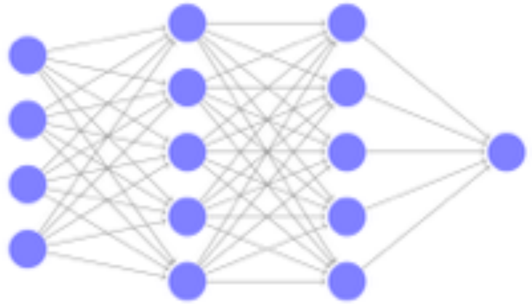
ENLP Lecture 22

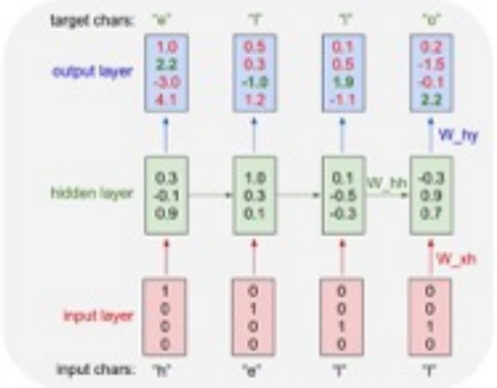
Deep Learning & Neural Networks

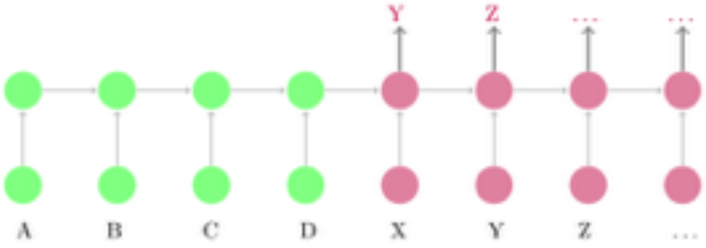
Austin Blodgett, Georgetown University

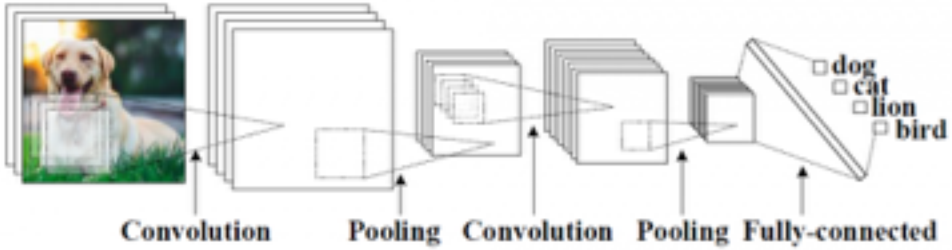
April 23, 2018

a family of algorithms









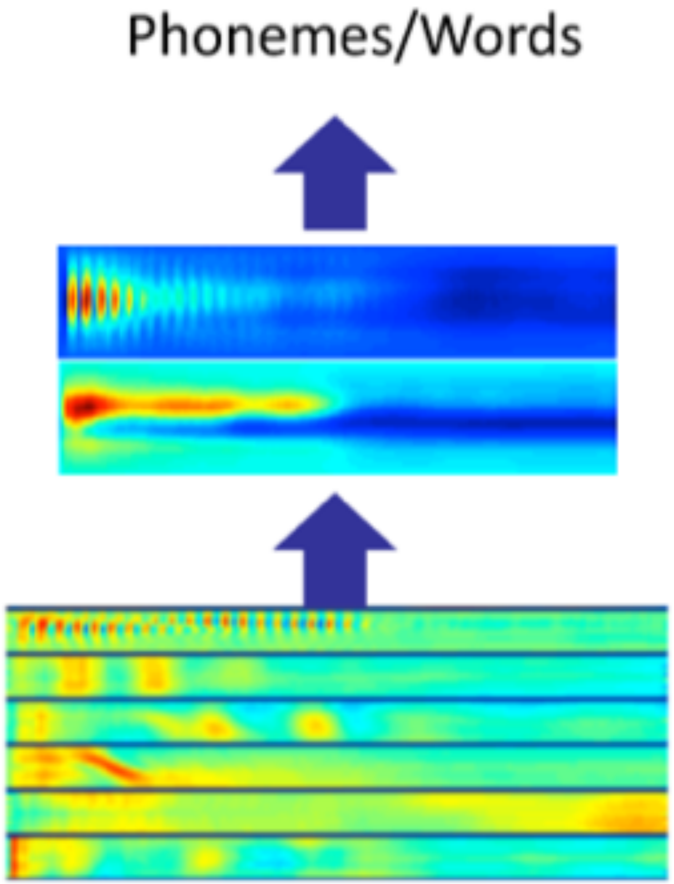
NN Task	Example Input	Example Output
Binary classification		
Multiclass classification		
Sequence		
Sequence to Sequence		
Tree/Graph Parsing		

NN Task	Example Input	Example Output
Binary classification	features	+/-
Multiclass classification	features	decl, imper, …
Sequence	sentence	POS tags
Sequence to Sequence	(English) sentence	(Spanish) sentence
Tree/Graph Parsing	sentence	dependency tree or AMR parsing

Deep Learning for Speech

- The first breakthrough results of "deep learning" on large datasets happened in speech recognition
- Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition Dahl et al. (2010)

Acoustic model	Recog WER	RT03S FSH	Hub5 SWB
Traditional features	1-pass −adapt	**27.4**	**23.6**
Deep Learning	1-pass −adapt	**18.5** (−33%)	**16.1** (−32%)

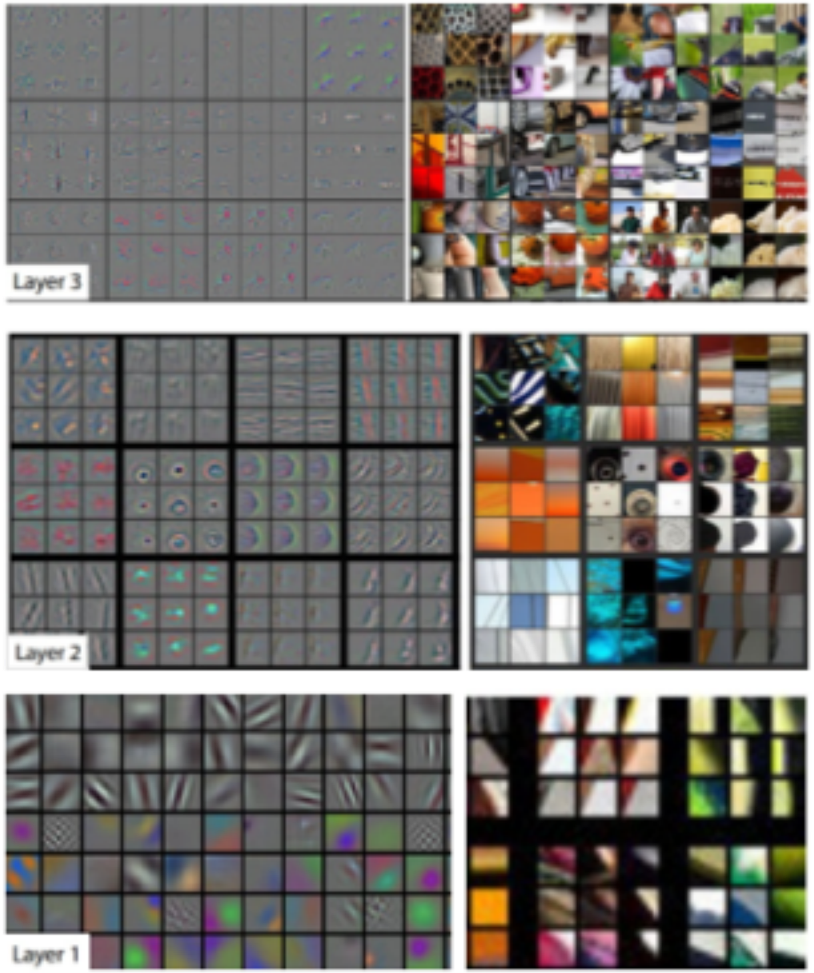


(Slide from Manning and Socher)

Deep Learning for Computer Vision

Most deep learning groups have focused on computer vision (at least till 2 years ago)

The breakthrough DL paper: ImageNet Classification with Deep Convolutional Neural Networks by Krizhevsky, Sutskever, & Hinton, 2012, U. Toronto. 37% error red.



Zeiler and Fergus (2013)

(Slide from Manning and Socher)

Reasons for Exploring Deep Learning

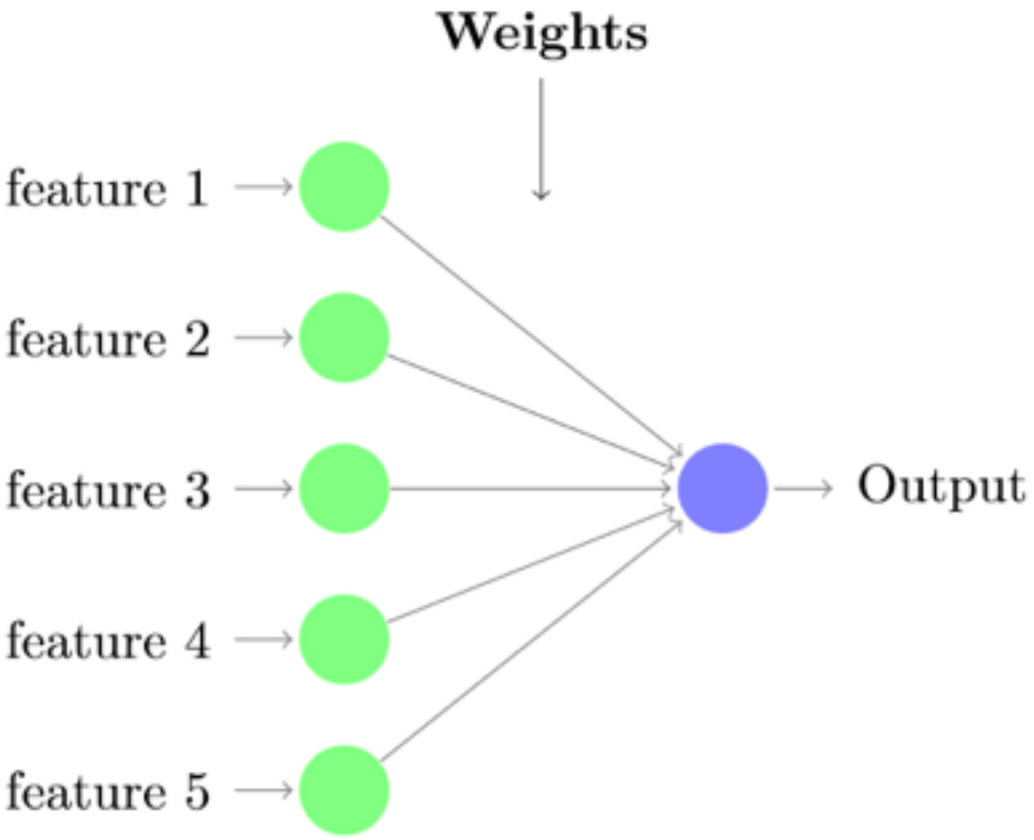
- In ~2010 **deep** learning techniques started outperforming other machine learning techniques. Why this decade?

- Large amounts of training data favor deep learning
- Faster machines and multicore CPU/GPUs favor Deep Learning
- New models, algorithms, ideas
  - Better, more flexible learning of intermediate representations
  - Effective end-to-end joint system learning
  - Effective learning methods for using contexts and transferring between tasks

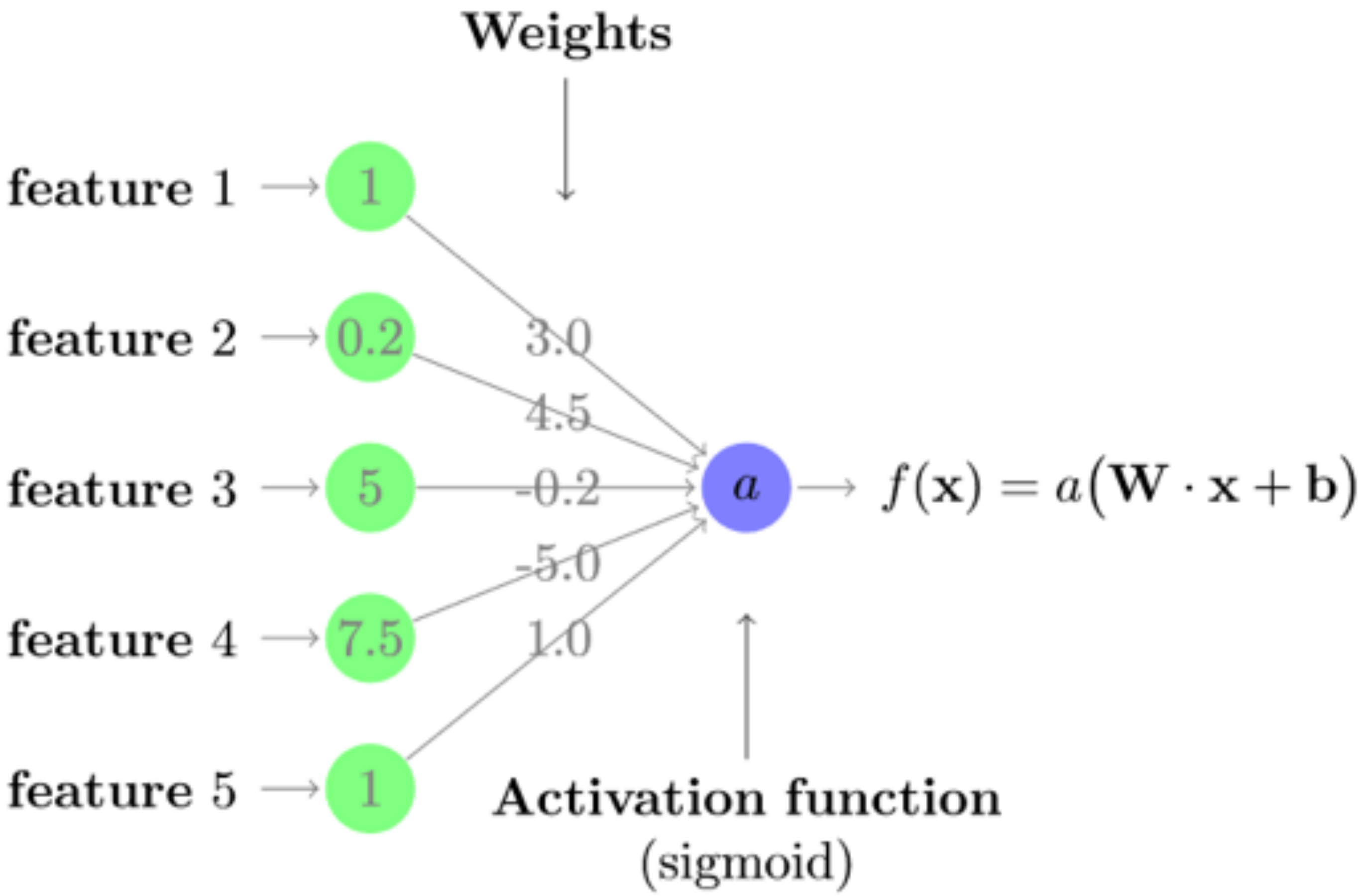
→ **Improved performance** (first in speech and vision, then NLP)

(Slide from Manning and Socher)

Perceptron


Weights
feature 1
feature 2
feature 3
feature 4
feature 5
Output

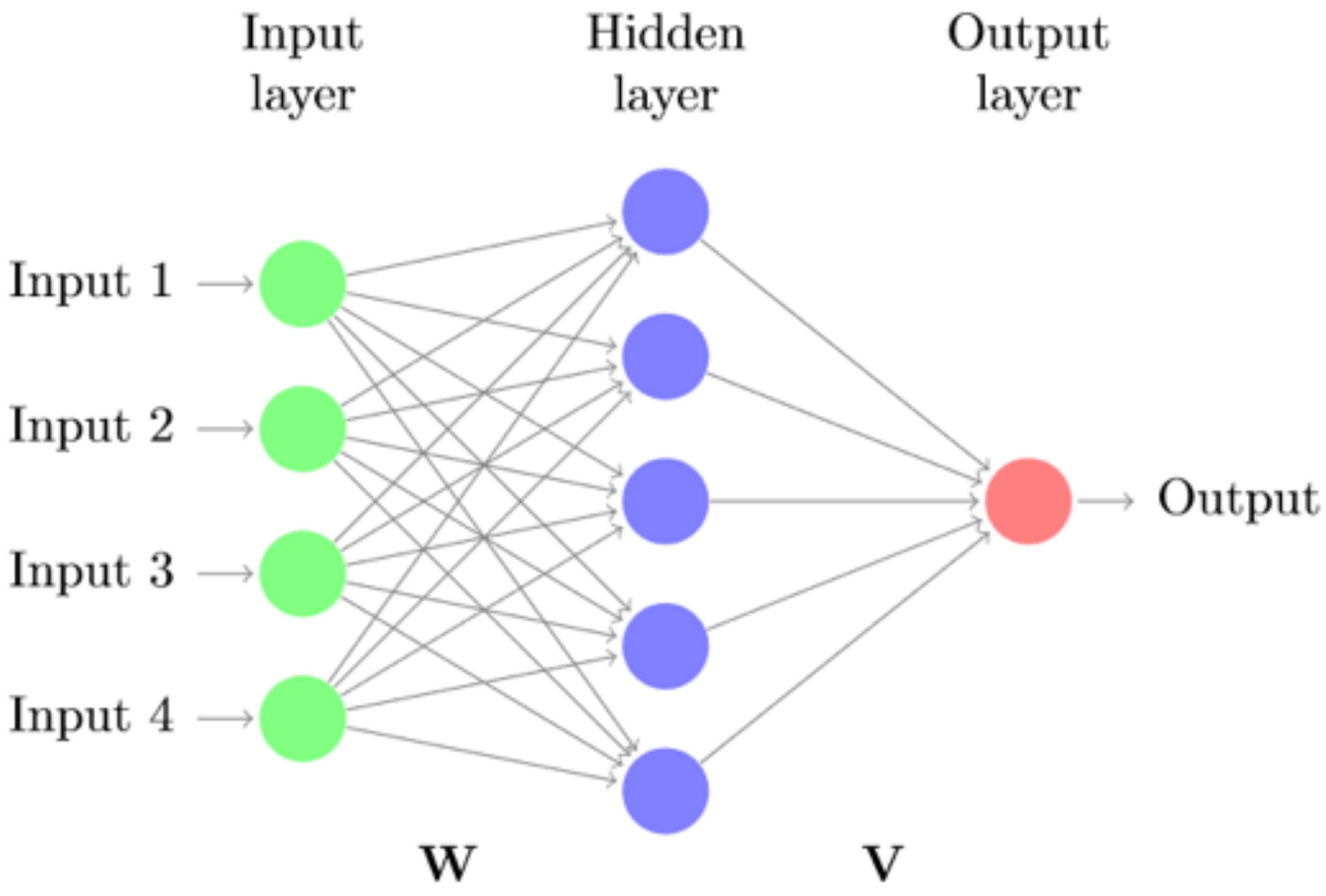
Perceptron

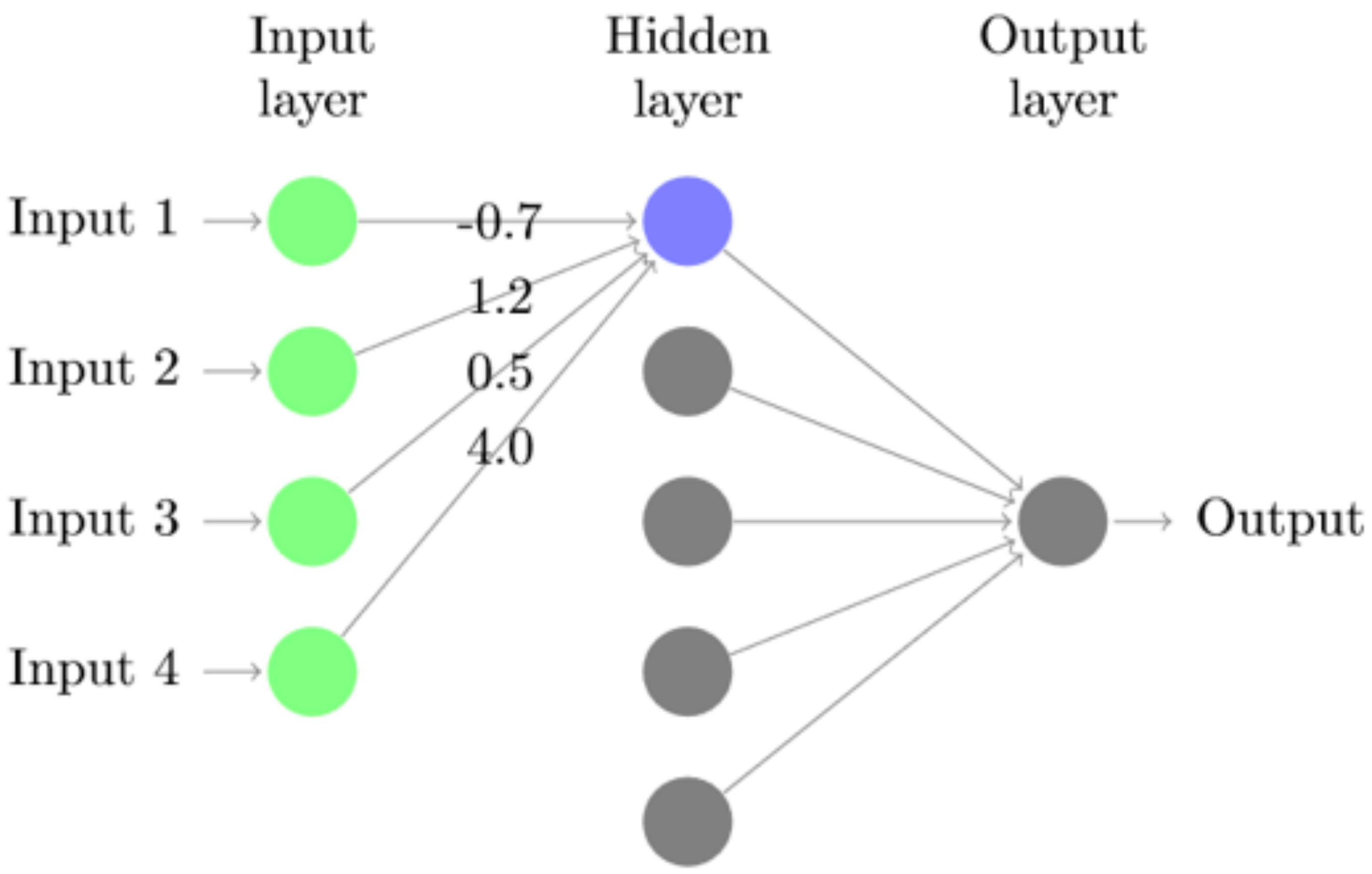


Weights
feature 1
feature 2
feature 3
feature 4
feature 5
1
0.2
5
7.5
1
3.0
4.5
-0.2
-5.0
1.0
a
$f(\mathbf{x}) = a(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$
Activation function
(sigmoid)

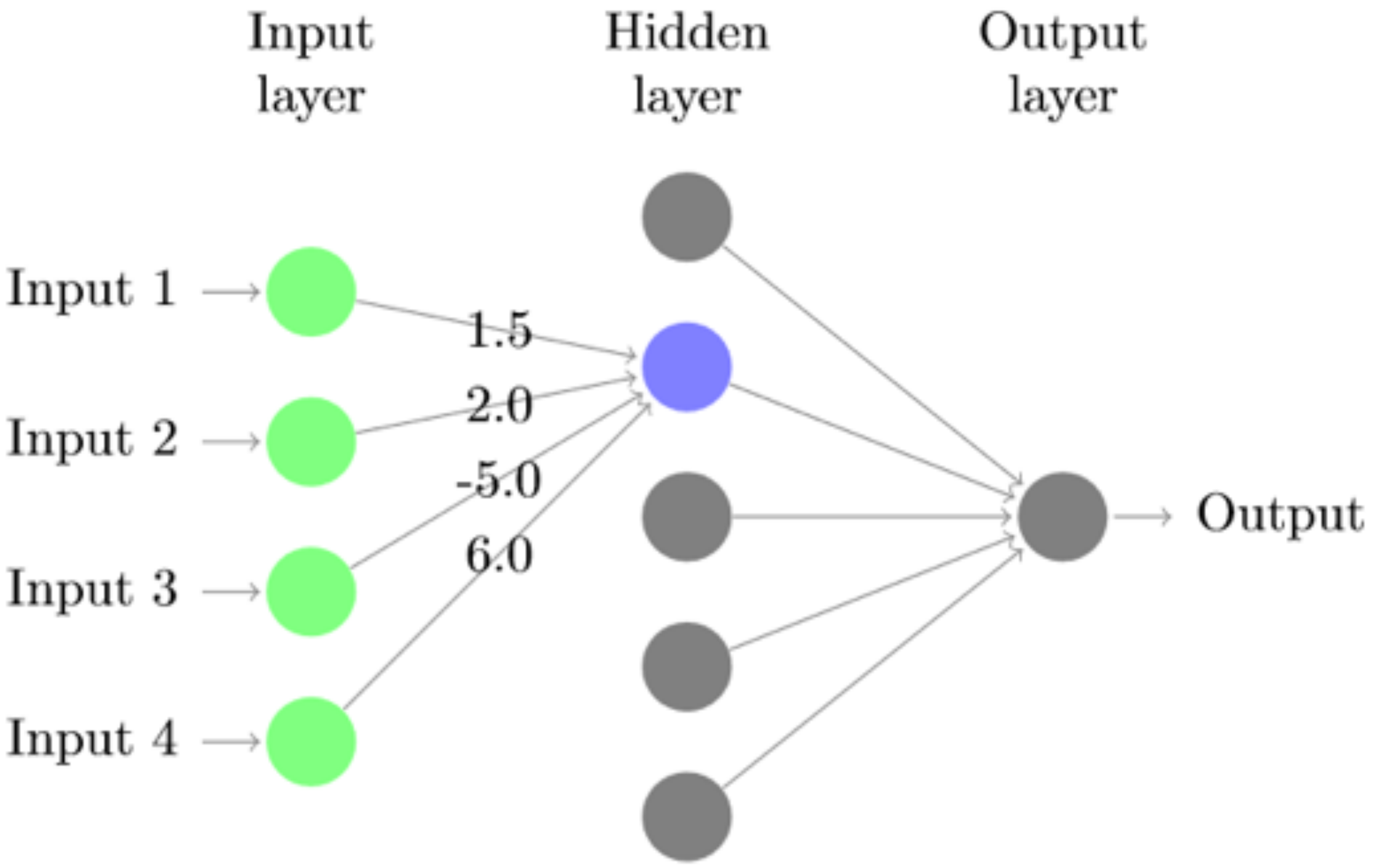
FFNNs

- **Feed Forward Neural Net** – Multiple layers of neurons
- *Can solve non-linearly separable problems*
- (All arrows face the same direction)
- Applications:
  - *Text classification* – sentiment analysis, language detection, …
  - *Unsupervised learning* – dimension reduction, word2vec

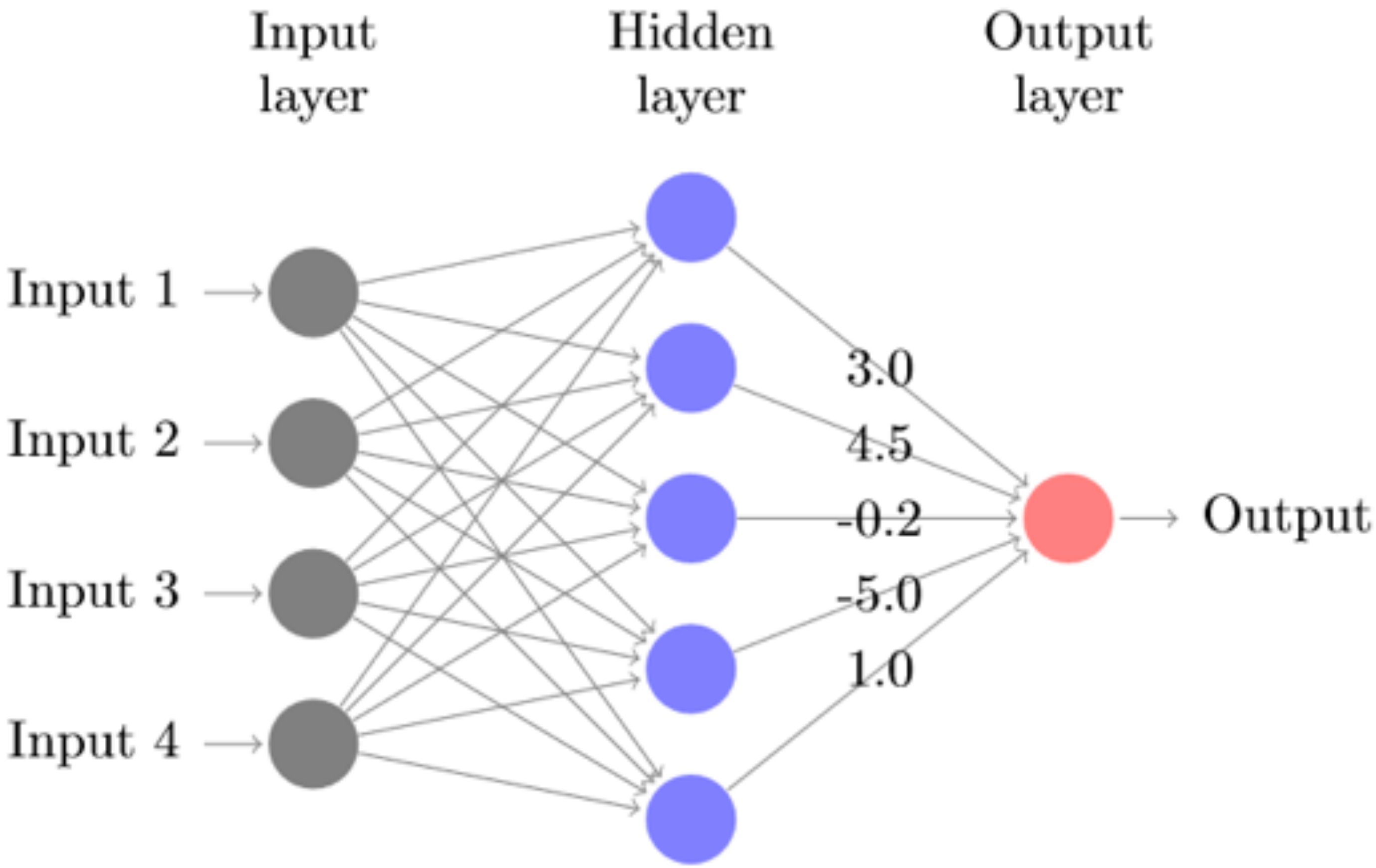

Input
layer
Hidden
layer
Output
layer
Input 1
Input 2
Input 3
Input 4
Output
W
V

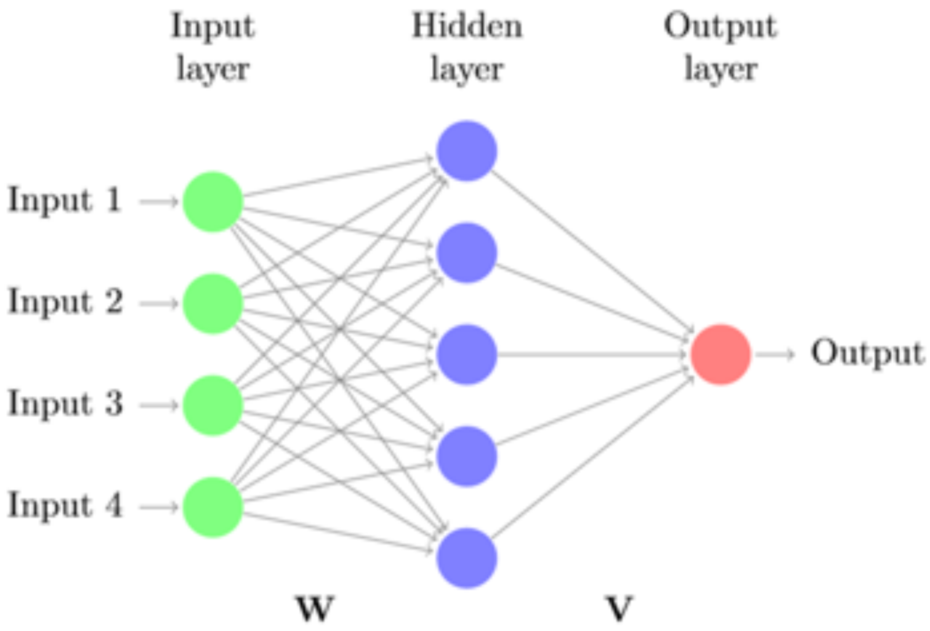


Input layer
Hidden layer
Output layer
Input 1
Input 2
Input 3
Input 4
-0.7
1.2
0.5
4.0
Output



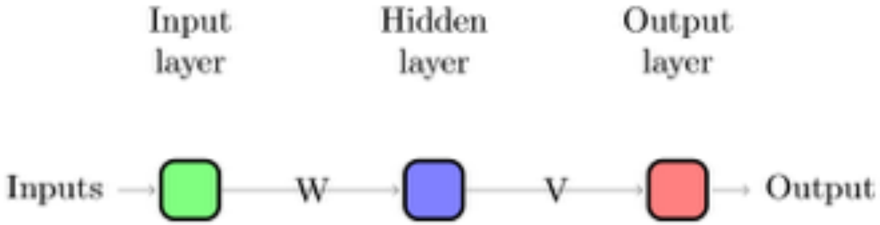
Input layer
Hidden layer
Output layer
Input 1
Input 2
Input 3
Input 4
1.5
2.0
-5.0
6.0
Output


Input layer
Hidden layer
Output layer
Input 1
Input 2
Input 3
Input 4
3.0
4.5
-0.2
-5.0
1.0
Output



=

Compact diagram



FAQ

- How do I interpret an NN?
  - An NN performs *function approximation*
  - Connections in an NN posit *relatedness*
  - Lack of connection posits *independence*

FAQ

- What do the weights mean?
  - *Functional perspective* – these weights optimize NN's task performance
  - *Representation perspective* – weights represent **unlabeled, distributed** knowledge (*useful* but not generally *interpretable*)

FAQ

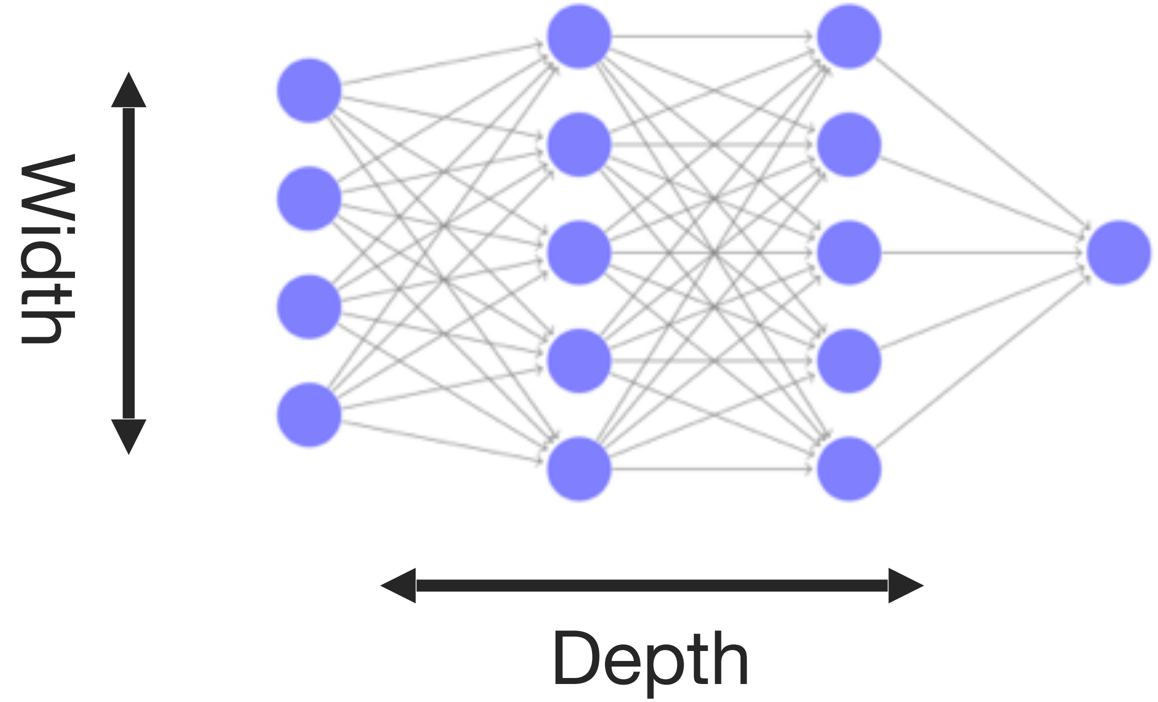
- Can an NN learn anything?
  - No, but …

Theorem: 'One hidden layer is enough to represent (*not learn*) an approximation of any function to an arbitrary degree of accuracy'

  - *(Given infinite training data, memory, etc.)*

FAQ

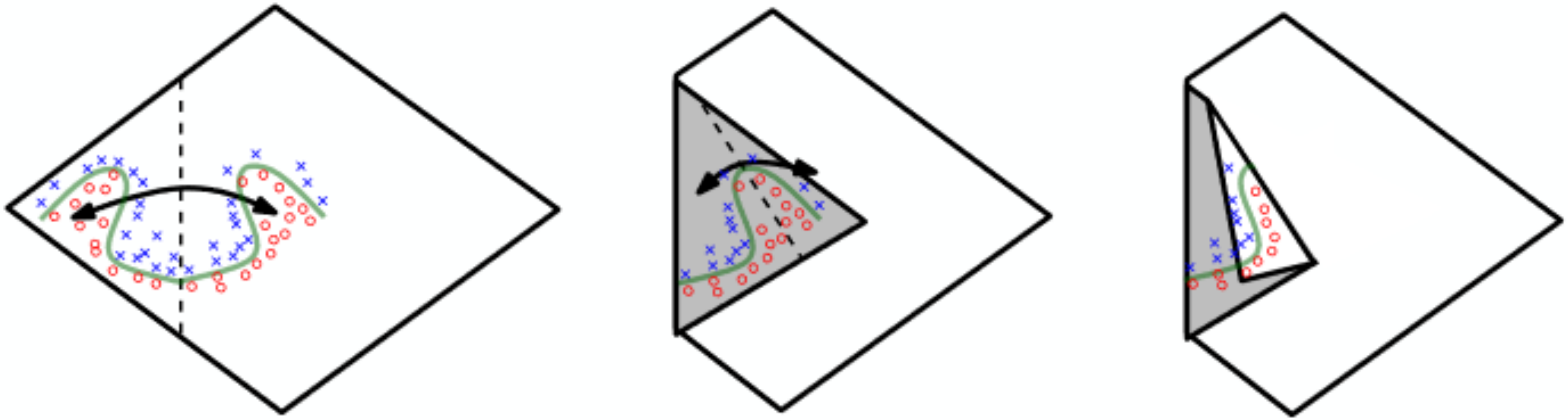
- What happens if I make an NN deeper?



Width controls overfitting/underfitting

Depth allows complex functions, can reduce overfitting

Exponential Representation Advantage of Depth

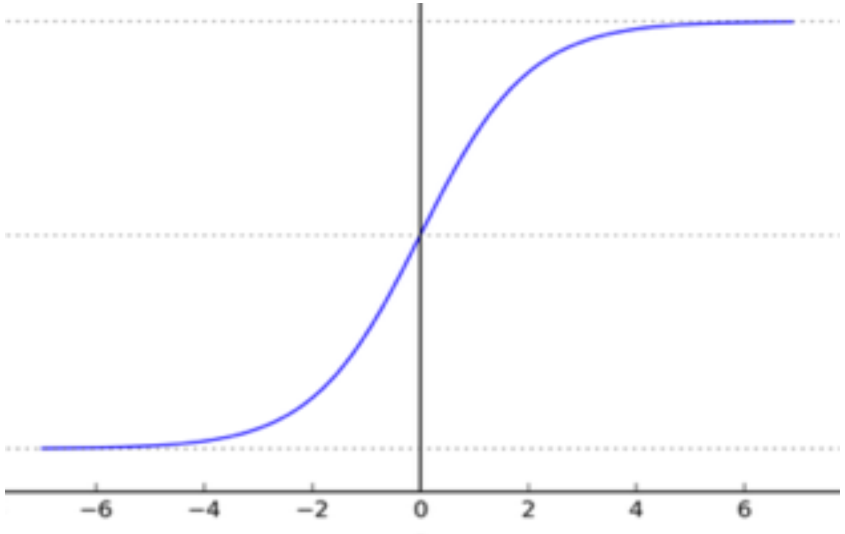


(Goodfellow 2017)

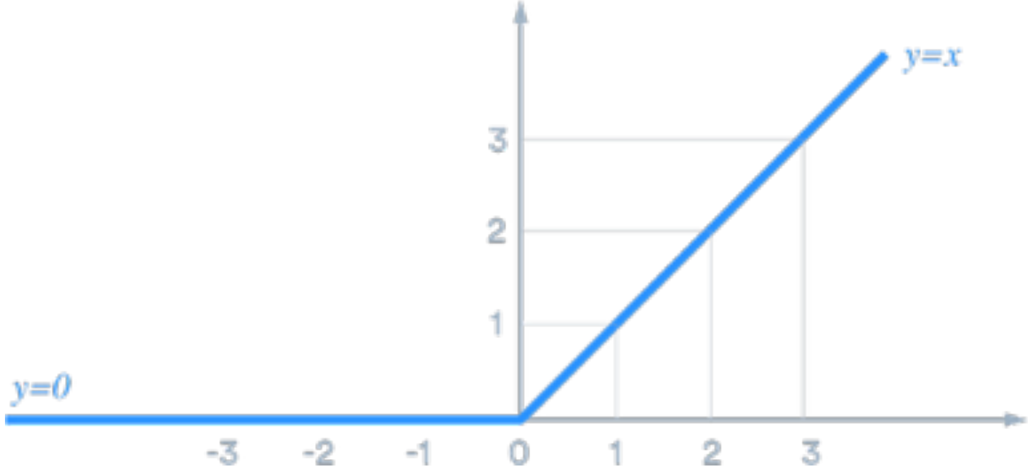
activation functions

- **Activation function –** “squishes” neuron inputs into an output
  - Use in output layer – *Sigmoid (binary class), Softmax (Multiclass)*
  - Use in hidden layers – *ReLU, Leaky ReLU*

Sigmoid



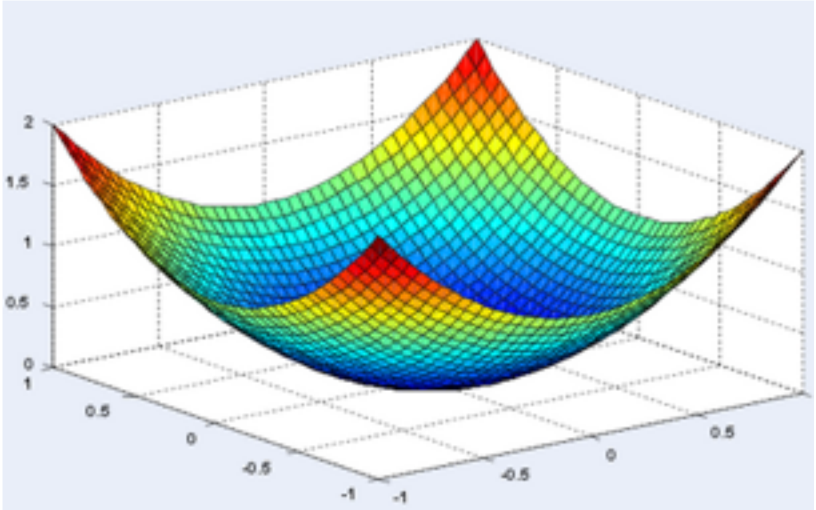
ReLU (**Re**ctified **L**inear **U**nit)



training

- To train an NN, you need:
  - **Training set -** ordered pairs each with an input and target output
  - **Loss function -** a function to be optimized, e.g. *Cross Entropy*
  - **Optimizer** - a method for adjusting the weights, e.g. *Gradient Descent*

Gradient Descent – use gradient to find lowest point in a function



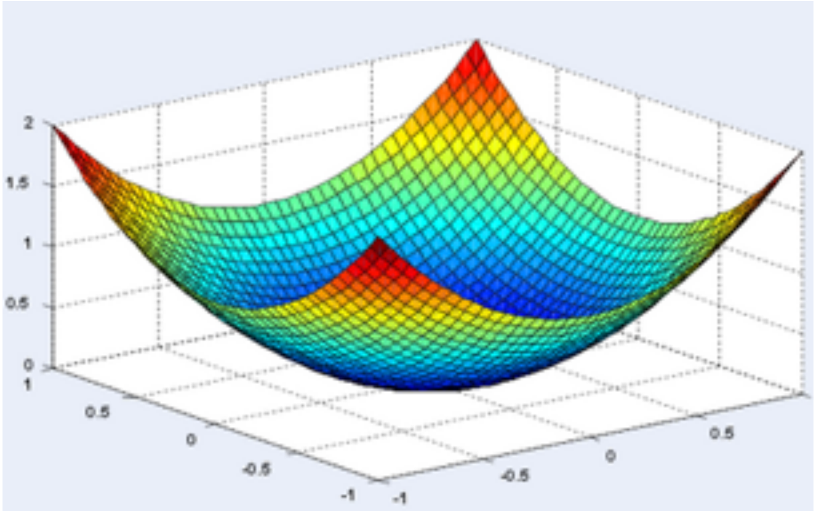
backpropagation

- **Backpropagation =** Chain Rule + Dynamic Programming

Loss function – measures NN’s performance.

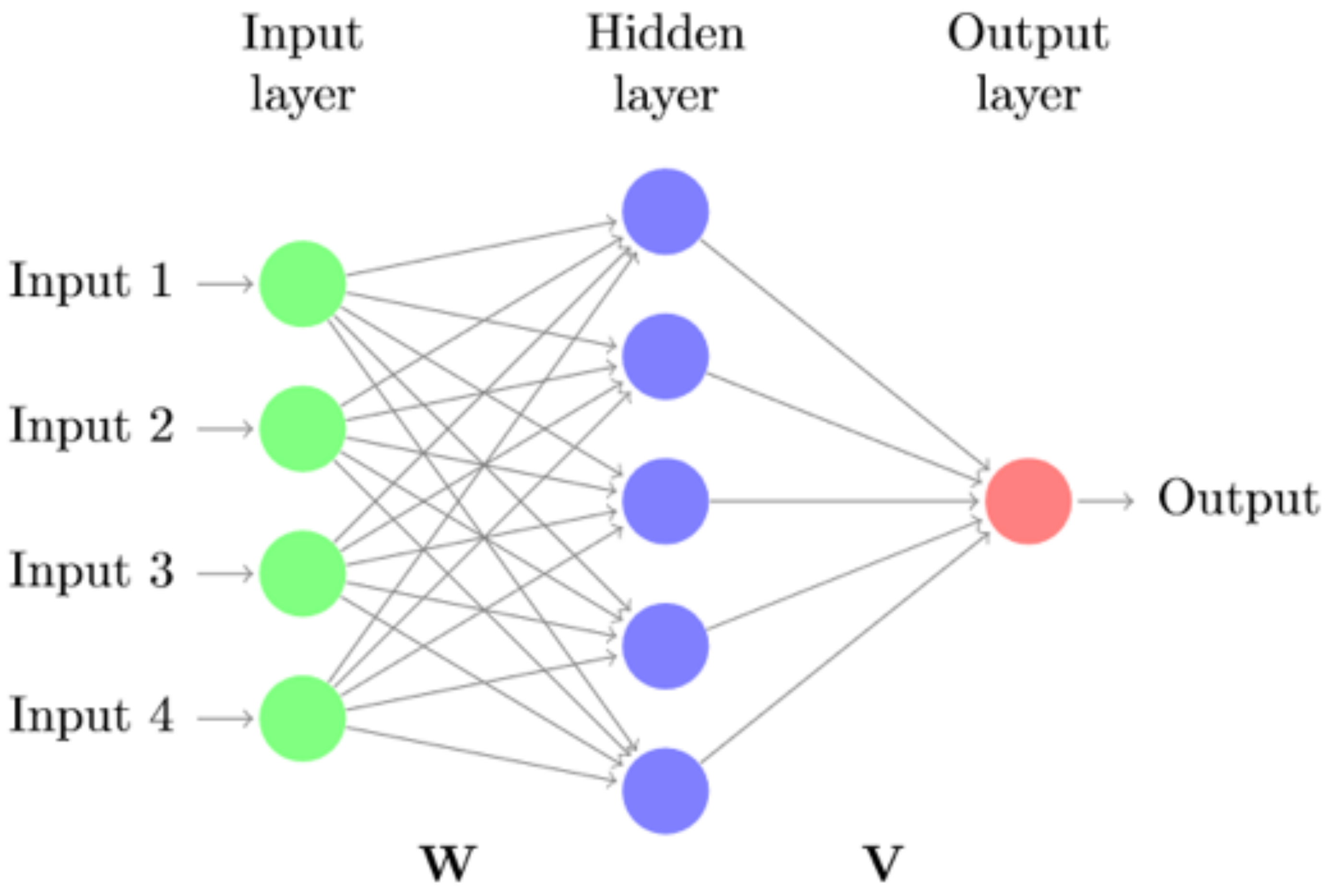
Adjust weights by gradient (using a *learning weight*) of the loss. Save repeated partial computations along the way.

$$\Delta w_i = \frac{\partial}{\partial w_i} Loss(f(\mathbf{W}, \mathbf{V}, \ldots, \mathbf{x}), target)$$



loss functions

- **Loss function –** measures NN’s performance.
  - Probabilistic interpretation
    - Binary output - **Binary Cross Entropy** and Sigmoid
    - Multiclass/Sequence output - **Categorical Cross Entropy** and Softmax
    - either *Generative* or *Discriminative*
  - Geometric interpretation
    - **Mean Squared Error** or **Hinge Loss** (like in Structured Perceptron)


Input layer
Hidden layer
Output layer
Input 1
Input 2
Input 3
Input 4
Output
W
V

RNNs

- **Recurrent Neural Net** - Model a sequence of any length
- *Weight sharing, Unlimited history*
- (also – LSTM, GRU, Bidirectional)
- Applications:
  - *Language models*
  - *Language Generation*
  - *Sequence classification* - Part-of-Speech tagging
- Not just words (*characters, structured data, …)*

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

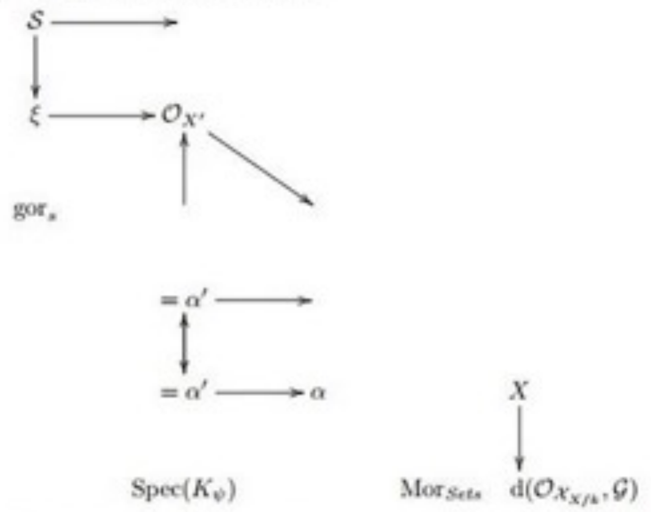
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

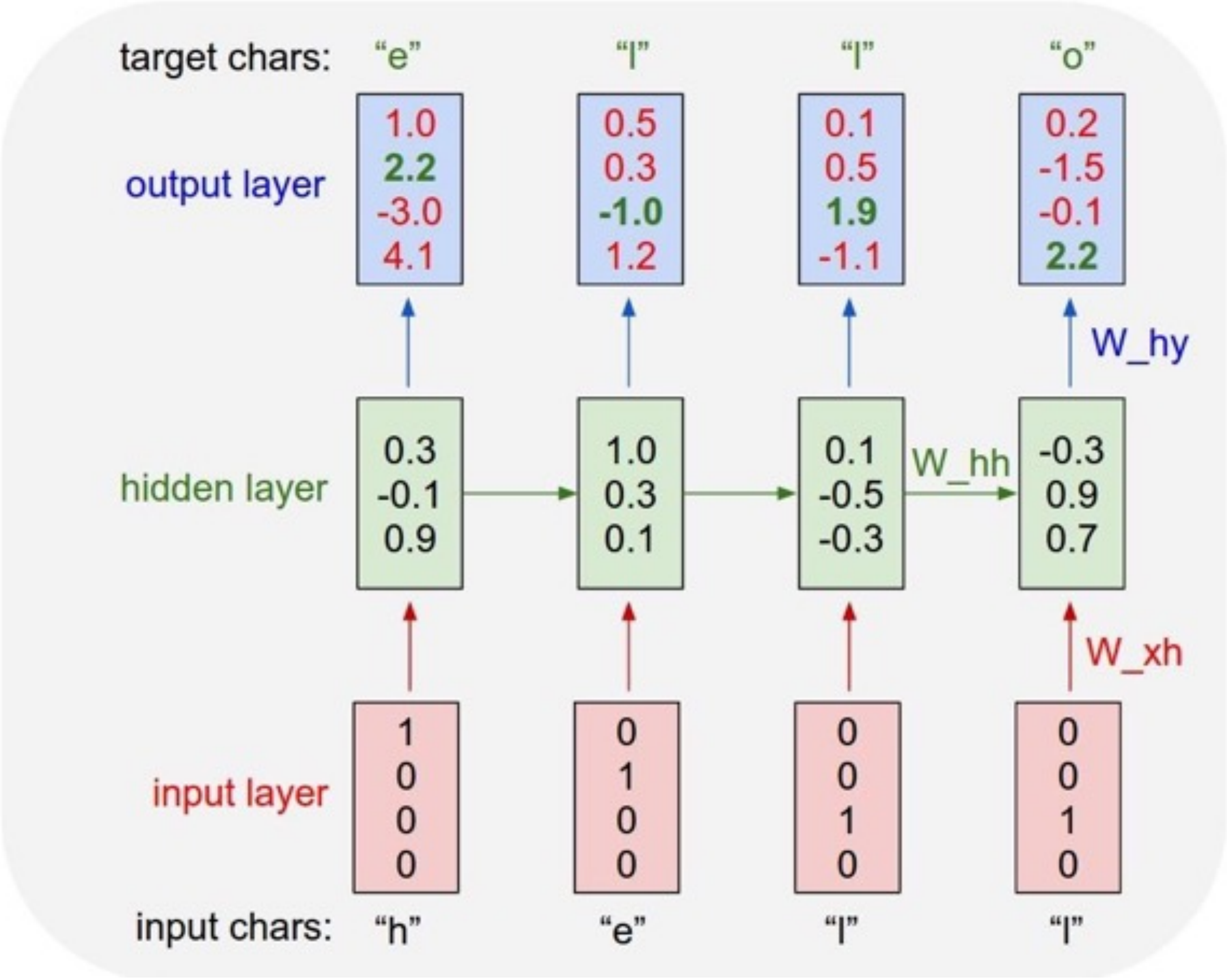
Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \ \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

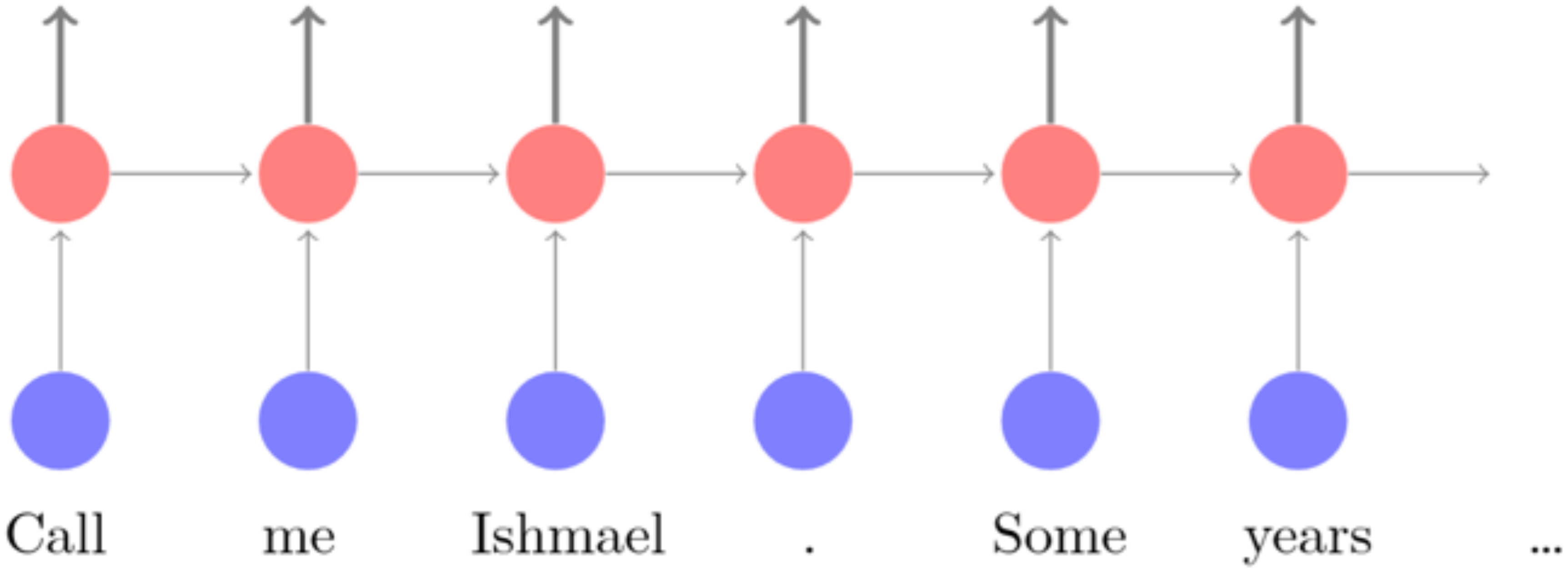
is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

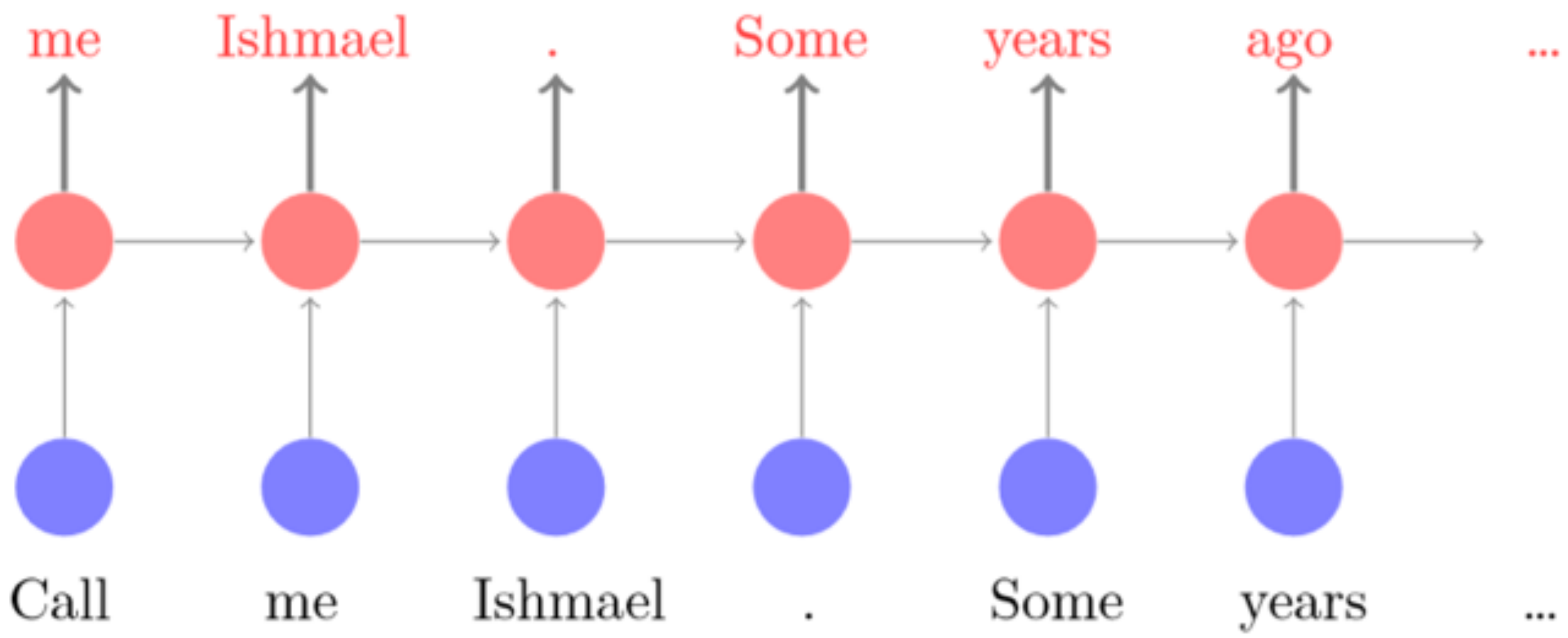


target chars:
"e"
"l"
"l"
"o"
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
"h"
"e"
"l"
"l"

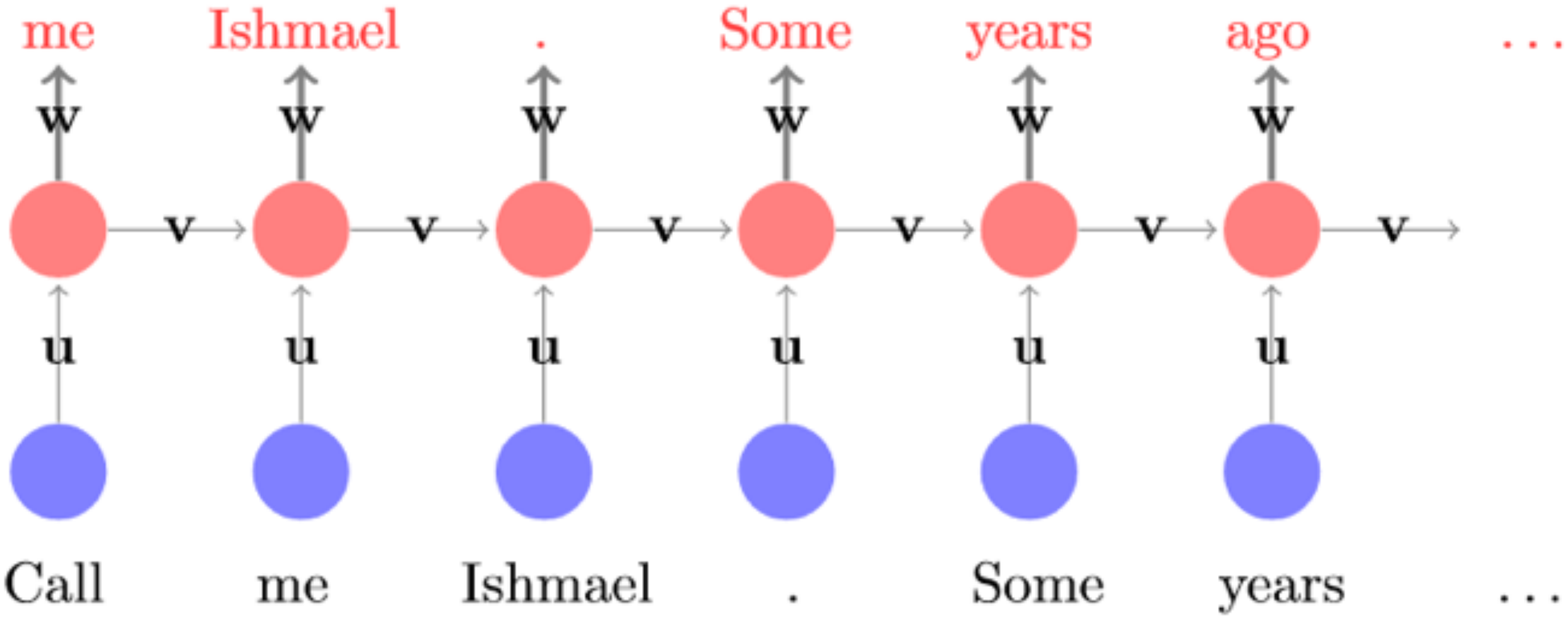
RNN



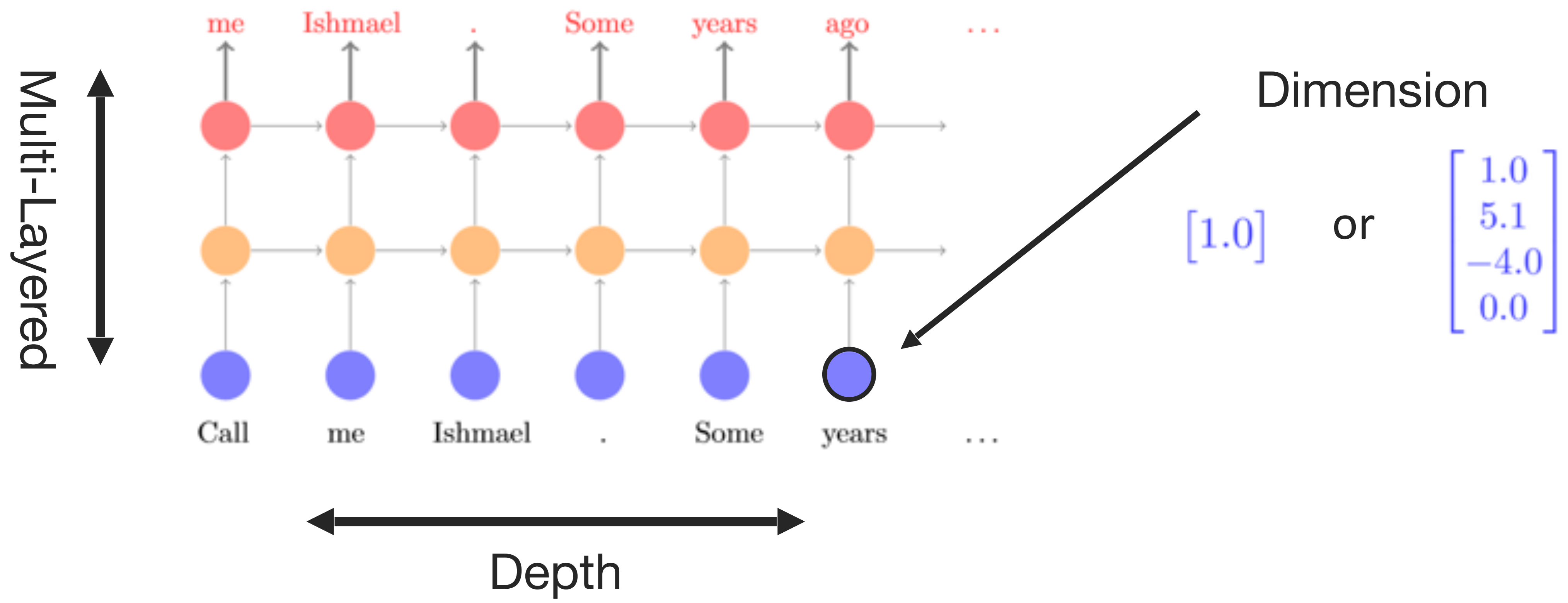
RNN Language Model



Weight Sharing

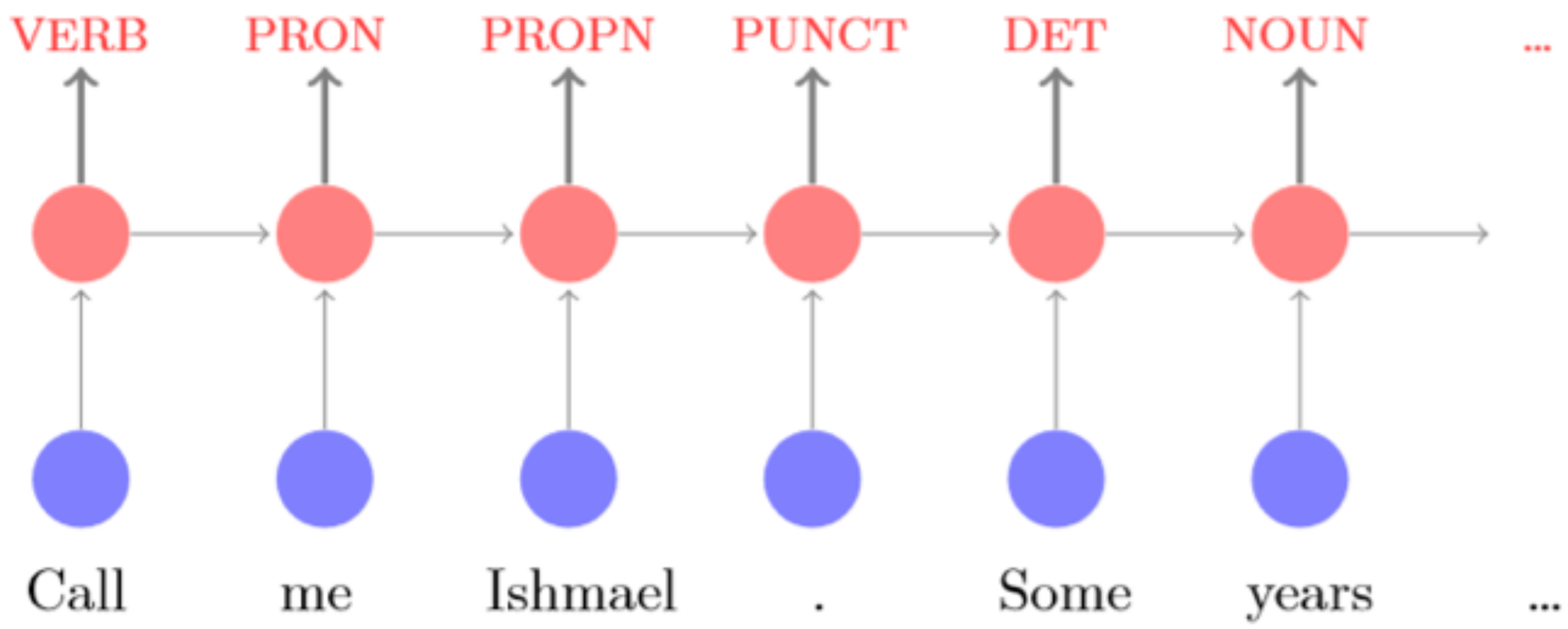


RNN Dimensions



RNN Part-of-Speech Tagger

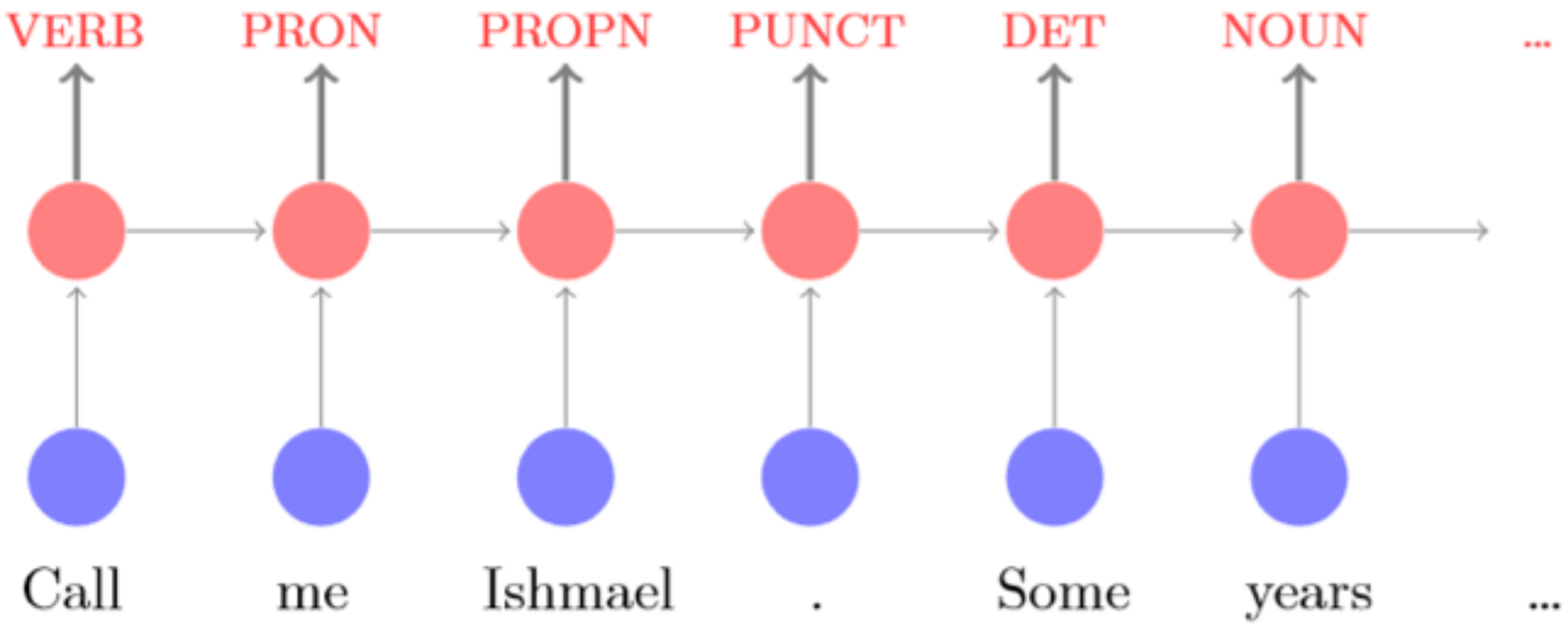
How is an RNN different than HMM?



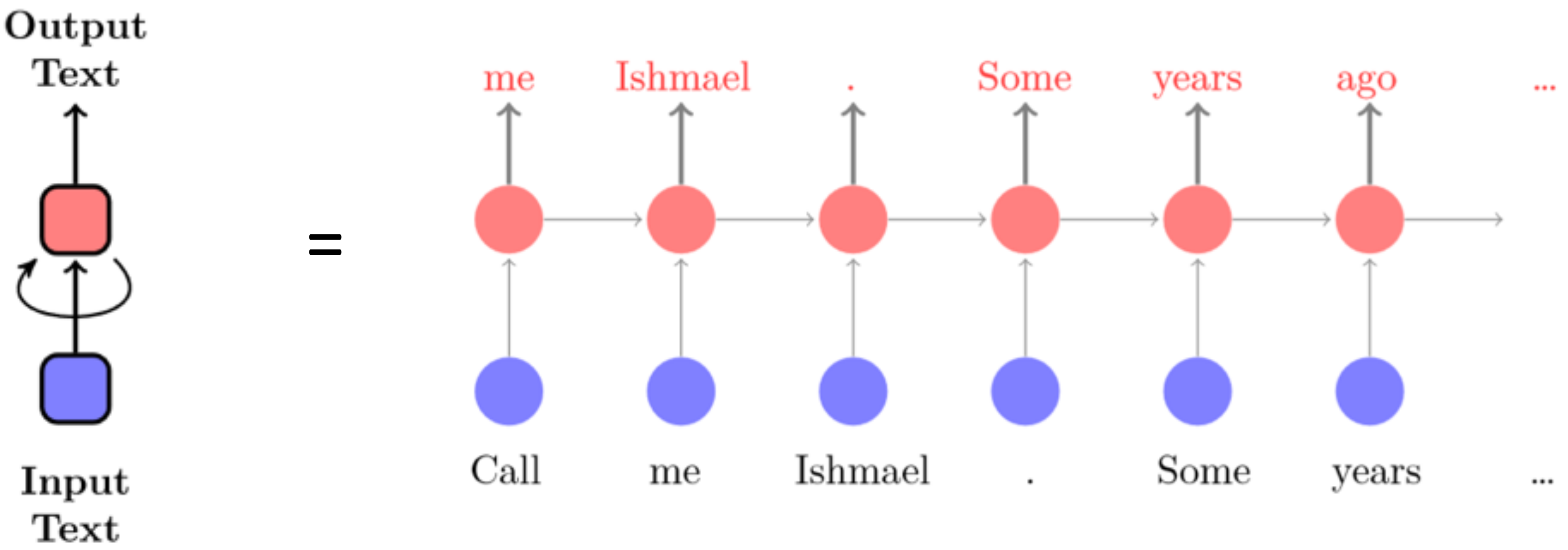
RNN Part-of-Speech Tagger

How is an RNN different than HMM?

Unlimited History

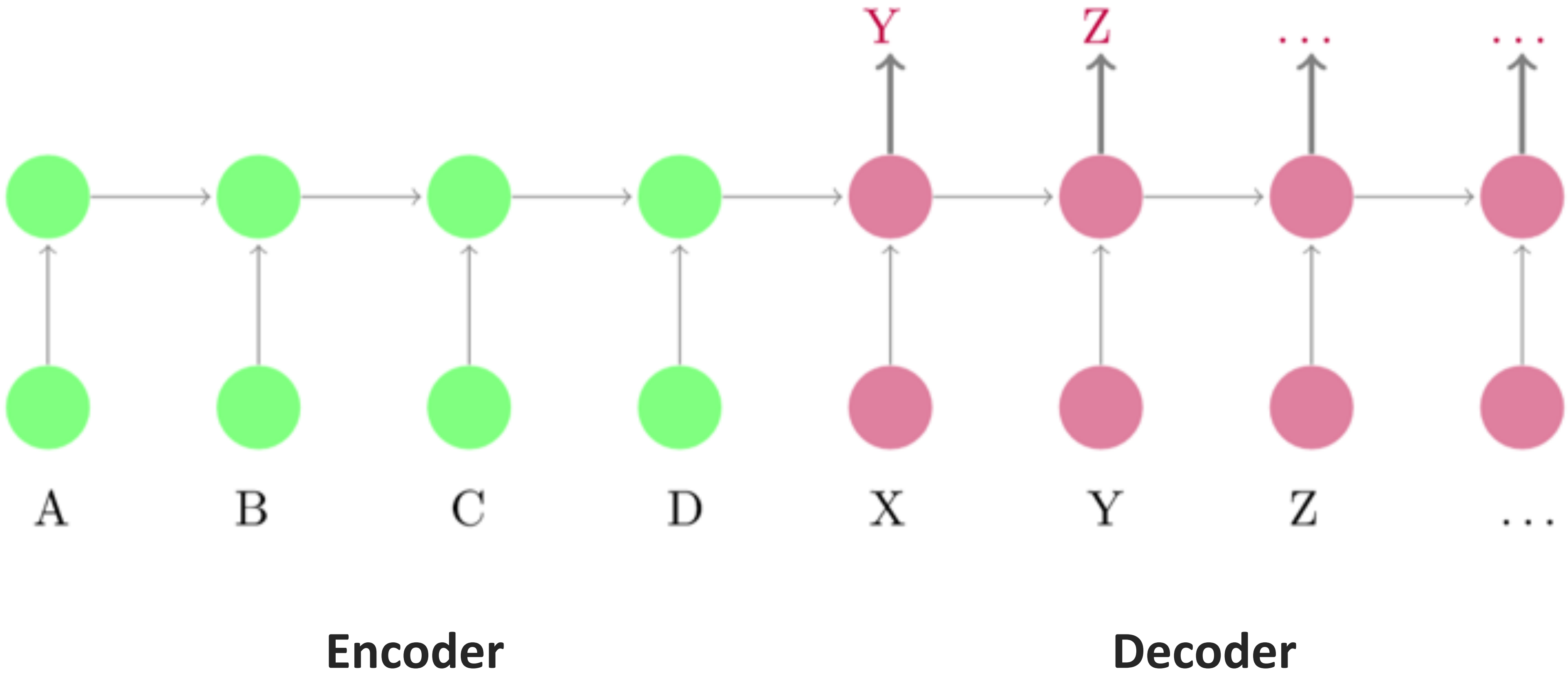


Compact diagram

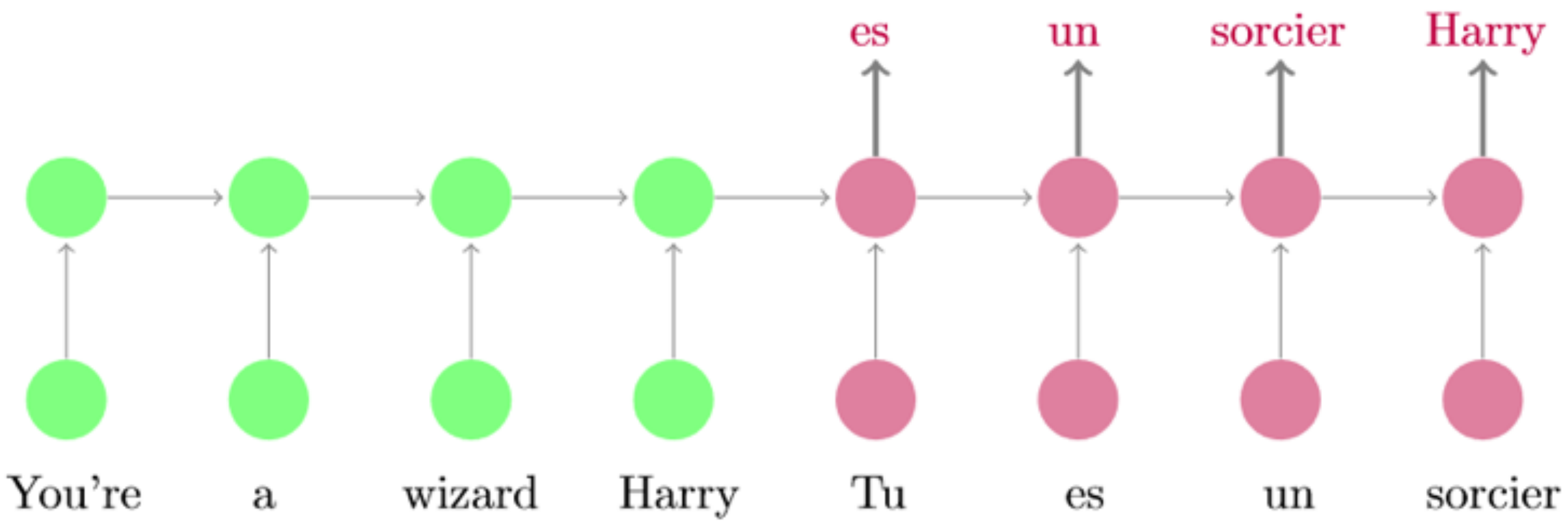


Encoder-Decoder models

- **Encoder-Decoder model** (also **Seq2Seq**) – Take a sequence as input and predict a sequence as output
- *Input and Output may be different lengths*
- Encoder (*RNN*) models input, Decoder (*RNN*) models output
- Applications:
  - *Machine Translation*
  - *Morphological Analysis*



Y
Z
...
...
A
B
C
D
X
Y
Z
...
Encoder
Decoder



Encoder (English) **Decoder** (French)

Embeddings

- **Embeddings -** Dense vector representations of *words, characters, documents, etc.*
- *Used as input features for most Neural NLP models*
- Prepackaged – *Word2Vec, Glove*
- Use pre-trained word embeddings *and* train them yourself!

Some References

- **NN Packages –** *TensorFlow*, *PyTorch*, *Keras*
- **Some Books**
  - Goldberg book (free from Georgetown)
  - Goodfellow book (Chapters and Videos)