Artificial neural networks for neuroscientists: A primer

Guangyu Robert Yang[1,*], **Xiao-Jing Wang**[2,*]
[1] Center for Theoretical Neuroscience, Columbia University
[2] Center for Neural Science, New York University
[*] Correspondence robert.yang@columbia.edu, xjwang@nyu.edu

Abstract

Artificial neural networks (ANNs) are essential tools in machine learning that are increasingly used for building computational models in neuroscience. Besides being powerful techniques for data analysis, ANNs provide a new approach for neuroscientists to build models that capture complex behaviors, neural activity and connectivity, as well as to explore optimization in neural systems. In this pedagogical Primer, we introduce conventional ANNs and demonstrate how they have been deployed to study neuroscience questions. Next, we detail how to customize the analysis, structure, and learning of ANNs to better address a wide range of challenges in brain research. To help the readers garner hands-on experience, this Primer is accompanied with tutorial-style code in PyTorch and Jupyter Notebook, covering major topics.

1 Artificial neural networks in neuroscience

Artificial neural networks (ANNs), or deep neural networks, have emerged as a dominant framework in machine learning (ML) nowadays [LeCun et al., 2015], leading to breakthroughs across a wide range of applications, including computer vision [Krizhevsky et al., 2012], natural language processing [Devlin et al., 2018], and strategic games [Silver et al., 2017]. Some key ideas in this field can be traced to brain research: supervised learning rules have its root in the theory of training perceptrons which in turn was inspired by the brain [Rosenblatt, 1962]; the hierarchical architecture [Fukushima and Miyake, 1982] and convolutional principle [LeCun and Bengio, 1995] were closely linked to our knowledge about the primate visual system [Hubel and Wiesel, 1962, Felleman and Van Essen, 1991]. Today, there is a continued exchange of ideas from neuroscience to the field artificial intelligence [Hassabis et al., 2017].

At the same time, machine learning offers new and powerful tools for systems neuroscience. One utility of the deep learning framework is to analyze neuroscientific data (Figure 1A). In particular, the advances in computer vision makes convolutional neural networks useful for processing a range of image and video data. Behaviors over time, for instance micro-movements of animals in a laboratory experiment, are

Preprint.

complex but can be tracked and quantified efficiently with the help of deep neural networks [Mathis et al., 2018]. With the advances of neurotechnologies, there is a deluge of big data from brain connectomics, transcriptome and neurophysiology, the analyses of which can benefit from machine learning. For instance, the convolutional network method was deployed for image segmentation to achieve detailed, μm scale, reconstruction of connectivity in a neural microcircuit [Januszewski et al., 2018, Helmstaedter et al., 2013]. Another example is reconstruction of neural activity from spiking data [Pandarinath et al., 2018].

This primer will not be focused on data analysis; instead, our primary aim is to present basic concepts and methods for the development of ANN models of biological neural circuits, in the field of computational neuroscience. It is note worthy that ANNs should not be confused with neural network models in general. Mathematical models are all "artificial" inasmuch as they are not biological. We denote by ANNs specifically models that are in part inspired by neuroscience yet for which biologically justification is not the primary concern, in contrast to other types of models that strive to be built on quantitative data from the two pillars of neuroscience: neuroanatomy and neurophysiology. The use of ANNs in neuroscience [Zipser and Andersen, 1988] and cognitive science [Cohen et al., 1990] dates back to the early days of ANNs [Rumelhart et al., 1988]. In recent years, ANNs are becoming increasingly common model systems in neuroscience [Yamins and DiCarlo, 2016, Sussillo, 2014]. Here we outline three reasons why ANNs or deep learning models have already been–and will likely continue to be–particularly useful for neuroscientists.

Over the past decades, computational neuroscience has made great strides and become an integrated part of systems neuroscience [Abbott, 2008]. Much insights have been gained through integration of experiments and theory, examples include the idea of excitation and inhibition balance [Van Vreeswijk and Sompolinsky, 1996, Shu et al., 2003] and normalization [Carandini and Heeger, 2012]. Progress was also made in developing models of basic cognitive functions such as simple decision-making [Gold and Shadlen, 2007, Wang, 2008]. However, real-life problems can be incredibly complex, the underlying brain systems are difficult to capture with "hand-constructed" computational models. For example, object classification in the brain is carried out through many layers of complex linear-nonlinear processing. Building functional models of the visual systems that achieve behavioral performance close to humans' remained a formidable challenge not only for neuroscientists, but also for computer vision researchers. By directly training neural network models on complex tasks and behaviors, deep learning provides a way to efficiently generate candidate models for behaviors that otherwise would be near impossible to model (Figure 1B). By learning to perform many complex behaviors of animals, ANNs can serve as model systems for biological neural networks, in the same way as nonhuman animal models for understanding the human brain.

A second reason for advocating deep networks in systems neuroscience is the acknowledgment that relatively simple models cannot account for a wide diversity of activity patterns in heterogeneous neural populations (Figure 1C-E). One can rightly argue that this is a virtue rather than defect because simplicity and generality are hallmarks of good theories. However, complex neural signals also tell us that

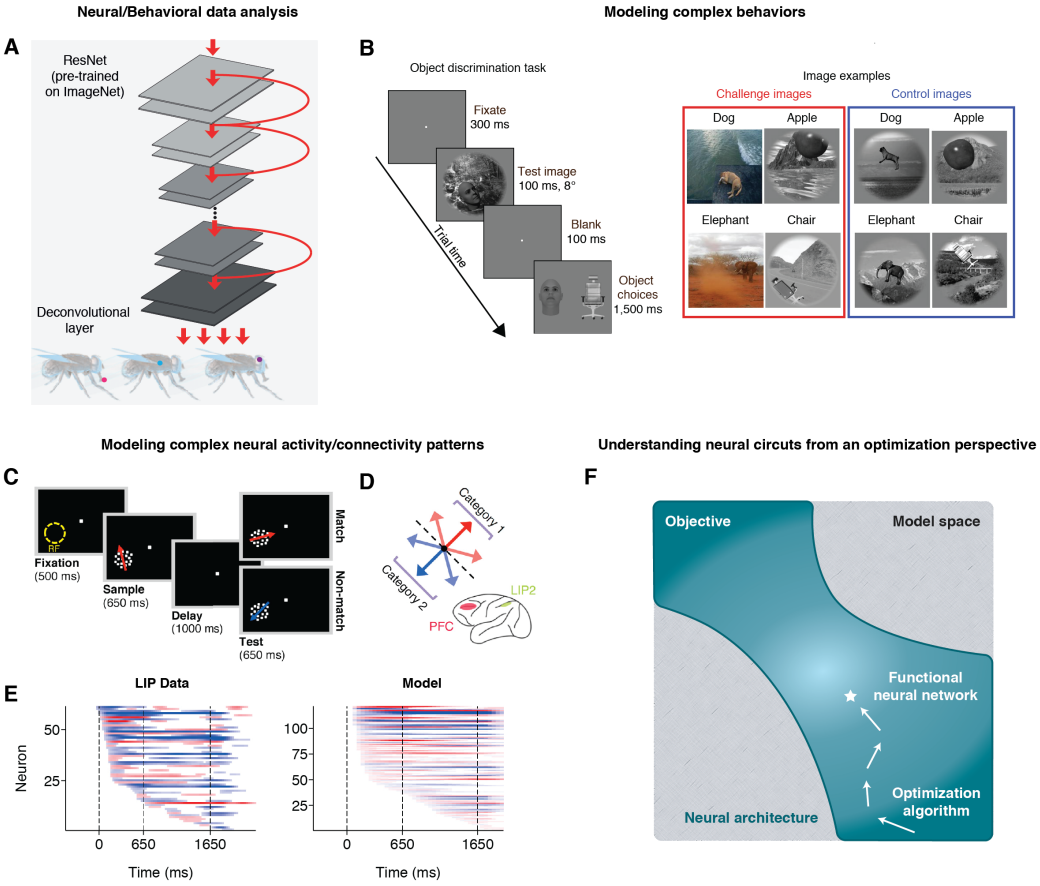

Figure 1: **Reasons for using ANNs for neuroscience research.** (A) Neural/Behavioral data analysis. ANNs can serve as image processing tools for efficient pose estimation. Figure adapted from Nath et al. [2019]. (B) Modeling complex behaviors. ANNs can perform object discrimination tasks involving challenging naturalistic visual objects. Figure adapted from Kar et al. [2019]. (C-E) Modeling complex neural activity/connectivity patterns. (C) Training both recurrent neural networks and monkeys on a delayed-match-to-category task [Freedman and Assad, 2006]. The task is to decide whether the test and sample stimuli (visual moving pattern) belong to the same category. (D) The two categories are defined based on the motion direction of the stimulus. (E) In a ANN trained to perform this categorization task, the recurrent units of the model display a wide heterogeneity of onset time for category selectivity (red: category 1; blue: category 2), similarly to single neurons recorded from monkey posterior parietal cortex (lateral intraparietal area, LIP) during the task. Figures adapted from Chaisangmongkon et al. [2017]. (F) Understanding neural circuits from an optimization perspective. In this view, functional neural networks are results of the optimization of an objective function in an abstract space of models constrained by the neural network architecture. Functional circuits can be understood in terms of the objective, optimization algorithm, and underlying network architecture.

existing models may be insufficient to elucidate mysteries of the brain. This is perhaps especially true in the case of the prefrontal cortex. Neurons in prefrontal cortex often show complex mixed selectivity to various task variables [Rigotti et al., 2010, 2013]. Such complex patterns are often not straightforward to interpret and understand using hand-built models that by design strive for simplicity. A new approach, possibly from machine learning, is needed to capture the dynamical and complex nature of neural activity.

Thirdly, besides providing mechanistic models to biological systems, machine learning can be used to probe the "why" question in neuroscience [Barlow et al., 1961]. Brains are biological machines evolved under pressure to compute efficiently. Even when we understand how a system works, we may still ask *why* brain circuits are structured and functioning as the way they do. Answering such questions can help us understand what evolutionary pressure led to the emergence of observed structures and functions. Similarly to biological systems evolving to survive, ANNs are trained to optimize objective functions given various structural constraints (Figure 1F). By retraining networks with new objective functions, structural constraints, and training algorithms, we can identify the necessary conditions for emerged structures and functions [Richards et al., 2019].

In this pedagogical primer, we will discuss how ANNs can and have benefited neuroscientists in the three ways described above. We will first introduce the key ingredients common in any study of ANNs. Next, we will describe two major applications of ANNs as neuroscientific models: convolutional networks as models for sensory, especially visual, systems, and recurrent neural networks as models for cognitive and motor systems. In the following sections, we will overview how to customize the analysis and architectural design of ANNs to better address a wide range of neuroscience questions. To help the readers gain hands-on experience, we accompany this primer with tutorial-style code in PyTorch and Jupyter Notebook (`https://github.com/gyyang/nn-brain`), covering all major topics.

2 Basic ingredients and variations in artificial neural networks

2.1 Basic ingredient: dataset/task, architecture, and algorithm

A typical study using deep networks consists of three basic ingredients: datasets or tasks, network architectures, and training algorithms. Connections in the neural network are constrained by the network architecture, but their specific values are randomly assigned at initialization. The training algorithm specifies how connection weights change to better fit datasets or perform tasks. We will go over a simple example, where a multi-layered-perceptron (MLP) is trained to perform a simple digit-classification task using supervised learning.

Dataset/Task and objective In a supervised learning setting, a dataset or a task is described by a set of input-output pairs $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}_{\text{target}}\}, i = 1, \cdots, N$. The goal is to learn parameters $\boldsymbol{\theta}$ of a neural network function $F(\cdot, \boldsymbol{\theta})$ that predicts the target outputs given inputs, $\boldsymbol{y}^{(i)} = F(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) \approx \boldsymbol{y}^{(i)}_{\text{target}}$. More precisely, the system is trained to optimize the value of an objective function, or commonly, minimize the

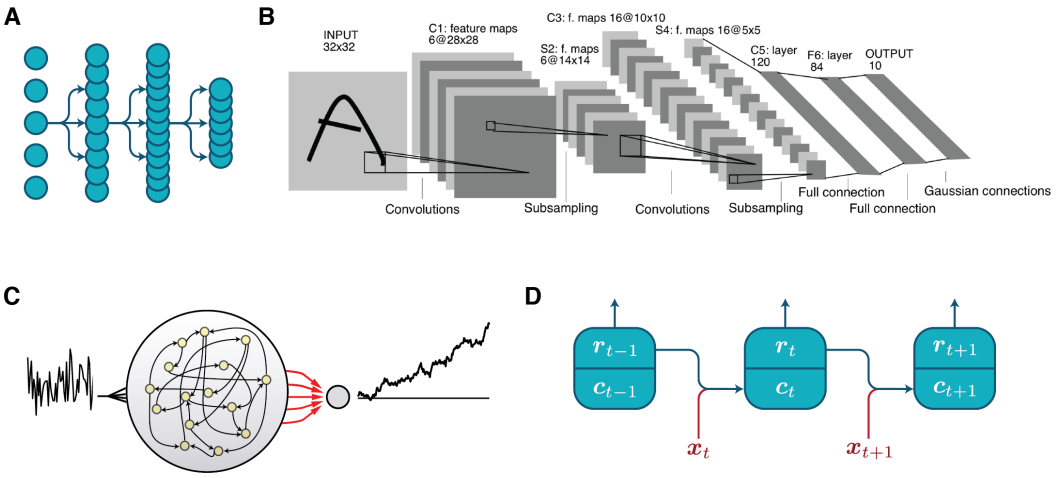

Figure 2: **Schematics of common neural network architectures.** (A) A multi-layer perceptron (MLP). (B) A convolutional neural network processing images. Figure adapted from LeCun et al. [1998]. (C) A recurrent neural network (middle) receives a stream of inputs (left). After training, an output unit (right) should report whether the mean input is positive or negative. Figure adapted from Mante et al. [2013]. (D) A recurrent neural network as unrolled in time as a feedforward system with layers corresponding to the network states over time steps.

value of a loss function L, which quantifies the difference between the target output $\boldsymbol{y}^{(i)}_{\text{target}}$ and the actual output $\boldsymbol{y}^{(i)}$. In the simple digit-classification task MNIST [LeCun et al., 1998], each input is an image containing a single digit, while the target output is an integer corresponding to the class of that object (1, 2,3, etc.).

Network architecture ANNs are incredibly versatile, including a wide range of architectures. Of all architectures, the most fundamental one is a Multi-Layer Perceptron (MLP) [Rosenblatt, 1958] (Figure 2A). A MLP consists of multiple layers of neurons, where neurons in the l-th layer only receive inputs from the $(l-1)$-th layer, and only project to the $(l+1)$-th layer.

$$\boldsymbol{r}^{(1)} = f(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}), \tag{1}$$

$$\boldsymbol{r}^{(l)} = f(\boldsymbol{W}^{(l)}\boldsymbol{r}^{(l-1)} + \boldsymbol{b}^{(l)}),\ 1 < l \leq N, \tag{2}$$

$$\boldsymbol{y} = \boldsymbol{W}^{(N)}\boldsymbol{r}^{(N)} + \boldsymbol{b}^{(N)}. \tag{3}$$

Here x is an external input, $\boldsymbol{r}^{(l)}$ denotes the neural activity of neurons in the l-th layer, and $\boldsymbol{W}^{(l)}$ is the connection matrix from the $(l-1)$-th to the l-th layer. $f(\cdot)$ is a (usually nonlinear) activation function of the model neurons. The output of the network is read out through connections $\boldsymbol{W}^{(N)}$. Parameters $\boldsymbol{b}^{(l)}$ and $\boldsymbol{b}^{(N)}$ are biases for model neurons and output units respectively.

When they are large enough, MLPs can in theory approximate arbitrary functions [Hornik et al., 1989]. However, in practice, the network size is limited, and good solutions may not be found through training even when they exist. MLPs are often used in combination or as parts of more modern neural network architectures.

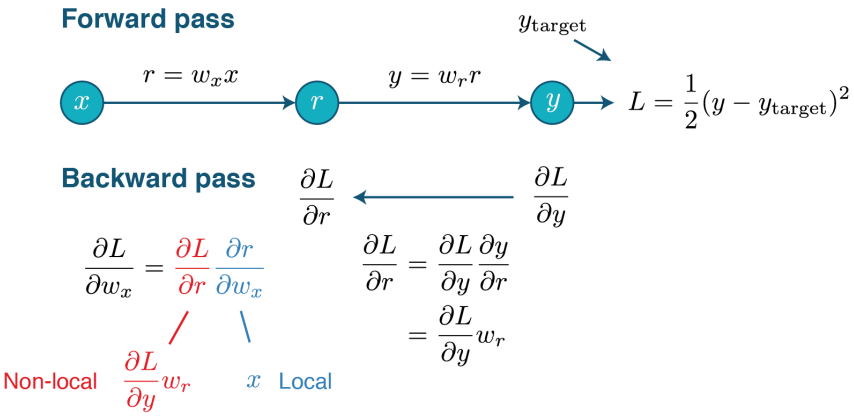

Figure 3: **Schematic of backpropagation.** In the forward pass of a simple neural network, an input unit with activity x projects to a hidden unit with activity $r = w_x x$, which then projects to an output unit with activity $y = w_r r$. To compute the gradient $\partial L/\partial w_x = (\partial L/\partial r)(\partial r/\partial w_x)$, a backward pass computes $\partial L/\partial r$ from $\partial L/\partial y$ using connection weight value w_r. Therefore computing $\partial L/\partial w_x$ requires non-local information about synaptic weight value of distant synapses.

Training algorithm The signature method of training in deep learning is stochastic gradient descent (SGD) [Robbins and Monro, 1951, Bottou et al., 2018]. Trainable parameters, collectively denoted as $\boldsymbol{\theta}$, are updated in the opposite direction of the gradient of the loss, $\frac{\partial L}{\partial \boldsymbol{\theta}}$. This is because, intuitively, $\boldsymbol{\theta}$ should be reduced by training if the to-be-minimized cost function L increases with it; and increased otherwise. For each step of training, the loss is computed using a small number (a minibatch) of randomly selected training examples, hence the name "stochastic". The gradient, $\frac{\partial L}{\partial \boldsymbol{\theta}}$ is the direction of parameter change that would lead to maximum increase in the loss when the change is small enough. To decrease the loss, trainable parameters are updated in the opposite direction of the gradient, with a magnitude proportional to the learning rate η,

$$\Delta \boldsymbol{\theta} = -\eta \frac{\partial L}{\partial \boldsymbol{\theta}}. \tag{4}$$

Parameters such as $\boldsymbol{W}$ and $\boldsymbol{b}$ are usually trainable. Other parameters are set by the modelers and called hyperparameters, for example, the learning rate η. A crucial requirement is the differentiability, namely derivatives of functions in the model are well defined.

For a feedforward network without any intermediate (hidden) layer [Rosenblatt, 1962],

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{5}$$

computing the gradient is straightforward,

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{x}^T. \tag{6}$$

For a multi-layer network, the differentiation is done using the back-propagation algorithm [Rumelhart et al., 1988, LeCun, 1988]. To compute the loss L, the

network is run in a "forward pass". Next, to efficiently compute the exact gradient $\frac{\partial L}{\partial \boldsymbol{\theta}}$, information about the loss needs to be passed backward, in the opposite direction of the forward pass, hence the name back-propagation.

To illustrate the concept by a simple example, consider a 3-neuron chain network with activity denoted by x, r and y (Figure 3). According to Eq. (4), and assuming that $L = \frac{1}{2}(y_{\text{target}} - y)^2$, the change of weight w_r is given by the gradient $\partial L/\partial w_r = -(y_{\text{target}} - y)r$, this modification only depends on local information about the input and output units of this connection. Hence, if $y_{\text{target}} > y$, w_r should change to increase the net input and Δw_r has the same sign as r. Conversely, if $y_{\text{target}} < y$, w_r should change to decrease the net input and Δw_r has the opposite sign from r.

The change of weight w_x is given by

$$\frac{\partial L}{\partial w_x} = \frac{\partial L}{\partial r}\frac{\partial r}{\partial w_x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial r}\frac{\partial r}{\partial w_x}, \tag{7}$$

according to the chain rule in calculus. Note that $\partial y/\partial r = w_r$, therefore modification of w_x is not local but depends on a synaptic weight downstream. The error $\partial L/\partial y = -(y_{\text{target}} - y)$ is "back-propagated" from the output side to the input side. Importantly, this algorithm provides a recipe for credit assignment in determining how much w_x and w_r should be modified for improving performance.

These points apply generally to multi-layer nonlinear networks with arbitrary number of units per layer. With an increasing number of layers in a deep network, weight modifications involve products of many partial derivatives. Backpropagation also applies to a recurrent neural network (RNN), its state at each time point is treated as a different layer. Therefore computing its gradient involves propagating information backward in time (backpropagation-through-time) [Werbos, 1990].

Products of a large number of partial derivatives can grow exponentially, if most derivatives are larger than 1; or vanish to zero if the majority of derivatives are smaller than 1, making it historically difficult to train recurrent networks and very deep feedforward networks [Bengio et al., 1994, Pascanu et al., 2013]. Such exploding and vanishing gradient problems can be alleviated with a combination of modern network architectures [Hochreiter and Schmidhuber, 1997, He et al., 2016] and training algorithms [Pascanu et al., 2013, Le et al., 2015] that tend to preserve the norm of the backpropagated gradient.

2.2 Variations of objective functions

Traditionally, learning problems are divided into three kinds: supervised, unsupervised, and reinforcement learning problems.

Supervised learning As mentioned before, for supervised learning tasks, input and target output pairs are provided $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}_{\text{target}}\}$. The goal is to minimize the difference between target outputs and actual outputs predicted by the network. In many common supervised learning problems, the target outputs are behavioral outputs. For example, in a typical object classification task, each input is an image containing a single object, while the target output is an integer corresponding to the

class of that object (e.g., dog, cat, etc.). In other cases, the target output can directly be neural recording data [McIntosh et al., 2016, Rajan et al., 2016, Andalman et al., 2019].

The classical perceptual decision-making task with random-dot motion [Britten et al., 1992, Roitman and Shadlen, 2002] can be formulated as a supervised learning problem. In this task, animals watch randomly moving dots and report the dots' overall motion direction by choosing one of two alternatives, A or B. This task can be simplified as a network receiving a stream of noisy inputs $x_t^{(i)}$ at every time point t, which represents the net evidence in support of one and against the other alternative. At the end of each trial $t = T$, the system should learn to report the sign of the average input $y_{\text{target}}^{(i)} = \text{sign}(\langle x_t^{(i)} \rangle_t)$.

Unsupervised learning For unsupervised learning, only inputs $\{\boldsymbol{x}^{(i)}\}$ are provided, the objective function is defined solely with the inputs and the network parameters $L(\boldsymbol{x}, \boldsymbol{\theta})$. For example, finding the first component in Principal Component Analysis (PCA) can be formulated as an unsupervised learning in a simple neural network where a single neuron y reads out from a group of input neurons $y = \boldsymbol{W}\boldsymbol{x}$ and tries to maximize its variance $\text{Var}(y)$ [Oja, 1982].

As a special form of unsupervised learning, a learning problem can be converted to a supervised one by turning part (or all) of the inputs into the target outputs. For example, the input itself can be used as the target output, $\boldsymbol{y}_{\text{target}}^{(i)} = \boldsymbol{x}^{(i)}$. Such datasets are used in the training of autoencoders [Rumelhart et al., 1988, Kingma and Welling, 2013], networks that learn to compress inputs $\boldsymbol{x}$ into a lower-dimensional latent representation using an encoder network, $\boldsymbol{z} = f_{\text{encode}}(\boldsymbol{x})$. The original inputs should be decoded back from the latent representation by a generative, decoder network, $\boldsymbol{y}_{\text{target}} = \boldsymbol{x} \approx f_{\text{decode}}(\boldsymbol{z})$. The use of a low-dimensional latent representation $\boldsymbol{z}$ prevents the autoencoder from learning the trivial identity mapping.

Reinforcement learning For reinforcement learning [Sutton and Barto, 2018], an interactive environment is provided. At time step t, complete information about the environment is described by its state $\boldsymbol{s}_t$. An agent (a model) interacting with the environment receives an observation $\boldsymbol{o}_t$, produces an action a_t that updates the environment to $\boldsymbol{s}_{t+1}$, and receives a scalar reward r_t (negative value for punishment). In many classical reinforcement learning problems, the observation is the environment state $\boldsymbol{o}_t = \boldsymbol{s}_t$. The objective is to produce appropriate actions a_t that maximize cumulative rewards $\sum_t r_t$.

Most tasks performed by lab animals can be described as reinforcement learning tasks, since animals are usually motivated to perform the task via rewards. Meanwhile, many of these tasks can also be formulated as supervised learning tasks whenever there exists a correct choice, like in the case of perceptual decision making. However, many tasks can only be described as reinforcement learning tasks. For instance, we choose dishes at a restaurant based on expected outcomes (nutritional value, taste etc) that are subjective; there is no wrong answer. A perceptual decision-making task where there is a correct answer (A, not B) can be extended to assess animals' confidence about their choice [Kiani and Shadlen, 2009]. In addition to the

two alternatives (one of them is correct and results in a large reward), monkeys are presented a sure-bet option that guarantees a small reward. In this case, subjects are more likely to choose the sure-bet option when they are less confident, because a small reward is better than no reward if the choice turns out to be incorrect. There is no ground-truth choice output, because the optimal choice depends on the animals' own confidence level at their perceptual decision.

The separation between these three types of tasks is not clear cut. For example, as mentioned above, unsupervised learning tasks can be converted to supervised ones in certain cases. A complex neural network performing reinforcement learning tasks can, and usually do, have components that are trained with supervised and/or unsupervised learning. Similarly, animals evolved to survive and reproduce, a reinforcement learning problem. It is an open question whether a given biological neural system is likely better approximated by artificial networks trained using supervised or unsupervised learning.

2.3 Variations of network architectures

Recurrent neural network Besides MLP, another fundamental ANN architecture is recurrent neural networks (RNNs) that process information in time (Figure 2C). In a "vanilla" or Elman RNN [Elman, 1990], activity of model neurons at time t, $\boldsymbol{r}_t$, is driven by recurrent connectivity $\boldsymbol{W}_r$, and by inputs $\boldsymbol{x}_t$ through connectivity $\boldsymbol{W}_x$. The output of the network is read out through connections $\boldsymbol{W}_y$.

$$\boldsymbol{c}_t = \boldsymbol{W}_r \boldsymbol{r}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_r, \tag{8}$$

$$\boldsymbol{r}_t = f(\boldsymbol{c}_t), \tag{9}$$

$$y_t = \boldsymbol{W}_y \boldsymbol{r}_t + b_y. \tag{10}$$

Here $\boldsymbol{c}_t$ represents the cell state, analogous to membrane potential or input current, while $\boldsymbol{r}_t$ represents the output activity. An RNN can be unrolled in time (Figure 2D) and viewed as a particular form of a MLP. Here, neurons in the t-th layer, $\boldsymbol{r}_t$ receive inputs from the $(t-1)$-th layer $\boldsymbol{r}_{t-1}$ and additional inputs from outside of the recurrent network $\boldsymbol{x}_t$. Unlike regular MLPs, the connections from each layer to the next are shared across time.

Convolutional neural networks A particularly important type of network architectures is convolutional neural network (Figure 2B). The use of convolution means that a group of neurons will each process its respective inputs using the same function, in other words, the same set of connection weights. In a typical convolutional neural network processing visual inputs [Fukushima et al., 1983, LeCun et al., 1990, Krizhevsky et al., 2012, He et al., 2016], neurons are organized into "channels" or "feature maps". All neurons within a single channel process different parts of the input space using the same shared set of connection weights, therefore these neurons have the same stimulus selectivity with receptive fields at different spatial locations.

For simplicity, consider a neural network processing a 1-dimensional image (pixels along a line). In an ordinary neural network, the i-th neuron in layer l receives connections from all neurons of the previous layer, $r_i^l = \sum_j w_{ij} r_j^{l-1}$. However, in a convolutional neural network with a single channel at each layer, each neuron is

indexed by its preferred spatial location i. Here, the i-th neuron receives connections from previous layer neurons with similar spatial preferences. For example,

$$r_i^l = w_{-1} r_{i-1}^{l-1} + w_0 r_i^{l-1} + w_1 r_{i+1}^{l-1} = \sum_{j=-1}^{+1} w_j r_{i+j}^{l-1}. \tag{11}$$

Notice that the connection weights do not depend on the absolute spatial location of the i-th neuron, instead they depend solely on the spatial displacement between the pre- and post-synaptic neurons. This reusing of weights not only reduces the number of trainable parameters, but also imposes invariance on processing. For visual processing, convolutional networks typically impose spatial invariance such that objects are processed with the same set of weights regardless of their spatial positions.

Activation function Most neurons in ANNs, like their biological counterparts, perform nonlinear computations based on their inputs. These neurons are usually point neurons with a single activation function $f(\cdot)$ that links the sum of inputs to the output activity. A common choice of activation function is the Rectified Linear Unit (ReLU) function, $f(x) = \max(x, 0)$ [Glorot et al., 2011]. ReLU and its variants [Clevert et al., 2015] are routinely used in feedforward networks, while the hyperbolic tangent (tanh) function is often used in recurrent networks [Hochreiter and Schmidhuber, 1997]. ReLU and similar activation functions are asymmetric and non-saturating at high value. Although biological neurons eventually saturate at high rate, they often operate in non-saturating regimes. Therefore, traditional neural circuit models with rate units have also frequently used non-saturating activation functions [Abbott and Chance, 2005, Rubin et al., 2015].

Normalization Normalization methods are important components of many ANNs, in particular very deep neural networks [Ioffe and Szegedy, 2015, Ba et al., 2016b, Wu and He, 2018]. Similar to normalization in biological neural circuits [Carandini and Heeger, 2012], normalization in ANNs keep inputs and/or outputs of neurons in desirable ranges. For example, layer normalization [Ba et al., 2016b] normalizes the mean and variance of inputs to, or activity of, all neurons in a layer. For inputs $\boldsymbol{x}$ to a layer of neurons, after normalization, the input $\hat{x}_i$ to the i-th neuron is

$$\hat{x}_i = \gamma(\frac{1}{\sigma}(x_i - \mu)) + \beta, \tag{12}$$

$$\mu = \langle x_j \rangle_j, \tag{13}$$

$$\sigma = \sqrt{\langle (x_j - \mu)^2 \rangle_j + \epsilon}. \tag{14}$$

μ and σ are the mean and variance of $\boldsymbol{x}$, while γ and β are trainable parameters that set the eventual mean and variance of $\hat{\boldsymbol{x}}$. ϵ is a small constant for stability. The notation $\langle x_j \rangle_j$ refers to average over all units indexed by j.

2.4 Variations of training algorithms

Variants of SGD-based methods Supervised and unsupervised learning tasks are usually trained with SGD-based methods. Due to the stochastic nature of the

estimated gradient, in certain situations directly applying SGD leads to poor training performance. Gradually decaying learning rate value η during training can often improve performance, since smaller learning rate during late training encourages finer-tuning of parameters [Bottou et al., 2018]. Various optimization methods based on SGD are used to improve learning [Kingma and Ba, 2014, Sutskever et al., 2013]. One simple and effective technique is momentum [Sutskever et al., 2013, Polyak, 1964], which on step j updates parameters with $\Delta\boldsymbol{\theta}^{(j)}$ based on temporally smoothed gradients $\boldsymbol{v}^{(j)}$,

$$\boldsymbol{v}^{(j)} = \mu\boldsymbol{v}^{(j-1)} + \frac{\partial L^{(j)}}{\partial\boldsymbol{\theta}}, \quad 0 \leq \mu \leq 1 \tag{15}$$

$$\Delta\boldsymbol{\theta}^{(j)} = -\eta\boldsymbol{v}^{(j)}. \tag{16}$$

In adaptive learning rate methods [Duchi et al., 2011, Kingma and Ba, 2014], the learning rate of individual parameter is adjusted based on the statistics (e.g., mean and variance) of its gradient over training steps. For example, in the Adam method [Kingma and Ba, 2014], value of a parameter is modified more rapidly if its gradient has been consistently small (low variance). Adaptive learning rate methods can be viewed as approximately taking into account curvature of the loss function [Duchi et al., 2011].

Value-based and policy-based methods for reinforcement learning In an interactive environment, an agent follows a policy π that produces actions a_t, and receives rewards r_t. The goal of reinforcement learning is to learn the policy π that maximizes the expected total future reward $\mathbb{E}[G_t]$, where $G_t = \sum_{k=0}^{\infty}\gamma^k r_{t+k}$ is the total future reward, and $0 \leq \gamma \leq 1$ is a temporal discount factor that devalues future reward. There are many variations of reinforcement learning algorithms [Sutton and Barto, 2018]. Most of them can be broadly separated into value-based and policy-based methods. Value-based methods attempt to learn value functions that estimate the expected total future reward $\mathbb{E}[G_t]$, and use the value functions to guide the choice of actions. In contrast, policy-based methods attempt to directly learn the policy that maximizes future reward $\mathbb{E}[G_t]$. In deep reinforcement learning, value functions and/or policy functions are parameterized as neural networks with parameters $\boldsymbol{\theta}$, and learned through gradient descent [Silver et al., 2017]. Here we go into more details describing these two methods.

Value-based methods learn a value function to estimate $\mathbb{E}[\sum_{k=0}^{\infty}\gamma^k r_{t+k}]$ given the current state, $V(\boldsymbol{s}_t)$ (state-value) [Sutton, 1988], or the current state and a potential action $Q(\boldsymbol{s}_t, a)$ (action-value) [Watkins and Dayan, 1992]. Value functions can be computed by neural networks that take the state representation $\boldsymbol{s}_t$ as inputs and output the state-value $V(\boldsymbol{s}_t)$ or the values of all potential actions $Q(\boldsymbol{s}_t, a)$, assuming a discrete set of possible actions.

To learn the state-value function (action-value can be learned similarly), consider what happens if the value function converges to the expected total future reward,

$$V^*(\boldsymbol{s}_t) = \mathbb{E}[\sum_{k=0}^{\infty}\gamma^k r_{t+k}] = \mathbb{E}[r_t] + \gamma\mathbb{E}[\sum_{k=0}^{\infty}\gamma^k r_{(t+1)+k}] = \mathbb{E}[r_t] + \gamma\mathbb{E}[V^*(\boldsymbol{s}_{t+1})]. \tag{17}$$

Learning of the value function can be cast as a supervised learning problem where $V(\boldsymbol{s}_t, \boldsymbol{\theta})$ should match the target, $\mathbb{E}[r_t] + \gamma\mathbb{E}[V(\boldsymbol{s}_{t+1})]$, by minimizing the value function loss,

$$L_V(\boldsymbol{\theta}) = \frac{1}{2}\left[V(\boldsymbol{s}_t, \boldsymbol{\theta}) - (\mathbb{E}[r_t] + \gamma\mathbb{E}[V(\boldsymbol{s}_{t+1})])\right]^2. \tag{18}$$

And just like SGD, this loss function can be estimated stochastically as the agent interacts with the environment,

$$L_V^{(t)}(\boldsymbol{\theta}) = \frac{1}{2}\left[V(\boldsymbol{s}_t, \boldsymbol{\theta}) - (r_t + \gamma V(\boldsymbol{s}_{t+1}))\right]^2. \tag{19}$$

Updating the value function parameters $\boldsymbol{\theta}$ would bring $V(\boldsymbol{s}_t, \boldsymbol{\theta})$ closer to $r_t + \gamma V(\boldsymbol{s}_{t+1})$,

$$\Delta\boldsymbol{\theta} \propto -\frac{\partial L_V^{(t)}}{\partial \boldsymbol{\theta}} = [r_t + \gamma V(\boldsymbol{s}_{t+1}) - V(\boldsymbol{s}_t)]\frac{\partial V(\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \delta_t \frac{\partial V(\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{20}$$

where $\delta_t = r_t + \gamma V(\boldsymbol{s}_{t+1}) - V(\boldsymbol{s}_t)$ is called the reward prediction error, and is correlated with activity of certain types of dopamine neurons [Schultz et al., 1997, Watabe-Uchida et al., 2017].

To maximize total reward, the agent may take a greedy action that maximizes the action-value function, $a_t^{\text{greedy}} = \text{argmax}_a Q(\boldsymbol{s}_t, a)$. This maximization requires the agent to have discrete actions. If the network is learning the state-value function $V(\boldsymbol{s})$, and the transition probability $p(\boldsymbol{s}', r|\boldsymbol{s}, a)$ between environment states are known, then the action-value can be computed from the state-value,

$$Q(\boldsymbol{s}, a) = \sum_{\boldsymbol{s}'} p(\boldsymbol{s}', r|\boldsymbol{s}, a)[r + \gamma V(\boldsymbol{s}')]. \tag{21}$$

Taking only greedy actions (i.e., exploitation) is, however, sub-optimal when the value function is not perfectly learned, therefore exploration is warranted. A simple ϵ-greedy exploration strategy chooses actions randomly for a small proportion ϵ of time steps.

Unlike value-based methods, policy-based methods can be used when the agent outputs continuous-valued actions, by learning a direct mapping from the state $\boldsymbol{s}_t$ to action a_t. The policy can be described by the probability, $\pi(a|\boldsymbol{s}_t, \boldsymbol{\theta})$, of choosing an action a given state $\boldsymbol{s}_t$. However, it is not obvious how to properly adjust parameters $\boldsymbol{\theta}$ of the policy since, unlike supervised learning, no target action is provided. This issue is addressed by the policy gradient theorem [Sutton et al., 2000], which provides a general recipe for directly estimating the gradient of the expected total reward with respect to policy parameters, $\partial\mathbb{E}_\pi[G_t]/\partial\boldsymbol{\theta}$. Intuitively, the policy gradient theorem states that the expected total reward $\mathbb{E}_\pi[G_t]$ can be improved by increasing the probability of an action a_t in proportion to the ensuing total reward G_t. More precisely,

$$\frac{\partial\mathbb{E}_\pi[G_t]}{\partial\boldsymbol{\theta}} = \mathbb{E}_\pi[G_t\frac{\partial \ln\pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial\boldsymbol{\theta}}]. \tag{22}$$

Again, the parameters $\boldsymbol{\theta}$ can be optimized and updated stochastically, leading to the REINFORCE algorithm [Williams, 1992],

$$\Delta\boldsymbol{\theta} \propto G_t\frac{\partial \ln\pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial\boldsymbol{\theta}}. \tag{23}$$

Since $\pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta})$ is the output of the policy network, its gradient with respect to $\boldsymbol{\theta}$ can be computed through backpropagation.

However, even with an optimal policy, both G_t and $\partial \ln \pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta})/\partial \boldsymbol{\theta}$ are generally non-zero. Therefore the policy parameters would keep being updated despite being optimal. Such high variance in the parameter update can be reduced by subtracting G_t with a baseline state-value function,

$$\Delta \boldsymbol{\theta} \propto (G_t - V(\boldsymbol{s}_t)) \frac{\partial \ln \pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{24}$$

Furthermore, G_t can be approximated with $r_t + \gamma V(\boldsymbol{s}_{t+1})$, leading to

$$\Delta \boldsymbol{\theta} \propto \delta_t \frac{\partial \ln \pi(a_t|\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{25}$$

Intuitively, an action a_t is more likely to be chosen in state s_t if it led to more reward than expected. These methods are called actor-critic methods, where an actor learns the policy to generate actions, while the critic learns the value function.

In most real-life reinforcement learning environments, true state of the environment is not observable, therefore the network needs to estimate the value/policy function (e.g., $V(\boldsymbol{s}_t)$) using information about present and past observations $\{\boldsymbol{o}_{t-k}\}$, $k = 0, 1, \ldots$, and optionally, augmenting that observations with additional information such as its past actions $\{a_{t-k}\}$ and rewards $\{r_{t-k}\}$, $k = 1, 2, \ldots$. Recurrent neural networks can be used to incorporate such temporal observations into a learnable state representation and to compute the value and/or policy functions.

Regularization Regularization techniques are important during training in order to improve performance in generalization by deep networks. Adding a L2 regularization term, $L_{\text{reg}} = \lambda \sum_{ij} W_{ij}^2$, to the loss function [Tikhonov, 1943] (equivalent to weight decay [Krogh and Hertz, 1992]) discourages the network from using large connection weights. Dropout [Srivastava et al., 2014] silences a randomly-selected portion of neurons during training. It reduces the network's reliance on particular neurons or a precise combination of neurons. Dropout can be thought of as loosely approximating spiking noise.

Besides trainable parameters, the network architecture and training algorithm are described by a number of hyperparameters. For example, learning rate η of the training algorithm is one of the most important hyperparameters. Appropriate tuning of hyperparameters is important for training and generalization performance.

3 Examples of building ANNs to address neuroscience questions

In this section, we overview two common usages of ANNs in addressing neuroscience questions.

3.1 Convolutional networks for visual systems

Deep convolutional neural networks are currently the standard tools in computer vision research and applications [Krizhevsky et al., 2012, Simonyan and Zisserman,

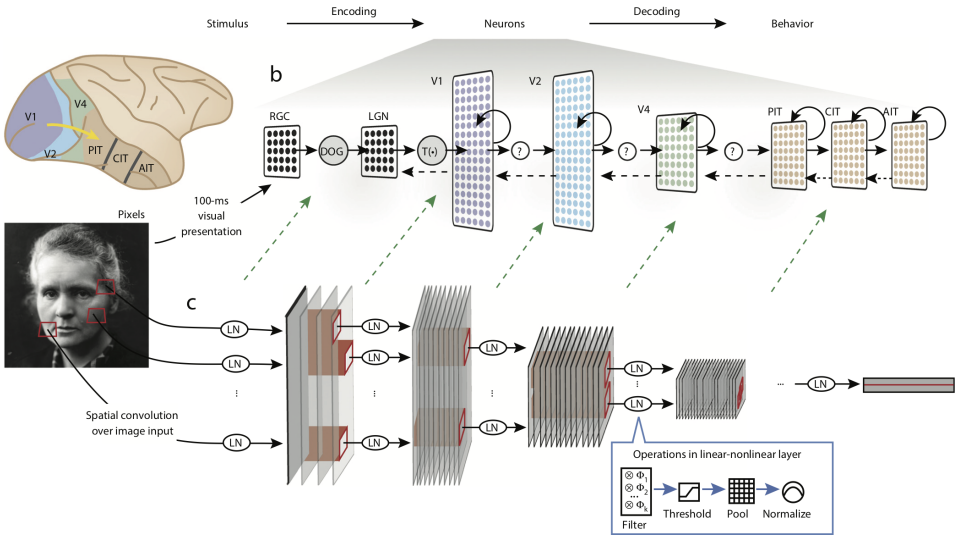

Figure 4: **Comparing the visual system and deep convolutional neural networks.** The same image can be passed through the visual cortex (top) and a deep convolutional neural network (bottom), allowing for side-by-side comparisons between biological and artificial neural networks. Figure adapted from Yamins and DiCarlo [2016].

2014, He et al., 2016, 2017]. These networks routinely consist of tens, sometimes hundreds, of layers of convolutional processing. Effective training of deep feed-forward neural networks used to be difficult. This trainability problem has been drastically improved by a combination of innovations in various areas. Modern deep networks would be too large and therefore too slow to run, not to mention train, if not for the rapid development of hardware such as GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units) [Jouppi et al., 2017]. Deep convolutional networks are usually trained with large naturalistic datasets containing millions of high resolution labeled images (e.g., Imagenet [Deng et al., 2009]), using training methods with adaptive learning rates [Kingma and Ba, 2014, Tieleman and Hinton, 2012]. Besides the default use of convolution, a wide range of network architecture innovations improve performance, including the adoption of ReLU activation function [Glorot et al., 2011], normalization methods [Ioffe and Szegedy, 2015], and the use of residual connections that provide an architectural shortcut from a network layer's inputs directly to its outputs [He et al., 2016].

Classical models based on hand-engineered features [Jones and Palmer, 1987, Freeman and Simoncelli, 2011, Riesenhuber and Poggio, 1999], unsupervised trained using the efficient coding principles [Barlow et al., 1961, Olshausen and Field, 1996]

Deep convolutional networks have been proposed as computational models of the visual systems, particularly of the ventral visual stream [Yamins and DiCarlo, 2016]. These models are typically trained using supervised learning on the same image classification tasks as the ones used in computer vision research, and in many cases, are the exact same convolutional networks developed in computer vision. In comparison, classical models of the visual systems typically rely on hand-designed features

(synaptic weights) [Jones and Palmer, 1987, Freeman and Simoncelli, 2011, Riesenhuber and Poggio, 1999], such as Gabor filters, or are trained with unsupervised learning based on the efficient coding principles [Barlow et al., 1961, Olshausen and Field, 1996]. Although classical models have had success at explaining many features of lower-level visual areas, deep convolutional networks surpass them at explaining neural activity in higher-level visual areas in both monkeys [Yamins et al., 2014, Cadieu et al., 2014, Yamins and DiCarlo, 2016] and humans [Khaligh-Razavi and Kriegeskorte, 2014] (Figure 4). Besides being trained to classify objects, convolutional networks can also be trained to directly reproduce patterns of neural activity recorded in various visual areas [McIntosh et al., 2016, Prenger et al., 2004].

Each layer in a typical convolutional network is characterized by its number of feature maps (channels) N_C (Figure 7a). The space is divided into a 2-d grid with height N_H and width N_W. Each channel contains $N_H \times N_W$ neurons with the same selectivity except for different spatial preferences. A neuron in layer l usually receives input connections from layer $l-1$ neurons of similar spatial preferences, across all channels. In a typical convolutional network as illustrated in Figure 4, across layers the number of neurons per channel decreases (with coarser spatial resolution) while more features are extracted (with an increasing number of channels). At the top-end of the system is a classifier which is trained to accomplish a particular task, such as categorization of visual objects.

As computational models of visual systems, convolutional networks can model complex, high-dimensional inputs to downstream areas, useful for large-scale models using pixel-based visual inputs [Eliasmith et al., 2012]. This process has been made particularly straightforward with the easy access of many pre-trained networks in standard deep learning frameworks like Pytorch [Paszke et al., 2019] and Tensorflow [Abadi et al., 2016].

3.2 Recurrent neural networks for cognitive and motor systems

Recurrent neural networks are common machine learning tools to process sequences, such as speech and text. In neuroscience, they have been used to model various aspects of the cognitive and motor systems [Mante et al., 2013, Sussillo et al., 2015, Yang et al., 2019, Wang et al., 2018, Cueva and Wei, 2018]. Unlike convolutional networks used to model visual systems that are trained on large-scale image classification tasks, recurrent networks are usually trained on the specific cognitive or motor tasks that neuroscientists are studying.

Although both called recurrent neural networks, the RNNs used in machine learning and neuroscience often treat time differently. Almost all RNNs used in machine learning are discrete time systems (but see [Chen et al., 2018]), (Figure 5A). In contrast, many RNNs used in neuroscience are based on or inspired by continuous time dynamical systems [Wilson and Cowan, 1972] (Figure 5B). In a discrete time system, state at time step t is obtained through a mapping from the state at time step $t-1$ (Eq. 9). In contrast, a continuous-time recurrent network is a dynamical

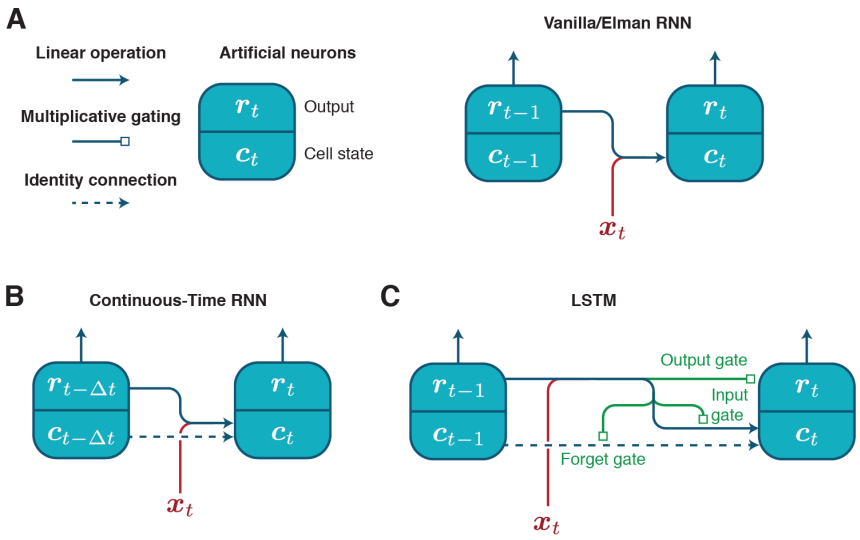

Figure 5: **Common RNN architectures** (A) A vanilla/Elman RNN. (B) A continuous time RNN, where cell state at time t, $\boldsymbol{c}_t$, depends on external inputs, recurrent inputs, and previous cell state $\boldsymbol{c}_{t-\Delta t}$. (C) A LSTM network. Recurrent inputs to cell state are gated by the input gate, contributions from past cell states are gated by the forget gate, and neuronal outputs are gated by the output gate.

system,

$$\tau \frac{d\boldsymbol{r}}{dt} = -\boldsymbol{r}(t) + f(W_r \boldsymbol{r}(t) + W_x \boldsymbol{x}(t) + \boldsymbol{b}_r). \tag{26}$$

Here τ is the single-unit time scale. This continuous-time system can then be discretized using the Euler method with a time step of Δt,

$$\boldsymbol{r}(t + \Delta t) = \boldsymbol{r}(t) + \Delta \boldsymbol{r} = \boldsymbol{r}(t) + \frac{\Delta t}{\tau}[-\boldsymbol{r}(t) + f(W_r \boldsymbol{r}(t) + W_x \boldsymbol{x}(t) + \boldsymbol{b}_r)]. \tag{27}$$

A main difference between this discretized system and a discrete-time RNN is that dynamics in the discretized system has a meaningful time-scale, governed by τ. We shall refer this type of models as “vanilla" RNNs that are not endowed with other complex features, in particular gating mechanisms that we now discuss.

Similar to deep feedforward networks, RNNs were difficult to train [Bengio et al., 1994]. Training of RNNs is made substantially easier with the adoption of gating using LSTM (Long Short-Term-Memory) networks [Hochreiter and Schmidhuber, 1997, Gers Felix et al., 2000] or GRU (Gated Recurrent Units) networks [Cho et al., 2014, Chung et al., 2014]. Gating variables dynamically control information flow within these networks through multiplicative interactions. In a LSTM network, there are three types of gating variables (Figure 5C). Input and output gates, $\boldsymbol{g}_t^i$ and $\boldsymbol{g}_t^o$, control the inputs to and outputs of the cell state $\boldsymbol{c}_t$, while forget gate $\boldsymbol{g}_t^f$ controls

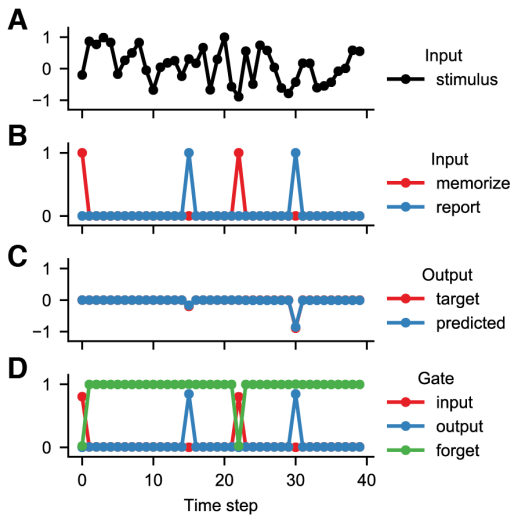

Figure 6: **Visualizing LSTM activity in a simple memory task.** (A) In this memory task, the network receives a stream of input stimulus, the value of which is randomly and independently sampled at each time point. (B) The network needs to memorize the value of the stimulus when the "memorize input" (red) is last active, and report that value through an output unit when the "report input" (blue) is active. (C) After training, a single-unit LSTM can perform the task almost perfectly for modest memory duration. (D) This network opens the input gate (allowing inputs) and closes the forget gate (forgetting previous memory) when receiving the memorize input, and opens the output gate when receiving the report input.

whether cell state $\boldsymbol{c}_t$ keeps its memory $\boldsymbol{c}_{t-1}$.

$$\begin{aligned}
\boldsymbol{g}_t^f &= \sigma_g(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{r}_{t-1} + \boldsymbol{b}_f), \\
\boldsymbol{g}_t^i &= \sigma_g(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{r}_{t-1} + \boldsymbol{b}_i), \\
\boldsymbol{g}_t^o &= \sigma_g(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{r}_{t-1} + \boldsymbol{b}_o), \\
\boldsymbol{c}_t &= \boldsymbol{g}_t^f \odot \boldsymbol{c}_{t-1} + \boldsymbol{g}_t^i \odot \sigma_c(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{r}_{t-1} + \boldsymbol{b}_c), \\
\boldsymbol{r}_t &= \boldsymbol{g}_t^o \odot \sigma_r(\boldsymbol{c}_t).
\end{aligned} \tag{28}$$

Here the symbol $\odot$ denotes the element-wise (Hadamard) product of two vectors of the same length ($\boldsymbol{z} = \boldsymbol{x} \odot \boldsymbol{y}$ means $z_i = x_i y_i$). Gating variables are bounded between 0 and 1 by the sigmoid function σ_g, which can be viewed as a smooth differentiable approximate of a binary step function. A gate is opened or closed when its corresponding gate value is near 1 or 0 respectively. Crucially, all the weights $\boldsymbol{W}_f$, $\boldsymbol{U}_f$, $\boldsymbol{W}_i$, $\boldsymbol{U}_i$, $\boldsymbol{W}_o$ and $\boldsymbol{U}_o$ are trained, together with connection weights for the cells $\boldsymbol{W}_c$ and $\boldsymbol{U}_c$. Consequently, gates change over time in a way to optimally achieve the objective (Figure 6).

In a continuous-time version of the LSTM network cell state $\boldsymbol{c}(t)$ equation,

$$\tau \odot \frac{d\boldsymbol{c}}{dt} = -(1 - \boldsymbol{g}^f(t)) \odot \boldsymbol{c}(t) + \boldsymbol{g}^i(t) \odot \sigma_c(\boldsymbol{W}_c \boldsymbol{x}(t) + \boldsymbol{U}_c \boldsymbol{r}(t) + \boldsymbol{b}_c), \tag{29}$$

$$\boldsymbol{r}(t) = \boldsymbol{g}^o(t) \odot \sigma_h(\boldsymbol{c}(t)), \tag{30}$$

the effective timescale of individual unit varies dynamically based on its corresponding forget gate value, which in turn is driven by the input sequence. Network unit i is leaky with the time constant of τ when $g_i^f(t) = 0$, but acts as a perfect integrator [Goldman et al., 2008] when $g_i^f(t) = 1$.

Training of vanilla RNNs (both discrete- and continuous-time ones) are made easier by better initialization [Le et al., 2015], among other machine learning innovations for training ANNs. Besides gradient descent through back-propagation, a different line of algorithms to train vanilla RNNs has been developed based on the idea of harnessing chaotic systems with weak perturbations [Jaeger and Haas, 2004]. In particular, the FORCE algorithm [Sussillo and Abbott, 2009] rapidly modifies the output connections of an RNN to match the target using a recursive least square algorithm. The network output is fed back to the RNN, therefore modifying the output connections amounts to a low-rank modification of the recurrent connection matrix.

4 Analyzing and understanding ANNs

Common ANNs used in ML or neuroscience are not easily interpretable. For many neuroscience problems, they may serve better as model systems that await further analyses. Unlike most ML applications, successful training of an ANN on a task is merely the prerequisite for analyzing that network to gain understandings.

Most systems neuroscience techniques to investigate biological neural circuits can be directly applied to understand artificial networks. To facilitate side-by-side comparison between artificial and biological neural networks, activity of an ANN can be visualized and analyzed with the same dimensionality reduction tools (e.g., PCA) used for biological recordings [Mante et al., 2013, Kobak et al., 2016, Williams et al., 2018]. To understand causal relationship from neurons to behavior, arbitrary set of neurons can be lesioned [Yang et al., 2019], or inactivated for a short time duration akin to optogenetic manipulation in physiological experiments. Similarly, connections between two selected groups of neurons can be lesioned to understand the causal contribution of cross-population interactions [Andalman et al., 2019].

In this section, we focus on methods that are unique or particularly useful for analyzing ANNs. These methods include optimization-based tuning analysis [Erhan et al., 2009], fixed-point-based dynamical system analysis [Sussillo and Barak, 2013], quantitative comparisons between a model and experimental data [Yamins et al., 2014], and understanding from the perspective of biological evolution [Lindsey et al., 2019, Richards et al., 2019].

Similarity comparison Analysis methods such as visualization, lesioning, tuning, fixed-point analysis can offer detailed intuition and understanding into neural mechanisms of individual networks. However, with the relative ease of training ANNs, it is possible to train different kinds of neural networks for the same task or dataset [Maheswaranathan et al., 2019, Yamins et al., 2014]. With such volume of data, it is necessary to take advantage of high-throughput quantitative methods that compare different models at scale. Similarity comparison methods compute

a scalar similarity score between the neural activity of two networks performing the same task [Kriegeskorte et al., 2008, Kornblith et al., 2019]. These methods are agnostic about the network form and size, and can be applied to artificial and biological networks alike.

Consider two networks (or two populations of neurons), sized N_1 and N_2 respectively. Their neural activity in response to the same D task conditions can be summarized by a D-by-N_1 matrix $\boldsymbol{R}_1$ and a D-by-N_2 matrix $\boldsymbol{R}_2$. Representational similarity analysis (RSA) [Kriegeskorte et al., 2008] first computes the dissimilarity or distances of neural responses between different task conditions within each network, yielding a D-by-D dissimilarity matrix for each network (Figure 7B). Next, the correlation between dissimilarity matrices of two networks is computed. A higher correlation corresponds to more similar representations.

Another related line of methods uses linear regression (as used in [Yamins et al., 2014]) to predict $\boldsymbol{R}_2$ through a linear transformation of $\boldsymbol{R}_1$, $\boldsymbol{R_2} \approx \boldsymbol{W}\boldsymbol{R_1}$. The similarity corresponds to the correlation between $\boldsymbol{R}_2$ and its predicted value $\boldsymbol{W}\boldsymbol{R_1}$.

Complex tuning analysis Studying tuning properties of single neurons has been one of the most important analysis techniques in neuroscience [Kuffler, 1953]. Classically, tuning properties are studied in sensory areas by showing stimuli parameterized in a low dimensional space (e.g., oriented bars or gratings in vision [Hubel and Wiesel, 1959]). This method is most effective when the neurons studied have relatively simple response properties. A new class of methods treats the mapping of tuning as a high-dimensional optimization problem and directly searches for the stimulus that most strongly activates a neuron. Gradient-free methods such as genetic algorithms have been used to study complex tuning of biological neurons [Yamane et al., 2008]. In deep neural networks, gradient-based methods can be used [Erhan et al., 2009, Zeiler and Fergus, 2014]. For a neuron with activity $r(\boldsymbol{x})$ given input $\boldsymbol{x}$, a gradient-ascent optimization starts with a random $\boldsymbol{x}_0$, and proceeds by updating the input $\boldsymbol{x}$ as

$$\boldsymbol{x} \longrightarrow \boldsymbol{x} + \Delta\boldsymbol{x}; \quad \Delta\boldsymbol{x} = \eta\frac{\partial r}{\partial \boldsymbol{x}}. \tag{31}$$

This method can be used for searching the preferred input to any neuron or any population of neurons in a deep network [Erhan et al., 2009, Bashivan et al., 2019], see Figure 7C for an example. It is particularly useful for studying neurons in higher layers that have more complex tuning properties.

The space of $\boldsymbol{x}$ may be too high dimensional (e.g., pixel space) for conducting an effective search, especially for gradient-free methods. Then we may utilize a lower dimensional space that is still highly expressive. Generative models can learn a function from a lower-dimensional latent space to a higher dimensional space such as pixel space [Kingma and Welling, 2013, Goodfellow et al., 2014]. Then the search can be conducted instead in the lower-dimensional latent space [Ponce et al., 2019].

ANNs can be used to build models for complex behavior that would not be easily done by hand-design, opening up new possibilities for the field such as studying tuning properties of more abstract form of information. For example, Yang et al. [2019] studied the tuning of the structure of rule-guided tasks rather than stimuli.

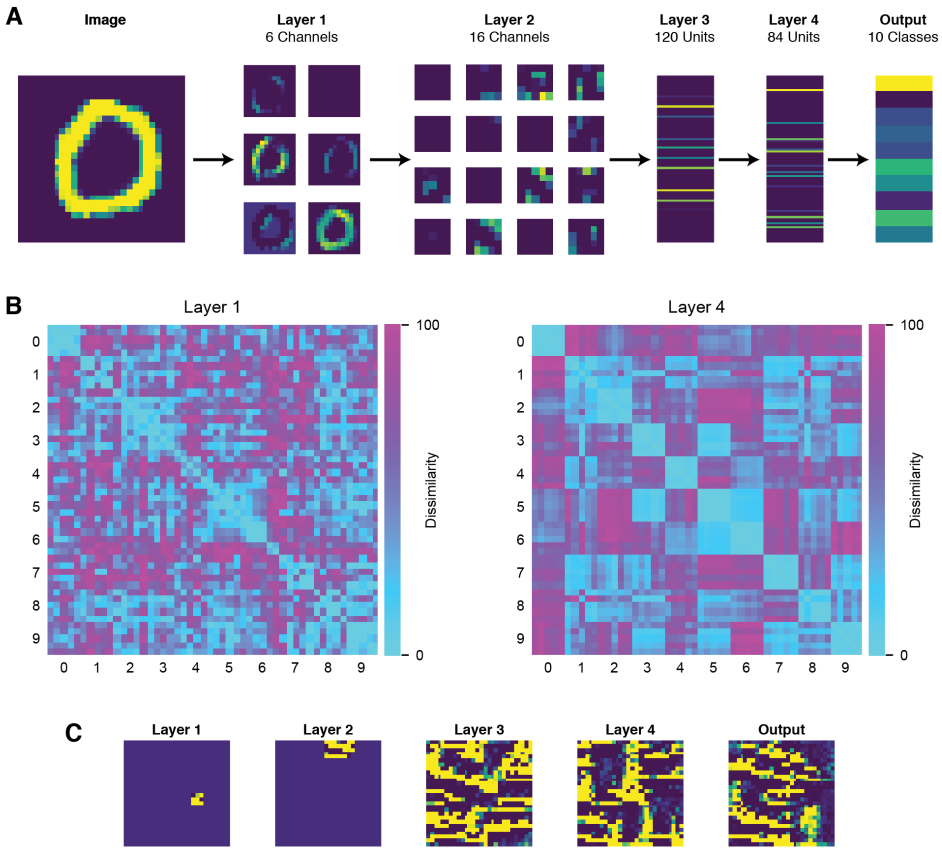

Figure 7: **Convolutional neural network responses and tuning.** (A) The neural response to an image in a convolutional neural network trained to classify handwritten digits. The network consists of two layers of convolutional processing, followed by two fully-connected layers. (B) Dissimilarity matrices (each D-by-D) assessing the similar or dissimilar neural responses to different input images. Dissimilarity matrices are computed for neurons in layers 1 and 4 of the network. $D = 50$ Images are organized by class (0, 1, etc.), 5 images per class. Neural responses to images in the same class are more similar, i.e. neural representation more category-based, in layer 4 (right) than layer 1. (C) Preferred image stimuli found through gradient-based optimization for sample neurons from each layer. Layers 1 and 2 are convolutional, therefore their neurons have localized preferred stimuli. In contrast, neurons from layers 3 and 4 have non-local preferred stimuli.

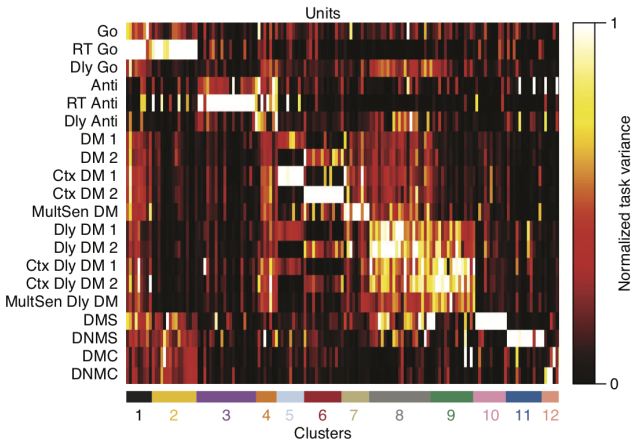

Figure 8: **Analyzing tuning properties of a neural network trained to perform 20 cognitive tasks.** In a network trained on multiple cognitive tasks, the tuning property of model units to individual task can be quantified. x-axis: recurrent units; y-axis: different tasks. Color measures the degree (between 0 and 1) to which each unit is engaged in a task. Twelve clusters are identified using a hierarchical clustering method (bottom, colored bars). For instance, cluster 3 is highly selective for pro- versus anti-response tasks (Anti) involving inhibitory control; clusters 10 and 11 are involved in delayed match-to-sample (DMS) and delayed non-match-to-sample (DNMS), respectively; cluster 12 is tuned to DMC. (Figure adapted from Yang et al. [2019].

An ANN was trained to perform many different cognitive tasks commonly used in animal experiments, including perceptual decision making, working memory, inhibitory control and categorization. Complex network organization is formed by training, in which recurrent neurons display selectivity for a subset of tasks (Figure 8).

Dynamical systems analysis Tuning properties provide a mostly static view of neural representation and computation. To understand how neural networks compute and process information in time, it is useful to study the dynamics of ANNs [Mante et al., 2013, Sussillo and Barak, 2013, Goudar and Buonomano, 2018, Chaisangmongkon et al., 2017].

One useful method to understand dynamics is to study fixed points and network dynamics around them [Strogatz, 2001]. In a generic dynamical system,

$$\frac{d\boldsymbol{r}}{dt} = \boldsymbol{F}(\boldsymbol{r}) \tag{32}$$

a fixed point $\boldsymbol{r}_{\text{ss}}$ is a steady state where the state does not change in time, $F(\boldsymbol{r}_{\text{ss}}) = 0$. The network dynamics at a state $\boldsymbol{r} = \boldsymbol{r}_{\text{ss}} + \Delta\boldsymbol{r}$ around a fixed point $\boldsymbol{r}_{\text{ss}}$ is approximately linear,

$$\frac{d\boldsymbol{r}}{dt} = \boldsymbol{F}(\boldsymbol{r}) = \boldsymbol{F}(\boldsymbol{r}_{\text{ss}} + \Delta\boldsymbol{r}) \approx \boldsymbol{F}(\boldsymbol{r}_{\text{ss}}) + J(\boldsymbol{r}_{\text{ss}})\Delta\boldsymbol{r}, \quad \frac{d\Delta\boldsymbol{r}}{dt} = J(\boldsymbol{r}_{\text{ss}})\Delta\boldsymbol{r}. \tag{33}$$

where J is the Jacobian of $\boldsymbol{F}$, $J_{ij} = \partial F_i / \partial r_j$, evaluated at $\boldsymbol{r}_{\text{ss}}$. This is a linear system which can be understood more easily. In ANNs, these fixed points can be

found by gradient-based optimization [Sussillo and Barak, 2013],

$$\text{argmin}_{\boldsymbol{r}}||\boldsymbol{F}(\boldsymbol{r}||^2. \tag{34}$$

Fixed points are particularly useful for understanding how networks store memories, accumulate information [Mante et al., 2013], and transition between discrete states [Chaisangmongkon et al., 2017]. This point can be illustrated in a network trained for a simple parametric working memory task [Romo et al., 1999] that requires keeping a scalar value in memory for several seconds. In the state-space of this network, neural trajectories during the delay period converge to different fixed points depending on the stored value (Figure 9A). These fixed points form an approximate line attractor [Seung, 1996] during the delay period (Figure 9B).

There is a dearth of examples in computational neuroscience that accounts for not just a single aspect of neural representation or dynamics, but a sequence of computation to achieve a complex task. ANNs offer a new tool to confront this difficulty. In a recurrent network trained to perform the DMC task (Figure 1C-E) [Chaisangmongkon et al., 2017], the trajectory of recurrent neural population in the state space reveals how computation is carried out through epochs of the task (Figure 9C).

Understanding neural circuits from objectives, architecture, and training All above methods seek a mechanistic understanding of ANNs after training. Another form of understanding links the three basic ingredients in deep learning: tasks/objectives, network architecture, and training algorithm to the solution after training [Richards et al., 2019]. This approach is similar to understanding biology from an evolutionary or developmental perspective, which links environments to functions in biological organisms. It can help explain the computational benefit or necessity of observed structures or functions. For example, compared to purely feedforward networks, recurrently-connected deep networks are better at predicting responses of higher visual area neurons to behaviorally challenging images [Kar et al., 2019]. This suggests a contribution of recurrent connections to classifying difficult images in the brain.

While re-running the biological processes of development and evolution may be difficult, re-training networks with different objectives, architectures, and algorithms is fairly straightforward thanks to recent advances in ML. For neuroscience applications, whenever training of an ANN leads to a conclusion, it is good practice to vary hyperparameters describing the basic ingredients (to a reasonable degree) to explore the necessary and sufficient conditions for the conclusion [Orhan and Ma, 2019, Yang et al., 2019, Lindsey et al., 2019].

The link from the three ingredients to the network solution is typically not rigorous. However, in certain simplified cases, the link can be firmly established by solving the training process analytically [Saxe et al., 2013, 2019].

5 Biologically realistic network architectures and learning

Although neuroscientists and cognitive scientists have had much success with standard neural network architectures (e.g., LSTM) and/or training algorithms (e.g.,

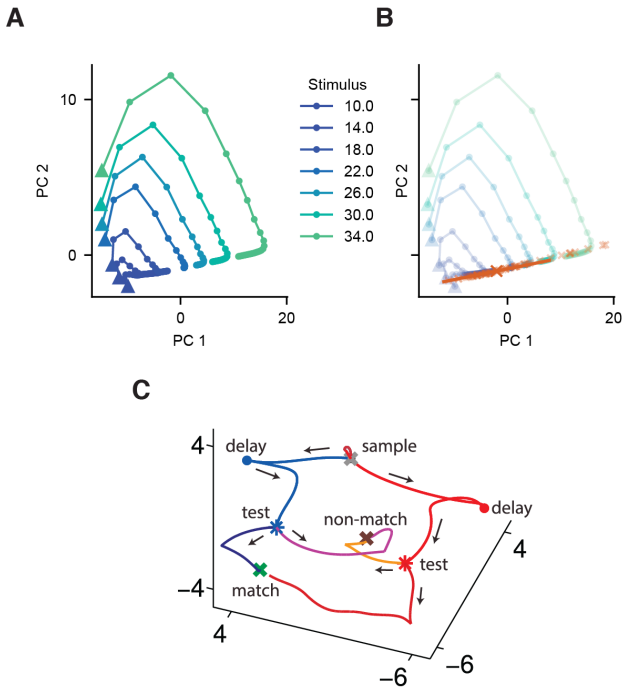

Figure 9: **Understanding network computation through state-space and dynamical system analysis.** (A-B) In a simple parametric working memory task [Romo et al., 1999], the network needs to memorize the value of a stimulus through a delay period. The network can achieve such parametric working memory by developing a line attractor. (A) Trial-averaged neural activity in the PCA space for different stimulus values. Triangles indicate the start of the delay period. (B) Fixed points found through optimization (orange cross). The direction of a line attractor can be estimated by finding the eigenvector with a corresponding eigenvalue close to 0. The orange line shows the line attractor estimated around one of the fixed points. (C) Neural dynamics of a recurrent neural network underlying the performance of the DMC task (Figure 1C). The portions of the trajectory in blue versus red correspond to trials when the sample belongs to category A or B, respectively, The final decision, match (AA or BB) or non-match (AB or BA) corresponds to distinct attractor states located at separate positions in the state space. Similar trajectories of population activity have been found in neurophysiological monkey experiments. Figure adapted from Chaisangmongkon et al. [2017].

SGD) used in machine learning, for many neuroscience questions, it is critical to build network architectures and utilize learning algorithms that are more biologically relevant. In this section, we outline methods to build networks with more biologically realistic structures, canonical computations, and plasticity rules.

5.1 Structured connections

Modern neuroscience experiments routinely record from multiple brain areas and/or multiple cell types during the same animal behavior. Computational efforts modeling these findings can be greatly facilitated by incorporating into neural networks fundamental biological structures, such as cell-type-specific connectivity and long-range connections across model areas/layers.

In common recurrent networks, the default connectivity is all-to-all. In contrast, both local and long-range connectivity in biological neural systems are sparse. Sparse connectivity can be strictly imposed with a non-trainable sparse mask $\boldsymbol{M}$ that element-wise multiplies a trainable matrix $\widetilde{\boldsymbol{W}}$ to obtain the effective connectivity, $\boldsymbol{W} = \widetilde{\boldsymbol{W}} \odot \boldsymbol{M}$. A soft sparsity constraint can be imposed with a L1 regularization term $\beta \sum_{ij} |W_{ij}|$ added to the loss function, here the scalar coefficient β controls the strength of the sparsity constraint.

To model cell-type-specific findings, it is important to build neural networks with multiple cell types. A vanilla recurrent network (Eq. 9) (or any other network) can be easily modified to obey Dale's law by separating excitatory and inhibitory neurons [Song et al., 2016],

$$\frac{d\boldsymbol{r}^E}{dt} = -\boldsymbol{r}^E + f_E(\boldsymbol{W}_{EE}\boldsymbol{r}^E - \boldsymbol{W}_{EI}\boldsymbol{r}^I + \boldsymbol{W}_{Ex}\boldsymbol{x} + \boldsymbol{b}^E), \tag{35}$$

$$\frac{d\boldsymbol{r}^I}{dt} = -\boldsymbol{r}^I + f_I(\boldsymbol{W}_{IE}\boldsymbol{r}^E - \boldsymbol{W}_{II}\boldsymbol{r}^I + \boldsymbol{W}_{Ix}\boldsymbol{x} + \boldsymbol{b}^I), \tag{36}$$

where an absolute function $|\cdot|$ constrains signs of the connection weights, e.g, $\boldsymbol{W}_{EE} = |\widetilde{\boldsymbol{W}}_{EE}|$. After training an ANN to perform the classical random dot motion direction discrimination task [Roitman and Shadlen, 2002], one can "open the box" and examine the resulting "wiring diagram" of recurrent network connectivity pattern (Figure 10). With the incorporation of the Dale's law, the connectivity emerging from training is a heterogeneous version of that of a biologically-based structured network model of decision-making [Wang, 2002], demonstrating that machine learning brought closer to brain's hardware can indeed be used to shed insights into biological neural networks.

The extensive long-range connectivity across brain areas [Felleman and Van Essen, 1991, Markov et al., 2014, Oh et al., 2014] can be included in ANNs. In classical convolutional neural networks [LeCun et al., 1990, Krizhevsky et al., 2012], each layer only receives feedforward inputs from the immediate preceding layer. However, in many recent networks, each layer also receives feedforward inputs from much earlier layers [Huang et al., 2017, He et al., 2016]. In convolutional recurrent networks, neurons in each layer further receive feedback inputs from later layers and local recurrent connections [Nayebi et al., 2018, Kietzmann et al., 2019].

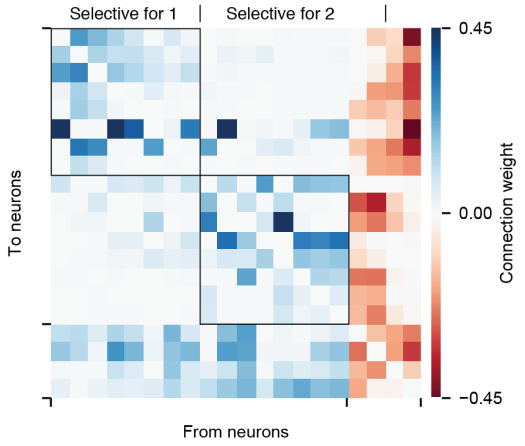

Figure 10: **Training networks with or without Dale's law.** Connectivity matrices of a recurrent network trained on a perceptual decision making task similar to the one in Figure 2C. The network respects Dale's law by having separate groups of excitatory (blue) and inhibitory (red) neurons. Only showing connectivity between neurons with high stimulus selectivity. Neurons are sorted based on their stimulus selectivity to choice 1 and 2. Recurrent excitatory connections between neurons selective to the same choice are indicated by the squares.

5.2 Canonical computations

Neuroscientists have identified several canonical computations that are carried out across a wide range of brain areas, including attention, normalization, and gating. Here we discuss how such canonical computations can be introduced into neural networks. They function as modular architectural components that can be plugged into many networks. Interestingly, the canonical computations mentioned above all have their parallels in ML-based neural networks. We will highlight the differences and similarities between the purely ML implementations and the more biological ones.

Normalization Divisive normalization is widely observed in biological neural systems [Carandini and Heeger, 2012]. In divisive normalization, activation of a neuron r_i is no longer determined by its immediate input I_i, $r_i = f(I_i)$. Instead, it is normalized by the average inputs $\langle I_j \rangle_j$ to a broader pool of neurons called the normalization pool,

$$r_i = f(\gamma \frac{I_i}{\langle I_j \rangle_j + \sigma}). \tag{37}$$

The specific choice of a normalization pool depends on the system studied. Biologically, although synaptic inputs are additive in the drive to neurons, feedback inhibition can effectively produce normalization [Ardid et al., 2007]. This form of divisive normalization is differentiable. So it can be directly incorporated into ANNs,

although the impact of divisive normalization on training performance in common neural networks remains unclear.

Normalization is also a critical part of many neural networks in machine learning. Similar to divisive normalization, ML-based normalization methods [Ioffe and Szegedy, 2015, Ba et al., 2016b, Ulyanov et al., 2016, Wu and He, 2018] aim at putting neuronal responses into a range appropriate for downstream areas to process. Unlike divisive normalization, the mean inputs to a pool of neurons is usually subtracted from, instead of dividing, the immediate input (Eq. 13). These methods also compute the standard deviation of inputs to the normalization pool, a step that may not be biologically plausible. Different ML-based normalization methods are distinguished based on their choice of a normalization pool.

Attention Attention has been extensively studied in neuroscience [Desimone and Duncan, 1995, Carrasco, 2011]. Computational models are able to capture various aspects of bottom-up [Koch and Ullman, 1987] and top-down attention [Ardid et al., 2007, Reynolds and Heeger, 2009]. In computational models, top-down attention usually takes the form of a multiplicative gain field to the activity of a specific group of neurons. In the case of spatial attention, consider a group of neurons, each with a preferred spatial location x_i, and pre-attention activity $\widetilde{r}(x_i)$ for a certain stimulus. The attended spatial location x_q results in attentional weights $\alpha_i(x_q)$, which is higher if x_q is similar to x_i. The attentional weights can then be used to modulate the neural response of neuron i, $r_i(x_q) = \alpha_i(x_q)\widetilde{r}(x_i)$. Similarly, feature attention strengthens the activity of neurons that are selective to the attended features (e.g., specific color). Such top-down spatial and feature attention can be included in convolutional neural networks [Lindsay and Miller, 2018, Yang et al., 2018].

Meanwhile, attention has become widely used in machine learning [Bahdanau et al., 2014, Xu et al., 2015, Lindsay, 2020], constituting a standard component in recent natural language processing models [Vaswani et al., 2017]. Although the ML attention mechanisms appear rather different from attention models in neuroscience, as we will show below, the two mechanisms are very closely related.

In deep learning, attention can be viewed as a differentiable dictionary retrieval process. A regular dictionary stores a number of key-value pairs (e.g. word-explanation pairs) $\{(\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)})\}$. Using a dictionary, such as looking up explanation of a queried word, involves searching for the key $\boldsymbol{k}^{(j)}$ that matches the query $\boldsymbol{q}$, $\boldsymbol{k}^{(j)} = \boldsymbol{q}$, and retrieving the corresponding value $\boldsymbol{y} = \boldsymbol{v}^{(j)}$. This process can be thought of as modulating each value $\boldsymbol{v}^{(i)}$ based on an attentional weight α_i that measures the similarity between the key $\boldsymbol{k}^{(i)}$ and the query $\boldsymbol{q}$, and outputting the sum of all modulated values,

$$\alpha_i = \begin{cases} 1, & \text{if } \boldsymbol{k}^{(i)} = \boldsymbol{q} \\ 0, & \text{otherwise} \end{cases}, \tag{38}$$

$$\boldsymbol{y} = \sum_i \alpha_i \boldsymbol{v}^{(i)}. \tag{39}$$

In the above case of spatial attention, the i-th key-value pair is $(x_i, \widetilde{r}(x, x_i))$, while the query is the attended spatial location x_q. Each neuron's response is modulated

based on how similar its preferred spatial location (its value) x_i is to the attended location (the query) x_q.

The use of ML attention makes the query-key comparison and the value-retrieval process differentiable. A query is compared with every key vector $\boldsymbol{k}^{(i)}$ to obtain an attentional weight (normalized similarity score) α_i,

$$c_i = \text{score}(\boldsymbol{q}, \boldsymbol{k}^{(i)}), \tag{40}$$

$$\alpha_1, \cdots, \alpha_N = \text{normalize}(c_1, \cdots, c_N), \tag{41}$$

Here the similarity scoring function can be a simple inner product, $\text{score}(\boldsymbol{q}, \boldsymbol{k}^{(i)}) = \boldsymbol{q}^\top \boldsymbol{k}^{(i)}$, and the normalization function can be a softmax function,

$$\alpha_i = \frac{e^{c_i}}{\sum_j e^{c_j}}. \tag{42}$$

Gating An important computation for biological neural systems is gating [Abbott, 2006, Wang and Yang, 2018]. Gating refers to the idea of controlling information flow without necessarily distorting its content. Gating in biological systems can be implemented with various mechanisms. One major mechanism, gain modulation, multiplies inputs to neurons by a gain factor [Salinas and Thier, 2000, Olsen et al., 2012]. Although basic feedforward and recurrent networks use additive neurons, where all inputs to a neuron is added together, multiplicative gating is essential for popular recurrent networks such as LSTMs (Eq. 28) [Hochreiter and Schmidhuber, 1997] and GRUs [Cho et al., 2014].

Predictive coding Another canonical computation proposed for the brain is to compute predictions [Rao and Ballard, 1999, Bastos et al., 2012, Heilbron and Chait, 2018]. In predictive coding, neural systems constantly tries to make inference about the external world. Brain areas will selectively propagate information that is unpredicted or surprising, while suppressing responses to expected stimuli. To implement predictive coding in ANNs, feedback connections from higher layers can be trained with a separate loss that compares the output of feedback connections with the neural activity in lower layers [Lotter et al., 2016, Sacramento et al., 2017]. In this way, feedback connections will learn to predict the activity of lower areas. The feedback inputs will then be used to inhibit neural activity in lower layers.

5.3 Learning and plasticity

Biological neural systems are products of evolution, development, and learning. In contrast, traditional ANNs are trained with SGD-based rules mostly from scratch. The back-propagation algorithm of computing gradient descent is well known to be biologically implausible [Zipser and Andersen, 1988]. Incorporating more realistic learning processes can help us build better models of brains.

Selective training and continual learning In typical ANNs, all connections are trained. However, in biological neural systems, synapses are not equally modifiable. Many synapses can be stable for years [Grutzendler et al., 2002, Yang et al., 2009]. To implement selective training of connections, the effective connection weight

$\boldsymbol{W}$ can be expressed as a sum of a sparse trainable synaptic weight matrix and a non-trainable one, $\boldsymbol{W} = \boldsymbol{W}_{\text{train}} + \boldsymbol{W}_{\text{fix}}$ [Rajan et al., 2016, Masse et al., 2018]. Or more generally, selective training can be imposed softly by adding to the loss a regularization term L_{reg} that makes it more difficult to change the weights of certain connections,

$$L_{\text{reg}} = \beta \|(\boldsymbol{W} - \boldsymbol{W}_{\text{fix}}) \odot \boldsymbol{M}\|_2. \tag{43}$$

Here, elements in mask $\boldsymbol{M}$ determine how strongly corresponding connections in $\boldsymbol{W}$ should stick close to the values in $\boldsymbol{W}_{\text{fix}}$.

Selective training of connections through this form of soft constraints has been used by continual learning techniques to combat catastrophic forgetting. The phenomenon of catastrophic forgetting is commonly observed when ANNs are learning new tasks, they tend to rapidly forget previous learned tasks that are not revisited [McCloskey and Cohen, 1989]. One major class of continual learning methods deals with this issue by selectively training synaptic connections that are deemed unimportant for previously learned tasks or knowledge, while protecting the important ones [Kirkpatrick et al., 2017, Zenke et al., 2017].

Hebbian plasticity The predominant idea for biological learning is Hebbian plasticity [Hebb, 2005] and its variants [Song et al., 2000, Bi and Poo, 2001]. Hebbian plasticity alone can drive learning of connection weights. However, machine learning techniques, especially those based on SGD, can be combined with Hebbian plasticity to develop ANNs that are both more powerful for certain tasks and more biologically realistic.

There are two methods to combine Hebbian plasticity with SGD. In the first kind, the effective connection matrix $\boldsymbol{W} = \widetilde{\boldsymbol{W}} + \boldsymbol{A}$ is the sum of two connection matrices, $\widetilde{\boldsymbol{W}}$ trained by SGD, and $\boldsymbol{A}$ driven by Hebbian plasticity [Ba et al., 2016a, Miconi et al., 2018],

$$\boldsymbol{A}(t+1) = \lambda \boldsymbol{A}(t) + \eta \boldsymbol{r}\boldsymbol{r}^T. \tag{44}$$

Or in component-form,

$$A_{ij}(t+1) = \lambda A_{ij}(t) + \eta r_i r_j. \tag{45}$$

In addition to training a separate matrix, SGD can be used to learn the plasticity rules itself [Bengio et al., 1992, Metz et al., 2018]. Here, the plasticity rule is a trainable function of pre- and post-synaptic activity,

$$A_{ij}(t+1) = \lambda A_{ij}(t) + f(r_i, r_j, \boldsymbol{\theta}). \tag{46}$$

Since the system is differentiable, parameters $\boldsymbol{\theta}$, which collectively describe the plasticity rules, can be updated with SGD-based methods. In its simplest form, $f(r_i, r_j, \boldsymbol{\theta}) = \eta r_i r_j$, where $\boldsymbol{\theta} = \{\eta\}$. Here, the system can learn to become Hebbian ($\eta > 0$) or anti-Hebbian ($\eta < 0$). Learning of a plasticity rule is a form of meta-learning (or learning-to-learn), where a meta-learning algorithm (here, SGD) is used to optimize an inner learning rule (here, Hebbian plasticity rule).

Such Hebbian plasticity networks can be extended to include more complex synapses with multiple hidden variables in a “cascade model" of synaptic plasticity [Fusi et al.,

2005]. In theory, properly designed complex synapses can substantially boost a neural network's memory capacity [Benna and Fusi, 2016]. Models of such complex synapses are differentiable, and therefore can be incorporated into ANNs [Kaplanis et al., 2018].

Short-term plasticity In addition to Hebbian plasticity that acts on the time scales from hours to years, biological synapses are subject to short-term plasticity mechanisms operating on the timescale of 100s of milliseconds [Zucker and Regehr, 2002] that can rapidly modify their effective weights. Classical short-term plasticity rules [Mongillo et al., 2008, Markram et al., 1998] are formulated with spiking neurons, but they can be adapted to rate forms. In these rules, each connection weight $w = \widetilde{w}ux$ is a product of an original weight $\widetilde{w}$, a facilitating factor u, and a depressing factor x. The facilitating and depressing factors are both influenced by the pre-synaptic activity $r(t)$,

$$\frac{dx}{dt} = \frac{1 - x(t)}{\tau_x} - u(t)x(t)r(t), \tag{47}$$

$$\frac{du}{dt} = \frac{U - u(t)}{\tau_u} + U(1 - u(t))r(t). \tag{48}$$

High pre-synaptic activity $r(t)$ increases the facilitating factor $u(t)$ and decreases the depressing factor $x(t)$. Again, the equations governing short-term plasticity are fully differentiable, so they can be incorporated into ANNs in the same way as Hebbian plasticity rules [Masse et al., 2019].

Masse et al. [2019] offers an illustration of how ANNs can be used to test new hypotheses in neuroscience. It was designed to investigate the neural mechanisms of working memory, the brain's ability to maintain and manipulate information internally in the absence of external stimulation. Working memory has been extensively studied in animal experiments using delayed response tasks, in which a stimulus and its corresponding motor response are separated by a temporal gap when the stimulus must be retained internally. Stimulus-selective self-sustained persistent activity during a mnemonic delay is amply documented and held as the neural substrate of working memory representation [Goldman-Rakic, 1995, Wang, 2001]. However, recent studies suggested that certain short-term memory trace may be realized by hidden variables instead of spiking activity, such as synaptic efficacy that by virtue of short-term plasticity represents past events. When an ANN endowed with short-term synaptic plasticity is trained to perform a delayed response task, it does not make an *a priori* assumption about whether working memory is represented by hidden synaptic efficacy or neural activity. It was found that activity-silent state can accomplish such a task only when the delay is sufficiently short, whereas persistent activity naturally emerges from training with delay periods longer than the biophysical time constants of short-term synaptic plasticity. More importantly, training always gives rise to persistent activity even with a short period when information must be manipulated internally, such as mentally rotating a directional stimulus by 90 degrees. Therefore, ANNs can contribute to resolving important debates in neuroscience.

Biologically-realistic gradient descent Backpropagation is commonly viewed as biologically unrealistic because the plasticity rule is not local (see Figure 3).

Efforts have been devoted to approximating gradient descent with algorithms more compatible with the brain's hardware [Lillicrap et al., 2016, Guerguiev et al., 2017, Roelfsema and Holtmaat, 2018, Lillicrap et al., 2020].

In feedforward networks, the backpropagation algorithm can be implemented with physical synaptic connections feeding back from the top layer [Xie and Seung, 2003]. This implementation requires the feedback connections to precisely mirror the feedforward connections. This requirement can be relaxed. Having random feedback connections does not compute the gradient exactly, but still allows for training loss to be decreased on average [Lillicrap et al., 2016]. Another challenge of approximating backpropagation with feedback connections is that the feedback inputs carrying loss information need to be processed differently from feedforward inputs carrying stimulus information. This issue can be addressed by introducing multi-compartmental neurons into ANNs [Guerguiev et al., 2017]. In such networks, feedforward and feedback inputs are processed differently because they are received by the model neurons' soma and dendrites respectively.

These methods of implementing the backpropagation algorithm through physical synapses propagating information backwards are so far only used for feedforward networks. For recurrent networks, the backpropagation algorithm propagates information backwards in time. Therefore, it is not clear how to interpret the backpropagating signal in terms of physical synapses. Instead, approximations can be made such that the network computes approximated gradient information as it runs forward in time [Williams and Zipser, 1989, Murray, 2019].

Finally, for many neuroscience applications, it is probably not necessary to justify backpropagation by neurobiology; ANNs start as "blank slate", thus training by backpropagation is tasked to accomplish what for the brain amounts to a combination of genetic programming, development and plasticity in adulthood.

6 Future directions and conclusion

ANN models of brains are rapidly becoming more biologically relevant and predictive. We have reviewed many of these efforts in the section *Biologically realistic network architectures and learning*. In this final section, we outline other existing challenges and ongoing work to make ANNs better models of brains.

Spiking neural networks Most biological neurons communicate with spikes. Harnessing the power of machine learning algorithms for spiking networks remains a daunting challenge. Gradient-descent-based training techniques typically require the system to be differentiable, making it challenging to train spiking networks, because spike generation is non-differentiable. However, several recent methods have been proposed to train spiking networks with gradient-based techniques [Zenke and Ganguli, 2018, Nicola and Clopath, 2017, Kim et al., 2019]. These methods generally involve approximating spiking networks with a differentiable system [Tavanaei et al., 2019].

Techniques in training spiking networks could prove increasingly important and practical as neuromorphic hardware that operates naturally with spikes become more powerful [Merolla et al., 2014, Pei et al., 2019].

Consistent protocols for developing brain-like recurrent networks In the study of mammalian visual systems, the use of large datasets such as ImageNet has consistently produced neural networks resembling the brain in many aspects. The same has not been shown for most other systems. Although many studies have shown success using neural networks to model cognitive and motor systems, each work usually has its own set of network architectures, training protocols, and other hyperparameters. Simply applying the most common architectures and training algorithms does not consistently lead to brain-like recurrent networks [Sussillo et al., 2015]. Much work remains to be done to search for datasets/tasks, network architectures, and training regimes that can produce brain-resembling artificial networks across a wide range of experimental tasks.

Detailed behavioral and physiological predictions Although many studies have reported similarities between brains and ANNs, more detailed comparisons have revealed striking differences. Deep convolutional networks can achieve similar or better performance on large image classification tasks compared to humans, however, the mistakes they make can be very different from the ones made by humans [Szegedy et al., 2013, Rajalingham et al., 2018]. While neural representations of natural images are overall similar between deep convolutional networks and mammalian visual systems, natural videos are represented by much smoother trajectories in humans compared to mainstream convolutional networks [Hénaff et al., 2019]. It will be important for future ANN models of brains to aim at simultaneously explaining a wider range of physiological and behavioral phenomena.

Interpreting learned networks and learning processes With the ease of training neural networks comes the difficulty of analyzing them. Granted, neuroscientists are not foreign to analysis of complex networks, and ANNs are still technologically easier to analyze compared to biological neural networks. However, compared to network models with built-in regularities and small numbers of free parameters, deep neural networks are notoriously complex to analyze and understand, and will likely become even more so as we build more and more sophisticated neural networks. This difficulty is rooted in the use of optimization algorithms to search for parameter values. Since the optimization process in deep learning has no unique optima, the results of optimization will necessarily be lacking the degree of regularities built in hand-designed models. Although we can attempt to understand ANNs from the perspective of its objectives, architectures, and training algorithms [Richards et al., 2019], which are described with a much smaller number of parameters, the link from these parameters to network representation, mechanism, and behavior is mostly informal and based on intuition.

Conclusion Artificial neural networks present a novel approach in computational neuroscience. They have already been used, with certain degree of success, to model various aspects of sensory, cognitive, and motor circuits. Efforts are underway

in making ANNs more biologically relevant and applicable to a wider range of neuroscientific questions. In a sense, ANNs can be viewed as model systems like fruit flies, mice, and monkeys, but are easily carried out to explore new task paradigms and computational ideas. Of course, one can be skeptical about RNNs as model systems, on the ground that they are not biological organisms. However, computational models span a wide range of biological realism; there should be no doubt that brain research will benefit from enhanced interactions with machine learning and artificial intelligence. In order for ANNs to have a broad impact in neuroscience, it will be important to devote our efforts in two areas. First, we should continue to bring RNNs closer to neurobiology. Second, we should endeavour to "open the box" thoroughly after learning to identify neural dynamics and network connectivity that emerge from learning, leading to testable insights and predictions by neurobiological experiments. Recurrent neural dynamics emphasized in this Primer are of central importance for understanding the brain mechanisms of cognition, further development of ANNs in this direction will contribute to acceleration of progress in neuroscience.

Acknowledgments: This work was supported by the Simons Foundation, NSF NeuroNex Award DBI-1707398 and the Gatsby Charitable Foundation to GRY; the ONR grant N00014-17-1-2041 to XJW.

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

L. Abbott. Where are the switches on this thing. *23 problems in systems neuroscience*, pages 423–31, 2006.

L. Abbott and F. S. Chance. Drivers and modulators from push-pull and balanced synaptic input. *Progress in brain research*, 149:147–155, 2005.

L. F. Abbott. Theoretical neuroscience rising. *Neuron*, 60:489–495, 2008.

A. S. Andalman, V. M. Burns, M. Lovett-Barron, M. Broxton, B. Poole, S. J. Yang, L. Grosenick, T. N. Lerner, R. Chen, T. Benster, et al. Neuronal dynamics regulating brain and behavioral state transitions. *Cell*, 177(4):970–985, 2019.

S. Ardid, X.-J. Wang, and A. Compte. An integrated microcircuit model of attentional processing in the neocortex. *J. Neurosci.*, 27:8486–8495, 2007.

J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016a.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016b.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

H. B. Barlow et al. Possible principles underlying the transformation of sensory messages. *Sensory communication*, 1:217–234, 1961.

P. Bashivan, K. Kar, and J. J. DiCarlo. Neural population control via deep image synthesis. *Science*, 364(6439):eaav9436, 2019.

A. M. Bastos, W. M. Usrey, R. A. Adams, G. R. Mangun, P. Fries, and K. J. Friston. Canonical microcircuits for predictive coding. *Neuron*, 76:695–711, 2012.

S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2. Univ. of Texas, 1992.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

M. K. Benna and S. Fusi. Computational principles of synaptic memory consolidation. *Nature neuroscience*, 19(12):1697, 2016.

G. Bi and M. Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annu Rev Neurosci*, 24:139–166, 2001.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

K. H. Britten, M. N. Shadlen, W. T. Newsome, and J. A. Movshon. The analysis of visual motion: a comparison of neuronal and psychophysical performance. *Journal of Neuroscience*, 12(12):4745–4765, 1992.

C. F. Cadieu, H. Hong, D. L. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, and J. J. DiCarlo. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS computational biology*, 10(12): e1003963, 2014.

M. Carandini and D. J. Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51, 2012.

M. Carrasco. Visual attention: The past 25 years. *Vision research*, 51(13):1484–1525, 2011.

W. Chaisangmongkon, S. K. Swaminathan, D. J. Freedman, and X.-J. Wang. Computing by robust transience: how the fronto-parietal network performs sequential, category-based decisions. *Neuron*, 93(6):1504–1517, 2017.

T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

J. D. Cohen, K. Dunbar, and J. L. McClelland. On the control of automatic processes: a parallel distributed processing account of the stroop effect. *Psychological review*, 97(3):332, 1990.

C. J. Cueva and X.-X. Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. *arXiv preprint arXiv:1803.07770*, 2018.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

R. Desimone and J. Duncan. Neural mechanisms of selective visual attention. *Annual review of neuroscience*, 18(1):193–222, 1995.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111): 1202–1205, 2012.

J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex (New York, NY: 1991)*, 1(1):1–47, 1991.

D. J. Freedman and J. A. Assad. Experience-dependent representation of visual categories in parietal cortex. *Nature*, 443(7107):85, 2006.

J. Freeman and E. P. Simoncelli. Metamers of the ventral stream. *Nature neuroscience*, 14(9):1195, 2011.

K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15:455–469, 1982.

K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834, 1983.

S. Fusi, P. J. Drew, and L. F. Abbott. Cascade models of synaptically stored memories. *Neuron*, 45(4):599–611, 2005.

A. Gers Felix, S. Jurgen, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

J. I. Gold and M. N. Shadlen. The neural basis of decision making. *Annual review of neuroscience*, 30, 2007.

M. Goldman, A. Compte, and X.-J. Wang. Neural integrators: recurrent mechanisms and models. In L. Squire, T. Albright, F. Bloom, F. Gage, and N. Spitzer, editors, *Encyclopedia of Neuroscience*, pages 165–178. MacMillan Reference Ltd., 2008.

P. S. Goldman-Rakic. Cellular basis of working memory. *Neuron*, 14:477–485, 1995.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

V. Goudar and D. V. Buonomano. Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks. *Elife*, 7:e31134, 2018.

J. Grutzendler, N. Kasthuri, and W.-B. Gan. Long-term dendritic spine stability in the adult cortex. *Nature*, 420(6917):812–816, 2002.

J. Guerguiev, T. P. Lillicrap, and B. A. Richards. Towards deep learning with segregated dendrites. *ELife*, 6:e22901, 2017.

D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95:245–258, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

M. Heilbron and M. Chait. Great expectations: is there evidence for predictive coding in auditory cortex? *Neuroscience*, 389:54–73, 2018.

M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168, 2013.

O. J. Hénaff, R. L. Goris, and E. P. Simoncelli. Perceptual straightening of natural videos. *Nature neuroscience*, 22(6):984–991, 2019.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol. (Lond.)*, 160:106–154, 1962.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

M. Januszewski, J. Kornfeld, P. H. Li, A. Pope, T. Blakely, L. Lindsey, J. Maitin-Shepard, M. Tyka, W. Denk, and V. Jain. High-precision automated reconstruction of neurons with flood-filling networks. *Nature methods*, 15(8):605, 2018.

J. P. Jones and L. A. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of neurophysiology*, 58(6):1233–1258, 1987.

N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

C. Kaplanis, M. Shanahan, and C. Clopath. Continual reinforcement learning with complex synapses. *arXiv preprint arXiv:1802.07239*, 2018.

K. Kar, J. Kubilius, K. Schmidt, E. B. Issa, and J. J. DiCarlo. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nature neuroscience*, page 1, 2019.

S.-M. Khaligh-Razavi and N. Kriegeskorte. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS computational biology*, 10 (11):e1003915, 2014.

R. Kiani and M. N. Shadlen. Representation of confidence associated with a decision by neurons in the parietal cortex. *science*, 324(5928):759–764, 2009.

T. C. Kietzmann, C. J. Spoerer, L. K. Sörensen, R. M. Cichy, O. Hauk, and N. Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43):21854–21863, 2019.

R. Kim, Y. Li, and T. J. Sejnowski. Simple framework for constructing functional spiking recurrent neural networks. *Proceedings of the National Academy of Sciences*, 116(45):22811–22820, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

D. Kobak, W. Brendel, C. Constantinidis, C. E. Feierstein, A. Kepecs, Z. F. Mainen, X.-L. Qi, R. Romo, N. Uchida, and C. K. Machens. Demixed principal component analysis of neural population data. *Elife*, 5:e10989, 2016.

C. Koch and S. Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. In *Matters of intelligence*, pages 115–141. Springer, 1987.

S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

N. Kriegeskorte, M. Mur, and P. A. Bandettini. Representational similarity analysis-connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4, 2008.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

S. W. Kuffler. Discharge patterns and functional organization of mammalian retina. *Journal of neurophysiology*, 16(1):37–68, 1953.

Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Y. LeCun. A theoretical framework for back-propagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28. Burlington, MA: Morgan Kaufmann, 1988.

Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. In M. A. Arbib, editor, *The handbook of brain theory and neural networks*, pages 255–258. Cambridge, MA: MIT Press, 1995.

Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, pages 1–12, 2020.

G. W. Lindsay. Attention in psychology, neuroscience, and machine learning. *Frontiers in Computational Neuroscience*, 14:29, 2020.

G. W. Lindsay and K. D. Miller. How biological attention mechanisms improve task performance in a large-scale visual system model. *eLife*, 7:e38105, 2018.

J. Lindsey, S. A. Ocko, S. Ganguli, and S. Deny. A unified theory of early visual representations from retina to cortex through anatomically constrained deep cnns. *arXiv preprint arXiv:1901.00945*, 2019.

W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

N. Maheswaranathan, A. H. Williams, M. D. Golub, S. Ganguli, and D. Sussillo. Universality and individuality in neural dynamics across large populations of recurrent networks. *arXiv preprint arXiv:1907.08549*, 2019.

V. Mante, D. Sussillo, K. V. Shenoy, and W. T. Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78, 2013.

N. T. Markov, M. M. Ercsey-Ravasz, A. R. Ribeiro Gomes, C. Lamy, L. Magrou, J. Vezoli, P. Misery, A. Falchier, R. Quilodran, M. A. Gariel, J. Sallet, R. Gamanut, C. Huissoud, S. Clavagnier, P. Giroud, D. Sappey-Marinier, P. Barone, C. Dehay, Z. Toroczkai, K. Knoblauch, D. C. Van Essen, and H. Kennedy. A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cereb. Cortex*, 24:17–36, 2014.

H. Markram, Y. Wang, and M. Tsodyks. Differential signaling via the same axon of neocortical pyramidal neurons. *Proceedings of the National Academy of Sciences*, 95(9):5323–5328, 1998.

N. Y. Masse, G. D. Grant, and D. J. Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018.

N. Y. Masse, G. R. Yang, H. F. Song, X.-J. Wang, and D. J. Freedman. Circuit mechanisms for the maintenance and manipulation of information in working memory. *Nature neuroscience*, page 1, 2019.

A. Mathis, P. Mamidanna, K. M. Cury, T. Abe, V. N. Murthy, M. W. Mathis, and M. Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. Technical report, Nature Publishing Group, 2018.

M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

L. McIntosh, N. Maheswaranathan, A. Nayebi, S. Ganguli, and S. Baccus. Deep learning models of the retinal response to natural scenes. In *Advances in neural information processing systems*, pages 1369–1377, 2016.

P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. *arXiv preprint arXiv:1804.00222*, 2018.

T. Miconi, J. Clune, and K. O. Stanley. Differentiable plasticity: training plastic neural networks with backpropagation. *arXiv preprint arXiv:1804.02464*, 2018.

G. Mongillo, O. Barak, and M. Tsodyks. Synaptic theory of working memory. *Science*, 319(5869):1543–1546, 2008.

J. M. Murray. Local online learning in recurrent networks with random feedback. *eLife*, 8:e43299, 2019.

T. Nath, A. Mathis, A. C. Chen, A. Patel, M. Bethge, and M. W. Mathis. Using deeplabcut for 3d markerless pose estimation across species and behaviors. *Nature protocols*, 14(7):2152–2176, 2019.

A. Nayebi, D. Bear, J. Kubilius, K. Kar, S. Ganguli, D. Sussillo, J. J. DiCarlo, and D. L. Yamins. Task-driven convolutional recurrent models of the visual system. In *Advances in Neural Information Processing Systems*, pages 5290–5301, 2018.

W. Nicola and C. Clopath. Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):2208, 2017.

S. W. Oh, J. A. Harris, L. Ng, B. Winslow, N. Cain, S. Mihalas, Q. Wang, C. Lau, L. Kuan, A. M. Henry, et al. A mesoscale connectome of the mouse brain. *Nature*, 508(7495):207, 2014.

E. Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

S. R. Olsen, D. S. Bortone, H. Adesnik, and M. Scanziani. Gain control by layer six in cortical circuits of vision. *Nature*, 483(7387):47–52, 2012.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

A. E. Orhan and W. J. Ma. A diverse range of factors affect the nature of neural representations underlying short-term memory. *Nature neuroscience*, page 1, 2019.

C. Pandarinath, D. J. O'Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg, et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, page 1, 2018.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

C. R. Ponce, W. Xiao, P. F. Schade, T. S. Hartmann, G. Kreiman, and M. S. Livingstone. Evolving images for visual neurons using a deep generative network reveals coding principles and neuronal preferences. *Cell*, 177(4):999–1009, 2019.

R. Prenger, M. C.-K. Wu, S. V. David, and J. L. Gallant. Nonlinear v1 responses to natural scenes revealed by neural network analysis. *Neural Networks*, 17(5-6): 663–679, 2004.

R. Rajalingham, E. B. Issa, P. Bashivan, K. Kar, K. Schmidt, and J. J. DiCarlo. Large-scale, high-resolution comparison of the core visual object recognition behavior of humans, monkeys, and state-of-the-art deep artificial neural networks. *Journal of Neuroscience*, 38(33):7255–7269, 2018.

K. Rajan, C. D. Harvey, and D. W. Tank. Recurrent network models of sequence generation and memory. *Neuron*, 90(1):128–142, 2016.

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*, 2(1):79, 1999.

J. H. Reynolds and D. J. Heeger. The normalization model of attention. *Neuron*, 61 (2):168–185, 2009.

B. A. Richards, T. P. Lillicrap, P. Beaudoin, Y. Bengio, R. Bogacz, A. Christensen, C. Clopath, R. P. Costa, A. de Berker, S. Ganguli, et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22:1761–1770, 2019.

M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

M. Rigotti, D. D. Ben Dayan Rubin, X.-J. Wang, and S. Fusi. Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses. *Frontiers in computational neuroscience*, 4:24, 2010.

M. Rigotti, O. Barak, M. R. Warden, X.-J. Wang, N. D. Daw, E. K. Miller, and S. Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585, 2013.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

P. R. Roelfsema and A. Holtmaat. Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience*, 19:166, 2018.

J. D. Roitman and M. N. Shadlen. Response of neurons in the lateral intraparietal area during a combined visual discrimination reaction time task. *J. Neurosci.*, 22: 9475–9489, 2002.

R. Romo, C. D. Brody, A. Hernández, and L. Lemus. Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature*, 399(6735):470–473, 1999.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

F. Rosenblatt. *Principles of Neurodynamics*. New York: Spartan, 1962.

D. B. Rubin, S. D. Van Hooser, and K. D. Miller. The stabilized supralinear network: a unifying circuit motif underlying multi-input integration in sensory cortex. *Neuron*, 85(2):402–417, 2015.

D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn. Dendritic error backpropagation in deep cortical microcircuits. *arXiv preprint arXiv:1801.00062*, 2017.

E. Salinas and P. Thier. Gain modulation: a major computational principle of the central nervous system. *Neuron*, 27(1):15–21, 2000.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

A. M. Saxe, J. L. McClelland, and S. Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116(23):11537–11546, 2019.

W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

H. S. Seung. How the brain keeps the eyes still. *Proc. Natl. Acad. Sci. (USA)*, 93: 13339–13344, 1996.

Y. Shu, A. Hasenstaub, and D. A. McCormick. Turning on and off recurrent balanced cortical activity. *Nature*, 423(6937):288–293, 2003.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

H. F. Song, G. R. Yang, and X.-J. Wang. Training excitatory-inhibitory recurrent neural networks for cognitive tasks: a simple and flexible framework. *PLoS computational biology*, 12(2):e1004792, 2016.

S. Song, K. D. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

S. Strogatz. Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering (studies in nonlinearity). 2001.

D. Sussillo. Neural circuits as computational dynamical systems. *Current opinion in neurobiology*, 25:156–163, 2014.

D. Sussillo and L. F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

D. Sussillo and O. Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.

D. Sussillo, M. M. Churchland, M. T. Kaufman, and K. V. Shenoy. A neural network that finds a naturalistic solution for the production of muscle activity. *Nature neuroscience*, 18(7):1025, 2015.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

A. N. Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943.

D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

C. Van Vreeswijk and H. Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274(5293):1724–1726, 1996.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

J. Wang, D. Narain, E. A. Hosseini, and M. Jazayeri. Flexible timing by temporal scaling of cortical responses. *Nature neuroscience*, 21(1):102, 2018.

X.-J. Wang. Synaptic reverberation underlying mnemonic persistent activity. *Trends in Neurosci.*, 24:455–463, 2001.

X.-J. Wang. Probabilistic decision making by slow reverberation in cortical circuits. *Neuron*, 36(5):955–968, 2002.

X.-J. Wang. Decision making in recurrent neuronal circuits. *Neuron*, 60(2):215–234, 2008.

X.-J. Wang and G. R. Yang. A disinhibitory circuit motif and flexible information routing in the brain. *Curr. Opin. Neurobiol.*, 49:75–83, 2018.

M. Watabe-Uchida, N. Eshel, and N. Uchida. Neural circuitry of reward prediction error. *Annual review of neuroscience*, 40:373–394, 2017.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

A. H. Williams, T. H. Kim, F. Wang, S. Vyas, S. I. Ryu, K. V. Shenoy, M. Schnitzer, T. G. Kolda, and S. Ganguli. Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis. *Neuron*, 98(6):1099–1115, 2018.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

H. R. Wilson and J. D. Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical journal*, 12(1):1–24, 1972.

Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

X. Xie and H. S. Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation*, 15(2):441–454, 2003.

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

Y. Yamane, E. T. Carlson, K. C. Bowman, Z. Wang, and C. E. Connor. A neural code for three-dimensional object shape in macaque inferotemporal cortex. *Nature neuroscience*, 11(11):1352, 2008.

D. L. Yamins and J. J. DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356, 2016.

D. L. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

G. Yang, F. Pan, and W.-B. Gan. Stably maintained dendritic spines are associated with lifelong memories. *Nature*, 462(7275):920–924, 2009.

G. R. Yang, I. Ganichev, X.-J. Wang, J. Shlens, and D. Sussillo. A dataset and architecture for visual reasoning with a working memory. In *European Conference on Computer Vision*, pages 729–745. Springer, 2018.

G. R. Yang, M. R. Joglekar, H. F. Song, W. T. Newsome, and X.-J. Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature neuroscience*, page 1, 2019.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

F. Zenke and S. Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.

D. Zipser and R. A. Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679, 1988.

R. S. Zucker and W. G. Regehr. Short-term synaptic plasticity. *Annual review of physiology*, 64(1):355–405, 2002.