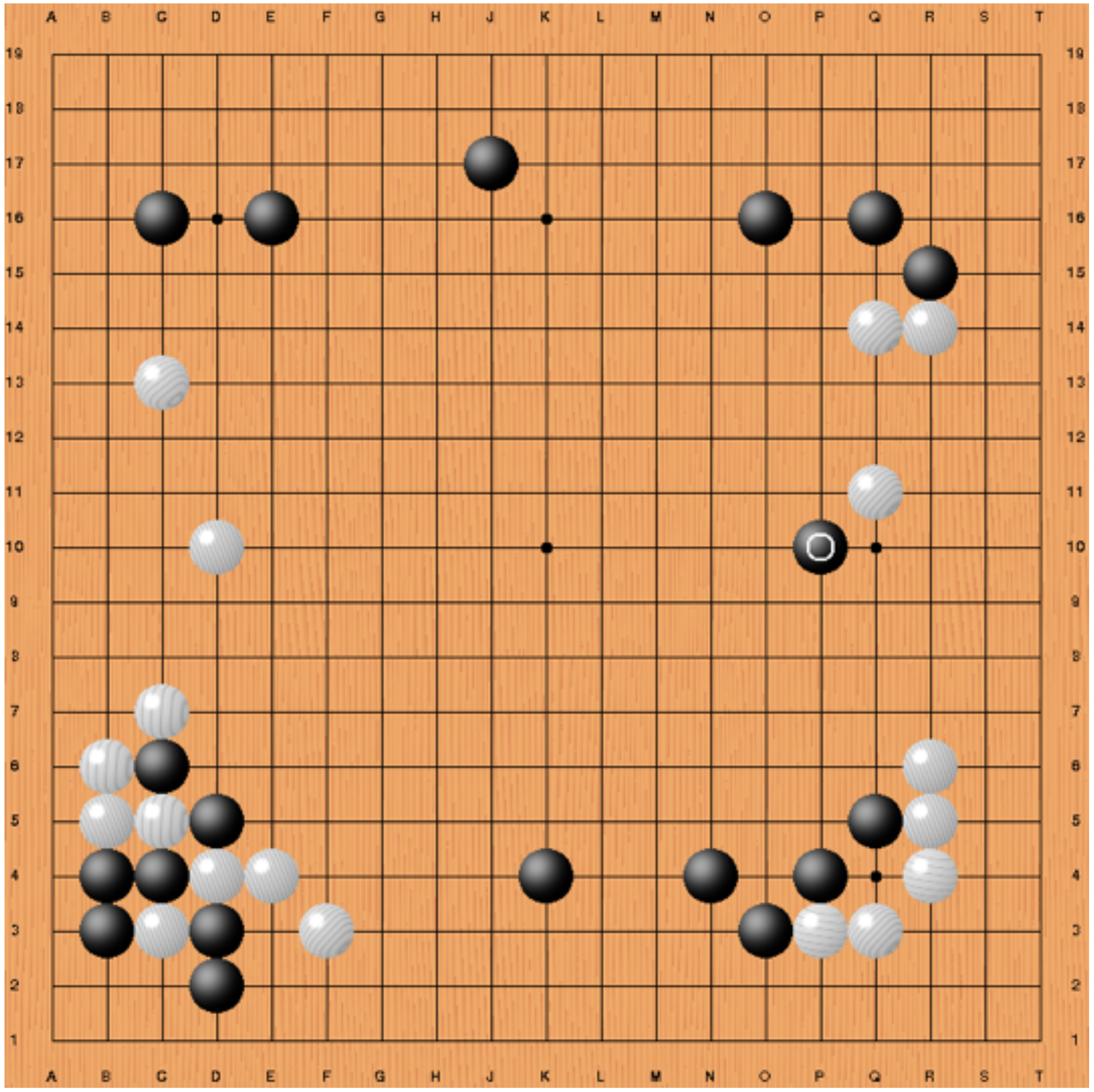
Linear logic and deep learning

Huiyi Hu, Daniel Murfet

Move 37, Game 2, Sedol-AlphaGo



"*It's not a human move. I've never seen a human play this move… So beautiful.*"
- Fan Hui, three-time European Go champion.

Outline

- Introduction to deep learning
- What is differentiable programming?
- Linear logic and the polynomial semantics
- Our model

Intro to deep learning

A *deep neural network* (DNN) is a particular kind of function

$$F_w : \mathbb{R}^m \longrightarrow \mathbb{R}^n$$

which depends in a “differentiable” way on a vector of weights.

Example: Image classification, as in: cat or dog?

$$F_w\Bigg($$ $$\Bigg) = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix} \in \mathbb{R}^2$$

$$\text{image} \in \mathbb{R}^{685 \cdot 685 \cdot 3}$$

$$p_{\text{cat}} = 0.7$$

$$p_{\text{dog}} = 0.3$$

Intro to deep learning

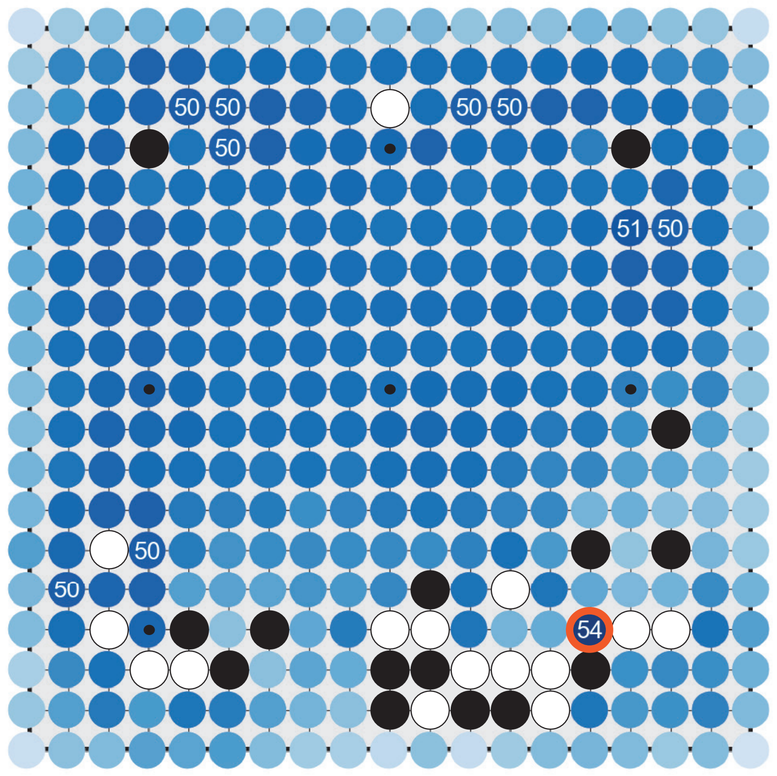
A *deep neural network* (DNN) is a particular kind of function

$$F_w : \mathbb{R}^m \longrightarrow \mathbb{R}^n$$

which depends in a “differentiable” way on a vector of weights.

Example: Playing the game of Go

$$\underset{\substack{\pitchfork \\ \mathbb{R}^{19\times 19}}}{F_w(\text{current board})} =$$



$$\in \mathbb{R}^{19\cdot 19}$$

Intro to deep learning

A *deep neural network* (DNN) is a particular kind of function

$$F_w : \mathbb{R}^m \longrightarrow \mathbb{R}^n$$

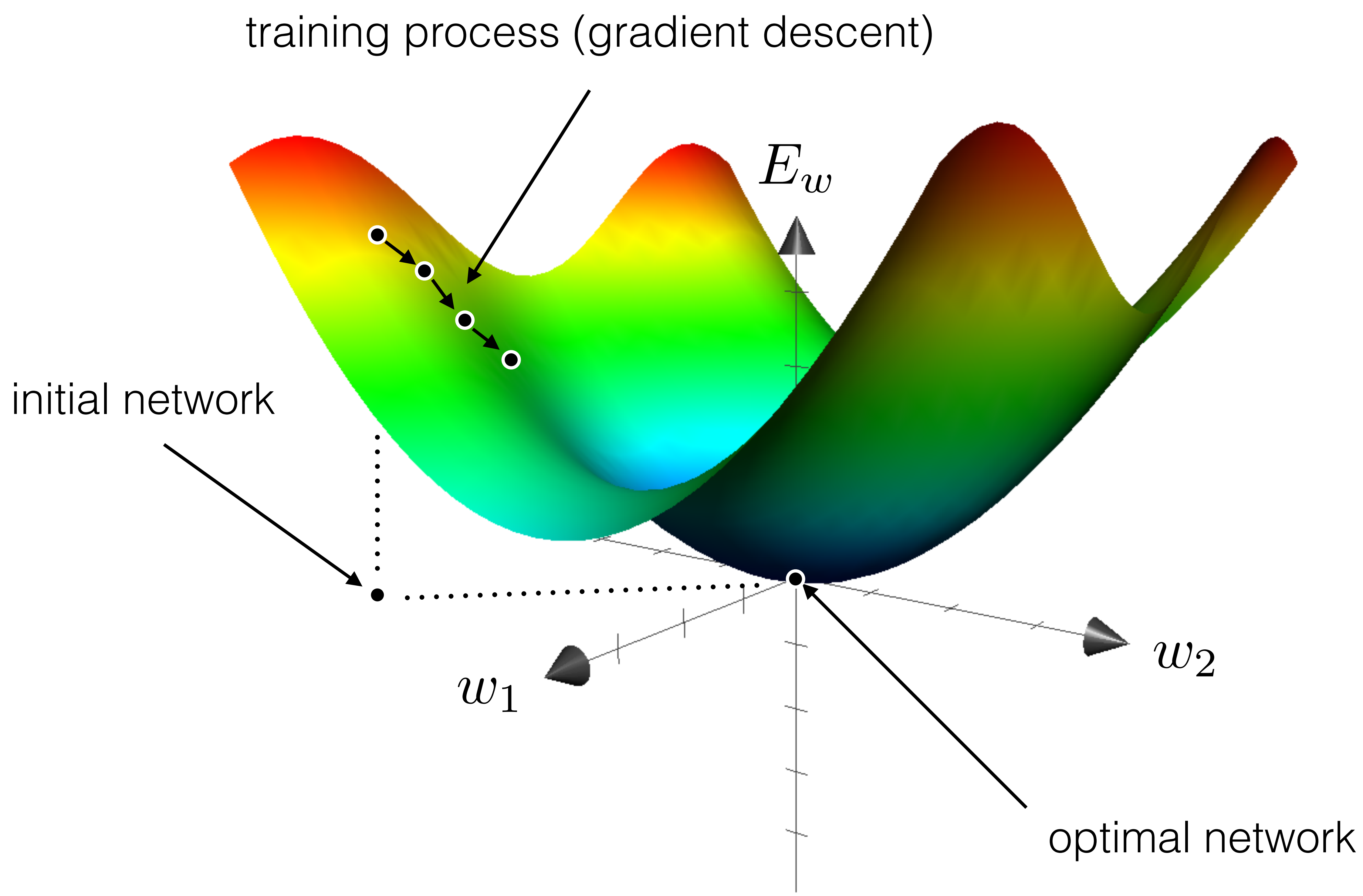
which depends in a “differentiable” way on a vector of weights.

A randomly initialised neural network is **trained** on sample data

$$\{(x_i, y_i)\}_{i \in I}\ , x_i \in \mathbb{R}^m, y_i \in \mathbb{R}^n$$

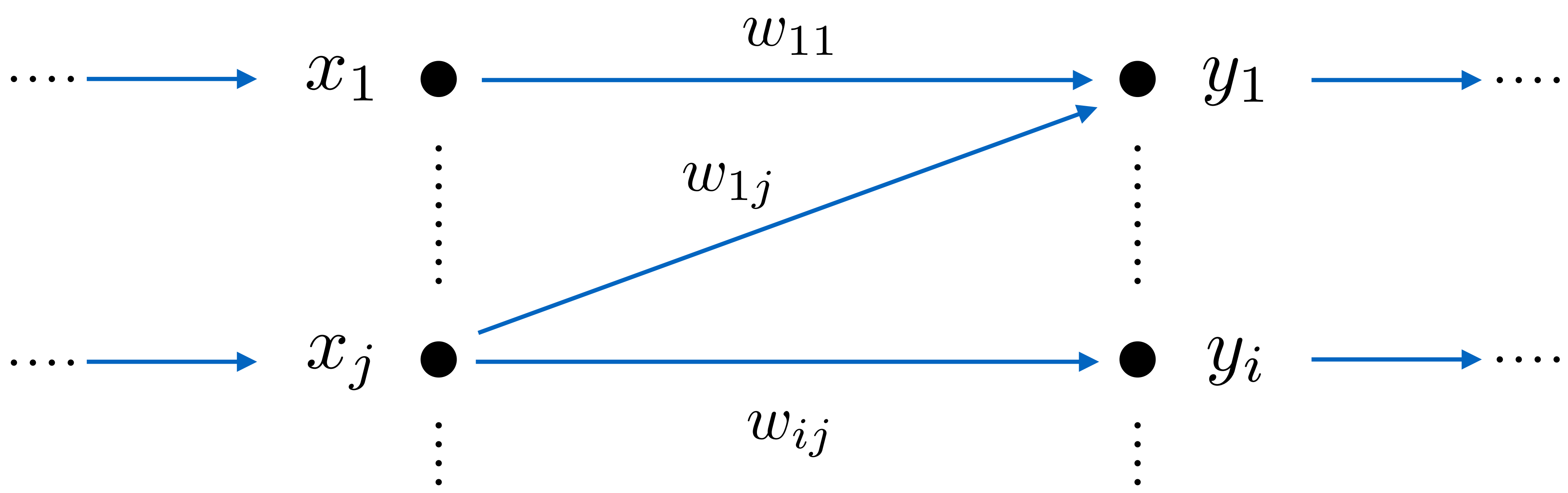
by varying the weight vector in order to minimise the error

$$E_w = \sum_i (F_w(x_i) - y_i)^2$$


training process (gradient descent)
E_w
initial network
w_1
w_2
optimal network

Intro to deep learning

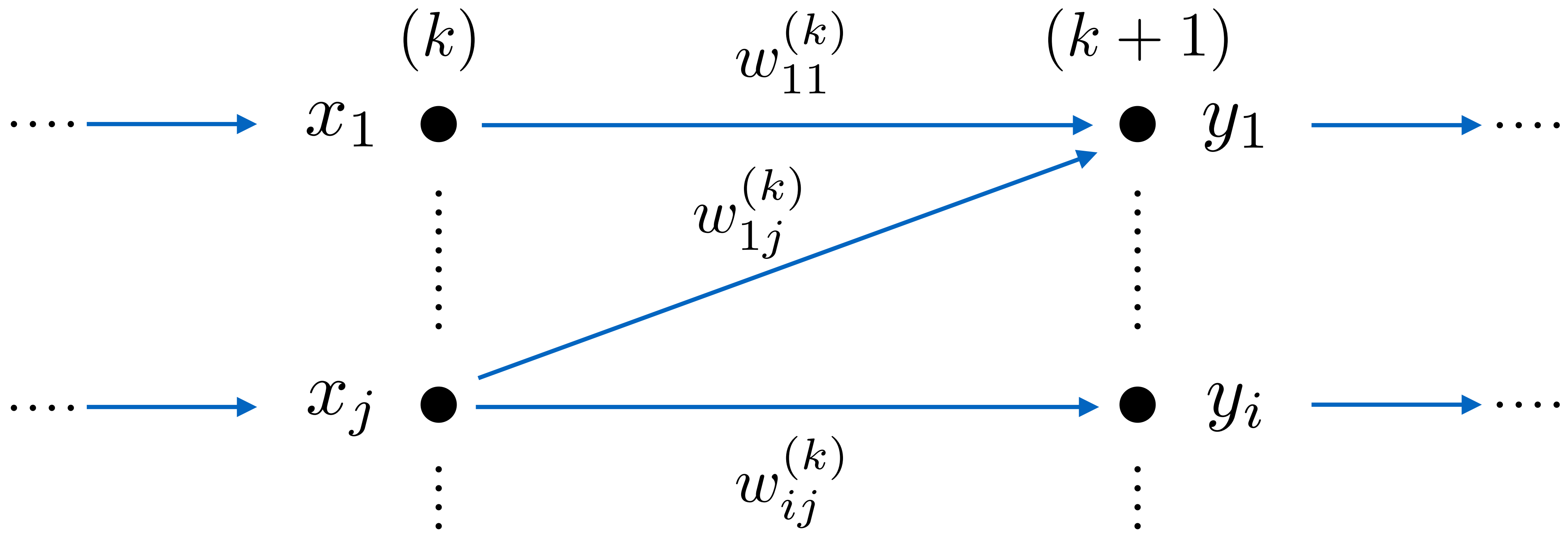
A DNN is made up of many layers of “neurons” connecting the input (on the left) to the output (on the right) as in the following diagram:



$$y_i = \sigma\left(\sum_j w_{ij} x_j\right) \qquad \sigma(x) = \tfrac{1}{2}(x + |x|)$$

Intro to deep learning

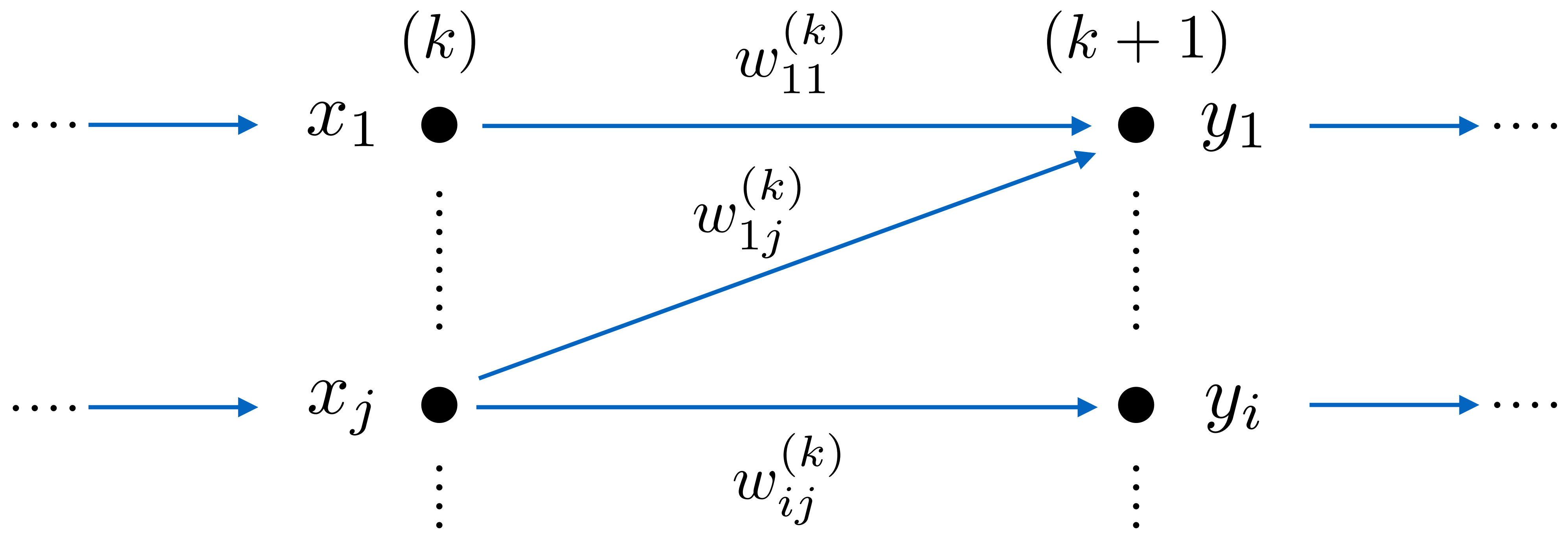
A DNN is made up of many layers of “neurons” connecting the input (on the left) to the output (on the right) as in the following diagram:



$$y_i = \sigma\Big(\sum_j w_{ij}^{(k)} x_j\Big) \qquad \sigma(x) = \tfrac{1}{2}(x + |x|)$$

Intro to deep learning

A DNN is made up of many layers of "neurons" connecting the input (on the left) to the output (on the right) as in the following diagram:

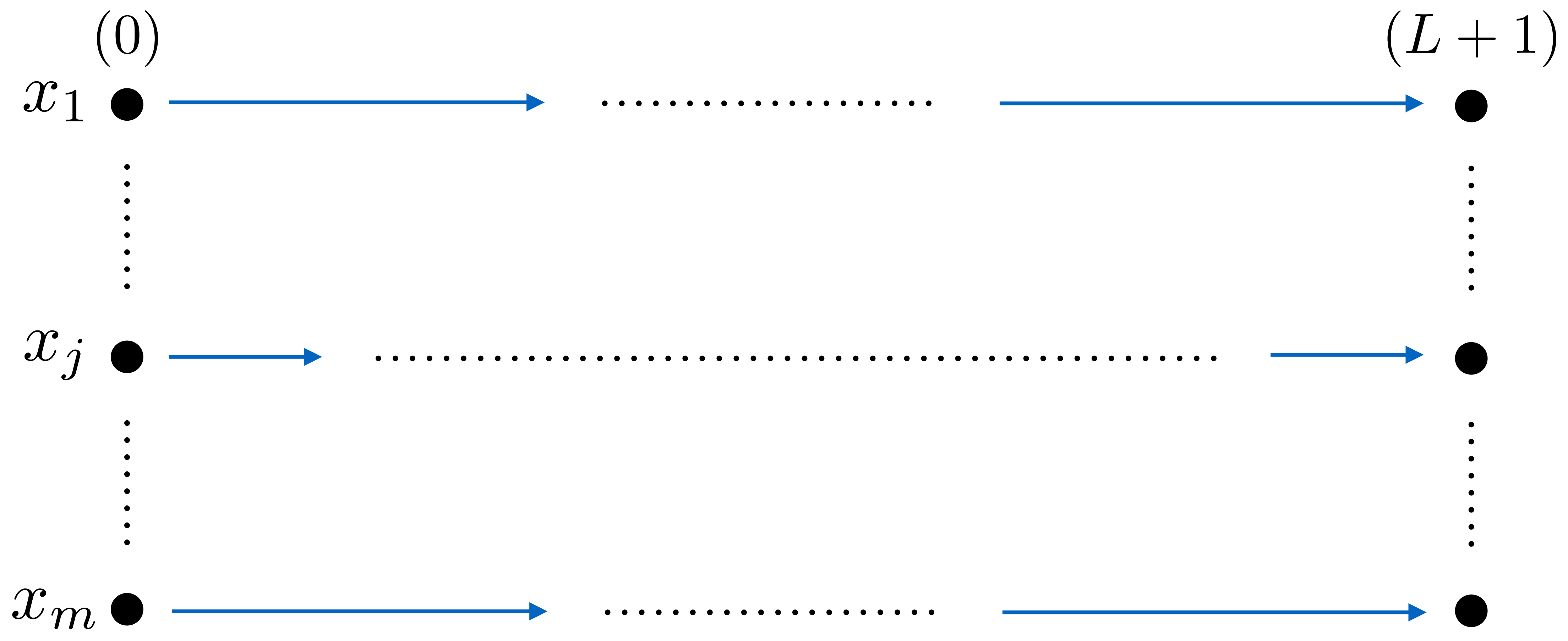


$y_i = \sigma\Big(\sum_j w_{ij}^{(k)} x_j\Big)$ $\quad \vec{y} = \sigma\left(W^{(k)}\vec{x}\right) \qquad \sigma(x) = \frac{1}{2}(x + |x|)$

$$\vec{x} = (x_1, x_2, \ldots, x_j, \ldots)^T \quad W^{(k)} = (w_{ij}^{(k)})_{i,j}$$

Intro to deep learning

A DNN is made up of many layers of “neurons” connecting the input (on the left) to the output (on the right) as in the following diagram:



$F_w : \mathbb{R}^m \longrightarrow \mathbb{R}^n$

$$F_w(\vec{x}) = \sigma\Big(W^{(L)} \cdots \sigma\Big(W^{(1)}\sigma\Big(W^{(0)}\vec{x}\Big)\cdots\Big)$$

Applied deep learning

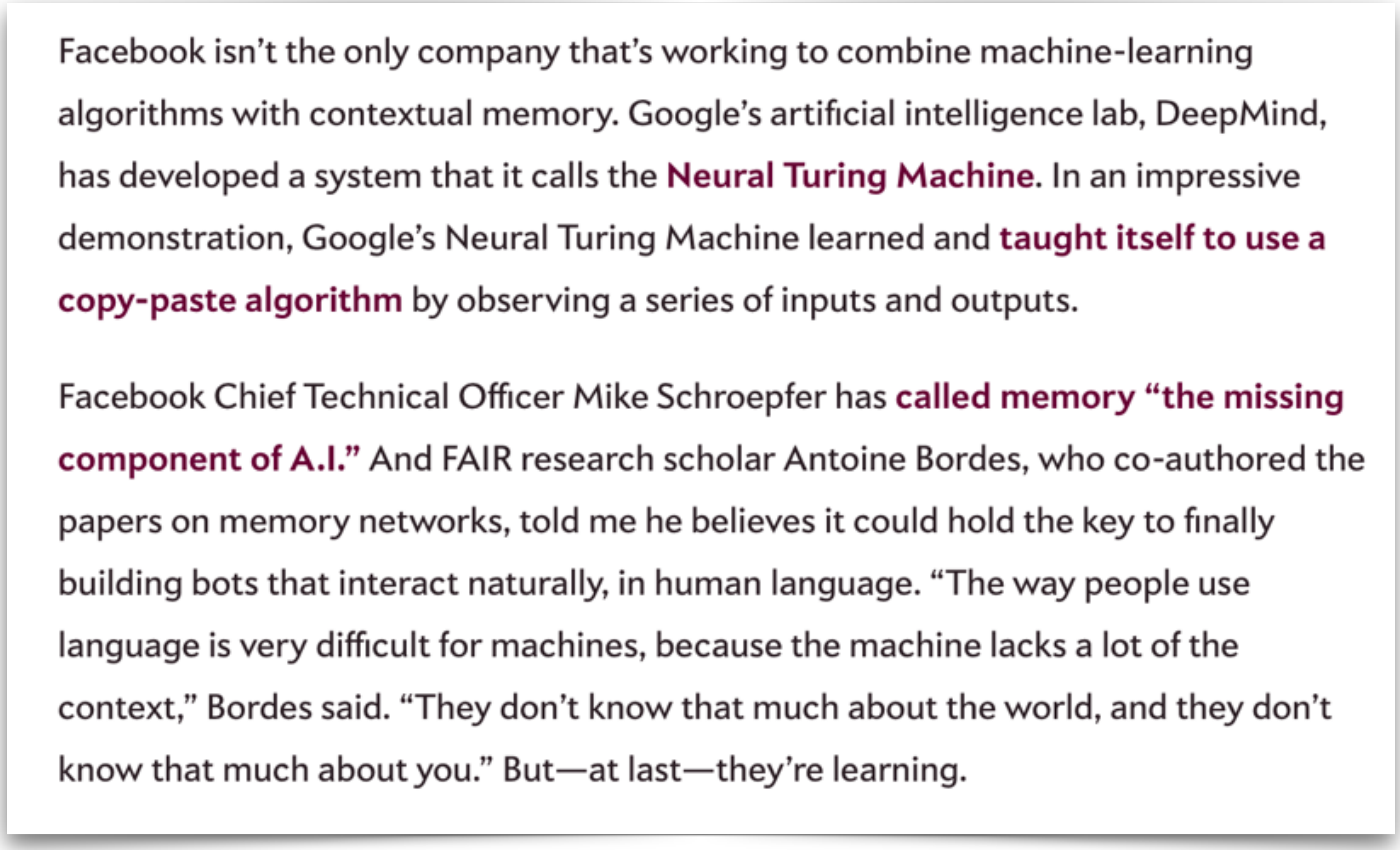
- Natural Language Processing (**NLP**) includes machine translation, processing of natural language queries (e.g. Siri) and question answering wrt. unstructured text (e.g. “where is the ring at the end of LoTR?”).

- Two common neural network architectures for NLP are Recurrent Neural Networks (**RNN**) and Long-Short Term Memory (**LSTM**). Both are variations on the DNN architecture.

- Many hard unsolved NLP problems require sophisticated reasoning about context and an understanding of algorithms. To solve these problems, recent work equips RNNs/LSTMs with “Von Neumann elements” e.g. stack **memory**.

- RNN/LSTM controllers + differentiable memory is one model of **differentiable programming**, currently being explored at e.g. Google, Facebook, …

- These “memory enhanced” neural networks are already being used in production, e.g. Facebook messenger (~900m users).


Facebook isn’t the only company that’s working to combine machine-learning algorithms with contextual memory. Google’s artificial intelligence lab, DeepMind, has developed a system that it calls the **Neural Turing Machine**. In an impressive demonstration, Google’s Neural Turing Machine learned and **taught itself to use a copy-paste algorithm** by observing a series of inputs and outputs.

Facebook Chief Technical Officer Mike Schroepfer has **called memory “the missing component of A.I.”** And FAIR research scholar Antoine Bordes, who co-authored the papers on memory networks, told me he believes it could hold the key to finally building bots that interact naturally, in human language. “The way people use language is very difficult for machines, because the machine lacks a lot of the context,” Bordes said. “They don’t know that much about the world, and they don’t know that much about you.” But—at last—they’re learning.

Salon, “Facebook Thinks It Has Found the Secret to Making Bots Less Dumb”, June 28 2016.

What is differentiable programming?

The idea is that a “differentiable” program

$$P_w : I \longrightarrow O$$

should depend smoothly on weights, *and,* by varying the weights, one would learn a program with desired properties.

- **Question**: what if instead of coupling neural net controllers to *imperative* algorithmic elements (e.g. memory) we couple a neural net to *functional* algorithmic elements?
- **Problem**: how to make an abstract functional language like Lambda calculus, System F, etc. differentiable?
- **Our idea**: use *categorical semantics* in a category of “differentiable” mathematical objects compatible with neural networks (e.g. matrices of polynomials).

Linear logic and semantics

- Discovered by Girard in the 1980s, linear logic is a substructural logic with contraction and weakening available only for formulas marked with a new connective "**!**" (called the exponential).

- The usual connectives of logic (e.g. conjunction, implication) are decomposed into **!** together with a *linearised* version of that connective (called resp. tensor $\otimes$, linear implication $\multimap$).

- Under the Curry-Howard correspondence, linear logic corresponds to a programming language with "resource management" and symmetric monoidal categories equipped with a special kind of comonad.

Linear logic and semantics

variables: $\alpha, \beta, \gamma, \ldots$

formulas: $!F, F \otimes F', F \multimap F', F \& F', \forall \alpha F$, constants

$$\boxed{\mathbf{int} = \forall \alpha \; !(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)}$$

$$\boxed{\mathbf{bint} = \forall \alpha \; !(\alpha \multimap \alpha) \multimap \big(!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)\big)}$$

Deduction rules for linear logic

(Axiom): $$\frac{}{A \vdash A}$$

(Cut): $$\frac{\Gamma \vdash A \qquad \Delta', A, \Delta \vdash B}{\Delta', \Gamma, \Delta \vdash B}\,\text{cut}$$

(Exchange): $$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

(Left $\otimes$): $$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \otimes B, \Delta \vdash C}\,\otimes\text{-}L$$

(Right $\otimes$): $$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\,\otimes\text{-}R$$

(Right $\multimap$): $$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\,\multimap R$$

(Left $\multimap$): $$\frac{\Gamma \vdash A \qquad \Delta', B, \Delta \vdash C}{\Delta', \Gamma, A \multimap B, \Delta \vdash C}\,\multimap L$$

(Promotion): $$\frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\,\text{prom}$$

(Dereliction): $$\frac{\Gamma, A, \Delta \vdash B}{\Gamma, !A, \Delta \vdash B}\,\text{der}$$

(Weakening): $$\frac{\Gamma, \Delta \vdash B}{\Gamma, !A, \Delta \vdash B}\,\text{weak}$$

(Contraction): $$\frac{\Gamma, !A, !A, \Delta \vdash B}{\Gamma, !A, \Delta \vdash B}\,\text{ctr}$$

$$\frac{\Gamma, A[B/x] \vdash C}{\Gamma, \forall x \,.\, A \vdash C}\,\forall L$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x \,.\, A}\,\forall R$$

Binary integers

$$\boxed{\mathbf{bint} = \forall\alpha\ !(\alpha \multimap \alpha) \multimap (!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha))}$$

$$S \in \{0,1\}^* \longmapsto \text{proof } t_S \text{ of } \vdash \mathbf{bint}$$

$$\begin{prooftree}
\AxiomC{}
\UnaryInfC{$\alpha \vdash \alpha$}
\AxiomC{}
\UnaryInfC{$\alpha \vdash \alpha$}
\AxiomC{}
\UnaryInfC{$\alpha \vdash \alpha$}
\AxiomC{}
\UnaryInfC{$\alpha \vdash \alpha$}
\RightLabel{$\multimap L$}
\BinaryInfC{$\alpha, \alpha \multimap \alpha \vdash \alpha$}
\RightLabel{$\multimap L$}
\BinaryInfC{$\alpha, \alpha \multimap \alpha, \alpha \multimap \alpha \vdash \alpha$}
\RightLabel{$\multimap L$}
\BinaryInfC{$\alpha, \alpha \multimap \alpha, \alpha \multimap \alpha, \alpha \multimap \alpha \vdash \alpha$}
\RightLabel{$\multimap R$}
\UnaryInfC{$\alpha \multimap \alpha, \alpha \multimap \alpha, \alpha \multimap \alpha \vdash \alpha \multimap \alpha$}
\doubleLine
\RightLabel{der}
\UnaryInfC{$!(\alpha \multimap \alpha), !(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \alpha \multimap \alpha$}
\RightLabel{ctr}
\UnaryInfC{$!(\alpha \multimap \alpha), !(\alpha \multimap \alpha) \vdash \alpha \multimap \alpha$}
\RightLabel{$\multimap R$}
\UnaryInfC{$!(\alpha \multimap \alpha) \vdash !(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$}
\RightLabel{$\multimap R$}
\UnaryInfC{$\vdash !(\alpha \multimap \alpha) \multimap (!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha))$}
\end{prooftree}$$

t_{001}

Polynomial semantics (sketch)

$$[\![\alpha]\!] = \mathbb{R}^d$$

$$[\![\alpha \multimap \alpha]\!] = L(\mathbb{R}^d, \mathbb{R}^d) = M_d(\mathbb{R})$$

$$[\![!(\alpha \multimap \alpha)]\!] = \mathbb{R}[\{a_{ij}\}_{1 \le i,j \le d}] = \mathbb{R}[a_{11}, \ldots, a_{ij}, \ldots, a_{dd}]$$

(one variable per entry in a d x d matrix)

$$[\![\mathbf{bint}]\!] = [\![!(\alpha \multimap \alpha) \multimap \big(!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)\big)]\!]$$

$$= M_d\big(\mathbb{R}[\{a_{ij}, a'_{ij}\}_{1 \le i,j \le d}]\big)$$

(matrices of polynomials in the variables a, a’)

Polynomial semantics (sketch)

$$[\![\mathbf{bint}]\!] = M_d\big(\mathbb{R}[\{a_{ij}, a'_{ij}\}_{1\leq i,j\leq d}]\big)$$

(matrices of polynomials in the variables a, a')

$$S \in \{0,1\}^* \qquad t_S : \mathbf{bint}, \;\; [\![t_S]\!] \in [\![\mathbf{bint}]\!]$$

$$[\![t_0]\!] = \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ a_{21} & \cdots & a_{2d} \\ \vdots & & \vdots \\ a_{d1} & \cdots & a_{dd} \end{pmatrix} \qquad [\![t_1]\!] = \begin{pmatrix} a'_{11} & \cdots & a'_{1d} \\ a'_{21} & \cdots & a'_{2d} \\ \vdots & & \vdots \\ a'_{d1} & \cdots & a'_{dd} \end{pmatrix}$$

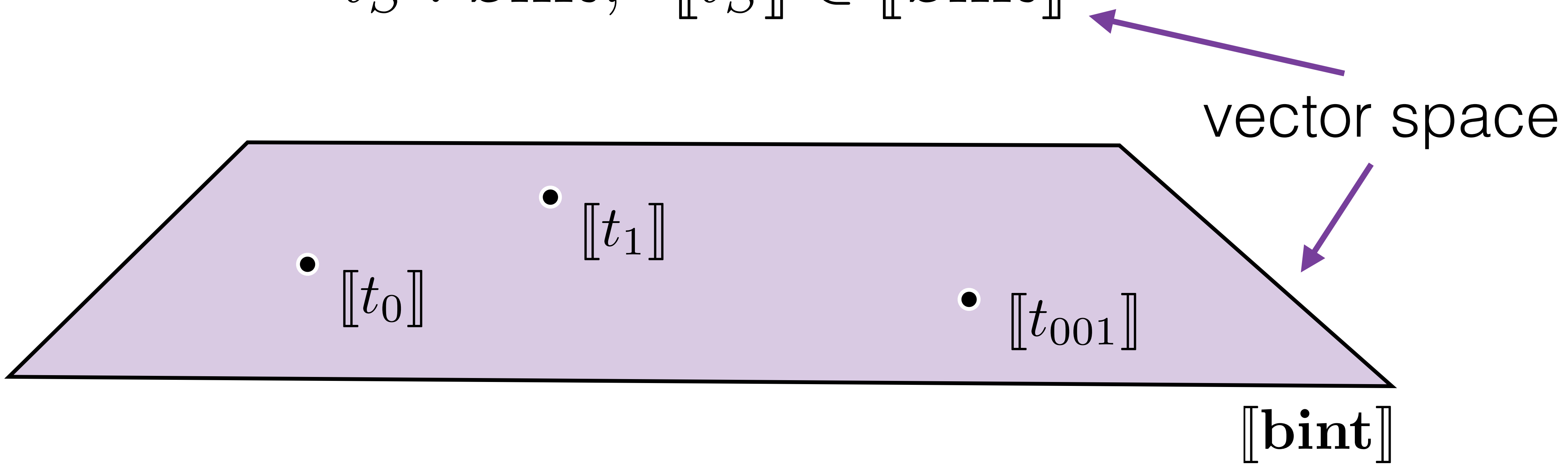
$$[\![t_{001}]\!] = [\![t_0]\!] \cdot [\![t_0]\!] \cdot [\![t_1]\!]$$

Polynomial semantics (sketch)

$$[\![\mathbf{bint}]\!] = M_d\big(\mathbb{R}[\{a_{ij}, a'_{ij}\}_{1\le i,j\le d}]\big)$$

(matrices of polynomials in the variables a, a')

$$t_S : \mathbf{bint}, \quad [\![t_S]\!] \in [\![\mathbf{bint}]\!]$$



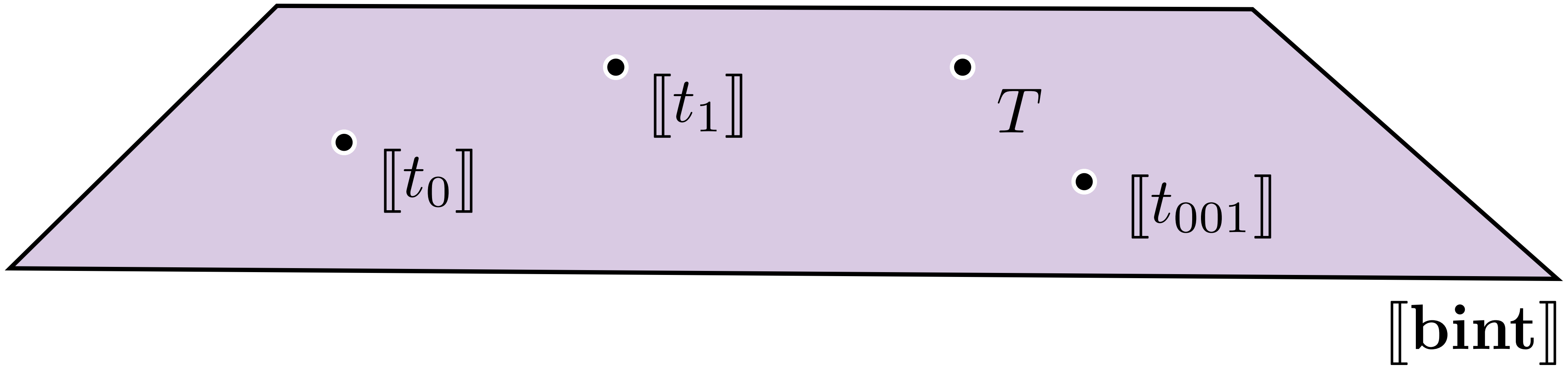
Upshot: proofs/programs to vectors

Polynomial semantics (sketch)

$$T \in [\![\mathbf{bint}]\!] = M_d\big(\mathbb{R}[\{a_{ij}, a'_{ij}\}_{1 \le i,j \le d}]\big)$$

(matrices of polynomials in the variables a, a')

$$\mathrm{eval}_T(A, B) = T|_{a_{ij}=A_{ij}, a'_{ij}=B_{ij}} \qquad A, B \in M_d(\mathbb{R})$$



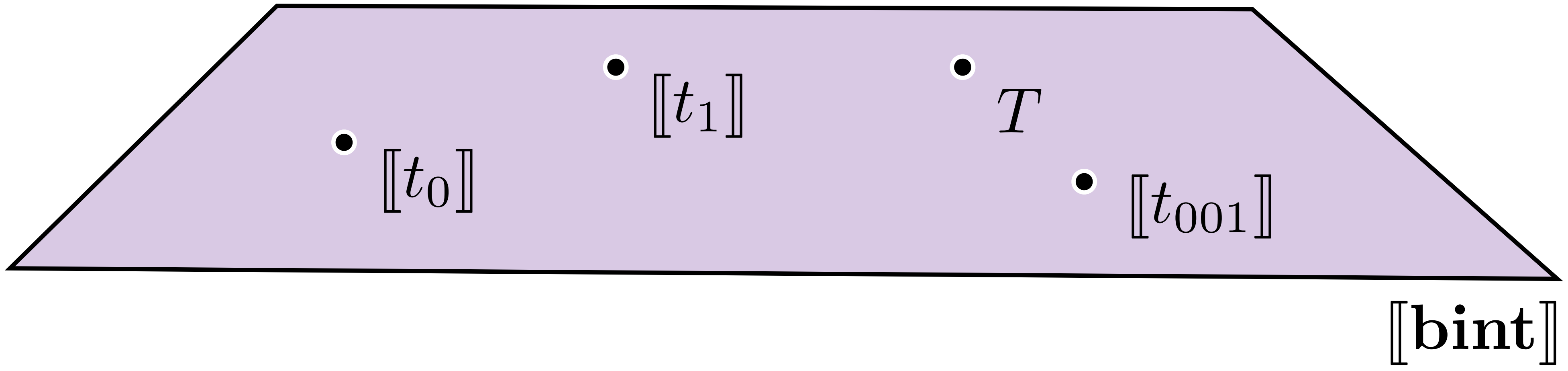
Upshot: proofs/programs to vectors

Polynomial semantics (sketch)

$$T \in [\![\mathbf{bint}]\!] = M_d\big(\mathbb{R}[\{a_{ij}, a'_{ij}\}_{1 \leq i,j \leq d}]\big)$$

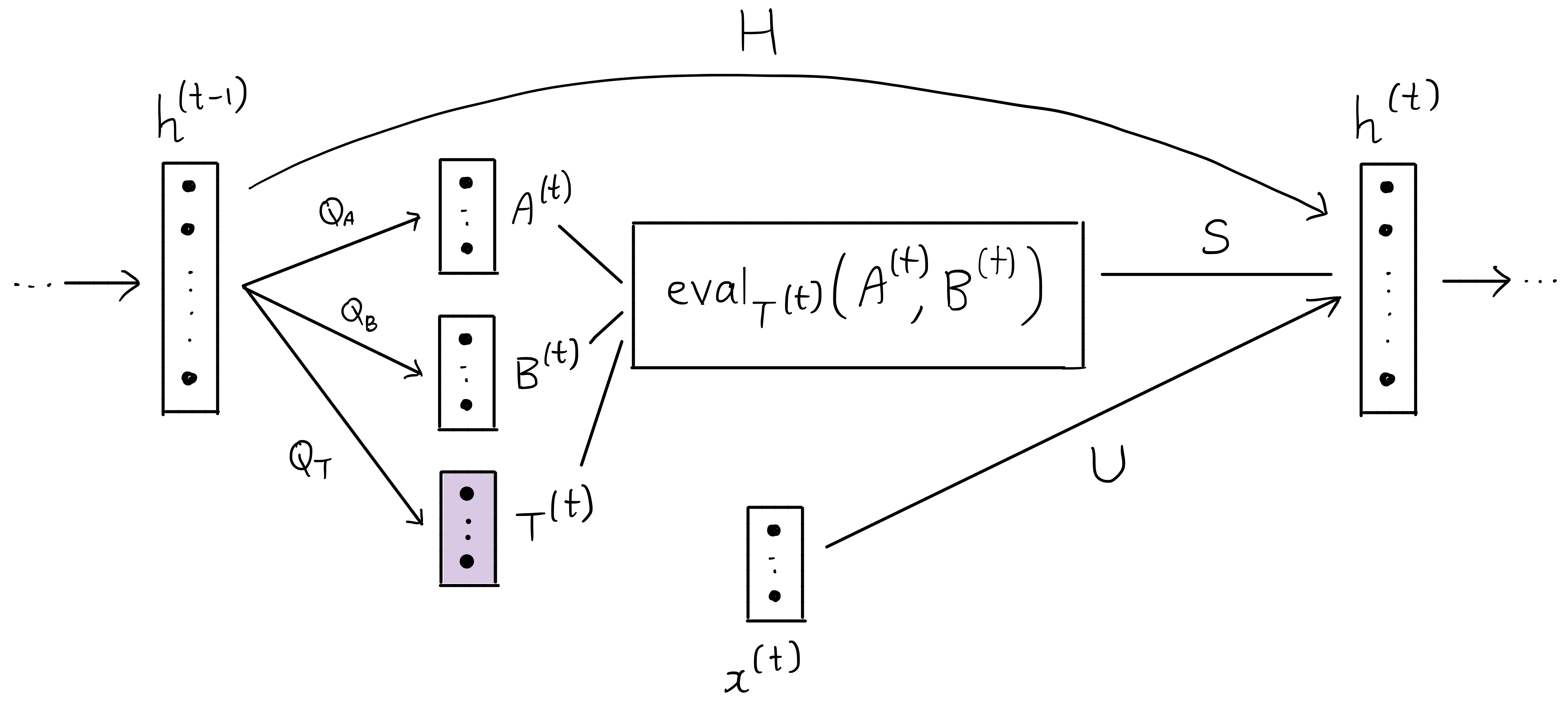
(matrices of polynomials in the variables a, a')

$$\mathrm{eval}_{[\![t_0]\!]}(A,B) = A \qquad \mathrm{eval}_{[\![t_{001}]\!]}(A,B) = AAB$$



Upshot: proofs/programs to vectors

Our model: RNN controller + differentiable "linear logic" module



$T^{(t)} \in [\![\mathbf{bint}]\!]$ $\quad H, S, U, Q_A, Q_B, Q_T$ are matrices of weights

$$h^{(t)} = \sigma\Big(S\,\mathrm{eval}_{T^{(t)}}(A^{(t)}, B^{(t)}) + Hh^{(t-1)} + Ux^{(t)}\Big)$$

$$A^{(t)} = \sigma\Big(Q_A h^{(t-1)}\Big) \qquad B^{(t)} = \sigma\Big(Q_B h^{(t-1)}\Big) \qquad T^{(t)} = \sigma\Big(Q_T h^{(t-1)}\Big)$$