$$\begin{aligned} \frac{P(y=1|x_1,\ldots,x_d)}{P(y=0|x_1,\ldots,x_d)} &= \frac{P(x_1,\ldots,x_d|y=1)}{P(x_1,\ldots,x_d|y=0)} \frac{P(y=1)}{P(y=0)} && (12.7) \\ &= \frac{P(y=1)}{P(y=0)} \prod_j \frac{P(x_j|y=1)}{P(x_j|y=0)} && (12.8) \end{aligned}$$

and take logarithms to obtain

$$\log \frac{P(y=1|x_1,\ldots,x_d)}{P(y=0|x_1,\ldots,x_d)} = \log \frac{P(y=1)}{P(y=0)} + \sum_j \log \frac{P(x_j|y=1)}{P(x_j|y=0)}. \tag{12.9}$$

If we want to convert the log posterior odds back to a posterior probability, we can do so by exploiting the property of the sigmoid,

$$p = \frac{1}{1 + \exp\{-\log \frac{p}{1-p}\}} \tag{12.10}$$

so

$$P(y=1|x_1,\ldots,x_d) = \frac{1}{1 + \exp\{-\log \frac{P(y=1|x_1,\ldots,x_d)}{P(y=0|x_1,\ldots,x_d)}\}} \tag{12.11}$$

or equivalently

$$P(y=1|x_1,\ldots,x_d) = \frac{1}{1 + \exp\{-\log \frac{P(y=1)}{P(y=0)} - \sum_j \log \frac{P(x_j|y=1)}{P(x_j|y=0)}\}}. \tag{12.12}$$

Now imagine solving the same problem using a simple neural network. In fact, take the simplest such network, with a single output y and no hidden layers – just a set of weights w_j mapping directly from x_j to y. Using a sigmoid activation function, we have

$$y = \frac{1}{1 + \exp\{-w_0 - \sum_j w_{ji} x_j\}} \tag{12.13}$$

[1] This is known as a **naïve Bayes** model, since it makes a naïve assumption about independence, but often performs well in classification settings as it has few parameters to estimate and is hence fairly robust (e.g., Rish, 2001).

where w_0 is an additional weight – known as a **bias** – that is included to modify the value of y when all x_j are 0.

Comparing Equations 12.12 and 12.13 suggests that there might be a relationship between these two models: both are a sigmoid of a linear function. $\log \frac{p(y=1)}{p(y=0)}$ is a single fixed value, which can potentially be captured by w_0. The catch is that $\log \frac{P(x_j|y=1)}{P(x_j|y=0)}$ takes on different values for $x_j = 1$ and $x_j = 0$, while $w_j x_j$ has the value w_j when $x_j = 1$ and 0 when $x_j = 0$. We can accommodate this by defining

$$w_j = \log \frac{P(x_j = 1|y = 1)}{P(x_j = 1|y = 0)} - \log \frac{P(x_j = 0|y = 1)}{P(x_j = 0|y = 0)} \tag{12.14}$$

$$w_0 = \log \frac{P(y = 1)}{P(y = 0)} + \sum_j \log \frac{P(x_j = 0|y = 1)}{P(x_j = 0|y = 0)} \tag{12.15}$$

and then the two models are directly equivalent: the value of the output y in the neural network is $P(y = 1|x_1, \ldots, x_d)$ in the Bayesian model.

This simple example illustrates how two models that start in quite different places can end up being formally equivalent, and how it is possible for neural networks to directly implement Bayesian inference. One interesting difference between these approaches is in how they formulate the problem. In the Bayesian approach, we start with a **generative model** specifying how features are generated based on the class and then use Bayes rule to work backwards from features to class labels. The neural network starts with the inverse problem, learning a function that maps from features to class labels directly. This is called a **discriminative model**. These approaches make quite different assumptions about the properties of the data – the generative approach explicitly models the distribution of the features, while the discriminative approach only models the distribution of the class labels. These different assumptions can have implications for the way that the observed data are interpreted, and in particular how missing data are handled (Hsu & Griffiths, 2009). However, generative-discriminative pairs like the relationship between naïve Bayes and a one-layer neural network (also known as **logistic regression**) show that there is significant potential for bridging these two perspectives (for more discussion of this point see Efron, 1975; Ng & Jordan, 2001)

12.2.2 A neural implementation of importance sampling

The structure of a neural network can also be used to implement approximate algorithms for Bayesian inference. In this section we illustrate this by showing how importance sampling – one of the Monte Carlo algorithms introduced in Chapter 6 – can be implemented in a simple neural network (Shi & Griffiths, 2009).

Assume a generative model in which observations x are generated from a Gaussian distribution centered on an unknown value z. We have a prior distribution over these unknown values $p(z)$, and want to compute the expectation of a function $f(z)$ over the posterior distribution $p(z|x)$, $E[f(z)|x] = \int f(z)p(z|x)\,dz$. For example, taking $f(z) = z$ would allow us to compute the posterior mean.

Evaluating expectations over the posterior distribution can be challenging: it requires computing the posterior and potentially a multidimensional integration. The expectation $E[f(z)|x]$ can be approximated using importance sampling. Recall from Chapter 6 that importance sampling approximates an expectation over the posterior by using a set of samples from some surrogate distribution $q(z)$ and assigning those samples weights proportional to the ratio $p(z|x)/q(z)$. We then have

$$E[f(z)|x] = \int f(z)\frac{p(z|x)}{q(z)}q(z)dz \simeq \sum_j f(z_j)\frac{p(z_j|x)}{q(z_j)} \qquad z_j \sim q(z). \tag{12.16}$$

If we choose $q(z)$ to be the prior $p(z)$, the weights reduce to the likelihood $p(x|z)$, giving

$$E[f(z)|x] \simeq \frac{\sum_{z_j} f(z_j)p(x|z_j)}{\sum_{z_j} p(x|z_j)} \qquad z_j \sim p(z) \tag{12.17}$$

which is the likelihood weighting algorithm discussed in Chapter 6, in which we approximate the posterior distribution with samples from the prior weighted by the likelihood.

We will construct an analogue of this algorithm using a particular kind of neural network – a **radial basis function (RBF)** network. This is a multi-layer neural network architecture in which the hidden units are parameterized by locations in a latent space z_j. On presentation of a stimulus x, these hidden units are activated according to a function that depends only on the distance $||x - z_j||$, e.g., $\exp(-|x - z_j|^2/2\sigma^2)$. RBF networks are popular because they have a simple structure with a clear interpretation and are easy to train, and they have been used as models of pattern recognition in neuroscience (Kouh & Poggio, 2008) and category learning in psychology (Kruschke, 1992).

Implementing importance sampling with RBF networks is straightforward. We create a network where the inputs correspond to x and the single output node approximates $E[f(z)|x]$. Each hidden unit represents a stored value z_j that is sampled from the prior. The activation function is taken to be proportional to $p(x|z_j)$. After the activations are computed for all of the hidden units they are normalized. The weight from hidden unit j to the output unit is set to $f(z_j)$. Such a network produces output exactly in the form of Equation 12.17. The same set of samples from the prior can be used to perform inference for any x, so this network instantiates a simple neural circuit for approximating the posterior mean.

This importance sampler is a neural network implementation of the exemplar model for Bayesian inference discussed in Chapter 11 (Shi, Griffiths, Feldman, & Sanborn, 2010). The exemplars correspond to the z_j represented by the hidden units. Given this insight, it is straightforward to define other neural network architectures that can approximate Bayesian inference by memorizing and generalizing from exemplars. For example, Abbott, Hamrick, and Griffiths (2013) showed how the Sparse Distributed Memory architecture of Kanerva (1988) can be used to perform Bayesian inference.

12.2.3 Learning to perform Bayesian inference

In Chapters 5 and 6, we saw that Bayesian inference in latent variable models can be very computationally expensive, and often intractable. More specifically, the posterior distribution $P(z|x)$ over a latent variable z given some data x often cannot be computed exactly and requires us to make approximations using methods like Monte Carlo or variational inferenc). These approximations are expensive to compute, requiring drawing many samples for Monte Carlo, or many optimization steps for variational inference.

One way to reduce the cost of probabilistic inference is to try to store and re-use computations wherever possible. When we compute a distribution $Q_x(z)$ that approximates the posterior $P(z|x)$ for some x, we can and simply store and re-use this previous estimate when we encounter x again. However, this quickly becomes infeasible. For example, when we have a large number of possible observations x, the chance of seeing the same x twice is small. We can re-use an inference only when we encounter the exact same x but have to start from scratch when we encounter observations that are *similar* but not exactly the same.

We can formulate learning how to approximate $P(z|x)$ as a problem to be solved by neural networks. In particular, we can learn a function that takes in x and produces (an estimate of) $Q_x(z)$, by learning from a database of $(x, Q_x(z))$ pairs generated by exact Bayesian inference. This is called **amortization**, since the costs of doing a new inference are spread out (or amortized) over several previous inferences (for a review of amortization in the context of cognitive science, see Dasgupta & Gershman, 2021). The network that implements this function is called a **recognition network** or an **inference network**. We

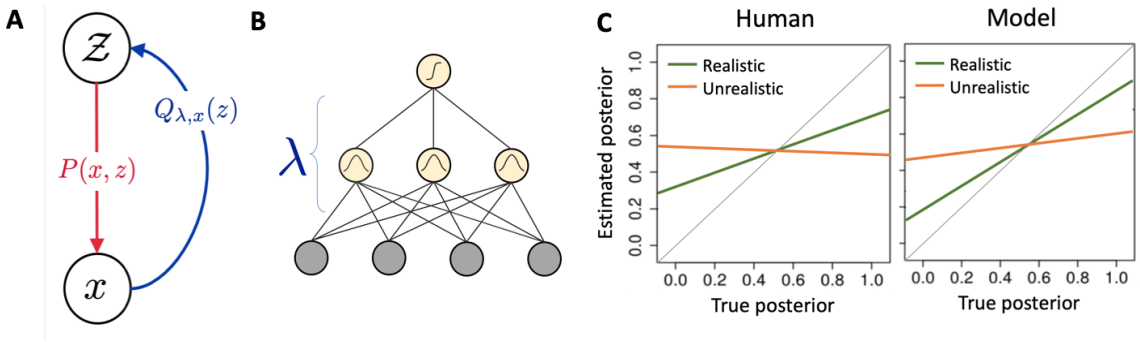


Figure 12.1: Amortizing posterior inference. (A) $P(x, z)$ is a generative model that produces latent variables z, and data x given z. $Q_{\lambda,x}(z)$ is an inference network, parameterized by λ, that maps observable data to the posterior distribution over underlying latent variables. (B) The inference network. (C) Humans are better at Bayesian inference when the provided probabilities are believable (replotted from (Cohen et al., 2017)). The same pattern arises in a recognition network with a small number of hidden units (replotted from (Dasgupta et al., 2018)). Figure adapted from Dasgupta and Gershman (2021).

refer to the estimate of the function representing the approximate posterior as $\tilde{Q}$. Note that this estimate contains two sources of error from the true P – the error inherent in the initial approximation of the posterior ($P \rightarrow Q$, called the **approximation error**) and the error from an imperfect inference network ($Q \rightarrow \tilde{Q}$, the **amortization error**).

To make this more concrete, we discuss amortization in the context of a variational approximation. As discussed in Chapter 6, a variational approximation $Q_\lambda(z)$ (where λ are the parameters of the distribution) to a true posterior $P(z|x)$ can be derived by maximizing the evidence lower bound (ELBO). Maximizing the ELBO is equivalent to minimizing the KL divergence between $Q_\lambda(z)$ and $P(z|x)$. For standard variational inference, this is computed for a given x, that is, for a given set of observed data. When we observe new data x', we have to recompute Q from scratch. The key conceptual leap for amortizing this computation is that we can instead learn a $\tilde{Q}_{\lambda,x}(z)$ that is a function of x. We can do this by learning a function mapping from x to some parameters that uniquely identify a $\tilde{Q}$ (e.g., the mean and variance of a Gaussian distribution; Figure 12.1). The parameters λ are optimized to minimize the KL (or in actual practice, maximize the ELBO) in expectation over the distribution of x

$$E_{P(x)}\left[D_{KL}(\tilde{Q}_{\lambda,x}(z)||P(z|x))\right] \tag{12.18}$$

where the x come from some query distribution $p(x)$. This optimization is easily done with gradient descent (for mathematical details, see Ranganath, Gerrish, & Blei, 2014).

Evidence that humans actually amortize inferences comes from investigating a core prediction of amortization – that past inferences influence future inferences by imprinting themselves onto the parameters of the recognition network. Since we are minimizing the KL divergence in expectation over the query distribution, frequent queries will be prioritized over less frequent ones and $\tilde{Q}$ will more closely approximate the true posterior for these frequent queries. In Dasgupta, Schulz, Tenenbaum, and Gershman (2020), this is used to model findings from human probabilistic inference showing that people are much better at Bayesian inferences when the probabilities are realistic, and the problem are embedded in believable real-world scenarios (Evans, Handley, Over, & Perham, 2002; Cohen et al., 2017). Concretely, a neural network with two hidden units in a single hidden layer with a radial basis activation function is trained as the inference network on a distribution of queries. A query consists of a prior and likelihood (both Bernoulli parameters in the example) and data (a sample from a Bernoulli distribution) that are sampled such that they result in a posterior sampled from a fixed distribution. The model is then tested on either

the same distribution of posteriors or a different distribution of posteriors, emulating the difference between believable (previously encountered) or unbelievable (unfamiliar) posteriors. This model performs significantly better on the believable than the unbelievable distribution, reflecting human behavior (Figure 12.1C). This finding refutes the view that the brain relies on a general-purpose inference engine that operates equally well on arbitrary probabilities, whereas a model where humans amortize and reuse past inferences captures these effects. A series of other findings in human behavior that can be explained by amortized inference are detailed in Dasgupta and Gershman (2021).

Another important use case for inference networks in cognitive science is as components of models that implement Bayesian inference in complex domains where inference would otherwise be intractably expensive. A notable example is in the probabilistic programming methods discussed in Chapter 18.

12.3 Bayesian inference *for* neural networks

The second path for developing correspondences between neural networks and Bayesian methods is based on treating neural networks as probabilistic models, and using Bayesian methods to estimate the parameters of these models. Since neural networks are typically large and complex, performing Bayesian inference in this setting can be challenging. However, some of the algorithms that are used to train neural networks already admit a Bayesian interpretation. In this section we first outline the Bayesian perspective on neural networks and then discuss how this relates to neural network learning algorithms, ultimately highlighting a surprising connection to hierarchical Bayesian inference.

12.3.1 Bayesian neural networks

A neural network can be viewed as a probabilistic model. For example, a multi-layer perceptron specifies a function that can be interpreted as a mapping from its input to a probability distribution over outputs. The parameters that define this mapping are the weights and biases of the network. Using θ to denote these parameters, we can define $p(t|\mathbf{x}, \theta)$ to be the probability given to the target value t by the network given input $\mathbf{x}$. A training set d consists of many $(t, \mathbf{x})$ pairs, and training the neural network yields parameters θ that minimize a loss function over this dataset. This training process also has a probabilistic interpretation.

Exactly how we specify $p(t|\mathbf{x}, \theta)$ depends on the nature of t. If t is continuous, then we can assume that $p(t|\mathbf{x}, \theta)$ is a Gaussian distribution with variance σ_t^2 centered on the output of the network y,

$$p(t|\mathbf{x}, \theta) = \frac{1}{2\sigma_t^2} \exp\{-(t-y)^2/2\sigma_t^2\}. \tag{12.19}$$

The part of this expression that is affected by θ is $-(t-y)^2$, so maximizing $p(d|\theta)$ is equivalent to minimizing the squared-error loss between t and y (Equation 12.3). If t is discrete, then it makes sense to use the cross-entropy loss, where we score the network's predictions for each output by $-\log p(t|\mathbf{x}, \theta)$. Minimizing this loss is equivalent to maximizing the likelihood $p(d|\theta)$. Given the correspondence between traditional methods for training neural networks and maximum likelihood estimation, it is natural to consider Bayesian approaches to neural network learning.

In this context, the Bayesian approach requires defining a prior distribution on the parameters of the neural network, $p(\theta)$. This implicitly defines a prior distribution over the functions that the neural network represents. Bayesian inference would involve calculating the posterior distribution $p(\theta|d)$, or finding the maximum a posteriori (MAP) value of θ rather than the maximum likelihood estimate. There are many ways we can imagine defining such a prior, but the simplest is to assume that the weights and biases of the neural network are drawn from a Gaussian distribution with zero mean and variance σ_w^2.

Actually computing posterior distributions over the weights of a neural network is potentially extremely computationally costly, as neural networks typically have very large numbers of weights and we are not able to rely upon conjugate priors or the other tricks that are used to make Bayesian inference tractable. However, there is one interesting case where increasing the size of neural networks turns out to be beneficial: Neal (1993) showed that in the limit of infinitely many hidden units, a Bayesian multi-layer perceptron becomes a Gaussian process with a specific kernel defined by the activation function of the hidden units (see Chapter 9 for a more detailed discussion of Gaussian processes). This deep connection between neural networks and nonparametric Bayesian statistics makes it possible to use models inspired by neural networks but retain desirable characteristics of probabilistic models, such as being able to express different degrees of uncertainty in their predictions.

12.3.2 Implicit priors and learning algorithms

While full Bayesian inference is typically intractable for neural networks, finding approximations to the MAP estimate of θ can be relatively straightforward. In fact, existing algorithms for training neural networks have been shown to correspond to MAP inference under specific prior distributions.

Consider the Gaussian prior on the weights of a neural network mentioned in the previous section. Using this prior, we can revisit the simple one-layer network we used to introduce the gradient descent algorithm. In this case, the parameters of the network are just the weight matrix $\mathbf{W}$. To perform Bayesian inference, we need to define the likelihood $p(d|\mathbf{W})$ and the prior $p(\mathbf{W})$. If we focus on a single observation with a set of target outputs t_i, our likelihood is

$$p(d|\mathbf{W}) = \prod_i \frac{1}{2\sigma_t^2} \exp\{-(t_i - y_i)^2/2\sigma_t^2\} \tag{12.20}$$

$$\propto \exp\left\{-\frac{1}{2\sigma_t^2}\sum_i (t_i - y_i)^2\right\}. \tag{12.21}$$

Assuming a Gaussian prior with zero mean on each w_{ji} we have

$$p(\mathbf{W}) = \prod_{ij} \frac{1}{2\sigma_w^2} \exp\{-w_{ji}^2/2\sigma_w^2\} \tag{12.22}$$

$$\propto \exp\left\{-\frac{1}{2\sigma_w^2}\sum_{ij} w_{ji}^2\right\}. \tag{12.23}$$

The posterior probability $p(\mathbf{W}|d)$ is proportional to $p(d|\mathbf{W})p(\mathbf{W})$. Since we just care about maximizing $p(\mathbf{W}|d)$, we can focus on the log posterior probability, which is $\log p(\mathbf{W}|d) = \log p(d|\mathbf{W}) + \log p(\mathbf{W})$. Taking the logarithms of the expressions above and summing, we have

$$\log p(\mathbf{W}|d) = -\frac{1}{2\sigma_t^2}\sum_i -(t_i - y_i)^2 - \frac{1}{2\sigma_w^2}\sum_{ij} w_{ji}^2 + C \tag{12.24}$$

where C is a constant that does not depend on $\mathbf{W}$. Multiplying this by a scalar doesn't change the optimal value of $\mathbf{W}$ so we can multiply by $2\sigma_t^2$ and change the sign to obtain

$$\arg\max_{\mathbf{W}} \log p(\mathbf{W}|d) = \arg\min_{\mathbf{w}} \left[\sum_i (t_i - y_i)^2 + \frac{\sigma_t^2}{\sigma_w^2}\sum_{ij} w_{ji}^2\right] \tag{12.25}$$

where the first term on the right hand side is easily recognized as $\mathcal{L}(\mathbf{W})$ from Equation 12.3. Consistent with results in previous chapters, the MAP solution can thus be interpreted as adding an additional regularization term to the function optimized for the maximum-likelihood estimate.

Differentiating Equation 12.25 with respect to w_{ji} yields $\frac{d\mathcal{L}}{dw_{ji}} + 2\frac{\sigma_t^2}{\sigma_w^2}w_{ji}$. If we apply the gradient descent algorithm, the weight update rule becomes

$$w_{ji} = w_{ji} - \eta\left(\frac{d\mathcal{L}}{dw_{ji}} + 2\frac{\sigma_t^2}{\sigma_w^2}w_{ji}\right) \tag{12.26}$$

$$= w_{ji}\left(1 - 2\eta\frac{\sigma_t^2}{\sigma_w^2}\right) - \eta\frac{d\mathcal{L}}{dw_{ji}}. \tag{12.27}$$

The second term on the right hand side is just the standard weight update from gradient descent. The Bayesian version of this algorithm introduces the first term, which "shrinks" w_{ji} towards zero with each weight update.

This idea of reducing w_{ji} with each weight update was independently developed in the neural network research community, where it goes by the name **weight decay** (Hanson & Pratt, 1988). It helps to stop weights from becoming overly large during training, and implicitly has the same effect as assuming a Gaussian prior on those weights. Weight decay is easy to implement, and converges to a local maximum of the posterior under the same conditions that allow gradient descent to converge to a local minimum of the loss.

The choice of a regularizer such as weight decay is one of many choices to be made when setting up a neural network model—we must also decide on a neural network architecture (how many layers, how many hidden units, what activation functions to use), the hyperparameters of the learning algorithm (such as the learning rate and the schedule on which it is modified), and schemes to initialize the parameters of the model. Surprisingly, we can show that many of these choices can be interpreted in terms of implicitly defining different priors. One such choice is the number of gradient descent steps used to optimize the parameters of the neural network model. It can be shown that stopping the optimization early, at t steps, is equivalent in special cases to fully optimizing a regularized loss in which the regularization penalty scales as $1/t$ (Santos, 1996; Ali, Kolter, & Tibshirani, 2019); in other words, the regularization penalty diminishes as the number of iterations of gradient descent increases. As in weight decay, the regularizer can be seen an implicit Gaussian prior on the weights of the neural network, here with a variance that scales with the number of iterations of training t.

Contemporary algorithms used for training deep neural networks have also been suggested as having connections to Bayesian inference. For example, the **dropout** algorithm (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014), in which a subset of weights are not included in each weight update, has been connected to the idea of having a distribution over the parameters θ, and in some cases this distribution can be shown to align with the Bayesian posterior (Gal & Ghahramani, 2016). Even the stochastic gradient descent algorithm itself has been characterized as performing approximate Bayesian inference (Mandt, Hoffman, & Blei, 2017).

12.3.3 Meta-learning and hierarchical Bayes

The previous section introduced the idea of implicit priors that result from an algorithmic choice such as the form of a regularizer or the early stopping iteration. These priors express relatively simple preferences for models that have weights that are close to zero, or close to their initial state. However, there are settings in which we would like the prior distribution over a model itself to depend on data. One such is the setting of meta-learning, where a learner is not presented with a single task such as learning a particular concept, but with many tasks that all have a similar character (Schmidhuber, 1987; Thrun & Pratt, 2012). The ideal learner in this setting makes use of commonalities across these tasks in order not only to become better at solving each individual task, but also to solve future tasks better and more quickly, effectively "learning to learn."

We can consider a straightforward way to implement a meta-learner in the context of the one-layer neural network from before. Recall that this model has a matrix of weights $\mathbf{W}$, which we tune by taking steps to solve the problem $\min_{\mathbf{W}} \mathcal{L}(\mathbf{W})$, where $\mathcal{L}(\mathbf{W})$ is a loss function such as the one from Equation 12.3. The meta-learning setting captures the case in which we have multiple losses, $\mathcal{L}_1, \mathcal{L}_2, \ldots$ to be simultaneously minimized:

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) = \min_{\mathbf{W}} \sum_i \mathcal{L}_i(\mathbf{W}). \tag{12.28}$$

These individual losses might represent, for example, losses on subgroups of data that are more similar within subgroups than across subgroups, or losses corresponding to different types of tasks. Rather than using a single set of weights $\mathbf{W}$ for all losses—which would correspond, in our example, to a single network for all of the losses—a meta-learning algorithm allows a separate set of weights $\mathbf{W}_i$ for each loss $\mathcal{L}_i$, and relates the weights $\mathbf{W}_i$ by global parameters θ. With this parameterization, the objective becomes

$$\min_{\theta} \mathcal{L}(\theta) = \min_{\theta} \sum_i \mathcal{L}_i(\mathbf{W}_i(\theta)) \ . \tag{12.29}$$

Each set of weights $\mathbf{W}_i$ in (12.29) is not individually learned as in Section 12.1, but are somehow derived from the global parameters θ; this setup allows the weights of the individual models $\mathbf{W}_i$ to be adapted for each loss while capturing information that is redundant across the losses via their dependence on the global parameters θ.

There are various ways in which the individual model weights $\mathbf{W}_i$ could relate to the global parameters θ. One simple way to set up a meta-learning algorithm is by taking θ to be the weight initialization for optimizing $\mathbf{W}_i$ with gradient descent on each of the individual losses (Finn, Abbeel, & Levine, 2017). Since the gradient descent algorithm is differentiable with respect to its parameter initialization, we can treat the whole procedure as a computational graph in which we backpropagate the error corresponding to the *final* value of each loss $\mathcal{L}_i$ all the way to the parameter initialization. If we truncate each optimization of the $\mathbf{W}_i$ to a fixed number of gradient descent steps, this meta-learning objective takes a simple form; in the case of one step ($t = 1$), we can write it as

$$\min_{\theta} \mathcal{L}(\theta) = \min_{\theta} \sum_i \mathcal{L}_i \left(\theta - \eta \frac{d\mathcal{L}_i}{d\theta} \right) \ , \tag{12.30}$$

which means that for each loss $\mathcal{L}_i$, we evaluate the loss of the weights $\mathbf{W}_i = \theta - \eta \frac{d\mathcal{L}_i}{d\theta}$, and use this to tune θ using backpropagation and gradient descent just as in Section 12.1.

Under certain conditions, the effect of early stopping—using a fixed and small number of gradient descent steps in the weights $\mathbf{W}_i$—in this meta-learning algorithm is the same as in the Section 12.3.2 in that it corresponds to a Gaussian prior with variance proportional to t over the weights $\mathbf{W}_i$. However, in contrast to the previous section, the mean of this prior is at the initialization parameter, θ. The meta-learning objective above thus learns the mean of a Gaussian prior over the weights $\mathbf{W}_i$ that can be used for MAP inference on a new dataset (Grant, Finn, Levine, Darrell, & Griffiths, 2018); Figure (12.2) visualizes this perspective.

Estimating the parameters of a prior via meta-learning in this way is an instance of hierarchical Bayes (see Chapter 8). While Bayesian inference indicates how a learner should integrate data with a prior distribution over hypotheses, a hierarchical Bayesian model learns that prior distribution. This idea has been widely used in Bayesian models of cognition, and in examples we have considered throughout this book. For example, hierarchical Bayes can be used to learn which properties of objects words tend to label (such as shape) while learning the meaning of individual words (Kemp, Perfors, & Tenenbaum, 2007), and to identify different kinds of causal relationships while learning those relationships (Mansinghka, Kemp, Tenenbaum, & Griffiths, 2006).

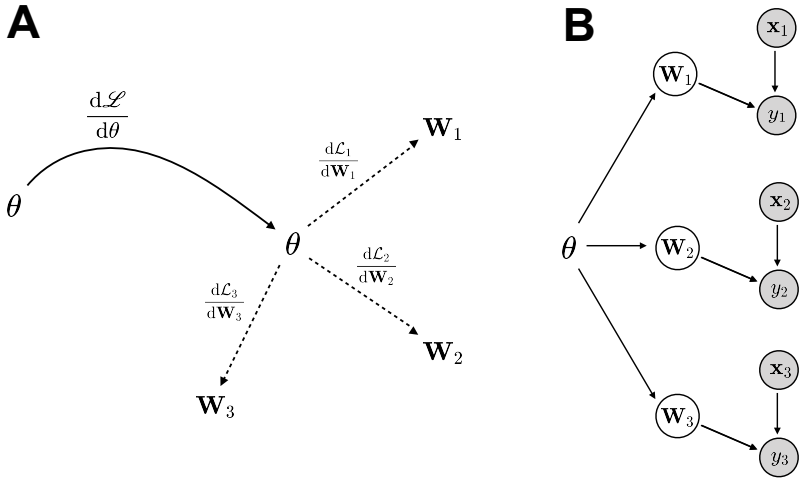


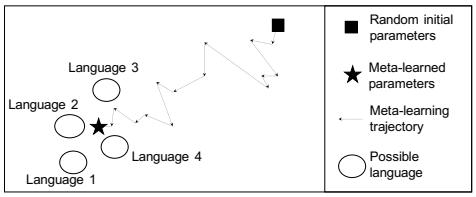
Figure 12.2: Meta-learning and hierarchical Bayes. (**A**) The gradient-based meta-learning algorithm of (Finn et al., 2017) optimizes via $\mathcal{L}$ the parameters θ of a set of models so that when one or a few gradient descent steps are taken from the initialization at θ using the loss $\mathcal{L}_i$, each model obtains new weights $\mathbf{W}_i$ that result in good generalization performance on examples $\mathbf{x}_i, y_i$ associated with that loss. (**B**) The probabilistic graphical model for which the algorithm in (**A**) provides a parameter estimation procedure (Grant et al., 2018). Each task-specific set of weights $\mathbf{W}_i$ is distinct from but influences the estimation of the others through the parameters of a prior θ shared across all $\mathbf{W}_i$. Figure adapted from Grant et al. (2018).

Hierarchical Bayesian models have been used extensively in cognitive science, but the computational costs involved can make them difficult to use for models outside specific classes (e.g., where conjugacy applies; see Chapter 3). Consequently, establishing a link between hierarchical Bayes and metalearning – which can be implemented efficiently for a wide range of models with continuous parameterizations, such as neural networks – potentially expands the scope of Bayesian modeling. For example, (McCoy, Grant, Smolensky, Griffiths, & Linzen, 2020) demonstrated that the meta-learning algorithm described above can be used to create neural networks with an implicit prior distribution that makes it easy to learn languages from a simplified language typology; this can be viewed as a step towards a neural network instantiation of a "universal grammar" that supports language learning (see Figure 12.3).

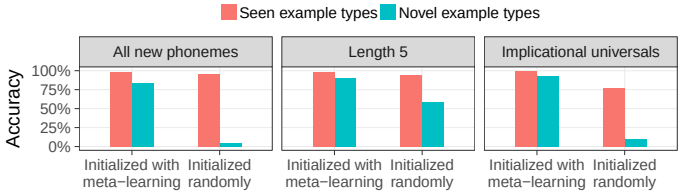
12.4 Future directions

Research on deep learning continues to develop rapidly, and there are many further innovations in amortized inference, Bayesian neural networks, and meta-learning that have yet to be absorbed into cognitive science. All of these topics provide fertile ground for innovation in developing probabilistic models of cognition an understanding how human minds and brains might deal with the computational challenges of Bayesian inference.

The development of novel neural network methods also offers the opportunity to push the limits of the kinds of inferences that these models are able to capture. Memory-based meta-learning is a recent approach where an algorithm for sequentially updating the state of a neural network is learned directly from data. Using this approach, meta-learned agents can solve problems that have traditionally been addressed using structured probabilistic models, including Bayesian inference (Mikulik et al., 2020), model-based reinforcement learning (Wang et al., 2016) and causal learning (Dasgupta et al., 2019). These models are fully amortized and therefore very efficient at run-time, and adapt very well to structure in



(a) Meta-learning finds a parameter initialization from which a model can acquire any language in the typology.



(b) A model initialized with meta-learning can acquire language features unattested in the training set while the randomly initialized model cannot, suggesting that meta-learning has imposed a universal inductive bias for learning languages in the typology.

Figure 12.3: Identifying linguistic inductive biases with metalearning. (a) The size and shape of syllables across the world's languages follow strong cross-linguistic tendencies that suggest universal constraints on human language learning. McCoy et al. (2020) used meta-learning to investigate such inductive biases by analyzing the initial state of a language model trained with meta-learning on a dataset that reflects the typology of natural language syllables). (b) Analysis of the initial state revealed, for example, a prior for implicational universals—that certain input-output mappings in a language imply other input-output mappings—which are widely attested in the syllable structure of natural languages. Figure adapted from McCoy et al. (2020).

the environment that is hard to express in explicit structured models. However, these models also inherit the same issues as neural networks in general – they require large amounts of data to learn from and can generalize poorly. Evaluating the capacities and limits of these models is an important topic for future work (for preliminary steps in this direction see Kumar, Dasgupta, Cohen, Daw, & Griffiths, 2021).

In addition to Bayesian inference *by* neural networks and Bayesian inference *for* neural networks, another productive direction to explore is Bayesian inference *as a model of* neural networks. With the ever-increasing complexity of deep learning models, it becomes harder and harder to understand the implicit assumptions that underlie these models. Our analysis of the inductive biases of neural networks in the previous section relied on simplifying assumptions about the structure of the model and the form of the loss function due to the complexity of analyzing even simple neural networks with one hidden layer. This raises addressing possibility: are artificial neural networks themselves complex enough that we can build cognitive models of them that can help us understand them? Li, Grant, and Griffiths (2021) explored this perspective by using a Bayesian model that was previously used to study inductive biases in human function learning (Wilson, Dann, Lucas, & Xing, 2015) to interpret and make predictions about the inductive biases of neural network models.

12.5 Conclusion

Rather than being competing frameworks for understanding human cognition, we view probabilistic models and neural networks as providing complementary insights that can be used for reverse-engineering the mind. These two approaches are at different levels of analysis and have different strengths and weaknesses: probabilistic models provide a powerful set of tools for abstractly characterizing human inductive biases, particularly in cases where those inductive bases are concisely expressed in terms of structured representations; neural networks are a flexible framework for understanding how efficient approximations to Bayesian inference can be learned from data, often making it possible to engage with problems at scales that go beyond the current limits of probabilistic models. Together, these two approaches provide a toolkit for building models that engage with a wide range of questions about human

cognition.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* (Software available from tensorflow.org)

Abbott, J., Hamrick, J., & Griffiths, T. (2013). Approximating Bayesian inference with a sparse distributed memory system. In *Proceedings of the 35th Annual Meeting of the Cognitive Science Society.*

Ali, A., Kolter, J. Z., & Tibshirani, R. J. (2019). A continuous-time view of early stopping for least squares regression. In *The 22nd International Conference on Artificial Intelligence and Statistics* (pp. 1370–1378).

Cohen, A. L., Sidlowski, S., & Staub, A. (2017). Beliefs and Bayesian reasoning. *Psychonomic Bulletin & Review*, *24*(3), 972–978.

Dasgupta, I., & Gershman, S. J. (2021). Memory as a computational resource. *Trends in Cognitive Sciences*, *25*(3), 240–251.

Dasgupta, I., Schulz, E., Tenenbaum, J. B., & Gershman, S. J. (2020). A theory of learning to infer. *Psychological Review*, *127*(3), 412-441.

Dasgupta, I., Smith, K. A., Schulz, E., Tenenbaum, J. B., & Gershman, S. J. (2018). Learning to act by integrating mental simulations and physical experiments. *bioRxiv*, 321497.

Dasgupta, I., Wang, J., Chiappa, S., Mitrovic, J., Ortega, P., Raposo, D., Hughes, E., Battaglia, P., Botvinick, M., & Kurth-Nelson, Z. (2019). Causal reasoning from meta-reinforcement learning. *arXiv preprint arXiv:1901.08162*.

Efron, B. (1975). The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association*, *70*(352), 892–898.

Evans, J. S. B., Handley, S. J., Over, D. E., & Perham, N. (2002). Background beliefs in Bayesian inference. *Memory & Cognition*, *30*(2), 179–190.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning* (pp. 1050–1059).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press.

Grant, E., Finn, C., Levine, S., Darrell, T., & Griffiths, T. L. (2018). Recasting gradient-based meta-learning as hierarchical Bayes. In *Proceeedings of the International Conference on Learning Representations (ICLR).*

Hanson, S., & Pratt, L. (1988). Comparing biases for minimal network construction with back-propagation. *Advances in Neural Information Processing Systems.*

Hsu, A., & Griffiths, T. L. (2009). Differential use of implicit negative evidence in generative and discriminative language learning. In *Advances in Neural Information Processing Systems 22* (p. 754-762).

Kanerva, P. (1988). *Sparse distributied memory.* MIT Press.

Kemp, C., Perfors, A., & Tenenbaum, J. B. (2007). Learning overhypotheses with hierarchical Bayesian models. *Developmental Science*, *10*(3), 307–321.

Kouh, M., & Poggio, T. (2008). A canonical neural circuit for cortical nonlinear operations. *Neural Computation*, *20*(6), 1427–1451.

Kruschke, J. K. (1992). ALCOVE: an exemplar-based connectionist model of category learning. *Psychological Review*, *99*, 22-44.

Kumar, S., Dasgupta, I., Cohen, J. D., Daw, N. D., & Griffiths, T. L. (2021). Meta-learning of structured task distributions in humans and machines. In *Proceedings of the 9th International Conference on Learning Representations (ICLR).*

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444.

Li, M. Y., Grant, E., & Griffiths, T. L. (2021). Meta-learning inductive biases of learning systems with gaussian processes. In *Fifth Workshop on Meta-learning at the Conference on Neural Information Processing Systems.*

MacKay, D. J. (1995). Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, *354*(1), 73–80.

Mandt, S., Hoffman, M. D., & Blei, D. M. (2017). Stochastic gradient descent as approximate Bayesian inference. *Journal of Machine Learning Research*, *18*, 1–35.

Mansinghka, V. K., Kemp, C., Tenenbaum, J. B., & Griffiths, T. L. (2006). Structured priors for structure learning. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence* (pp. 324–331).

Marr, D. (1982). *Vision.* W. H. Freeman.

McClelland, J., & Rumelhart, D. (Eds.). (1986). *Parallel distributed processing: Explorations in the microstructure of cognition.* MIT Press.

McCoy, R. T., Grant, E., Smolensky, P., Griffiths, T. L., & Linzen, T. (2020). Universal linguistic inductive biases via meta-learning. In *Proceedings of the 42nd Annual Meeting of the Cognitive Science Society.*

Mikulik, V., Delétang, G., McGrath, T., Genewein, T., Martic, M., Legg, S., & Ortega, P. (2020). Meta-trained agents implement bayes-optimal agents. *Advances in Neural Information Processing Systems*, *33*, 18691–18703.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the International Conference on Machine Learning.*

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods* (Tech. Rep. No. CRG-TR-93-1). University of Toronto.

Ng, A., & Jordan, M. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. *Advances in Neural Information Processing Systems 14.*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035).

Ranganath, R., Gerrish, S., & Blei, D. (2014). Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics* (pp. 814–822).

Rish, I. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence* (Vol. 3, pp. 41–46).

Rumelhart, D. E., Hinton, G. E., & Wilson, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533-536.

Santos, R. J. (1996). Equivalence of regularization and truncated iteration for general ill-posed problems. *Linear Algebra and its Applications*, *236*, 25–33.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning.* Unpublished doctoral dissertation, Institut für Informatik, Technische Universität München.

Shi, L., & Griffiths, T. (2009). Neural implementation of hierarchical Bayesian inference by importance sampling. In *Advances in Neural Information Processing Systems 22* (pp. 1669–1677).

Shi, L., Griffiths, T. L., Feldman, N. H., & Sanborn, A. N. (2010). Exemplar models as a mechanism for performing Bayesian inference. *Psychological Bulletin and Review*, *17*, 443-464.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Thrun, S., & Pratt, L. (2012). *Learning to learn.* Springer.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., & Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763.*

Wilson, A. G., Dann, C., Lucas, C., & Xing, E. P. (2015). The human kernel. In *Advances in Neural Information Processing Systems 28.*