Dataflow Matrix Machines as a Model of Computations with Linear Streams

Michael Bukatin (HERE Technologies), Jon Anthony (Boston College)

Goal

Create a generalization of neural nets sufficiently powerful for general-purpose programming

RNN

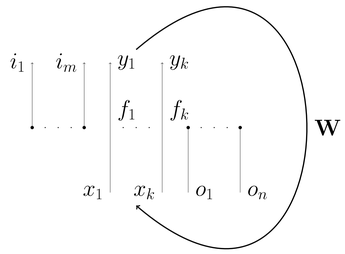

Fig. 1: "Two-stroke engine" for an RNN. "Down movement": $(x_1^{t+1}, \ldots, x_k^{t+1}, o_1^{t+1}, \ldots, o_n^{t+1})^\top = \mathbf{W} \cdot (y_1^t, \ldots, y_k^t, i_1^t, \ldots, i_n^t)^\top$.
"Up movement": $y_1^{t+1} = f_1(x_1^{t+1}), \ldots, y_k^{t+1} = f_k(x_k^{t+1})$.

Linear streams

Instead of streams of numbers, any streams allowing to take **linear combinations of streams**.

A finite or countable collection of **kinds of linear streams**.

We currently consider:

Streams of approximate representations of arbitrary vectors.

With every **kind of linear streams** k, associate a vector space V_k and a way to compute an approximate representation of vector $\alpha_1 v_{1,k} + \ldots + \alpha_n v_{n,k}$ from approximate representations of vectors $v_{1,k}, \ldots, v_{n,k}$.

E.g. streams of sparse tensors of particular shape.

Infinite-dimensional vector spaces are OK too. E.g. streams of positively and negatively marked probabilistic samples representing elements of the vector space of finite signed measures over some X are allowed.

DMM: main ingredients

- arbitrary **linear streams**
- arbitrary input and output arity of neurons
- countable network with finite active part

Standard DMM

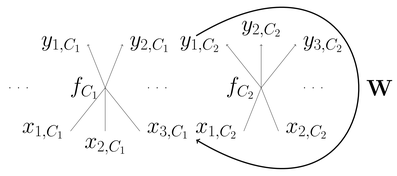

Fig. 2: "Two-stroke engine" for a standard DMM

"Down movement": for all inputs x_{i,C_k} such that there is a non-zero weight $w^t_{(i,C_k),(j,C_l)}$:

$$x^{t+1}_{i,C_k} = \sum_{\{(j,C_l) | w^t_{(i,C_k),(j,C_l)} \neq 0\}} w^t_{(i,C_k),(j,C_l)} * y^t_{j,C_l}.$$

Note that x^{t+1}_{i,C_k} and y^t_{j,C_l} are no longer numbers, but vectors (here, the formulas are written in terms of vectors themselves, and not in terms of their approximate representations actually used by the DMM in question), so the type correctness condition states that $w^t_{(i,C_k),(j,C_l)}$ can be non-zero only if x_{i,C_k} and y_{j,C_l} belong to the same vector space.

"Up movement": for all active neurons C:

$$y^{t+1}_{1,C}, ..., y^{t+1}_{n_C,C} = f_C(x^{t+1}_{1,C}, ..., x^{t+1}_{m_C,C}).$$

Because input and output arities are allowed to be zero, special handling of network inputs and outputs which has been required for RNNs is not required here.

A neuron is active if there is at least one non-zero weight associated with one of its inputs or outputs. The network matrix can change with time, but we require that it has only a finite number of non-zero elements at any given time, hence only a finite part of the network is active at any given moment of time.

Goal

Simplicity of RNNs (one kind of linear streams, hence no type correctness conditions, and all neuron activation functions having input and output arity one) together with the expressive power of DMMs

Vector Space V

Consider a countable set L of tokens (pragmatically speaking, L is often the set of all legal keys of hash dictionaries in a given programming language). Consider the set L^* of finite sequences of non-negative length of elements of L.

The vector space V: the space of finite formal linear combinations of elements of L^* over reals.

Several fruitful ways to view elements of V:

- Finite linear combinations of finite strings
- Finite prefix trees with numerical leaves
- Sparse "tensors of mixed rank"
- Recurrent maps $L \to \mathbb{R} \oplus V$

Variadic neurons

Activation functions $f : V \to V$.

The collection F of neuron types consists of functions f.

Labels from L on the first level of elements of V are names of inputs and outputs.

The network matrix $\mathbf{W}$ has a natural structure of multidimensional tensor.

"Down movement":

$$x^{t+1}_{f,n_f,i} = \sum_{g \in F} \sum_{n_g \in L} \sum_{o \in L} w^t_{f,n_f,i,g,n_g,o} * y^t_{g,n_g,o}.$$

"Up movement":

$$y^{t+1}_{f,n_f} = f(x^{t+1}_{f,n_f}).$$

Self-referential mechanism

Streams of matrices or tensors shaped like $\mathbf{W}$.

Dedicate one specific neuron `Self`, and use one particular output of `Self` emitting streams of such matrices.

On each "down movement", the latest value emitted by `Self` on that output will be used as $\mathbf{W}$.

The simplest way is to use as `Self` an accumulator neuron taking additive updates from other neurons in the network.

A neuron with identity activation function, or with activation function $y = x + \Delta x$ is convenient for that.

For $y = x + \Delta x$, one sets $w_{x,y} = 1$ to make it an accumulator, and accepts additive updates via the row of $\mathbf{W}$ corresponding to Δx.

This mechanism allows to encode **algorithms changing network weights or topology** within the network itself.

This supports traditional learning, various **learning to learn** schemas, use of **fast weights** [arxiv:1610.06258], etc.

Fuzzy `if` ("gating")

A two-argument neuron: $y = x_1 * x_2$ [Pollack 1987]
Occurs implicitly in LSTM and Gated Recurrent Unit nets.

DMM links and references therein:

One-page overview:
http://www.cs.brandeis.edu/~bukatin/dataflow-matrix-machines-2016.pdf
This paper:
http://www.cs.brandeis.edu/~bukatin/dmm_learn_aut.pdf
Background:
http://www.cs.brandeis.edu/~bukatin/partial_inconsistency.html
Core DMM primitives in Clojure:
https://github.com/jsa-aerial/DMM