Semantics, Representations and Grammars for Deep Learning

David Balduzzi

Deep learning is currently the subject of intensive study. However, fundamental concepts such as representations are not formally defined – researchers "know them when they see them" – and there is no common language for describing and analyzing algorithms. This essay proposes an abstract framework that identifies the essential features of current practice and may provide a foundation for future developments.

The backbone of almost all deep learning algorithms is backpropagation, which is simply a gradient computation distributed over a neural network. The main ingredients of the framework are thus, unsurprisingly: (i) game theory, to formalize distributed optimization; and (ii) communication protocols, to track the flow of zeroth and first-order information. The framework allows natural definitions of semantics (as the meaning encoded in functions), representations (as functions whose semantics is chosen to optimized a criterion) and grammars (as communication protocols equipped with first-order convergence guarantees).

Much of the essay is spent discussing examples taken from the literature. The ultimate aim is to develop a graphical language for describing the structure of deep learning algorithms that backgrounds the details of the optimization procedure and foregrounds how the components interact. Inspiration is taken from probabilistic graphical models and factor graphs, which capture the essential structural features of multivariate distributions.

Contents

0. INTRODUCTION

Deep learning has achieved remarkable successes in object and voice recognition, machine translation, reinforcement learning and other tasks [1–5]. From a practical standpoint the problem of supervised learning is well-understood and has largely been solved – at least in the regime where both labeled data and computational power are abundant. The workhorse underlying most deep learning algorithms is error backpropagation [6–9], which is simply gradient descent distributed across a neural network via the chain rule.

Gradient descent and its variants are well-understood when applied to convex or nearly convex objectives [10–13]. In particular, they have strong performance guarantees in the stochastic and adversarial settings [14–17]. The reasons for the success of gradient descent in non-convex settings are less clear, although recent work has provided evidence that most local minima are good enough [18,19]; that modern convolutional networks are close enough to convex for many results on rates of convergence apply [20]; and that the rate of convergence of gradient-descent can control generalization performance, even in nonconvex settings [21].

Taking a step back, gradient-based optimization provides a well-established set of computational primitives [22], with theoretical backing in simple cases and empirical backing in others. First-order optimization thus falls in broadly the same category as computing an eigenvector or inverting a matrix: given

sufficient data and computational resources, we have algorithms that reliably find good enough solutions for a wide range of problems.

This essay proposes to abstract out the optimization algorithms used for weight updates and focus on how the components of deep learning algorithms interact. Treating optimization as a computational primitive encourages a shift from low-level algorithm design to higher-level mechanism design: we can shift attention to designing architectures that are guaranteed to learn distributed representations suited to specific objectives. The goal is to introduce a language at a level of abstraction where designers can focus on formal specifications (grammars) that specify how plug-and-play optimization modules combine into larger learning systems.

0.1. What is a representation?

Let us recall how representation learning is commonly understood. Bengio *et al* describe representation learning as "learning transformations of the data that make it easier to extract useful information when building classifiers or other predictors" [23]. More specifically, "a deep learning algorithm is a particular kind of representation learning procedure that discovers multiple levels of representation, with higher-level features representing more abstract aspects of the data" [24]. Finally, LeCun *et al* state that multiple levels of representations are obtained "by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations" [5].

The quotes describe the operation of a successful deep learning algorithm. What is lacking is a characterization of what makes a deep learning algorithm work in the first place. What properties must an algorithm have to learn layered representations? What does it mean for the representation learned by one layer to be useful to another? What, exactly, is a representation?

In practice, almost all deep learning algorithms rely on error backpropagation to "align" the representations learned by different layers of a network. This suggests that the answers to the above questions are tightly bound up in first-order (that is, gradient-based) optimization methods. It is therefore unsurprisingly that the bulk of the paper is concerned with tracking the flow of first-order information. The framework is intended to facilitate the design of more general first-order algorithms than backpropagation.

Semantics. To get started, we need a theory of the meaning or semantics encoded in neural networks. Since there is nothing special about neural networks, the approach taken is inclusive and minimalistic. Definition 1 states that the meaning of *any* function is how it implicitly categorizes inputs by assigning them to outputs. The next step is to characterize those functions whose semantics encode knowledge, and for this we turn to optimization [25].

Representations from optimizations. Nemirovski and Yudin developed the black-box computational model to analyze the computational complexity of first-order optimization methods [26–29]. The black-box model is a more abstract view on optimization than the Turing machine model: it specifies a *communication protocol* that tracks how often an algorithm makes *queries* about the objective. It is useful to refine Nemirovski and Yudin's terminology by distinguishing between black-boxes, which *respond* with zeroth-order information (the value of a function at the query-point), and gray-boxes[1], which respond with zeroth- and first-order information (the gradient or subgradient).

With these preliminaries in hand, Definition 4 proposes that a *representation* is a function that is a *local* solution to an optimization problem. Since we do not restrict to convex problems, finding global solutions is not feasible. Indeed, recent experience shows that global solutions are often not necessary practice [1–5]. The local solution has similar semantics to – that is, it represents – the ideal solution. The ideal solution usually cannot be found: due to computational limitations, since the problem is nonconvex, because we only have access to a finite sample from an unknown distribution, etc.

To see how Definition 4 connects with representation learning as commonly understood, it is necessary to take a detour through distributed optimization and game theory.

0.2. Distributed representations

Game theory provides tools for analyzing distributed optimization problems where a set of players aim to minimizes losses that depend not only on their actions, but also the actions of all other players in the

[1] Gray for gradient.

game [30, 31]. Game theory has traditionally focused on convex losses since they are more theoretically amenable. Here, the only restriction imposed on losses is that they are differentiable almost everywhere.

Allowing nonconvex losses means that error-backpropagation can be reformulated as a game. Interestingly, there is enormous freedom in choosing the players. They can correspond to individual units, layers, entire neural networks, and a variety of other, intermediate choices. An advantage of the game-theoretic formulation is thus that it applies at many different scales.

Nonconvex losses and local optima are essential to developing a *scale-free* formalism. Even when it turns out that particular units or a particular layer of a neural network are solving a convex problem, convexity is destroyed as soon as those units or layers are combined to form larger learning systems. Convexity is not a property that is preserved in general when units are combined into layers or layers into networks. It is therefore convenient to introduce the computational primitive `arglocopt` to denote the output of a first-order optimization procedure, see Definition 4.

A concern about excessive generality. A potential criticism is that the formulation is too broad. Very little can be said about nonconvex optimization in general; introducing games where many players jointly optimize a set of arbitary nonconvex functions only compounds the problem.

Additional structure is required. A successful case study can be found in [20], which presents a detailed game-theoretic analysis of rectifier neural networks. The key to the analysis is that rectifier units are almost convex. The main result is that the rate of convergence of a neural network to a local optimum is controlled by the (waking-)regret of the algorithms applied to compute weight updates in the network.

Whereas [20] relied heavily on specific properties of rectifer nonlinearities, this paper considers a wide-range of deep learning architectures. Nevertheless, it is possible to carve out an interesting subclass of nonconvex games by identifying the composition of simple functions as an essential feature common to deep learning architectures. Compositionality is formalized via distributed communication protocols and grammars.

Grammars for games. Neural networks are constructed by composing a series of elementary operations. The resulting feedforward computation is captured via as a computation graph [32–37]. Backpropagation traverses the graph in reverse and recursively computes the gradient with respect to the parameters at each node.

Section 2 maps the feedforward and feedback computations onto the queries and responses that arise in Nemirovski and Yudin's model of optimization. However, queries and responses are now highly structured. In the query phase, players feed parameters into a computation graph (the Query graph Q) that performs the feedforward sweep. In the response phase, oracles reveal first-order information that is fed into a second computation graph (the Response graph R).

In most cases the Response graph simply implements backpropagation. However, there are examples where it does not. Three are highlighted here, see section 2.3, and especially sections 2.4 and 2.5. Other algorithms where the Response graphs do not simply implement backprop include difference target propagation [38] and feedback alignment [39] (both discussed briefly in section 2.5) and truncated backpropagation through time [40–42], where a choice is made about where to cut backprop short. Examples where the query and response graph differ are of particular interest, since they point towards more general classes of deep learning algorithms.

A *distributed communication protocol* is a game with additional structure: the Query and Response graphs, see Definition 7. The graphs capture the compositional structure of the functions learned by a neural network and the compositional structure of the learning procedure respectively. It is important for our purposes that (i) the feedforward and feedback sweeps correspond to two distinct graphs and (ii) the communication protocol is kept distinct from the optimization procedure. That is, the communication protocol specifies how information flows through the networks without specifying how players make use of it. Players can be treated as plug-and-play rational agents that are provided with carefully constructed and coordinated first-order information to optimize as they see fit [43, 44].

Finally, a *grammar* is a distributed communication protocol equipped with a guarantee that the response graph encodes sufficient information for the players to jointly find a local optimum of an objective function. The paradigmatic example of a grammar is backpropagation. A grammar is a thus a game designed to perform a task. A representation learned by one (p)layer is useful to another if the game is guaranteed to converge on a local solution to an objective – that is, if the players interact though a grammar. It follows that the players build representations that jointly encode knowledge about the task.

Caveats. What follows is provisional. The definitions are a first attempt to capture an interesting, and perhaps useful, perspective on deep learning. The essay contains no new theorems, algorithms or experiments, see [20, 45, 46] for "real work" based on the ideas presented here. The essay is not intended to be comprehensive. Many details are left out and many important aspects are not covered: most notably, probabilistic and Bayesian formulations, and various methods for unsupervised pre-training.

A series of worked examples. In line with its provisional nature, much of the essay is spent applying the framework to worked examples: error backpropagation as a supervised model [8]; variational autoencoders [47] and generative adversarial networks [48] for unsupervised learning; the deviator-actor-critic (DAC) model for deep reinforcement learning [46]; and kickback, a biologically plausible variant of backpropagation [45]. The examples were chosen, in part, to maximize variety and, in part, based on familiarity. The discussions are short; the interested reader is encouraged to consult the original papers to fill in the gaps.

The last two examples are particularly interesting since their Response graphs differ substantially from backpropagation. The DAC model constructs a zeroth-order black-box to estimate gradients rather than querying a first-order gray-box. Kickback prunes backprop's Response graph by replacing most of its gray-boxes with black-boxes and approximating the chain rule with (primarily) local computations.

0.3. Related work

Bottou and Gallinari proposed to decompose neural networks into cooperating modules [49, 50]. Decomposing more general algorithms or models into collections of interacting agents dates back to the shrieking demons that comprised Selfridge's Pandemonium [51] and a long line of related work [52–59]. The focus on components of neural networks as players, or rational agents, in their own right developed here derives from work aimed at modeling biological neurons game-theoretically, see [60–64].

A related approach to semantics based on general value functions can be found in Sutton *et al* [65], see remark 1. Computation graphs as applied to backprop are the basis of the Python library Theano [34–36] and provide the backbone for automatic/algorithmic differentiation [32, 33].

Grammars are a technical term in the theory of formal languages relating to the Chomsky hierarchy [66]. There is no apparent relation between that notion of grammar and the one presented here, aside from both relating to structural rules governing composition. Formal languages and deep learning are sufficiently disparate fields that there is little risk of terminological confusion. Similarly, the notion of semantics introduced here is distinct from semantics in the theory of programming languages.

Although game theory was originally developed to model human interactions [30], it has been pointed out that it may be more directly applicable to interacting populations of algorithms, so-called *machina economicus* [67–72]. This paper goes one step further to propose that games played over first-order communication protocols are a key component of the foundations of deep learning.

A source of inspiration for the essay is Bayesian networks and Markov random fields. Probabilistic graphical models and factor graphs provide simple, powerful ways to encode a multivariate distribution's independencies into a diagram [73–75]. They have greatly facilitated the design and analysis of probabilistic algorithms. However, there is no comparable framework for distributed optimization and deep learning. The essay is intended as a first step in this direction.

1. SEMANTICS AND REPRESENTATIONS

This section defines semantics and representations. In short, the semantics of a function is how it categorizes its inputs; a function is a representation if it is selected to optimize an objective. The connection between the definition of representation below and "representation learning" is clarified in section 2.1.

Possible world semantics was introduced by Lewis to formalize the meaning of sentences in terms of counterfactuals [76]. Let $\mathcal{P}$ be a proposition about the world. Its truth depends on its content and the state of the world. Rather than allowing the state of the world to vary, it is convenient to introduce the set W of all possible worlds.

Let us denote proposition $\mathcal{P}$ applied in world $w \in W$ by $\mathcal{P}(w)$. The meaning of $\mathcal{P}$ is then the mapping $v_{\mathcal{P}} : W \to \{0, 1\}$ which assigns 1 or 0 to each $w \in W$ according to whether or not proposition $\mathcal{P}(w)$ is true. Equivalently, the meaning of the proposition is the ordered pair consisting of: all worlds, and the subset of worlds where it is true:

$$\underbrace{W}_{\text{set of possible worlds}} \quad \supset \quad \underbrace{v_{\mathcal{P}}^{-1}(1)}_{\text{subset of worlds where } \mathcal{P} \text{ is true}}$$

For example, the meaning of $\mathcal{P}_{blue}(that)$ ="*that* is blue" is the subset $v^{-1}_{\mathcal{P}_{blue}}(1)$ of possible worlds where I am pointing at a blue object. The concept of blue is rendered explicit in an exhaustive list of possible examples.

A simple extension of possible world semantics from propositions to arbitrary functions is as follows [77]:

Definition 1 (semantics)**.**
*Given function $f : X \to Y$, the **semantics** or **meaning** of output $y \in Y$ is the ordered pair of sets*

$$\underbrace{X}_{\text{set of possible inputs}} \quad \supset \quad \underbrace{f^{-1}(y)}_{\text{subset causing } f \text{ to output } y}$$

Functions implicitly categorize inputs by assigning outputs to them; the meaning of an output is the category.

Whereas propositions are true or false, the output of a function is neither. However, if two functions both optimize a criterion, then one can refer to how *accurately* one function *represents* the other. Before we can define representations we therefore need to take a quick detour through optimization:

Definition 2 (optimization problem)**.**
*An **optimization problem** is a pair $(\Theta, \mathcal{R})$ consisting in parameter-space $\Theta \subset \mathbb{R}^d$ and objective $\mathcal{R} : \Theta \to \mathbb{R}$ that is differentiable almost everywhere.*
*The **solution** to the global optimization problem is*

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\Theta}{\texttt{argopt}}\, \mathcal{R}(\boldsymbol{\theta}),$$

which is either a maximum or minimum according to the nature of the objective.

The solution may not be unique; it also may not exist unless further restrictions are imposed. Such details are ignored here.

Next recall the black-box optimization framework introduced by Nemirovski and Yudin [26–29].

Definition 3 (communication protocol)**.**
*A **communication protocol** for optimizing an unknown objective $\mathcal{R} : \Theta \to \mathbb{R}$ consists in a User (or Player) and an Oracle. On each round, User presents a **query** $\boldsymbol{\theta} \in \Theta$. Oracle can **respond** in one of two ways, depending on the nature of the protocol:*

- Black-box (zeroth-order) protocol.
  Oracle responds with the value $\mathcal{R}(\boldsymbol{\theta})$.

- Gray-box (first-order) protocol.
  Oracle responds with either the gradient $\nabla\mathcal{R}(\boldsymbol{\theta})$ or with the gradient together with the value.

The protocol specifies how Player and Oracle interact without specifying the algorithm used by Player to decide which points to query. The next section introduces *distributed communication protocols* as a general framework that includes a variety of deep learning architectures as special cases – again without specifying the precise algorithms used to perform weight updates.

Unlike [26, 28] we do not restrict to convex problems. Finding a global optimum is not always feasible, and in practice often unnecessary.

Definition 4 (representation)**.**
Let $\mathcal{F} \subset \{f : X \to Y\}$ be a function space and

$$f : \Theta \to \mathcal{F} : \boldsymbol{\theta} \mapsto f_{\boldsymbol{\theta}}(\bullet)$$

be a map from parameter-space to functions. Further suppose that objective function $\mathcal{R} : \mathcal{F} \to \mathbb{R}$ is given.

*A **representation** is a local solution to the optimization problem*

$$f_{\hat{\boldsymbol{\theta}}} \quad \textit{where} \quad \hat{\boldsymbol{\theta}} \in \underset{\boldsymbol{\theta}\in\Theta}{\texttt{arglocopt}}\, \mathcal{R}(f_{\boldsymbol{\theta}}),$$

corresponding to a local *maximum or minimum according to whether the objective is minimized or maximized.*

Intuitively, the objective quantifies the extent to which functions in $\mathcal{F}$ categorize their inputs similarly. The operation `arglocopt` *applies a first-order method to find a function whose semantics resembles the optimal solution $f_{\boldsymbol{\theta}^*}$ where $\boldsymbol{\theta}^* = \texttt{argopt}_{\boldsymbol{\theta}\in\Theta}\, \mathcal{R}(f_{\boldsymbol{\theta}})$.*

In short, representations are functions with useful semantics, where usefulness is quantifed using a specific objective: the lower the loss or higher the reward associated with a function, the more useful it is. The relation between Definition 4 and representations as commonly understood in the deep learning literature is discussed in section 2.1 below.

Remark 1 (value function semantics)**.**
In related work, Sutton et al *[65] proposed that semantics – i.e. knowledge about the world – can be encoded in general value functions that provide answers to specific questions about expected rewards. Definition 1 is more general than their approach since it associates a semantics to* any *function. However, the function must arise from optimizing an objective for its semantics to accurately represent a phenomenon of interest.*

1.1. Supervised learning

The main example of a representation arises under supervised learning.

Representation 1 (supervised learning)**.**
Let X and Y be an input space and a set of labels and $\boldsymbol{\ell} : Y \times Y \to \mathbb{R}$ be a loss function. Suppose that $\{f_{\boldsymbol{\theta}} : X \to Y \,|\, \theta \in \Theta\}$ is a parametrized family of functions.

- Nature *which samples labeled pairs (x, y) i.i.d. from distribution $\mathbb{P}_{XY}$, singly or in batches.*
- Predictor *chooses parameters $\boldsymbol{\theta} \in \Theta$.*
- Objective *is*

$$\mathcal{R}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{(x,y)\sim\mathbb{P}_{XY}} [\boldsymbol{\ell}(f_{\boldsymbol{\theta}}(x), y)].$$

The query and responses phases can be depicted graphically as

The predictor $f_{\hat{\boldsymbol{\theta}}} = \texttt{arglocmin}_{\boldsymbol{\theta}\in\Theta}\, \mathcal{R}(\boldsymbol{\theta})$ is then a representation of the optimal predictor $f_{\boldsymbol{\theta}^*} = \texttt{argmin}_{\boldsymbol{\theta}\in\Theta}\, \mathcal{R}(\boldsymbol{\theta})$.

A commonly used mapping from parameters to functions is

$$f : \Theta \to \mathcal{F} : \boldsymbol{\theta} \mapsto f_{\boldsymbol{\theta}}(\bullet) := \langle \phi(\bullet), \boldsymbol{\theta} \rangle$$

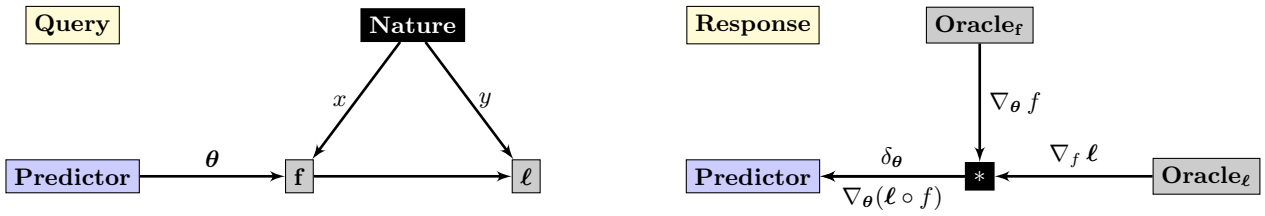
where a feature map $\phi : X \to \mathbb{R}^d$ is fixed.

The setup admits a variety of complications in practice. Firstly, it is typically infeasible even to find a local optimum. Instead, a solution that is within some small $\epsilon > 0$ of the local optimum suffices. Secondly, the distribution $\mathbb{P}_{XY}$ is unknown, so the expectation is replaced by a sum over a finite sample. The quality of the resulting representation has been extensively studied in statistical learning theory [78]. Finally, it is often convenient to modify the objective, for example by incorporating a regularizer. Thus, a more detailed presentation would conclude that

$$\hat{\boldsymbol{\theta}} \approx \underset{\boldsymbol{\theta}\in\Theta}{\texttt{arglocmin}} \sum_{i=1}^{n} \boldsymbol{\ell}(f_{\boldsymbol{\theta}}(x_i), y_i) + \Omega(\boldsymbol{\theta})$$

yields a representation $f_{\hat{\boldsymbol{\theta}}}$ of the solution to $\texttt{argmin}_{\boldsymbol{\theta}}\, \mathbb{E}_{\mathbb{P}_{XY}}[\boldsymbol{\ell}(f_{\boldsymbol{\theta}}(x), y)]$. To keep the discussion and notation simple, we do not consider any of these important details.

It is instructive to unpack the protocol, by observing that the objective $\mathcal{R}$ is a composite function involving $f(\boldsymbol{\theta}, x)$, $\boldsymbol{\ell}(f, y)$ and $\mathbb{E}[\bullet]$:



The notation $\delta_{\boldsymbol{\theta}}$ is borrowed from backpropagation. It is shorthand for the derivative of the objective with respect to parameters $\boldsymbol{\theta}$.

Nature is not a deterministic black-box since it is not queried directly: Nature produces (x, y) pairs stochastically, rather than in response to specific inputs. Our notion of black-box can be extended to stochastic black-boxes, see e.g. [37]. However, once again we prefer to keep the exposition as simple as possible.

1.2. Unsupervised learning

The second example concerns fitting a probabilistic or generative model to data. A natural approach is to find the distribution under which the observed data is most likely:

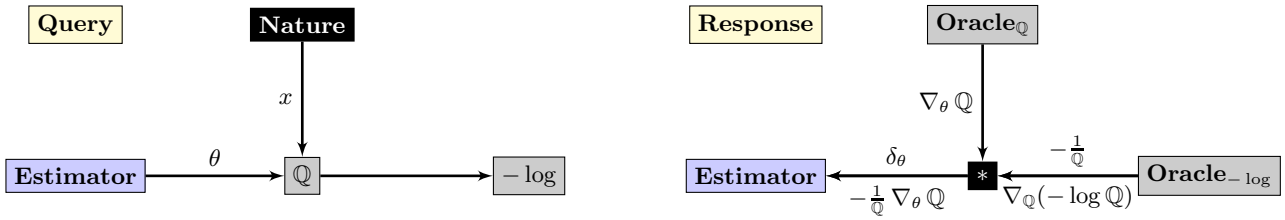
Representation 2 (maximum likelihood estimation)**.**
Let X be a data space.

- Nature *samples points from distribution* $\mathbb{P}_X$.
- Estimator *chooses parameters* $\boldsymbol{\theta} \in \Theta$.
- Operator $\mathbb{Q}(x; \boldsymbol{\theta}) = \mathbb{Q}_{\boldsymbol{\theta}}(x)$ *computes a probability density on* X *that depends on parameter* θ.
- Operator $-\log(\cdot)$ *acts as a loss. The objective is to mimimize*

$$\mathcal{R}(\boldsymbol{\theta}) := -\mathop{\mathbb{E}}_{x\sim\mathbb{P}_X}\big[\log \mathbb{Q}_{\boldsymbol{\theta}}(x)\big].$$

The estimate $\mathbb{Q}(x; \hat{\boldsymbol{\theta}})$, where $\hat{\boldsymbol{\theta}} \in \texttt{arglocmin}_{\boldsymbol{\theta}\in\Theta}\,\mathcal{R}(\boldsymbol{\theta})$, is a representation of the optimal solution, and can also be considered a representation of $\mathbb{P}_X$. The setup extends easily to maximum *a posteriori* estimation.

As for supervised learning, the protocol can be unpacked by observing that the objective has a compositional structure:



1.3. Reinforcement learning

The third example is taken from reinforcement learning [79]. We will return to reinforcement learning in section 2.4, so the example is presented in some detail. In reinforcement learning, an agent interacts with its environment, which is often modeled as a Markov decision process consisting of state space $\mathcal{S} \subset \mathbb{R}^m$, action space $\mathcal{A} \subset \mathbb{R}^d$, initial distribution $\mathbb{P}_1(\mathbf{s})$ on states, stationary transition distribution $\mathbb{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The agent chooses actions based on a *policy*: a function $\boldsymbol{\mu}_\theta : \mathcal{S} \to \mathcal{A}$ from states to actions. The goal is to find the optimal policy.

Actor-critic methods break up the problem into two pieces [80]. The critic estimates the expected value of state-action pairs given the current policy, and the actor attempts to find the optimal policy using the estimates provided by the critic. The critic is typically trained via temporal difference methods [81, 82].

Let $\mathbb{P}_t(\mathbf{s} \to \mathbf{s}', \boldsymbol{\mu})$ denote the distribution on states $\mathbf{s}'$ at time t given policy $\boldsymbol{\mu}$ and initial state $\mathbf{s}$ at $t = 0$ and let $\rho^{\boldsymbol{\mu}}(\mathbf{s}') = \int_{\mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_1(\mathbf{s}) \mathbb{P}_t(\mathbf{s} \to \mathbf{s}', \boldsymbol{\mu}) d\mathbf{s}$. Let $r_t^\gamma = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r(\mathbf{s}_\tau, \mathbf{a}_\tau)$ be the discounted future reward. Define the value of a state-action pair as

$$Q^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a}) = \mathbb{E}[r_1^\gamma | \mathbf{S}_1 = \mathbf{s}, \mathbf{A}_1 = \mathbf{a}; \boldsymbol{\mu}].$$

Unfortunately, the value-function $Q^{\boldsymbol{\mu}}(\mathbf{s}, \mathbf{a})$ cannot be queried. Instead, temporal difference methods take a bootstrapped approach by minimizing the Bellman error:

$$\boldsymbol{\ell}_{BE}(\mathbf{v}) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a}) \sim (\rho^{\boldsymbol{\mu}}, \boldsymbol{\mu})} \left[\Big(r(\mathbf{s}, \mathbf{a}) + \gamma Q^{\mathbf{v}}(\mathbf{s}', \boldsymbol{\mu}(\mathbf{s}')) - Q^{\mathbf{v}}(\mathbf{s}, \mathbf{a}) \Big)^2 \right]$$

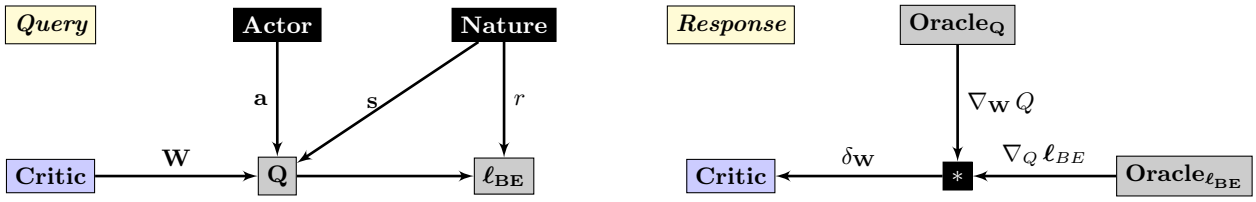
where $\mathbf{s}'$ is the state subsequent to $\mathbf{s}$.

Representation 3 (temporal difference learning)**.**
Critic interacts with black-boxes Actor and Nature.[2]

- Critic *plays parameters* $\mathbf{v}$.
- Operator Q *and* $\boldsymbol{\ell}_{BE}$ *estimates the value function and compute the Bellman error. In practice, it turns out to* clone *the value-estimate periodically and compute a slightly modified Bellman error:*

$$\boldsymbol{\ell}_{BE}(\mathbf{v}) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a}) \sim (\rho^{\boldsymbol{\mu}}, \boldsymbol{\mu})} \left[\Big(r(\mathbf{s}, \mathbf{a}) + \gamma Q^{\tilde{\mathbf{v}}}(\mathbf{s}', \boldsymbol{\mu}(\mathbf{s}')) - Q^{\mathbf{v}}(\mathbf{s}, \mathbf{a}) \Big)^2 \right]$$

where $Q^{\tilde{\mathbf{v}}}$ *is the cloned estimate. Cloning improves the stability of TD-learning [4]. A nice conceptual side-effect of cloning is that TD-learning reduces to gradient descent.*



The estimate is a representation of the true value function.

Remark 2 (on temporal difference learning as first-order method)**.**
Temporal difference learning is not strictly speaking a gradient-based method [82]. The residual gradient method performs gradient descent on the Bellman error, but suffers from double sampling [83]. Projected fixpoint methods minimize the projected *Bellman error via gradient descent and have nice convergence properties [84–86]. An interesting recent proposal is implicit TD learning [87], which is based on implicit gradient descent [88].*

Section 2.4 presents the Deviator-Actor-Critic model which simultaneously learns a value-function estimate and a locally optimal policy.

2. PROTOCOLS AND GRAMMARS

It is often useful to decompose complex problems into simpler subtasks that can handled by specialized modules. Examples include variational autoencoders, generative adversarial networks and actor-critic models. Neural networks are particularly well-adapted to modular designs, since units, layers and even entire networks can easily be combined analogously to bricks of lego [49].

However, not all configurations are viable models. A methodology is required to distinguish good designs from bad. This section provides a basic language to describe how bricks are glued together that may be a useful design tool. The idea is to extend the definitions of optimization problems, protocols and representations from section 1 from single to multi-player optimization problems.

[2] Nature's outputs depend on Actor's actions, so the Query graph should technically have an additional arrow from Actor to Nature.

Definition 5 (game)**.**
*A **distributed optimization problem** or **game** $([N], \Theta, \boldsymbol{\ell})$ is a set $[N] = \{1, \dots N\}$ of players, a parameter space $\Theta = \prod_{i=1}^{N} \Theta_i$, and loss vector $\boldsymbol{\ell} = (\ell_1, \dots, \ell_N) : \Theta \to \mathbb{R}^N$. Player i picks moves from $\Theta_i \subset \mathbb{R}^{d_i}$ and incurs loss determined by $\ell_i : \Theta \to \mathbb{R}$. The goal of each player is to minimize its loss, which depends on the moves of the other players.*

The classic example is a *finite game* [30], where player i has a menu of d_i-actions and chooses a distribution over actions, $\boldsymbol{\theta}_i \in \Theta_i = \triangle_{d_i} = \{(\theta_1, \dots, \theta_{d_i}) : \sum_{j=1}^{d_i} \theta_j = 1 \text{ and } \theta_j \geq 0\}$ on each round. Losses are specified for individual actions, and extended linearly to distributions over actions. A natural generalization of finite games is *convex games* where the parameter spaces are compact convex sets and each loss ℓ_i is a convex function in its i^{th}-argument [89]. It has been shown that players implementing no-regret algorithms are guaranteed to converge to a correlated equilibrium in convex games [89–91].

The notion of game in Definition 5 is too general for our purposes. Additional structure is required.

Definition 6 (computation graph)**.**
*A **computation graph** is a directed acyclic graph with two kinds of nodes:*

- Inputs *are set externally (in practice by Players or Oracles).*
- Operators *produce outputs that are a fixed function of their parents' outputs.*

Computation graphs are a useful tool for calculating derivatives [32–36]. For simplicity, we restrict to deterministic computation graphs. More general stochastic computation graphs are studied in [37].

A *distributed* communication protocol extends the communication protocol in Definition 3 to multi-player games using two computation graphs.

Definition 7 (distributed communication protocol)**.**
*A **distributed communication protocol** is a game where each round has two phases, determined by two computation graphs:*

- Query phase. *Players provide inputs to the Query graph* (Q) *that Operators transform into outputs.*
- Response phase. *Operators in* Q *act as Oracles in the Response graph* (R)*: they input subgradients that are transformed and communicated to the Players.*

The moves chosen by Players depend only on their prior moves and the information communicated to them by the Response graph.

The protocol specifies how Players and Oracles communicate without specifying the optimization algorithms used by the Players. The addition of a Response graph allows more general computations than simply backpropagating the gradients of the Query phase. The additional flexibility allows the design of new algorithms, see sections 2.4 and 2.5 below. It is also sometimes necessary for computational reasons. For example, backpropagation through time on recurrent networks typically runs over a truncated Response graph [40–42].

Suppose that we wish to optimize an objective function $\mathcal{R} : \Theta \to \mathbb{R}$ that depends on all the moves of all the players. Finding a global optimum is clearly not feasible. However, we may be able to construct a protocol such that the players are jointly able to find local optima of the objective. In such cases, we refer to the protocol as a grammar:

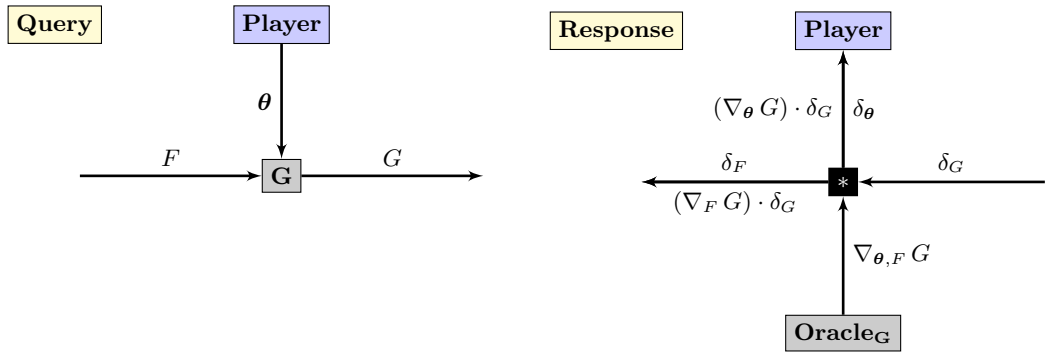
Definition 8 (grammar)**.**
*A **grammar** for objective $\mathcal{R} : \Theta \to \mathbb{R}$ is a distributed communication protocol where the Response graph provides* sufficient *first-order information to find a local optimum of* $(\mathcal{R}, \Theta)$.

The guarantee ensures that the representations constructed by Players in a grammar can be combined into a coherent distributed representation. That is, it ensures that the representations constructed by the Players transform data in a way that is useful for optimizing the shared objective $\mathcal{R}$.

The Players' losses need not be explicitly computed. All that is necessary is that the Response phase communicate the gradient information needed for Players to locally minimize their losses – and that doing so yields a local optimum of the objective.

Basic building blocks: function composition (Q) and the chain rule (R). Functions can be inserted into grammars as lego-like building blocks via function composition during queries and the chain rule during responses. Let $G(\boldsymbol{\theta}, F)$ be a function that takes inputs $\boldsymbol{\theta}$ and F, provided by a Player and by

upstream computations respectively. The output of G is communicated downstream in the Query phase:



The chain rule is implemented in the Response phase as follows. Oracle_G reports the gradient $\nabla_{\boldsymbol{\theta},F} G := (\nabla_{\boldsymbol{\theta}} G, \nabla_F G)$ in the Response phase. Operator "$*$" computes the products $(\nabla_{\boldsymbol{\theta}} G \cdot \delta_G, \nabla_F G \cdot \delta_G)$ via matrix multiplication. The projection of the product onto the first and second components[3] are reported to Player and upstream respectively.

Summary of guarantees. A selection of examples are presented below. Guarantees fall under the following broad categories:

1. *Exact gradients.*
   Under error backpropagation the Response graph implements the chain rule, which guarantees that Players receive the gradients of their loss functions; see section 2.1.
2. *Surrogate objectives.*
   The variational autoencoder uses a surrogate objective: the variational lower bound. Maximizing the surrogate is guaranteed to also maximize the true objective, which is computational intractable; see section 2.2.
3. *Learned objectives.*
   In the case of generative adversarial network and the DAC-model, some of the players learn a loss that is guaranteed to align with the true objective, which is unknown; see sections 2.3 and 2.4.
4. *Estimated gradient.*
   In the DAC-model and kickback, gradient estimates are substituted for the true gradient; see sections 2.4 and 2.5. Guarantees are provided on the estimates.

Remark 3 (fine- and coarse-graining)**.**
There is considerable freedom regarding the choice of players. In the examples below, players are typically chosen to be layers or entire neural networks to keep the diagrams simple. It is worth noting that zooming in, such that players correspond to individual units, has proven to be a useful tool when analyzing neural networks [20, 45, 46].

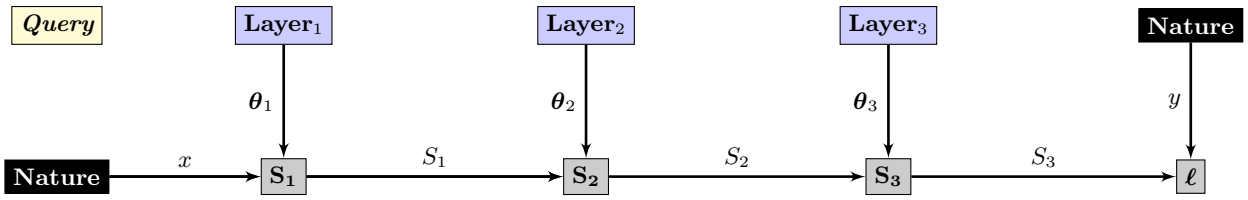
The game-theoretic formulation is thus scale-free and can be coarse- or fine-grained as required. A mathematical language for tracking the structure of hierarchical systems at different scales is provided by operads, see [92] and the references therein, which are the natural setting to study the composition of operators that receive multiple inputs.

2.1. Error backpropagation

The main example of a grammar is a neural network using error backpropagation to perform supervised learning. Layers in the network can be modeled as players in a game. Setting each (p)layer's objective as the network's loss, which it minimizes using gradient ascent, yields backpropagation.

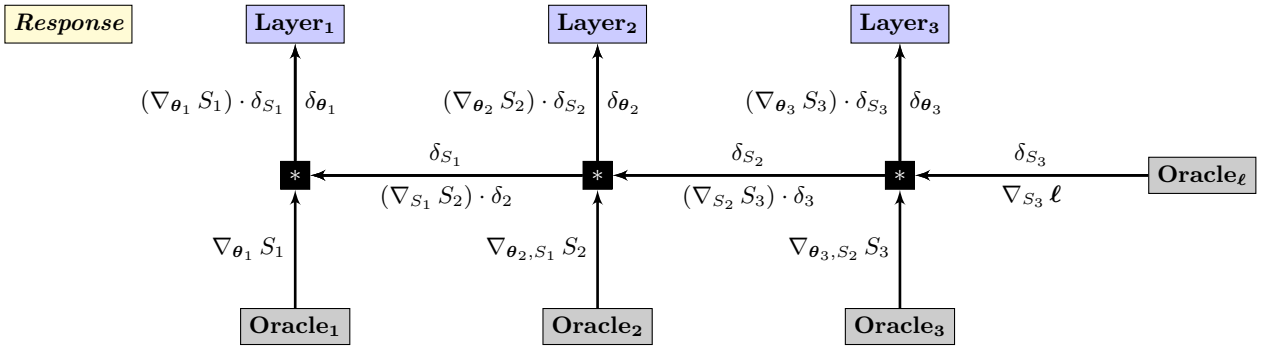
Grammar 1 (backpropagation)**.**
An L-layer neural network can be reformulated as a game played between $L+1$ players, corresponding to Nature *and the* Layers *of the network. The query graph for a 3-layer network is:*

[3] Alternatively, to avoid having "$*$" produce two outputs, the entire vector can be reported in both direction with the irrelevant components ignored.



- Nature *plays samples datapoints* (x, y) *i.i.d. from* $\mathbb{P}_{X\times Y}$ *and acts as the zeroth player.*
- Layer$_i$ *plays weight matrices* $\boldsymbol{\theta}_i$.
- Operators *compute* $S_i(\boldsymbol{\theta}_i, S_{i-1}) := S_i(\boldsymbol{\theta}_i \cdot S_{i-1})$ *for each layer, along with loss* $\boldsymbol{\ell}(S_L, y)$.

The response graph performs error backpropagation:



The protocol can be extended to convolutional networks by replacing the matrix multiplications performed by each operator, $S_i(\boldsymbol{\theta}_i \cdot S_{i-1})$, with convolutions and adding parameterless max-pooling operators [93].

Guarantee. The loss of every (p)layer is

$$\ell(\boldsymbol{\theta}, x, y) = \boldsymbol{\ell}_y \circ S_{\boldsymbol{\theta}_L} \circ \cdots \circ S_{\boldsymbol{\theta}_1}(x) \quad \text{where} \quad \boldsymbol{\ell}_y(\bullet) := \boldsymbol{\ell}(\bullet, y) \quad \text{where} \quad S_{\boldsymbol{\theta}_i}(\bullet) := S_i(\boldsymbol{\theta}_i \cdot \bullet).$$

It follows by the chain rule that R communicates $\nabla_{\boldsymbol{\theta}_i} \ell$ to player i. □

Representation learning. We are now in a position to relate the notion of representation in definition 4 with the standard notion of representation learning in neural networks. In the terminology of section 1, each player learns a representation. The representations learned by the different players form a coherent distributed representation because they jointly optimize a single objective function.

Abstractly, the objective can be written as

$$\mathcal{R}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L) = \mathop{\mathbb{E}}_{(x,y)\sim\mathbb{P}_{XY}} \Big[\boldsymbol{\ell}\big(S(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L, x), y\big)\Big],$$

where $S(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L, x) = S_{\boldsymbol{\theta}_L} \circ \cdots \circ S_{\boldsymbol{\theta}_1}(x)$. The goal is to minimize the composite objective.

If we set $\hat{\boldsymbol{\theta}}_{1:L} \in \texttt{arglocmin}_{(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_L)\in\Theta} \mathcal{R}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L)$ then the function $S_{\hat{\boldsymbol{\theta}}_{1:L}} : X \to Y$ fits the definition of representation above. Moreover, the compositional structure of the network implies that $S_{\hat{\boldsymbol{\theta}}_{1:L}}$ is composed of subrepresentations corresponding to the optimizations performed by the different players in the grammar: each function $S_{\hat{\boldsymbol{\theta}}_j}(\bullet)$ is a local optimum – where $\hat{\boldsymbol{\theta}}_j \in \texttt{arglocmin}_{\boldsymbol{\theta}_j\in\Theta_j} \mathcal{R}(\hat{\boldsymbol{\theta}}_1, \ldots, \boldsymbol{\theta}_j, \ldots, \hat{\boldsymbol{\theta}}_L)$ is optimized to transform its inputs into a form that is useful to network as a whole.

Detailed analysis of convergence rates. Little can be said in general about the rate of converge of the layers in a neural network since the loss is not convex. However, neural networks can be decomposed further by treating the individual units as players. When the units are linear or rectilinear, it turns out that the network is a *circadian game*. The circadian structure provides a way to convert results about the convergence of convex optimization methods into results about the global convergence a rectifier network to a local optimum, see [20].

2.2. Variational autoencoders

The next example extends the unsupervised setting described in section 1.2. Suppose that observations $\{\mathbf{x}^{(i)}\}_{i=1}^N$ are sampled i.i.d. from a two-step stochastic process: a latent value $\mathbf{z}^{(i)}$ is sampled from $\mathbb{P}(\mathbf{z})$, after which $\mathbf{x}^{(i)}$ is sampled from $\mathbb{P}(\mathbf{x}|\mathbf{z}^{(i)})$.

The goal is to (i) find the maximum likelihood estimator for the observed data and (ii) estimate the posterior distribution on $\mathbf{z}$ conditioned on an observation $\mathbf{x}$. A straightforward approach is to maximize the marginal likelihood

$$\boldsymbol{\theta}^* := \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^N \mathbb{Q}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \text{ where } \mathbb{Q}_{\boldsymbol{\theta}}(\mathbf{x}) = \int \mathbb{Q}_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\mathbb{Q}_{\boldsymbol{\theta}}(\mathbf{z})d\mathbf{z}, \tag{1}$$

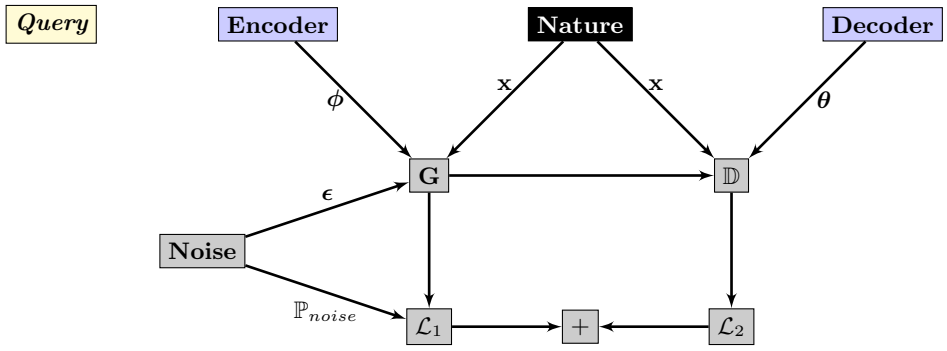
and then compute the posterior

$$\mathbb{Q}_{\boldsymbol{\theta}^*}(\mathbf{z}|\mathbf{x}) = \frac{\mathbb{Q}_{\boldsymbol{\theta}^*}(\mathbf{x}|\mathbf{z})\mathbb{Q}_{\boldsymbol{\theta}^*}(\mathbf{z})}{\mathbb{Q}_{\boldsymbol{\theta}^*}(\mathbf{x})}.$$

However, the integral in Eq. (1) is typically untractable, so a more roundabout tactic is required. The approach proposed in [47] is to construct two neural networks, a decoder $\mathbb{D}_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ that learns a generative model approximating $\mathbb{P}(\mathbf{x}|\mathbf{z})$, and an encoder $\mathbb{E}_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ that learns a recognition model or posterior approximating $\mathbb{P}(\mathbf{z}|\mathbf{x})$.

It turns out to be useful to replace the encoder with a deterministic function, $G_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x})$, and a noise source, $\mathbb{P}_{noise}(\boldsymbol{\epsilon})$ that are compatible. Here, compatible means that sampling $\tilde{\mathbf{z}} \sim \mathbb{E}_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is equivalent to sampling $\boldsymbol{\epsilon} \sim \mathbb{P}_{noise}(\boldsymbol{\epsilon})$ and computing $\tilde{\mathbf{z}} := G_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x})$.

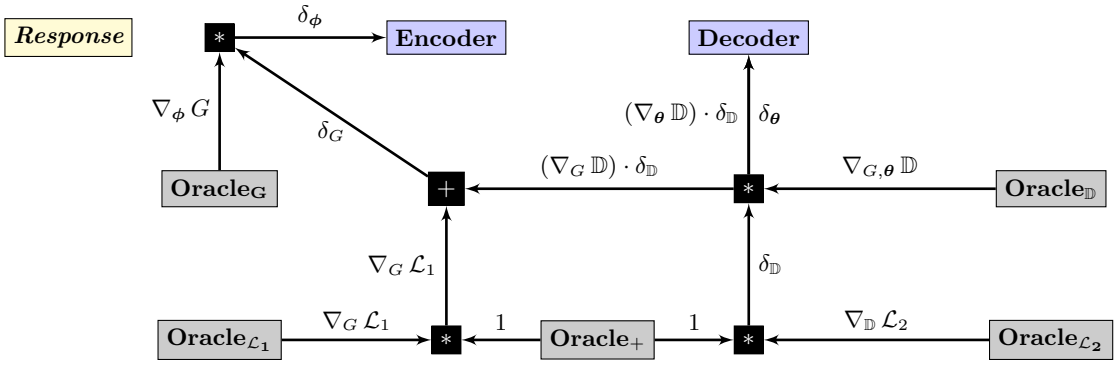
Grammar 2 (variational autoencoder)**.**
A variational autoencoder is a game played between Encoder, Decoder, Noise and Environment. The query graph is



- Environment *plays i.i.d. samples from* $\mathbb{P}(\mathbf{x})$
- Noise *plays i.i.d. samples from* $\mathbb{P}_{noise}(\boldsymbol{\epsilon})$*. It also communicates its density function* $\mathbb{P}_{noise}(\boldsymbol{\epsilon})$*, which is analogous to a gradient – and the reason that* Noise *is gray rather than black-box.*
- Encoder *and* Decoder *play parameters* $\boldsymbol{\phi}$ *and* $\boldsymbol{\theta}$ *respectively.*
- Operator $\mathbf{z} = G_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x})$ *is a neural network that encodes samples into latent variables.*
- Operator $\mathbb{D}_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{x})$ *is a neural network that estimates the probability of* $\mathbf{x}$ *conditioned on* $\mathbf{z}$*.*
- *The remaining operators compute the (negative) variational lower bound*

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \underbrace{\int \mathbb{P}_{noise}(\boldsymbol{\epsilon}) \log \frac{\mathbb{P}_{noise}(\boldsymbol{\epsilon})}{\mathbb{P}_{prior}(G_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x}))}}_{\mathcal{L}_1} + \underbrace{\mathop{\mathbb{E}}_{\boldsymbol{\epsilon} \sim \mathbb{P}_{noise}(\boldsymbol{\epsilon})} \Big[-\log \mathbb{D}_{\boldsymbol{\theta}}\big(G_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x}), \mathbf{x}\big) \Big]}_{\mathcal{L}_2}.$$

The response graph implements backpropagation:



Guarantee. The guarantee has two components:

1. Maximizing the variational lower bound yields (i) a maximum likelihood estimator and (ii) an estimate of the posterior on the latent variable [47].
2. The chain rule ensures that the correct gradients are communicated to Encoder and Decoder.

The first guarantee is that the surrogate objective computed by the query graph yields good solutions. The second guarantee is that the response graph communicates the correct gradients. □

2.3. Generative-Adversarial networks

A recent approach to designing generative models is to construct an adversarial game between Forger and Curator [48]. Forger generates samples; Curator aims to discriminate the samples produced by Forger from those produced by Nature. Forger aims to create samples realistic enough to fool Curator.

If Forger plays parameters $\boldsymbol{\theta}$ and Curator plays $\boldsymbol{\phi}$ then the game is described succinctly via

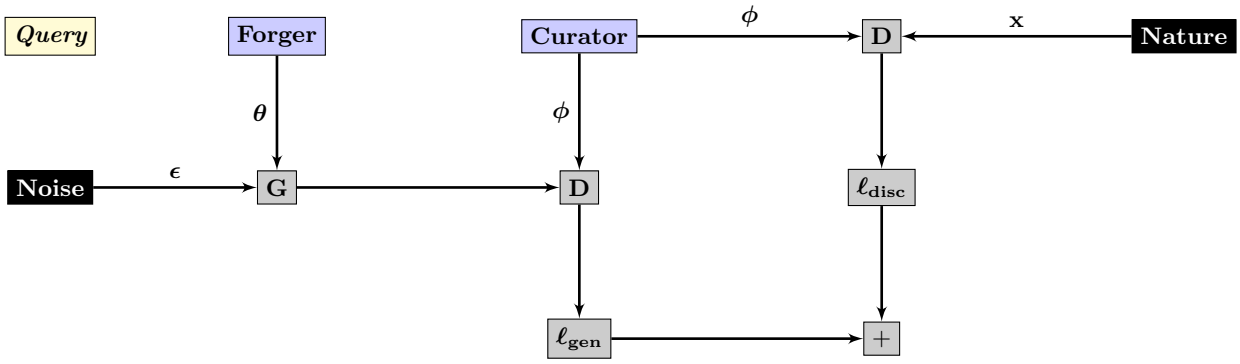
$$\underset{\boldsymbol{\theta}}{\texttt{arglocmin}}\ \underset{\boldsymbol{\phi}}{\texttt{arglocmax}} \left[\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}(\mathbf{x})} \left[\log D_{\boldsymbol{\phi}}(\mathbf{x}) \right] + \mathop{\mathbb{E}}_{\boldsymbol{\epsilon}\sim\mathbb{P}_{noise}(\boldsymbol{\epsilon})} \left[\log(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{\epsilon}))) \right] \right],$$

where $G_{\boldsymbol{\theta}}(\boldsymbol{\epsilon})$ is a neural network that converts noise in samples and $D_{\boldsymbol{\phi}}(\mathbf{x})$ classifies samples as fake or not.

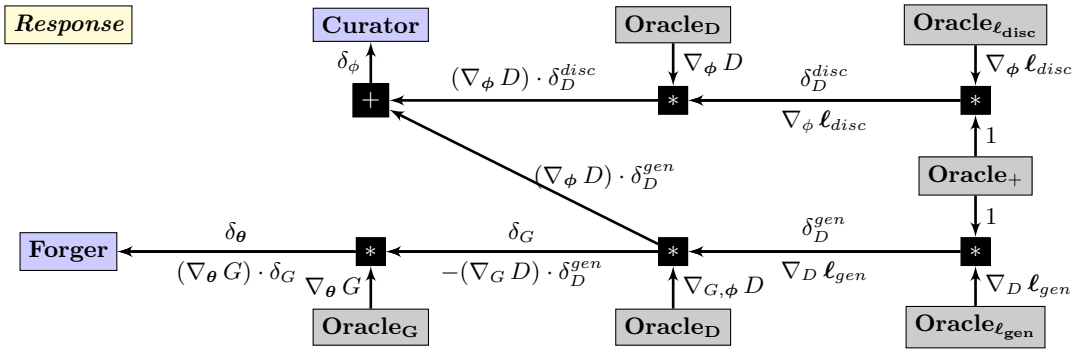
Grammar 3 (generative adversarial networks)**.**
Construct a game played between Forger and Curator, with ancillary players Noise and Environment:

- Environment *samples images i.i.d. from* $\mathbb{P}(\mathbf{x})$.
- Noise *samples i.i.d. from* $\mathbb{P}(\boldsymbol{\epsilon})$.
- Forger *and* Curator *play parameters* $\boldsymbol{\theta}$ *and* $\boldsymbol{\phi}$ *respectively.*
- Operator $G_{\boldsymbol{\theta}}(\boldsymbol{\epsilon})$ *is a neural network that produces fake image* $\tilde{\mathbf{x}} = G_{\boldsymbol{\theta}}(\boldsymbol{\epsilon})$.
- Operator $D_{\boldsymbol{\phi}}(\tilde{\mathbf{x}})$ *is a neural network that estimates the probability that an image is fake.*
- *The remaining operators compute a loss that* Curator *minimizes and* Forger *maximizes*

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}(\mathbf{x})} \left[\log D_{\boldsymbol{\phi}}(\mathbf{x}) \right]}_{\boldsymbol{\ell}_{disc}} + \underbrace{\mathop{\mathbb{E}}_{\boldsymbol{\epsilon}\sim\mathbb{P}(\boldsymbol{\epsilon})} \left[\log \left(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{\epsilon})) \right) \right]}_{\boldsymbol{\ell}_{gen}}$$



Note there are two copies of Operator D in the Query graph. The response graph implements the chain rule, with a tweak that multiplies the gradient communicated to Forger *by* (-1) *to ensure that* Forger *maximizes the loss that* Curator *is minimizing.*



Guarantee. For a fixed Forger that produces images with probability $\mathbb{P}_{\text{Forger}}(\mathbf{x})$, the optimal Curator would assign

$$D^*_{\mathbb{P}_{\text{Forger}},\mathbb{P}_{\text{Nature}}}(\mathbf{x}) = \frac{\mathbb{P}_{\text{Nature}}(\mathbf{x})}{\mathbb{P}_{\text{Nature}}(\mathbf{x}) + \mathbb{P}_{\text{Forger}}(\mathbf{x})} \tag{2}$$

The guarantee has two components:

1. For fixed Forger, the Curator in (2) is the global optimum for $\mathcal{L}$.
2. The chain rule ensures the correct gradients are communicated to Curator and Forger.

It follows that the network converges to a local optimum where Curator represents (2) and Forger represents the "ideal Forger" that would best fool Curator. □

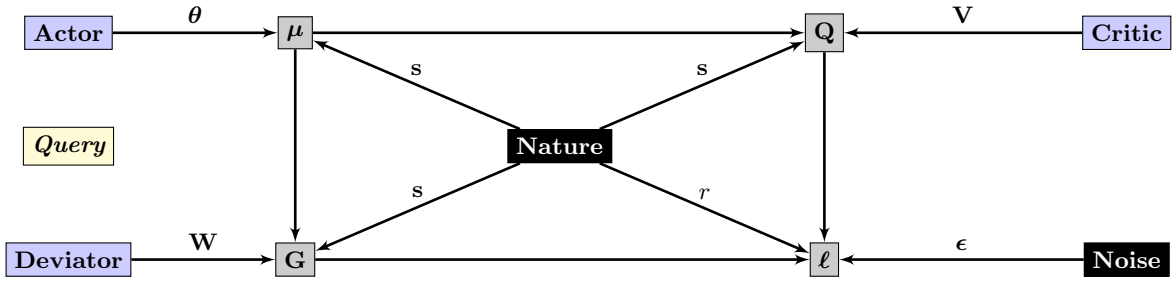
The generative-adversarial network is the first example where the Response graph does not simply backpropagate gradients: the arrow labeled δ_G is computed as $-(\nabla_G D) \cdot \delta_D$, whereas backpropagation would use $(\nabla_G D) \cdot \delta_D$. The minus sign arises due to the adversarial relationship between Forger and Curator – they do not optimize the same objective.

2.4. Deviator-Actor-Critic (DAC) model

As discussed in section 1.3, actor-critic algorithms decompose the reinforcement learning problem into two components: the critic, which learns an approximate value function that predicts the total discounted future reward associated with state-action pairs, and the actor, which searches for a policy that maximizes the value appoximation provided by the critic. When the action-space is continuous, a natural approach is to follow the gradient [94–96]. In [94], it was shown how to compute the policy gradient given the true value function. Furthermore, sufficient conditions were provided for an approximate value function learned by the critic to yield an unbiased estimator of the policy gradient. More recently [96] provided analogous results for deterministic policies.

The next example of a grammar is taken from [46], which builds on the above work by introducing a third algorithm, Deviator, that directly estimates the gradient of the value function estimated by Critic.

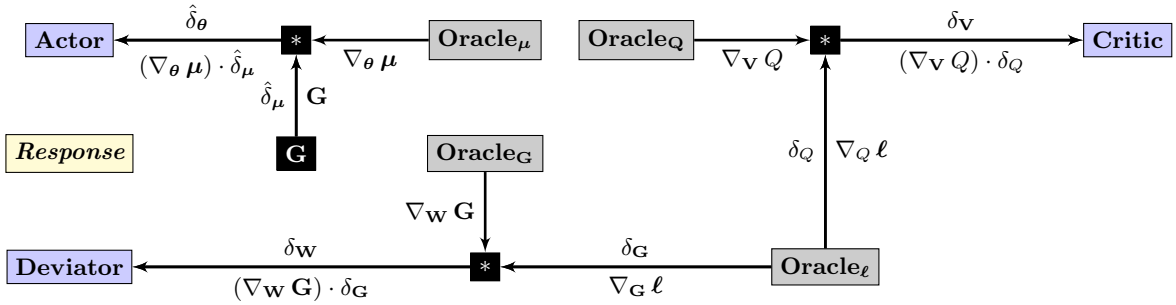
Grammar 4 (DAC model)**.**
Construct a game played by Actor, Critic, Deviator, Noise and Environment:



- Nature *samples states from* $\mathbb{P}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ *and announces rewards* $r(\mathbf{s}_t, \mathbf{a}_t)$ *that are a function of the prior state and action;* Noise *samples* $\boldsymbol{\epsilon} \sim N(0, \sigma^2 \cdot \mathbf{I}_d)$.
- Actor, Critic *and* Deviator *play parameters* $\boldsymbol{\theta}$, $\mathbf{V}$ *and* $\mathbf{W}$ *respectively.*
- Operator $\boldsymbol{\mu}$ *is a neural network that computes actions* $\mathbf{a} = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s})$.
- Operator $Q^{\mathbf{V}}(\mathbf{s}, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}))$ *is a neural network that estimates the value of state-action pairs.*
- Operator $\mathbf{G}^{\mathbf{W}}(\mathbf{s}, \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}))$ *is a neural network that estimates the gradient of the value function.*
- *The remaining* Operator *computes the Bellman gradient error (BGE) which* Critic *and* Deviator *minimize*

$$\ell_{BGE}(r_t, Q, \tilde{Q}, \mathbf{G}, \boldsymbol{\epsilon}) = \Big(r_t + \gamma\tilde{Q} - Q - \big\langle \mathbf{G}, \boldsymbol{\epsilon} \big\rangle\Big)^2.$$

The response graph backpropagates the gradient of ℓ_{BGE} *to* Critic *and* Deviator*, and communicates the output of* Operator $\mathbf{G}$*, which is a* gradient estimate*, to* Actor*:*



Note that instead of backpropagating first-order information in the form of gradient $\nabla_{\boldsymbol{\mu}} \mathbf{G}$, the Response graph instead backpropagates zeroth-order information in the form of *gradient-estimate* $\mathbf{G}$, which is computed by the Query graph during the feedforward sweep. We therefore write $\hat{\delta}_{\boldsymbol{\mu}}$ and $\hat{\delta}_{\boldsymbol{\theta}}$ (instead of $\delta_{\boldsymbol{\mu}}$ and $\delta_{\boldsymbol{\theta}}$) to emphasize that the gradients communicated to Actor are estimates.

As in section 1.3, an arrow from Actor to Nature is omitted from the Query graph for simplicity.

Guarantee. The guarantee has the following components:

1. *Critic* estimates the value function via TD-learning [79] with cloning for improved stability [4].
2. *Deviator* estimates the value gradient via TD-learning and the gradient perturbation trick [46].
3. *Actor* follows the correct gradient by the policy gradient theorem [94, 96].
4. The internal workings of each neural network are guaranteed correct by the chain rule.

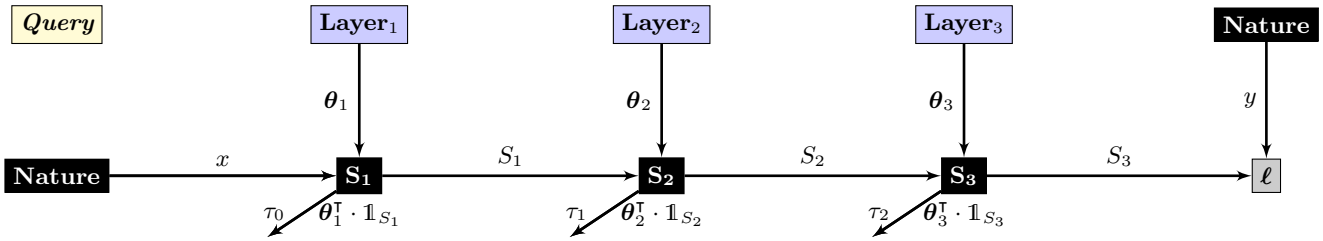
It follows that Critic and Deviator represent the value function and its gradient; and that Actor represents the optimal policy. □

Two appealing features of the algorithm are that (i) Actor is insulated from Critic, and only interacts with Deviator and (ii) Critic and Deviator learn different features adapted to representing the value function and its gradient respectively. Previous work used the derivative of the value-function estimate, which is not guaranteed to have compatible function approximation, and can lead to problems when the value-function is estimated using functions such as rectifiers that are not smooth [97–99].

2.5. Kickback (truncated backpropagation)

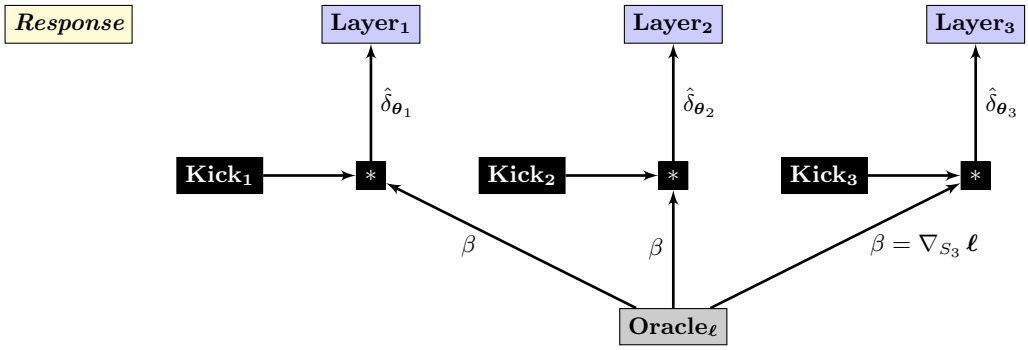
Finally we consider Kickback, a biologically-motivated variant of Backprop with reduced communication requirements [45]. The problem that kickback solves is that backprop requires two distinct kinds of signals to be communicated between units – feedforward and feedback – whereas only one signal type – spikes – are produced by cortical neurons. Kickback computes an estimate of the backpropagated gradient using the signals generated during the feedforward sweep. Kickback also requires the gradient of the loss with respect to the (one-dimensional) output to be broadcast to all units, which is analogous to the role played by diffuse chemical neuromodulators [100–102].

Grammar 5 (kickback)**.**
The query graph is the same as for backpropagation, except that the Operator for each layer produces the additional output $\tau_{i-1} := \boldsymbol{\theta}_{i+1}^\intercal \cdot \mathbb{1}_{S_{i+1}}$*:*

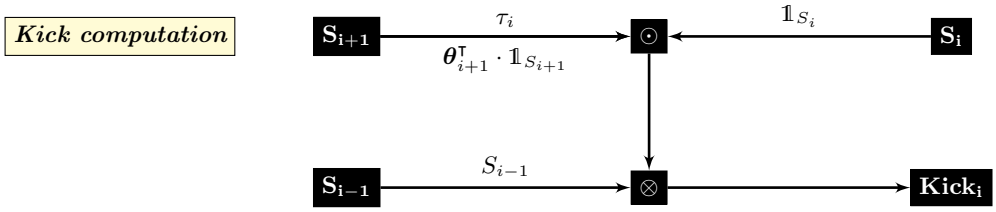


- Nature *samples labeled data* (x, y) *from* $\mathbb{P}_{X\times Y}$*.*
- Layers *by weight matrices* $\boldsymbol{\theta}_i$*. The output of the neural network is required to be one-dimensional*
- Operators *for each layer compute two outputs:* $S_i = \max(0, \boldsymbol{\theta}_i \cdot S_{i-1})$ and $\tau_{i-1} = \boldsymbol{\theta}_i^\intercal \cdot \mathbb{1}_{S_i}$ *where* $\mathbb{1}_a = 1$ *if* $a \geq 0$ *and 0 otherwise.*
- *The task is regression or binary classification with loss given by the mean-squared or logistic error. It follows that the derivative of the loss with respect to the network's output* $\beta = \nabla_{S_3} \boldsymbol{\ell}$ *is a* scalar*.*

The response graph contains a single Oracle that broadcasts the gradient of the loss with respect to the network's output (which is a scalar). Gradient estimates *for each* Layer *are computed using a mixture of Oracle and local zeroth-order information referred to as* Kicks*:*



Kick$_i$ *is computed using locally available zeroth-order information as follows*



where $\odot$ *is coordinatewise multiplication and* $\otimes$ *is the outer product. If* $i = 1$ *then* Nature *is substituted for* S_{i-1}*. If* $i = L$ *then* S_{i+1} *is replaced with the scalar value* 1*.*

The loss functions for the layers are not computed in the query graph. Nevertheless, the gradients communicated to the layers by the response graph are exact with respect to the layers' losses, see [45]. For our purposes it is more convenient to focus on the global objective of the neural network and treat the gradients communicated to the layers as *estimates* of the gradient of the global objective with respect to the layers' weights.

Guarantee. Define unit j to be coherent if $\tau_j > 0$. A network is coherent if all its units are coherent. A sufficient condition for a rectifier to be coherent is that its weights are positive.

The guarantee for Kickback is that, if the network is coherent, then the gradient estimate $\hat{\delta}_{\boldsymbol{\theta}_i}$ computed using the zeroth-order Kicks has the same sign as the backpropagated error $\delta_{\boldsymbol{\theta}_i}$ computed using gradients, see [45] for details. As a result, smalls steps in the direction of the gradient estimates are guaranteed to decrease the network's loss. □

Remark 4 (biological plausibility of kickback)**.**
Kickback uses a single oracle, analogous to a neuromodulatory signal, in contrast to Backprop which requires an oracle per layer. The rest of the oracles are replaced by kicks – zeroth-order information from which gradient-estimates are constructed. Importantly, the kick computation for layer i only requires locally available information produced by its neighboring layers $i - 1$ *and* $i + 1$ *during the feedforward sweep. The feedback signals* τ_i *are analogous to the signals transmitted by NMDA synapses.*

Finally, rectifier units with nonnegative weights (for which coherence holds) can be considered a simple model of excitatory neurons [60, 64, 103].

Two recent alternatives to backprop that also do not rely on backpropagating exact gradients are target propagation [38] and feedback alignment [39]. Target propagation makes do without gradients by implementing autoencoders at each layer. Unfortunately, optimization problems force the authors to introduce a correction term involving *differences* of targets. As a consequence, and in contrast to Kickback, the information required by layers in difference target propagation cannot be computed locally but instead requires recursively backpropagating differences from the output layer.

Feedback alignment solves a different problem: that feedback and forward weights are required to be equal in backprop (and also in kickback). The authors observe that using random feedback weights can suffice. Unfortunately, as for difference target propagation, feedback alignment still requires separate feedforward and recursively backpropagated training signals, so weight updates are not local.

Unfortunately, at a conceptual level kickback, target propagation and feedback alignment all tackle the wrong problem. The cortex performs reinforcement learning: mammals are not provided with labels, and there is no clearly defined output layer from which signals could backpropagate. A biologically-plausible deep learning algorithm should take advantage of the particularities of the reinforcement learning setting.

Acknowledgements. I am grateful to Marcus Frean, JP Lewis and Brian McWilliams for useful comments and discussions.

REFERENCES

[1] Krizhevsky A, Sutskever I, Hinton GE: **Imagenet classification with deep convolutional neural networks**. In *Advances in Neural Information Processing Systems (NIPS)* 2012.

[2] Hinton G, Deng L, Yu D, Dahl GE, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, Kingsbury B: **Deep Neural Networks for Acoustic Modeling in Speech Recognition**. *IEEE Signal Processing Magazine* 2012, **29**:82–97.

[3] Sutskever I, Vinyals O, Le Q: **Sequence to Sequence Learning with Neural Networks**. In *NIPS* 2014.

[4] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D: **Human-level control through deep reinforcement learning**. *Nature* 2015, **518**(7540):529–533.

[5] LeCun Y, Bengio Y, Hinton G: **Deep learning**. *Nature* 2015, **521**:436–444.

[6] Werbos PJ: **Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences**. *PhD thesis*, Harvard 1974.

[7] Rumelhart DE, Hinton GE, Williams RJ: **Learning representations by back-propagating errors**. *Nature* 1986, **323**:533–536.

[8] Rumelhart D, Hinton G, Williams R: *Parallel Distributed Processing. Vol I: Foundations*. MIT Press 1986.

[9] Schmidhuber J: **Deep Learning in Neural Networks: An Overview**. *Neural Networks* 2015, **61**:85–117.

[10] Robbins H, Monro S: **A stochastic approximation method**. *Annals of Math Stat* 1951, **22**(3):400–407.

[11] Nemirovski A, Yudin DB: **On Cezari's convergence of the steepest descent method for approximating saddle point of convex-concave functions**. *Sov. Math. Dokl.* 1978, **19**.

[12] Nemirovski A: **Efficient methods for large-scale convex optimization problems**. *Ekonomika i Matematicheskie Metody* 1979, **15**.

[13] Nemirovski A, Juditsky A, Lan G, Shapiro A: **Robust stochastic approximation approach to stochastic programming**. *SIAM J. Optim.* 2009, **19**(4):1574–1609.

[14] Zinkevich M: **Online Convex Programming and Generalized Infinitesimal Gradient Ascent**. In *ICML* 2003.

[15] Cesa-Bianchi N, Lugosi G: *Prediction, Learning and Games*. Cambridge University Press 2006.

[16] Bottou L, Bousquet O: **The Tradeoffs of Large Scale Learning**. In *NIPS* 2010.

[17] Shalev-Shwartz S: **Online Learning and Online Convex Optimization**. *Foundations and Trends in Machine Learning* 2011, **4**(2):107–194.

[18] Choromanska A, Henaff M, Mathieu M, Arous GB, LeCun Y: **The loss surface of multilayer networks**. In *AISTATS* 2014.

[19] Choromanska A, LeCun Y, Arous GB: **Open Problem: The landscape of the loss surfaces of multilayer networks**. In *COLT* 2015.

[20] Balduzzi D: **Deep Online Convex Optimization by Putting Forecaster to Sleep**. In *arXiv:1509.01851* 2015.

[21] Hardt M, Recht B, Singer Y: **Train faster, generalize better: Stability of stochastic gradient descent**. In *arXiv:1509.01240* 2015.

[22] Gordon GJ: **No-regret algorithms in online convex programs**. In *NIPS* 2006.

[23] Bengio Y, Courville A, Vincent P: **Representation Learning: A Review and New Perspectives**. *IEEE Trans. Pattern Analysis and Mach. Intell.* 2013, **35**(8):1798–1828.

[24] Bengio Y: **Deep Learning of Representations: Looking Forward**. In *Statistical Language and Speech Processing*. Edited by Dediu AH, Martín-Vide C, Mitkov R, Truthe B, Springer 2013.

[25] Sra S, Nowozin S, Wright SJ: *Optimization for Machine Learning*. MIT Press 2012.

[26] Nemirovski AS, Yudin DB: *Problem complexity and method efficiency in optimization*. Wiley-Interscience 1983.

[27] Agarwal A, Bartlett P, Ravikumar P, Wainwright MJ: **Information-theoretic lower bounds on the oracle complexity of convex optimization**. In *NIPS* 2009.

[28] Raginsky M, Rakhlin A: **Information-Based Complexity, Feedback and Dynamics in Convex Programming**. *IEEE Trans. Inf. Theory* 2011, **57**(10):7036–7056.

[29] Arjevani Y, Shalev-Shwartz S, Shamir O: **On Lower and Upper Bounds for Smooth and Strongly Convex Optimization Problems**. In *arXiv:1503.06833* 2015.

[30] von Neumann J, Morgenstern O: *Theory of Games and Economic Behavior*. Princeton University Press 1944.

[31] Nisan N, Roughgarden T, Tardos É, Vazirani V (Eds): *Algorithmic Game Theory*, Cambridge University Press 2007.

[32] Griewank A, Walther A: *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM 2008.

[33] Baydin AG, Pearlmutter BA: **Automatic Differentiation of Algorithms for Machine Learning**. *JMLR: Workshop and Conference Proceedings* 2014, **ICML 2014 AutoML Workshop**:1–7.

[34] Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, Bengio Y: **Theano: A CPU and GPU Math Expression Compiler**. In *Proc. Python for Scientific Comp. Conf. (SciPy)* 2010.

[35] Bastien F, Lamblin P, Pascanu R, Bergstra J, Goodfellow I, Bergeron A, Bouchard N, Bengio Y: **Theano: new features and speed improvements**. In *NIPS Workshop: Deep Learning and Unsupervised Feature Learning* 2012.

[36] van Merriënboer B, Bahdanau D, Dumoulin V, Serdyuk D, Warde-Farley D, Chorowski J, Bengio Y: **Blocks and Fuel: Frameworks for deep learning**. In *arXiv:1506.00619* 2015.

[37] Schulman J, Heess N, Weber T, Abbeel P: **Gradient Estimation Using Stochastic Computation Graphs**. In *NIPS* 2015.

[38] Lee DH, Zhang S, Fischer A, Bengio Y: **Difference Target Propagation**. In *ECML PKDD* 2015.

[39] Lillicrap TP, Cownden D, Tweed DB, Ackerman CJ: **Random feedback weights support learning in deep neural networks**. In *arXiv:1411.0247* 2014.

[40] Elman J: **Finding structure in time**. *Cognitive Science* 1990, **14**(2):179–211.

[41] Williams RJ, Peng J: **An efficient gradient-based algorithm for on-line training of recurrent network trajectories**. *Neural Comp* 1990, **2**(4):490–501.

[42] Williams RJ, Zipser D: **Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity**. In *Backpropagation: Theory, Architectures, and Applications*. Edited by Chauvin Y, Rumelhart D, Lawrence Erlbaum Associates 1995.

[43] Russell S, Norvig P: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition 2009.

[44] Gershman SJ, Horvitz EJ, Tenenbaum J: **Computational rationality: A converging paradigm for intelligence in brains, minds, and machines**. *Science* 2015, **349**(6245):273–278.

[45] Balduzzi D, Vanchinathan H, Buhmann J: **Kickback cuts Backprop's red-tape: Biologically plausible credit assignment in neural networks**. In *AAAI* 2015.

[46] Balduzzi D, Ghifary M: **Compatible Value Gradients for Reinforcement Learning of Continuous Deep Policies**. In *arXiv:1509.03005* 2015.

[47] Kingma DP, Welling M: **Auto-Encoding Variational Bayes**. In *ICLR* 2014.

[48] Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y: **Generative Adversarial Nets**. In *NIPS* 2014.

[49] Bottou L, Gallinari P: **A framework for the cooperation of learning algorithms**. In *NIPS* 1991.

[50] Bottou L: **From machine learning to machine reasoning: An essay**. *Machine Learning* 2014, **94**:133–149.

[51] Selfridge OG: **Pandemonium: a paradigm for learning**. In *Mechanisation of Thought Processes: Proc Symposium Held at the National Physics Laboratory* 1958.

[52] Klopf AH: *The hedonistic neuron: A theory of memory, learning and intelligence*. Washington: Hemi-sphere 1982.

[53] Barto AG: **Learning by statistical cooperation of self-interested neuron-like computing elements**. *Human Neurobiol* 1985, **4**:229–256.

[54] Minsky M: *The society of mind*. Simon and Schuster 1986.

[55] Baum EB: **Toward a Model of Intelligence as an Economy of Agents**. *Machine Learning* 1999, **35**(155-185).

[56] Kwee I, Hutter M, Schmidhuber J: **Market-based reinforcement learning in partially observable worlds**. In *ICANN* 2001.

[57] von Bartheld CS, Wang X, Butowt R: **Anterograde Axonal Transport, Transcytosis, and Recycling of Neurotrophic Factors: The Concept of Trophic Currencies in Neural Networks**. *Molecular Neurobiology* 2001, **24**.

[58] Seung HS: **Learning in Spiking Neural Networks by Reinforcement of Stochastic Synaptic Transmission**. *Neuron* 2003, **40**(1063-1073).

[59] Lewis SN, Harris KD: **The Neural Market Place: I. General Formalism and Linear Theory**. In *bioRxiv* 2014.

[60] Balduzzi D, Besserve M: **Towards a learning-theoretic analysis of spike-timing dependent plasticity**. In *Advances in Neural Information Processing Systems (NIPS)* 2012.

[61] Balduzzi D, Ortega PA, Besserve M: **Metabolic cost as an organizing principle for cooperative learning**. *Advances in Complex Systems* 2013, **16**(2/3).

[62] Balduzzi D, Tononi G: **What can neurons do for their brain? Communicate selectivity with spikes**. *Theory in Biosciences* 2013, **132**:27–39.

[63] Balduzzi D: **Randomized co-training: from cortical neurons to machine learning and back again**. *Randomized Methods for Machine Learning Workshop, Neural Inf Proc Systems (NIPS)* 2013.

[64] Balduzzi D: **Cortical prediction markets**. In *Proc. 13th Int Conf on Autonomous Agents and Multiagent Systems (AAMAS)* 2014.

[65] Sutton R, Modayil J, Delp M, Degris T, Pilarski PM, White A, Precup D: **Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Motor Interaction**. In *Proc. 10th Int. Conf. on Aut Agents and Multiagent Systems (AAMAS)* 2011.

[66] Hopcroft JE, Ullman JD: *Introduction to automata theory, languages, and computation*. Addison-Wesley 1979.

[67] Lay N, Barbu A: **Supervised aggregation of classifiers using artificial prediction markets**. In *27th International Conference on Machine Learning (ICML)* 2010.

[68] Abernethy J, Frongillo R: **A Collaborative Mechanism for Crowdsourcing Prediction Problems**. In *Adv in Neural Information Processing Systems (NIPS)* 2011.

[69] Storkey A: **Machine Learning Markets**. In *14th International Conference on Artificial Intelligence & Statistics (AISTATS)* 2011.

[70] Parkes DC, Wellman MP: **Economic reasoning and artificial intelligence**. *Science* 2015, **349**(6245):267–272.

[71] Frongillo R, Reid M: **Convergence Analysis of Prediction Markets via Randomized Subspace Descent**. In *NIPS* 2015.

[72] Syrgkanis V, Agarwal A, Luo H, Schapire R: **Fast Convergence of Regularized Learning in Games**. In *NIPS* 2015.

[73] Pearl J: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann 1988.

[74] Kschischang F, Frey BJ, Loeliger HA: **Factor graphs and the sum-product algorithm**. *IEEE Trans. Inf. Theory* 2001, **47**(2):498–519.

[75] Wainwright MJ, Jordan MI: **Graphical Models, Exponential Families, and Variational Inference**. *Foundations and Trends in Machine Learning* 2008, **1**(1-2):1–305.

[76] Lewis D: *On the Plurality of Worlds*. Oxford & New York: Basil Blackwell 1986.

[77] Balduzzi D: **Falsification and Future Performance**. In *Algorithmic Probability and Friends: Bayesian Prediction and Artificial Intelligence*, *Volume 7070 of* LNAI. Edited by Dowe D, Springer 2013:65–78.

[78] Vapnik V: *The Nature of Statistical Learning Theory*. Springer 1995.

[79] Sutton RS, Barto AG: *Reinforcement Learning: An Introduction*. MIT Press 1998.

[80] Barto AG, Sutton RS, Anderson CW: **Neuronlike Adapative Elements That Can Solve Difficult Learning Control Problems**. *IEEE Trans. Systems, Man, Cyb* 1983, **13**(5):834–846.

[81] Sutton R: **Learning to predict by the method of temporal differences**. *Machine Learning* 1988, **3**:9–44.

[82] Dann C, Neumann G, Peters J: **Policy Evaluation with Temporal Differences: A Survey and Comparison**. *JMLR* 2014, **15**:809–883.

[83] Baird LC: **Residual algorithms: Reinforcement learning with function approximation**. In *ICML* 1995.

[84] Sutton R, Maei HR, Precup D, Bhatnagar S, Silver D, Szepesvári C, Wiewiora E: **Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation**. In *ICML* 2009.

[85] Sutton R, Szepesvári C, Maei HR: **A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation**. In *Adv in Neural Information Processing Systems (NIPS)* 2009.

[86] Maei HR, Szepesvári C, Bhatnagar S, Sutton R: **Toward Off-Policy Learning Control with Function Approximation**. In *ICML* 2010.

[87] Tamar A, Toulis P, Mannor S, Airoldi EM: **Implicit Temporal Differences**. In *NIPS workshop on large-scale reinforcement learning and Markov decision problems* 2014.

[88] Toulis P, Rennie J, Airoldi EM: **Statistical analysis of stochastic gradient methods for generalized linear models**. In *ICML* 2014.

[89] Stoltz G, Lugosi G: **Learning correlated equilibria in games with compact sets of strategies**. *Games and Economic Behavior* 2007, **59**:187–208.

[90] Foster DP, Vohra RV: **Calibrated Learning and Correlated Equilibrium**. *Games and Economic Behavior* 1997, **21**:40–55.

[91] Blum A, Mansour Y: **From External to Internal Regret**. *JMLR* 2007, **8**:1307–1324.

[92] Spivak DI: **The operad of wiring diagrams: Formalizing a graphical language for databases, recursion, and plug-and-play circuits**. In *arXiv:1305.0297* 2013.

[93] LeCun Y, Bottou L, Bengio Y, Haffner P: **Gradient-based Learning Applied to Document Recognition**. *Proc. of the IEEE* 1998, **86**:2278–2324.

[94] Sutton R, McAllester D, Singh S, Mansour Y: **Policy gradient methods for reinforcement learning with function approximation**. In *NIPS* 1999.

[95] Deisenroth MP, Neumann G, Peters J: **A Survey on Policy Search for Robotics**. *Foundations and Trends in Machine Learning* 2011, **2**(1-2):1–142.

[96] Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M: **Deterministic Policy Gradient Algorithms**. In *ICML* 2014.

[97] Prokhorov DV, Wunsch DC: **Adaptive Critic Designs**. *IEEE Trans. Neur. Net.* 1997, **8**(5):997–1007.

[98] Hafner R, Riedmiller M: **Reinforcement learning in feedback control: Challenges and benchmarks from technical process control**. *Machine Learning* 2011, **84**:137–169.

[99] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D: **Continuous control with deep reinforcement learning**. In *NIPS* 2015.

[100] Schultz W, Dayan P, Montague P: **A neural substrate of prediction and reward**. *Science* 1997, **275**(1593-1599).

[101] Pawlak V, Wickens JR, Kirkwood A, Kerr JND: **Timing is not everything: neuromodulation opens the STDP gate**. *Front. Syn. Neurosci* 2010, **2**(146).

[102] Dayan P: **Twenty-Five Lessons from Computational Neuromodulation**. *Neuron* 2012, **76**:240–256.

[103] Glorot X, Bordes A, Bengio Y: **Deep Sparse Rectifier Neural Networks**. In *Proc. 14th Int Conference on Artificial Intelligence and Statistics (AISTATS)* 2011.