CS60050 Machine learning

Neural Network Architectures

Somak Aditya

Sudeshna Sarkar

Some slides are taken from: Matt Gormley 10-301/10-601 Introduction to Machine Learning

Neural Networks Properties

- Practical considerations
  - Large number of neurons → Danger for overfitting
  - Gradient descent can easily get stuck local optima

- Universal Approximation Theorem:
  - A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

The success of NN

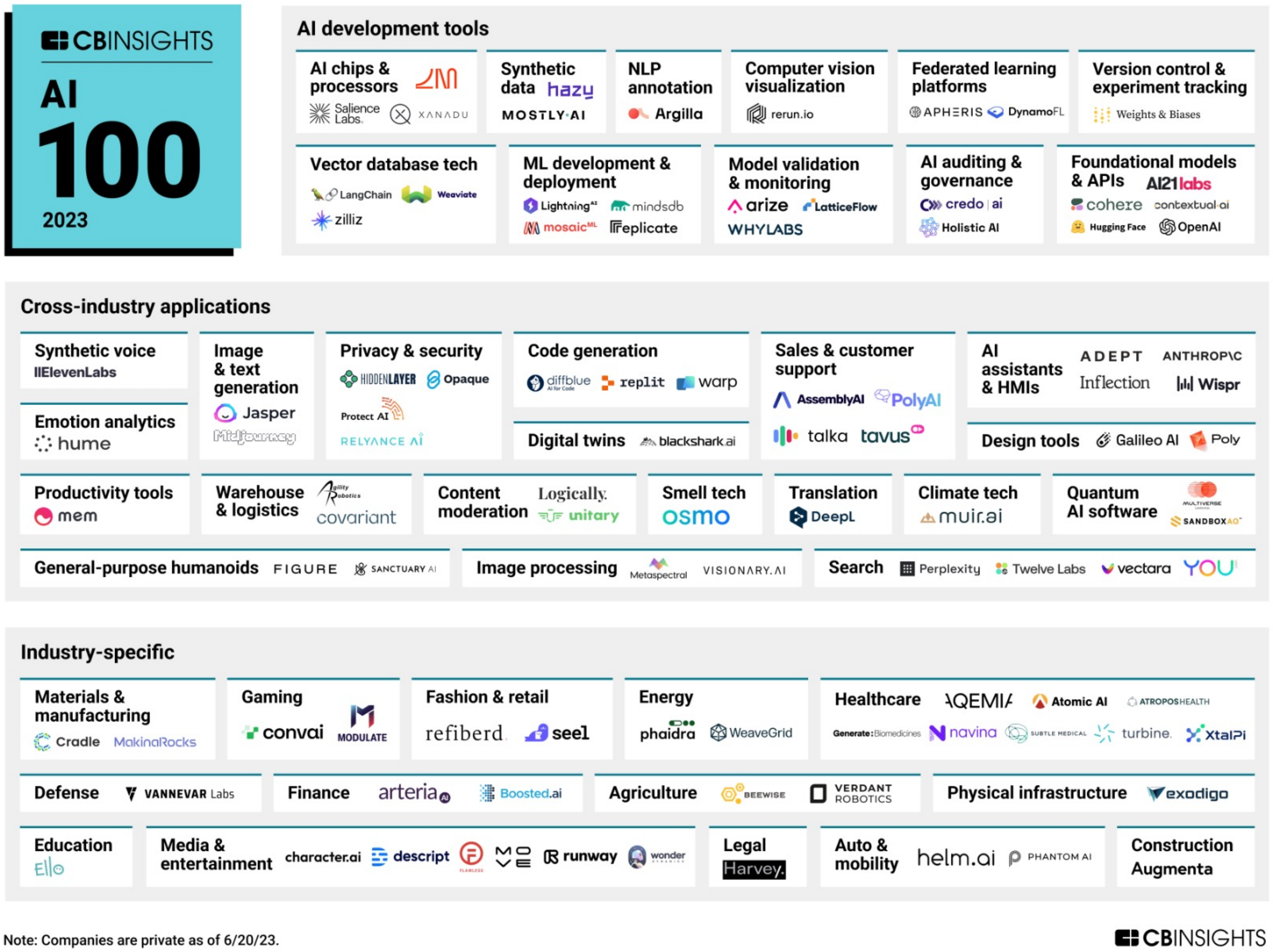
Yann Lecun: DNNs require: “an interplay between intuitive insights, theoretical modeling, practical implementations, empirical studies, and scientific analyses”

1. More data
2. More computational power
3. Improved techniques (though they’re not brand-new)

But, Driven primarily by intuition and empirical success

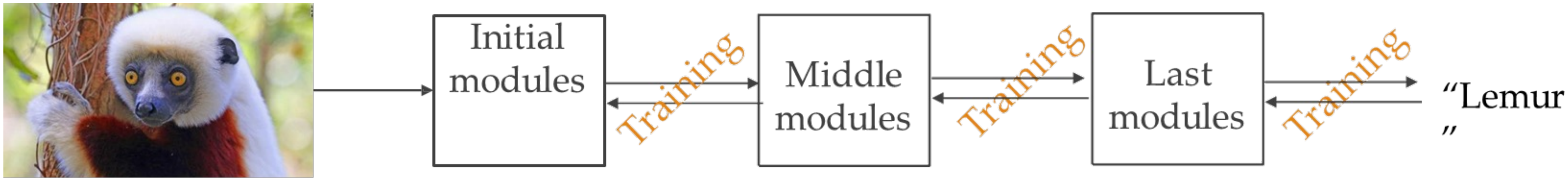
1. Good research and progress based on
   - Intuition, Practice (empirical findings)
2. Theory lags dramatically
   - No guarantees, little understanding of limitations, limited interpretability
3. More interestingly, classic theory suggests currently successful DL practices, wouldn’t be likely to succeed.

Deep Learning Applications are Everywhere



Deep learning ⇔ Learning Hierarchical Representations

- A pipeline of successive, differentiable modules (transformations)
  - Each module's output is the input for the next module
- Each subsequent module produce higher abstraction features



CNN and Computer Vision

CNN Attributed mainly to Yann Lecun and supervisor Hinton (Turing Award Winner)

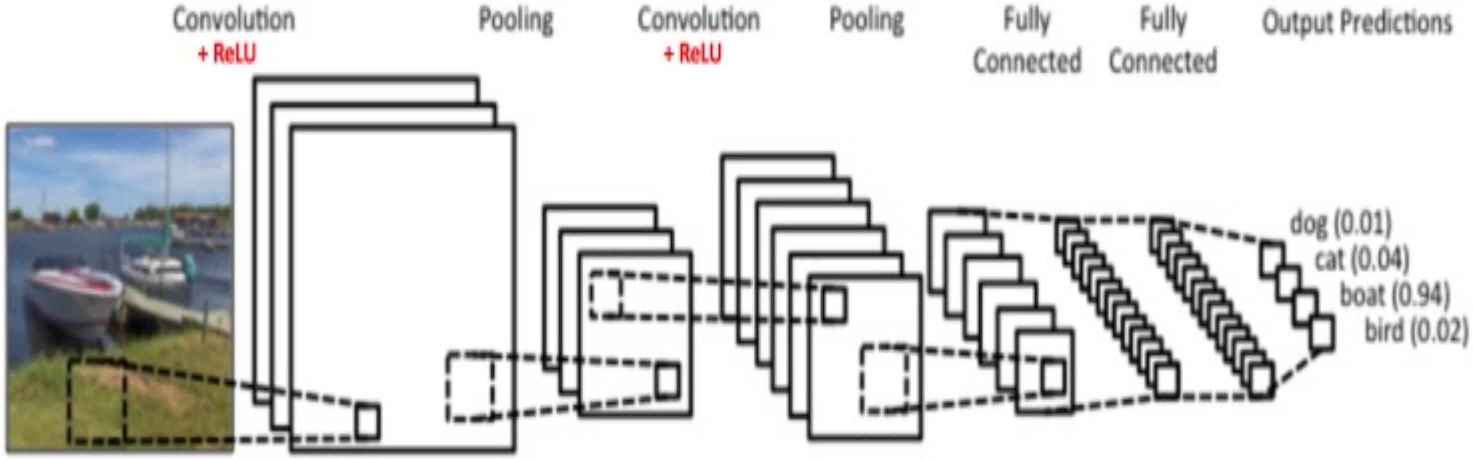
Convolutional Neural Networks

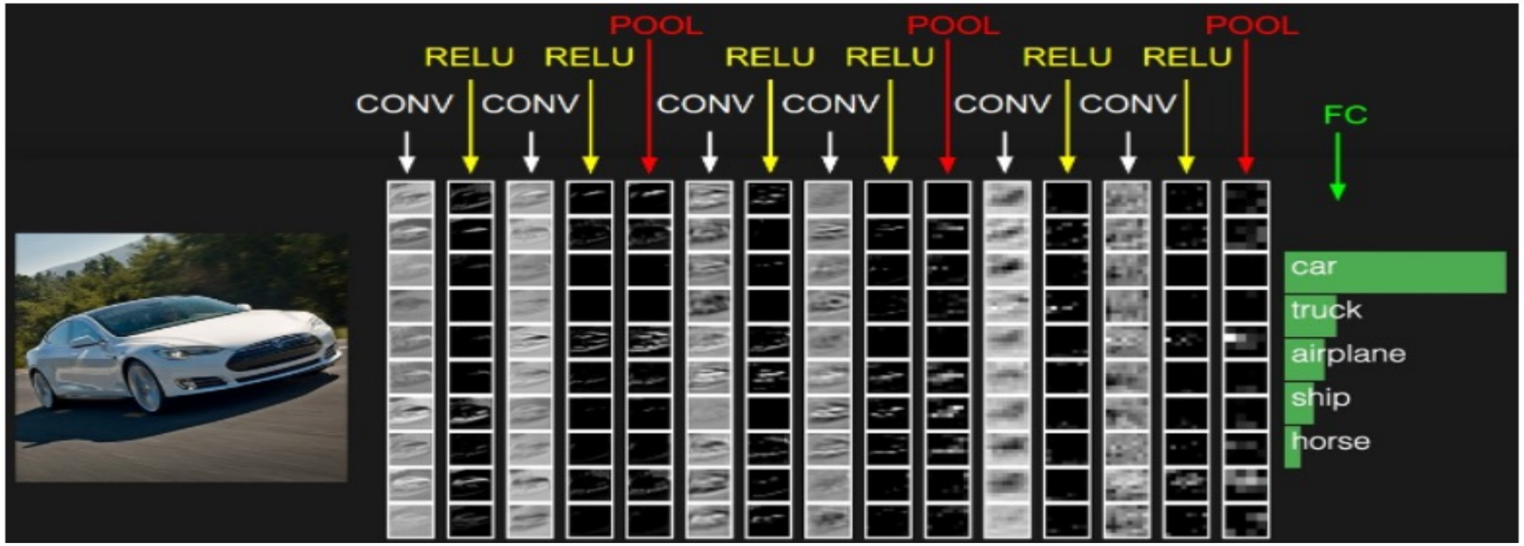
- Some neural networks have "Special" structures
- There are sparse connections between adjacent layers (except the last layer)
- Many edges between two layers have "shared weights"
  - This reduces number of parameters, and helps to capture local properties of the input
- Especially suitable for "structured" inputs such as images

Convolutional Neural Networks

- Traditional ML:
  - Hand-coded: $X_i \rightarrow x_i$
  - Learn: $f(x)$ s.t. $\mathcal{L}(f(x_i), y_i)$ is minimized.
- Deep Learning
  - Learn $f(X)$ s.t. $\mathcal{L}(f(X_i), y_i)$ is minimized.

- CNN = Convolution, ReLU Pooling, Fully Connected Networks





Visualization Playground: [2D Visualization of a Convolutional Neural Network (ryerson.ca)]

Convolution

Vertical Edge detection

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

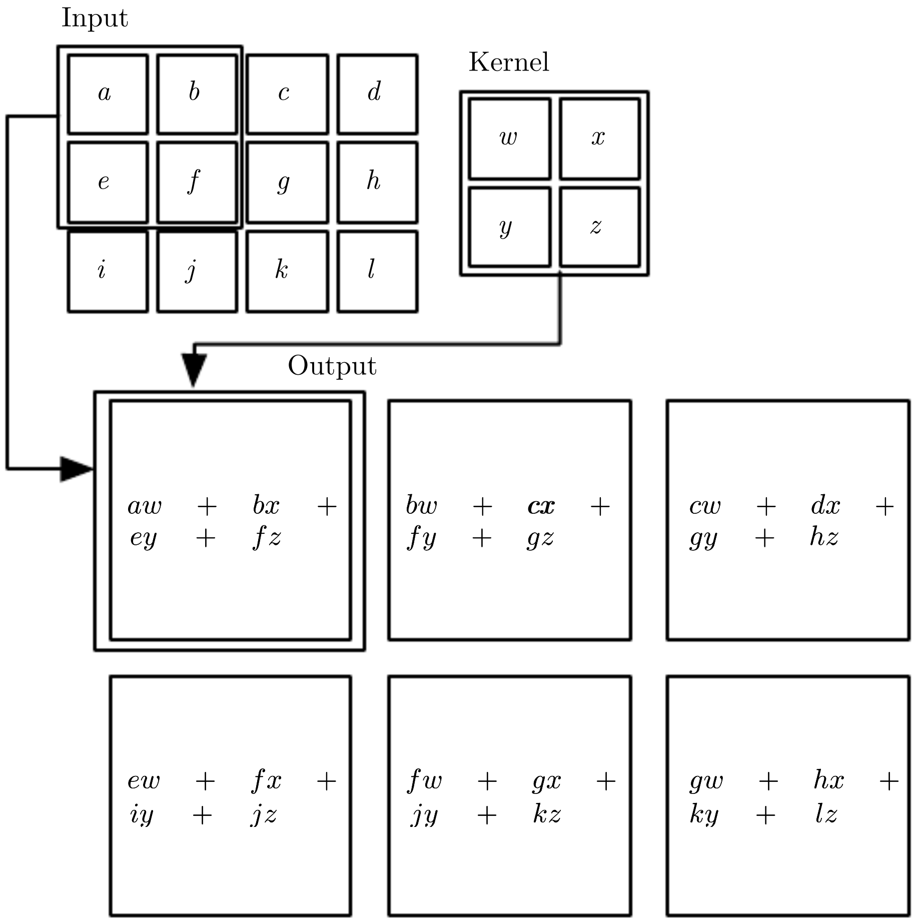
A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

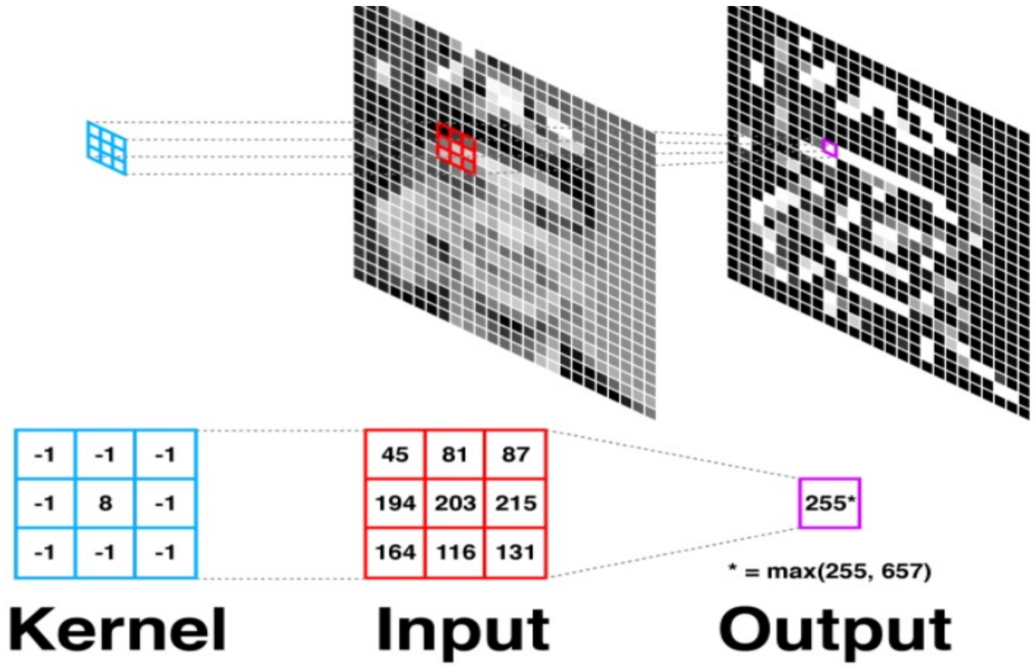
Basic idea:

- Pick a 3x3 matrix F of weights
- Slide this over an image and compute the "inner product" (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation

- Key point:
  - Different convolutions extract different types of low-level "features" from an image (automatically)
  - All that we need to vary to generate these different features is the weights of F

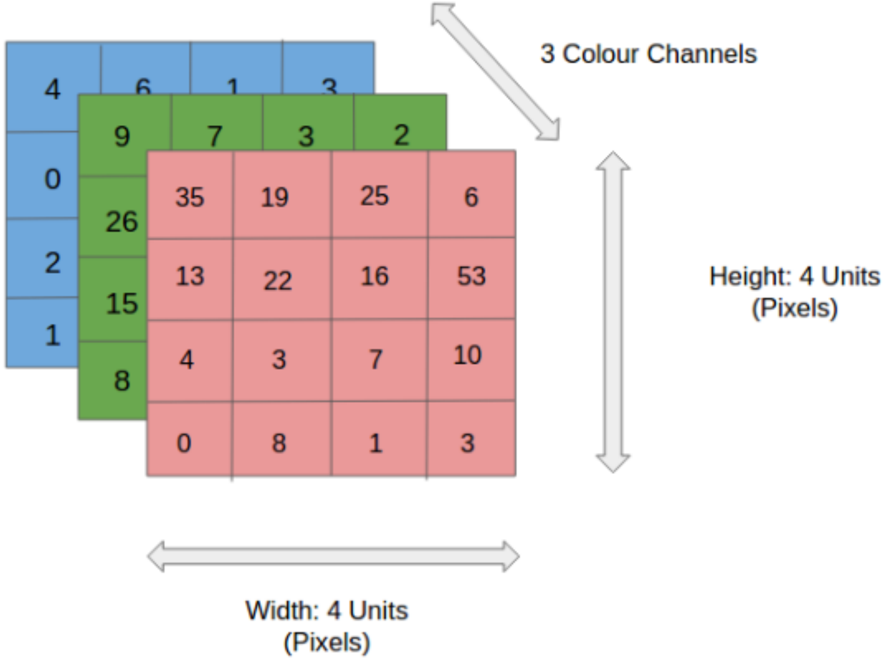
Convolution Operator



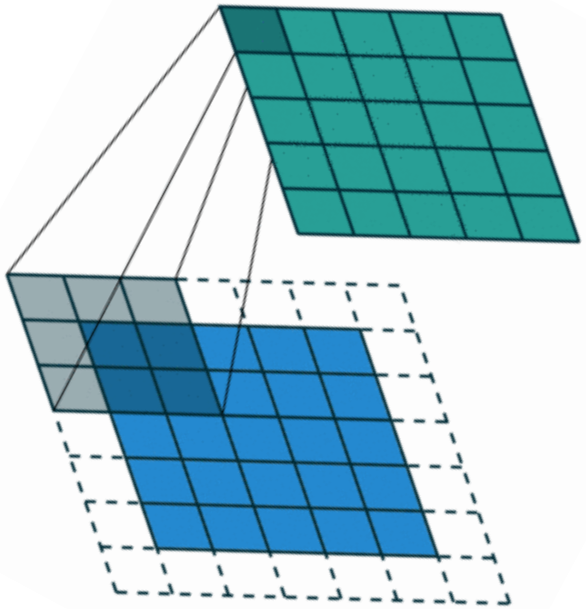


(-1*45 + -1 * 81 + -1 *87) + (-1*194 + 8 * 203 + -1 *215) + (-1*164 + 8 * 116 + -1 *131)

Convolution operator

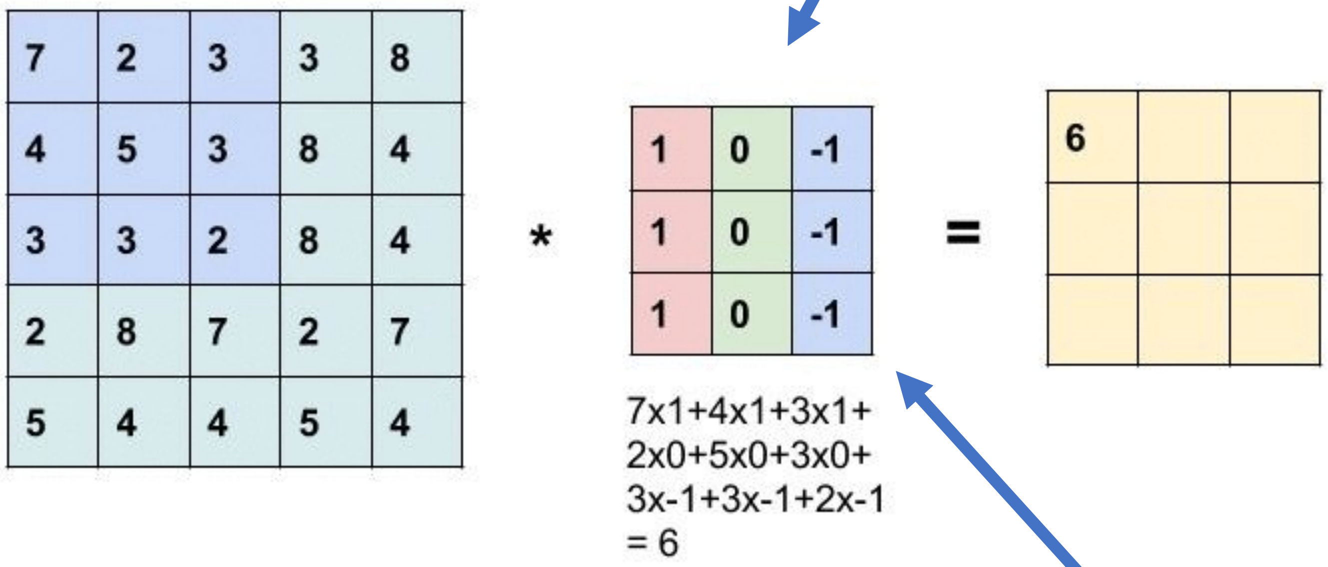


Kernel Size: 3x3x1
Stride: 1
Zero-Padding



Convolution: Math

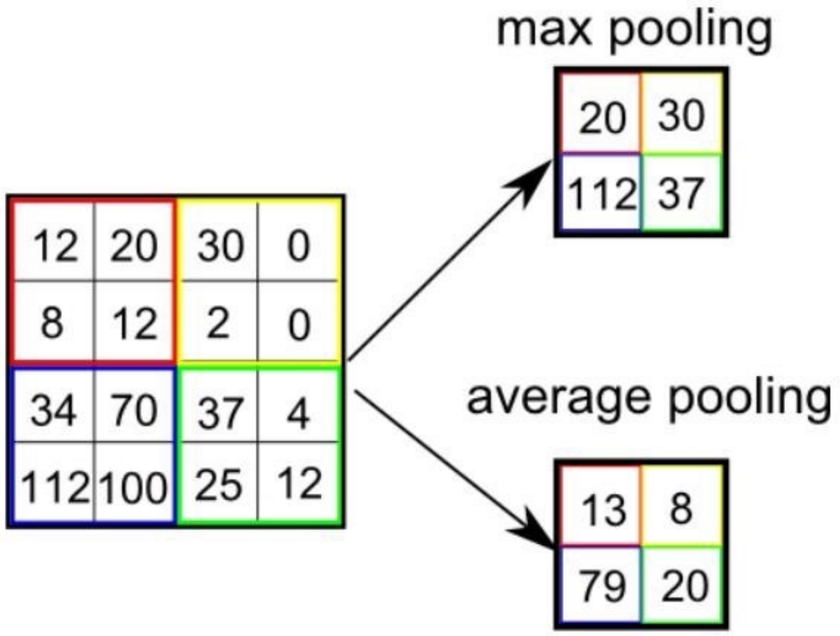
Filters capture spatial/temporal dependencies
Reduces dimensionality → Faster



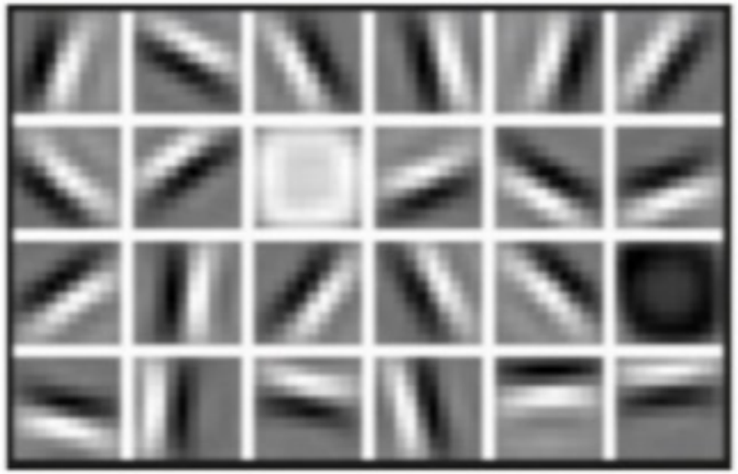
BUT: Filter weights need to be learned

Pooling

- Reduce Image Size
- Max/Min/Average Pooling



Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

Facial Structure

Input---**Shallow Layers**---------**Middle Layers**-------**Deeper Layers** ----> Output

CNN - Properties

- Reduced amount of parameters to learn (local features)
- More efficient than dense multiplication
- Specifically thought for images or data with grid-like topology

Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies

PROC. OF THE IEEE, NOVEMBER 1998 7

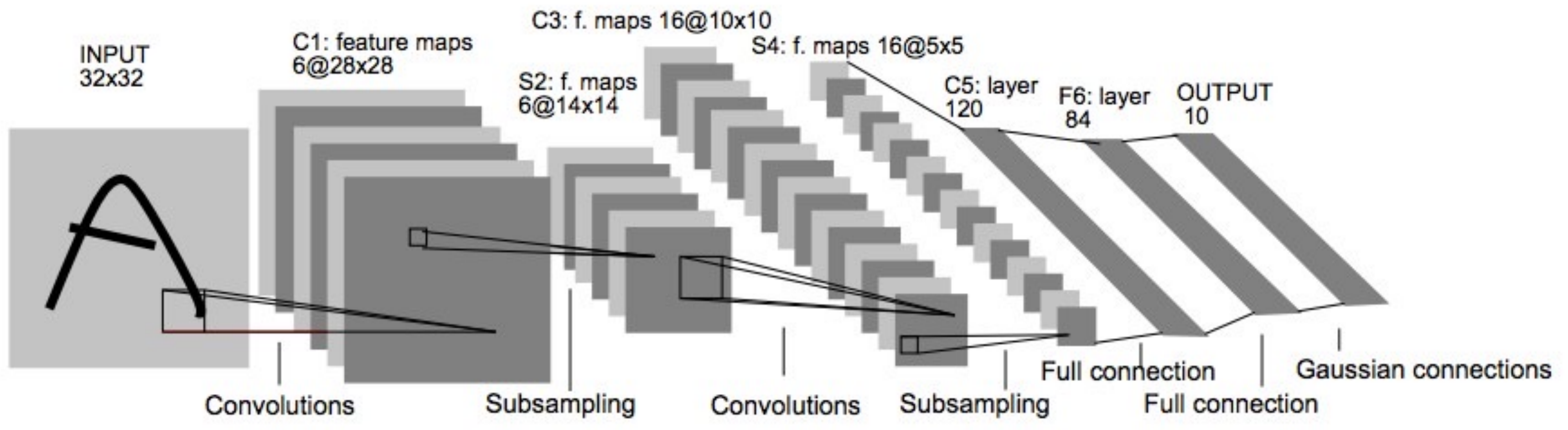
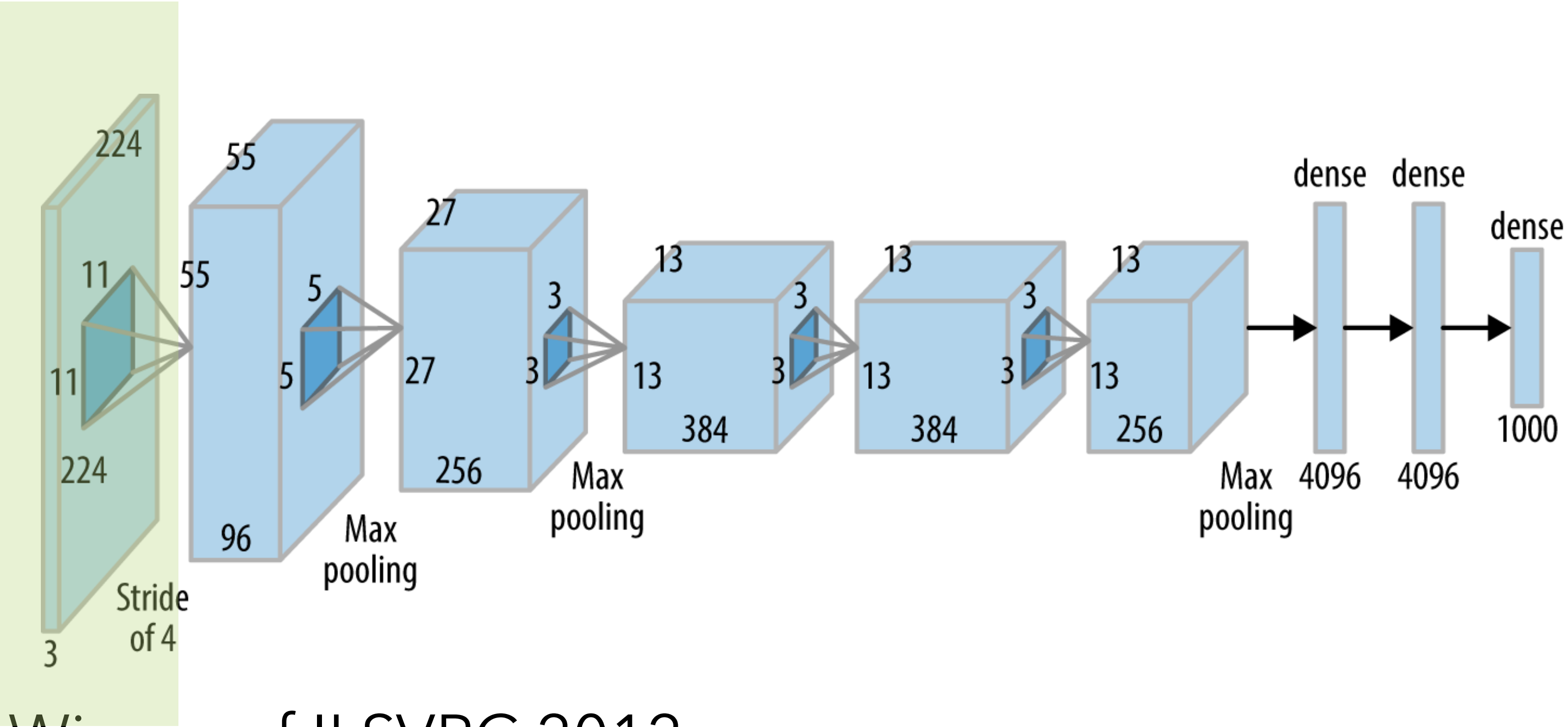


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5

AlexNet



- Winner of ILSVRC 2012
- Marked the beginning of recent deep learning revolution

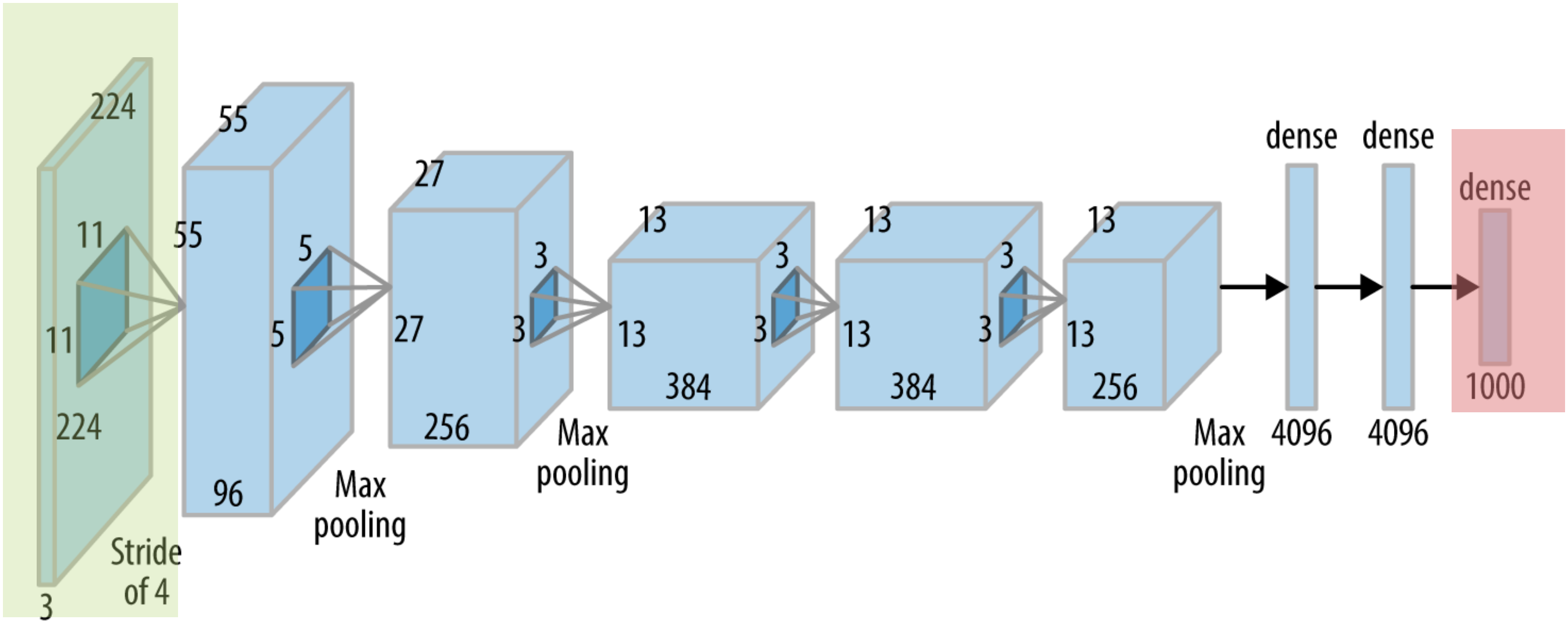
A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet Classification with Deep Convolutional Neural." In *NIPS*, pp. 1-9. 2014.

AlexNet

Input image (pixels)

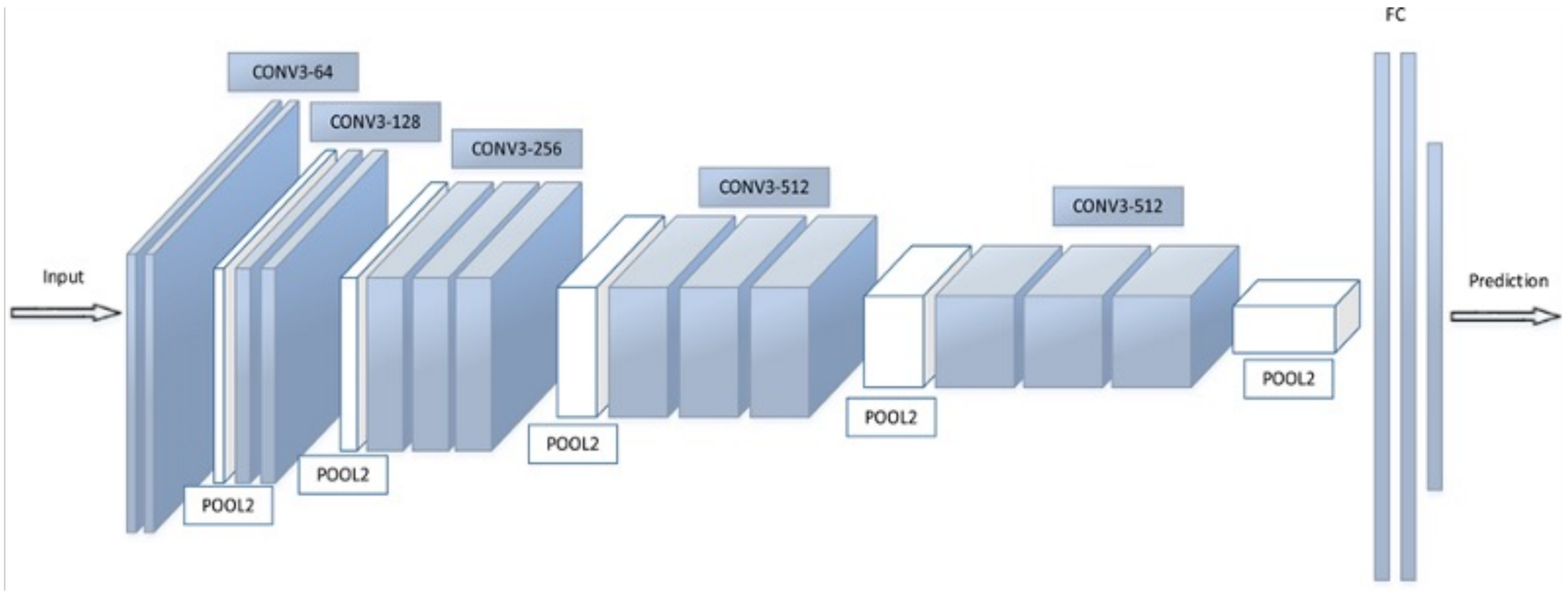
- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet Classification with Deep Convolutional Neural." In *NIPS*, pp. 1-9. 2014.

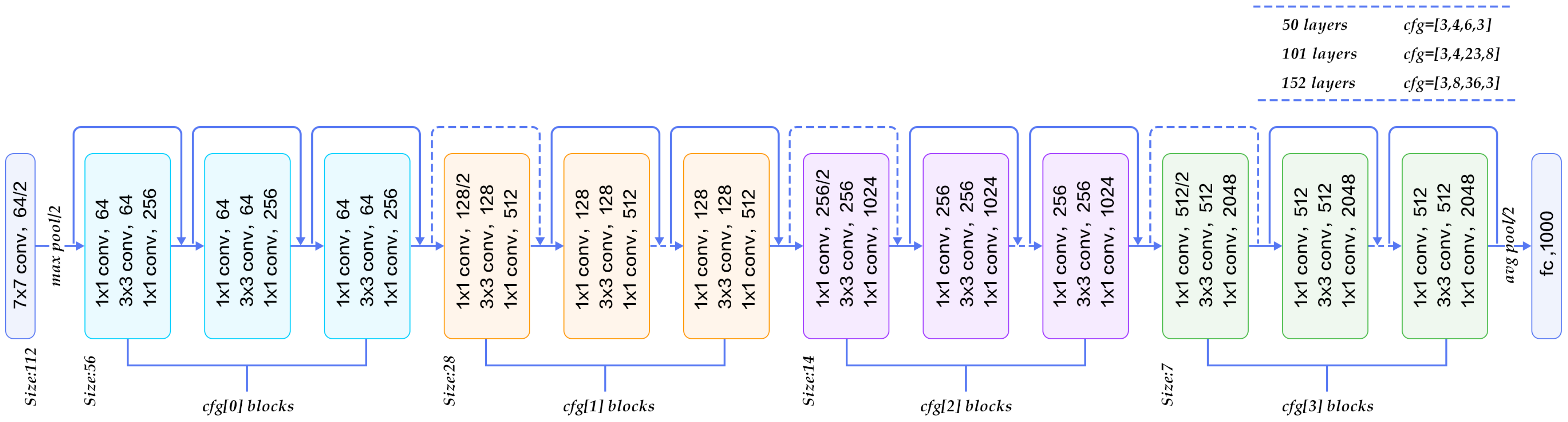
VGG-16



- Very small filters (3x3)
- Deeper than AlexNet:16 layers

K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations, 2015.

ResNet

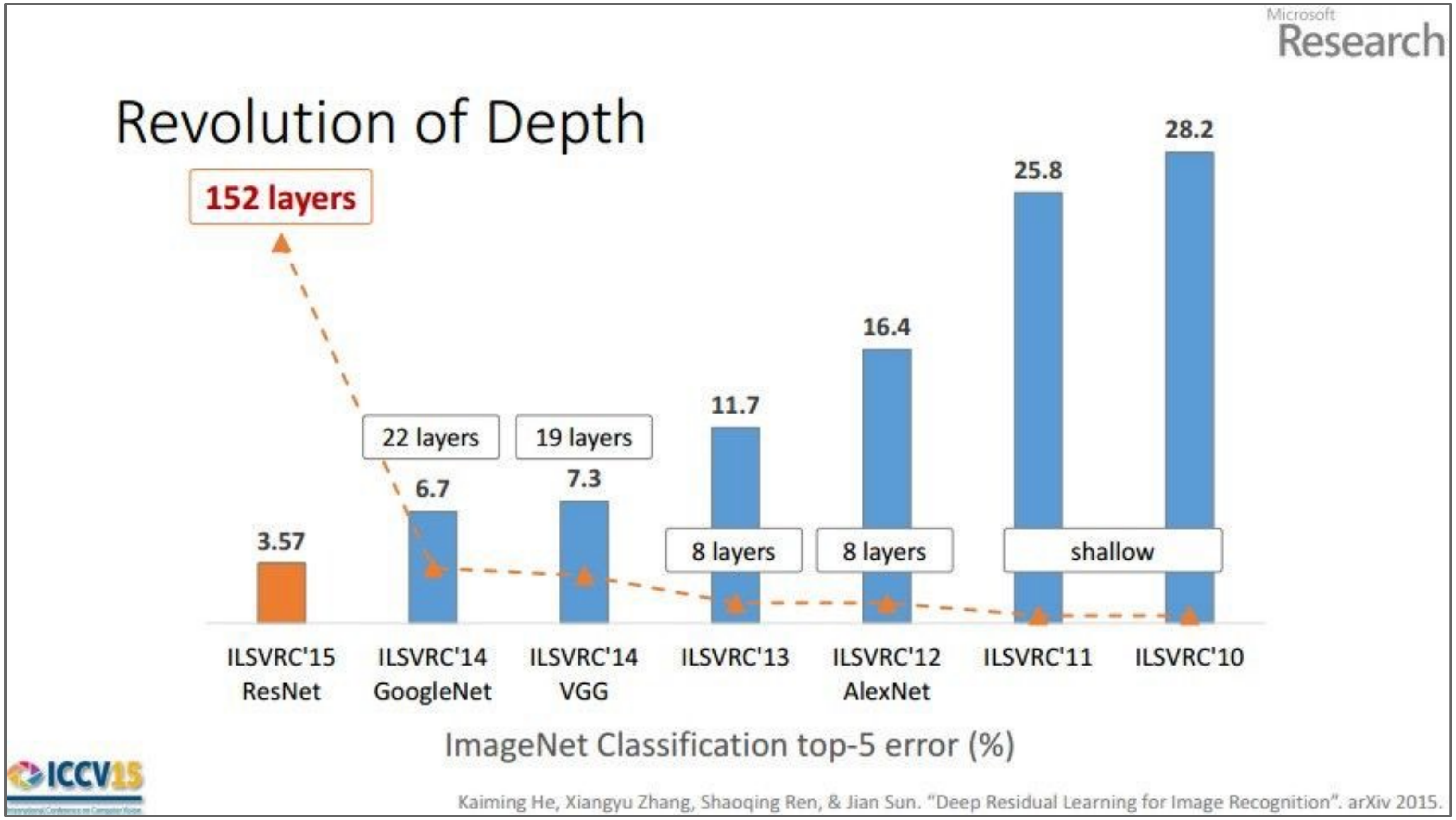


From: https://www.codeproject.com/Articles/1248963/Deep-Learning-using-Python-plus-Keras-Chapter-Re

- Increase the number of layers by introducing a residual connection
- Blocks are actually learning residual functions: easier!

K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

CNNs for Image Recognition



Slide from Kaiming He

CNN Summary

- **CNNs**
  - Are used for all aspects of **computer vision**, and have won numerous pattern recognition competitions
  - Able learn **interpretable features** at different levels of abstraction
  - Typically, consist of **convolution** layers, **pooling** layers, **nonlinearities**, and **fully connected** layers

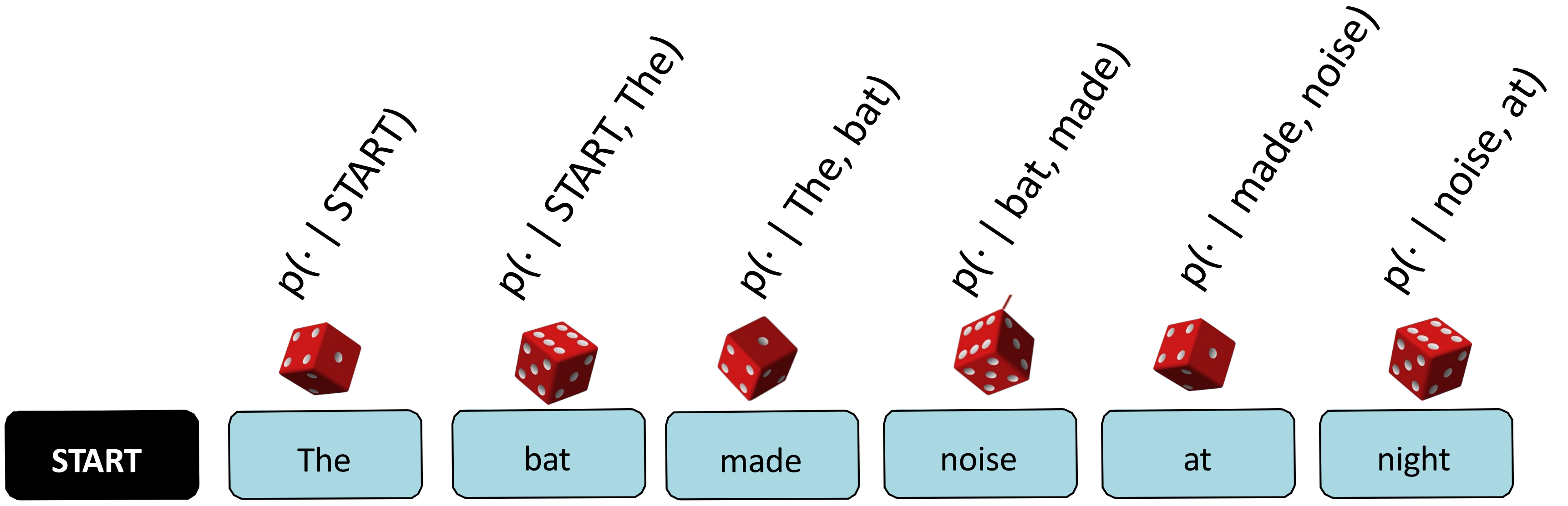
Transfer Learning

- Training such networks is a long and complex process, requiring lots of labelled data very powerful machines.
- Instead of retraining a new network completely from scratch, we could recycle existing networks, already built and trained by others on **similar** data.
- In Transfer Learning a model developed for a task is reused as the starting point for another model on a second task.
- On top of a previously trained network we add one or more neural layers
- We freeze all or some of the previously trained layers
- And we retrain only the remaining part of the whole network on our new task

NLP and RNN

n-Gram Language Model

- Goal: Generate realistic looking sentences in a human language
- Key Idea: condition on the last n-1 words to sample the n^{th} word



n-Gram Language Model

Question: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w1	w2	w3	w4	w5	w6

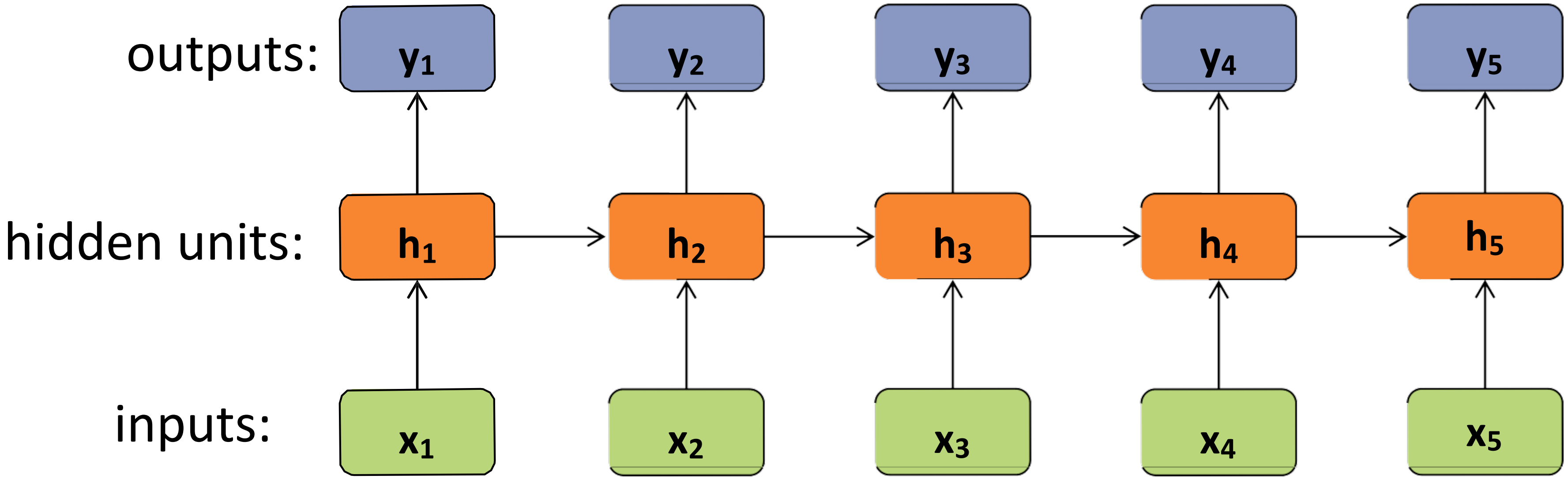
- n-Gram Model (n=3)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

- We made an assumption about how many previous words to condition on $(n - 1)$

RECURRENT NEURAL NETWORK (RNN) LANGUAGE MODELS

Recurrent Neural Networks (RNNs)



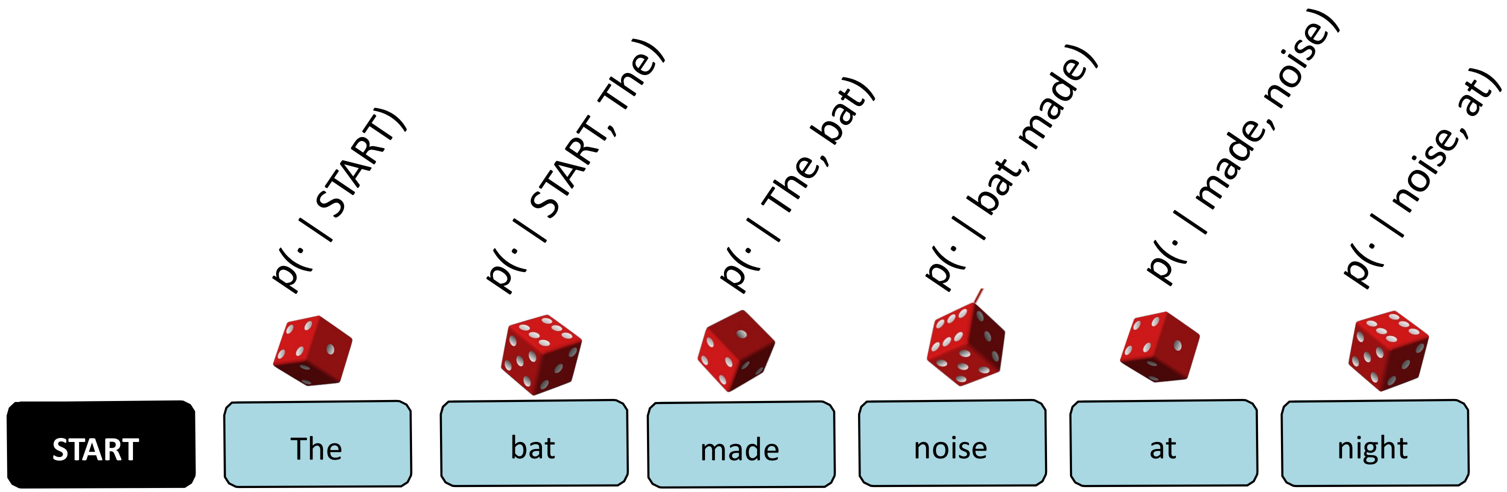
Definition of the RNN:

$$h_t = g\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right) \; y_t$$

$$= W_{hy} h_t + b_y$$

N-gram language models

- Goal: Generate realistic looking sentences in a human language
- Key Idea: condition on the last n-1 words to sample the nth word



n-Gram Language Model

<u>Question</u>: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=2)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The — $p(w_1)$

The bat — $p(w_2 \mid$

bat made — $p(w_1)$ $p(w_3 \mid w_2)$

made noise — $p(w_4 \mid w_3)$

noise at — $p(w_5 \mid w_4)$

at night — $p(w_6 \mid w_5)$

n-Gram Language Model

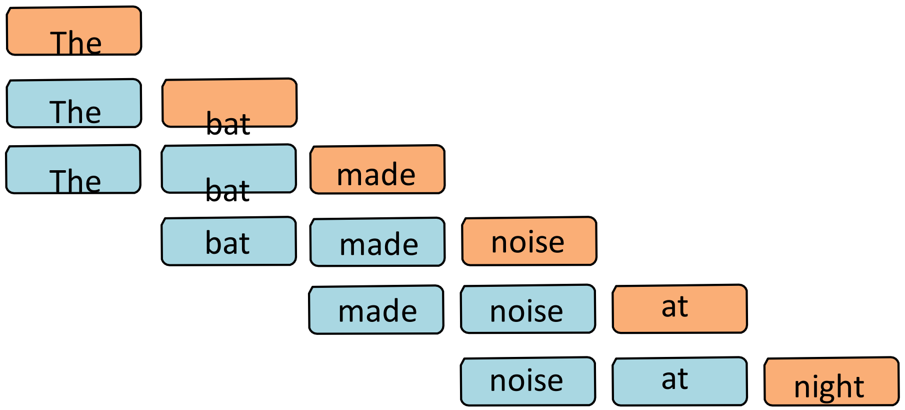
<u>Question</u>: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=3)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$



$p(w_1)$

$p(w_2 \mid$

$p(w_1)$ $p(w_3 \mid w_2, w_1)$

$p(w_4 \mid w_3, w_2)$

$p(w_5 \mid w_4, w_3)$

$p(w_6 \mid w_5, w_4)$

n-Gram Language Model

<u>Question</u>: How can we **define** a probability distribution over a sequence of length T?

The	bat	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n-Gram Model (n=3)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The						$p(w_1)$
The	bat					$p(w_2 \mid$
The	bat	made				$p(w_3 \mid w_2, w_1)$
	bat	made	noise			$p(w_4 \mid w_3, w_2)$
		made	noise	at		$p(w_5 \mid w_4, w_3)$
			noise	at	night	$p(w_6 \mid w_5, w_4)$

Note: This is called a **model** because we made some **assumptions** about how many previous words to condition on.

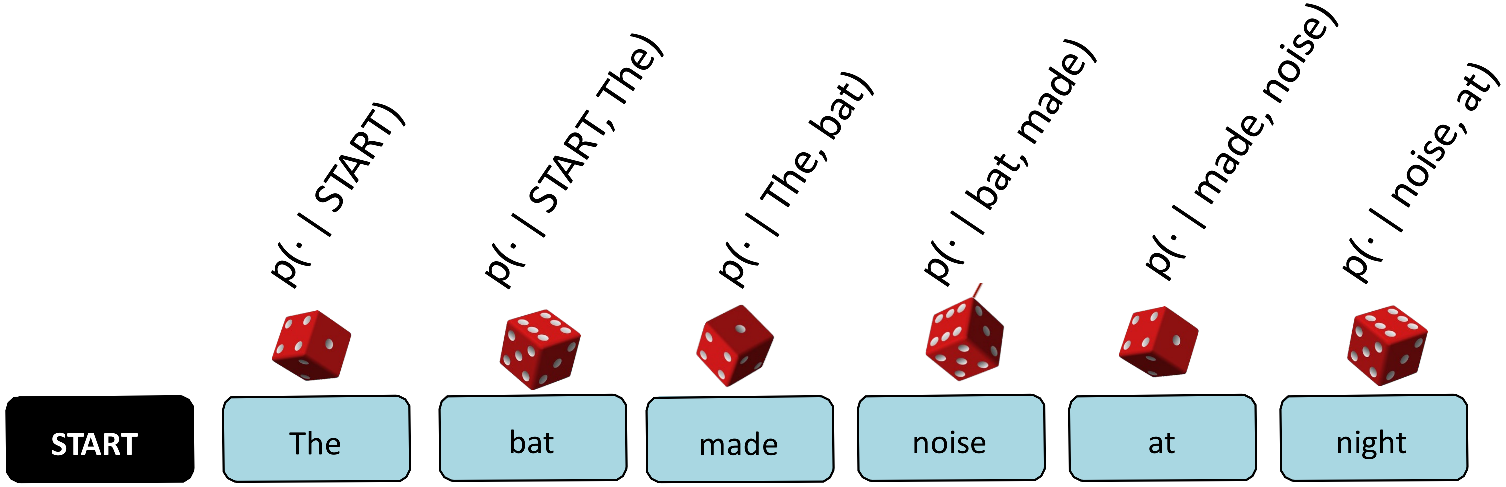
Learning an n-Gram Model

- Question: How do we learn the probabilities for the n-Gram Model?
- Answer: From data! Just count n-gram frequencies

Sampling from a Language Model

Question: How do we sample from a Language Model? ***Answer***:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T)$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T)$

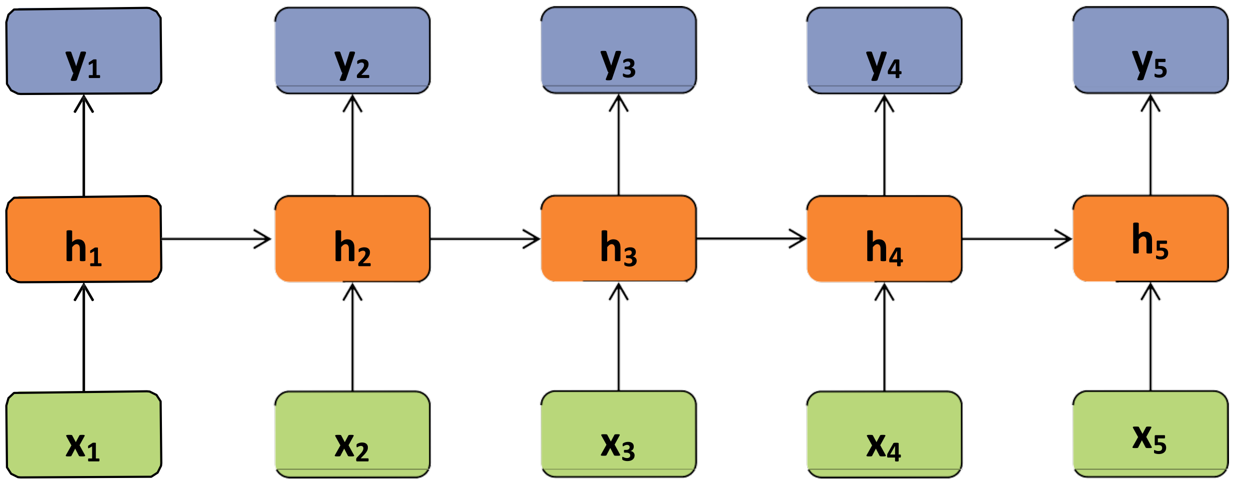
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T)$

nonlinearity: g

Definition of the RNN:

$h_t = g(W_{xh} x_t + W_{hh} h_{t-1} + b_h)$ $y_t =$

$W_{hy} h_t + b_y$

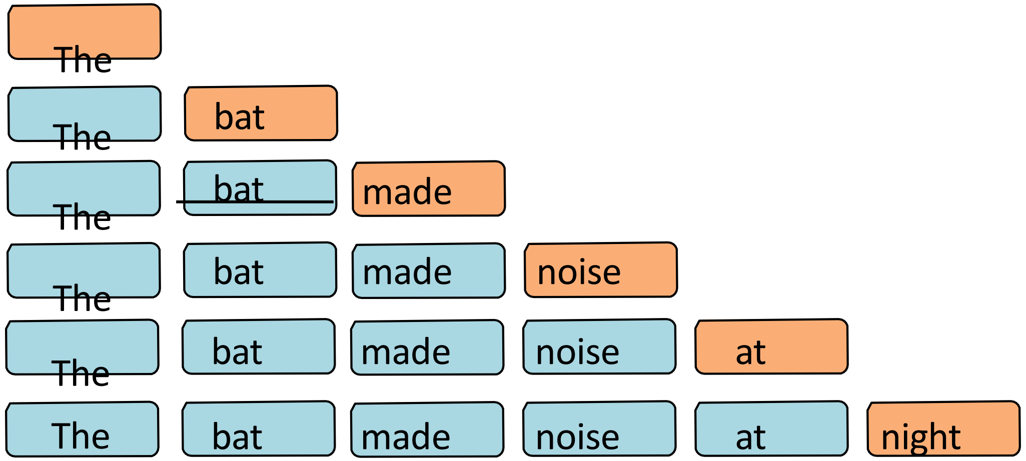


The Chain Rule of Probability

- *<u>Question</u>*: How can we **define** a probability distribution over a sequence of length T?
- **Chain rule of probability:**

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, \ldots, w_1)$$

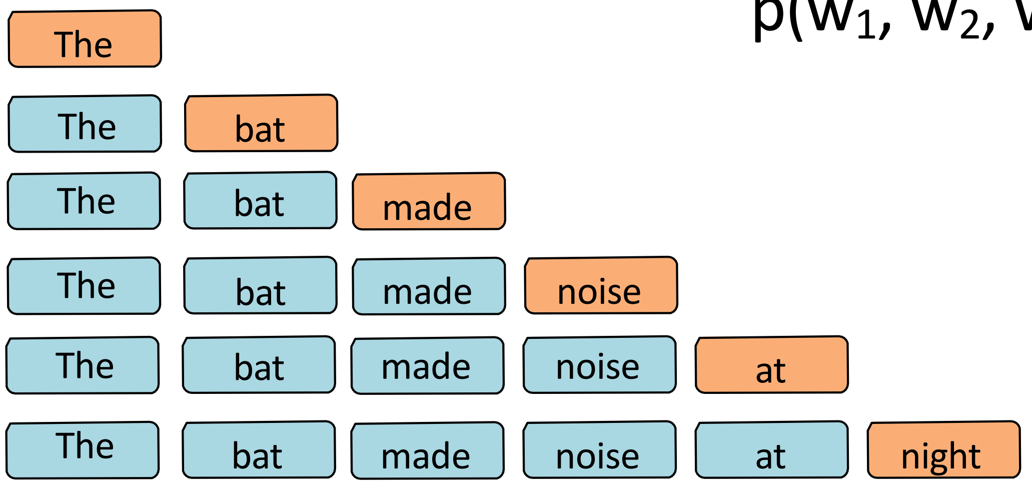
$$p(w_1, w_2, w_3, \ldots, w_6) =$$



$$p(w_1)$$
$$p(w_2 \mid w_1)$$
$$p(w_3 \mid w_2, w_1)$$
$$p(w_4 \mid w_3, w_2, w_1)$$
$$p(w_5 \mid w_4, w_3, w_2, w_1)$$
$$p(w_6 \mid w_5, w_4, w_3, w_2, w_1)$$

RNN Language Model

RNN Language Model: $$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid f_{\boldsymbol{\theta}}(w_{t-1}, \ldots, w_1))$$



$p(w_1, w_2, w_3, \ldots, w_6) =$

$p(w_1)$

$p(w_2 \mid f_\theta(w_1))$

$p(w_3 \mid f_\theta(w_2, w_1))$

$p(w_4 \mid f_\theta(w_3, w_2, w_1))$

$p(w_5 \mid f_\theta(w_4, w_3, w_2, w_1))$

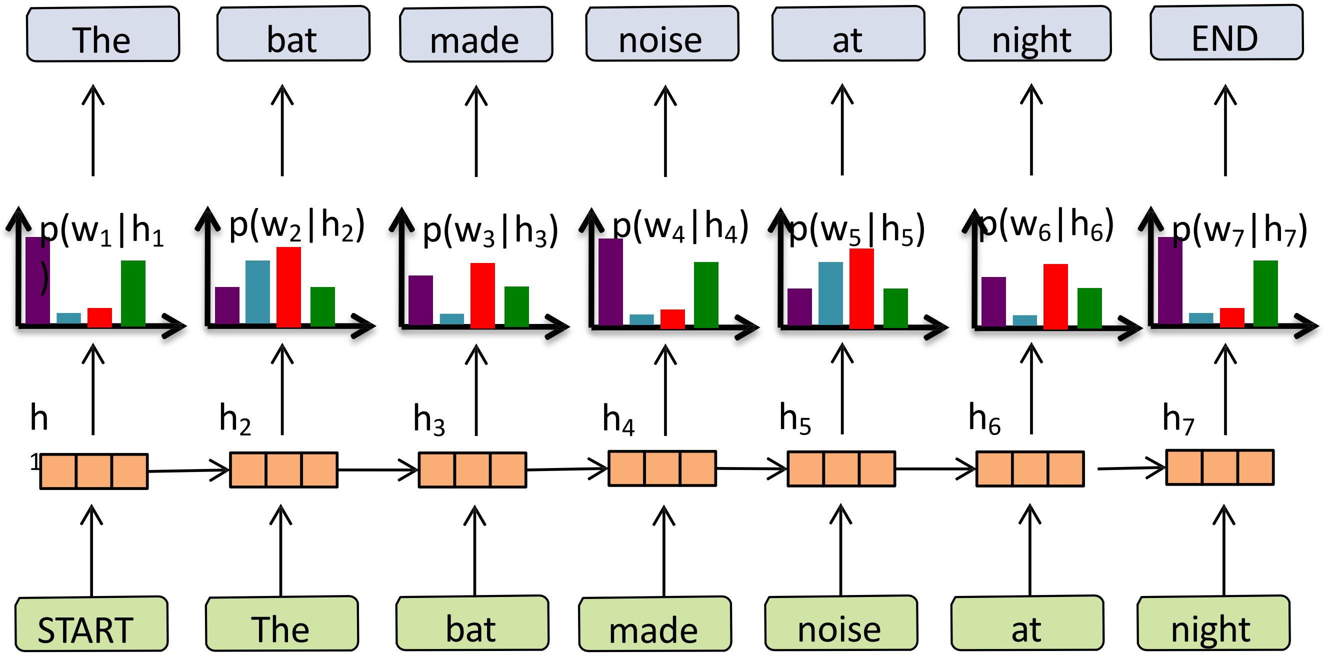
$p(w_6 \mid f_\theta(w_5, w_4, w_3, w_2, w_1))$

Key Idea:

(1)convert all previous words to a **fixed length vector**

(2)define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector

RNN Language Model

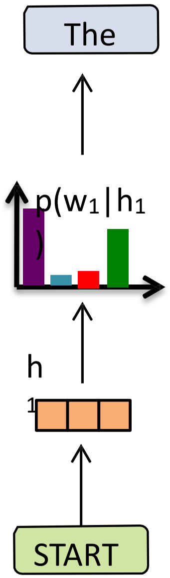


Key Idea:

(1)convert all previous words to a **fixed length vector**

(2)define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

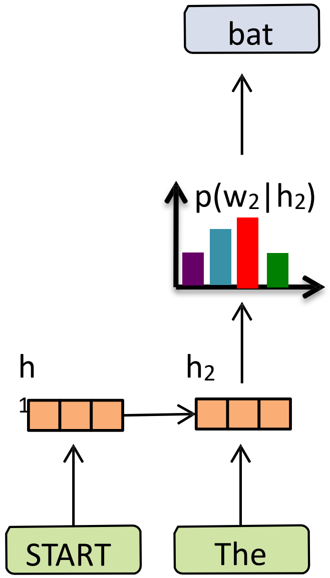


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

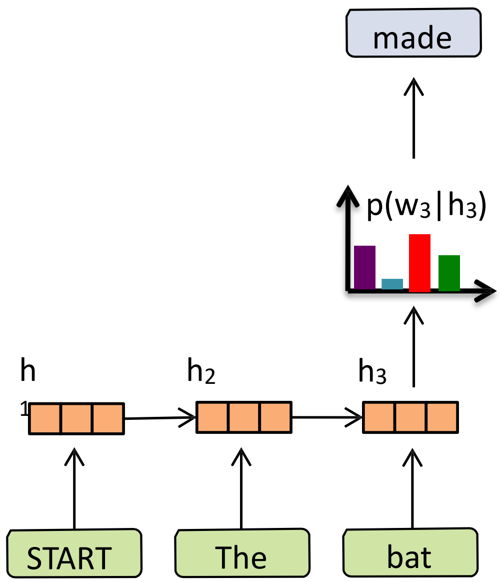


Key Idea:

(1) convert all previous words to a **fixed length vector**

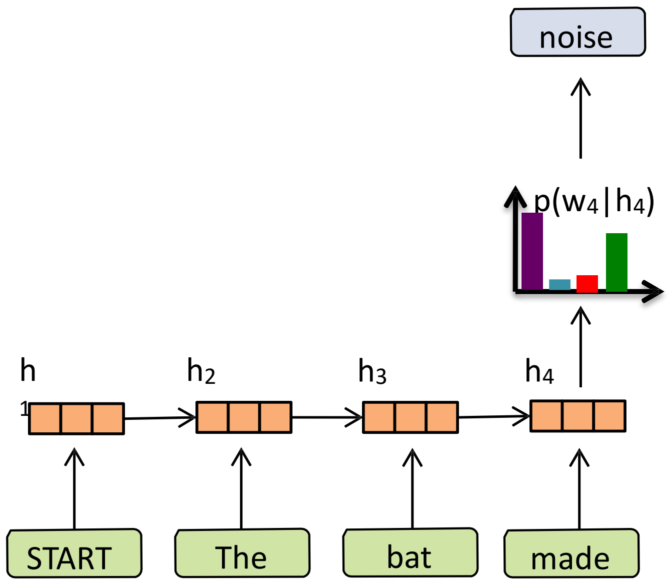
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



__Key Idea__:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

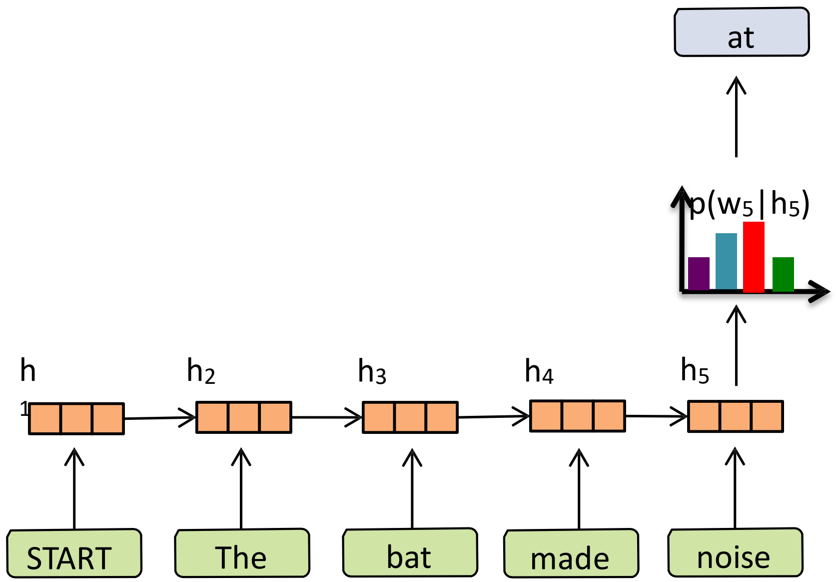


Key Idea:

(1) convert all previous words to a **fixed length vector**

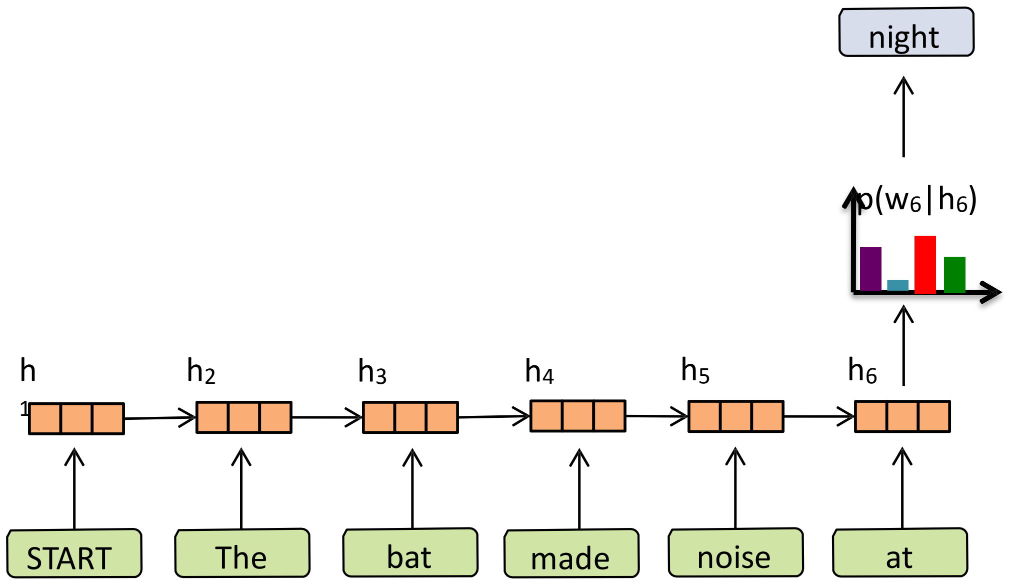
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



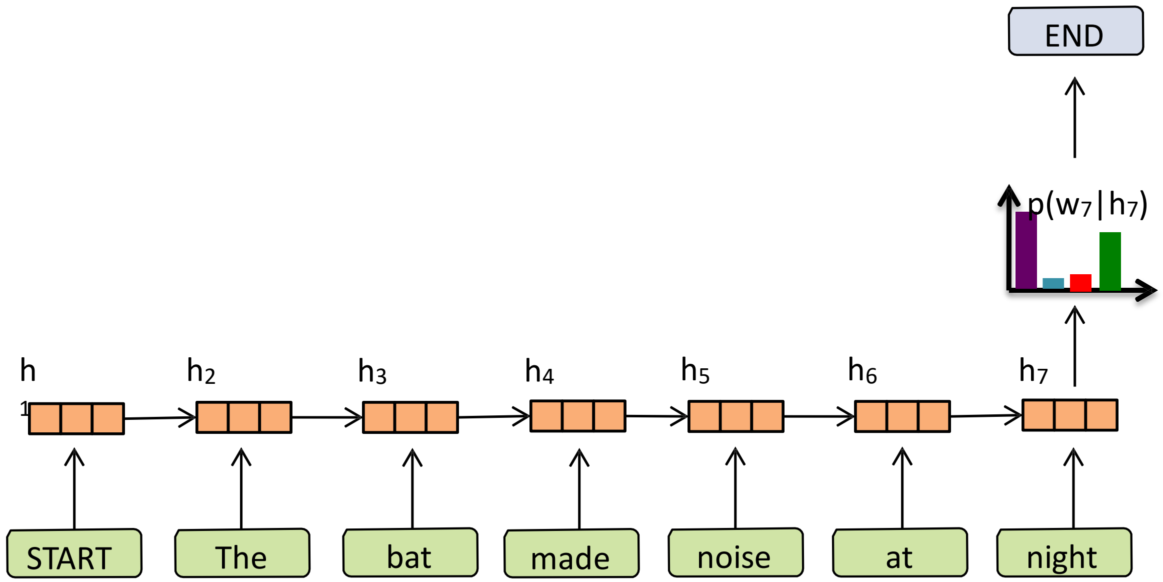
Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model



Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

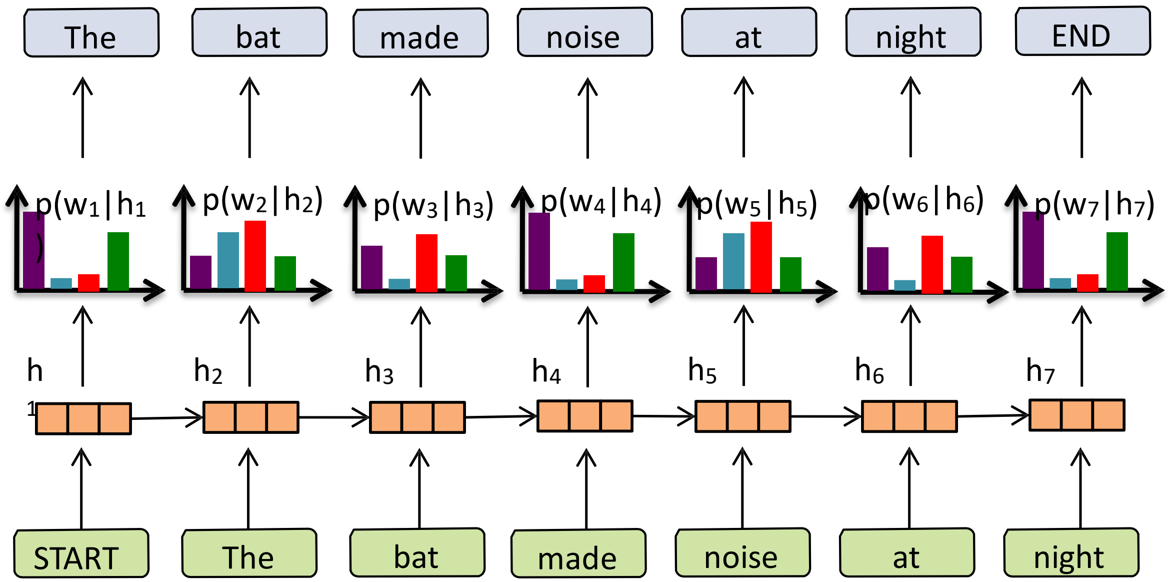


Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

RNN Language Model

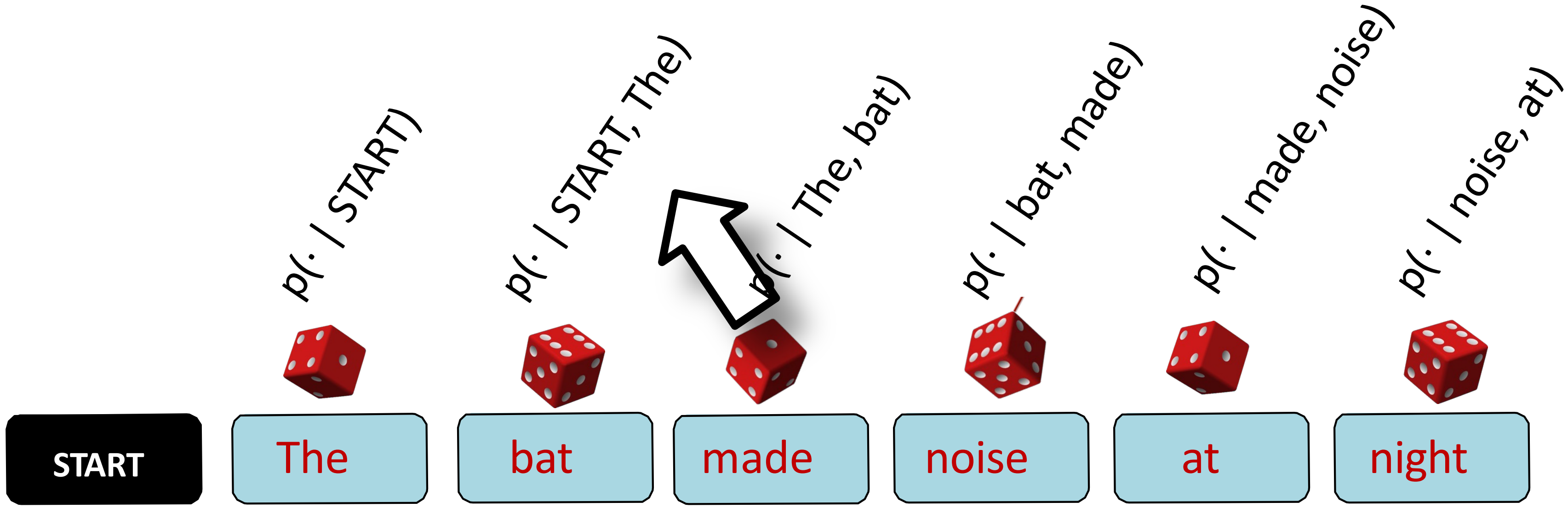


$$p(w_1, w_2, w_3, \ldots, w_T) = p(w_1 \mid h_1)\, p(w_2 \mid h_2) \ldots p(w_2 \mid h_T)$$

Sampling from a Language Model

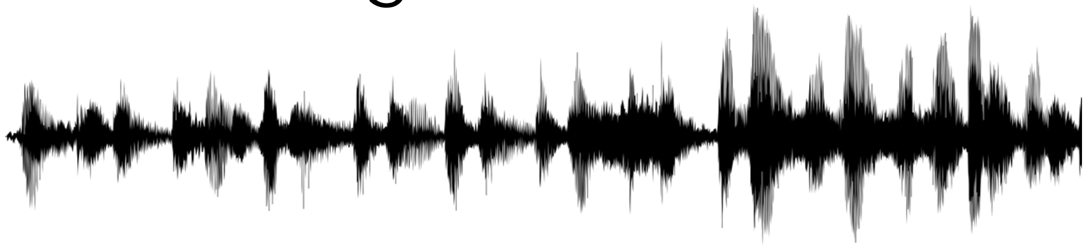
- *__Question__*: How do we sample from a Language Model? *__Answer__*:

1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to $p(w_t \mid w_{t-2}, w_{t-1})$
3. Roll that die and generate whichever word w_t lands face up
4. Repeat



Sequence to Sequence Model

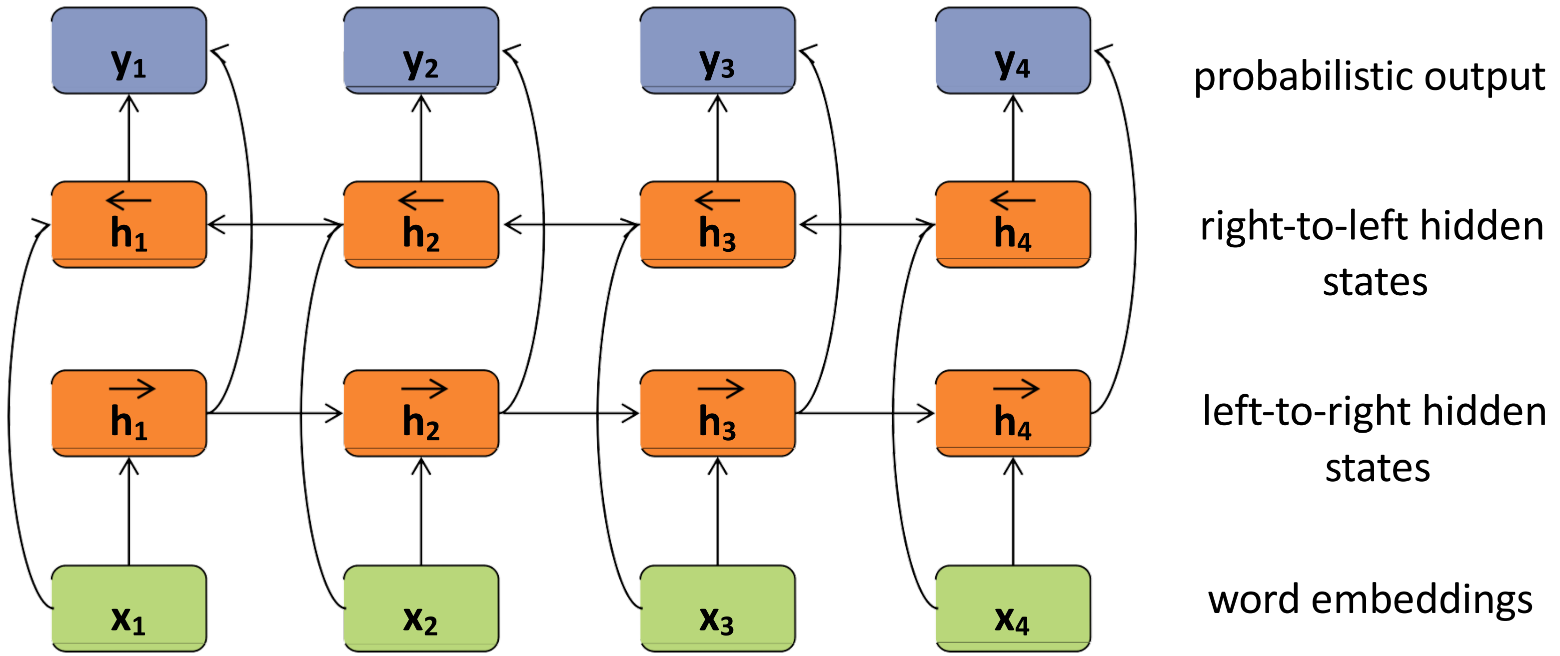
- Speech recognition



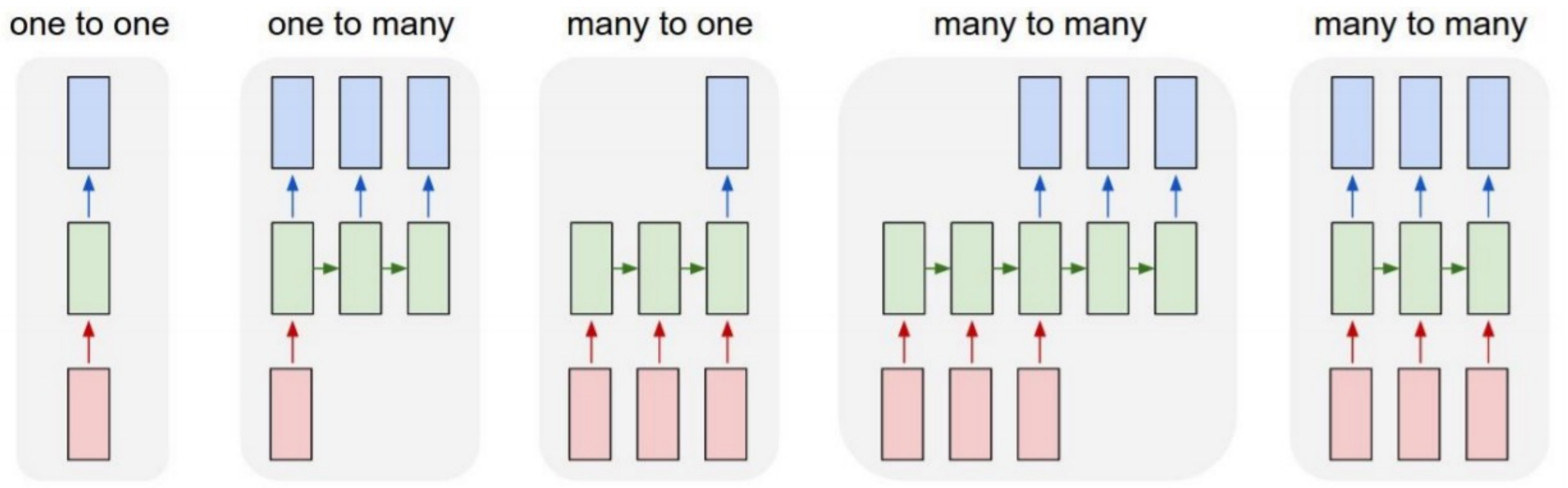
- Machine Translation
- Summarization

Bidirectional RNN

RNNs are a now commonplace backbone in deep learning approaches to natural language processing



Recurrent Neural Networks

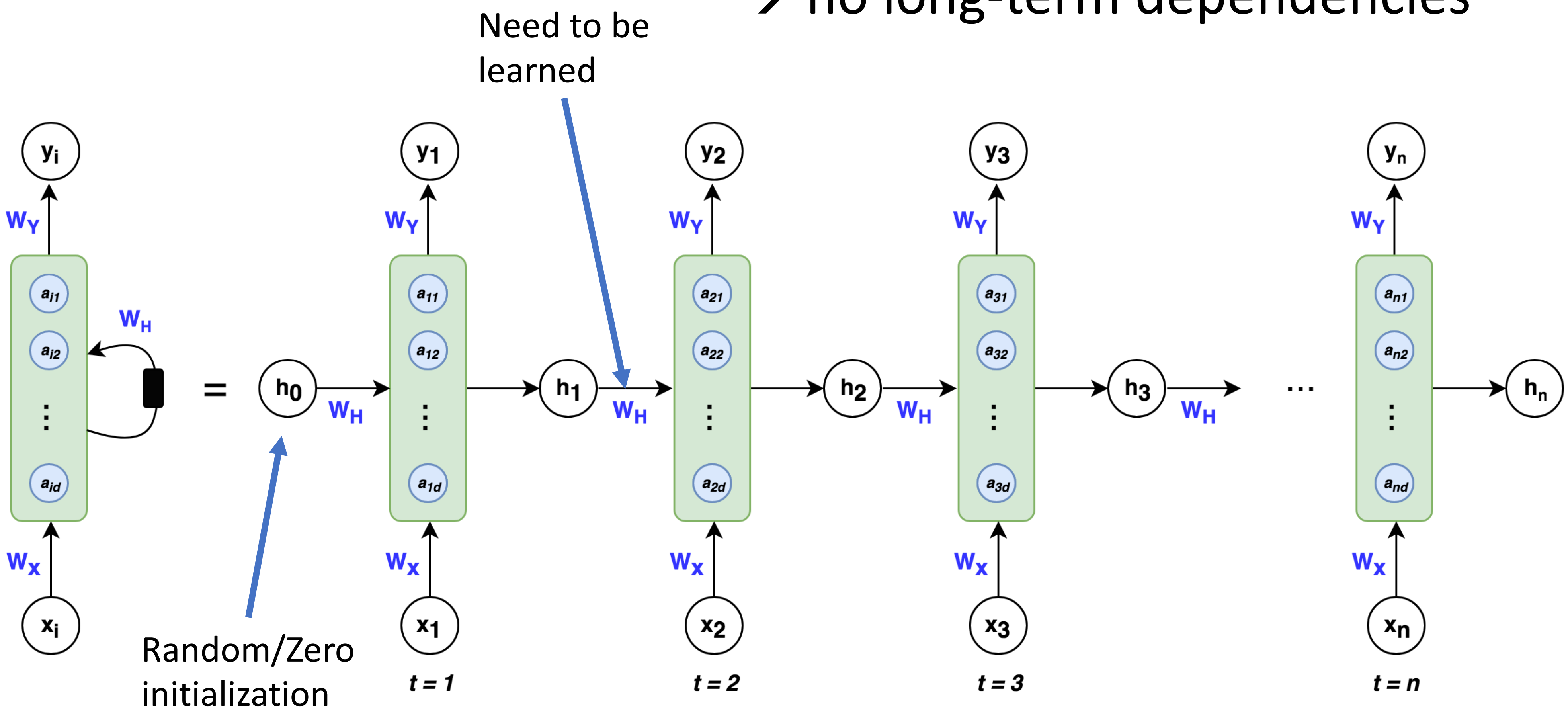


Original RNN Network	Image → Sequence of Words	Sequence of Words → Summary	Machine Translation	Video Classification on frame level

Recurrent Neural Networks

- State is saved in hidden vector h → last step is preserved

  → no long-term dependencies



Sequence to Sequence Model

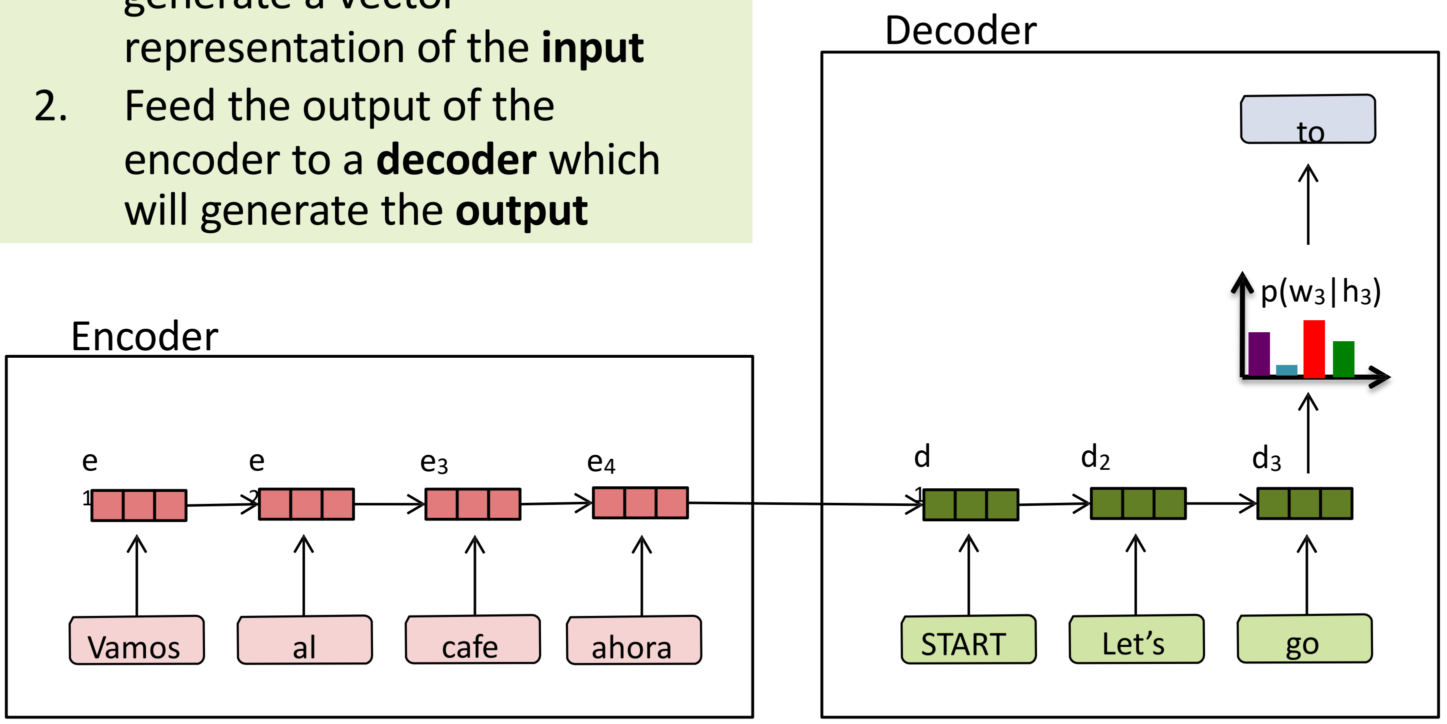
Now suppose you want generate a sequence conditioned on another input

Key Idea:

1. Use an **encoder** model to generate a vector representation of the **input**
2. Feed the output of the encoder to a **decoder** which will generate the **output**

Applications:

- translation:
  Spanish à English
- summarization:
  article à summary
- speech recognition:
  speech signal à transcription

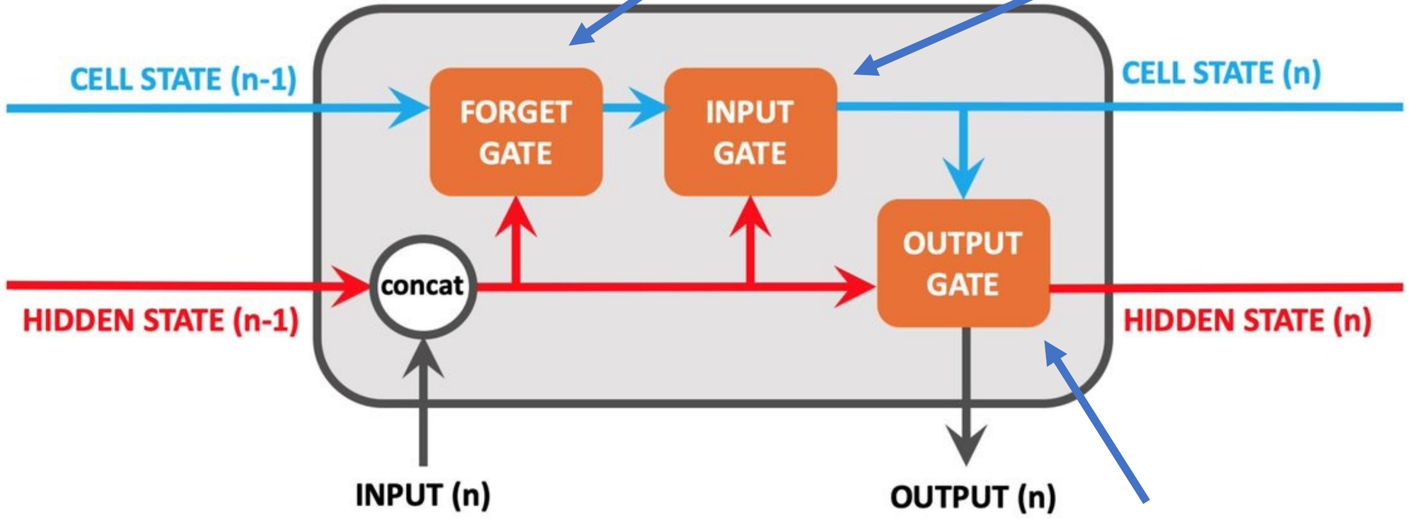


RNN 2.0: Long short-term Memory (LSTM)

- Hidden State: holds previous information (Short-term memory)
- Cell State: memory of the network (Long-term memory)

What should be forget in cell state

How much new data should be taken



How much does memory affect output