A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27

Yann LeCun
Courant Institute of Mathematical Sciences, New York University `yann@cs.nyu.edu`
Meta - Fundamental AI Research `yann@fb.com`

June 27, 2022

Abstract

How could machines learn as efficiently as humans and animals? How could machines learn to reason and plan? How could machines learn representations of percepts and action plans at multiple levels of abstraction, enabling them to reason, predict, and plan at multiple time horizons? This position paper proposes an architecture and training paradigms with which to construct autonomous intelligent agents. It combines concepts such as configurable predictive world model, behavior driven through intrinsic motivation, and hierarchical joint embedding architectures trained with self-supervised learning.

Keywords: Artificial Intelligence, Machine Common Sense, Cognitive Architecture, Deep Learning, Self-Supervised Learning, Energy-Based Model, World Models, Joint Embedding Architecture, Intrinsic Motivation.

1 Prologue

This document is not a technical nor scholarly paper in the traditional sense, but a position paper expressing my vision for a path towards intelligent machines that learn more like animals and humans, that can reason and plan, and whose behavior is driven by intrinsic objectives, rather than by hard-wired programs, external supervision, or external rewards. Many ideas described in this paper (almost all of them) have been formulated by many authors in various contexts in various form. The present piece does not claim priority for any of them but presents a proposal for how to assemble them into a consistent whole. In particular, the piece pinpoints the challenges ahead. It also lists a number of avenues that are likely or unlikely to succeed.

The text is written with as little jargon as possible, and using as little mathematical prior knowledge as possible, so as to appeal to readers with a wide variety of backgrounds including neuroscience, cognitive science, and philosophy, in addition to machine learning, robotics, and other fields of engineering. I hope that this piece will help contextualize some of the research in AI whose relevance is sometimes difficult to see.

2 Introduction

Animals and humans exhibit learning abilities and understandings of the world that are far beyond the capabilities of current AI and machine learning (ML) systems.

How is it possible for an adolescent to learn to drive a car in about 20 hours of practice and for children to learn language with what amounts to a small exposure. How is it that most humans will know how to act in many situation they have never encountered? By contrast, to be reliable, current ML systems need to be trained with very large numbers of trials so that even the rarest combination of situations will be encountered frequently during training. Still, our best ML systems are still very far from matching human reliability in real-world tasks such as driving, even after being fed with enormous amounts of supervisory data from human experts, after going through millions of reinforcement learning trials in virtual environments, and after engineers have hardwired hundreds of behaviors into them.

Problem

The answer may lie in the ability of humans and many animals to learn *world models*, internal models of how the world works.

There are three main challenges that AI research must address today:

1. How can machines learn to represent the world, learn to predict, and learn to act largely by observation?
   Interactions in the real world are expensive and dangerous, intelligent agents should learn as much as they can about the world without interaction (by observation) so as to minimize the number of expensive and dangerous trials necessary to learn a particular task.

2. How can machine reason and plan in ways that are compatible with gradient-based learning?
   Our best approaches to learning rely on estimating and using the gradient of a loss, which can only be performed with differentiable architectures and is difficult to reconcile with logic-based symbolic reasoning.

3. How can machines learn to represent percepts and action plans in a hierarchical manner, at multiple levels of abstraction, and multiple time scales?
   Humans and many animals are able to conceive multilevel abstractions with which long-term predictions and long-term planning can be performed by decomposing complex actions into sequences of lower-level ones.

The present piece proposes an architecture for intelligent agents with possible solutions to all three challenges.

Approach

The main contributions of this paper are the following:

1. an overall cognitive architecture in which all modules are differentiable and many of them are trainable (Section 3, Figure 2).

2. JEPA and Hierarchical JEPA: a non-generative architecture for predictive world models that learn a hierarchy of representations (Sections 4.4 and 4.6, Figures 12 and 15).

3. a non-contrastive self-supervised learning paradigm that produces representations that are simultaneously informative and predictable (Section 4.5, Figure 13).

Contributions

4. A way to use H-JEPA as the basis of predictive world models for hierarchical planning under uncertainty (section 4.7, Figure 16 and 17).

Impatient readers may prefer to jump directly to the aforementioned sections and figures.

2.1 Learning World Models

Human and non-human animals seem able to learn enormous amounts of background knowledge about how the world works through observation and through an incomprehensibly small amount of interactions in a task-independent, unsupervised way. It can be hypothesized that this accumulated knowledge may constitute the basis for what is often called *common sense*. Common sense can be seen as a collection of *models of the world* that can tell an agent what is likely, what is plausible, and what is impossible. Using such *world models*, animals can learn new skills with very few trials. They can predict the consequences of their actions, they can reason, plan, explore, and imagine new solutions to problems. Importantly, they can also avoid making dangerous mistakes when facing an unknown situation.

The idea that humans, animals, and intelligent systems use world models goes back a long time in psychology (Craik, 1943). The use of *forward models* that predict the next state of the world as a function of the current state and the action being considered has been standard procedure in optimal control since the 1950s (Bryson and Ho, 1969) and bears the name *model-predictive control*. The use of differentiable world models in reinforcement learning has long been neglected but is making a comeback (see for example (Levine, 2021))

A self-driving system for cars may require thousands of trials of reinforcement learning to learn that driving too fast in a turn will result in a bad outcome, and to learn to slow down to avoid skidding. By contrast, humans can draw on their intimate knowledge of intuitive physics to predict such outcomes, and largely avoid fatal courses of action when learning a new skill.

Common sense knowledge does not just allow animals to predict future outcomes, but also to fill in missing information, whether temporally or spatially. It allows them to produce interpretations of percepts that are consistent with common sense. When faced with an ambiguous percept, common sense allows animals to dismiss interpretations that are not consistent with their internal world model, and to pay special attention as it may indicate a dangerous situation and an opportunity for learning a refined world model.

I submit that devising learning paradigms and architectures that would allow machines to learn *world models* in an unsupervised (or self-supervised) fashion, and to use those models to predict, to reason, and to plan is one of the main challenges of AI and ML today. One major technical hurdle is how to devise trainable world models that can deal with complex uncertainty in the predictions.

2.2 Humans and Animals learn Hierarchies of Models

Humans and non-human animals learn basic knowledge about how the world works in the first days, weeks, and months of life. Although enormous quantities of such knowledge are acquired quite quickly, the knowledge seems so basic that we take it for granted. In the first few months of life, we learn that the world is three-dimensional. We learn that every

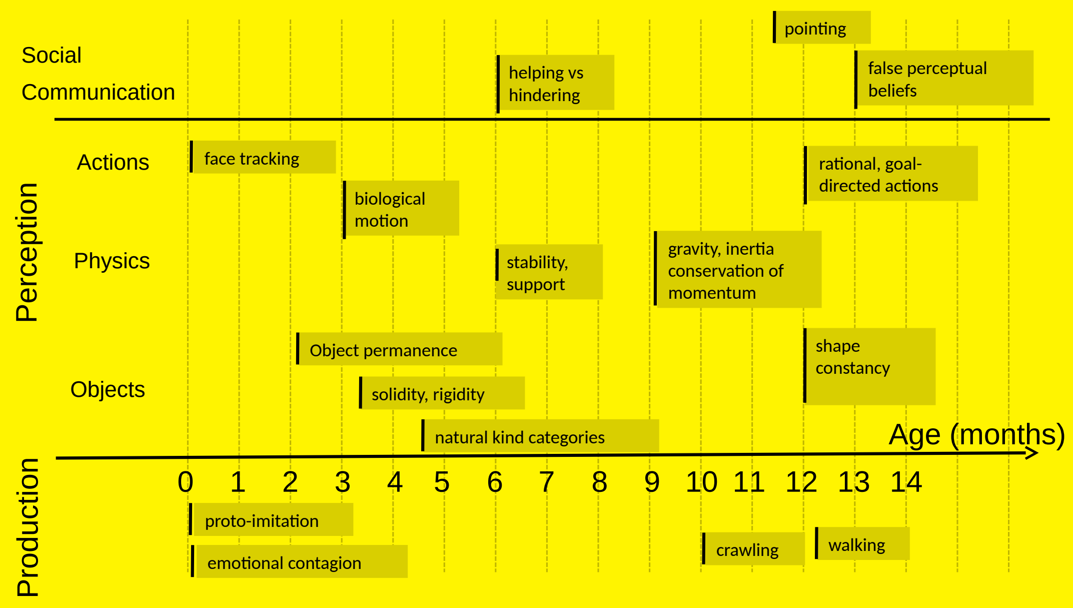


Figure 1: *This chart, (courtesy of Emmanuel Dupoux), indicates at what age infants generally acquire various concepts about how the world works. It is consistent with the idea that abstract concepts, such as the fact that objects are subject to gravity and inertia, are acquired on top of less abstract concepts, like object permanence and the assignment of objects to broad categories. Much of this knowledge is acquired mostly by observation, with very little direct intervention, particularly in the first few weeks and months.*

source of light, sound, and touch in the world has a distance from us. The fact that every point in a visual percept has a distance is the best way to explain how our view of the world changes from our left eye to our right eye, or when our head is being moved. Parallax motion makes depth obvious, which in turn makes the notion of object obvious, as well as the fact that objects can occlude more distant ones. Once the existence of objects is established, they can be automatically assigned to broad categories as a function of their appearance or behavior. On top of the notion of object comes the knowledge that objects do not spontaneously appear, disappear, change shape, or teleport: they move smoothly and can only be in one place at any one time. Once such concepts are acquired, it becomes easy to learn that some objects are static, some have predictable trajectories (inanimate objects), some behave in somewhat unpredictable ways (collective phenomena like water, sand, tree leaves in the wind, etc), and some seem to obey different rules (animate objects). Notions of intuitive physics such as stability, gravity, inertia, and others can emerge on top of that. The effect of animate objects on the world (including the effects of the subject's own actions) can be used to deduce cause-and-effect relationships, on top of which linguistic and social knowledge can be acquired.

Figure 1, courtesy of Emmanuel Dupoux, shows at what age infants seem to acquire basic concepts such as object permanence, basic categories, intuitive physics, etc. Concepts at higher levels of abstraction seem to develop on top of lower-level ones.

Equipped with this knowledge of the world, combined with simple hard-wired behaviors and intrinsic motivations/objectives, animals can quickly learn new tasks, predict the consequences of their actions and plan ahead, foreseeing successful courses of actions and avoiding dangerous situations.

But can a human or animal brain contain all the world models that are necessary for survival? One hypothesis in this paper is that animals and humans have only one world model engine somewhere in their prefrontal cortex. That world model engine is dynamically configurable for the task at hand. With a single, configurable world model engine, rather than a separate model for every situation, knowledge about how the world works may be shared across tasks. This may enable reasoning by analogy, by applying the model configured for one situation to another situation.

To make things concrete, I will directly dive into a description of the proposed model.

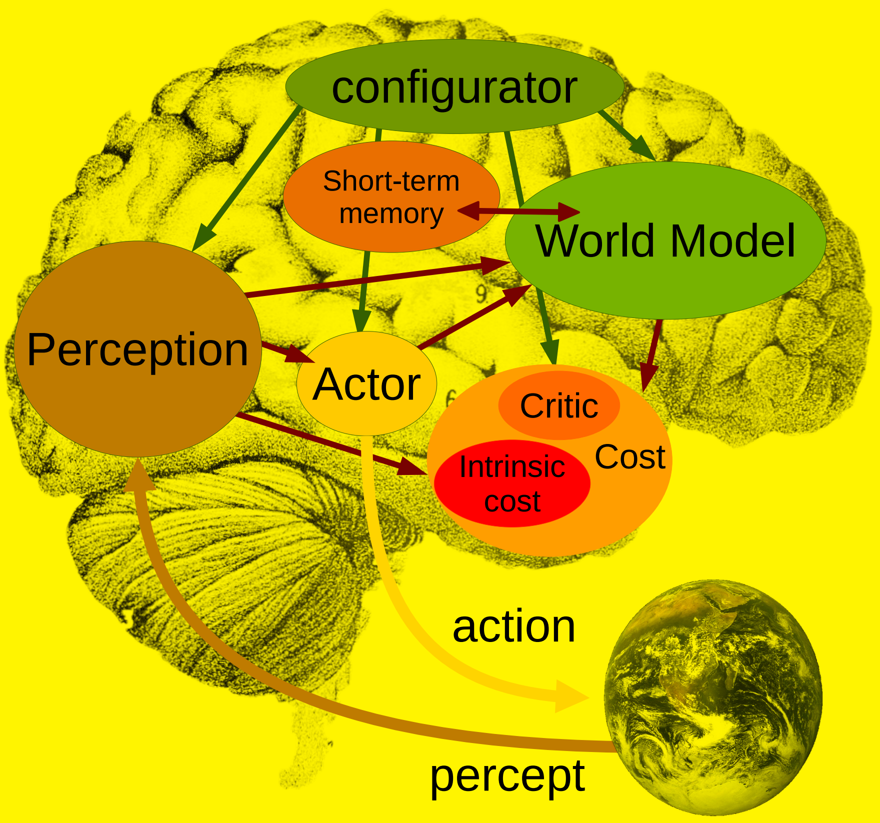


Figure 2: *A system architecture for autonomous intelligence. All modules in this model are assumed to be "differentiable", in that a module feeding into another one (through an arrow connecting them) can get gradient estimates of the cost's scalar output with respect to its own output.*
The configurator module *takes inputs (not represented for clarity) from all other modules and configures them to perform the task at hand.*
The perception module *estimates the current state of the world.*
The world model module *predicts possible future world states as a function of imagined actions sequences proposed by the actor.*
The cost module *computes a single scalar output called "energy" that measures the level of discomfort of the agent. It is composed of two sub-modules, the intrinsic cost, which is immutable (not trainable) and computes the immediate energy of the current state (pain, pleasure, hunger, etc), and the critic, a trainable module that predicts future values of the intrinsic cost.*
The short-term memory module *keeps track of the current and predicted world states and associated intrinsic costs.*
The actor module *computes proposals for action sequences. The world model and the critic compute the possible resulting outcomes. The actor can find an optimal action sequence that minimizes the estimated future cost, and output the first action in the optimal sequence.*
See Section 3 for details.

3 A Model Architecture for Autonomous Intelligence

The proposed architecture for autonomous intelligent agents is depicted in Figure 2.

It is composed of a number of modules whose functions are described below. Some of the modules are configurable on the fly, i.e. their precise function is determined by the configurator module. The role of the configurator is executive control: given a task to be executed, it pre-configures the perception, the world model, the cost and the actor for the task at hand. The configurator modulates the parameters of the modules it feeds into.

The configurator module takes input from all other modules and configures them for the task at hand by modulating their parameters and their attention circuits. In particular, the configurator may prime the perception, world model, and cost modules to fulfill a particular goal.

The perception module receives signals from sensors and estimates the current state of the world. For a given task, only a small subset of the perceived state of the world is relevant and useful. The perception module may represent the state of the world in a hierarchical fashion, with multiple levels of abstraction. The configurator primes the perception system to extract the relevant information from the percept for the task at hand.

The world model module constitutes the most complex piece of the architecture. Its role is twofold: (1) estimate missing information about the state of the world not provided by perception, (2) predict plausible future states of the world. The world model may predict natural evolutions of the world, or may predict future world states resulting from a sequence of actions proposed by the actor module. The world model may predict multiple plausible world states, parameterized by latent variables that represent the uncertainty about the world state. The world model is a kind of "simulator" of the relevant aspects of world. What aspects of the world state is relevant depends on the task at hand. The configurator configures the world model to handle the situation at hand. The predictions are performed within an abstract representation space that contains information relevant to the task at hand. Ideally, the world model would manipulate representations of the world state at multiple levels of abstraction, allowing it to predict over multiple time scales.

A key issue is that the world model must be able to represent multiple possible predictions of the world state. The natural world is not completely predictable. This is particularly true if it contains other intelligent agents that are potentially adversarial. But it is often true even when the world only contains inanimate objects whose behavior is chaotic, or whose state is not fully observable.

There are two essential questions to answer when building the proposed architectures: (1) How to allow the world model to make multiple plausible prediction and represent uncertainty in the predictions, and (2) how to train the world model.

The cost module measures the level of "discomfort" of the agent, in the form of a scalar quantity called the *energy*. The energy is the sum of two energy terms computed by two sub-modules: the Intrinsic Cost module and the Trainable Critic module. The overall objective of the agent is to take actions so as to remain in states that minimize the average energy.

The *Intrinsic Cost module* is hard-wired (immutable, non trainable) and computes a single scalar, the *intrinsic energy* that measures the instantaneous "discomfort" of the agent

– think pain (high intrinsic energy), pleasure (low or negative intrinsic energy), hunger, etc. The input to the module is the current state of the world, produced by the perception module, or potential future states predicted by the world model. *The ultimate goal of the agent is minimize the intrinsic cost over the long run.* This is where basic behavioral drives and intrinsic motivations reside. The design of the intrinsic cost module determines the nature of the agent's behavior. Basic drives can be hard-wired in this module. This may include feeling "good" (low energy) when standing up to motivate a legged robot to walk, when influencing the state of the world to motivate agency, when interacting with humans to motivate social behavior, when perceiving joy in nearby humans to motivate empathy, when having a full energy supplies (hunger/satiety), when experiencing a new situation to motivate curiosity and exploration, when fulfilling a particular program, etc. Conversely, the energy would be high when facing a painful situation or an easily-recognizable dangerous situation (proximity to extreme heat, fire, etc), or when wielding dangerous tools. The intrinsic cost module may be modulated by the configurator, to drive different behavior at different times.

The *Trainable Critic module* predicts an estimate of future intrinsic energies. Like the intrinsic cost, its input is either the current state of the world or possible states predicted by the world model. For training, the critic retrieves past states and subsequent intrinsic costs stored in the associative memory module, and trains itself to predict the latter from the former. The function of the critic module can be dynamically configured by the configurator to direct the system towards a particular sub-goal, as part of a bigger task.

Because both sub-modules of the cost module are differentiable, the gradient of the energy can be back-propagated through the other modules, particularly the world model, the actor and the perception, for planning, reasoning, and learning.

The short-term memory module stores relevant information about the past, current, and future states of the world, as well as the corresponding value of the intrinsic cost. The world model accesses and updates the short-term memory while temporally predicting future (or past) states of the world, and while spatially completing missing information or correcting inconsistent information about the current world state. The world model can send queries to the short-term memory and receive retrieved values, or store new values of states. The critic module can be trained by retrieving past states and associated intrinsic costs from the memory. The architecture may be similar to that of Key-Value Memory Networks (Miller et al., 2016) This module can be seen as playing some of same roles as the hippocampus in vertebrates.

The actor module computes proposals for sequences of actions and outputs actions to the effectors. The actor proposes a sequence of actions to the world model. The world model predicts future world state sequences from the action sequence, and feeds it to the cost. Given a goal defined by the cost (as configured by the configurator), the cost computes the estimated future energy associated with the proposed action sequence. Since the actor has access to the gradient of the estimated cost with respect to the proposed action sequence, it can compute an optimal action sequence that minimizes the estimated cost using gradient-based methods. If the action space is discrete, dynamic programming may be used to find an optimal action sequence. Once the optimization is completed, the actor outputs the first action (or a short sequence of actions) to the effectors. This process is akin to *model-predictive control* in optimal control (Bryson and Ho, 1969).

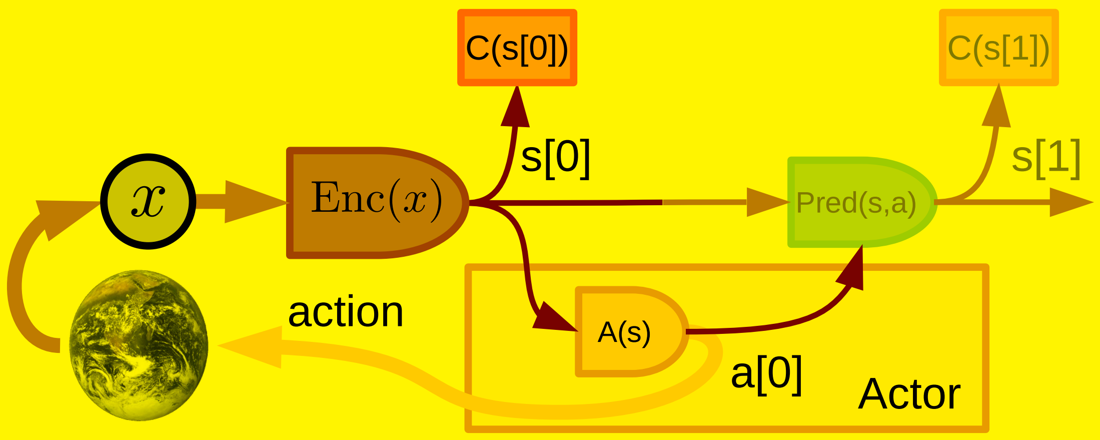


Figure 3: *Mode-1 perception-action episode. The perception module estimates the state of the world* $s[0] = \mathrm{Enc}(x)$. *The actor directly computes an action, or a short sequence of actions, through a policy module* $a[0] = A(s[0])$.
This reactive process does not make use of the world model nor of the cost. The cost module computes the energy of the initial state $f[0] = \mathrm{C}(s[0])$ *and stores the pairs* $(s[0], f[0])$ *in the short-term memory. Optionally, it may also predict the next state using the world model* $s[1] = \mathrm{Pred}(s[0], a[0])$, *and the associated energy* $f[0] = \mathrm{C}(s[0])$ *so that the world model can be adjusted once the next observation resulting from the action taken becomes available.*

The actor may comprise two components: (1) a policy module that directly produces an action from the world state estimate produced by the perception and retrieved from the short-term memory, and (2) the action optimizer, as described above, for model-predictive control. The first mode is similar to Daniel Kahneman's "System 1", while the second mode is similar to "System 2" (Kahneman, 2011)

In the following, we will use specific symbols to represent various components in architectural diagrams. An brief explanation is given in Appendix 8.3.3.

3.1 Typical Perception-Action Loops

There are two possible modes that the model can employ for a perception-action episode. The first one involves no complex reasoning, and produces an action directly from the output of the perception and a possible short-term memory access. We will call it "Mode-1", by analogy with Kahneman's "System 1". The second mode involves reasoning and planning through the world model and the cost. It is akin to model-predictive control (MPC), a classical planning and reasoning paradigm in optimal control and robotics. We will call it "Mode-2" by analogy to Kahneman's "System 2". We use the term "reasoning" in a broad sense here to mean constraint satisfaction (or energy minimization). Many types of reasoning can be viewed as forms of energy minimization.

3.1.1 Mode-1: Reactive behavior

A perception-action episode for Mode-1 is depicted in Figure 3.

The perception module, through an encoder module, extracts a representation of the state of the world $s[0] = \mathrm{Enc}(x)$ containing relevant information for the task at hand. A policy module, a component of the actor, produces an action as a function of the state $a[0] = A(s[0])$. The resulting action is sent to the effectors.

The function of the policy module is modulated by the configurator, which configures it for the task at hand.

The policy module implements a purely reactive policy that does not involve deliberate planning nor prediction through the world model. Yet, its structure can be quite sophisticated. For example, in addition to the state $s[0]$, the policy module may access the short-term memory to acquire a more complete information about previous world states. It may use the short-term memory for the associative retrieval of an action given the current state.

While the cost module is differentiable, its output $f[0] = C(s[0])$ is indirectly influenced by previous actions through the external world. Since the world is not differentiable, one cannot back-propagate gradients from the cost through the chain cost $\leftarrow$ perception $\leftarrow$ world $\leftarrow$ action. In this mode, gradients of the cost $f[0]$ with respect to actions can only be estimated by polling the world with multiple perturbed actions, but that is slow and potentially dangerous. This process would correspond to classical policy gradient methods in reinforcement learning.

During Mode-1, the system can optionally adjust the world model. It runs the world model for one step, predicting the next state $s[1]$, then it waits for the next percept resulting from the action taken, and uses the observed world state as a target for the predictor.

With the use of a world model, the agent can imagine courses of actions and predict their effect and outcome, lessening the need to perform an expensive and dangerous search for good actions and policies by trying multiple actions in the external world and measuring the result.

3.1.2 Mode-2: reasoning and planning using the world model

A typical perception-action episode for Mode 2 is depicted in Figure 4.

1. **perception**: the perception system extract a representation of the current state of the world $s[0] = P(x)$. The cost module computes and stores the immediate cost associated with that state.

2. **action proposal**: the actor proposes an initial sequence of actions to be fed to the world model for evaluation $(a[0], \ldots, a[t], \ldots, a[T])$.

3. **simulation**: the world model predicts one or several likely sequence of world state representations resulting from the proposed action sequence $(s[1], \ldots, s[t], \ldots, s[T])$.

4. **evaluation**: the cost module estimates a total cost from the predicted state sequence, generally as a sum over time steps $F(x) = \sum_{t=1}^{T} C(s[t])$

5. **planning**: the actor proposes a new action sequence with lower cost. This can be done through a gradient-based procedure in which gradients of the cost are back-propagated through the compute graph to the action variables. The resulting minimum-cost action sequence is denoted $(\check{a}[0], \ldots, \check{a}[T])$. Full optimization may require iterating steps 2-5.

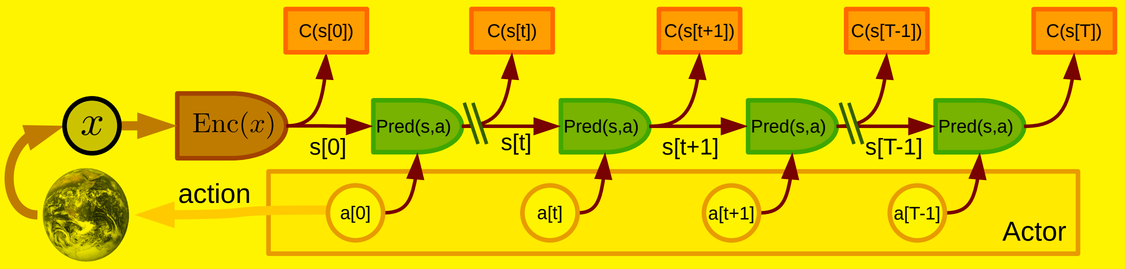


Figure 4: *Mode-2 perception-action episode. The perception module estimates the state of the world $s[0]$. The actor proposes a sequence of actions $a[0], a[1], \ldots, a[t], a[t+1], \ldots, a[T]$. The world model recursively predicts an estimate of the world state sequence using $s[t+1] = \text{Pred}(s[t], a[t])$. The cost $C(s[t])$ computes an energy for each predicted state in the sequence, the total energy being the sum of them. Through an optimization or search procedure, the actor infers a sequence of actions that minimizes the total energy. It then sends the first action in the sequence (or the first few actions) to the effectors. This is, in effect, an instance of classical model-predictive control with receding-horizon planning. Since the cost and the model are differentiable, gradient-based methods can be used to search for optimal action sequences as in classical optimal control. Since the total energy is additive over time, dynamic programming can also be used, particularly when the action space is small and discretized. Pairs of states (computed by the encoder or predicted by the predictor) and corresponding energies from the intrinsic cost and the trainable critic are stored in the short-term memory for subsequent training of the critic.*

6. **acting**: after converging on a low-cost action sequence, the actor sends the first action (or first few actions) in the low-cost sequence to the effectors. The entire process is repeated for the next perception-action episode.

7. **memory**: after every action, the states and associated costs from the intrinsic cost and the critic are stored in the short-term memory. These pairs can be used later to train or adapt the critic.

This procedure is essentially what is known as Model-Predictive Control (MPC) with receding horizon in the optimal control literature. The difference with classical optimal control is that the world model and the cost function are learned.

In principle, any form of optimization strategy can be used, for step 5. While gradient-based optimization methods can be efficient when the world model and cost are well-behaved, situations in which the action-cost mapping has discontinuities may require to use other optimization strategies, particularly if the state and/or action spaces can be discretized. These strategies include dynamic programming, combinatorial optimization, simulate annealing and other gradient-free methods, heuristic search techniques (e.g. tree search with pruning), etc.

To simplify, the process was described in the deterministic case, i.e. when there is no need to handle the possibility of multiple predictions for $s[t+1]$ resulting from a given initial state $s[t]$ and action $a[t]$. In real situations, the world is likely to be somewhat unpredictable. Multiple states may result from a single initial state and action due to the fact that the world is intrinsically stochastic (aleatoric uncertainty), or that the state representation $s[t]$ contains incomplete information about the true world state (epistemic uncertainty), or that the world model's prediction accuracy is imperfect due to limited training data, representational power, or computational constraints.

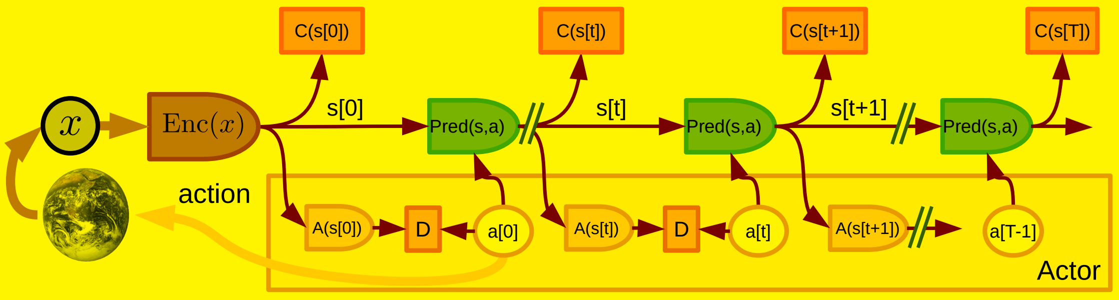


Figure 5: *Training a reactive policy module from the result of Mode-2 reasoning. Using Mode-2 is onerous, because it mobilizes all the resources of the agent for the task at hand. It involves running the world model for multiple time steps repeatedly. This diagram depicts how to train a policy module $A(s[t])$ to approximate the action that results from Mode-2 optimization. The system first operates in Mode-2 and produces an optimal sequence of actions $(\check{a}[0], \dots, \check{a}[T])$. Then the parameters of the policy module are adjusted to minimize a divergence $D(\check{a}[t]), A(s[t]))$ between the optimal action and the output of the policy module. This results in a policy module that performs* amortized inference, *and produces an approximation for a good action sequence. The policy module can then be used to produce actions reactively in Mode-1, or to initialize the action sequence prior to Mode-2 inference and thereby accelerate the optimization.*

3.1.3 From Mode-2 to Mode-1: Learning New Skills

Using Mode-2 is onerous. The agent only possesses one world model "engine". It is configurable by the configurator for the task at hand, but it can only be used for a single task at a time. Hence, similarly to humans, *the agent can only focus on one complex task at a time.*

Mode-1 is considerably less onerous, since it only requires a single pass through a policy module. The agent may possess multiple policy modules working simultaneously, each specialized for a particular set of tasks.

The process described in Figure 5 shows how a policy module $A(s[t])$ can be trained to produce approximations of the optimal actions resulting from Mode-2 reasoning. The system is run on Mode-2, producing an optimal action sequence $(\check{a}[0], \dots, \check{a}[t], \dots, \check{a}[T])$. Then, the parameters of the policy module $A(s[t])$ are updated to minimize a divergence measure between its output and the optimal action at that time $D(\check{a}[t], A(s[t]))$. Once properly trained, the policy module can be used to directly produce an action in Mode-1 $\tilde{a}[0] = A(s[0])$. It can also be used to recursively compute an initial action sequence proposal before Mode-2 optimization:

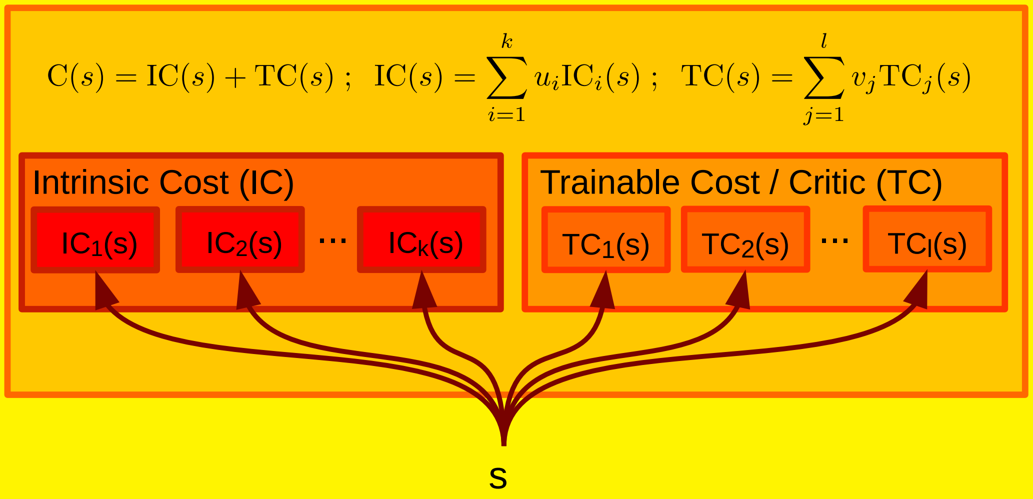
$$s[t+1] = \text{Pred}(s[t], a[t]) \ ; \quad \tilde{a}[t+1] = A(s[t+1])$$

The policy module can be seen as performing a form of amortized inference.

This process allows the agent to use the full power of its world model and reasoning capabilities to acquire new skills that are then "compiled" into a reactive policy module that no longer requires careful planning.

3.1.4 Reasoning as Energy Minimization

The process of elaborating a suitable action sequence in Mode-2 can be seen as a form of reasoning. This form of reasoning is based on *simulation* using the world model, and



$C(s) = \mathrm{IC}(s) + \mathrm{TC}(s)$; $\mathrm{IC}(s) = \sum_{i=1}^{k} u_i \mathrm{IC}_i(s)$; $\mathrm{TC}(s) = \sum_{j=1}^{l} v_j \mathrm{TC}_j(s)$
Intrinsic Cost (IC)
IC1(s)
IC2(s)
ICk(s)
Trainable Cost / Critic (TC)
TC1(s)
TC2(s)
TCl(s)
s

Figure 6: *Architecture of the cost module. The cost module comprises the intrinsic cost module which is immutable* $\mathrm{IC}_i(s)$ *(left) and the critic or Trainable Cost* $\mathrm{TC}_j(s)$ *(right), which is trainable. Both IC and TC are composed of multiple submodules whose output energies are linearly combined. Each submodule imparts a particular behavioral drive in the agent. The weights in the linear combination,* u_i *and* v_j*, are determined by the configurator module and allow the agent to focus on different subgoals at different times.*

optimization of the energy with respect to action sequences. More generally, the "actions" can be seen as latent variables representing abstract transformations from one state to the next. This type of planning though simulation and optimization may constitute the kind of reasoning that is most frequent in natural intelligence.

Many classical forms of reasoning in AI can actually be formulated as optimization problems (or constraint satisfaction problems). It is certainly the case for the kind of probabilistic inference performed with factor graphs and probabilistic graphical models. The proposed architecture is, in fact, a factor graph in which the cost modules are log factors. But the kind of reasoning that the proposed architecture enables goes beyond traditional logical and probabilistic reasoning. It allows reasoning by simulation and by analogy.

3.2 The Cost Module as the Driver of Behavior

The overall architecture of the cost module is shown in Figure 6. It is composed of the intrinsic cost module which is immutable $\mathrm{IC}_i(s)$ and the critic or Trainable Cost $\mathrm{TC}_j(s)$, which is trainable. Both IC and TC are composed of multiple submodules whose output energies are linearly combined

$$C(s) = \mathrm{IC}(s) + \mathrm{TC}(s) \tag{1}$$

$$\mathrm{IC}(s) = \sum_{i=1}^{k} u_i \mathrm{IC}_i(s) \tag{2}$$

$$\mathrm{TC}(s) \quad = \quad \sum_{j=1}^{l} v_j \mathrm{TC}_j(s) \qquad (3)$$

Each submodule imparts a particular behavioral drive to the agent. The weights in the linear combination, u_i and v_j, are modulated by the configurator module and allow the agent to focus on different subgoals at different times.

The intrinsic cost module (IC) is where the basic behavioral nature of the agent is defined. It is where basic behaviors can be indirectly specified.

For a robot, these terms would include obvious proprioceptive measurements corresponding to "pain", "hunger", and "instinctive fears", measuring such things as external force overloads, dangerous electrical, chemical, or thermal environments, excessive power consumption, low levels of energy reserves in the power source, etc.

They may also include basic drives to help the agent learn basic skills or accomplish its missions. For example, a legged robot may comprise an intrinsic cost to drive it to stand up and walk. This may also include social drives such as seeking the company of humans, finding interactions with humans and praises from them rewarding, and finding their pain unpleasant (akin to empathy in social animals). Other intrinsic behavioral drives, such as curiosity, or taking actions that have an observable impact, may be included to maximize the diversity of situations with which the world model is trained (Gottlieb et al., 2013)

The IC can be seen as playing a role similar to that of the amygdala in the mammalian brain and similar structures in other vertebrates.

To prevent a kind of behavioral collapse or an uncontrolled drift towards bad behaviors, the IC must be immutable and not subject to learning (nor to external modifications).

The role of the critic (TC) is twofold: (1) to anticipate long-term outcomes with minimal use of the onerous world model, and (2) to allow the configurator to make the agent focus on accomplishing subgoals with a learned cost.

In general, the behavioral nature of an AI agent can be specified in four ways:

1. by explicitly programming a specific behavior activated when specific conditions are met

2. by defining an objective function in such a way that the desired behavior is executed by the agent as a result of finding action sequences that minimize the objective.

3. by training the agent to behave a certain way through direct supervision. The agent observes the actions of an expert teacher, and trains a Mode-1 policy module to reproduce it.

4. by training the agent through imitation learning. The agent observes expert teachers, and infers an objective function that their behavior appears to be optimizing when they act. This produces a critic submodule for Mode-2 behavior. This process is sometimes called *inverse reinforcement learning.*

The second method is considerably simpler to engineer than the first one, because it merely requires to design an objective, and not design a complete behavior. The second method is also more robust: a preordained behavior may be invalidated by unexpected conditions or a changing environment. With an objective, the agent may adapt its behavior

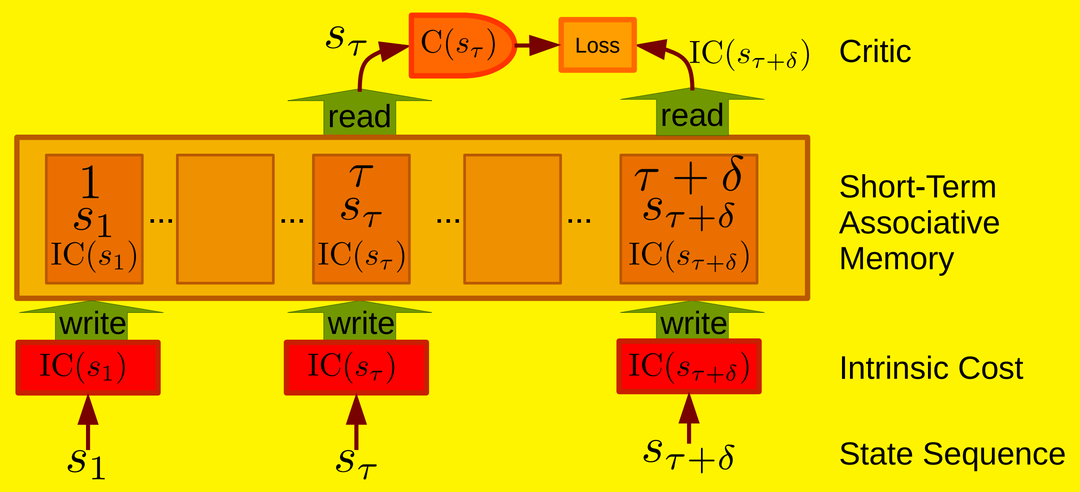


Figure 7: *Training the critic. During planning episodes, the intrinsic cost module stores triplets (time, state, intrinsic energy): $(\tau, s_\tau, IC(s_\tau))$ into the associative short-term memory. During critic training episodes, the critic retrieves a past state vector s_τ, together with an intrinsic energy at a later time* $\text{IC}(s_{\tau+\delta})$. *In the simplest scenario, the critic adjusts its parameters to minimize a divergence measure between the target* $\text{IC}(s_{tau+\delta})$ *and the predicted energy* $\text{C}(s_\tau)$. *In more complex schemes, it may use combinations of future intrinsic energies as targets. Note that the state sequence may contain information about the actions planned or taken by the agent.*

to satisfy the objective despite unexpected conditions and changes in the environment. The second method exploits the learning and inference abilities of the agent to minimize the amount of priors hard-wired by the designer that are likely to be brittle.

3.3 Training the Critic

An essential question is how to train the critic.

The principal role of the critic is to predict future values of the intrinsic energy. To do so, it uses the short-term memory module. This module is an associative memory in which the intrinsic cost module stores triplets (time, state, intrinsic energy): $(\tau, s_\tau, IC(s_\tau))$. The stored states and corresponding intrinsic energies may correspond to a perceived state or to a state imagined by the world model during a Mode-2 episode. The memory may retrieve a state s_τ given a time τ, and may retrieve an energy $IC(s_\tau)$ given a time τ or a state s_τ. With a suitable memory architecture, the retrieval may involve interpolations of keys and retrieved values. The process is shown in Figure 7

The critic can be trained to predict future intrinsic energy values by retrieving a past state vector s_τ together with an intrinsic energy at a later time $\text{IC}(s_{\tau+\delta})$. The parameters of the critic can then be optimized to minimize a prediction loss, for example $||IC(s_{\tau+\delta}) - TC(s_\tau)||^2$. This is a simple scenario. More complex schemes can be devised to predict expectations of discounted future energies, or distributions thereof. Note that the state vectors may contain information about the actions taken or imagined by the actor.

At a general level, this is similar to critic training methods used in such reinforcement learning approaches as A2C.

The short-term memory can be implemented as the memory module in a key-value memory network **??**: a query vector is compared to a number of key vectors, producing a vector of scores. The scores are normalized and used as coefficients to output a linear combination of the stored values. It can be seen as a "soft" associative memory capable of interpolation. One advantage of it is that, with a proper allocation scheme of new key/value slots, it is capable of one-shot learning, yet can interpolate between keys and is end-to-end differentiable.

4 Designing and Training the World Model

Arguably, *designing architectures and training paradigms for the world model constitute the main obstacles towards real progress in AI over the next decades.* One of the main contributions of the present proposal is precisely a hierarchical architecture and a training procedure for world models that can represent multiple outcomes in their predictions.

Training the world model is a prototypical example of Self-Supervised Learning (SSL), whose basic idea is pattern completion. The prediction of future inputs (or temporarily unobserved inputs) is a special case of pattern completion. In this work, the primary purpose of the world model is seen as *predicting future representations of the state of the world.*

There are three main issues to address. First, quite evidently, the quality of the world model will greatly depend on the diversity of state sequences, or triplets of (state, action, resulting state) it is able to observe while training. Second, because the world is not entirely predictable, there may be multiple plausible world state representations that follow a given world state representation and an action from the agent. The world model must be able to meaningfully represent this possibly-infinite collection of plausible predictions. Third, the world model must be able to make predictions at different time scales and different levels of abstraction.

The first issue touches on one of the main questions surrounding learning for sequential decision processes: the diversity of the "training set" depends on the actions taken. The issue is discussed in Section 4.10 below.

The second issue is even more dire: the world is not entirely predictable. Hence, the world model should be able to represent multiple plausible outcomes from a given state and (optionally) an action. This may constitute one of the most difficult challenges to which the present proposal brings a solution. This issue is discussed in Section 4.8 below.

The third issue relates to the problem of long-term prediction and planning. Humans plan complex goals at an abstract level and use high-level descriptions of the world states and actions to make predictions. High-level goals are then decomposed into sequences of more elementary sequences of subgoals, using shorter-term prediction from the world model to produce lower-level actions. This decomposition process is repeated all the way down to millisecond-by-millisecond muscle control, informed by local conditions. The question of how world models could represent action plans at multiple time scales and multiple levels of abstraction is discussed in Section 4.6

4.1 Self-Supervised Learning

Self-Supervised Learning (SSL) is a paradigm in which a learning system is trained to capture the mutual dependencies between its inputs. Concretely, this often comes down to training a system to tell us if various parts of its input are consistent with each other.

For example, in a video prediction scenario, the system is given two video clips, and must tell us to what degree the second video clip is a plausible continuation of the first one. In a pattern completion scenario, the system is given part of an input (image, text, audio signal) together with a proposal for the rest of the input, and tells us whether the proposal is a plausible completion of the first part. In the following, we will denote the observed part of the input by x and the possibly-unobserved part by y.

Importantly, we do not impose that the model be able to *predict* y from x. The reason is that there may be an infinite number of y that are compatible with a given x. In a video prediction setting, there is an infinite number of video clips that are plausible continuations of a given clip. It may be difficult, or intractable, to explicitly represent the set of plausible predictions. But it seems less inconvenient to merely ask the system to tell us if a proposed y is compatible with a given x.

A general formulation can be done with the framework of *Energy-Based Models* (EBM). The system is a scalar-valued function $F(x, y)$ that produces low energy values when x and y are compatible and higher values when they are not. The concept is depicted in Figure 8. Data points are black dots. The energy function produces low energy values around the data points, and higher energies away from the regions of high data density, as symbolized by the contour lines of the energy landscape. The EBM implicit function formulation enables the system to represent multi-modal dependencies in which multiple values of y are compatible with a given x. The set of y compatible with a given x may be a single point, multiple discrete points, a manifold, or a collection of points and manifolds.

To enable Mode-2 planning, a predictive world model should be trained to capture the dependencies between past and future percepts. It should be able to *predict representations of the future from representations of the past and present.* The general learning principle is as follows: given two inputs x and y, learn two functions that compute representations $s_x = g_x(x)$ and $s_y = g_y(y)$ such that (1) s_x and s_y are maximally informative about x and y and (2) s_y can easily be predicted from s_x. This principle ensures a trade-off between making the evolution of the world predictable in the representation space, and capturing as much information as possible about the world state in the representation.

What concepts could such an SSL system learn by being trained on video? Our hypothesis is that a hierarchy of abstract concepts about how the world works could be acquired.

Learning a representation of a small image region such that it is predictable from neighboring regions surrounding it in space and time would cause the system to extract local edges and contours in images, and to detect moving contours in videos. Learning a representation of images such that the representation of a scene from one viewpoint is predictable from the representation of the same scene from a slightly different viewpoint would cause the system to implicitly represent a depth map. A depth map is the simplest way to explain how a view of a scene changes when the camera moves slightly. Once the notion of depth has been learned, it would become simple for the system to identify occlusion edges, as well as the collective motion of regions belonging to a rigid object. An implicit representation of 3D

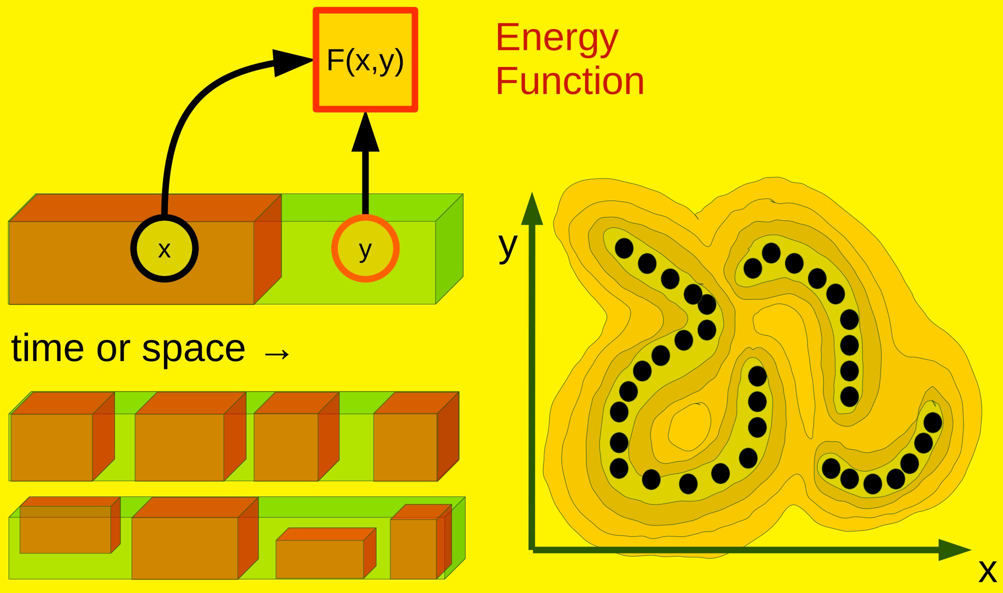


Figure 8: *Self-Supervised Learning (SSL) and Energy-Based Models (EBM). SSL is a learning paradigm in which a learning system is trained to "fill in the blanks", or more precisely to capture the dependencies between observed parts of the input and possibly unobserved parts of the input. Part of the input signal is observed and denoted x (in pink), and part of the input signal is either observed or unobserved and denoted y (in blue). In a temporal prediction scenario, x represents past and present observations, and y represent future observations. In a general pattern completion scenario, various parts of the input may be observed or unobserved at various times. The learning system is trained to capture the dependencies between x and y through a scalar-valued* energy function $F(x, y)$ *that takes low values when x and y are consistent or compatible, and higher values if x and y are inconsistent or incompatible. In a video prediction scenario, the system would produce a low energy value if a video clip y is a plausible continuation of the video clip x. This* energy-based model *(EBM) formulation enables the system to represent multi-modal dependencies in which multiple values of y (perhaps an infinite set) may be compatible with a given x. In the right panel, an energy landscape is represented in which dark discs represent data points, and closed lines represents contours (level sets) of the energy function.*

objects may spontaneously emerge. Once the notion of object emerges in the representation, concepts like object permanence may become easy to learn: objects that disappear behind others due to parallax motion will invariably reappear. The distinction between inanimate and animate object would follow: inanimate object are those whose trajectories are easily predictable. Intuitive physics concepts such as stability, gravity, momentum, may follow by training the system to perform longer-term predictions at the object representation level. One may imagine that through predictions at increasingly abstract levels of representation and increasingly long time scales, more and more complex concepts about how the world works may be acquired in a hierarchical fashion.

The idea that abstract concepts can be learned through prediction is an old one, formulated in various way by many authors in cognitive science, neuroscience, and AI over several decades. The question is how to do it, precisely.

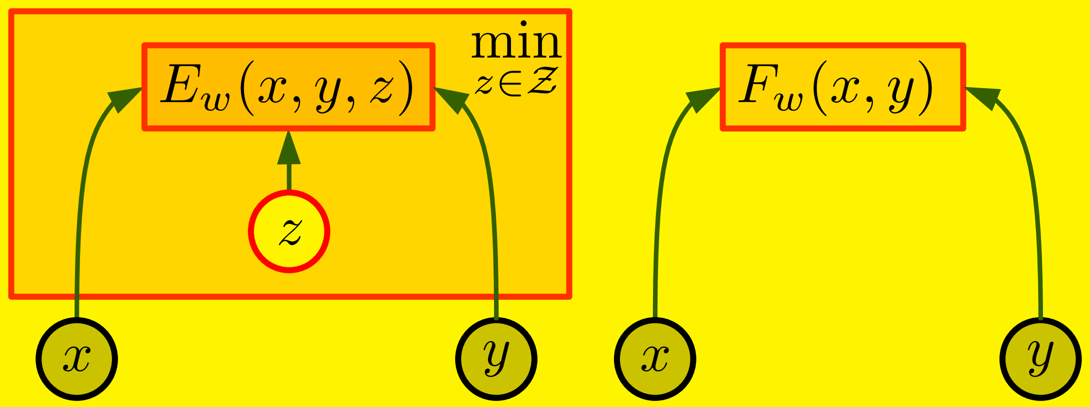


Figure 9: *Latent-Variable Energy-Based Model (LVEBM).*
To evaluate the degree of compatibility between x and y, an EBM may need the help of a latent variable z. The latent variable can be seen as parameterizing the set of possible relationships between an x and a set of compatible y. Latent variables represent information about y that cannot be extracted from x. For example, if x is a view of an object, and y another view of the same object, z may parameterize the camera displacement between the two views. Inference consists in finding the latent that minimizes the energy $\check{z} = \text{argmin}_{z\in\mathcal{Z}} E_w(x,y,z)$. The resulting energy $F_w(x,y) = E_w(x,y,\check{z})$ only depends on x and y. In the dual view example, inference finds the camera motion that best explains how x could be transformed into y.

4.2 Handling Uncertainty with Latent Variables

As was pointed out above, one of the main issues is enabling the model to represent multiple predictions. This may require the use of a latent variable. A latent variable is an input variable whose value is not observed but inferred. A latent variable can be seen as parameterizing the set of possible relationships between an x and a set of compatible y. Latent variables are used to represent information about y that cannot be extracted from x.

Imagine a scenario in which x is a photo of a scene, and y a photo of the same scene from a slightly different viewpoint. To tell whether x and y are indeed views from the same scene, one may need to infer the displacement of the camera between the two views. Similarly, if x is a picture of a car coming to a fork in the road, and y is a picture of the same car a few seconds later on one of the branches of the fork, the compatibility between x and y depends on a binary latent variable that can be inferred: did the car turn left or right.

In a temporal prediction scenario, the latent variable represents what cannot be predicted about y (the future) solely from x and from past observations (the past). It should contain all information that would be useful for the prediction, but is not observable, or not knowable. I may not know whether the driver in front of me will turn left or right, accelerate or brake, but I can represent those options by a latent variable.

A latent-variable EBM (LVEBM) is a parameterized energy function that depends on x, y, and z: $E_w(x,y,z)$. When presented with a pair (x,y) the *inference procedure* of the EBM finds a value of the latent variable z that minimizes the energy

$$\check{z} = \underset{z\in\mathcal{Z}}{\text{argmin}}\, E_w(x,y,z) \tag{4}$$

This latent-variable inference by minimization allows us to eliminate z from the energy function:

$$F_w(x, y) = \min_{z \in \mathcal{Z}} E_w(x, y, z) = E_w(x, y, \check{z}) \tag{5}$$

Technically, $F_w(x, y)$ should be called a *zero-temperature free energy*, but we will continue to call it the energy.

4.3 Training Energy-Based Models

Before we discuss EBM training, it is important to note that the definition of EBM *does not make any reference to probabilistic modeling.* Although many EBMs can easily be turned into probabilistic models, e.g. through a Gibbs distribution, this is not at all a necessity. Hence the energy function is viewed as the fundamental object and is *not* assumed to implicitly represent the unnormalized logarithm of a probability distribution.

Training an EBM consists in constructing an architecture (e.g. a deep neural network) to compute the energy function $F_w(x, y)$ parameterized with a parameter vector w. The training process must seek a w vector that gives the right shape to the energy function. For a given x from the training set, a well-trained $F_w(x, y)$ will produce lower energies for values of y that are associated with x in the training set, and higher energies to other values of y.

Given a training sample (x, y), training an EBM comes down to devising a suitable loss functional $L(x, y, F_w(x, y))$, which can be expressed directly as a function of the parameter vector $L(x, y, w)$, and such that minimizing this loss will make the energy of the training sample $F_w(x, y)$ lower than the energies $F_w(x, \hat{y})$ of any $\hat{y}$ different from y.

Making the energy of the training sample low is easy: it is sufficient for the loss to be an increasing function of the energy, and for the energy to have a lower bound.

The difficult question is *how to ensure that the energies of $\hat{y}$ different from y are higher than the energy of y.* Without a specific provision to ensure that $F_w(x, y') > F_w(x, y)$ whenever $\hat{y} \neq y$ the energy landscape may suffer a *collapse*: given an x the energy landscape could become "flat", giving essentially the same energy to all values of y.

What EBM architectures are susceptible to collapse? Whether an EBM may be susceptible to collapse depends on its architecture. Figure 10 shows a number of standard architectures and indicates whether they can be subject to collapse.

A regular predictive or deterministic-generative architecture (Figure 10(a)) cannot collapse. For any x, a single $\tilde{y}$ is produced. The energy is zero whenever $y = \tilde{y}$. Any y different from $\tilde{y}$ will have a higher energy, as long as $D(y, \tilde{y})$ is strictly larger than zero whenever y is different from $\tilde{y}$.

A generative latent-variable architecture (non-deterministic generative) (Figure 10(b)) can collapse when the latent variable has too much information capacity. When the latent variable z varies over the set $\mathcal{Z}$, the prediction $\tilde{y}$ varies over a set $\text{Pred}(s_x, \mathcal{Z})$, which must match the set of y that are compatible with x. If $\mathcal{Z}$ is too "large" then the region of low-energy y may be larger than the region of high data density. If z has the same dimension as y, the system could very well give zero energy to the entire y space.

An auto-encoder (AE) (Figure 10(c)) can collapse when the representation s_y has too much information capacity. For example, if the dimension of s_y is equal or higher than that

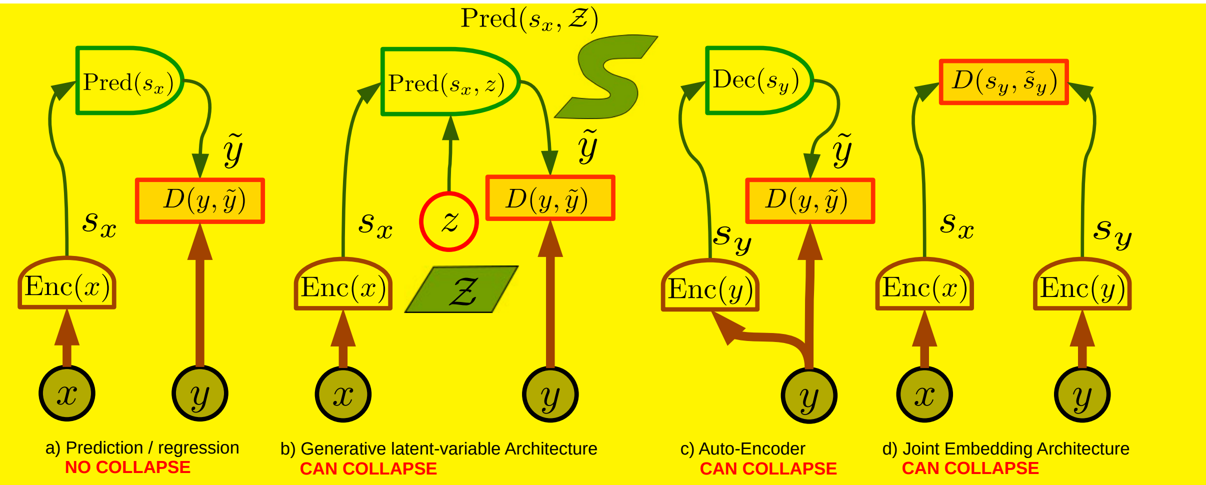


Figure 10: *A few standard architectures and their capacity for collapse.*
(a) Deterministic generative architecture: cannot collapse because it can only produce a single output. For a given x, only one value of y may have zero energy: $y = \tilde{y}$. Other values of y will have higher energy if $D(u, \tilde{y})$ is larger than zero when $y \neq \tilde{y}$.
(b) Non-deterministic generative architecture: can collapse when the latent variable has excessive information capacity. If for a given x and for all y there exists a z that produces zero prediction energy (e.g. if z has the same or higher dimension as y), the entire y space will have low energy. The information capacity of z should be just enough so that varying z over its set will produce all the plausible $\tilde{y}$ for a given x.
(c) Auto-encoder: can collapse if the system learns the identity function or if it can correctly reconstruct a region of y space that is much larger than the region of high data density, thereby giving low energy to an overly large region.
(d) Simple joint embedding architecture: can collapse if the encoders ignore the inputs and produce representations that remain constant and equal, or if the encoders are invariant over overly broad regions of the space.

of y, the AE could learn the identity function, producing a reconstruction error equal to zero over the entire y space.

Lastly, a Joint Embedding Architecture (JEA) (Figure 10(d)) can collapse when the information carried by s_x and/or s_y are insufficient. If the encoders ignore the inputs, and produce constant and equal codes $s_x = s_y$, the entire space will have zero energy.

These are but a few examples of architectures.

How do we design the loss to prevent collapse? There are two approaches: *contrastive methods* and *regularized methods*. In the following, I will argue that contrastive methods have flaws and that regularized (non contrastive) methods are much more likely to be preferable in the long run.

Contrastive methods consist in using a loss functional whose minimization has the effect of pushing down on the energies of training samples (x, y), and pulling up on the energies of suitably-hallucinated "contrastive" samples $(x, \hat{y})$. The contrastive sample $\hat{y}$ should be picked in such a way as to ensure that the EBM assigns higher energies to points outside the regions of high data density. This translates into designing a loss that is an increasing function of $F_w(x, y)$ and a decreasing function of $F_w(x, \hat{y})$, at least whenever $F_w(x, \hat{y})$ is not sufficiently higher than $F_w(x, y)$. There are many such contrastive loss

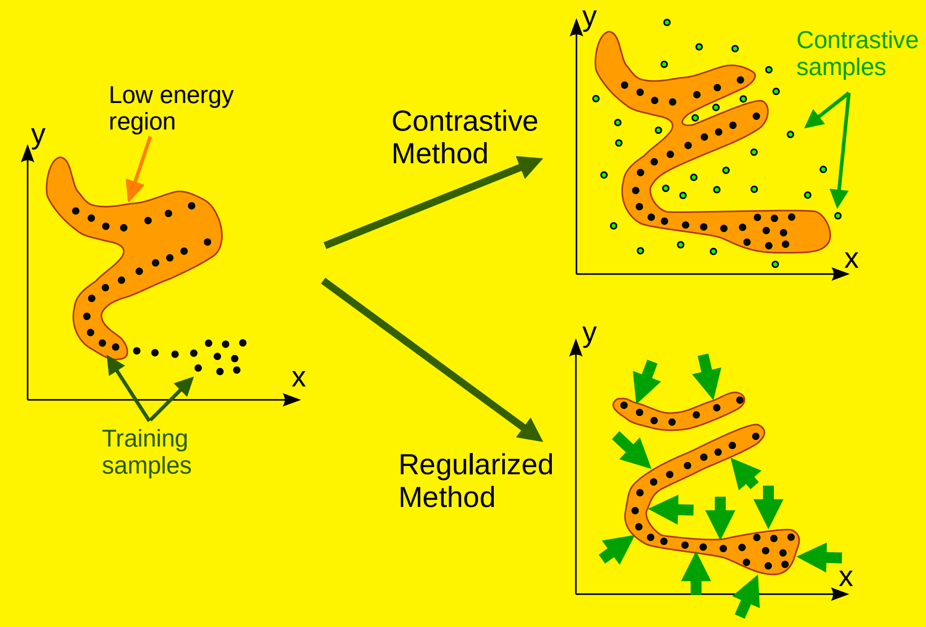


Figure 11: *Contrastive and regularized methods for EBM training. A conceptual diagram of an energy landscape is shown on the left. Training samples are blue dots. The region of low energy is shown in orange (a level set of the energy function).*
Contrastive methods *(top right) push down on the energy of training samples (blue dots) and pulls up on the energies of suitably-placed contrastive samples (green dots).*
regularized methods *(bottom right) push down on the energy of training samples and use a regularizer term that minimizes the volume of low-energy regions. This regularization has the effect of "shrink-wrapping" the regions of high data density within the low-energy regions, to the extent that the flexibility of the energy function permits it.*
One issue with contrastive methods is that the energy will only be pulled up wherever contrastive samples have been placed. One must devise methods that preferentially place contrastive samples in regions of low energy, which is what Monte-Carlo and MCMC methods do. However, a disadvantage of contrastive methods is that the number of contrastive samples necessary to make an energy surface adopt a good shape may grow exponentially with the dimension of y space.

functions, some of them taking a single triplet $(x, y, \hat{y})$, others requiring a batch of positive and contrastive values of y.

A simple example of contrastive loss functions is as follows:

$$L(w, x, y, \hat{y}) = H(F_w(x, y), F_w(x, \hat{y}), m(y, \hat{y})) \tag{6}$$

where H is an increasing function of $F_w(x, y)$, and a decreasing function of $F_w(x, \hat{y})$ whenever the latter is less than the former plus a positive margin function $m(y, \hat{y})$. A simple instance of such loss is the distance-dependent hinge loss:

$$L(w, x, y, \hat{y}) = \left[F_w(x, y) - F_w(x, \hat{y}) + \mu||y - \hat{y}||^2\right]^+ \tag{7}$$

where $[a]^+$ is the identity when a is positive and zero otherwise. This makes the energy grow at least quadratically with the distance to the data manifold. Other contrastive loss functionals take multiple contrastive samples into consideration:

$$L(w, x, y, \hat{y}[1], \dots, \hat{y}[K]) = H(F_w(x, y), F_w(x, \hat{y}[1]), \dots, F_w(x, \hat{y}[K])) \tag{8}$$

Which must be an increasing function of the first argument, and a decreasing function of all other arguments. An example of such loss is the popular InfoNCE:

$$L(w, x, y, \hat{y}[1], \dots, \hat{y}[K]) = F_w(x, y) + \log \left[\exp(-F_w(x, y)) + \sum_{k=1}^{K} \exp(-F_w(x, \hat{y}[k])) \right] \tag{9}$$

Contrastive methods are very popular, particularly for Siamese network architectures trained with pairs where x is a distorted or corrupted version of y and $\hat{y}$ another random (or suitably chosen) training sample. This includes such methods as the original Siamese net, as well as more recent methods including DrLIM, PIRL, MoCO, SimCLR, CPT, and others. Contrastive methods also include such classical methods as probabilistic models trained with maximum likelihood that are not automatically normalized. Contrastive samples $\hat{y}$ are often produced using Monte Carlo methods, Markov-Chain Monte Carlo methods, or approximate versions thereof, such as Contrastive Divergence. Generative Adversarial Networks can also be seen as contrastive methods in which the $\hat{y}$ are produced by the trainable generator network. Denoising Auto-Encoders and their special case, Masked Auto-Encoders, are also examples of contrastive training methods in which the $\hat{y}$ is generated by corrupting the clean y. A more detailed discussion of various contrastive methods is given in appendix 8.3.3.

But there are two main issues with contrastive methods. First, one has to design a scheme to generate or pick suitable $\hat{y}$. Second, when y is in a high-dimensional space, and if the EBM is flexible, it may require a very large number of contrastive samples to ensure that the energy is higher in all dimensions unoccupied by the local data distribution. Because of the curse of dimensionality, in the worst case, the number of contrastive samples may grow exponentially with the dimension of the representation. This is the main reason why I will argue against contrastive methods.

Regularized methods for EBM training are much more promising in the long run than contrastive methods because they can eschew the curse of dimensionality that plagues contrastive methods. They consist in constructing a loss functional that has the effect of pushing down on the energies of training samples, and simultaneously *minimizing the volume of y space to which the model associates a low energy.* The volume of the low-energy region is measured by a regularization term in the energy and/or in the loss. By minimizing this regularization term while pushing down on the energies of data points, the regions of low energy will "shrink-wrap" the regions of high data density. The main advantage of non-contrastive regularized methods is that they are less likely than contrastive methods to fall victim to the curse of dimensionality. The main question is precisely how to design such volume-minimizing regularizers. The answer depends greatly on the architecture of the model, which is discussed in the next sections. However, non-contrastive methods have existed for a long time. Examples include sparse modeling, sparse auto-encoders, and auto-encoders with noisy latent variables, such as VAE.

It is important to note that contrastive and regularized methods are not incompatible with each other, and can be used simultaneously on the same model.

How would regularized methods apply to the architectures of Figure 10(b-d)?

In the latent-variable generative architecture, restricting the information capacity of z will restrict the volume of y space that can take low energy. If z is discrete with possible k values, at most k points in y space will have zero energy. if $\mathcal{Z}$ is a manifold of dimension d then the region of y space with zero energy will have at most d dimensions.

Similarly, in the auto-encoder architecture, restricting the information capacity of s_y will restrict the volume of y space that can be reconstructed with low energy.

Lastly, in the Joint Embedding Architecture, Maximizing the information that s_x contains about x and s_y contains about y will minimize the volume of y space that can take low energy.

In the following, we will focus on an architecture for SSL the Joint Embedding Predictive Architectures (JEPA) which can seen as a combination of the Joint Embedding Architecture and the Latent-Variable Generative Architecture. The JEPA is non-generative in that it does not actually predict y, but predicts the representation of y, s_y from that of x, s_x.

4.4 Joint Embedding Predictive Architecture (JEPA)

The centerpiece of this paper is the *Joint Embedding Predictive Architecture (JEPA)*. JEPA is *not generative* in the sense that it cannot easily be used to predict y from x. It merely capture the dependencies between x and y without explicitly generating predictions of y.

A generic JEPA is shown in Figure 12. The two variables x and y are fed to two encoders producing two presentations s_x and s_y. These two encoders may be different. They are *not* required to possess the same architecture nor are they required to share their parameters. This allows x and y to be different in nature (e.g. video and audio). A predictor module predicts the representation of y from the representation of x. The predictor may depend on a latent variable z. The energy is simply the prediction error in representation space:

$$E_w(x, y, z) = D(s_y, \text{Pred}(s_x, z)) \tag{10}$$

The overall energy is obtained by minimizing over z:

$$\begin{aligned} \check{z} &= \operatorname*{argmin}_{z \in \mathcal{Z}} E_w(x, y, z) = \operatorname*{argmin}_{z \in \mathcal{Z}} D(s_y, \text{Pred}(s_x, z)) && (11) \\ F_w(x, y) &= \min_{z \in \mathcal{Z}} E_w(x, y, z) = D(s_y, \text{Pred}(s_x, \check{z})) && (12) \\ & && (13) \end{aligned}$$

The main advantage of JEPA is that *it performs predictions in representation space*, eschewing the need to predict every detail of y. This is enabled by the fact that the encoder of y may choose to produce an abstract representation from which irrelevant details have been eliminated.

But there are two ways a JEPA may represent the multiplicity of values of y compatible with x. The first one is invariance properties of the y encoder, the second one is the latent variable z, as explained below.

multi-modality through encoder invariance: The encoder function $s_y = \text{Enc}(y)$ may have invariance properties. If all the y's in a set map to the same value of s_y, all those y's will have identical energies. With JEPA, we lose the ability to generate outputs, but we gain a powerful way to represent multi-modal dependencies between inputs and outputs.

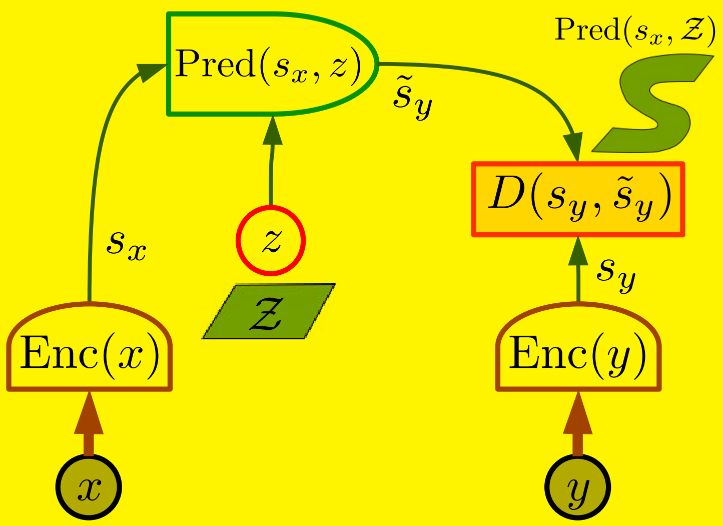


Figure 12: *The Joint-Embedding Predictive Architecture (JEPA) consists of two encoding branches. The first branch computes s_x, a representation of x and the second branch s_y a representation of y. The encoders do not need to be identical. A predictor module predicts s_y from s_x with the possible help of a latent variable z. The energy is the prediction error. Simple variations of the JEPA may use no predictor, forcing the two representations to be equal, or may use a fixed predictor with no latent, or may use simple latents such as discrete variables.*
The main advantage of JEPA is that it performs predictions in representation space, eschewing the need to predict every detail of y, and enabling the elimination of irrelevant details by the encoders. More precisely, the main advantage of this architecture for representing multi-modal dependencies is twofold: (1) the encoder function $s_y = \mathrm{Enc}(y)$ may possess invariance properties that will make it produce the same s_y for a set of different y. This makes the energy constant over this set and allows the model to capture complex multi-modal dependencies; (2) The latent variable z, when varied over a set $\mathcal{Z}$, can produce a set of plausible predictions $\mathrm{Pred}(s_x, \mathcal{Z}) = \{\tilde{s}_y = \mathrm{Pred}(s_x, z)\ \forall z \in \mathcal{Z}\}$
If x is a video clip of a car approaching a fork in the road, s_x and s_y may represent the position, orientation, velocity and other characteristics of the car before and after the fork, respectively, ignoring irrelevant details such as the trees bordering the road or the texture of the sidewalk. z may represent whether the car takes the left branch or the right branch of the road.

multi-modality through latent variable predictor: The predictor may use a latent variable z to capture the information necessary to predict s_y that is not present in s_x. When z is varied over a set $\mathcal{Z}$, the predictor produces a set of plausible predictions $\mathrm{Pred}(s_x, \mathcal{Z}) = \{\tilde{s}_y = \mathrm{Pred}(s_x, z)\ \forall z \in \mathcal{Z}\}$. For example, if x is a video clip of a car approaching a fork in the road, s_x and s_y may represent the past and future positions, orientations, velocities and other characteristics of the car, ignoring irrelevant details such as the trees bordering the road or the texture of the sidewalk. The latent z may be a binary variable indicating whether the car takes the left branch ($z = 0$) or the right branch ($z = 1$ if the road. If the car takes the left branch, the value $z = 0$ will produce a lower energy $D(s_y, \tilde{s}_y)$ than $z = 1$.

4.5 Training a JEPA

Like any EBM, a JEPA can be trained with contrastive methods. But, as pointed out above, contrastive methods tend to become very inefficient in high dimension. The relevant

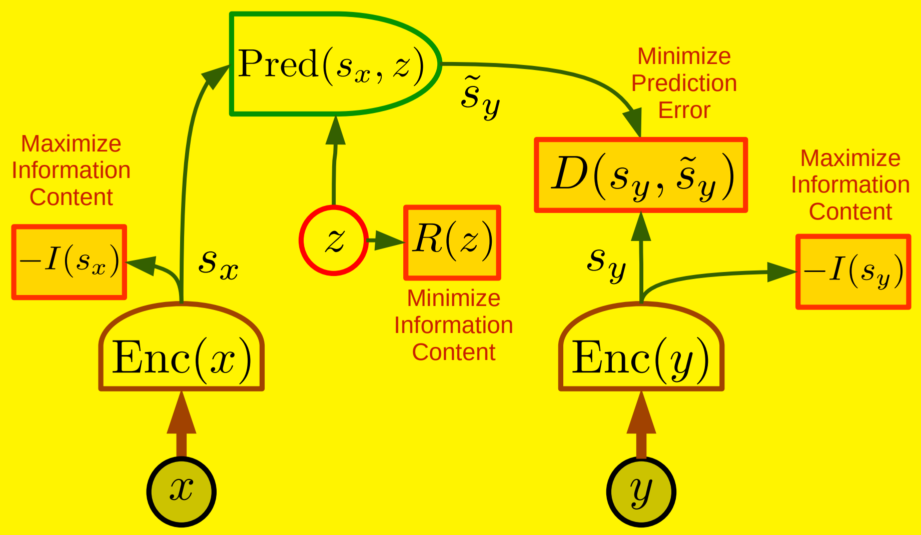

Pred(s_x, z)
$\tilde{s}_y$
Minimize Prediction Error
$D(s_y, \tilde{s}_y)$
Maximize Information Content
$-I(s_x)$
s_x
z
$R(z)$
Minimize Information Content
s_y
Maximize Information Content
$-I(s_y)$
Enc(x)
Enc(y)
x
y

Figure 13: *Non-contrastive training of JEPA.*
The main attraction of JEPAs is that they can be trained with non-contrastive methods. The basic principle of such training is that (1) s_x should be maximally informative about x; (2) s_y should be maximally informative about y; (3) s_y should be easily predictable from s_x; and (4) z should have minimal information content. Criteria 1, 2, and 4 collectively prevent a collapse of the energy function.
Examples of such non-contrastive criteria for JEPA training include VICReg and Barlow Twins.
As with every EBM, JEPAs can also be trained with contrastive methods. But doing so runs into the curse of dimensionality and limits the practical dimension of s_y.

dimension here is that of s_y, which may be considerably smaller than y, but still too high for efficient training.

What makes JEPAs particularly interesting is that we can devise *non-contrastive methods* to train them. As explained in section 4.3, non-contrastive methods use regularizers that measure the volume of space that can take low energy values. In the case of the JEPA, this can be done through four criteria, as depicted in Figure 13:

1. maximize the information content of s_x about x
2. maximize the information content of s_y about y
3. make s_y easily predictable from s_x
4. minimize the information content of the latent variable z used in the prediction.

Criteria 1 and 2 prevent the energy surface from becoming flat by *informational collapse*. They ensure that s_x and s_y carry as much information as possible about their inputs. Without these criteria the system could choose to make s_x and s_y constant, or weakly informative, which would make the energy constant over large swaths of the input space.
Criterion 3 is enforced by the energy term $D(s_y, \tilde{s}_y)$ and ensures that y is predictable from *x in representation space.*
Criterion 4 prevents the system from falling victim to another type of informational collapse by forcing the model to predict s_y with as little help from the latent as possible. This

type of collapse can be understood with the following thought experiment. Imagine that z has the same dimension as s_y. Assume that the predictor is a parameterized function (e.g. a neural net) that can choose to ignore s_x and to simply copy z on its output $\tilde{s}_y = z$. For any s_y it is possible to set $\check{z} = s_y$, which would make the energy $D(s_y, \tilde{s}_y)$ zero. This corresponds to a totally flat and collapsed energy surface.
How do we prevent this collapse from happening?
By limiting or minimizing the information content of the latent variable.
How can this be done?
By making z discrete, low-dimensional, sparse, or noisy, among other methods.

A few concrete examples may help build an intuitive understanding of the phenomenon. Suppose that $D(s_y, \tilde{s}_y) = ||s_y - \tilde{s}_y||^2$ and that z is discrete with K possible integer values $[0, K-1]$. For a given x, there can be only K possible values of $\tilde{s}_y$:

$$\text{Pred}(s_x, 0), \text{Pred}(s_x, 1), \ldots, \text{Pred}(s_x, K-1).$$

Hence, these can be the only values of s_y with zero energy, and there are only K of them. Consider a point s_y that starts from $\text{Pred}(s_x, 0)$ and moves towards $\text{Pred}(s_x, 1)$. Its energy will start from zero, increase quadratically as s_y moves away from $\text{Pred}(s_x, 0)$, until s_y. When s_y becomes closer to $\text{Pred}(s_x, 1)$ than to $\text{Pred}(s_x, 0)$, the energy will decrease, reaching zero when s_y reaches $\text{Pred}(s_x, 1)$. In representation space, the energy will be the minimum of K quadratic energy wells.

Similarly, imagine that z is a vector whose dimension d is lower than that of $\tilde{s}_y$. Then, assuming that $\text{Pred}(s_x, z)$ is a smooth function of z, the set of possible predictions will be at most a d-dimensional manifold in the space of s_y.

More to the point, imagine that the energy function is augmented by a regularization term on z of the form $R(z) = \alpha \sum_{i=1}^{d} |z_i|$, i.e. the L_1 norm of z. This will drive $\check{z}$ to be sparse. As with classical sparse coding, this will cause the region of low energy to be approximated by a union of low-dimensional manifolds (a union of low-dimensional linear subspaces if $\text{Pred}(s_x, z)$ is linear in z), whose dimension will be minimized by the L_1 regularizer.

Making z a stochastic sample from a distribution whose entropy is maximized will also have a proper regularization effect. This is the basis of Variational Auto-Encoders and similar models.

A more complete discussion of regularizers that can minimize the information content of latent variables is beyond the scope of this paper. For now, we can mention four classes of methods: discretization/quantification (e.g. as in VQ-VAE (Walker et al., 2021), dimensionality/rank minimization (e.g. as in Implicit Rank-Minimizing AE (Jing et al., 2020), sparsification (as in linear sparse modeling (Olshausen and Field, 1996), LISTA (Gregor and LeCun, 2010b), and non-linear sparse modeling (Evtimova and LeCun, 2022)), and fuzzyfication (as in noisy AE (Doi et al., 2007), VAE (Kingma and Welling, 2013), and variants used in control problems (Henaff et al., 2019)).

The ability of the JEPA to predict in representation space makes it considerably preferable to generative models that directly produce a prediction of y. In a video prediction scenario, it is essentially impossible to predict every pixel value of every future frame. The details of the texture on a carpet, the leaves of a tree moving in the wind, or the ripples on a pond, cannot be predicted accurately, at least not over long time periods and not without consuming enormous resources. A considerable advantage of JEPA is that *it can choose to*

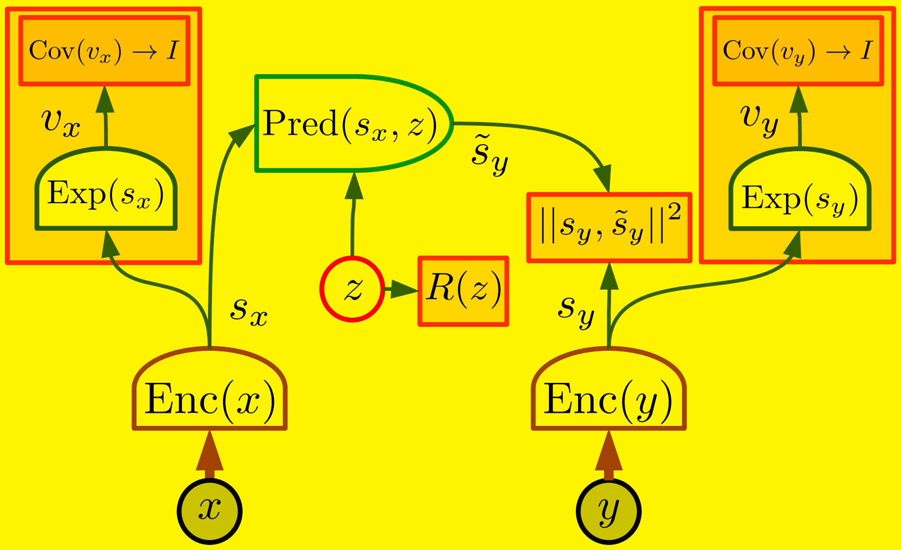


Figure 14: *Training a JEPA with VICReg.*
VICReg is a non sample-contrastive method for training embeddings. The information content of the representations s_x and s_y is maximized by first mapping them to higher-dimensional embeddings v_x and v_y through an expander (e.g. a trainable neural net with a few layers). The loss function drives the covariance matrix of the embeddings towards the identity (e.g. computed over a batch).
VICReg can be seen as a dimension-contrastive method *as opposed to sample-contrastive methods.*

ignore details of the inputs that are not easily predictable. Yet, Criteria 1 and 2 will ensure that the information content of the ignored details are kept to a minimum.

How can we implement Criteria 1 and 2?
In other words, given a parameterized deterministic encoding function $s_y = \mathrm{Enc}_w(y)$, how do we maximize the information content of s_y?
If $\mathrm{Enc}_w(y)$ is invertible, s_y contains all the information about y, but that may be sub-optimal for Criterion 3, as s_y will contain many irrelevant or hard-to-predict details about y. More precisely, s_y is maximally informative about y if the function $\mathrm{Enc}_w(y)$ is minimally surjective, i.e. if the volume of sets of y that map to the same s_y is minimal. The same reasoning applies to the x encoder. To turn this criterion into a differentiable loss, we need to make some assumptions.

4.5.1 VICReg

The VICReg method (Bardes et al., 2021) makes a few assumptions about the distributions of s_x and s_y. A graphical representation is shown in Figure 14. To maximize the information content of s_x, VICReg uses the following two sub-criteria: (1) the components of s_x must not be constant, (2) the components of s_x must be as independent of each other as possible. This is approximated by first non-linearly mapping s_x and s_y to higher-dimensional embeddings v_x and v_y through a trainable expander module (e.g. a neural net with a few layers), and using a loss function with two differentiable loss terms computed over a batch of samples:

1. **Variance**: a hinge loss that maintains the standard deviation of each component of s_y and v_y above a threshold over a batch.

2. **Covariance**: a covariance loss in which the covariance between pairs of different components of v_y are pushed towards zero. This has the effect of decorrelating the components of v_y, which will in turn make the components of s_y somewhat independent.

The same criteria are applied to s_x and v_x separately.

The third criterion of VICReg is the representation prediction error $D(s_y, \tilde{s}_y)$. In the simplest implementations of VICReg, the predictor is constant (equal to the identity function), making the representations **invariant** to the transformation that turns x into y. In more sophisticated versions, the predictor may have no latent variable, or may depend on a latent variable that is either discrete, low dimensional, or stochastic.

The fourth criterion is necessary when the predictor uses a latent variable whose information content must be minimized, for example a vector whose dimension approaches or surpasses that of $\tilde{s}_y$.

A simple instantiation of VICReg to learn invariant representations consists in making x and y be different views (or distorted versions) of the same content, setting the predictor to the identity function, and defining $D(s_y, \tilde{s}_y) = D(s_y, s_x) = ||s_y - s_x||^2$.

Inferring the latent variable through gradient-based methods may be onerous. But the computational cost can be greatly reduced by using amortized inference, as explained in Appendix 8.3.3.

While contrastive methods ensure that representations of different inputs in a batch are different, VICReg ensures that different components of representations over a batch are different. VICReg is contrastive over components, while traditional contrastive methods are contrastive over vectors, which requires a large number of contrastive samples.

But the most promising aspect of JEPA trained with VICReg and similar non-contrastive methods is for *learning hierarchical predictive world models*, as we examine in the next section.

4.5.2 Biasing a JEPA towards learning "useful" representations

With the training criteria listed above, the JEPA finds a trade-off between the completeness and the predictability of the representations. What is predictable and what does not get represented is determined implicitly by the architectures of the encoders and predictor. They determine a inductive bias that defines what information is predictable or not.

But it would be useful to have a way to bias the system towards representations that contain information relevant to a class of tasks. This can be done by adding prediction heads that take $\tilde{s}_y$ as input and are trained to predict variables that are easily derived from the data and known to be relevant to the task.

4.6 Hierarchical JEPA (H-JEPA)

JEPA models trained non-contrastively may constitute our best tool for learning world models that are able to learn relevant abstractions. When trained with VICReg and similar

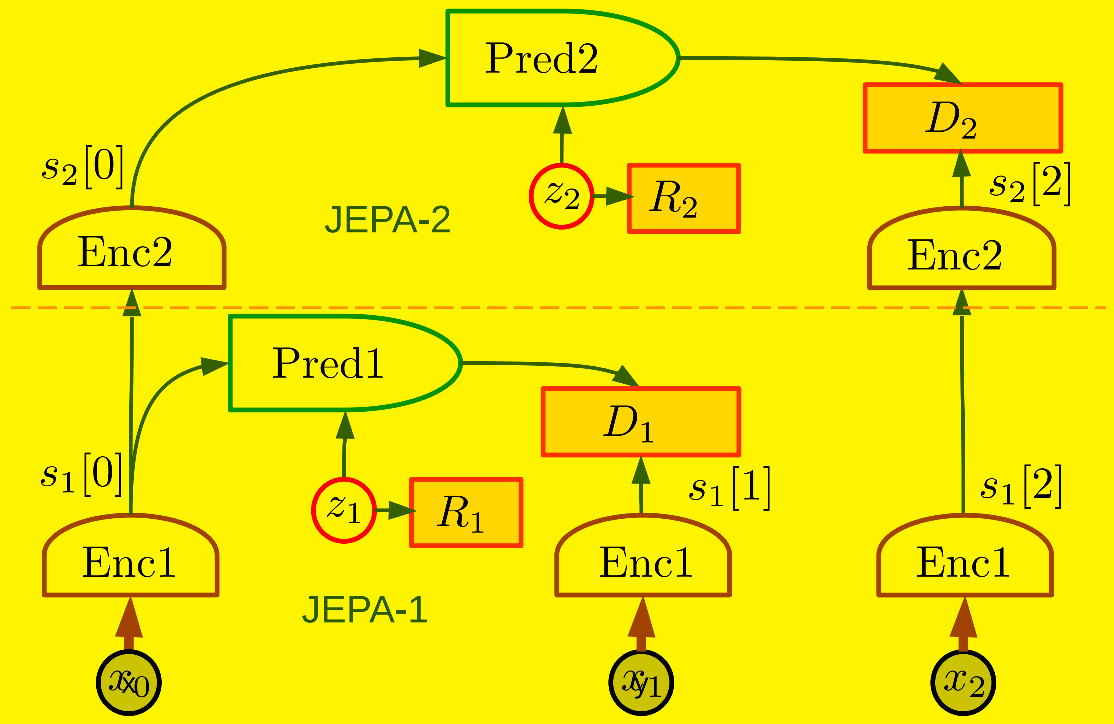


Figure 15: *Hierarchical JEPA (H-JEPA)*
The ability of the JEPA to learn abstract representations in which accurate prediction can be performed allows hierarchical stacking. In this diagram JEPA-1 extracts low-level representations and performs short-term predictions. JEPA-2 takes the representations extracted by JEPA-1 as inputs and extracts higher-level representations with which longer-term predictions can be performed. More abstract representations ignore details of the inputs that are difficult to predict in the long term, enabling them to perform longer-term predictions with coarser descriptions of the world state.

criteria, a JEPA can choose to train its encoders to eliminate irrelevant details of the inputs so as to make the representations more predictable. In other words, a JEPA will learn abstract representations that make the world predictable. Unpredictable details will be eliminated by the invariance properties of the encoder, or will be pushed into the predictor's latent variable. The amount of information thereby ignored will be minimized by the training criteria and by the latent variable regularizer.

It is important to note that *generative latent-variable models are not capable of eliminating irrelevant details*, other than by pushing them into a latent variable. This is because they do not produce abstract (and invariant) representations of y. This is why we advocate *against* the use of generative architectures.

The capacity of JEPA to learn abstractions suggests an extension of the architecture to handle prediction at multiple time scales and multiple levels of abstraction. Intuitively, low-level representations contain a lot of details about the input, and can be used to predict in the short term. But it may be difficult to produce accurate long-term predictions with the same level of details. Conversely high-level, abstract representation may enable long-term predictions, but at the cost of eliminating a lot of details.

Let's take a concrete example. When driving a car, given a proposed sequence of actions on the steering wheel and pedals over the next several seconds, drivers can accurately predict the trajectory of their car over the same period. The details of the trajectory over longer periods are harder to predict because they may depend on other cars, traffic lights, pedestrians, and other external events that are somewhat unpredictable. But the driver can still make accurate predictions at a higher level of abstraction: ignoring the details of trajectories, other cars, traffic signals, etc, the car will probably arrive at its destination within a predictable time frame. The detailed trajectory will be absent from this level of description. But the approximate trajectory, as drawn on a map, is represented. A discrete latent variable may be used to represent multiple alternative routes.

Figure 15 shows a possible architecture for multilevel, multi-scale world state prediction. Variables x_0, x_1, x_2 represent a sequence of observations. The first-level network, denoted JEPA-1 performs short-term predictions using low-level representations. The second-level network JEPA-2 performs longer-term predictions using higher-level representations. One can envision architectures of this type with many levels, possibly using convolutional and other modules, and using temporal pooling between levels to coarse-grain the representation and perform longer-term predictions. Training can be performed level-wise or globally, using any non-contrastive method for JEPA.

I submit that the ability to represent sequences of world states at several levels of abstraction is essential to intelligent behavior. With multi-level representations of world states and actions, a complex task can be decomposed into successively more detailed sub-tasks, instantiated into actions sequences when informed by local conditions. For example, planning a complex task, like commuting to work, can be decomposed into driving to the train station, catching a train, etc. Driving to the train station can be decomposed into walking out of the house, starting the car, and driving. Getting out of the house requires standing up, walking to the door, opening the door, etc. This decomposition descends all the way down to millisecond-by-millisecond muscle controls, which can only be instantiated when the relevant environmental conditions are perceived (obstacles, traffic lights, moving objects, etc).

4.7 Hierarchical Planning

If our world model can perform predictions hierarchically, can it be used to perform Mode-2 reasoning and planning hierarchically?

Hierarchical planning is a difficult topic with few solutions, most of which require that the intermediate vocabulary of actions be predefined. But if one abides by the deep learning philosophy, those *intermediate representations of action plans should also be learned.*

Figure 16 shows a possible architecture for hierarchical Mode-2 planning that can exploit the hierarchical nature of a multi-scale world model.

A percept is encoded into representations at multiple levels of abstractions by a cascade of encoders:

$$s[0] = \text{Enc1}(x); \quad s2[0] = \text{Enc2}(s[0]); \; \ldots \tag{14}$$

Prediction takes place at all levels. Higher levels perform longer-term prediction, while lower levels perform shorter-term predictions. The overall task is defined by a high-level objective, depicted as $C(s_2[4])$ in the diagram. The top level infers a sequence of high-level actions

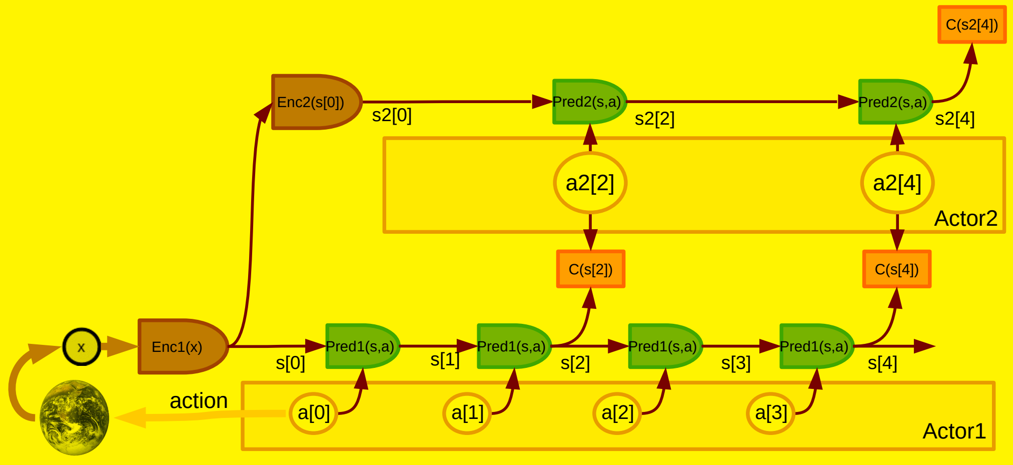


Figure 16: *Hierarchical JEPA for Mode-2 hierarchical planning.*
A complex task is defined by a high-level cost computed from a high-level world-state representation $C(s2[4])$. A sequence of high-level abstract actions $(a2[2], a2[4])$ is inferred that minimizes $C(s2[4])$. The inferred abstract actions are fed to lower-level cost modules $C(s[2]), C(s[4])$ which define subgoals for the lower layer. The lower layer then infers an action sequence that minimizes the subgoal costs. Although only a 2-layer hierarchy is shown here, it is straightforward to extend the concept to multiple levels.
The process described here is sequential top-down, but a better approach would be to perform a joint optimization of the actions in all the layers.

$(a2[2], a2[4])$ to optimize this objective. These high-level "actions" are not real actions but targets for the lower level predicted states. One can think of them as conditions that the lower-level state must satisfy in order for the high-level predictions to be accurate. Whether these conditions are satisfied can be computed by cost modules $C(s[2])$ and $C(s[4])$. They take a lower-level state $s[2]$ and a high-level condition $a2[2]$ and measure to what extent the state satisfies the condition. With these subgoals defined, the lower level can perform inference and find a low-level action sequence that minimizes the mid-level subgoals $C(s[2])$ and $C(s[4])$.

The process just described is top down and greedy. But one may advantageously iterate the optimization so that high level and low-level action sequences are optimized jointly. The cost modules may be configured by the configurator for the situation at hand.

The idea that an action is merely a condition to be satisfied by the level below is actually an old one in control theory. For a example, a classical proportional servomechanism can be seen as being given a target state. A quadratic cost measures the squared distance between the target and the current state, and the control is simply proportional to the negative gradient of the cost with respect to the action variables.

4.8 Handling uncertainty

The real world is not entirely predictable. Uncertainty in predictions of future world states may be due to a number of reasons:

- the world is intrinsically stochastic (aleatoric uncertainty, type 1)

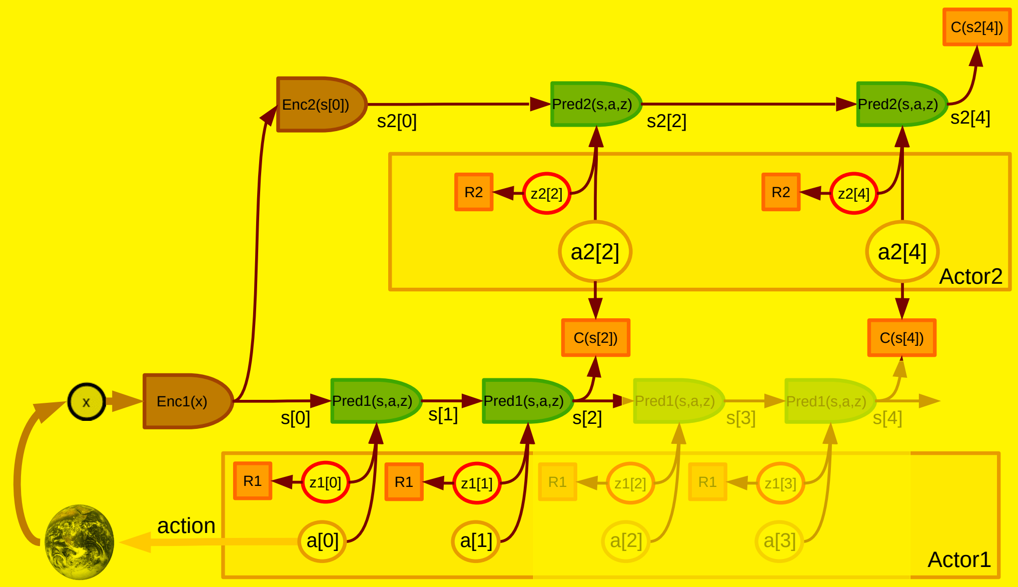


Figure 17: *Hierarchical JEPA for Mode-2 hierarchical planning in an uncertain environment. Realistic environments are not entirely predictable, even when using highly-abstract representations. Uncertainty about predictions can be handled by predictors with latent variables. The latent variables (red circles) contain information about the prediction that cannot be derived from the prior observation. The latent variables must be regularized to prevent an energy collapse and to force the system to predict as much as possible without the help of it.*
At planning time, latent variables are sampled from distributions obtained by applying a Gibbs distribution to the regularizers. Each sample leads to a different prediction. To produce consistent latent sequences, the parameters of the regularizer can be functions of previous states and retrieved memories.
As the prediction progresses, the number of generated state trajectories may grow exponentially. If each latent variable has k possible discrete values, the number of possible trajectories will grow as k^t, where t is the number of time steps. Directed search and pruning strategies must be employed. With multiple predicted trajectories, optimal action sequences can be computed that minimize the average cost, or a combination of average and variance of the cost so as to minimize risk.

- the world is deterministic but chaotic, hence difficult to predict without infinitely precise perception (aleatoric uncertainty, type 2)
- the world is deterministic but partially observable (aleatoric uncertainty type 3).
- the world is fully observable, but the sensors only give partial information about the world state (epistemic uncertainty, type 1)
- the representation of the world state extracted by the perception module does not contain the full information necessary for accurate prediction (epistemic uncertainty, type 2).
- the world model is inaccurate due to limitations of its representational power (bounded rationality or epistemic uncertainty, type 3).

- the world model is inaccurate due to it having been trained with limited amount of data (epistemic uncertainty, type 4).

Much of the literature in reinforcement learning is focused on dealing with the stochastic nature of the environment. It is often assumed from the start that models, critics and policies must represent distributions. In the present work, we push the possible stochasticity of a predicted variable into a latent variable, which may be optimized, predicted, or sampled. This is what is often referred to in the ML literature as "the reparameterization trick". We do not need to use this trick here, since we view the latent-variable parameterization of the predictions as fundamental.

Figure 17 represents a hierachical planning episode in the presence of uncertainty.

A prediction at a given level and time step, e.g. $s2[2]$ requires a sample of the corresponding latent variable $z2[2]$. The sample may come from the distribution whose negative logarithm is the regularizer $R2(z2[2]))$. The parameters of the regularizer may be constant (e.g. fixed Gaussian), predicted from currently-available data using amortized inference (e.g. a multinomial or Gaussian whose parameters are computed from $s2[0]$) or produced by the configurator. Using previous predictions to configure the latent regularizer biases the system towards generating "good" trajectories.

As the prediction progresses, the number of generated state trajectories may grow exponentially: if each latent variable has k possible discrete values, the number of possible trajectories will grow as k^t, where t is the number of time steps. Directed search and pruning strategies can be employed, as in classical Monte-Carlo Tree Search (MCTS). In the case of continuous latents, one may sample latents from the continuous distributions defined by the regularizer.

Given a sample of all the latents, the optimal action sequences at every levels can be inferred. However, the prediction process may need to be repeated for multiple drawings of the latents, so as to cover the set of plausible outcomes. The inference process may be used for multiple predictions to produce an action that does not just minimize the expected cost, but also minimizes the uncertainty on the expected cost.

4.8.1 World Model Architecture

The details of the architecture of the world model should depend on the type of environment the agent evolves in.

It is likely that the best module architectures in a JEPA should include some sort of gating or dynamic routing mechanism.

For example, the best way to handle low-level, short-term predictions in videos is by extracting simple local feature vectors and displacing those feature vectors from one frame to the next, depending on predicted motions. The latent variables may encode a map of displacements, which can modulate routing connections between one frame and the next.

For longer-term prediction at a higher level of abstraction, the relevant features are objects and their interactions. The evolution may be best modeled by a transformer architecture, which has the property of being equivariant to permutation and is appropriate to capture interactions between discrete objects (Vaswani et al., 2017; Carion et al., 2020; Battaglia et al., 2016).

Separating the World Model from the Ego Model: The natural world is complex and somewhat unpredictable, requiring a powerful model with latent variables to account for the unpredictability.

On the other hand, the agent itself is somewhat more predictable: a particular action on effector will produce a motion that can often be predicted deterministically. This suggests that the agent should possess a separate model of itself, perhaps without latent variables (Sobal et al., 2022) as the effect of actions on proprioception somewhat easier to predict than the evolution of the external world or the effect of actions on it.

In turn, the ego-model that the agent has of itself can be used as a template for a model of other agents in a multi-agent scenario.

4.9 Keeping track of the state of the world

Traditionally, modules in deep learning architectures communicate states through vectors or multi-dimensional arrays. But this tends to be a very inefficient method when the state of the object being modeled only changes in minor ways from one time to the next.

A typical action of an agent will only modify a small portion of the state of the world. If a bottle is being moved from the kitchen to the dining room, the states of the bottle, the kitchen, and the dining room will be modified. But the rest of the world will be unaffected.

This suggests that the state of the world should be maintained in some sort of writable memory. Whenever an event occurs, only the part of the world-state memory affected by the event is to be updated, while the rest is to be left unchanged.

A conventional key-value associative memory can be used for this purpose, similar to what has been proposed in the context of memory-augmented networks (Bordes et al., 2015; Sukhbaatar et al., 2015; Miller et al., 2016), and entity networks (Henaff et al., 2017).

The output of the world model at a given time step is a set of query-value pairs $(q[i], v[i])$, which are used to modify existing entries in the world-state memory, or to add new entries.

Given a query q, the world-state memory returns

$$\begin{aligned} \text{Mem}(q) &= \sum_j c_j v_j && (15) \\ \tilde{c}_j &= \text{Match}(k_j, q) && (16) \\ c &= \text{Normalize}(\tilde{c}) && (17) \end{aligned}$$

where the k_j are keys, the v_j are stored values, function $\text{Match}(k, q)$ measures a divergence or dissimilarity between a key and a query, vector c contains scalar coefficients c_j, and function $\text{Normalize}(\tilde{c})$ performs some sort of competitive normalization or thresholding, such as the commonly-used $c_j = \exp(\tilde{c}_j)/[\gamma + \sum_k \exp(\tilde{c}_k)]$, where γ is a positive constant.

Writing a value r using query (or address) q into the memory can be done by updating existing entries:

$$\begin{aligned} \tilde{c}_j &= \text{Match}(k_j, q) && (18) \\ c &= \text{Normalize}(\tilde{c}) && (19) \\ v_j &= \text{Update}(r, v_j, c_j) && (20) \end{aligned}$$

Function $\text{Update}(r, v, c)$ may be as simple as $cr + (1 - c)v$.

If the query is distant from all keys, the memory may allocate a new entry whose key is q and corresponding value is r. The γ constant in the example Normalize function above may serve as a threshold for acceptable key-query divergence.

One can view each entry as representing the state of an entity in the world. In the above example of the bottle, the world model may contain keys $k_{\text{bottle}}, k_{\text{kitchen}}, k_{\text{dining-room}}$ respectively representing the bottle, the kitchen and the dining room. The initial value of v_{bottle} encodes its location as "kitchen", the inital value of v_{kitchen} encodes its content as including the bottle, and the initial value of $v_{\text{dining-room}}$ encodes its content as not including the bottle. After the event, the location and contents are updated.

All of these operations can be done in a differentiable manner, and would hence allow to back-propagate gradients through them.

4.10 Data Streams

Much knowledge about the world is learnable through pure observation. The laws of motion of physical objects can, in principle, be derived from observation, without a need for intervention. But training a world model *efficiently* may require more active or "agentive" information gathering.

One can list five modes of information gathering with which an agent can learn about how the world works:

1. **passive observation**: the agent is being fed a sensor stream (e.g. video, audio, etc)

2. **active foveation**: the agent is being fed a stream within which the focus of attention can be directed without affecting the environment. For example, watching a scene while being able to orient the vision and sound sensors, or being being fed a wide-angle, high resolution video and/or audio stream within which the focus of attention can be directed.

3. **passive agency**: sensory streams in which another agent acting on the environment is being observed, enabling the inference of causal effects of agent actions on the state of the environment.

4. **active egomotion**: the agent receives sensory streams from a real or virtual environment within which the position of the sensors can be modified without significantly affecting the environment. This may include steerable active sensors (e.g. range sensors, heat sensors, chemical sensors) as well as touch sensors.

5. **active agency**: sensory streams that are influenced by the agent's actions. This enables the establishment of causal models in which the agent can learn to predict the consequences of its actions. This mode brings the exploration-exploitation dilemma to the forefront.

In a complex environment, it may not be practical to collect enough passive data for the world model to capture a sufficient portion of the environment's behavior. Mode 2, 4, and 5 allow the agent to collect information that maximizes its understanding of the environment. But to do so may require intrinsic motivation modules that drive attention, curiosity, and exploration into corners of the state space in which the world model's prediction are currently inexact or uncertain.

The main open question is how much can be learned using passive observation (modes 1, 2, 4), how much requires egomotion (mode 3), and how much requires full agency (mode 5).

5 Designing and Training the Actor

The role of the actor module is threefold:

1. inferring optimal action sequences that minimize the cost, given the predictions produced by the world model for Mode-2 actions.

2. producing multiple configurations of latent variables that represent the portion of the world state the agent does not know.

3. training policy networks for producing Mode-1 actions.

There is no conceptual difference between an action and a latent variable. The configurations of both sets of variables must be explored by the actor. For latent variables, configurations must be explored to plan under uncertainty. For action variables configurations must be explored to produce an optimal one that minimizes the cost. In adversarial scenarios (such as games), the latent configurations must be explored that *maximize* the cost. In effect, the actor plays the role of an optimizer and explorer.

When the world model and the cost are well-behaved, the actor module can use a gradient-based optimization process to infer an optimal action sequence. To do so, it receives estimates of the gradient of the cost computed by backpropagating gradients through the cost and the unfolded world model. It uses those estimates to update the action sequence.

When the world model or the cost are not so well-behaved, a gradient-based search for an optimal action sequence may fail. In this case another search/planning method may be applied. If the action space is discrete or can be discretized, one can use dynamic programming methods or approximate dynamic programming methods such as beam search or Monte-Carlo tree search. In effect, any planning method developed in the context of optimal control, robotic, or "classical" AI may be used in this context.

Once an optimal action sequence is obtained through the planning / inference / optimization process, one can use the actions as targets to train a policy network. The policy network may subsequently be used to act quickly, or merely to initialize the proposed action sequence to a good starting point before the optimization phase. Multiple policy networks can be trained for multiple tasks.

The actor also produces configurations of latent variables. These latent variables represent the portion of the world state that the agent does not know. Ideally, the actor would systematically explore likely configurations of the latents. Ideally, the regularizer for the latents, $R1$ and $R2$ in Figure 17, would represent log-priors from which the latent could be sampled. But in a similar way as the policy network, one may devise a latent amortized inference module that learns distributions of latent variables. Good distributions would produce predictions that are plausible. The distribution primate's may depend on all the variables available at that time.

6 Designing the Configurator

The configurator is the main controller of the agent. It takes input from all other modules and modulates their parameters and connection graphs. The modulation can route signals, activate sub-networks, focus attention, etc. In a scenario in which the predictor and the upper layers of the perception encoder are transformer blocks, the configurator outputs may constitute extra input tokens to these transformer blocks, thereby modulating their connection graphs and functions.

The configurator module is necessary for two reasons: hardware reuse, and knowledge sharing. There is an obvious advantage to be able to reuse the same circuit for multiple tasks, particularly if the tasks can be accomplished sequentially, and if the amount of resources (e.g. parameter memory) is limited. But there is another advantage: knowledge reuse. A reasonable hypothesis is that a world model trained for a given environment can be used for a range of different tasks with minor changes. One can imagine a "generic" world model for the environment with a small portion of the parameters being modulated by the configurator for the task at hand. This would be more data efficient and computationally efficient than having separate world models for each skill. The disadvantage is that the agent can only accomplish one task at a time.

The configurator may prime the perception module for a particular task by modulating the parameters at various levels. The human perceptual system can be primed for a particular task, such as detecting an item in a cluttered drawer, detecting fruits or preys in a forest, reading, counting certain events, assembling two parts, etc. For tasks that require a rapid detection of simple motifs, the configurator may modulate the weights of low-level layers in a convolutional architecture. For tasks that involve satisfying relationships between objects (such as assembling two parts with screws) the configuration may be performed by modulating tokens in high-level transformer modules.

The predictor part of the world model must be able to perform a wide range of functions depending on the task at hand. For predictors performing short-term predictions at a low level of abstraction, configuration may mean dynamic signal routing. In a low-level retinotopic feature array representation, prediction may be reduced to local displacements of individual feature vectors, accompanied with small transformations of those vectors. This may be advantageously implemented with local gating/routing circuits. For longer-term prediction at higher-levels of abstraction, it may be preferable to use a transformer architecture. Transformer blocks are particularly appropriate for object-based reasoning in which objects interact. The reason is that the function of transformer blocks is equivariant to permutation. Thanks to that property, one does not need to worry about which object is assigned to which input token: the result will be identical and consistent with the input assignment. Recent work in model-based robotics have proposed to use a transformer operating at the level of an entire trajectory, imposing constraints on the attention circuits to configure the predictor for causal prediction or other tasks (Janner et al., 2021).

Conveniently, the function of a transformer block is easy to configure by adding extra input tokens. Those extra inputs have the effect of modulating the connection graph used by the rest of the network, thereby allowing the specification of a wide range of input-output functions.

Perhaps the most important function of the configurator is to set subgoals for the agent and to configure the cost module for this subgoal. As mentioned in Section 3.2, a simple way to make the cost configurable is by modulating the weights of a linear combination of elementary cost sub-modules. This may be appropriate for the immutable Intrinsic Cost submodule: allowing for a complex modulation of the Intrinsic Cost may make the basic drives of the agent difficult to control, including cost terms that implement safety guardrails. In contrast, one can imagine more sophisticated architectures allowing the Trainable Critic part of the cost to be flexibly modulated. As with the predictor, if the high-level cost is formulated as a set of desired relationships between objects ("is the nut set on the screw?") one may use a transformer architecture trained to measure to what extent the state of the world diverges from the condition to be satisfied. As with the predictor, extra token inputs can be used to modulate the function.

One question that is left unanswered is how the configurator can learn to decompose a complex task into a sequence of subgoals that can individually be accomplished by the agent. I shall leave this question open for future investigation.

7 Related Work

Most of the ideas presented in the paper are not new, and have been discussed at length in various forms in cognitive science, neuroscience, optimal control, robotics, AI, and machine learning, particularly in reinforcement learning.

Perhaps the main original contributions of the paper reside in

- an overall cognitive architecture in which all modules are differentiable and many of them are trainable.
- H-JEPA: a non-generative hierarchical architecture for predictive world models that learn representations at multiple levels of abstraction and multiple time scales.
- a family of non-contrastive self-supervised learning paradigm that produces representations that are simultaneously informative and predictable.
- A way to use H-JEPA as the basis of predictive world models for hierarchical planning under uncertainty.

Below is an attempt to connect the present proposal with relevant prior work. Given the scope of the proposal, the references cannot possibly be exhaustive.

7.1 Trained World Models, Model-Predictive Control, Hierarchical PLanning

The use of models in optimal control goes back to the early days with the Kelley-Bryson method (see (Bryson and Ho, 1969) and reference therein, or the review (Morari and Lee, 1997)). Some methods allowed for online system identification (Richalet et al., 1978).

Using neural networks to learn models for control is an old idea, going back to the early 1990s (Jordan and Rumelhart, 1992; Narendra and Parthasarathy, 1990; Miller et al., 1995).

In the context of optimal control, learning a Mode-1 like policy network is known as direct inverse control.

The idea of Mode-2 style inference over actions using a predictive model in the context of reinforcement learning is also an old idea with, for example, Sutton's Dyna architecture (Sutton, 1991). See (Bertsekas, 2019) for an extensive review.

The idea of learnable models has recently enjoyed a renewal of interest in various contexts (Ha and Schmidhuber, 2018b; Ha and Schmidhuber, 2018a; Hafner et al., 2018; Hafner et al., 2020) (see (Moerland et al., 2020) for a recent survey of model-based reinforcement learning).

Learning world models is particularly important in the context of robotics, especially for grasping and manipulation where sample efficiency is paramount and simulation is often inaccurate. In fact, because classical reinforcement learning approaches require too many trials for real-world applications, interesting advances in learned model for control have emerged from ML-based robotics research (Agrawal et al., 2016; Finn and Levine, 2017; Chua et al., 2018; Srinivas et al., 2018; Yu et al., 2020; Yarats et al., 2021). For a recent review, see (Levine, 2021) and references therein.

A difficult setting is one in which the main input is visual, and a world model must be learned from video. Early attempts to train predictive models without latent variables from simple video produced blurry predictions (Lerer et al., 2016). To handle uncertainty in the predictions, one can use various flavors of latent variable models such as generative adversarial networks (GAN) (Goodfellow et al., 2014), variational auto-encoders (VAE) (Kingma and Welling, 2013) vector-quantized VAE (VQ-VAE) (van den Oord et al., 2017). Variations of these methods have been applied to video prediction and help represent multi-modal outputs and reduce blurriness using GAN (Mathieu et al., 2015; Luc et al., 2020), VAE (Babaeizadeh et al., 2017; Denton and Fergus, 2018; Henaff et al., 2019), or VQ-VAE (Walker et al., 2021). Although many of these methods have not been applied to control problems, some have been applied to vehicle trajectory prediction for autonomous driving (Henaff et al., 2019; Mercat et al., 2020), or various robot control tasks (Oh et al., 2015; Fragkiadaki et al., 2015; Agrawal et al., 2016; Finn et al., 2016; Nagabandi et al., 2017; Babaeizadeh et al., 2017; Srinivas et al., 2018). Unlike the proposed JEPA, these models are generative. The key issue of how to represent uncertainty in the prediction remains.

The alternative to regularized latent-variable models is constrative methods, such as Contrastive Predictive Coding (CPC) (Hénaff et al., 2019), which has been applied to learning visual representations through video prediction (van den Oord et al., 2018).

To solve the multi-modality/blurriness problem, other works have proposed to perform video prediction in representations spaces. In some works, the representation space is obtained from a vision pipeline that has been trained in supervised mode, for example to perform semantic segmentation (Luc et al., 2017; Luc et al., 2018). Unfortunately, the requirement for a pre-trained vision pipeline reduces the general usability of these methods for learning world models by observation.

In the same spirit as JEPA, there have been proposals for automatically learning representations of video frames so they can be easily predicted. These proposals are generally limited to learning low-level features, and often use reconstruction through a decoder as a way to prevent collapse (Goroshin et al., 2015a; Srivastava et al., 2015). Some authors have proposed to use temporal invariance (or consistency) to separate the content of an image region from its instantiation parameters (Wiskott and Sejnowski, 2002; Gregor and LeCun, 2010a; Goroshin et al., 2015b).

At least one recent work has applied non-contrastive SSL methods to a joint embedding architecture for robotics control with some success (Pari et al., 2021; **?**).

Contrastive methods applied to joint-embedding and prediction have been applied successfully to speech recognition (Baevski et al., 2020) (see (Mohamed et al., 2022) for a recent review of SSL to speech).

To perform state trajectory predictions, recent works have advocated the use of transformers, as proposed in the present paper. Transformers are ideal to represent the dynamics of discrete objects in interaction, and have successfully been applied to the prediction of car trajectories (Mercat et al., 2020).

An interesting proposal is the trajectory transformer architecture in which a transformer is fed with the sequence of predicted states over an entire episode (Janner et al., 2021). The pattern of attention can be constrained so as to force the system to only attend to the past so it can be operated in a causal manner (without looking at the future), and trained to predict the next state, actions, and cost from previously observed or predicted states, actions, and costs.

Hierarchical planning is a largely unsolved problem. Wayne and Abbott proposed an architecture that uses a stack of trained forward models that specify intermediate goals for the lower layers (Wayne and Abbott, 2014). Some recent works specify intermediate goals for robots in terms of pose parameters (Gehring et al., 2021). A more recent proposal is the Director system (Hafner et al., 2022) which contains a hierarchical world model and planning architecture trained end-to-end through reinforcement learning.

The idea of intrinsic motivation to train an agent has been studied in the context of robotics (Gottlieb et al., 2013). The presence of an Intrinsic Cost provides a differentiable and efficient way to direct the agent to follow certain behaviors and to learn certain skills.

7.2 Energy-Based Models and Joint-Embedding Architectures

For many authors, Energy-Based Model (EBM) designates a probabilistic model whose distribution is the normalized negative exponential of an energy function.

In this paper, EBM designates a much broader category of models that treat the energy function as fundamental, and directly manipulate its landscape through learning. Many methods have been proposed in the past that directly manipulate the energy. In fact, all traditional optimization-based learning methods can be interpreted as energy-based methods (LeCun et al., 2006). In particular, discriminative training methods for structure prediction problems can be formulated as EBM (LeCun et al., 1998; LeCun et al., 2006).

Most EBM approaches for unsupervised or self-supervised learning have been contrastive. The earliest example is the Boltzmann Machine (Hinton and Sejnowski, 1983), which is a probabilistic generative energy-based model trained contrastively.

Joint Embedding Architectures (JEA) trained with contrastive methods and mutual information maximization methods have a long history. The first non-contrastive JEA was (Becker and Hinton, 1992) which was based on maximizing a measure of mutual information between the representations from two branches seeing to different views of the same scene. Perhaps the first contrastive method for JEA is the so-called "Siamese Network" (Bromley et al., 1994). This was trained contrastively for the purpose of verifying signatures handwritten on a pen tablet.

The idea of JEA remained largely untouched for over a decade, until it was revived in a series of papers from my group (Chopra et al., 2005; Hadsell et al., 2006), and Geoffrey Hinton's group (Goldberger et al., 2005). Following the rebirth of deep learning, a few papers used JEA for fine-grained recognition, including face recognition (Taigman et al., 2014).

With the emergence of SSL approaches, the use of JEA trained contrastively has exploded in the last few years with methods such as PIRL (Misra and Maaten, 2020), MoCo and MoCo-v2 (He et al., 2020; Chen et al., 2020b), and SimCLR (Chen et al., 2020a).

Some methods can be seen as "distillation" approaches in which one branch of the Siamese network is a teacher whose output are used as targets for the other branch. This includes methods in which the output vectors are quantized to discrete cluster prototypes (see (Caron et al., 2020) and predecessors).

In recent years, a number of new *non-contrastive* methods have appeared, such as BYOL (Grill et al., 2020). But the class of non-contrastive methods advocated in the present proposal prevent collapse by maximizing the information content of the embeddings. This includes Barlow Twins (Zbontar et al., 2021), VICReg (Bardes et al., 2021), whitening-based methods (Ermolov et al., 2021), and Maximum Coding Rate Reduction methods (see (Dai et al., 2022) and references therein).

7.3 Human and animal cognition

The limitations of current approaches to machine learning when compared with human learning are obvious (Lake et al., 2017a; Zaadnoordijk et al., 2022).

Young children quickly learn abstract concepts (Murphy, 2002), and models that allow them to navigate, to form goals, and to plan complex action sequences to fulfill them (Gopnik and Meltzoff, 1997; Spelke and Kinzler, 2007; Carey, 2009; Gopnik et al., 2001).

In cognitive science, the idea that the brain builds predictive world models is a common one, and have inspired attempt to reproduce the process in machines (Lake et al., 2017b; Orhan et al., 2020). Some efforts have been devoted to building video datasets to test intuitive physics common sense in machines and infants (Riochet et al., 2019).

The ability to plan is a well-studied feature of human intelligence (Mattar and Lengyel, 2022). There is evidence that people construct simplified representations of the world for planning in which irrelevant details are abstracted away (Ho et al., 2022)

Consciousness is a rather speculative topic, owing to the difficulty of defining what consciousness is. I will not speculate about whether some version of the proposed architecture could possess a property assimilable to consciousness, but will only mention the work of Dehaene and collaborators who have proposed two types of consciousness that they call C1 and C2. C1 is largely related with the modulation of attention, while C2 requires a self-monitoring ability, perhaps assimilable to what the configurator module needs to do in the present proposal (Dehaene et al., 2021).

8 Discussion, Limitations, Broader Relevance

Constructing the cognitive architecture of the present proposal, instantiating all the details, and making the system work for non-trivial tasks will not be an easy task. The path to success is likely riddled with unforeseen obstacles. It will probably take many years to work them all out.

8.1 What is missing from the Proposed Model?

A lot of hard work needs to be done to instantiate the proposed architecture and turn it into a functional system. There may be flaws and pitfalls that may appear to be unsolvable within the specifications of the proposed architecture.

The first question is whether a Hierarchical JEPA can be built and trained from videos. Could it learn the type of abstract concept hierarchy mentioned in section 4.1?

One somewhat open question relative to the JEPA is how precisely to regularize the latent variable so as to minimize its information content. A number of possible mechanisms are proposed: making the latent variable discrete, low-dimensional, sparse, or stochastic. But it is not clear which approach will ultimately be the best.

The current proposal does not prescribe a particular way for the actor to infer latent variable instantiations and optimal action sequences. While the differentiability of all the modules makes it possible *in principle* to use gradient-based optimization to infer optimal action sequences, the optimization problem may be very difficult in practice. In particular, when the action space is discrete, or when the function from actions to cost is highly non smooth, gradient-based method may be ineffective, requiring to use other (gradient-free) search methods (dynamic programming, belief propagation, MCTS, SAT, etc).

Instantiating multiple configurations of latent variables in Mode-2 planning/reasoning may require additional mechanisms not described in the present proposal. Humans seem to be endowed with an ability to spontaneously cycle through alternative interpretations of a percept, as demonstrated by the Necker cube and other visual illusions that have several equally-plausible interpretations. In the context of the present model, different interpretation of an ambiguous percept may be represented by different values of a latent variable. While one could imagine a number of exploratory mechanisms to systematically explore the space of possible latent variable values, no such mechanism is described here.

The present proposal does not specify the details of the architecture of the various modules. For example, it is probable that the predictor will require some sort of dynamic routing and gating circuits in its micro-architecture. Predictors for low-level representation may have to be specialized to represent the kind of small transformations of the representation that can occur in the short term. Predictor modules dealing with higher level representations may require more generic architectures that manipulate objects and their relationships. But none of this is specified in the present proposal.

Similarly, the precise architecture and function of the short-term memory and how it may be used to represent beliefs about the state of the world are somewhat fuzzy. The original Memory Network system and its successors contained the idea that a neural net could use an associative memory as a *working memory* to store and retrieve beliefs about the state of the world between compute cycles (Bordes et al., 2015; Sukhbaatar et al., 2015). But getting such an architecture to work for complex planning and control may prove difficult.

Of all the least understood aspects of the current proposal, the configurator module is the most mysterious. In particular, while planning a complex task, the configurator is supposed to identify sequences of subgoals and configure the agent to successively accomplish those subgoals. Precisely how to do that is not specified.

This is merely a list of foreseeable questions, but many questions and problems will inevitably surface as instances of the proposed systems are put together.

8.2 Broader Relevance of the Proposed Approach

Although the proposed architecture is not specifically designed to model autonomous intelligence, reasoning, and learning in humans and other animals, one can draw some parallels.

The following is somewhat speculative and provided as a way to connect some concepts in cognitive science and neuroscience that have inspired the present work.

8.2.1 Could this Architecture be the Basis of a Model of Animal Intelligence?

Many of the modules in the proposed architecture have counterparts in the mammalian brain that perform similar functions.

The perception module corresponds to the visual, auditory, and other sensory areas of the cortex, as well as some of the association areas. The world model and the critic correspond to various part of the prefrontal cortex. The intrinsic cost module corresponds to structures in the basal ganglia involved in rewards, including the amygdala. The trainable critic may correspond to part of the prefrontal cortex involved in reward prediction. The function of the short-term memory overlaps with what is known of the hippocampus. The configurator may correspond to structures in the prefrontal cortex that perform executive control and modulate attention. The actor regroups areas in the pre-motor cortex that elaborate and encode motor plans.

The idea of predictive world model has long been a prominent concept in cognitive science, and the idea of predictive coding has been a prominent concept in neuroscience. The JEPA architecture and the corresponding non-sample-contrastive self-supervised learning method are somewhat consistent with ideas of predictive coding and efficient coding.

The proposed architecture has a single world model engine that can be configured for the task at hand by the configurator. I have argued that this may not only confer a computational advantage through hardware reuse, but also allow knowledge to be shared across multiple tasks. The hypothesis of a single, configurable world model engine in the human brain may explain why humans can essentially perform a single "conscious" reasoning and planning task at a time. A highly-speculative idea is that the illusion of consciouness may be a side-effect of a configurator-like module in the brain that oversees the function of the rest of brain and configures it for the task at hand. Perhaps if the brain were large enough to contain many independent, non-configurable world models, a configurator would be unnecessary, and the illusion of consciousness would disappear.

What is the substrate of emotions in animals and humans? Instantaneous emotions (e.g. pain, pleasure, hunger, etc) may be the result of brain structures that play a role similar to the Intrinsic Cost module in the proposed architecture. Other emotions such as fear or elation may be the result of *anticipation of outcome* by brain structures whose function is similar to the Trainable Critic.

The presence of a cost module that drives the behavior of the agent by searching for optimal actions suggests that autonomous intelligent agents of the type proposed here will inevitably possess the equivalent of emotions. In an analogous way to animal and humans, machine emotions will be the product of an intrinsic cost, or the anticipation of outcomes from a trainable critic.

8.2.2 Could this be a Path towards Machine Common Sense?

It is a widely-held opinion that none of the current AI systems possess any level of common sense, even at the level that can be observed in a house cat. Animals seem to be able to acquire enough background knowledge about how the world works to exhibit some level of common sense. By contrast, AI systems, even when (pre-)trained with self-supervised mode (e.g. from text) seem to exhibit very limited levels of common sense, making them somewhat brittle.

For example, Large language models (LLMs) seem to possess a surprisingly large amount of background knowledge extracted from written text. But much of human common-sense knowledge is not represented in any text and results from our interaction with the physical world. Because LLMs have no direct experience with an underlying reality, the type of common-sense knowledge they exhibit is very shallow and can be disconnected from reality.

A possible characterization of common sense is the ability to use models of the world to fill in blanks, for example predicting the future, or more generally filling in information about the world that is unavailable from perception or from memory. With this definition, common sense is an ability that emerges from a collection of models of the world or from a single model engine configurable to handle the situation at hand. This view of common sense sits squarely in the camp of "grounded intelligence": common sense is a collection of models from low-levels of abstraction to high levels, all the way up to knowledge acquired through language.

Could SSL applied to configurable H-JEPA constitute the substrate of machine common sense? Could a properly-trained and configured H-JEPA embed enough predictive knowledge and capture enough dependencies about the world to exhibit some level of common sense?

I speculate that common sense may emerge from learning world models that capture the self-consistency and mutual dependencies of observations in the world, allowing an agent to fill in missing information and detect violations of its world model.

8.3 Is it all about scaling? Is reward really enough?

The section reviews a few potential paths towards human-level intelligence that have been proposed in recent years. The surprising power of large transformer architectures trained to predict text and other modalities have led some to claim that we merely need to scale up those models (Brown et al., 2020; Brown et al., 2020). The surprising power of reinforcement learning for games and other simple environments have led other to claim that reward is enough (Silver et al., 2021). finally, the limitations of current deep-learning systems when it comes to reasoning have led some to claim that deep learning systems need to be augmented by hard-wired circuitry to enable symbol manipulation (Marcus and Davis, 2019)

8.3.1 Scaling is not enough

Large Language Models (LLMs), and more generally, large-scale transformer architectures trained with a form of generative self-supervised learning, have been astonishingly successful at capturing knowledge present in text. This has led to a debate in the AI community as to whether human-level AI can be attained by scaling up these architectures. My position in this debate is that I do not believe that scaling is enough for two main reasons.

First, current models operate on "tokenized" data and are generative. Every input modality must be turned into a sequence (or a collection) of "tokens" encoded as vectors. While this works well for text, which is already a sequence of discrete tokens, it is less suitable for continuous, high dimensional signals such as video. The type of SSL training used for LLM-style models can be seen as a particular kind of latent-free generative model trained with a particular kind of contrastive method called denoising auto-encoder (Vincent et al., 2010), or in this case masked auto-encoder (Devlin et al., 2018). Hence, they are subject to the limitations of generative models, latent-variable free models, and contrastive methods. Generative models have difficulty representing complex uncertainties in continuous spaces. LLMs simplify the representation of uncertainty in the prediction by only dealing with discrete objects from a finite collection (e.g. words from a dictionary). Representing uncertainty about a word being predicted comes down to producing a vector whose components are scores or probabilities for each word (or discrete token) in the dictionary. But this approach doesn't work for high-dimensional continuous modalities, such as video. To represent such data, it is necessary to eliminate irrelevant information about the variable to be modeled through an encoder, as in the JEPA. Furthermore, the high-dimensionality of the signal precludes the representation of uncertainty through a normalized distribution.

Second, current models are only capable of very limited forms of reasoning. The absence of abstract latent variables in these models precludes the exploration of multiple interpretations of a percept and the search for optimal courses of action to achieve a goal. In fact, dynamically specifying a goal in such models is essentially impossible.

8.3.2 Reward is not enough

The proposed architecture is designed to minimize the number of actions a system needs to take in the real world to learn a task. It does so by learning a world model that capture as much knowledge about the world as possible *without* taking actions in the world. It uses intrinsic costs that are differentiable functions of measured or predicted world states. This makes the proposal more similar to optimal control than to reinforcement learning. In the proposed model, much of learning takes place at the level of the world model (perceptual encoder and predictor). In this context, what is the role of reinforcement learning (RL)?

In most RL settings the reward (or the cost, which is a negative reward) is fed to the agent by the environment. In other words, Intrinsic Cost module is the environment itself, and is therefore an unknown function. The value of the function can be probed by observing the state of the world, taking an action, and observing the resulting reward. The gradient of the reward with respect to the action or the state is unknown and must be estimated by multiple action trials as in policy gradient methods. In Actor-Critic methods, the reward function is approximated by a critic module that is trained to approximate expected future values of the reward. The critic provides a differentiable approximation of the reward function.

But model-free RL is extremely sample-inefficient, at least when compared with human and animal learning, requiring very large numbers of trials to learn a skill. Scalar rewards provide low-information feedback to a learning system. As a consequence, a pure RL system requires a very large number of trials to learn even relatively simple tasks. Model-based RL clearly has the potential of being considerably more sample efficient. But the question becomes how to train the world model: is it trained from taking actions and getting rewards, or is it trained by predicting the world state? In the latter case, *reward is clearly not enough*: most of the parameters in the systems are trained to predict large amounts of observations in the world. Contrary to the title of a recent position paper by Silver et al. (Silver et al., 2021), the reward plays a relatively minor role in this scenario.

8.3.3 Do We Need Symbols for Reasoning?

In the proposed architecture, reasoning comes down to energy minimization or constraint satisfaction by the actor using various search methods to find a suitable combination of actions and latent variables, as stated in Section 3.1.4.

If the actions and latent variables are continuous, and if the predictor and the cost modules are differentiable and relatively well behaved, one can use gradient-based methods to perform the search. But there may be situations where the predictor output changes quickly as a function of the action, and where the action space is essentially discontinuous. This is likely to occur at high levels of abstractions where choices are more likely to be qualitative. A high-level decision for a self-driving car may correspond to "turning left or right at the fork", while the low-level version would be a sequence of wheel angles.

If the action space is discrete with low cardinality, the actor may use exhaustive search methods. If the action set cardinality, and hence the branching factor, are too large, the actor may have to resort to heuristic search methods, including Monte-Carlo Tree Search, or other gradient-free methods. If the cost function satisfied Bellman's equations, one may use dynamic programming.

But the efficiency advantage of gradient-based search methods over gradient-free search methods motivates us to find ways for the world-model training procedure to find hierarchical representations with which the planning/reasoning problem constitutes a continuous relaxation of an otherwise discrete problem.

A remain question is whether the type of reasoning proposed here can encompass all forms of reasoning that humans and animals are capable of.

Acknowledgments

Ideas in this paper are a distillation of years of interactions with many people. It is impossible to list them all here.

I will list some of the more salient ones: Léon Bottou, Yoshua Bengio, Geoffrey Hinton, Rob Fergus, and Jitendra Malick; Emmanuel Dupoux and Stanislas Dehaene; Mikael Henaff, Alfredo Canziani, and Nicolas Carion at NYU; Li Jing, Yubei Chen, Randall Balestriero, and Stéphane Deny at FAIR; Vlad Sobal, Jiachen Zhu, and Katrina Evtimova at NYU.

I am thankful to the following colleagues for their comments on the manuscript: Olivier Delalleau, Gus Xia, Yoshua Bengio, and Emmanuel Dupoux.

References

Agrawal, P., Nair, A., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419.

Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. (2017). Stochastic variational video prediction. *CoRR*, abs/1710.11252.

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460. Curran Associates, Inc.

Bardes, A., Ponce, J., and LeCun, Y. (2021). Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations (ICLR 2022)*. arXiv preprint arXiv:2105.04906.

Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., et al. (2016). Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29.

Becker, S. and Hinton, G. E. (1992). Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163.

Bertsekas, D. (2019). *Reinforcement learning and optimal control*. Athena Scientific.

Bordes, A., Usunier, N., Chopra, S., and Weston, J. (2015). Large-scale simple question answering with memory networks. *arXiv:1506.02075*.

Bromley, J., Guyon, I., LeCun, Y., Sackinger, E., and Shah, R. (1994). Signature verification using a "siamese" time delay neural network. In *NeurIPS*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Bryson, A. and Ho, Y. (1969). *Applied optimal control*. Blaisdell, Waltham, MA.

Carey, S. (2009). *The Origin of Concepts*. Oxford University Press, New York, New York, USA.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *16th European Conference, Glasgow, UK (ECCV 2020)*, page 213–229.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems*.

Carreira-Perpiñán, M. A. and Hinton, G. (2005). On contrastive divergence learning. In Cowell, R. G. and Ghahramani, Z., editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, volume R5 of *Proceedings of Machine Learning Research*, pages 33–40. PMLR. Reissued by PMLR on 30 March 2021.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. (2020a). Big self-supervised models are strong semi-supervised learners. In *NeurIPS*.

Chen, X., Fan, H., Girshick, R., and He, K. (2020b). Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*.

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114.

Craik, K. J. W. (1943). *The nature of explanation.* University Press, Macmillan.

Dai, X., Tong, S., Li, M., Wu, Z., Psenka, M., Chan, K. H. R., Zhai, P., Yu, Y., Yuan, X., Shum, H.-Y., and Ma, Y. (2022). Ctrl: Closed-loop transcription to an ldr via minimaxing rate reduction. *Entropy*, 24(4):456.

Dehaene, S., Lau, H., and Kouider, S. (2021). What is consciousness, and could machines have it? *Robotics, AI, and Humanity*, pages 43–56.

Denton, E. and Fergus, R. (2018). Stochastic video generation with a learned prior. *arXiv preprint arXiv 1802.07687*.

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

Doi, E., Balcan, D. C., and Lewicki, M. S. (2007). Robust coding over noisy overcomplete channels. *IEEE Transactions on Image Processing*, 16(2):442–452.

Ermolov, A., Siarohin, A., Sangineto, E., and Sebe, N. (2021). Whitening for self-supervised representation learning.

Evtimova, K. and LeCun, Y. (2022). Sparse coding with multi-layer decoders using variance regularization. *arXiv:2112.09214*.

Finn, C., Goodfellow, I. J., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. *CoRR*, abs/1605.07157.

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE.

Fragkiadaki, K., Agrawal, P., Levine, S., and Malik, J. (2015). Learning visual predictive models of physics for playing billiards. *CoRR*, abs/1511.07404.

Gehring, J., Synnaeve, G., Krause, A., and Usunier, N. (2021). Hierarchical skills for efficient exploration. *Advances in Neural Information Processing Systems*, 34:11553–11564.

Goldberger, J., S.Roweis, Hinton, G., and Salakhutdinov, R. (2005). Neighbourhood components analysis. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, Cambridge, MA.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Gopnik, A. and Meltzoff, A. N. (1997). *Words, Thoughts, and Theories.* MIT Press, Cambridge, MA.

Gopnik, A., Meltzoff, A. N., and Kuhl, P. K. (2001). *The Scientist in the Crib: What Early Learning Tells Us About the Mind.* Perennial, New York, NY.

Goroshin, R., Bruna, J., Tompson, J., Eigen, D., and LeCun, Y. (2015a). Unsupervised feature learning from temporal data. In *International Conference on Computer Vision (ICCV 2015).*

Goroshin, R., Mathieu, M., and LeCun, Y. (2015b). Learning to linearize under uncertainty. In *Advances in Neural Information Processing Systems (NIPS 2015)*, volume 28.

Gottlieb, J., Oudeyer, P. Y., Lopes, M., and Baranes, A. (2013). Information-seeking, curiosity, and attention: Computational and neural mechanisms. *Trends in Cognitive Sciences*, 17:585–593.

Gregor, K. and LeCun, Y. (2010a). Emergence of complex-like cells in a temporal product network with local receptive fields. *arXiv preprint arXiv:1006.0448.*

Gregor, K. and LeCun, Y. (2010b). Learning fast approximations of sparse coding. In *Proc. International Conference on Machine learning (ICML'10).*

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS.*

Ha, D. and Schmidhuber, J. (2018a). Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31.

Ha, D. and Schmidhuber, J. (2018b). World models. *arXiv preprint arXiv:1803.10122*.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *CVPR*.

Hafner, D., Lee, K.-H., Fischer, I., and Abbeel, P. (2022). Deep hierarchical planning from pixels. *arXiv preprint arXiv:2206.04114*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning latent dynamics for planning from pixels. *arXiv 1811.04551*.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *CVPR*.

Henaff, M., Canziani, A., and LeCun, Y. (2019). Model-predictive policy learning with uncertainty regularization for driving in dense traffic. In *ICLR-19*. arXiv:1901.02705.

Henaff, M., Weston, J., Szlam, A., Bordes, A., and LeCun, Y. (2017). Tracking the world state with recurrent entity networks. In *International Conference on Learning Representations (ICLR 2017)*.

Hinton, G. and Sejnowski, T. (1983). Optimal perceptual inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 448–453, Washington 1983. IEEE, New York.

Ho, M. K., Abel, D., Correa, C. G., Littman, M. L., Cohen, J. D., and Griffiths, T. L. (2022). People construct simplified mental representations to plan. *Nature*, 606(7912):129–136.

Hénaff, O. J., Srinivas, A., De Fauw, J., Razavi, A., Doersch, C., Eslami, S. M. A., and van den Oord, A. (2019). Data-efficient image recognition with contrastive predictive coding. In *ICML*.

Janner, M., Li, Q., and Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*.

Jing, L., Zbontar, J., et al. (2020). Implicit rank-minimizing autoencoder. *Advances in Neural Information Processing Systems*, 33:14736–14746.

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354.

Kahneman, D. (2011). *Thinking, fast and slow.* Macmillan.

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

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017a). Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017b). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:E253.

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

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. In Bakir, G., Hofman, T., Schölkopf, B., Smola, A., and Taskar, B., editors, *Predicting Structured Data*. MIT Press.

Lerer, A., Gross, S., and Fergus, R. (2016). Learning physical intuition of block towers by example. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 430–438.

Levine, S. (2021). Understanding the world through action. *arXiv:2110.12543*. `https://arxiv.org/abs/2110.12543`.

Luc, P., Clark, A., Dieleman, S., Casas, D. d. L., Doron, Y., Cassirer, A., and Simonyan, K. (2020). Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*.

Luc, P., Couprie, C., Lecun, Y., and Verbeek, J. (2018). Predicting future instance segmentation by forecasting convolutional features. In *Proceedings of the european conference on computer vision (ECCV)*, pages 584–599.

Luc, P., Neverova, N., Couprie, C., Verbeek, J., and LeCun, Y. (2017). Predicting deeper into the future of semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 648–657.

Marcus, G. and Davis, E. (2019). *Rebooting AI: Building artificial intelligence we can trust*. Vintage.

Mathieu, M., Couprie, C., and LeCun, Y. (2015). Deep multi-scale video prediction beyond mean square error. In *ICLR 16*. arXiv preprint arXiv:1511.05440.

Mattar, M. G. and Lengyel, M. (2022). Planning in the brain. *Neuron*, 110(6):914–934.

Mercat, J., Gilles, T., El Zoghby, N., Sandou, G., Beauvois, D., and Gil, G. P. (2020). Multi-head attention for multi-modal joint vehicle motion forecasting. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9638–9644. IEEE.

Miller, A. H., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016). Key-value memory networks for directly reading documents. In *EMNLP-16*.

Miller, W. T., Sutton, R. S., and Werbos, P. J. (1995). *Neural networks for control*. MIT press.

Misra, I. and Maaten, L. v. d. (2020). Self-supervised learning of pretext-invariant representations. In *CVPR*.

Moerland, Thomas, M., Broekens, J., and Jonker, Catholijn, M. (2020). Model-based reinforcement learning: A survey. *arXiv:2006.16712*. `https://arxiv.org/abs/2006.16712`.

Mohamed, A., Lee, H.-y., Borgholt, L., Havtorn, J. D., Edin, J., Igel, C., Kirchhoff, K., Li, S.-W., Livescu, K., Maaløe, L., et al. (2022). Self-supervised speech representation learning: A review. *arXiv preprint arXiv:2205.10643*.

Morari, M. and Lee, J. H. (1997). Model predictive control: Past, present and future. *Computers and Chemical Engineering*, 23:667–682.

Murphy, G. L. (2002). *The Big Book of Concepts*. MIT Press, Cambridge, MA.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2017). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596.

Narendra, K. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks*, 1(1):4–27.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. *Advances in neural information processing systems*, 28.

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

Orhan, E., Gupta, V., and Lake, B. M. (2020). Self-supervised learning through the eyes of a child. *Advances in Neural Information Processing Systems*, 33:9960–9971.

Pari, J., Shafiullah, N. M., Arunachalam, S. P., and Pinto, L. (2021). The surprising effectiveness of representation learning for visual imitation. In *Robotics Science and Systems 2022*. arXiv preprint arXiv:2112.01511.

Richalet, J., Rault, A., Testud, J. L., and Papon, J. (1978). Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14(5):413–428.

Riochet, R., Castro, M. Y., Bernard, M., Lerer, A., Fergus, R., Izard, V., and Dupoux, E. (2019). Intphys: A benchmark for visual intuitive physics reasoning. *arXiv:1803.07616*.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.

Sobal, V., Canziani, A., Carion, N., Cho, K., and LeCun, Y. (2022). Separating the world and ego models for self-driving. *arXiv:2204.07184*.

Spelke, E. S. and Kinzler, K. D. (2007). Core knowledge. *Developmental Science*, 10:89–96.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. (2018). Universal planning networks. *CoRR*, abs/1804.00645.

Srivastava, N., Mansimov, E., and Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 843–852, Lille, France. PMLR.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. *Advances in neural information processing systems*, 28.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017). Neural discrete representation learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).

Walker, J., Razavi, A., and Oord, A. v. d. (2021). Predicting video with vqvae. *arXiv preprint arXiv:2103.01950*.

Wayne, G. and Abbott, L. (2014). Hierarchical control using networks trained with higher-level forward models. *Neural Computation*, 26(10):2163–2193.

Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770.

Yarats, D., Kostrikov, I., and Fergus, R. (2021). Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *ICLR*.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C., and Ma, T. (2020). Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*.

Zaadnoordijk, L., Besold, T., and Cusack, R. (2022). Lessons from infant learning for unsupervised machine learning. *Nature Machine Intelligence*, 4:510–520.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR.

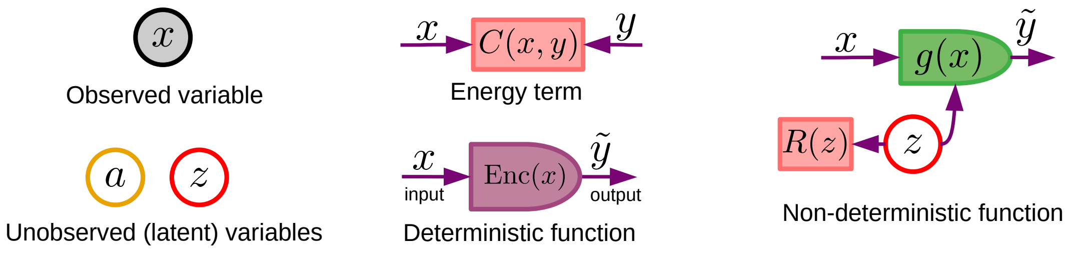


Figure 18: *Symbols used in architectural diagrams.*
Architectural diagrams use symbols commonly used to draw factor graphs – circles for variables, rectangles for factors – plus rounded rectangles to represent deterministic functions. Filled circles represent observed variables, or variables that are outputs of deterministic functions.
Hollow circles represent latent variables, i.e. variables that must be inferred by minimizing some cost, or sampled from a distribution.
Red rectangles represent energy terms. These modules have an implicit scalar output that contributes additively to the total energy of the system. This is similar to the convention used for factor graphs.
Rounded rectangles represent deterministic functions, which may have one or several inputs. Given a set of inputs, the output is assumed to be easily computable and unique. The function is generally assumed to be differentiable. It may contain trainable parameters.
Non-deterministic functions are represented as shown on the right. They are composed of a deterministic function $g(x,z)$ in which one of the inputs is a latent variabl z. The latent variable is seen as varying within a level set of a regularizing energy term $R(z)$. When z varies in the level set $\mathcal{Z}_h = \{z|R(z) < h\}$, the output $\tilde{y}$ will vary over the set $\mathcal{Y}_h = \{y|y = g(x,z), \forall z \in \mathcal{Z}_h\}$.
In some cases, the energy term can be transformed into a probability distribution (see text).

Appendix: Symbols and Notations

Architectural diagrams in this paper use the symbols shown in Figure 18.

We use symbols that are somewhat similar to the representation of factor graphs: circles for variables, rectangles for factors. There are two major differences, however. First, the factors represent additive energy terms, not multiplicative probability factors. Second, we use an additional symbol, rounded rectangles, to represent deterministic functions with a clear directionality from inputs to outputs.

More precisely:

- Filled circles represent observed variables, or variables that are outputs of deterministic functions.
- Hollow circles represent latent variables, i.e. variables that must be inferred by minimizing some cost, varied over a set, or sampled from a distribution.
- Red rectangles represent energy terms. These modules have an implicit scalar output that contributes additively to the total energy of the system.
- Rounded rectangles represent deterministic functions, which may have one or several inputs. Given a set of inputs, the output is assumed to be easily computable and unique. The function is generally assumed to be differentiable. It may contain trainable parameters. Such a module is typically implemented as a deep neural net.

Non-deterministic functions do not have a dedicated symbol, but must be represented as a combination of deterministic functions, energy modules, and latent variables. An example is shown on the right of Figure 18. A non-deterministic function is represented by a deterministic function $\tilde{y} = g(x, z)$ whose output depends on a latent variable z. The latent variable is fed to a regularizing energy term $R(z)$. We first define $\mathcal{Z}_h$ as the level set of z, i.e. the set of z for which $R(z)$ is less than a threshold h:

$$\mathcal{Z}_h = \{z/R(z) < h\} \tag{21}$$

As z varies over $\mathcal{Z}_h$, the output will vary over the set:

$$\mathcal{Y}_h = \{y|y = g(x, z), \forall z \in \mathcal{Z}_h\} \tag{22}$$

In some cases, this setup may be used to represent probability distributions. Forst, the energy term is transformed into a probability distribution using a Gibbs-Boltzmann formula:

$$P(z) = \frac{\exp(-R(z))}{\int_{z'} \exp(-R(z'))} \tag{23}$$

Drawing the latent variable from this distribution implies a distribution over y:

$$P(y|x) = \int_z \delta(y - g(x, z))P(z) \tag{24}$$

where $\delta()$ is the Dirac delta function.

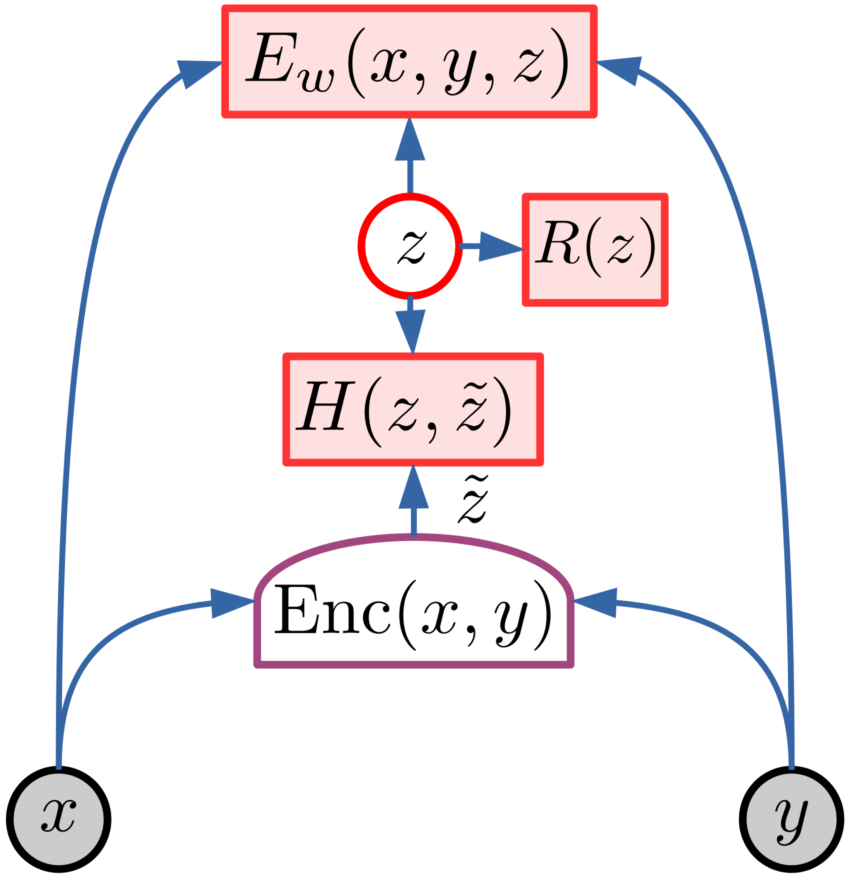


Figure 19: *Amortized Inference with an EBM. An encoder is trained to produce* $\tilde{z} = \mathrm{Enc}(s_x, y)$ *to approximate the value of the latent that minimizes the energy* $\check{z} = \mathrm{argmin}_{z \in \mathcal{Z}} E_w(x, y, z)$*. The regularizer* $R(z)$ *plays the essential role of limiting the information that* z *contains about* y*. This is particularly important here because the system has access to* y *and can "cheat" by carrying the complete information about* y *through the encoder.*

Appendix: Amortized Inference for Latent Variables

Inference in latent variable models consists in performing the optimization $\check{z} = \mathrm{argmin}_{z \in \mathcal{Z}} E_w(x, y, z)$. When z is continuous, this may be best performed through a gradient-based optimization that involves backpropagating gradients through the model dowen to z for multiple iterations. In generative architectures, this may be expensive, requiring to back-propagate through the decoder and the predictor. One way to reduce the cost of inference is to use *amortized inference*. The idea is to train a neural net to predict an approximate solution to the inference optimization problem.

The architecture is depicted in Figure 20. An encoder $\tilde{z} = \mathrm{Enc}(s_x, y)$ is trained to minimize a divergence measure $H(\check{z}, \tilde{z})$ between the encoder output and the optimal latent variable $\check{z} = \mathrm{argmin}_{z \in \mathcal{Z}} E_w(x, y, z)$. Once trained, the prediction $\tilde{z}$ may be use as an estimate of $\check{z}$ or as an initial value for the inference optimization.

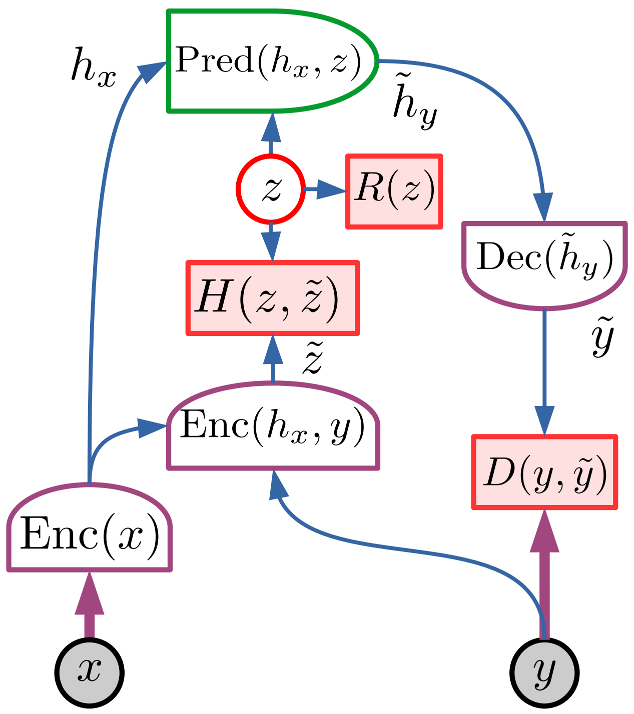


Figure 20: *Amortized Inference with a Regularized Generative Latent-Variable EBM architecture. An encoder is trained to produce* $\tilde{z} = \text{Enc}(s_x, y)$ *to approximate the value* $\check{z}$ *that minimizes the energy. The regularizer* $R(z)$ *plays the essential role of limiting the information that* z *contains about* y*. This is particularly important here because the system has access to* y *and can "cheat" by carrying the complete information about* y *through the encoder.*

The regularizer $R(z)$ is even more important than in the regular inference case because the prediction pathway has access to y and can "cheat" by carrying the complete information about y through the encoder. Without an information-limiting regularizer This would cause a collapse of the energy function, since it would allow any y to be reconstructed perfectly. The regularizer is there to minimize the information that $\check{z}$ may contains about y.

Variational Auto-Encoders, and LISTA-style sparse Auto-Encoders belong to the family of Regularized GLVEBM with amortized inference. Most of those models are *unconditional* and do not possess an x nor an $\text{Enc}(x)$ module.

Appendix: Loss functions for Contrastive Training of EBM

Much can be said about contrastive methods. Table 1 lists a few examples of contrastive methods, together with their strategy for picking contrastive samples $\hat{y}$ and their loss functional.

Rows 1-2 in the table are exact maximum likelihood methods. They assume that the gradient of the log partition function can be computed exactly. Rows 2-4 are approximate maximum likelihood methods. Rows 5-10 are not interpretable within a probabilistic framework.

Row 1: Maximum Conditional Likelihood for discrete y is used whenever the energy needs to be turned into a probability distribution. through a Gibbs formula $P(y|x) = \exp(-F_w(x,y))/\sum_{y' \in \mathcal{Y}} \exp(-F_w(x,y'))$. The loss is the negative log conditional likelihood. This is the prevalent approach when y is a discrete variable within a finite set (e.g. for classification).

Row 2 and 3: Maximum Conditional Likelihood is used for any model that should produce probability estimates. Row 2 only applies to tractable models in which the integral in the contrastive term (or its gradient) can be computed analytically. Row 3 applies to situations where the integral is intractable and its gradient must be approximated by Monte Carlo sampling methods. It then comes down to devising good methods to sample $\hat{y}$ values from the model's Gibbs distribution: $P_w(y|x) = \exp(-\beta F_w(x,y)/\int_{y'} \exp(-\beta F_w(x,y'))$.

Row 4: Contrastive Divergence. MCMC sampling methods for Row 3 may take a long time to mix. One may start from a training sample and let the Markov chain evolve for a short time, and then accept or reject the resulting sample so as to satisfy detailed balance (Carreira-Perpiñán and Hinton, 2005).

Row 5: Pairwise hinge, also known as triplet loss, drives the energy of the correct output to be lower than the energy of the contrastive output by at least a margin $m(y, \hat{y})$ which may grow with a measure of divergence between y and $\hat{y}$. The difficulty is to find suitable contrastive samples whose energy is low and "threatening", a task sometimes called "hard negative mining".

Rows 6-8: Min-hinge, Square-hinge, Square-exp. Assumes that the energy has a lower bound. minimizes the energy of the correct output and pushes the energy of contrastive outputs above a margin equal to $m(y, \hat{y})$ for rows 6 and 7, and infinite for row 8.

Row 8: Logistic. As with the pairwise hinge, the logistic loss maximizes the difference between the energies of the contrastive output and the correct output. Unlike the pairwise hinge, the difference is pushed to infinity, but with a force that decreases quickly.

Row 9: GAN. A GAN differs from other contrastive methods in the way contrastive samples are generated. The contrastive samples are produced by a generator network that is trained preferentially generate samples that have low energy according to the model. In principle, any loss function can be used, as long as it increases with the energy of the correct output, and decreases with the energy of the contrastive sample.

Row 10: Denoising Auto-Encoder. A denoising AE produces contrastive samples by corrupting outputs from training samples. The corruption can be performed by adding noise or by masking parts of the output. The energy function is the reconstruction error, hence the energy is trained to be zero on the data manifold, and to grow with $D(y, \hat{y})$ as $\hat{y}$ moves away from y on the data manifold.

	Method	**Energy**	**$\hat{y}$ Generation**	**Loss**
1	Max Likelihood	discrete y	exhaustive	$F_w(x, y) + \log \sum_{y' \in \mathcal{Y}} \exp(-F_w(x, y'))$
2	Max Likelihood	tractable	exhaustive	$F_w(x, y) + \log \int_{y' \in \mathcal{Y}} \exp(-F_w(x, y'))$
3	Max likelihood	any	MC or MCMC	$F_w(x, y) - F_w(x, \hat{y})$
4	Contr. Divergence	any	trunc'd MCMC	$F_w(x, y) - F_w(x, \hat{y})$
5	Pairwise Hinge	any	most offending	$[F_w(x, y) - F_w(x, \hat{y}) + m(y, \hat{y})]^+$
6	Min-Hinge	positive	most offending	$F_w(x, y) + [m(y, \hat{y}) - F_w(x, \hat{y})]^+$
6	Square-Hinge	divergence	most offending	$F_w(x, y)^2 + \left([m(y, \hat{y}) - F_w(x, \hat{y})]^+\right)^2$
7	Square-Exp	any	most offending	$F_w(x, y)^2 + \exp(-\beta F_w(x, \hat{y}))$
8	Logistic	any	most offending	$\log(1 + \exp(F_w(x, y) - F_w(x, \hat{y}))$
9	GAN	any	$\hat{y} = g_u(z)$	$H(F_w(x, y), F_w(x, \hat{y}), m(y, \hat{y}))$
10	Denoising AE	$D(y, g_w(y))$	$\hat{y} = N(y)$	$D(y, g_w(\hat{y}))$

Table 1: *A of list of contrastive methods and loss functions to train energy-based models. They are all use loss functions with two terms, one that pushes down on the energy of a training sample, and one the pulls up the energies of one or several contrastive samples.*
They differ by the strategy they employ to generate contrastive samples, and by the precise form of the loss function.
Exact or approximate Maximum Likelihood methods (rows 1-4) are used whenever the model needs to produce probability estimates. When the second term is intractable, its gradient may be approximated through Monte-Carlo methods, which can be seen as particular ways to produce $\hat{y}$. Many contrastive self-supervised methods for joint embedding architectures (Siamese nets) use Row 1 (InfoNCE).
A number of contrastive methods (Rows 5-8) are based on finding a $\hat{y}$ that is "highly offending", meaning different from the desired y, yet given a low energy by the model. Pairs of energies for y and $\hat{y}$ are fed to a loss function that pushes the former to low values and the latter to higher values. This can be done with a variety of losses including hinge loss.
GANs (row 9) are contrastive methods in which the contrastive samples are produced by a generator network whose input is a random vector. The generator is trained to produce samples to which the model currently attributes a low energy, but should attribute a high energy.
Denoising Auto-Encoders (row 10) apply a corruption process to training samples to produce contrastive samples $\hat{y} = N(y)$. The energy function is the reconstruction error $F_w(y) = D(y, g_w(y))$ where $D()$ is a symmetric divergence measure and $g_w(y)$ a parameterized function. By training $g_w()$ to map $\hat{y}$ to y, the energy for $\hat{y}$ is trained to be equal to $D(\hat{y}, y)$, while the energy of y is trained to be zero.