Notes on "Backpropamine"

Xavier Garcia

April 28, 2019

1 Introduction

Throughout these notes, we are exclusively interested in tasks which involve a sequence of steps, such as traversing a maze or completing a sentence given the first few words. More formally, we will have three sequences of interest:

1. states $s_t \in \mathcal{S}$
2. actions $a_t \in \mathcal{A}$
3. rewards $r_t \in \mathbb{R}$.

At every time step t, the agent should use the entire history $\mathcal{H}_{t-1}$ up until time t i.e.

$$\mathcal{H}_{t-1} := \{(s_0, a_0, r_0), ..., (s_{t-1}, a_{t-1}, r_{t-1})$$

coupled with the current state s_t to choose an action a_t which leads to a reward r_t and continue this process until some final time T. The agent's goal is to choose as to maximize the average cumulitive reward:

$$R = \mathbb{E} \sum_{t=0}^{T} r_t.$$

We choose expectation because the the states, actions and rewards may contain a probabilisic component.

Traditionally, we impose a Markov assumption which means that the state and future rewards at time t only depend on the history $\mathcal{H}_{t-1}$ through the last entry $(s_{t-1}, a_{t-1}, r_{t-1})$. This is a reasonable assumption for many tasks. For example, to make the optimal play in Chess, all one needs is the current position, not the set of moves that led up to it. Nevertheless, there are many tasks which require the entire history. For example, if the goal was to generate a sentence from the first few words, every time one generates a word, one should be using all the words you've seen so far rather than just the last one. As a way to deal with an evergrowing history, we use recurrent network, which keep a hidden state vector h_t which serves as a proxy for $\mathcal{H}_t$. After each time step, the recurrent network updates the hidden state with the new input at that time

and uses it, as well as the input, to make its prediction. Assuming the set of actions $\mathcal{A}$ is finite, we can model the probability of choosing an action at time t by use of a recurrent network f_w with weights w:

$$\mathbb{P}(a_t = \cdot) = \text{softmax}(F(h_t))$$

where $h_t = f_w(x_t, h_{t-1})$ and F could be a fully-connected layer, or even more complicated if one so wishes. One of the advantages of using a recurrent network is the weight-sharing mechanism. The set of weights remained fixed at every timestep, which can act as a regularization when contrasted with the other extreme of using different sets of weights $w(t)$ at every time step, as well as reducing memory costs.

Nevertheless, h_t can only encode so much information, especially given its fixed dimensionality. Reinforcement learning agents trained to solve multiple tasks sometimes suffer from catastrophic forgetting, which leads to the agent forgetting how to do one of the tasks as it learns a new one, signaling some limit to its memory and some notion of underfitting to all the tasks. Given this underfitting, it might seem that parameter sharing is overly regularizing the network.

2 Time-dependent weights.

Given the problems formulated before, we need to enlarge the predictive power of our network. One approach is to enlarge the hidden state and the other is to make the weights of the network time dependent i.e. $w := w(t)$, allowing them to store information about the past as well. Choosing unrelated weights for each time t would result in a huge growth on the number of weights and over-parametrization. Instead, the paper proposes to have one set of fixed weights, w, and write the time-varying ones as a function of the fixed ones and time, i.e.

$$w(t) = \psi(w, t, h_{t-1}, x_t)$$

for some function ψ where x_t is a function of (s_t, a_t, r_t) such as concatenation. We want ψ to shape the weights depending on the information it has currently seen and the new information at time t, which is why we also pass h_{t-1} and x_t respectively as input. The paper further simplifies ψ by forcing its dependence on w to be strictly a translation by w i.e.

$$\psi(w, t, h_{t-1}, x_t) := w + \alpha \otimes \text{Hebb}(t, h_{t-1}, x_t)$$

for some set of learnable parameters α.

For the moment, we omit the dependencies on h_{t-1} and x_t and write

$$\text{Hebb}(t) := \text{Hebb}(t, h_{t-1}, x_t)$$

similarly for other objects in the following set of equations. With this notation in mind, we restrict the explicit time dependence to satisfy the following form:

$$\begin{aligned} \text{Hebb}(0) &= 0 \\ \text{Hebb}(t+1) &= \text{Clip}(\text{Hebb}(t) + M(t)E(t)) \end{aligned}$$

where $M(t)$ is a scalar, $E(t)$ has the same dimension as Hebb(t) and Clip is the following function applied elementwise:

$$\text{Clip}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } x \in [-1, 1] \\ 1 & \text{else} \end{cases}$$

The clipping is done to mitigate some of the instability and to allow the weights themselves to be the dominant force. The $M(t)$ will be a scalar that the network will compute at every time step t, in addition to the action chosen at that time.

The exact formulation for $E(t)$ actually varies depending on the type of recurrent network. To remain faithful to the paper, we present their formulation for the case when f_w is an LSTM. Recall the update equations for an LSTM:

$$\begin{aligned} i_t &= \tanh(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\ j_t &= \tanh(W_{jx}x_t + W_{jh}h_{t-1} + b_j) \\ f_t &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\ o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\ c_t &= h_{t-1} \otimes f_t + i_t \otimes j_t \\ h_t &= \tanh(c_t) \otimes o_t \end{aligned}$$

Here, i_t is the injection of the new data x_t into the hidden state and the rest are the results of gates to control the data. For our purposes, we will write $E(t)$ as a function of h_{t-1} and i_t, i.e.

$$E(t) := E(t, h_{t-1}, i_t).$$

Restricting ourselves to smooth E, we can take a Taylor series around $(h_{t-1}, i_t) = (0, 0)$ and truncate it to second order:

$$E(t, h_{t-1}, i_t) \approx A + Bh_{t-1} + Ci_t + h_{t-1}^T Dh_{t-1} + i_t^T Gi_t + i_t^T Hh_t$$

for tensors A, B, C, D, G and H. For simplicity, makes the assumption that A, B, C, D and G are all 0 and H is the identity, yielding:

$$E(t) = h_{t-1}i_t^T$$

The other formulation takes inspiration from eligibility traces in reinforcement learning, and instead performs an exponential average of the same quantities accross time:

$$E(t) = (1 - \eta)E(t) + \eta h_{t-1}i_t^T$$