Q-learning for mathematical applications

Andreas Bøgh Poulsen
201805425

28th June 2020

1 Preliminaries

1.1 Abstract framework of deep learning

In abstract, modern machine learning is about fitting a universal approximator to some given data. In recent instances of it, this universal approximator would be a deep neural network, possibly with some variations like convolution, recurrence or attention. Using differentiation of a neural network and gradient descent we can minimize the error between the predictions of the network and some training data.

I will not spend much time on this part and take the training of a neural network for granted. The important point is that we can minimize any error function, not just the difference between a prediction and some known data.

This is an important insight. If we can check if the network is producing right result after the fact, we don't have to know the facit beforehand. This is good because we can use the network to discover solutions without having to actually solve the problem ourselves. For example, if we wanted to learn to find a divisor of a number, it's a lot easier to check a solution than finding a factor ourselves, so this is a good problem for machine learning. On the other hand, it's difficult to check wether a number is a prime number or not, so this probably wouldn't be a good problem to solve with machine learning.

1.2 Languages and strings

Before introducing labelling schemes formally, we being by fixing notation and terminology and recalling important definitions relating to formal languages and graphs. The presentation in this first section is based on [FS91] and differs in a few ways from the usual presentation of formal languages in the computer science literature: In particular, we do not consider strings as formal sequences of symbols, but instead as mathematical sequences in a finite but otherwise unspecified set. This allows notions such as concatenation and substrings to easily be made precise. In practice we may of course use these formal strings to model strings over a character set such as ASCII or Unicode.

First a word on notation: Throughout this report we let $\mathbb{N}$ denote the natural numbers including 0. If $n \in \mathbb{N}$, we write $S_n = \{0, \ldots, n-1\}$. We also use the notation $\log$ to refer to the binary logarithm, i.e. $\log_2$. Given $n \in \mathbb{N}$, we write $\lg n = \log \max\{n, 1\}$.

1.1 · Definition. For a set X, a finite sequence in X is a map $s\colon S_n \to X$ for some $n \in \mathbb{N}$. The number n is called the length of s, and we write $|s| = n$.

The set of finite sequences in X with length n is denoted $\mathsf{Seq}_n(X)$. We write

$$X^* = \bigcup_{n \in \mathbb{N}} \mathsf{Seq}_n(X)$$

for the set of all finite sequences in X.

Notice that $S_0 = \emptyset$, so the null map is also a sequence, called the empty sequence, which we denote λ.

1.2 · Definition. Let X be a set, and consider sequences $s_1 \in \mathsf{Seq}_n(X)$ and $s_2 \in \mathsf{Seq}_m(X)$ for some $n, m \in \mathbb{N}$. The concatenation of s_1 and s_2 is the finite sequence $t \in \mathsf{Seq}_{n+m}(X)$ given by

$$t(i) = \begin{cases} s_1(i), & 0 \leq i \leq n-1, \\ s_2(i-n), & n \leq i \leq n+m-1. \end{cases}$$

The concatenation of s_1 and s_2 is denoted $s_1 \circ s_2$.

Concatenation is clearly associative, so we omit parentheses when concatenation three or more strings. Also note that $s \circ \lambda = \lambda \circ s = s$ for all $s \in X^*$.

If $s \in X^*$ and $n \in \mathbb{N}$, we define the n'th power of s recursively as follows: $s^0 = \lambda$, and for $n > 0$ we set $s^n = s \circ s^{n-1}$.

Also, if $s \in X^*$ and $k \in \mathbb{N}$ with $k \leq |s|$, we denote by s_{-k} the sequence in X we obtain from s by deleting the last k elements of the sequence. More precisely, we define the sequence $s_{-k} \in \mathsf{Seq}_{|s|-k}(X)$ by $s_{-k}(i) = s(i)$. If $k > |s|$, then we let $s_{-k} = \lambda$.

From now on we restrict ourselves to finite X, and we usually use Σ instead of X. In that case:

1.3 · Definition. An alphabet is a finite non-empty set, and elements of a given alphabet are called symbols.

If Σ is an alphabet, then elements of Σ^* are called strings or words over Σ, and the empty sequence in Σ is called the empty string.

If $x, y, z, w \in \Sigma^*$ such that $w = x \circ y \circ z$, we say that y is a substring of w.

We will chiefly be interested in binary strings, i.e. strings over the alphabet $\Sigma = \{0,1\}$. If $n \in \mathbb{N}$, then n has a unique representation as a sum of powers of 2, i.e. there exist $p \in \mathbb{N}$ and $a_0, \ldots, a_p \in \{0,1\}$ such that

$$n = \sum_{j=0}^{p} a_j 2^j.$$

Uniqueness should be understood as follows: If $n = \sum_{j=0}^{q} b_j 2^j$ is another such representation of n, then $a_0 = b_0, \ldots, a_{\min\{p,q\}} = b_{\min\{p,q\}}$, and all other coefficients are zero. By successive division by 2 with remainder it is clearly possible to obtain the coefficients a_j given n.

Let $n = \sum_{j=0}^{p} a_j 2^j \in \mathbb{N}$ with $a_p \neq 0$. We define the binary representation of n as the string $\mathsf{bin}(n) \in \mathsf{Seq}_{p+1}(\{0,1\})$ given by $\mathsf{bin}(n)(j) = a_{p-j}$, which is well-defined by the uniqueness above. This induces a map $\mathsf{bin} \colon \mathbb{N} \to \{0,1\}^*$ that maps a natural number to is binary representation as a string. As an example, if $n = 6$ we would have the representation

$$6 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0,$$

so in this case $p = 2$, $a_0 = 0$ and $a_1 = a_2 = 1$. We would like to write this binary string as 110, but to distinguish this from the integer one hundred and ten, we write it $\mathsf{bin}(6) = \mathbf{110}$. We also use this notation for more general binary strings where we allow leading zeros. Also note that $|\mathsf{bin}(x)| = \lfloor \lg x \rfloor + 1$.

When concatenating binary strings we might not have any knowledge about the length of each substring in the concatenation, so it might not be possible to extract each substring. To remedy this, we use a variation of the Elias γ encoding of a natural number n: For $n = 0$ we let $\mathsf{enc}_\gamma(0) = \mathbf{0}$, and for $n > 0$ we let

$$\mathsf{enc}_\gamma(n) = \mathbf{1} \circ \mathbf{0}^{\lfloor \lg n \rfloor} \circ \mathsf{bin}(n).$$

Recall that $\mathsf{bin}(n)$ has no leading zeros. When reading a binary γ encoded string $\mathsf{enc}_\gamma(n)$, we first read the first bit. If it is $\mathbf{0}$, then $n = 0$. Otherwise, the first bit is $\mathbf{1}$, and we consider the remainder of the string. Read the number of leading zeros to find $\lfloor \lg n \rfloor$, and then read the following $\lfloor \lg n \rfloor + 1$ symbols to find $\mathsf{bin}(n)$. In total we thus have $|\mathsf{enc}_\gamma(n)| = 2\lfloor \lg n \rfloor + 2$. Notice however that to extract $\mathsf{bin}(n)$ from a string on the form $x \circ \mathsf{enc}_\gamma(n) \circ y$ for $x, y \in \{0,1\}^*$ we need to know the length of x, but not the length of y.

1.3 Graphs

Before introducing labelling schemes formally, we recall simple properties of graphs. The terminology used is largely consistent with [Cor+09]. First the basic definitions:

1.4 • Definition. A graph G is a pair (V, E), where V is a finite set and E is a collection of two-element[1] subsets of V. We call $V = V(G)$ the set of vertices or nodes and $E = E(G)$ the set of edges. The number of vertices $|V|$ in G is also denoted $|G|$.

If $e = \{u, v\} \in E$ for some $u, v \in V$, we say that the vertices u and v are adjacent in G and call them neighbours. Furthermore, we call u and v the endpoints of the edge e, and say that e is incident on u and v.

The degree of a vertex is the number of distinct edges incident on it.

We also use the notation (u, v) for the edge $\{u, v\}$, keeping in mind that the pair is not ordered. We do not consider directed graphs in this report.

Henceforth let $G = (V, E)$ be a graph.

1.5 • Definition. A subgraph $G' = (V', E')$ of G is a graph such that $V' \subseteq V$ and $E' \subseteq E$.

For a subset $S \subseteq V$ the induced subgraph $G[S]$ is the subgraph of G whose vertex set is S and those edges are all edges in E whose endpoints lie in S.

1.6 • Definition.

(i) A path of length k in G is a finite sequence[2] $\langle u_0, \ldots, u_k \rangle$ of vertices of G such that $(u_i, u_{i+1}) \in E$ for $i = 0, \ldots, k-1$. The path contains the vertices $u_0, \ldots, u_k$ and the edges $(u_0, u_1), \ldots, (u_{k-1}, u_k)$. We say that the path above connects the vertices u_0 and u_k.

(ii) A path is simple if all vertices on the path are distinct.

(iii) A path $\langle u_0, \ldots, u_k \rangle$ is a cycle if $u_0 = u_k$ and $k \geq 3$.

(iv) A cycle $\langle u_0, \ldots, u_k \rangle$ is simple if the vertices $u_1, \ldots, u_k$ are distinct.

(v) If there is a simple path in G that contains all vertices of G, we also call G itself a path.

1.7 • Definition.

(i) The graph G is connected if for any pair $u, v \in V$ of vertices there is a path in G that connects u and v.

(ii) The graph G is acyclic if it does not contain a simple cycle.

(iii) A connected acyclic graph is called a tree. The vertices with degree 1 are called leaves, and the remaining vertices are called inner vertices.

[1] Notice that we do not allow loops or multiple edges.

[2] We use angle brackets to denote paths to distinguish them from edges.

1.8 • Definition. A caterpillar is a tree whose inner vertices induce a path. More precisely, let S be the set of inner vertices of a tree T. If the induced subgraph $T[S]$ is a path, then T is a caterpillar.

Since trees are connected and acyclic, there is a unique path between any pair of vertices in the tree. This leads to the following definition, following [Wes01]:

1.9 • Definition. A rooted tree is a tree $T = (V, E)$ with one vertex r chosen as root. For each vertex $u \in V$ let $P(u)$ be the unique path that connects u and r. If $u \neq r$, then the parent of u is the neighbour of u that lies on $P(u)$. The children of u are its other neighbours. The ancestors of u are the vertices in $P(u)$ distinct from u, and the descendants of u are the vertices v such that u lies on $P(v)$. The leaves of T are the vertices with no children.

If $u \in V$, the subtree rooted in u is the subgraph induced by the descendants of u. This is clearly also a tree which we denote T_u.

Note that the definitions of a leaf in a tree and in a rooted tree are different. For instance, a path of length at least 2 is a tree, and rooting it at one of the endpoints yields two leaves in the first sense, but only one leaf in the second sense, since the root is not a leaf. Whenever we consider rooted trees, we will only talk about leaves in the second sense.

We now introduce the heavy-light decomposition of trees, modified from [ADK17]. Given a rooted tree T and a vertex $u \in T$, let $v_1, \dots, v_k$ be the children of u, and order them such that $|T_{v_k}| \geq |T_{v_i}|$ for $i = 1, \dots, k$. Thus the subtree rooted in v_k is the 'heaviest' among the T_{v_i}, and we write $v_k = \mathsf{heavy}(u)$.[3] We furthermore define the light subtree of u, denoted T_u^l, as the subtree induced by the vertices in T_u, except for the vertices in the $T_{\mathsf{heavy}(u)}$, i.e.

$$T_u^l = T_u\Big[V(T_u) \setminus V(T_{\mathsf{heavy}(u)})\Big].$$

Intuitively, T_u^l is the tree we obtain by removing from T_u the 'heavy subtree' $T_{\mathsf{heavy}(u)}$. Notice that u itself lies in the light subtree of u.

A particular kind of rooted tree that is of great importance in this report is the binary tree:

1.10 • Definition. A binary tree is a rooted tree $T = (V, E)$ where each vertex has at most two children. Each child of a vertex is designated as its left or right child, and subtrees rooted in each child are called the left and right subtrees respectively.

[3]If there is no unique 'heaviest' child, we pick v_k arbitrarily among the heaviest children.

References

[ADK17] Stephen Alstrup, Søren Dahlgaard and Mathias Bæk Tejs Knudsen. 'Optimal Induced Universal Graphs and Adjacency Labeling for Trees'. In: Journal of the ACM 64.4 (Aug. 2017). DOI: 10.1145/3088513.

[Cor+09] Thomas H. Cormen et al. Introduction to Algorithms. 3rd ed. The MIT Press, 2009. 1292 pp. ISBN: 978-0-262-53305-8.

[FS91] Peter A. Fejer and Dan A. Simovici. Mathematical Foundations of Computer Science. Volume I: Sets, Relations, and Induction. 1st ed. Springer-Verlag New York Inc., 1991. 425 pp. ISBN: 978-1-4612-7792-7. DOI: 10.1007/978-1-4612-3086-1.

[Wes01] Douglas B. West. Introduction to Graph Theory. 2nd ed. Pearson Education, Inc., 2001. 588 pp. ISBN: 81-7808-830-4.