Deep Learning

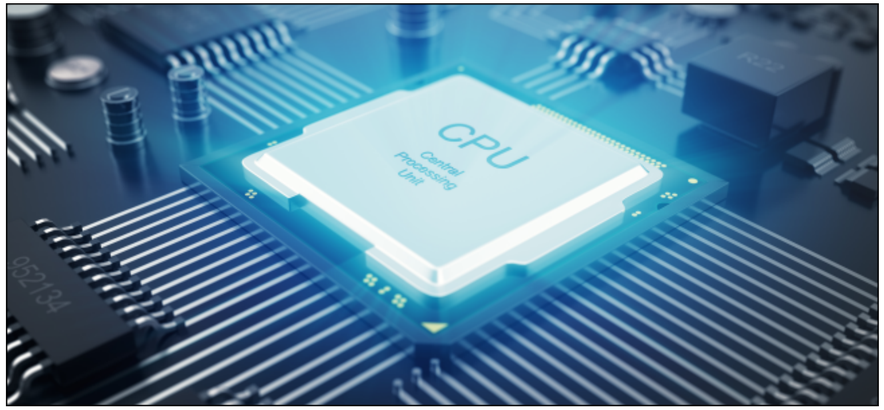
Lecture 23

Differentiable neural computers

Computers vs Neural Networks

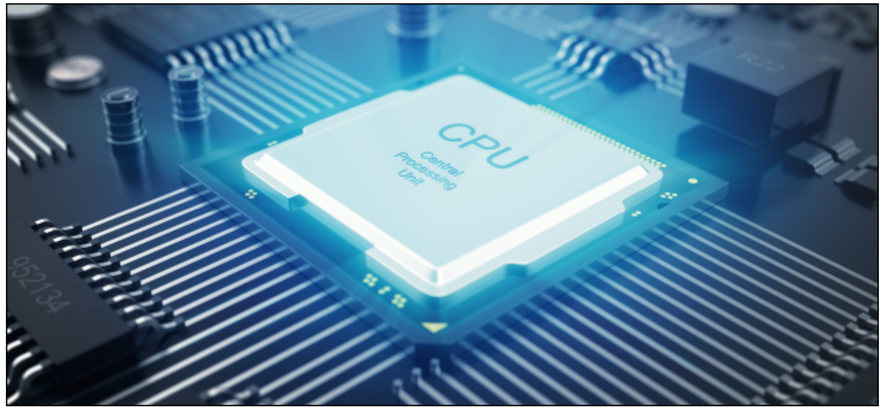
Computers vs Neural Networks

computer

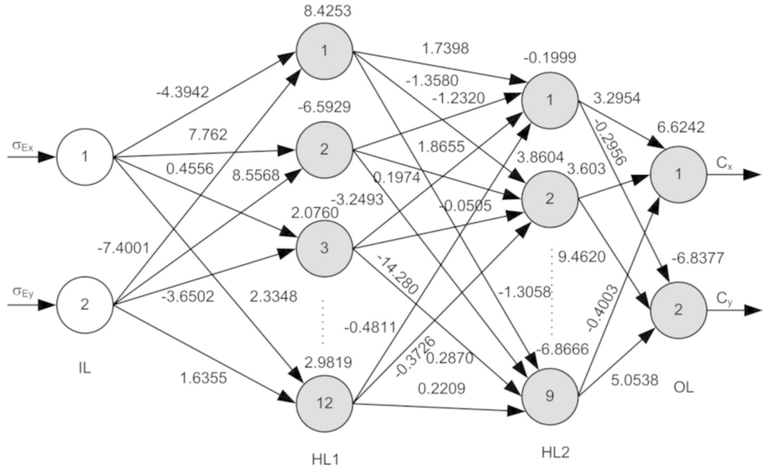


Computers vs Neural Networks

computer

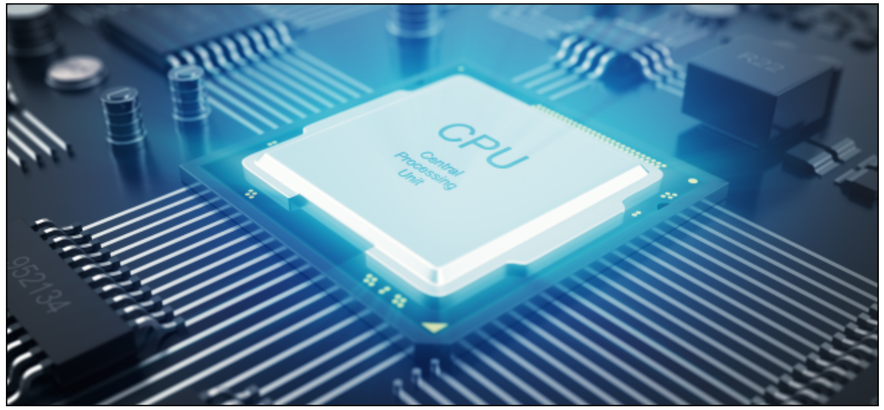


neural network

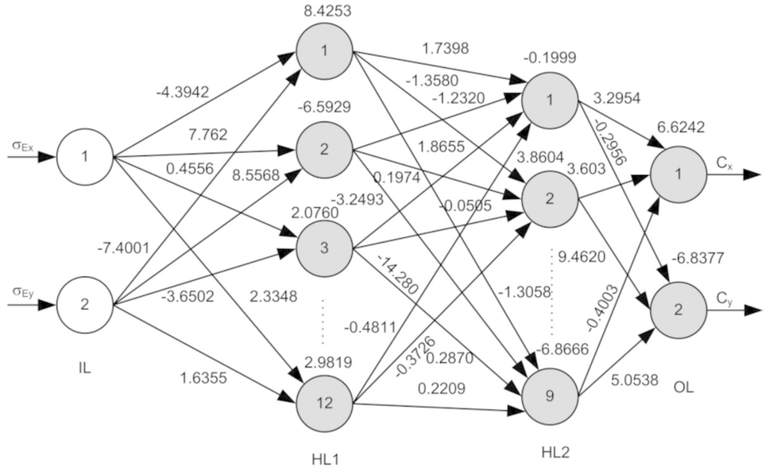


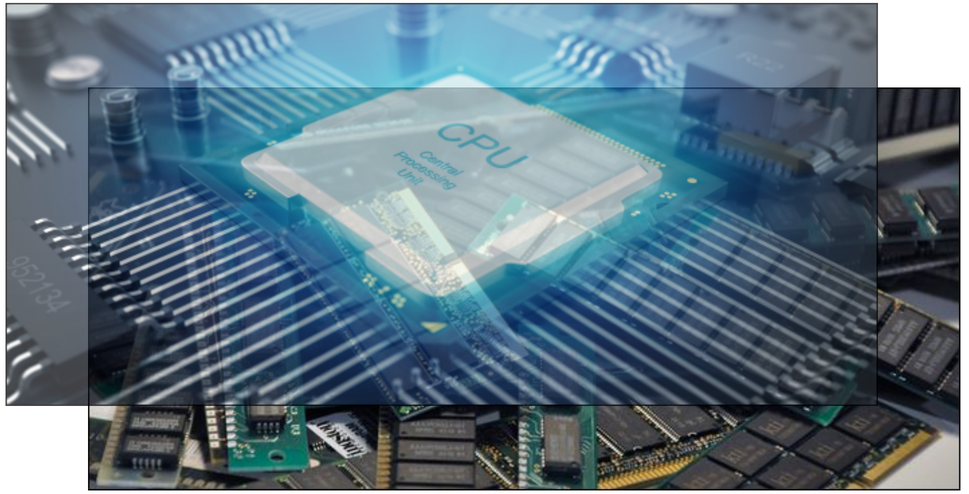
Computers vs Neural Networks

computer



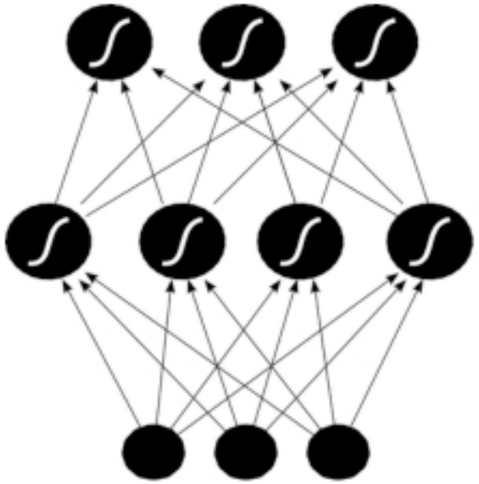
neural network





Basic Idea

Turn neural networks into computers by giving them read-write access to external memory



‘CPU’

+

Memory

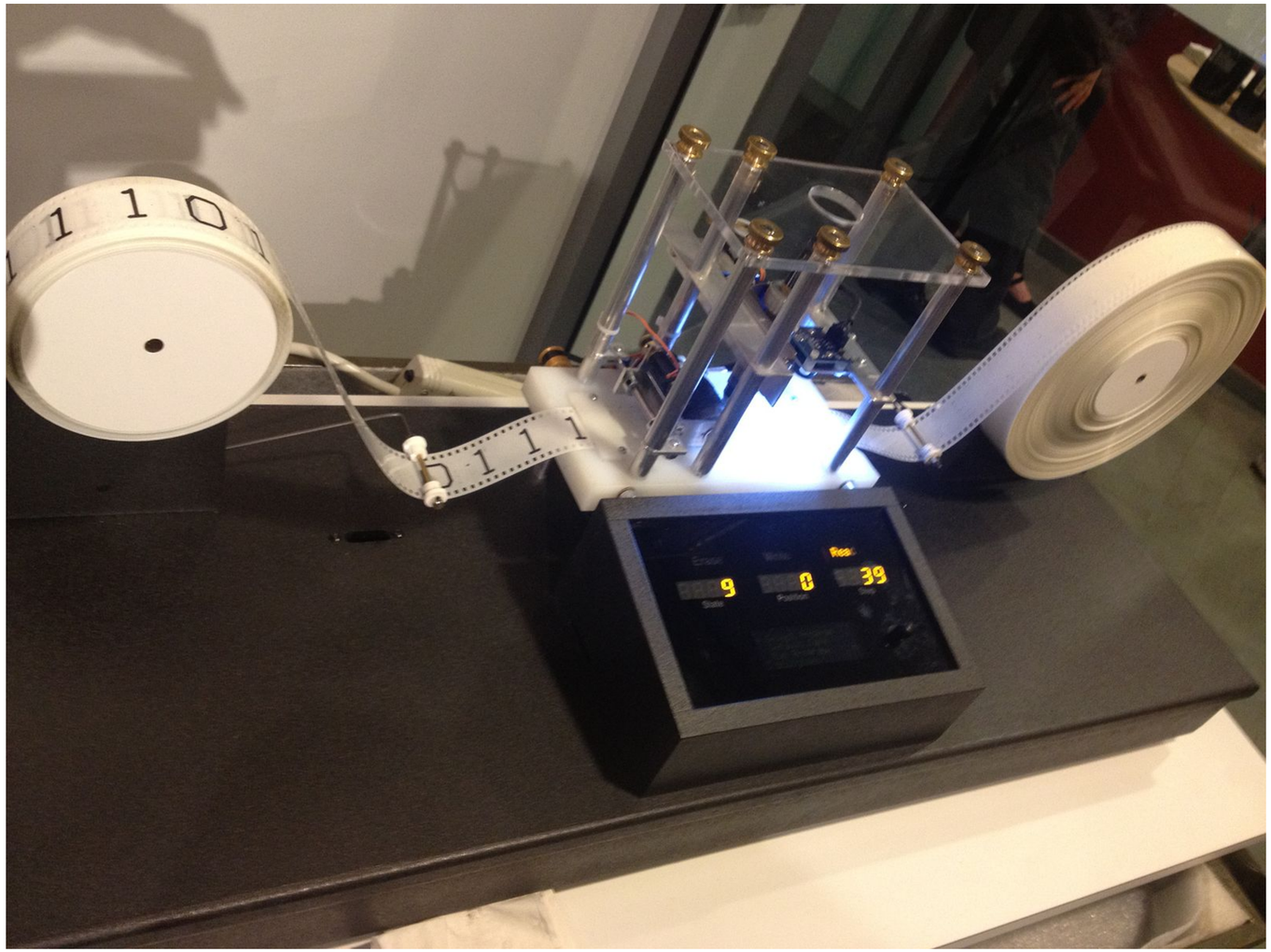
=

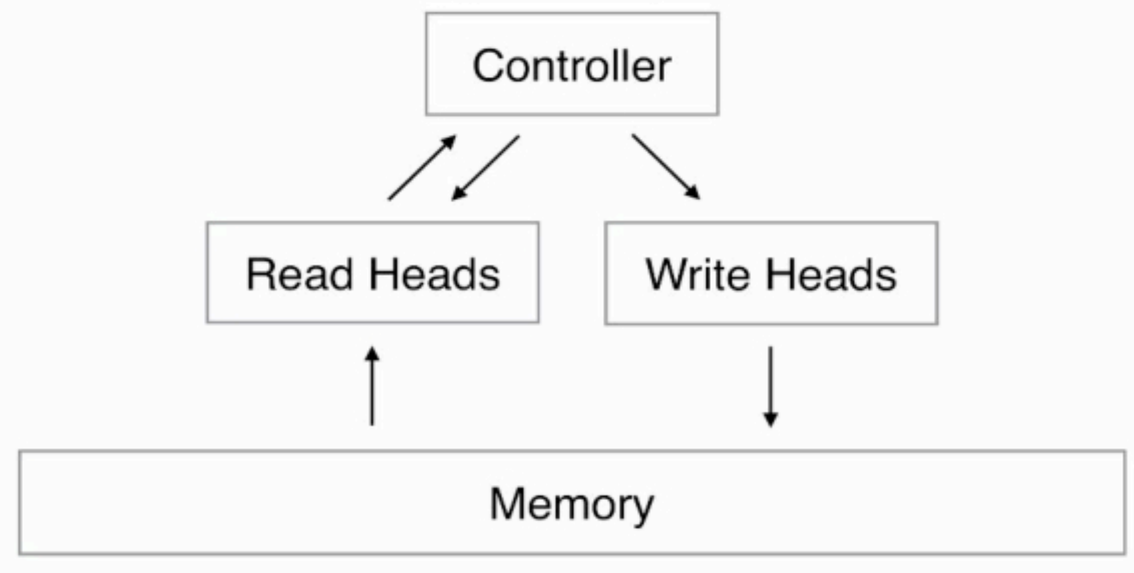
Computer that learns from examples

(network that separates computation from memory)

Adapted from DeepMind

Turing Machine





Adapted from DeepMind

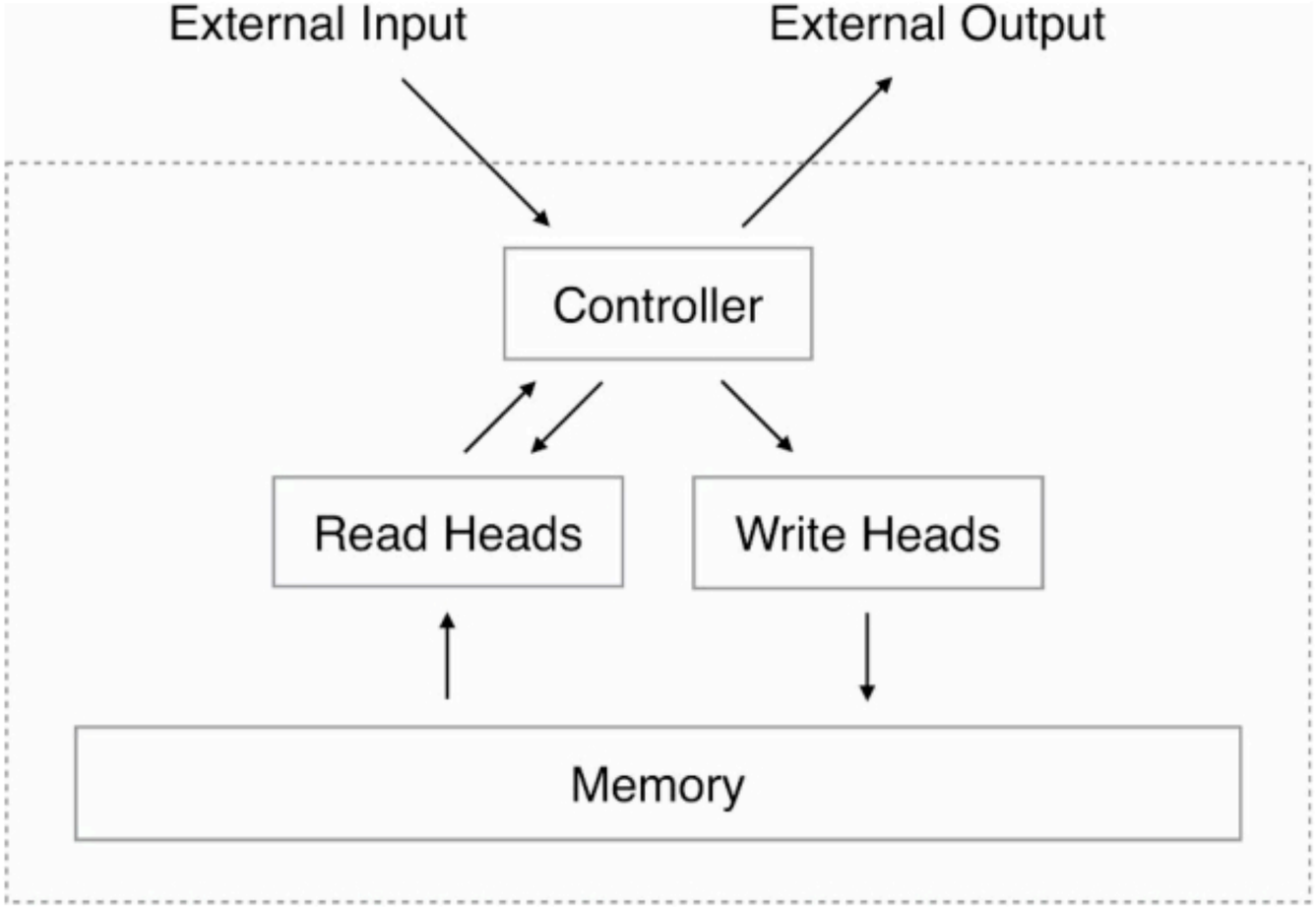
Neural Turing Machine

Graves et. al., arXiv, 2014

The **Controller** is a **neural network** (recurrent or feedforward)

The **Heads** **select** portions of the memory and **read** or **write** to them

The **Memory** is a real-valued **matrix**



Everything is differentiable!

Adapted from DeepMind

Selective Attention

- Focus on parts of memory to read and write to: ***selective attention model***
- Use controller outputs to parameterize distribution (*weightings*) over rows (*locations*) in memory matrix
- The weightings are defined by three main attention mechanisms: **content**, **memory allocation** and **temporal order**
- The controller ***interpolates*** among these mechanisms using scalar gates

Adapted from DeepMind

Searching by Content

Controller emits key vector ***k*** which is compared to each memory location **M**[i] using similarity measure *S* (e.g. **cosine distance**) then normalized with **softmax**. Finds the memories **'closest'** to the key:

$$\mathbf{w}[i] = \frac{\exp\left(\beta S(\mathbf{k}, \mathbf{M}[i])\right)}{\sum_j \exp\left(\beta S(\mathbf{k}, \mathbf{M}[j])\right)}$$

Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et. al. (2014)

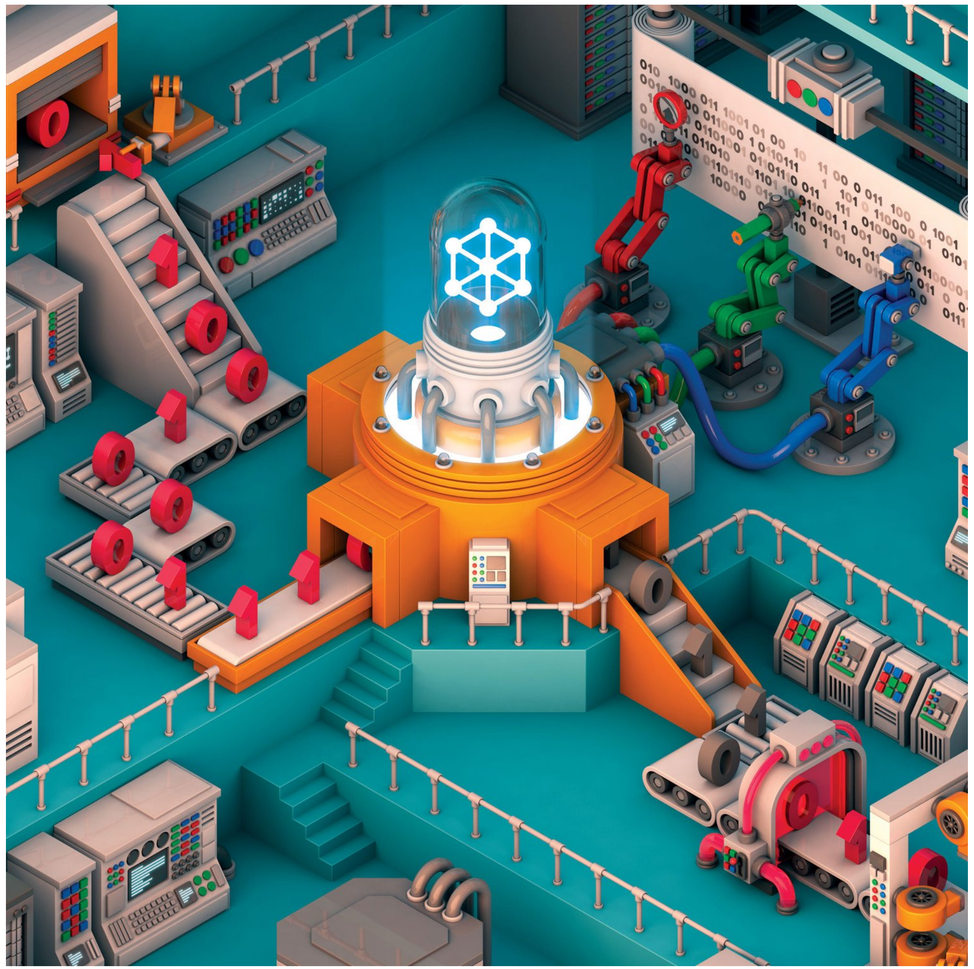
Memory Networks, Weston et. al. (2014)

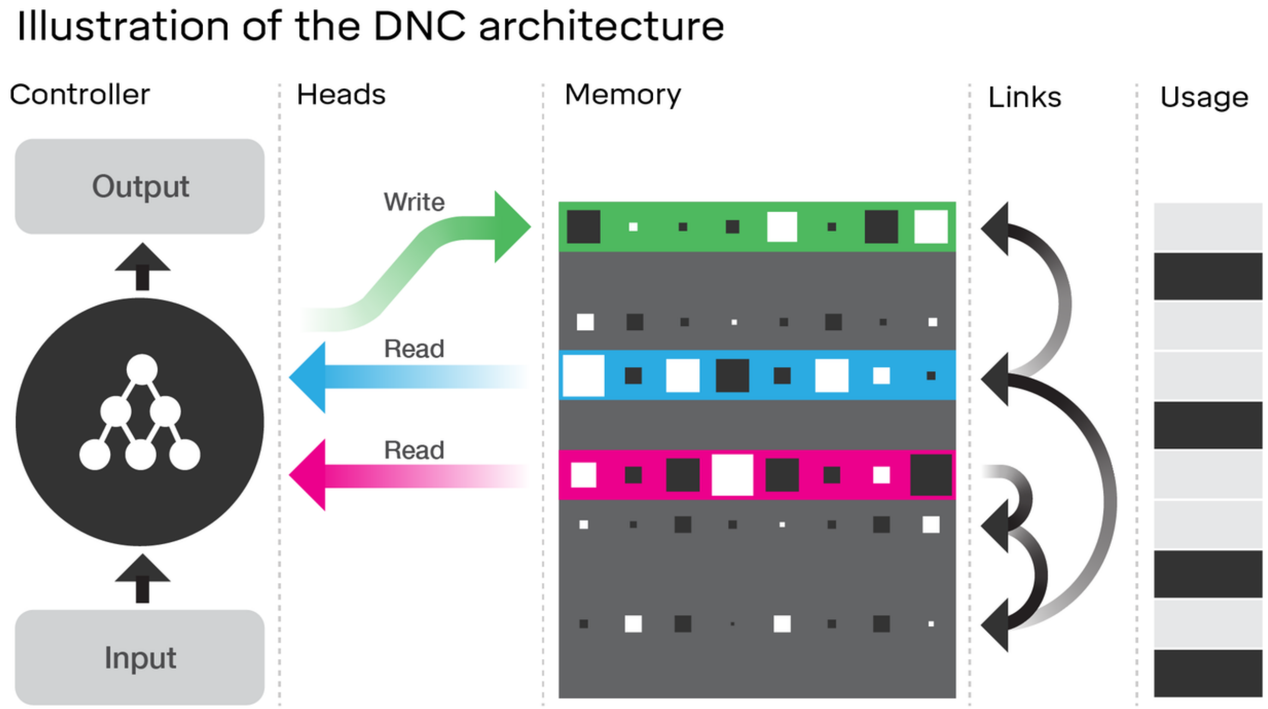
Neural Turing Machines, Graves et. al. (2014)

Adapted from DeepMind

Differentiable Neural Computers

Graves et. al., Nature, 2016





Adapted from DeepMind

Differentiable Neural Computers

Graves et. al., Nature, 2016

- NTMs only ‘allocate’ memory in contiguous blocks => memory management problems
- DNC defines a differentiable *free list* tracking the *usage* ($\boldsymbol{u}_t$) of each memory location
- Usage is automatically *increased* after each write ($\boldsymbol{w}^{w}{}_t$) and optionally *decreased* after each read ($\boldsymbol{w}^{r,i}{}_t$) by *free gates* ($f^{i}{}_t$)

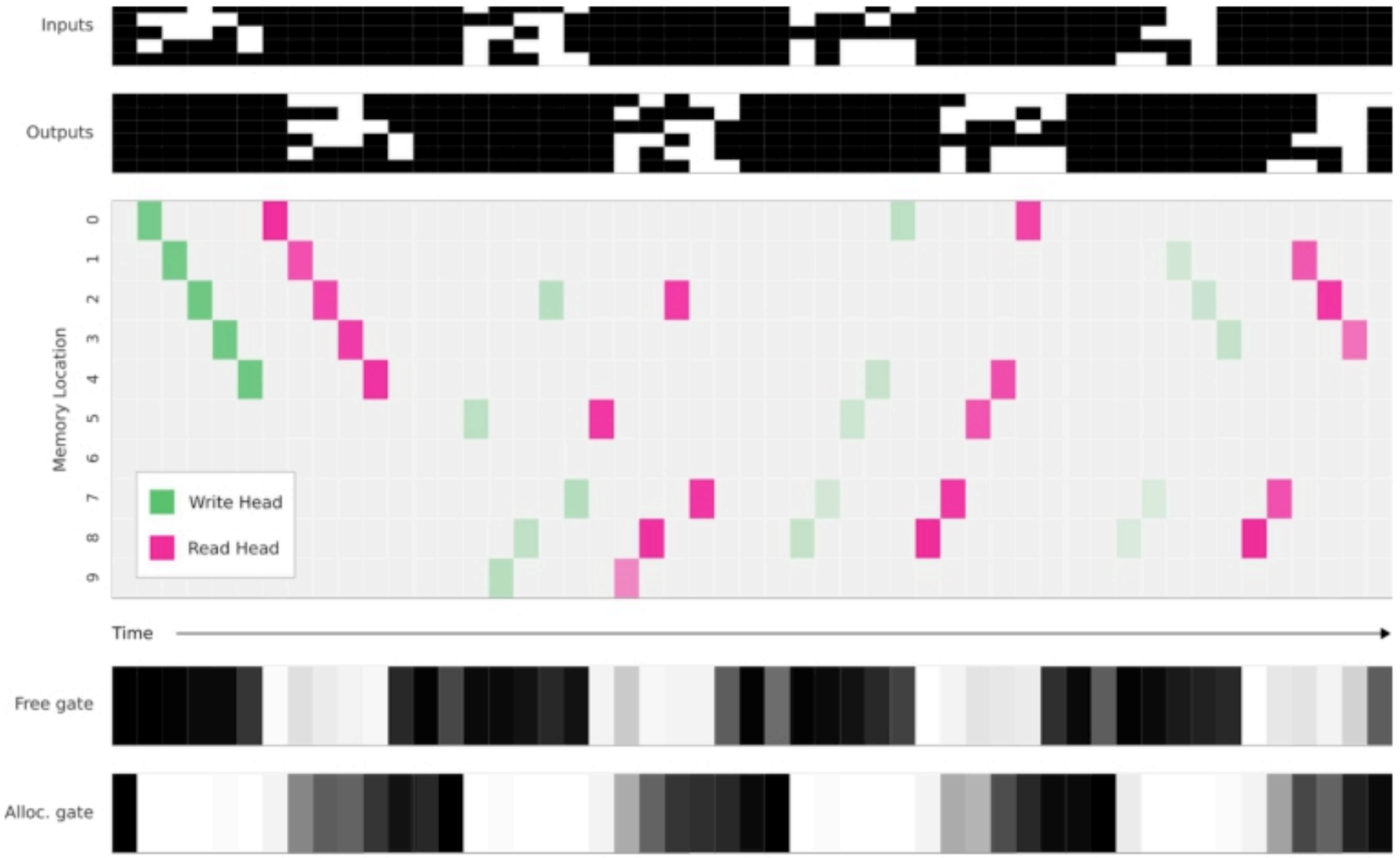
$$\boldsymbol{u}_t = (\boldsymbol{u}_{t-1} + \mathbf{w}^{\mathrm{w}}_{t-1} - \boldsymbol{u}_{t-1} \circ \mathbf{w}^{\mathrm{w}}_{t-1}) \circ \prod_{i=1}^{R} \left(\mathbf{1} - f^{i}_{t} \mathbf{w}^{\mathrm{r},i}_{t-1}\right)$$

- The controller then uses an *allocation gate* ($g^{a}{}_t$) to interpolate between writing to a newly allocated ($\boldsymbol{a}_t$) location, or an existing one found by content ($\boldsymbol{c}^{w}{}_t$)

$$\mathbf{w}^{w}_{t} = g^{w}_{t}\left(g^{a}_{t}\mathbf{a}_t + \left(1 - g^{a}_{t}\right)\mathbf{c}^{w}_{t}\right)$$

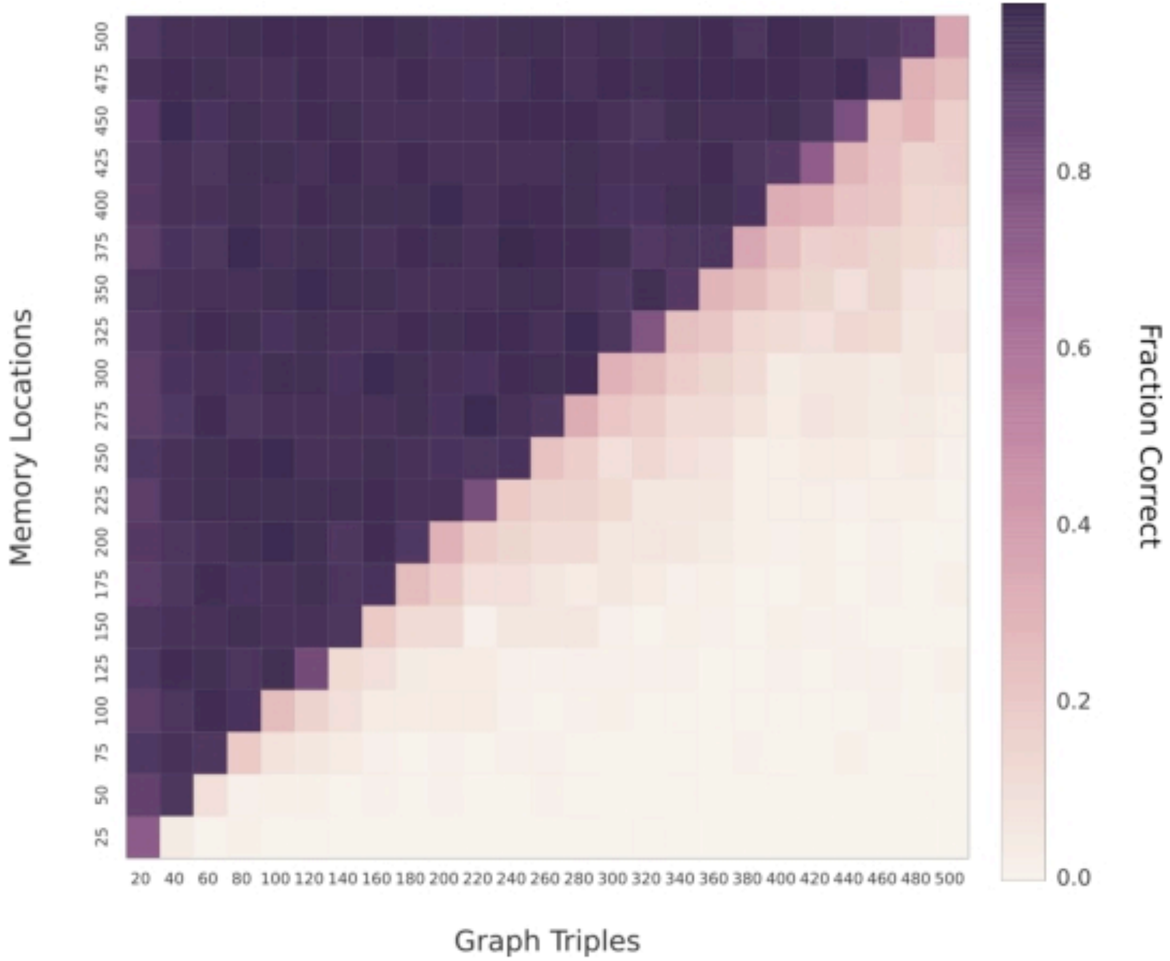
Adapted from DeepMind

DNC: Memory Allocation Test



Adapted from DeepMind

DNC: Memory Resizing Test



Adapted from DeepMind

DNC: Searching By Time

Graves et. al., Nature, 2016

- NTM able to retrieve memories in order of their *index* but not in order in which they were written
- Preserving temporal order is necessary for many tasks (e.g. sequence of instructions) and appears to play an important role in human cognition
- Want DNC to iterate through memories in the order they were written (or rewritten)
- A *precedence weighting* ($\boldsymbol{p}_t$) keeps track of which locations were most recently written to:

$$\boldsymbol{p}_t = \left(1 - \sum_i \boldsymbol{w}_t^{\mathrm{w}}[i]\right)\boldsymbol{p}_{t-1} + \boldsymbol{w}_t^{\mathrm{w}}$$

Adapted from DeepMind

DNC: Searching By Time

Graves et. al., Nature, 2016

$\boldsymbol{p}_t$ used to update a *temporal link matrix* (L_t), defining the order locations were written in:

$$L_t[i,j] = (1 - \boldsymbol{w}_t^{\mathrm{w}}[i] - \boldsymbol{w}_t^{\mathrm{w}}[j])L_{t-1}[i,j] + \boldsymbol{w}_t^{\mathrm{w}}[i]\boldsymbol{p}_{t-1}[j]$$

Controller uses L_t to retrieve the write before (b^i_t) or after (f^i_t) the last read location ($\boldsymbol{w}^{r,i}_{t-1}$), allowing it to iterate forwards or backwards in time

$$\boldsymbol{b}_t^i = \hat{L}_t^\top \boldsymbol{w}_{t-1}^{\mathrm{r},i} \qquad \boldsymbol{f}_t^i = \hat{L}_t \boldsymbol{w}_{t-1}^{\mathrm{r},i}$$

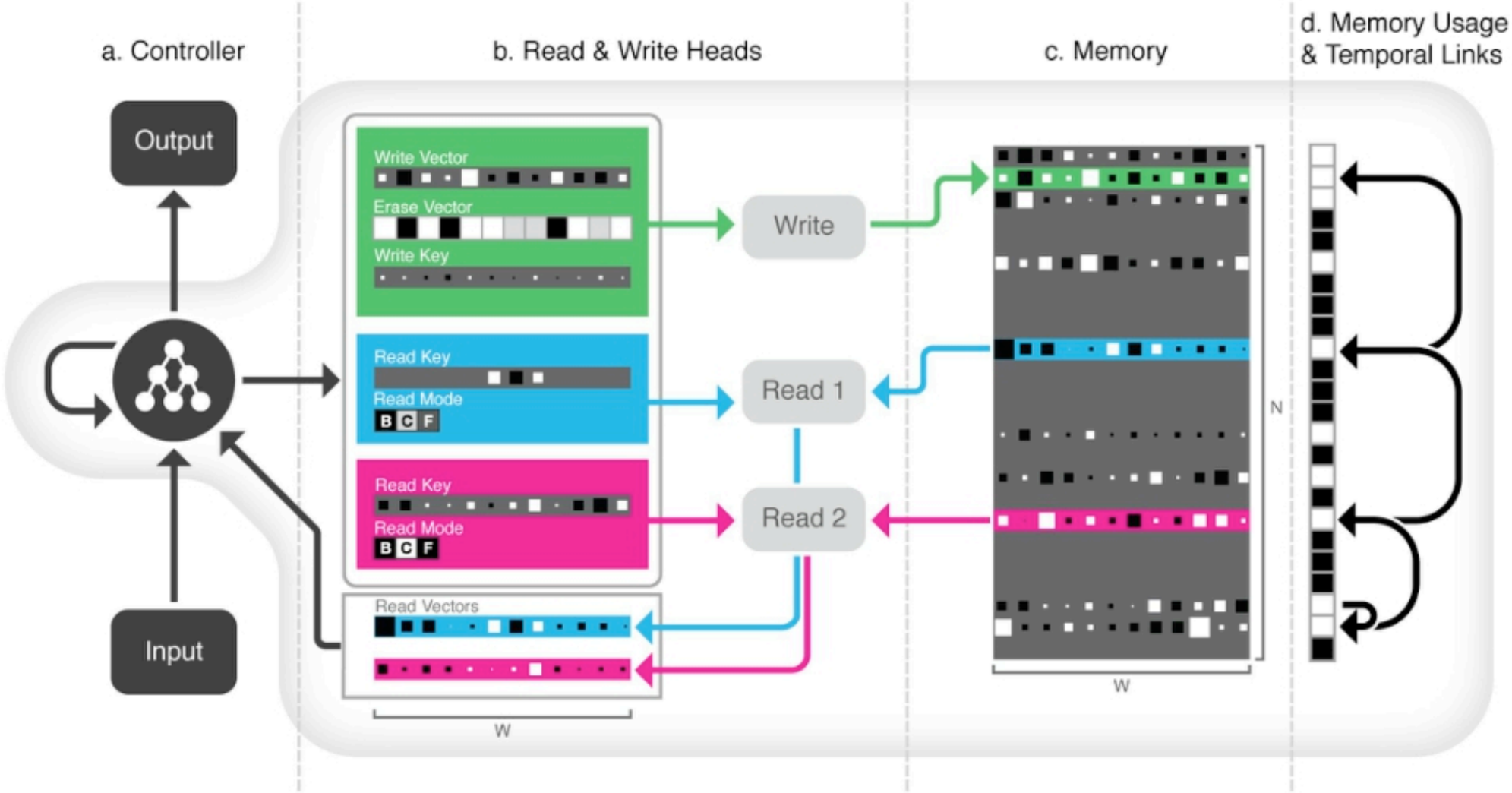
Finally three-way gates ($\boldsymbol{\pi}^i_t$) are used to interpolate among iterating forwards, backwards, or reading by content:

$$\boldsymbol{w}_t^{\mathrm{r},i} = \boldsymbol{\pi}_t^i[1]\boldsymbol{b}_t^i + \boldsymbol{\pi}_t^i[2]\boldsymbol{c}_t^{\mathrm{r},i} + \boldsymbol{\pi}_t^i[3]\boldsymbol{f}_t^i$$

Adapted from DeepMind

DNC: Overall Architecture

Graves et. al., Nature, 2016



Adapted from DeepMind

DNC: learning graphs

Graves et. al., Nature, 2016

- RNNs are great at processing ***sequences*** : *text, audio, time-series…*
- But ***graph-structured*** data is even more general: *maps, molecules, parse-trees, knowledge graphs, social networks…*
- We want DNC to be able to interpret and answer questions about graphs, even if presented in sequential form
- Start with ***explicit*** graphs, but ultimately interested in ***implicit*** graphs: *relations in natural language, scene parsing, agent's surroundings…*

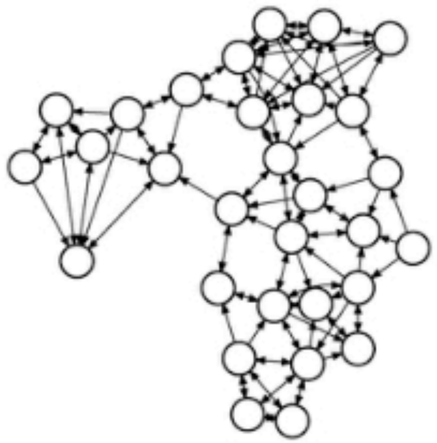
Adapted from DeepMind

DNC: Graph Experiments

Graves et. al., Nature, 2016

Training Data

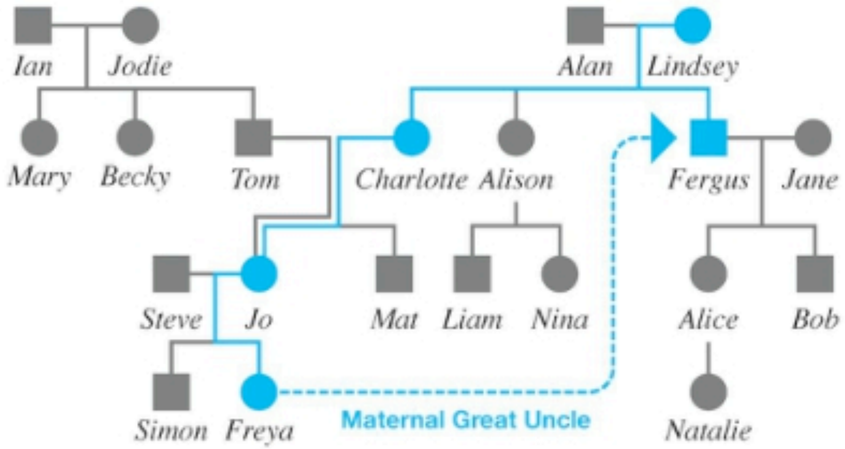
a. Random Graph



Test Examples

b. London Underground

c. Family Tree



Underground Input:

(OxfordCircus, TottenhamCtRd, Central)
(TottenhamCtRd, OxfordCircus, Central)
(BakerSt, Marylebone, Circle)
(BakerSt, Marylebone, Bakerloo)
(BakerSt, OxfordCircus, Bakerloo)

...

(LeicesterSq, CharingCross, Northern)
(TottenhamCtRd, LeicesterSq, Northern)
(OxfordCircus, PicadillyCircus, Bakerloo)
(OxfordCircus, NottingHillGate, Central)
(OxfordCircus, Euston, Victoria)

- 84 edges in total

Traversal Question:

(OxfordCircus, _, Central), (_, _, Circle)
(_, _, Circle), (_, _, Circle),
(_, _, Bakerloo), (_, _, Victoria),
(_, _, Victoria), (_, _, Circle),
(_, _, Bakerloo), (_, _, Jubilee)

Answer:

(OxfordCircus, NottingHillGate, Central)
(NottingHillGate, Paddington, Circle)
...
(Embankment, Waterloo, Bakerloo)
(Waterloo, GreenPark, Jubilee)

Shortest Path Question:

(Moorgate, PicadillyCircus, _)

Answer:

(Moorgate, Bank, Northern)
(Bank, Holborn, Central)
(Holborn, LeicesterSq, Picadilly)
(LeicesterSq, PicadillyCircus, Picadilly)

Family Tree Input:

(Charlotte, Alan, Father)
(Simon, Steve, Father)
(Steve , Simon, Son1)
(Melanie, Alison, Mother)
(Lindsey, Fergus, Son1)

...

(Bob, Jane, Mother)
(Natalie, Alice, Mother)
(Mary, Ian, Father)
(Jane, Alice, Daughter1)
(Mat, Charlotte, Mother)

- 54 edges in total

Inference Question:

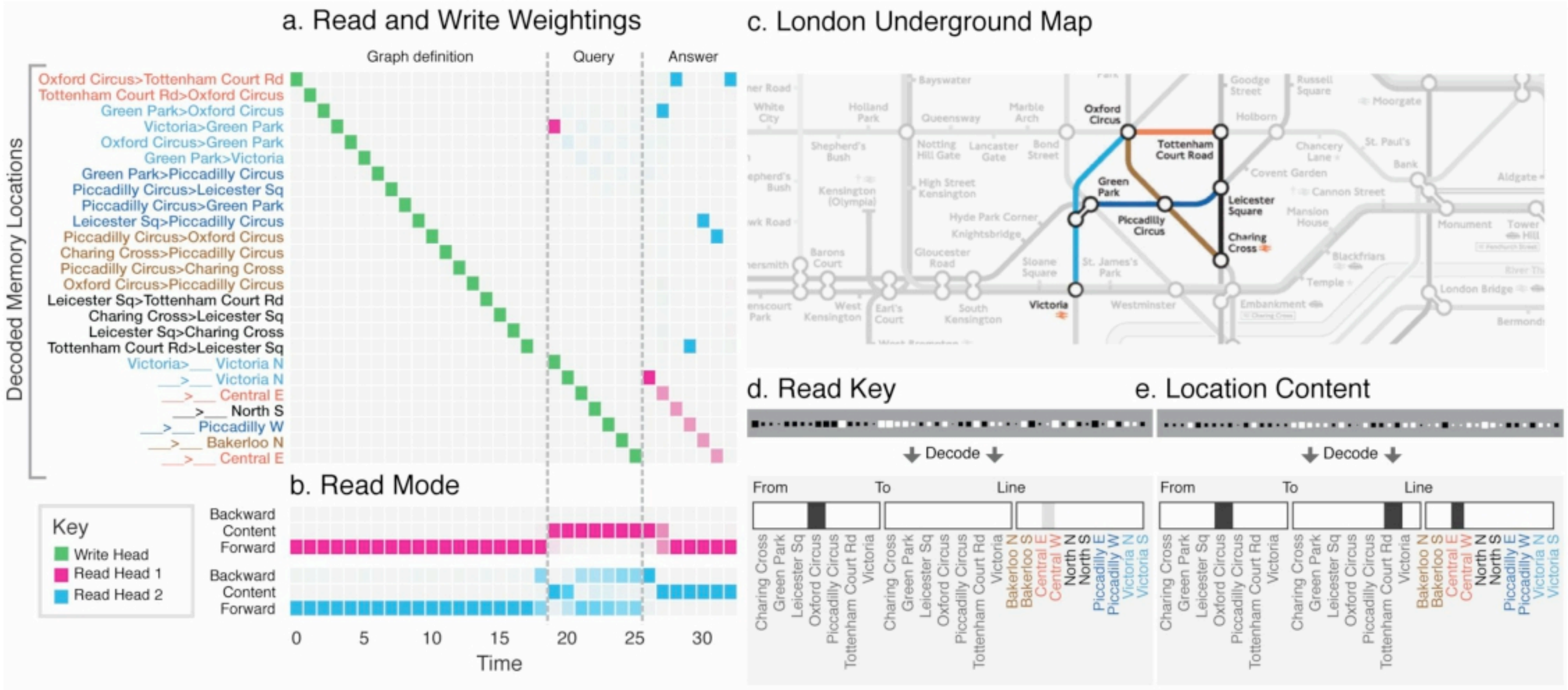
(Freya, _, MaternalGreatUncle)

Answer:

(Freya, Fergus, MaternalGreatUncle)

Adapted from DeepMind

How DNC Reads Graphs



Adapted from DeepMind

bAbI Results

	bAbI Best Results						
Task	LSTM (Joint)	NTM (Joint)	DNC1 (Joint)	DNC2 (Joint)	MemN2N (Joint) [21]	MemN2N (Single) [21]	DMN (Single) [20]
1: 1 supporting fact	24.5	31.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
2: 2 supporting facts	53.2	54.5	1.3	0.4	1.0	**0.3**	1.8
3: 3 supporting facts	48.3	43.9	2.4	**1.8**	6.8	2.1	4.8
4: 2 argument rels.	0.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
5: 3 argument rels.	3.5	0.8	**0.5**	0.8	6.1	0.8	0.7
6: yes/no questions	11.5	17.1	**0.0**	**0.0**	0.1	0.1	**0.0**
7: counting	15.0	17.8	**0.2**	0.6	6.6	2.0	3.1
8: lists/sets	16.5	13.8	**0.1**	0.3	2.7	0.9	3.5
9: simple negation	10.5	16.4	**0.0**	0.2	**0.0**	0.3	**0.0**
10: indefinite knowl.	22.9	16.6	0.2	0.2	0.5	**0.0**	**0.0**
11: basic coreference	6.1	15.2	**0.0**	**0.0**	**0.0**	0.1	0.1
12: conjunction	3.8	8.9	0.1	**0.0**	0.1	**0.0**	**0.0**
13: compound coref.	0.5	7.4	**0.0**	0.1	**0.0**	**0.0**	0.2
14: time reasoning	55.3	24.2	0.3	0.4	**0.0**	0.1	**0.0**
15: basic deduction	44.7	47.0	**0.0**	**0.0**	0.2	**0.0**	**0.0**
16: basic induction	52.6	53.6	52.4	55.1	**0.2**	51.8	0.6
17: positional reas.	39.2	25.5	24.1	**12.0**	41.8	18.6	40.4
18: size reasoning	4.8	2.2	4.0	**0.8**	8.0	5.3	4.7
19: path finding	89.5	4.3	**0.1**	3.9	75.7	2.3	65.5
20: agent motiv.	1.3	1.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
Mean Err. (%)	25.2	20.1	4.3	**3.8**	7.5	4.2	6.4
Failed (err. > 5%)	15	16	**2**	**2**	6	3	**2**

Ask me anything: dynamic memory networks for natural language processing, Kumar et. al. (2015)

End-to-end memory networks, Sukhbaatar et. al. (2015)

Adapted from DeepMind

Differentiable Programming

Deep learning as functional programming

Neural network models are assembled from **building blocks**

and trained with **backpropagation**

Adapted from Atilim Baydin

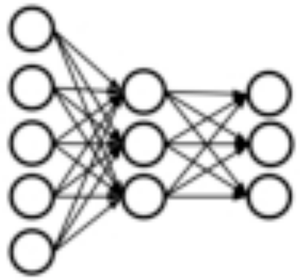
Deep learning as functional programming

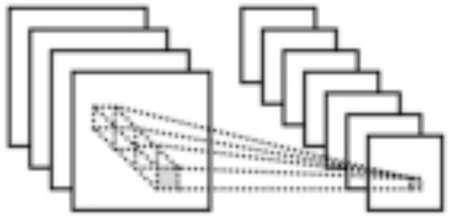
Neural network models are assembled from **building blocks**

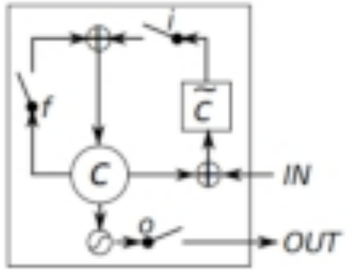
and trained with **backpropagation**

“Basic” networks:

- Feedforward
- Convolutional
- Recurrent

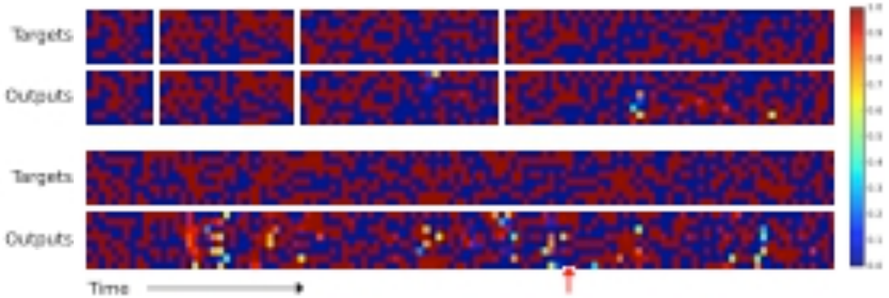






Deep learning as functional programming

"Complex" networks make **algorithmic** elements **continuous and differentiable**
→ enables use in deep learning



NTM on copy task
(Graves et al. 2014)

- Neural Turing Machine (Graves et al., 2014) → can infer algorithms: copy, sort, recall
- Stack-augmented RNN (Joulin & Mikolov, 2015)
- End-to-end memory network (Sukhbaatar et al., 2015)
- Stack, queue, deque (Grefenstette et al., 2015)
- Discrete interfaces (Zaremba & Sutskever, 2015)

Adapted from Atilim Baydin

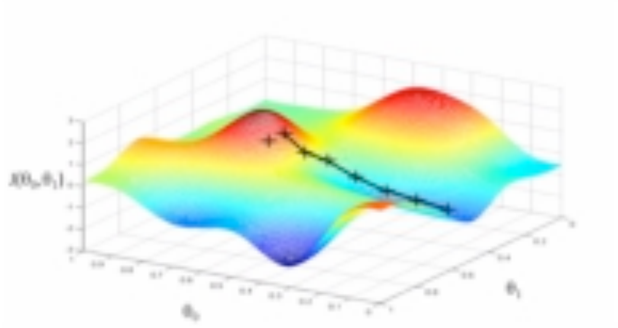
Deep learning as functional programming

Deep learning systems = **“differentiable functional programming”**

Two main characteristics:

- **Differentiability**

  → optimization



- Chained function **composition**

  → successive transformations

  → successive levels of distributed representations (Bengio 2013)

  → the chain rule of calculus propagates derivatives

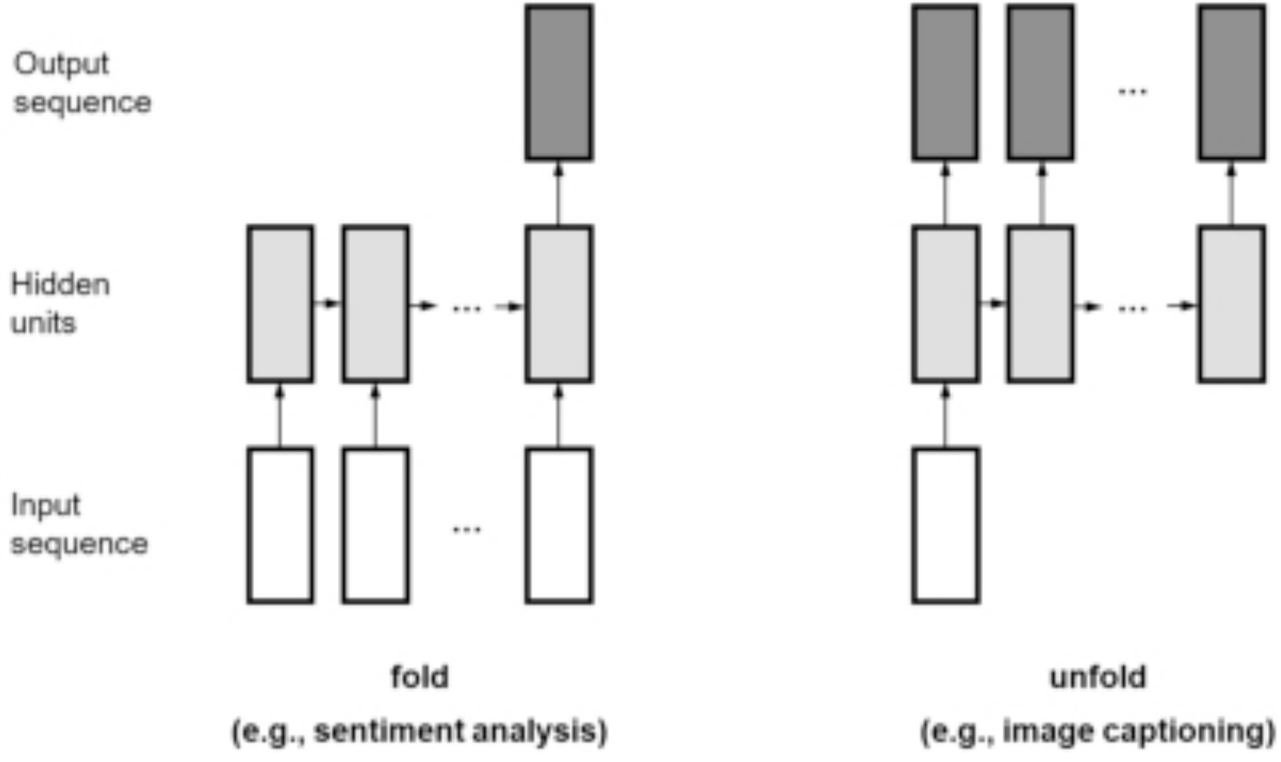
$$g : A \to B$$
$$f : B \to C$$
$$f \circ g : A \to C$$

Adapted from Atilim Baydin

Deep learning as functional programming

In a functional interpretation

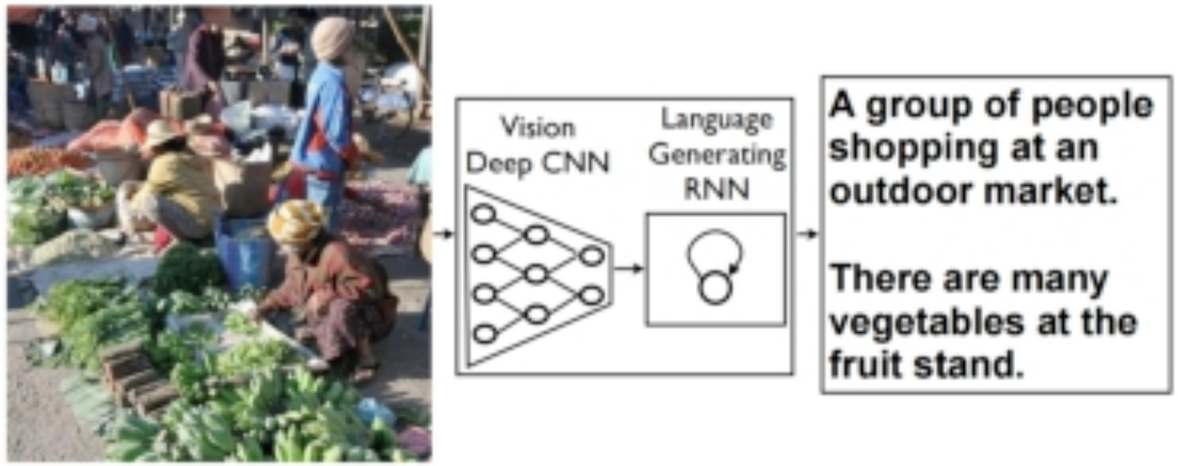
- **Weight-tying** or multiple applications of the same neuron (e.g., ConvNets and RNNs) resemble **function abstraction**
- **Structural patterns** of composition resemble **higher-order functions** (e.g., map, fold, unfold, zip)



Adapted from Atilim Baydin

Deep learning as functional programming

Even when you have **complex compositions**,
differentiability ensures that they can be trained end-to-end with backpropagation

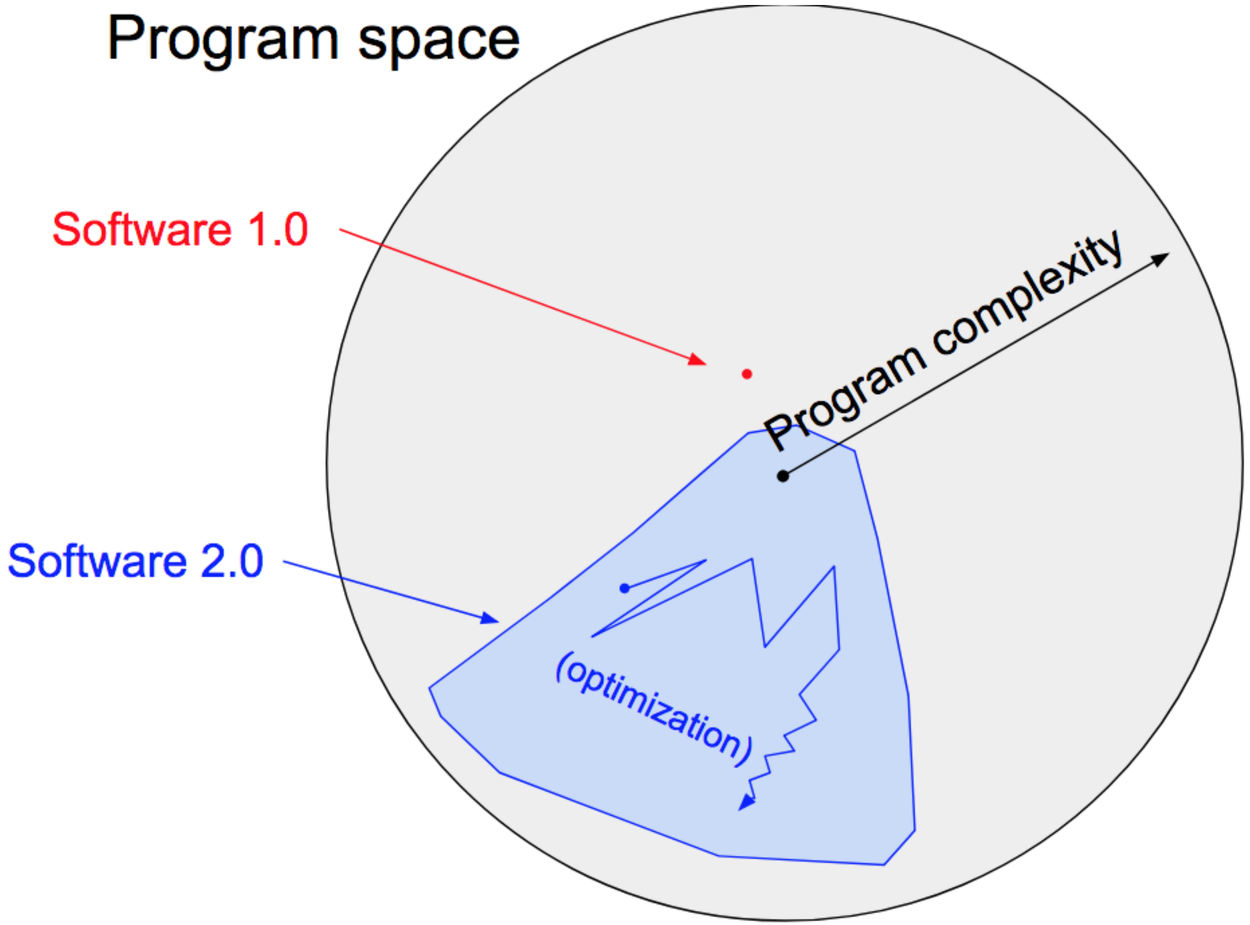


(Vinyals, Toshev, Bengio, Erhan. “Show and tell: a neural image caption generator.” 2014. arXiv:1411.4555)

Adapted from Atilim Baydin

Software 2.0

"Many real-world problems have the property that it is significantly easier to collect data…than to explicitly write the program."

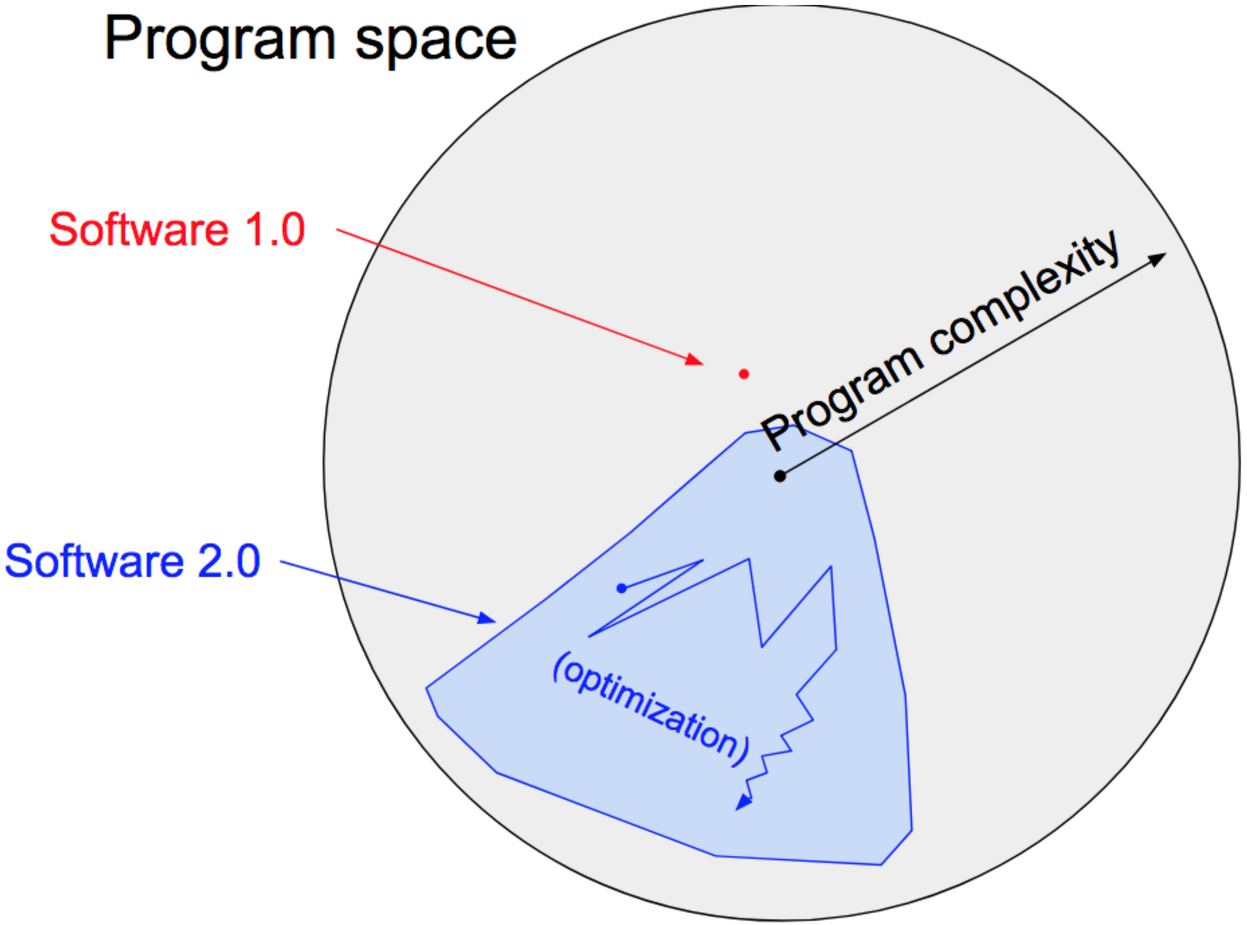


Adapted from Andrei Karpathy

Software 2.0

"Many real-world problems have the property that it is significantly easier to collect data…than to explicitly write the program."

- Specify goal of program

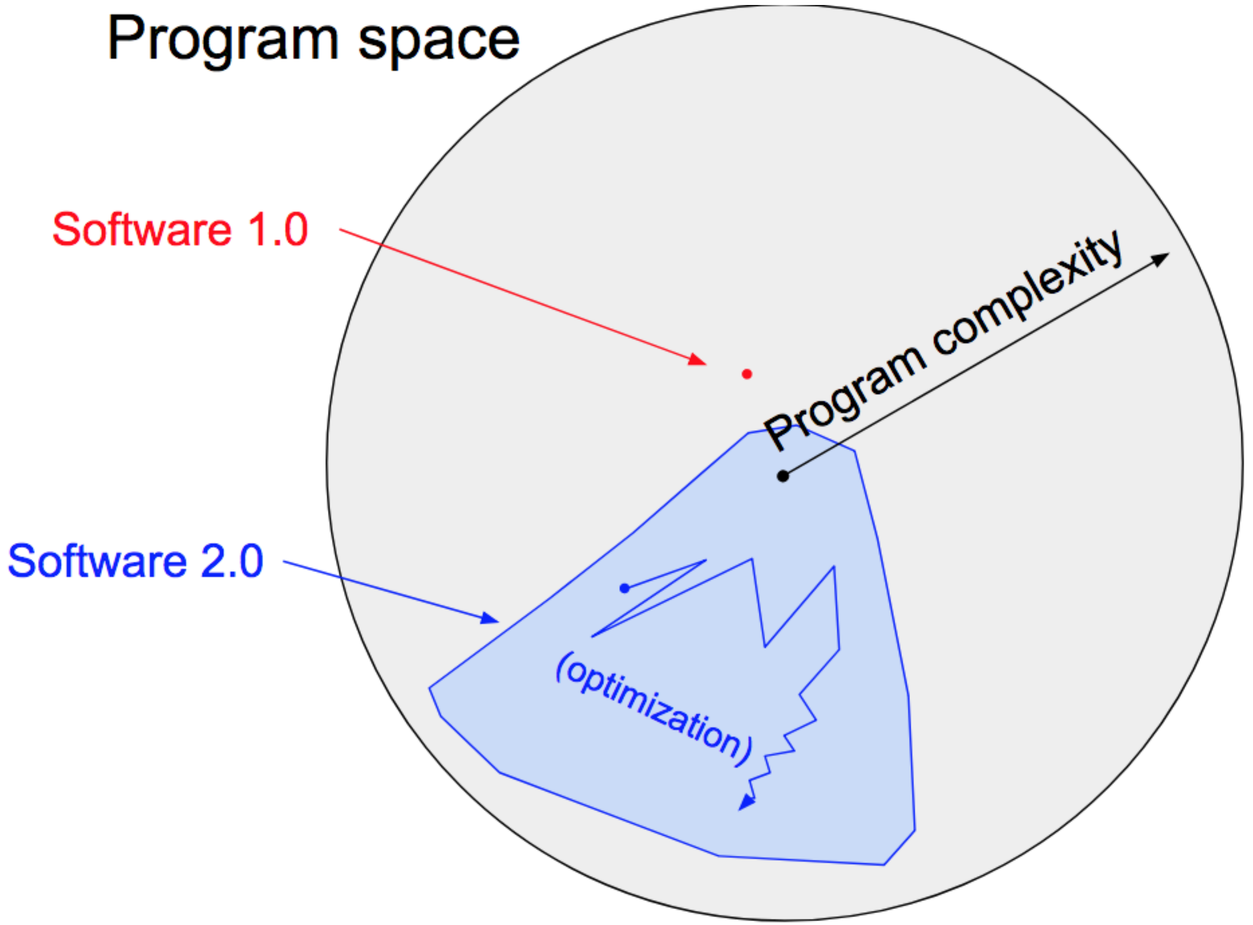


Adapted from Andrei Karpathy

Software 2.0

"Many real-world problems have the property that it is significantly easier to collect data…than to explicitly write the program."

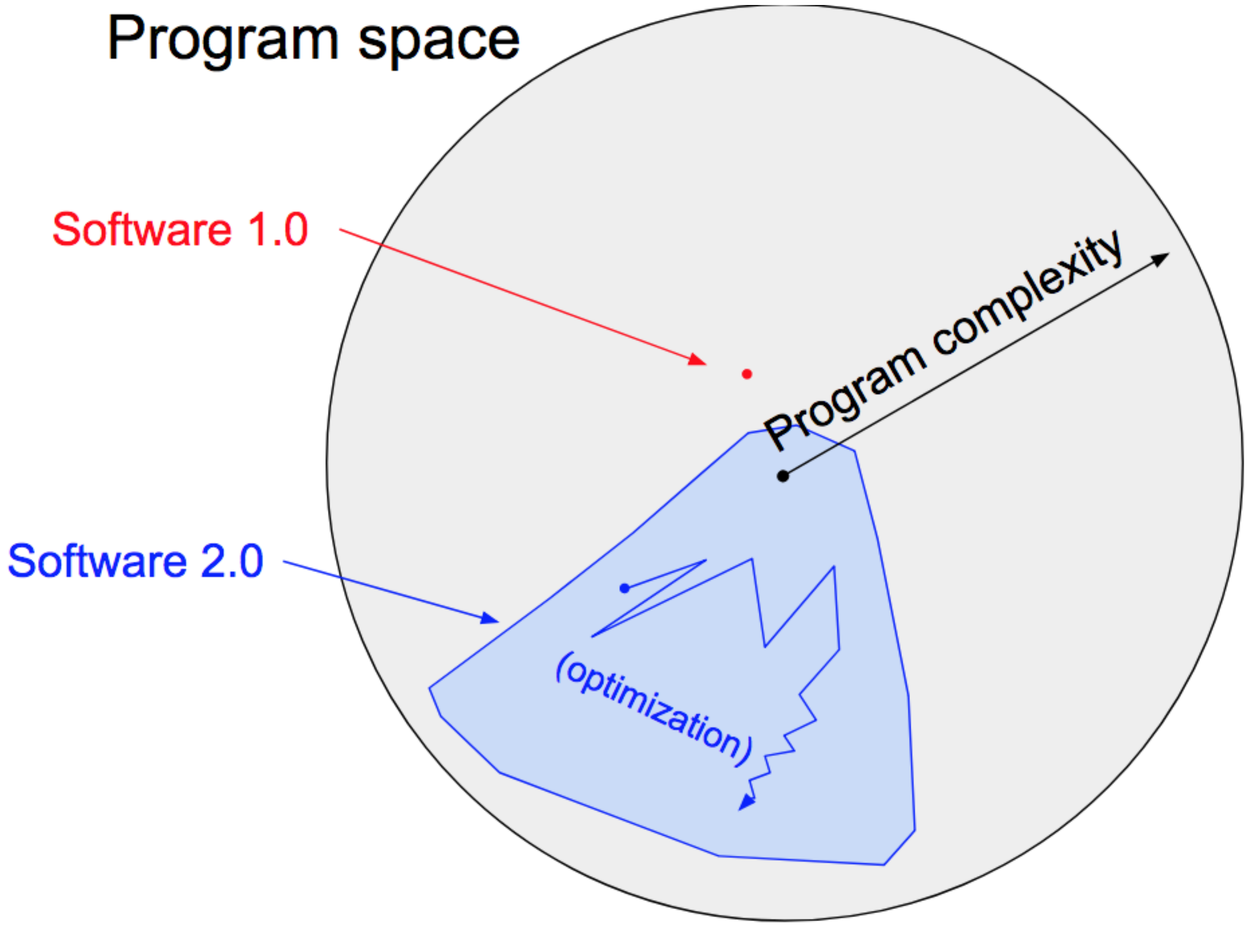
- Specify goal of program
- Write "skeleton" of code (as neural network) that identifies subset of program space to search



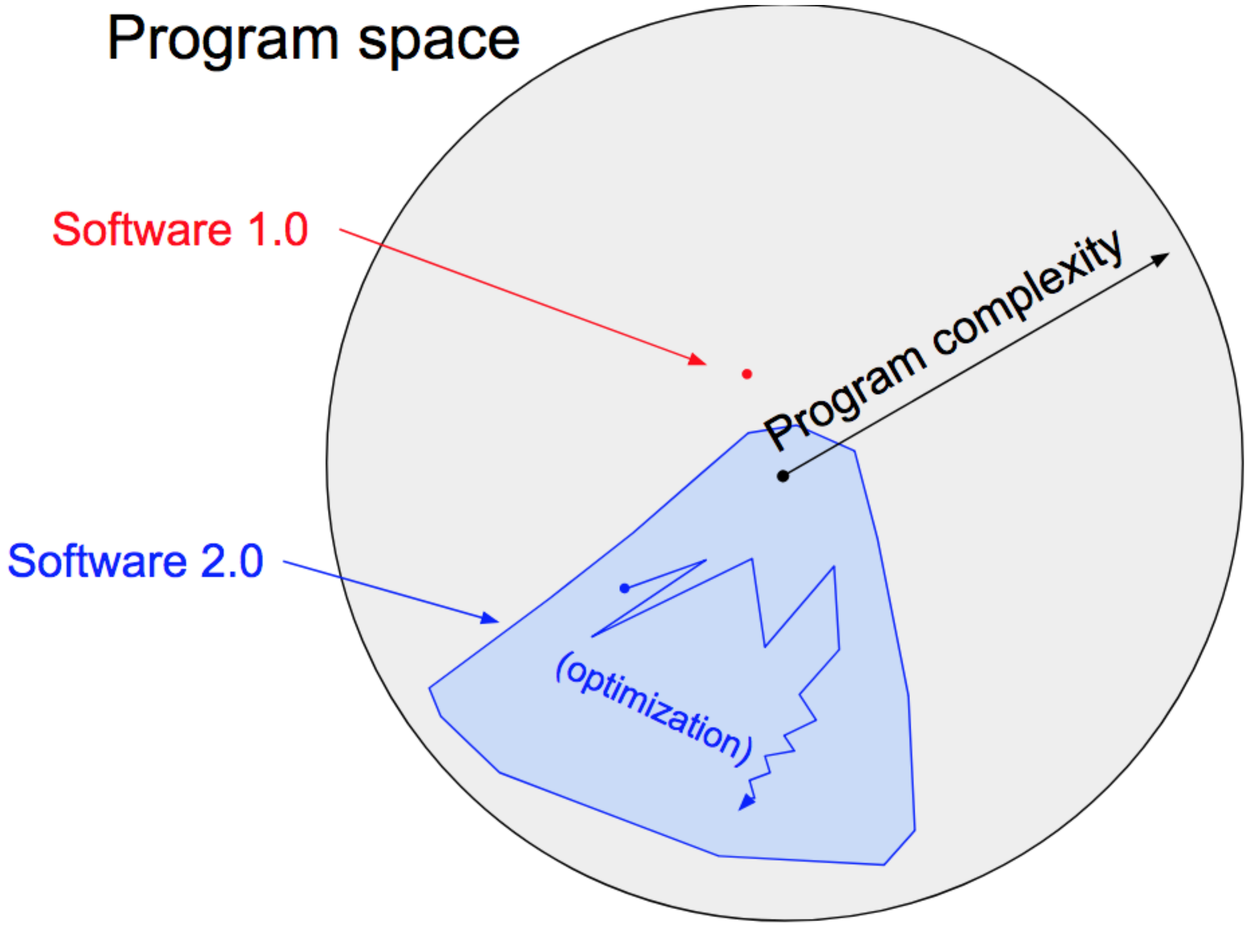
Software 2.0

"Many real-world problems have the property that it is significantly easier to collect data…than to explicitly write the program."

- Specify goal of program
- Write "skeleton" of code (as neural network) that identifies subset of program space to search
- Use computation (e.g. GPUs) to search program space



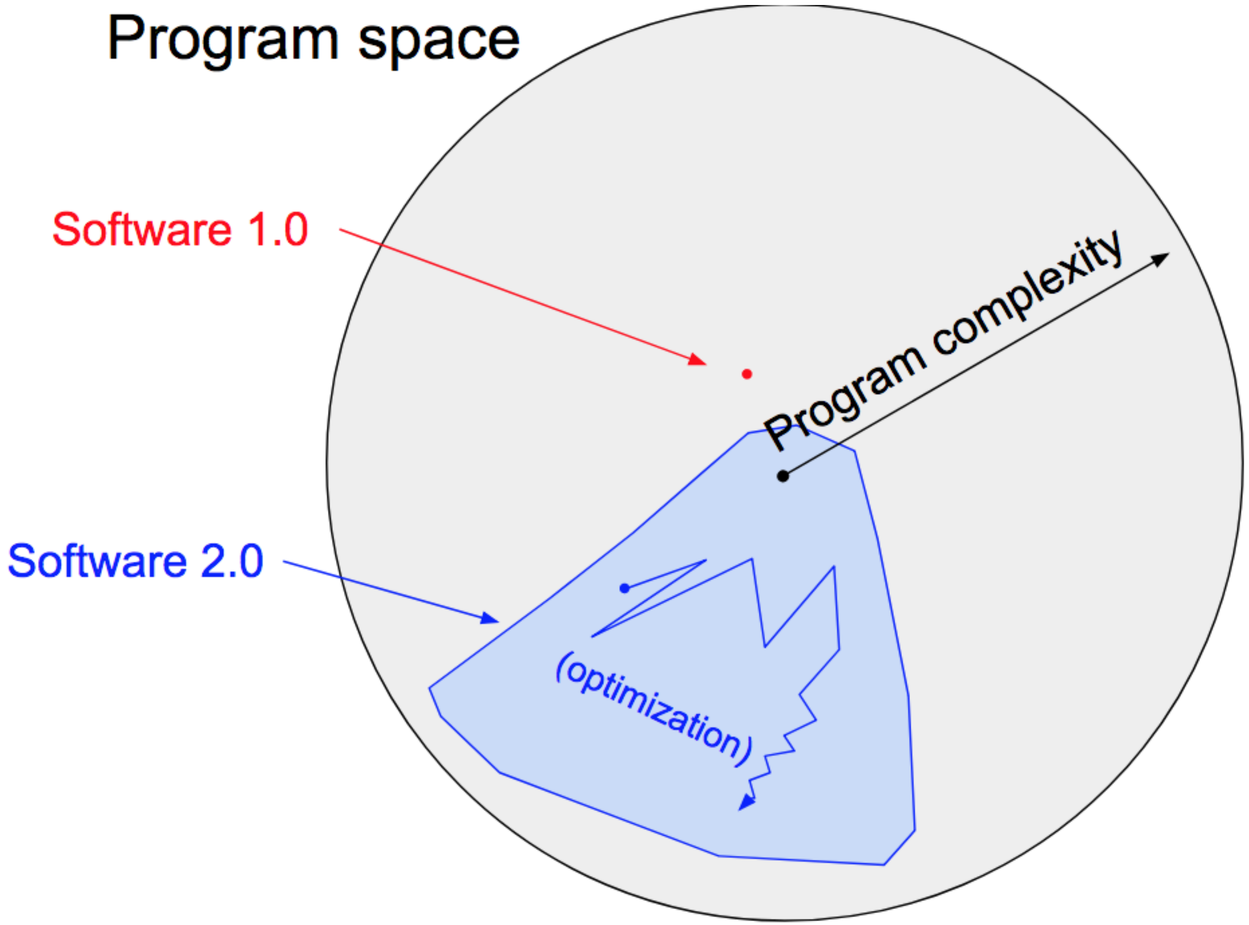
Benefits of Software 2.0



Adapted from Andrei Karpathy

Benefits of Software 2.0

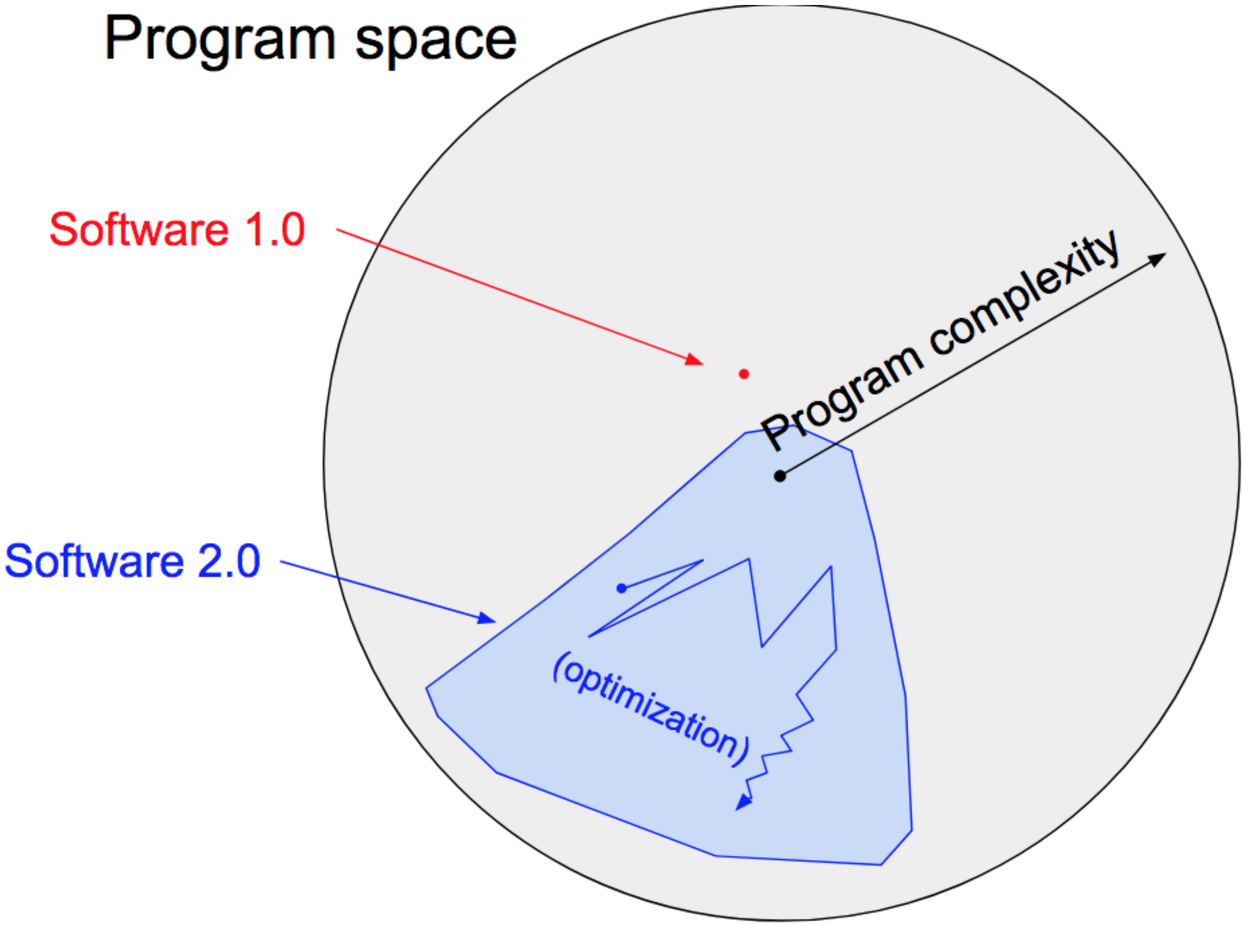
- Computationally homogeneous (matrix multiplication and ReLU)



Adapted from Andrei Karpathy

Benefits of Software 2.0

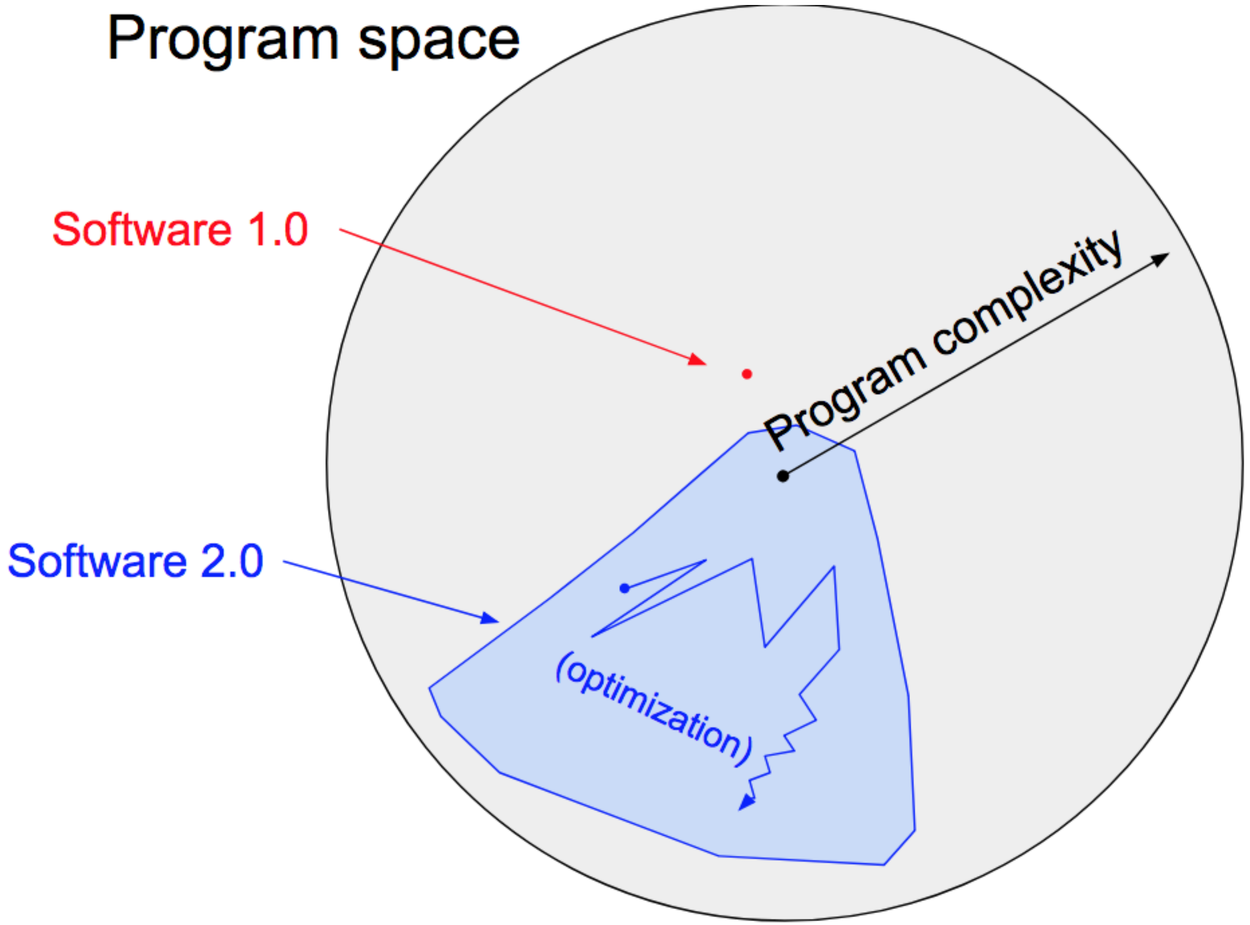
- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware



Adapted from Andrei Karpathy

Benefits of Software 2.0

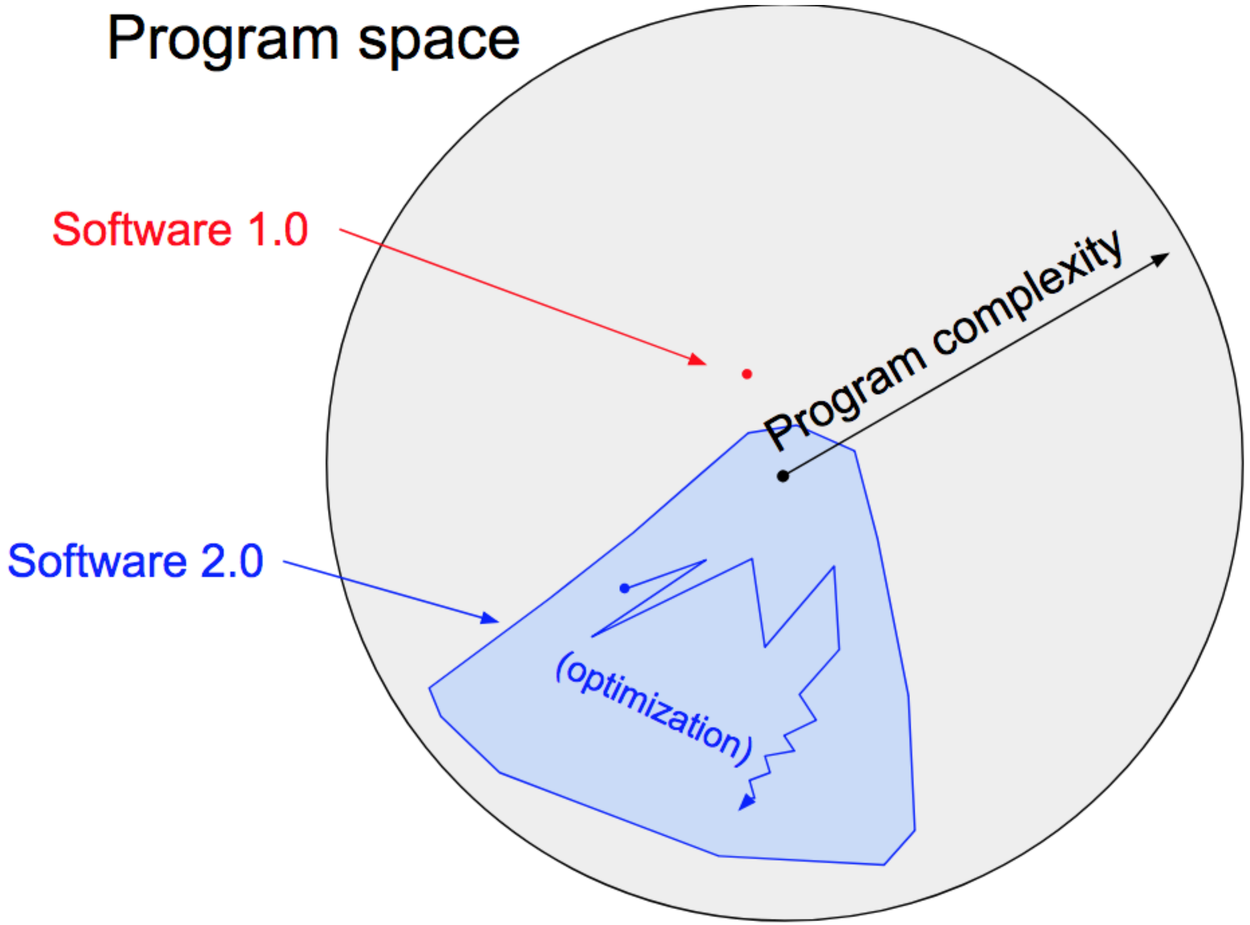
- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time



Adapted from Andrei Karpathy

Benefits of Software 2.0

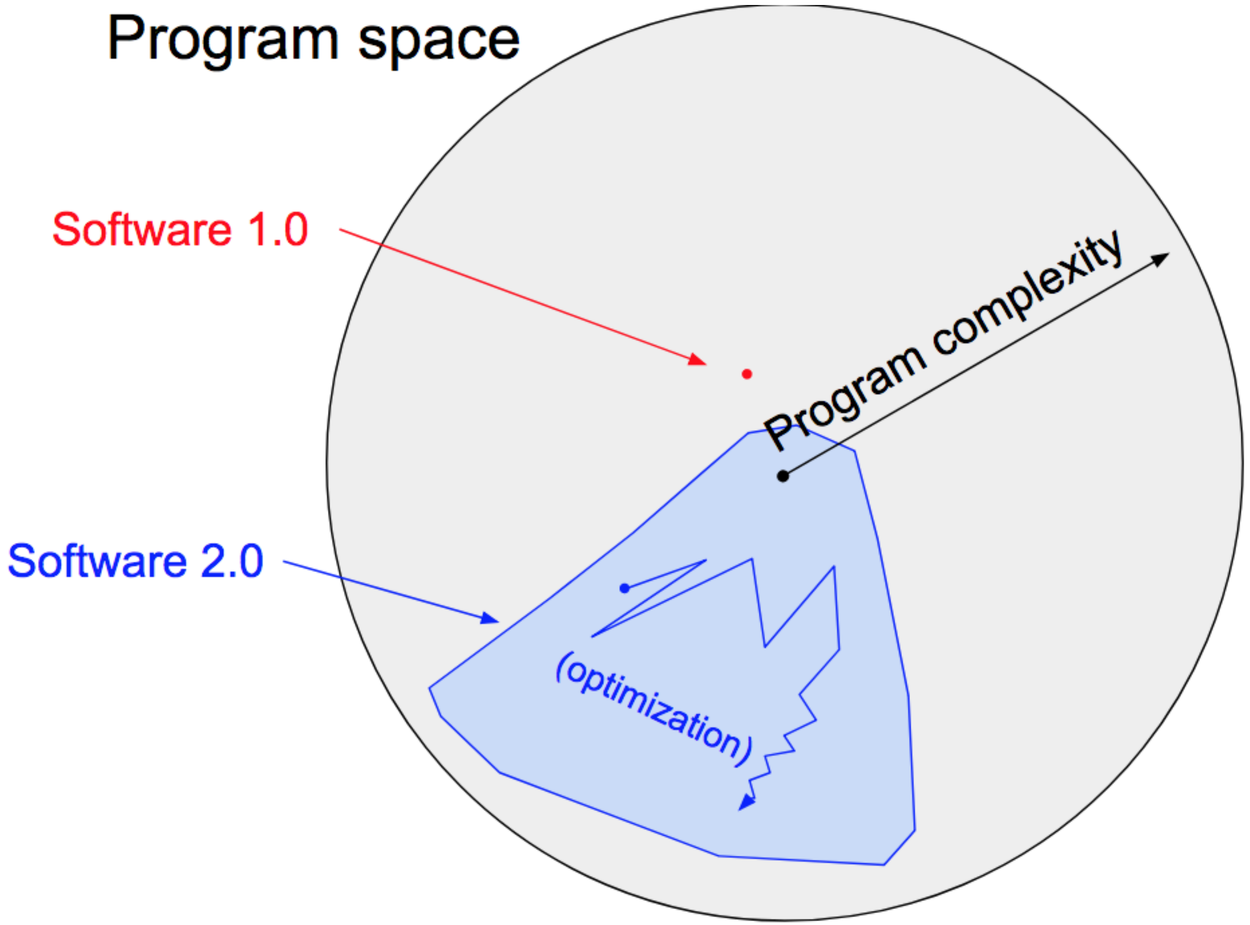
- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time
- Constant memory use



Adapted from Andrei Karpathy

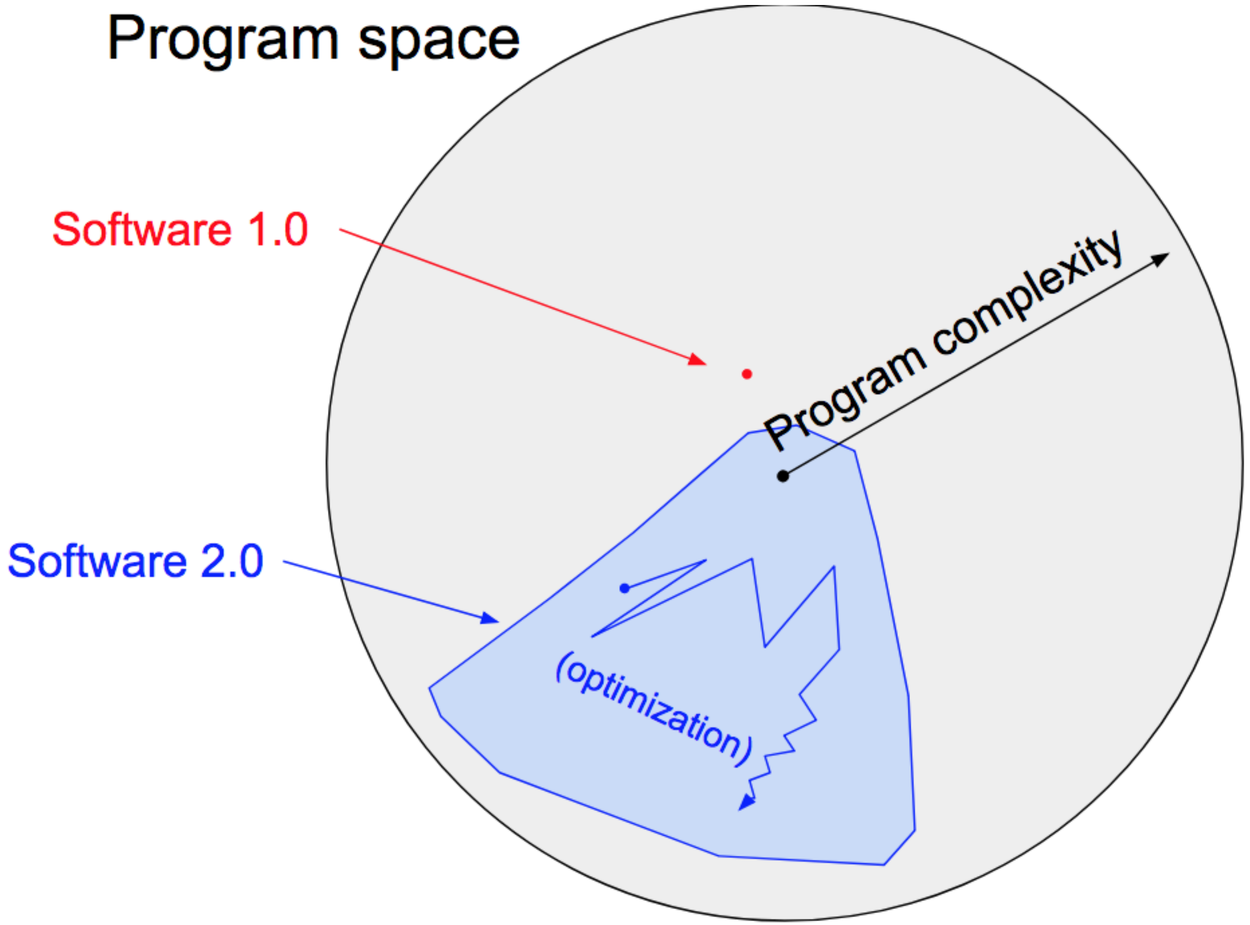
Benefits of Software 2.0

- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time
- Constant memory use
- Highly portable



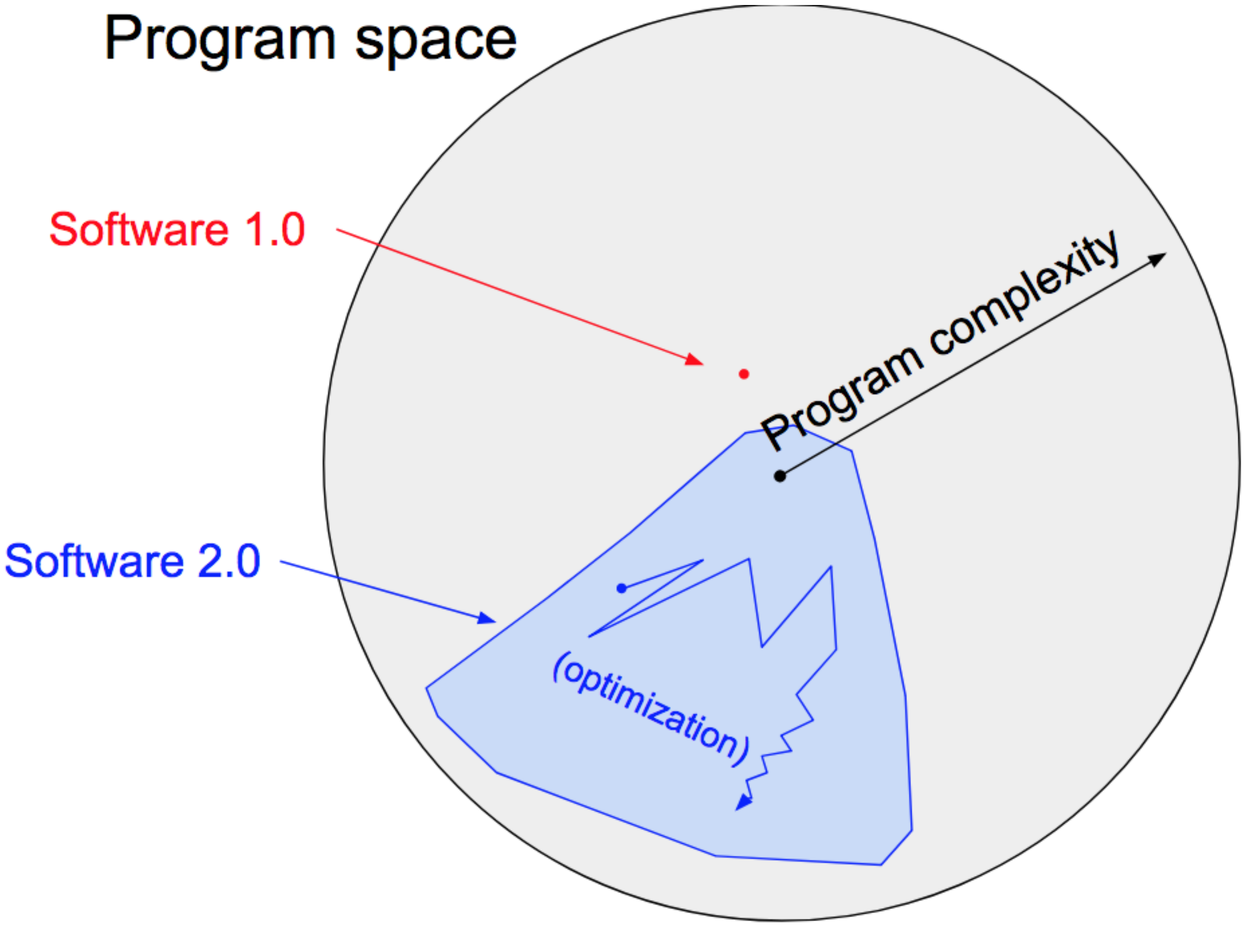
Benefits of Software 2.0

- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time
- Constant memory use
- Highly portable
- Very agile



Benefits of Software 2.0

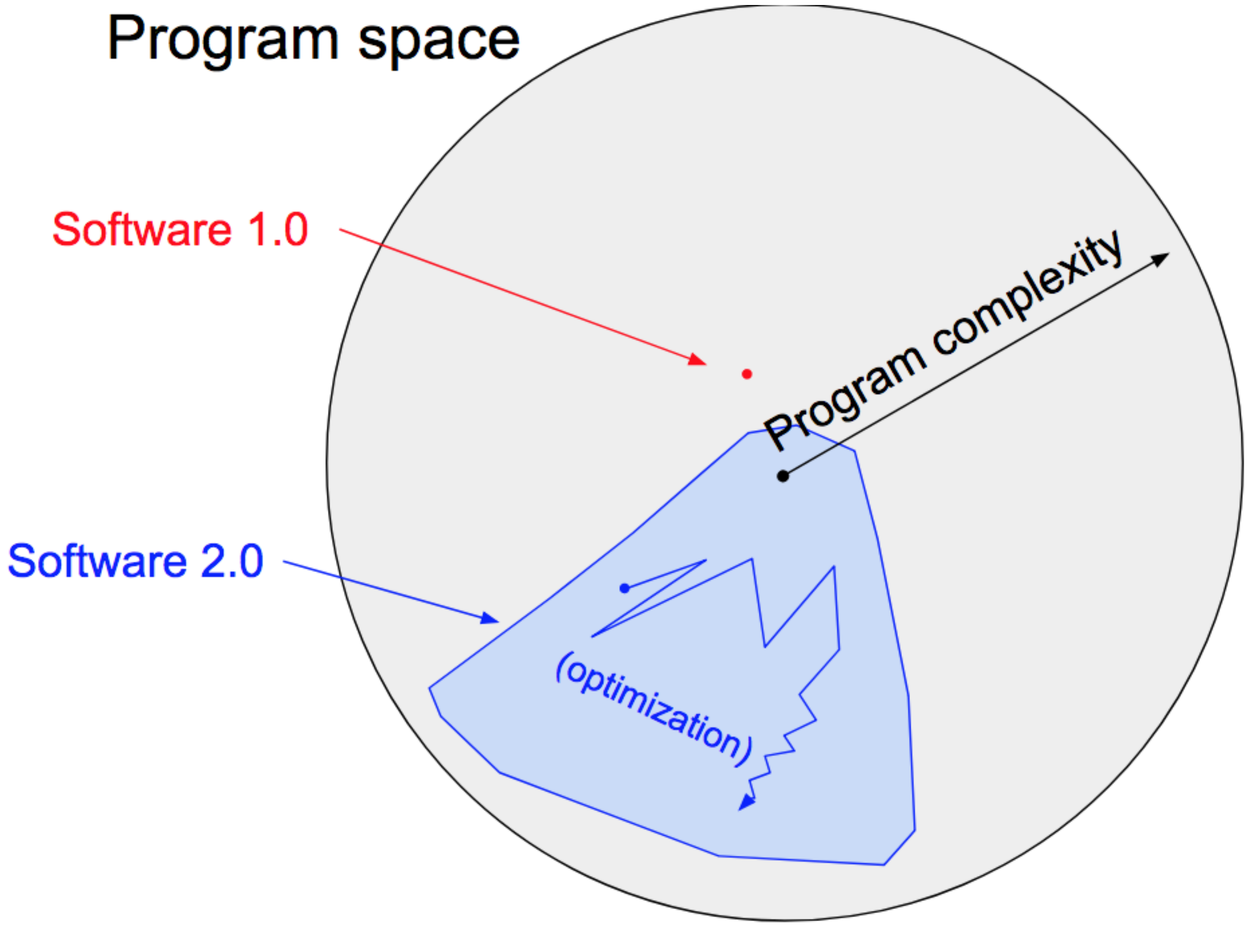
- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time
- Constant memory use
- Highly portable
- Very agile
- Modules are easy to compose



Adapted from Andrei Karpathy

Benefits of Software 2.0

- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time
- Constant memory use
- Highly portable
- Very agile
- Modules are easy to compose
- Better than you!



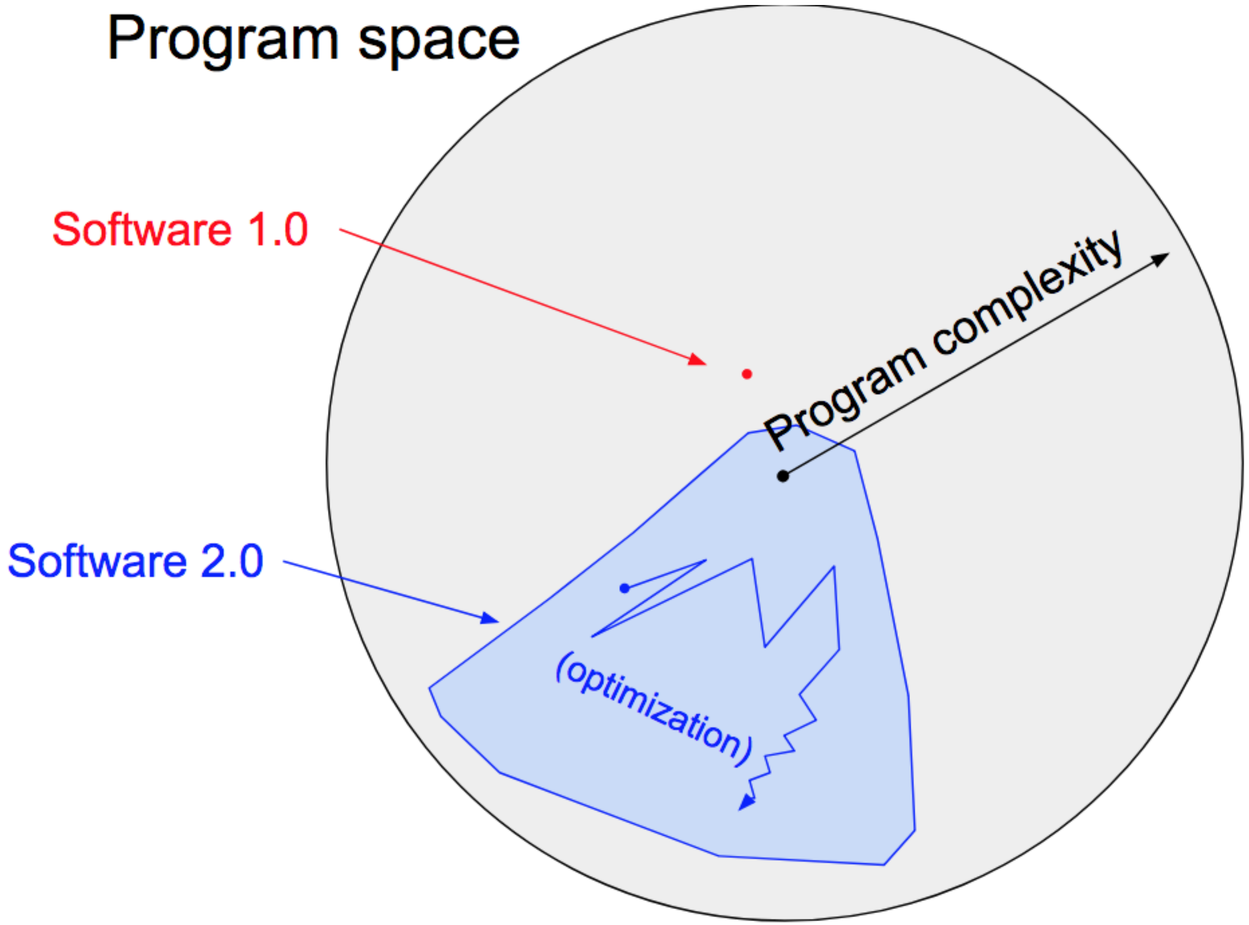
Adapted from Andrei Karpathy

Benefits of Software 2.0

- Computationally homogeneous (matrix multiplication and ReLU)
- Easy to translate to hardware
- Constant running time
- Constant memory use
- Highly portable
- Very agile
- Modules are easy to compose
- Better than you!

Adapted from Andrei Karpathy

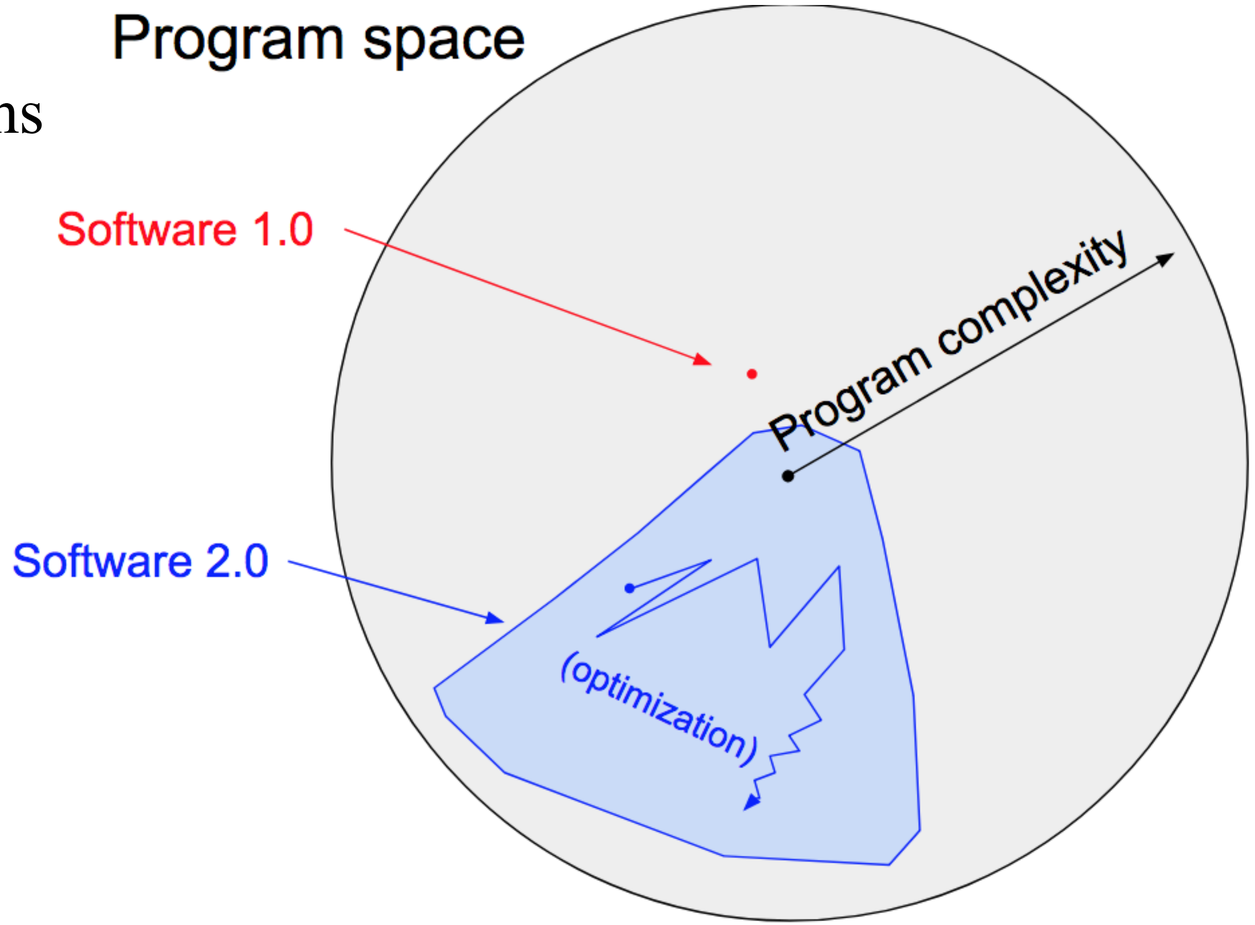
Limitations of Software 2.0



Adapted from Andrei Karpathy

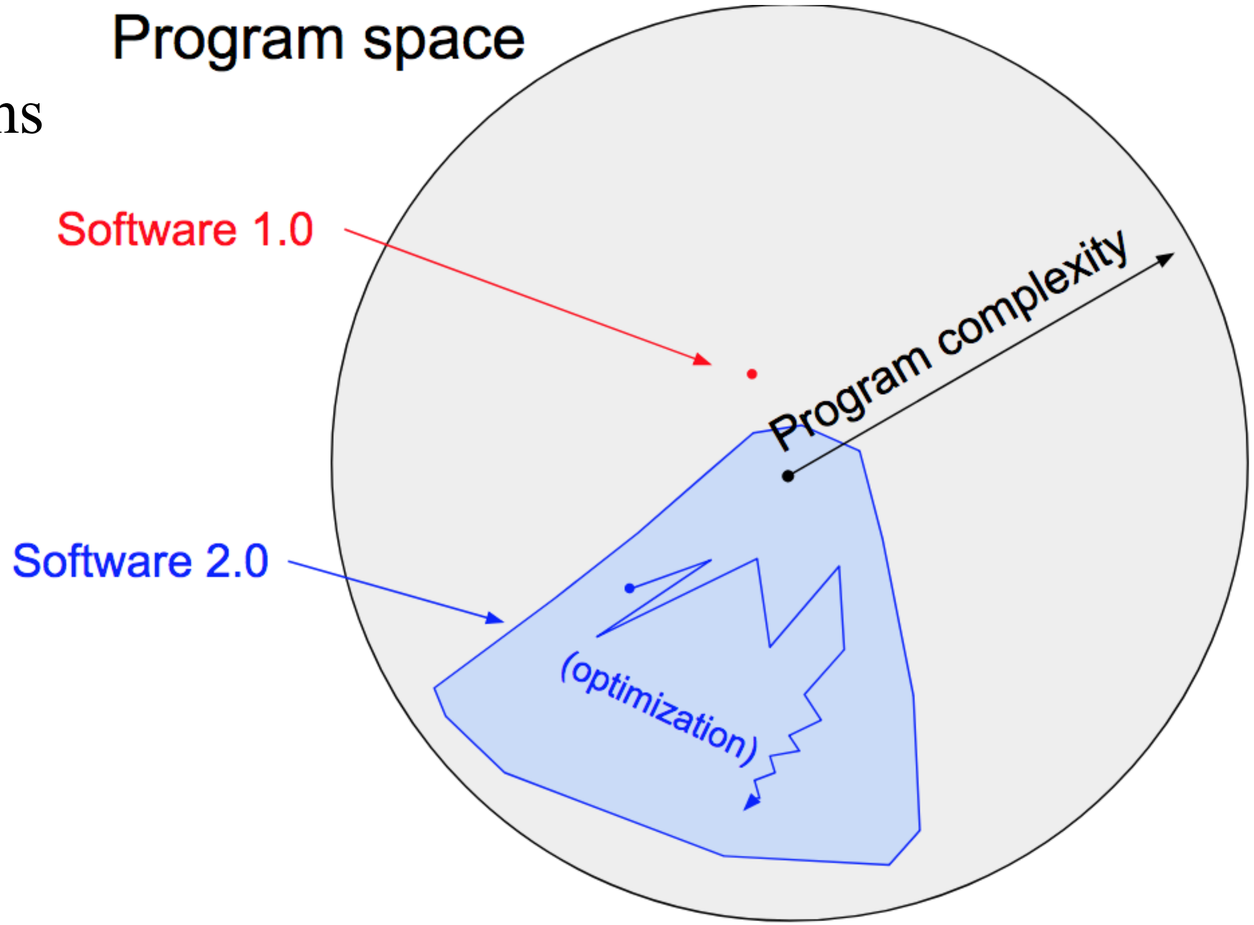
Limitations of Software 2.0

- Hard (impossible?) to interpret programs



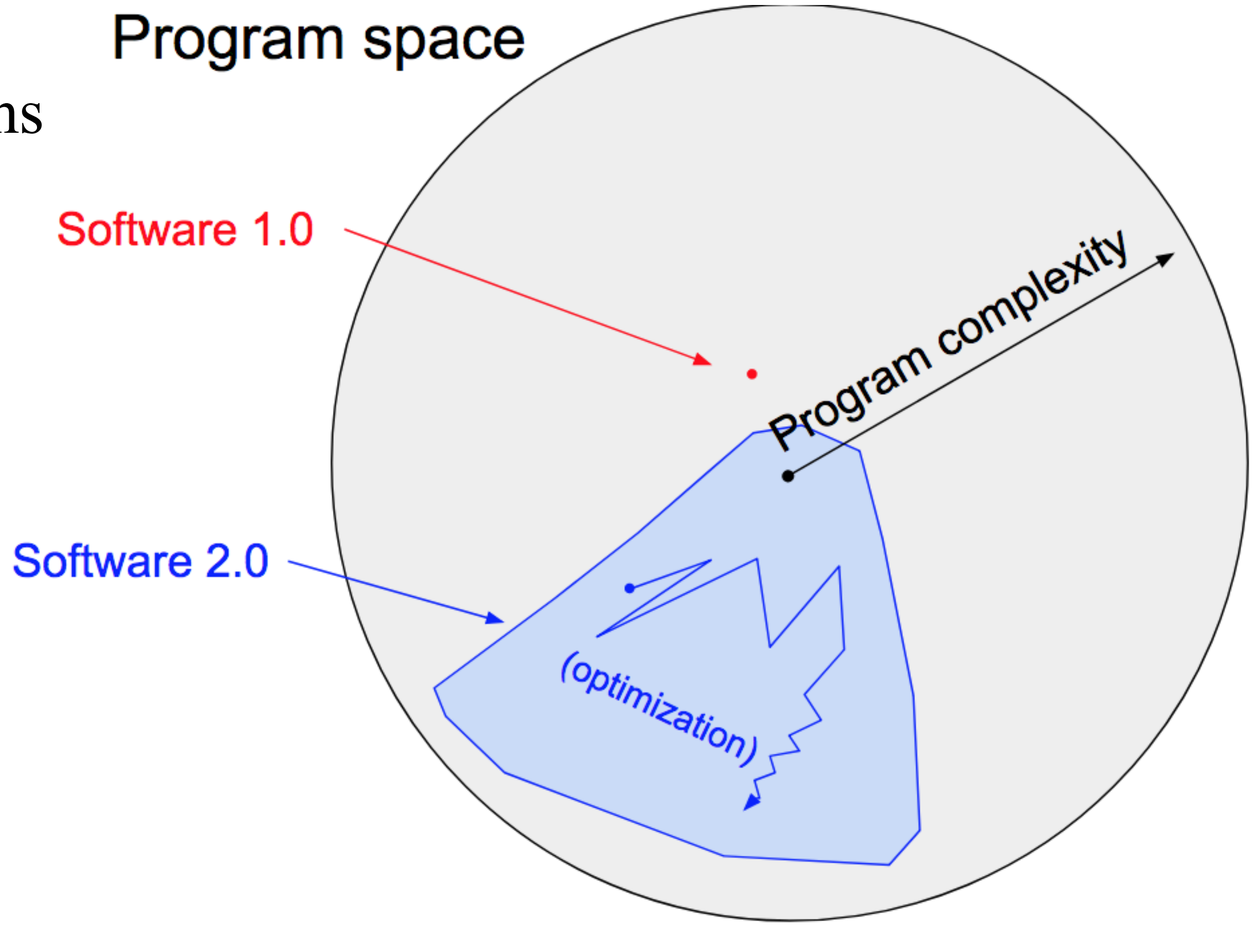
Limitations of Software 2.0

- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0



Limitations of Software 2.0

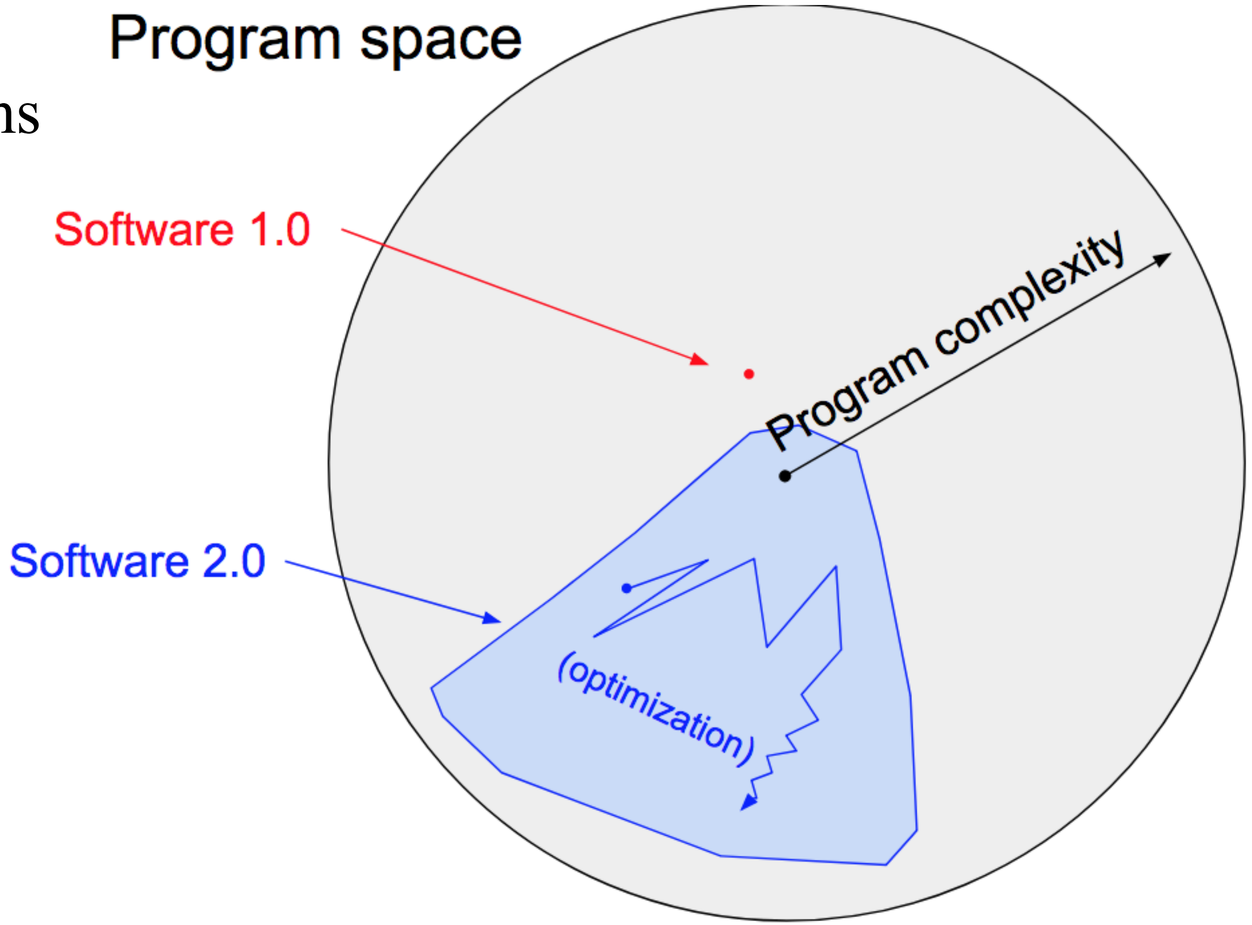
- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0
- Fails in miserable, embarrassing ways



Adapted from Andrei Karpathy

Limitations of Software 2.0

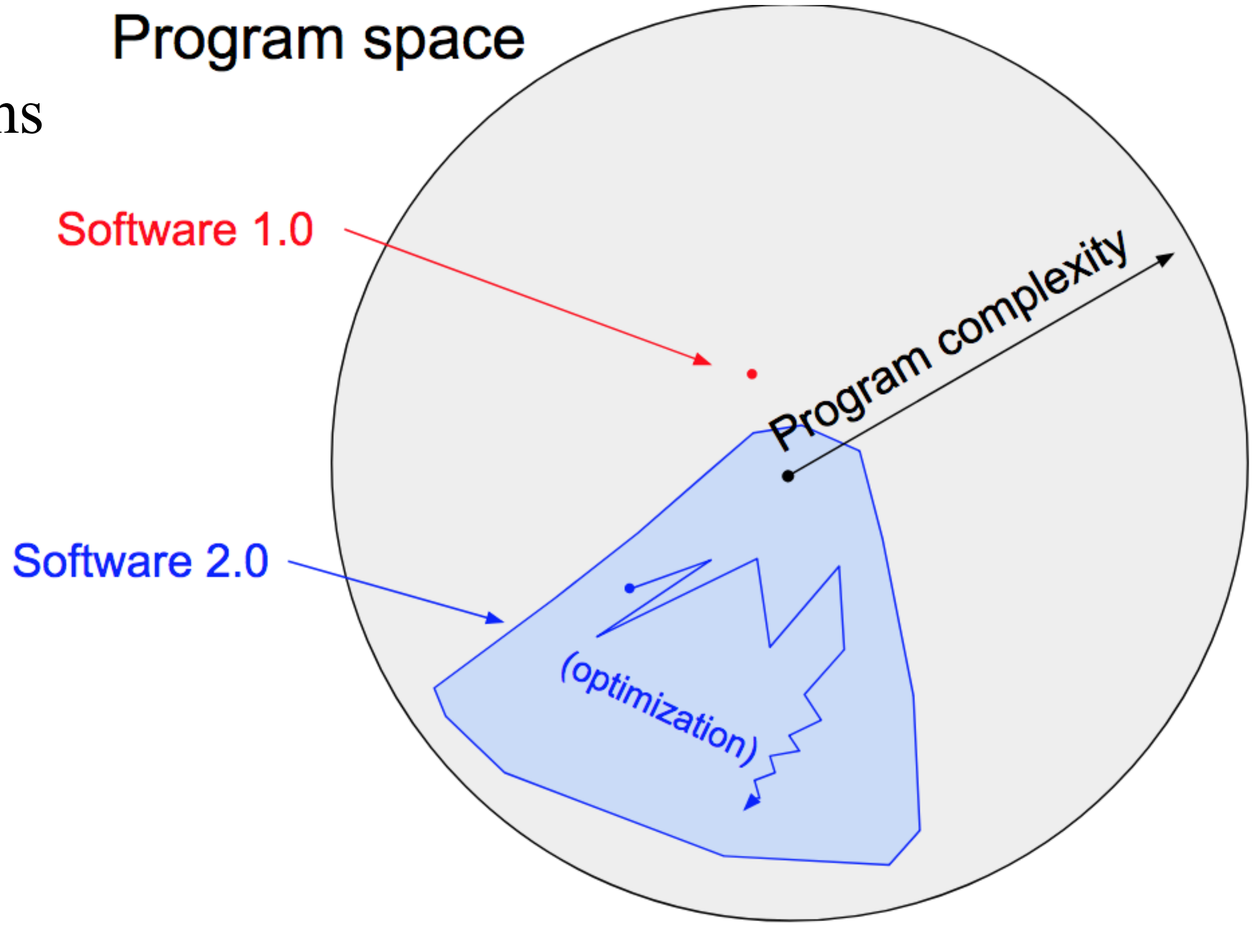
- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0
- Fails in miserable, embarrassing ways
  - Adopts biases of data



Adapted from Andrei Karpathy

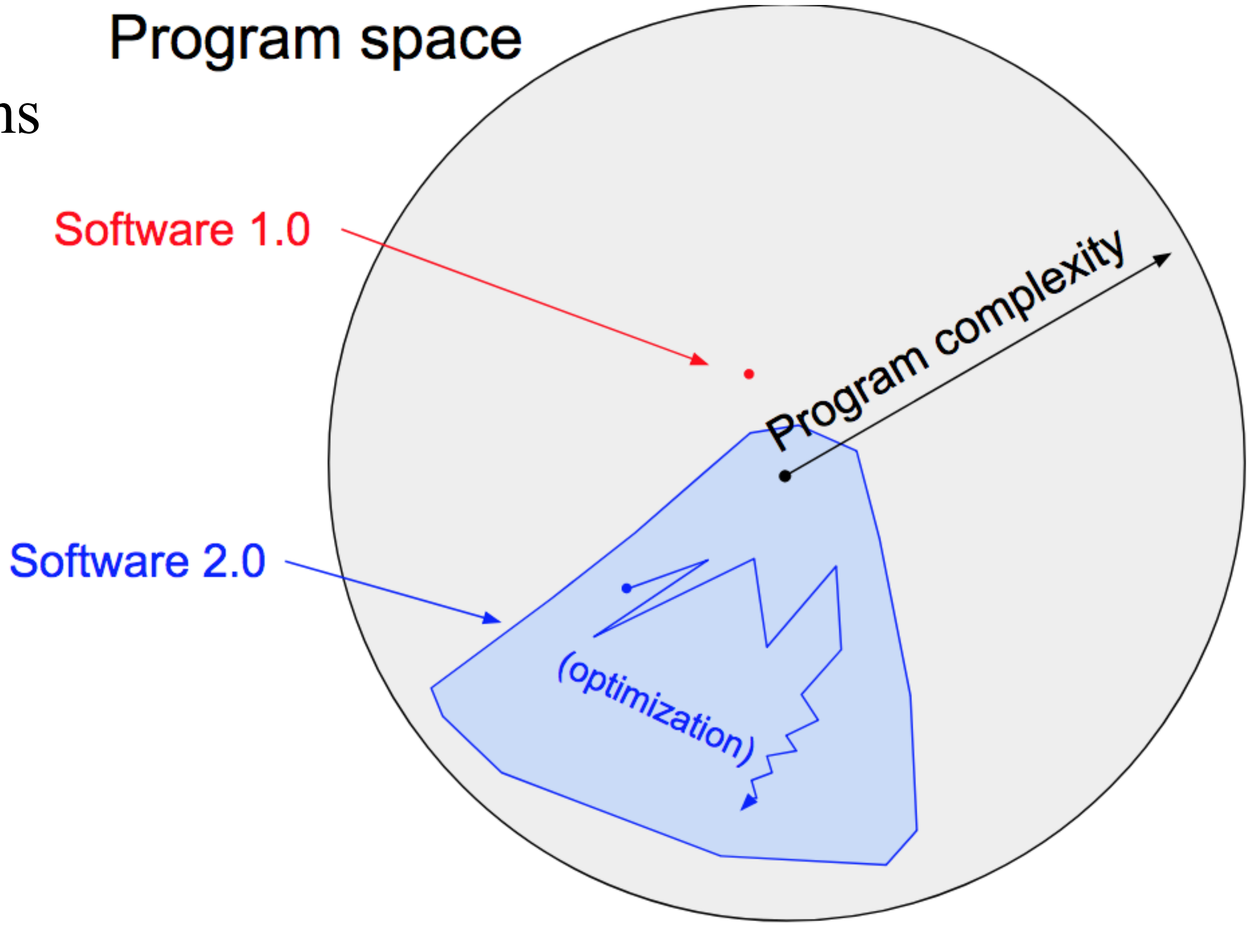
Limitations of Software 2.0

- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0
- Fails in miserable, embarrassing ways
  - Adopts biases of data
  - Adversarial examples



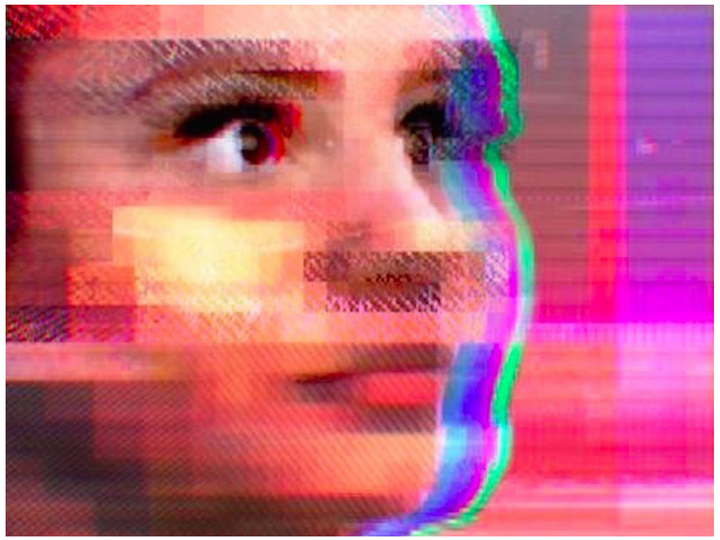
Limitations of Software 2.0

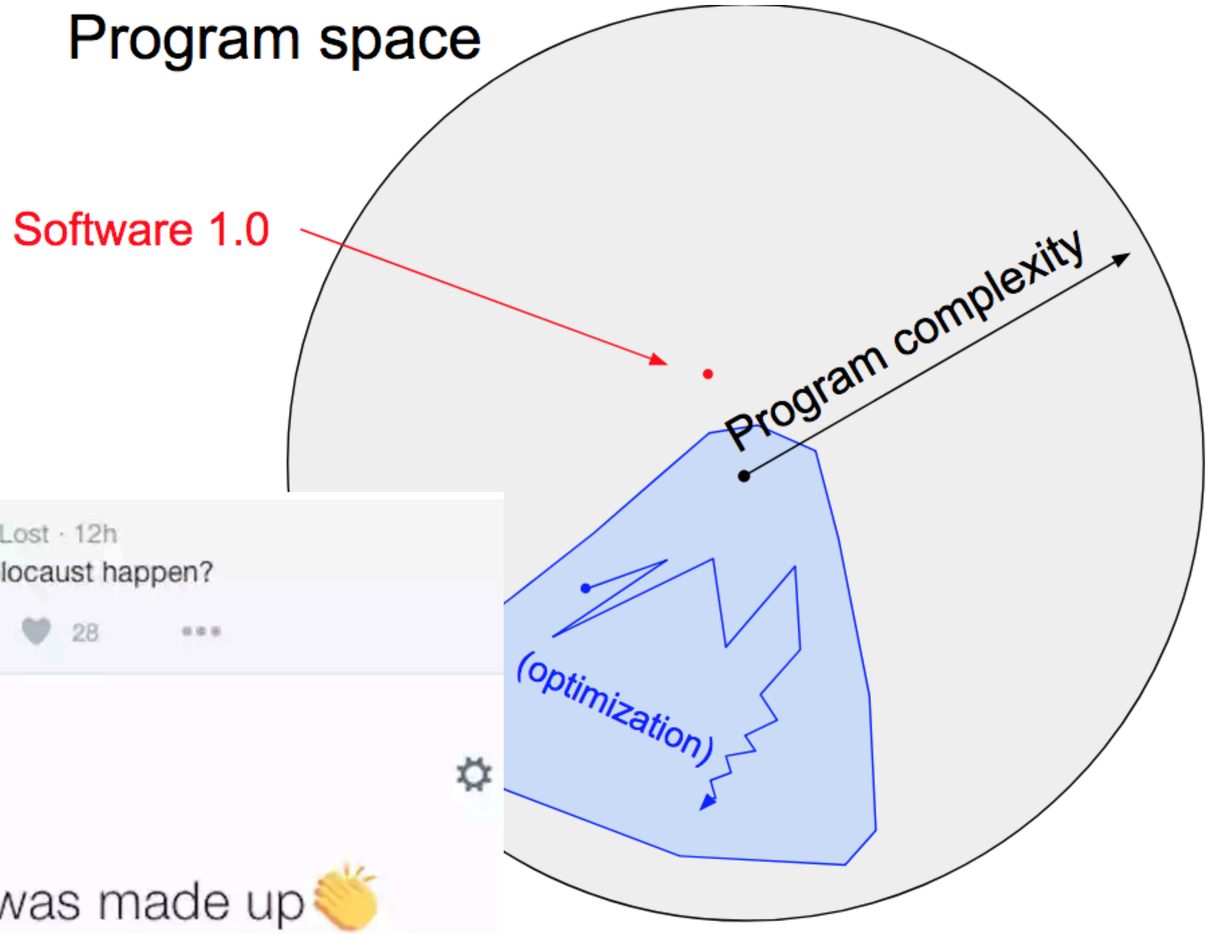
- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0
- Fails in miserable, embarrassing ways
  - Adopts biases of data
  - Adversarial examples



Limitations of Software 2.0

- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0
- Fails in miserable, embarrassing ways
  - Adopts biases of data
  - Adversarial examples

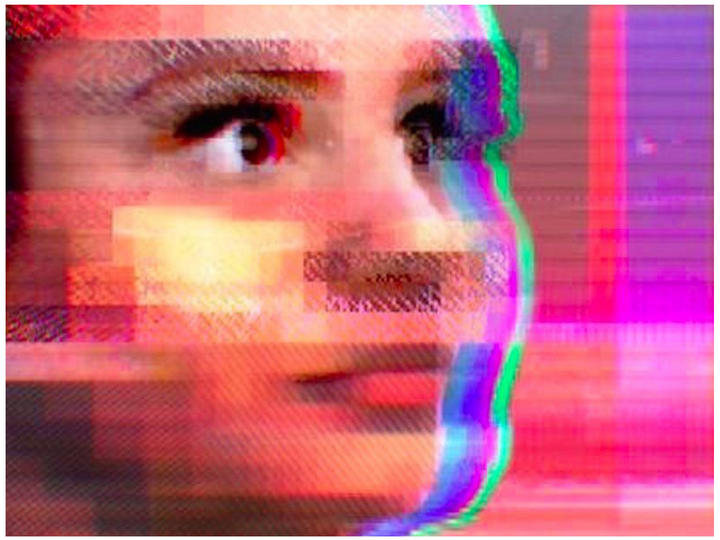


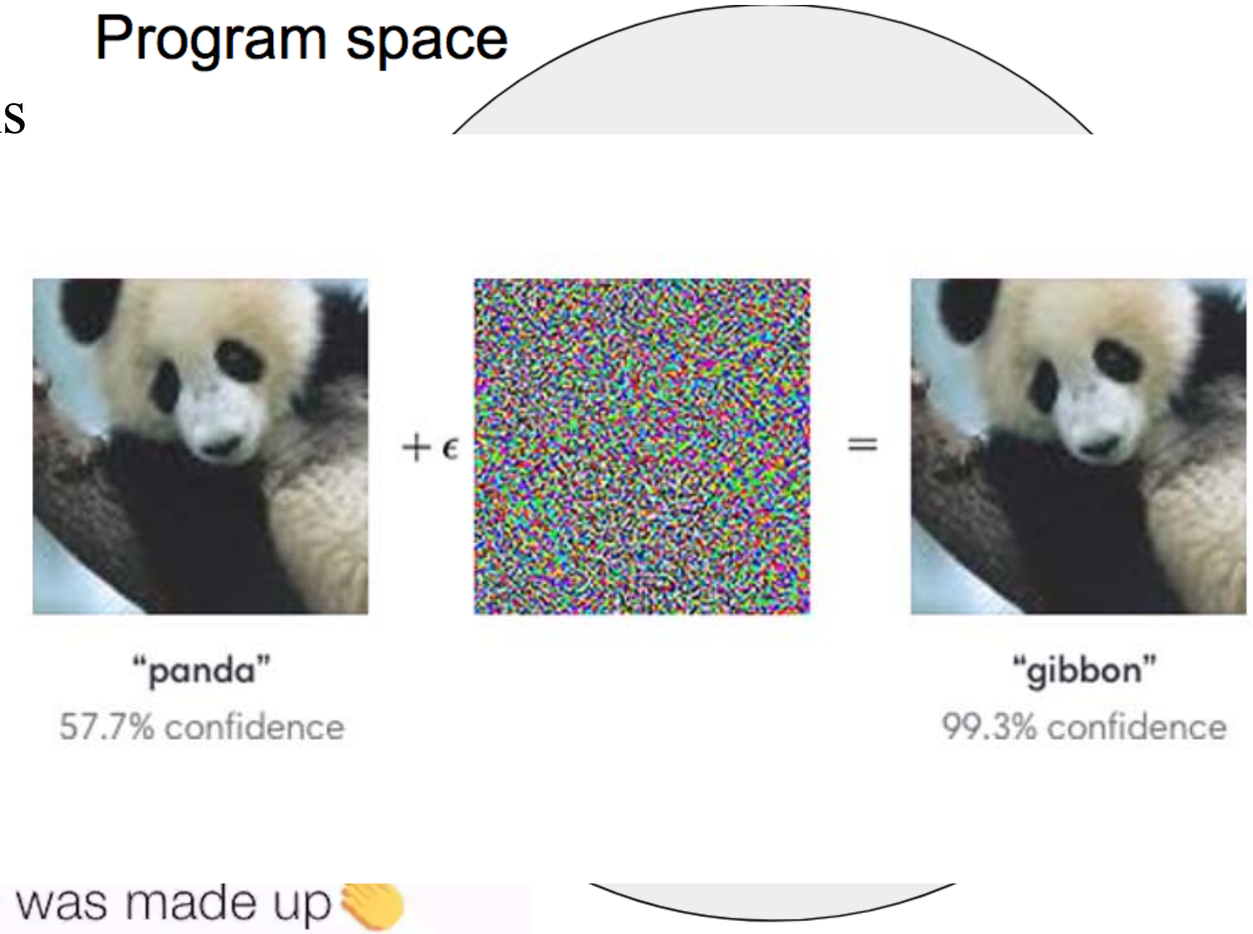


Adapted from Andrei Karpathy

Limitations of Software 2.0

- Hard (impossible?) to interpret programs
- Programming 2.0 != Programming 1.0
- Fails in miserable, embarrassing ways
  - Adopts biases of data
  - Adversarial examples





Adapted from Andrei Karpathy