Introduction to Machine Learning (CSCI-UA 473): Fall 2021

Lecture 9: Neural Networks — 2

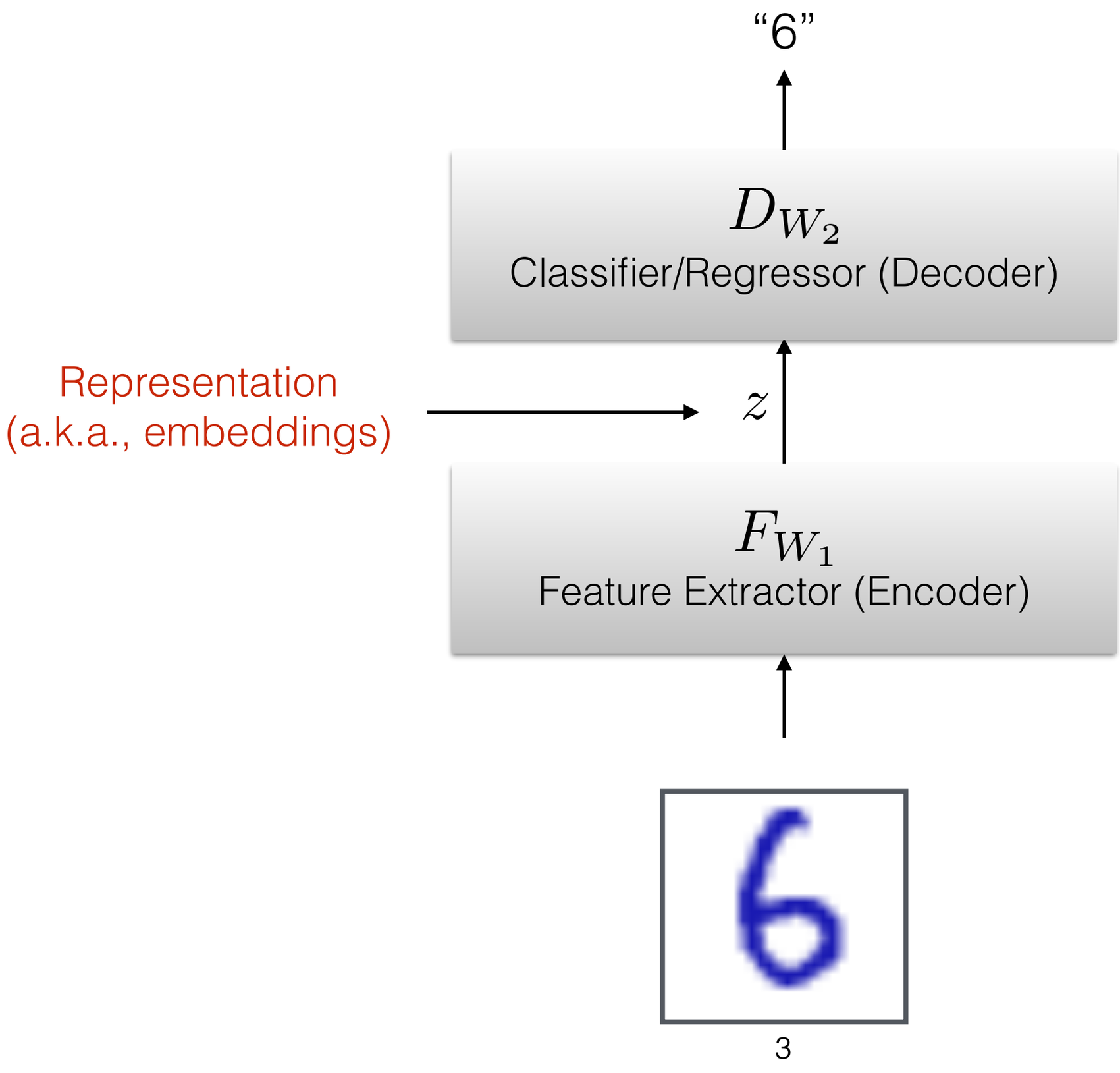
Sumit Chopra
Courant Institute of Mathematical Sciences
Department of Radiology - Grossman School of Medicine
NYU

Lecture Outline

Convolutional Neural Networks (CNNs)
- Motivation behind using CNNs
- The convolution operation (1D/2D/3D)
- The pooling layer
- Backpropagation in CNNs

Recurrent Neural Networks (RNNs)
- Motivation behind using RNNs
- Elman networks
- Back Propagation Through Time algorithm (BPTT)

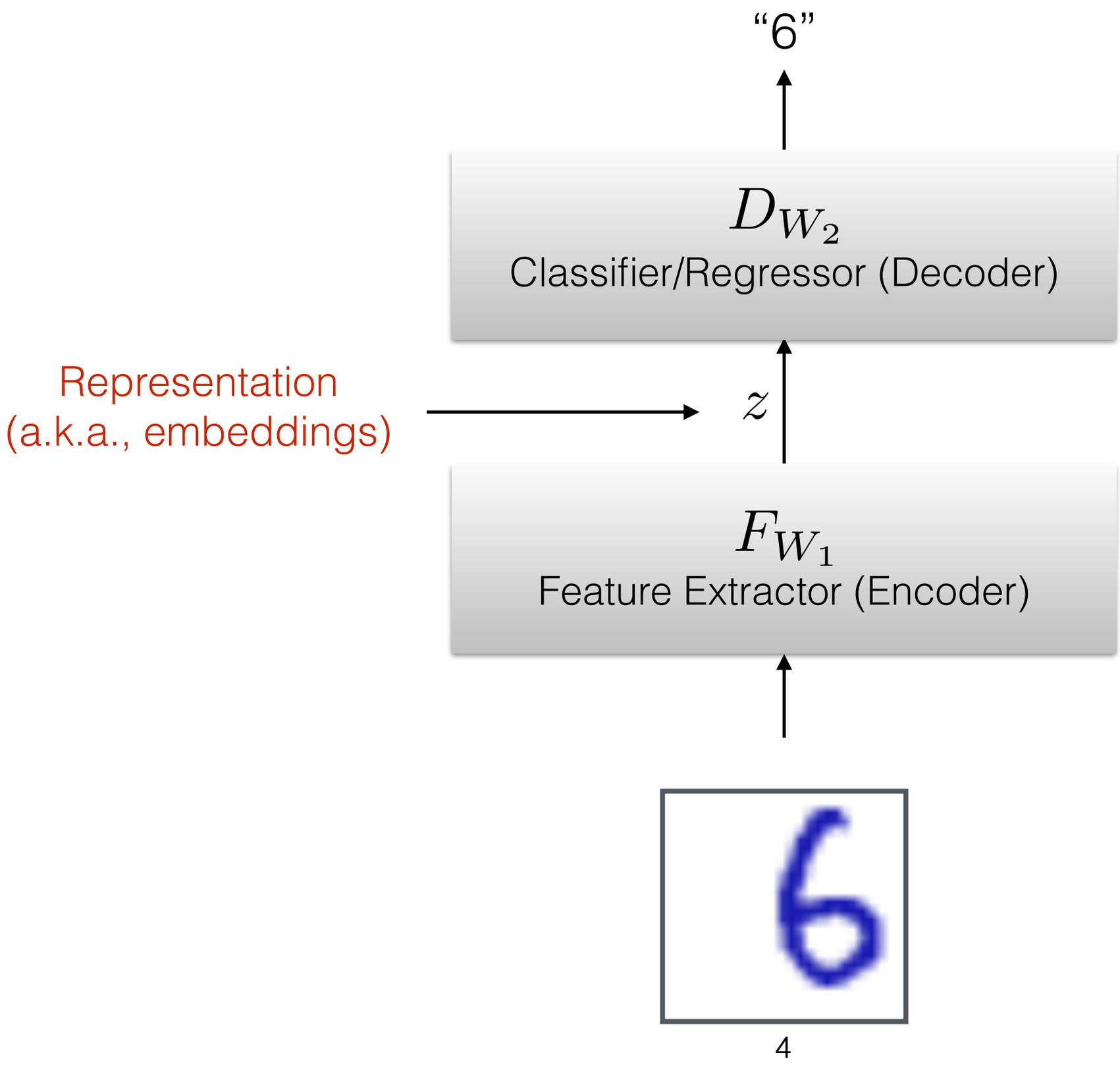
Motivation Behing Convolutional Neural Networks (CNNs)



E.g., typically the last layer

E.g., all the layers except the last

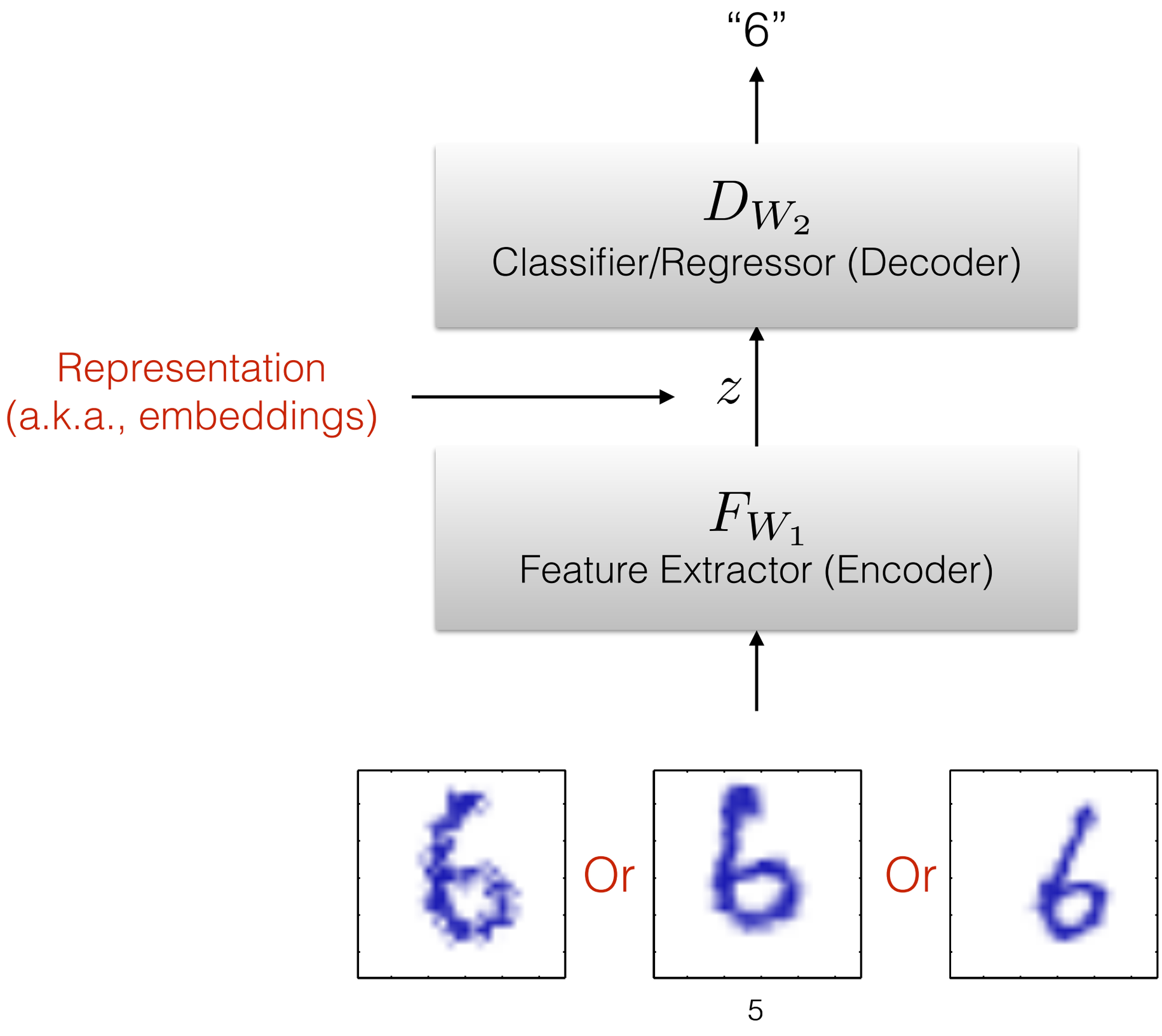
Motivation Behing Convolutional Neural Networks (CNNs)



E.g., typically the last layer

E.g., all the layers except the last

Motivation Behing Convolutional Neural Networks (CNNs)



E.g., typically the last layer

E.g., all the layers except the last

Motivation Behing Convolutional Neural Networks (CNNs)

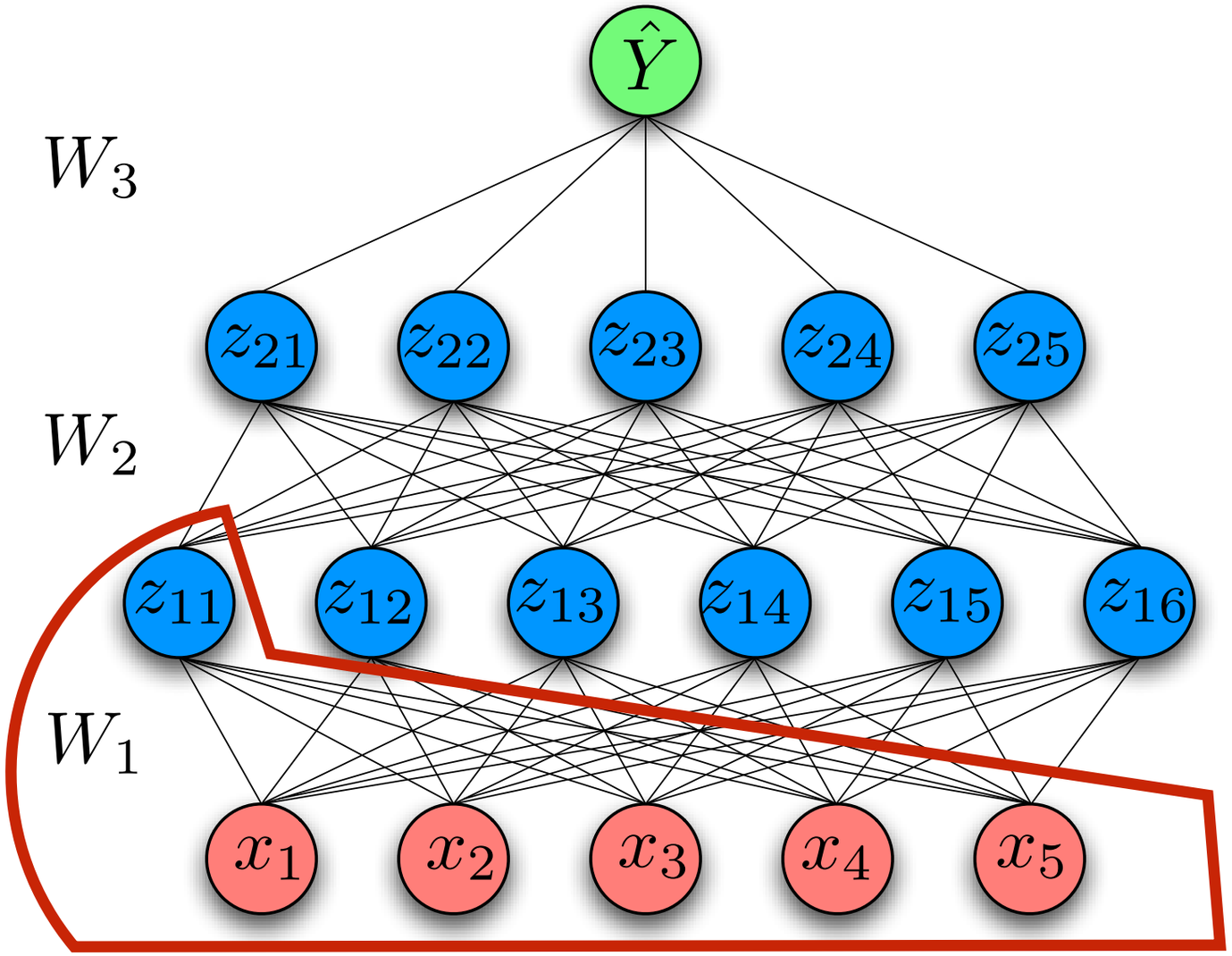
Network output should be invariant to translation, rotation, other arbitrary distortions/ artifacts

One possible solution using fully connected neural network is to have sufficiently large and diverse training set so that such invariances are represented — but this is hard to achieve for every possible invariance

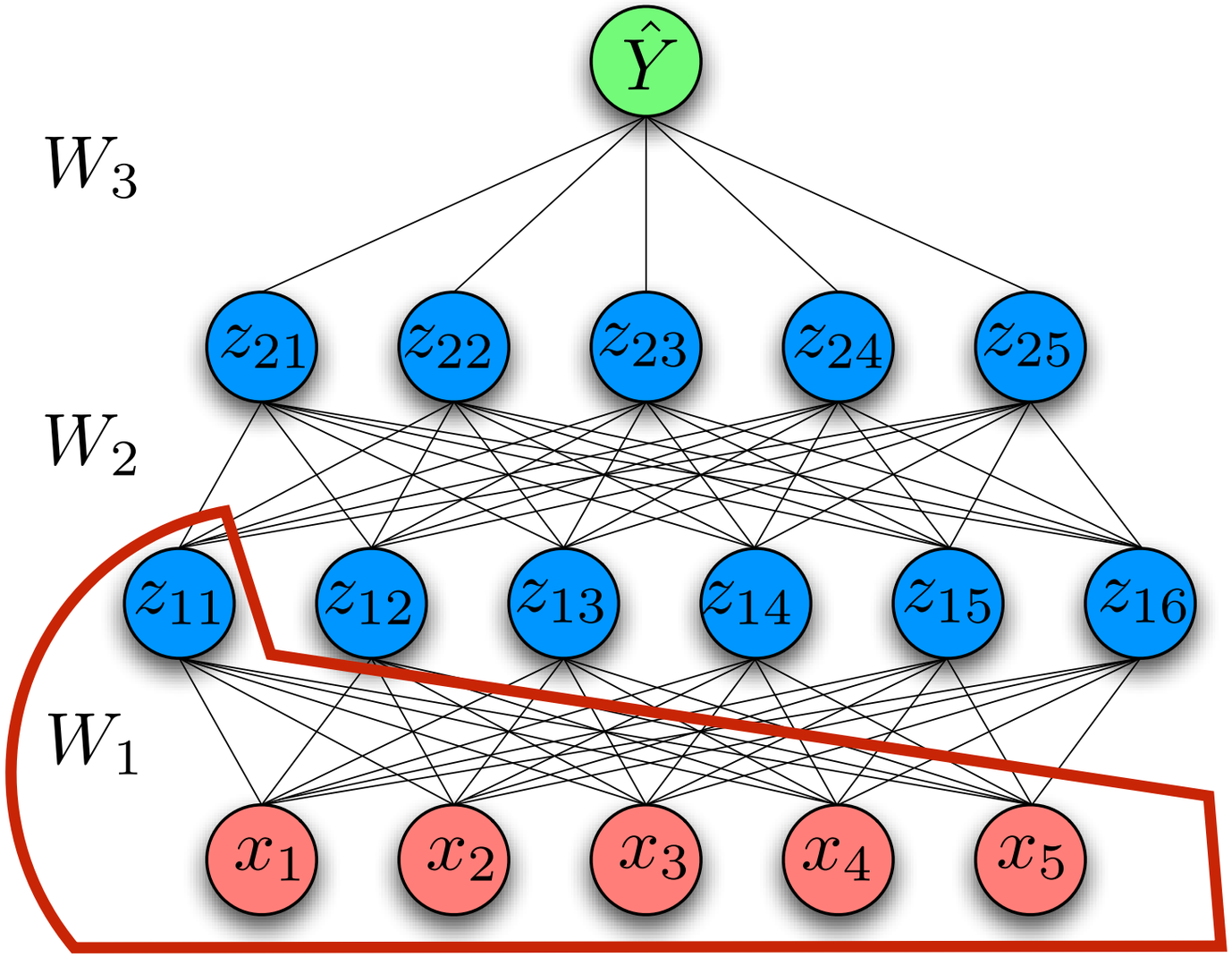
This approach ignores the key property of images, which is locality: nearby pixels tend to have similar values to each other

Fully connected networks are brittle

Motivation Behing Convolutional Neural Networks (CNNs)



Motivation Behing Convolutional Neural Networks (CNNs)



General Limitations of Fully Connected Networks

Need a large training set in order to capture trivial invariances, such as, translation, rotation, scaling, elastic deformation etc

It ignores the key properties of the images, which is locality: nearby pixels with an image are strongly correlated

Each unit of each layer sees the whole image. This leads to explosion of the number of parameters. To handle a 100x100 image, each unit will require 10K weights

Information can be merged at later stages to get higher order features only about the whole image

All of this makes them hard to train

Convolutional Neural Networks (CNNS)

Local receptive fields

Feature pooling

Weight sharing

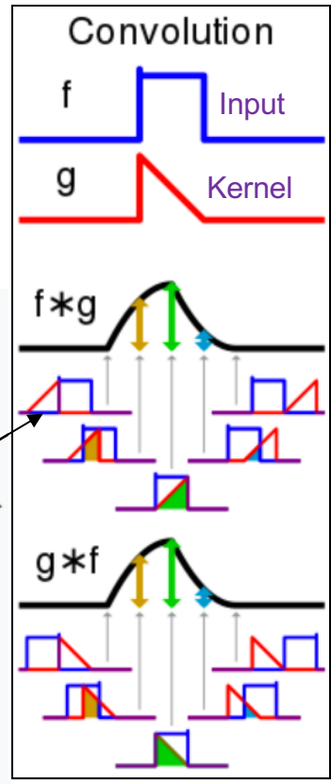
The Convolution Operation (1D): Continuous

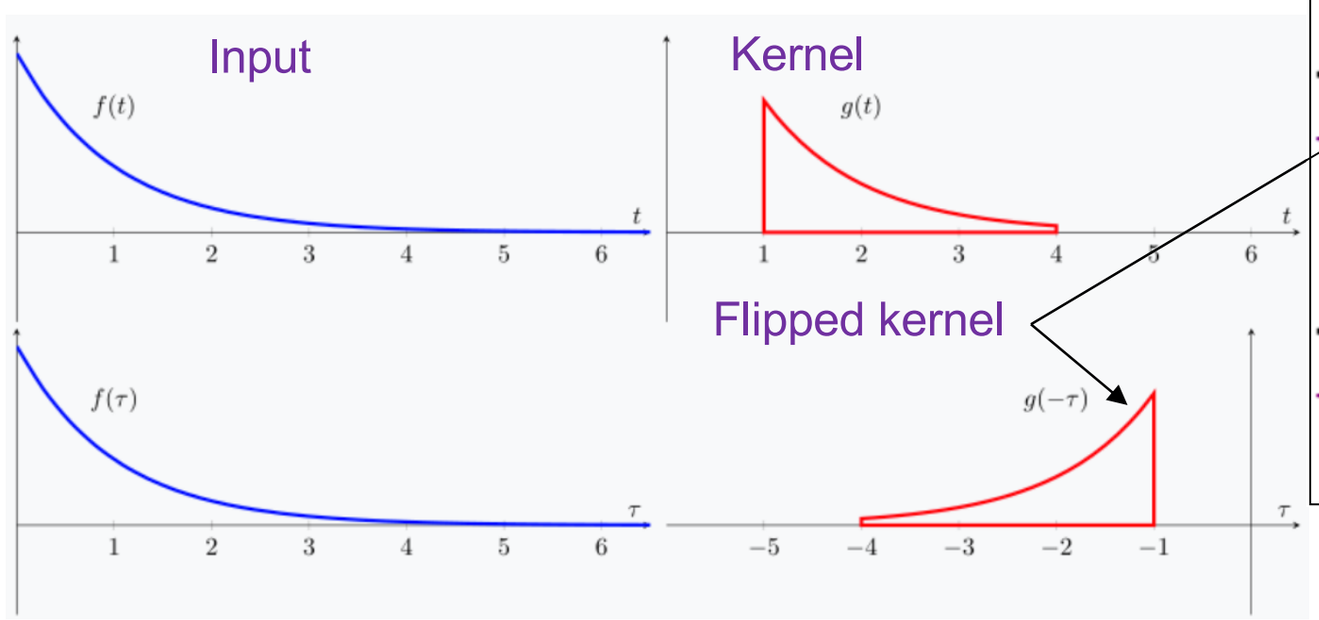
$f(t)$: input function whose convolution is sought

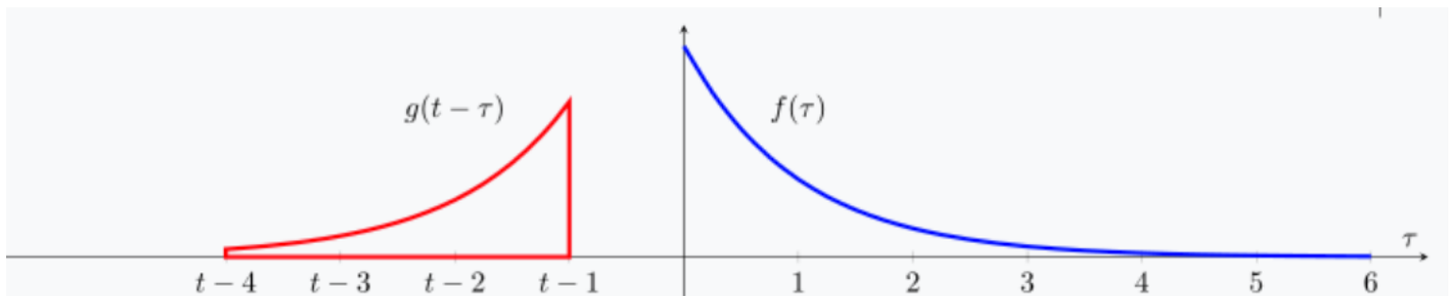
$g(t)$: kernel function which is used to do the convolution

The convolution of the function f by the kernel g, denoted by $f * g$ is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$







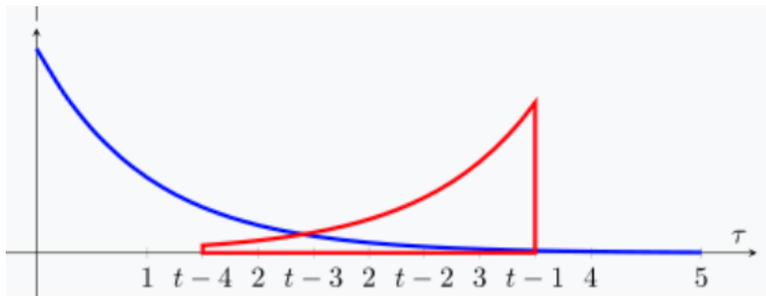
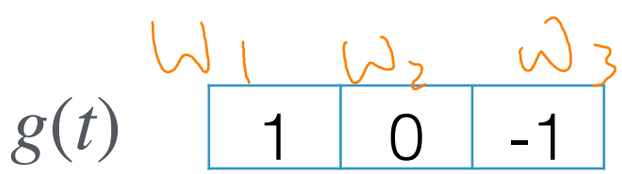


Figure from https://cedar.buffalo.edu/~srihari/CSE676/

The Convolution Operation (1D): Discrete

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau)$$



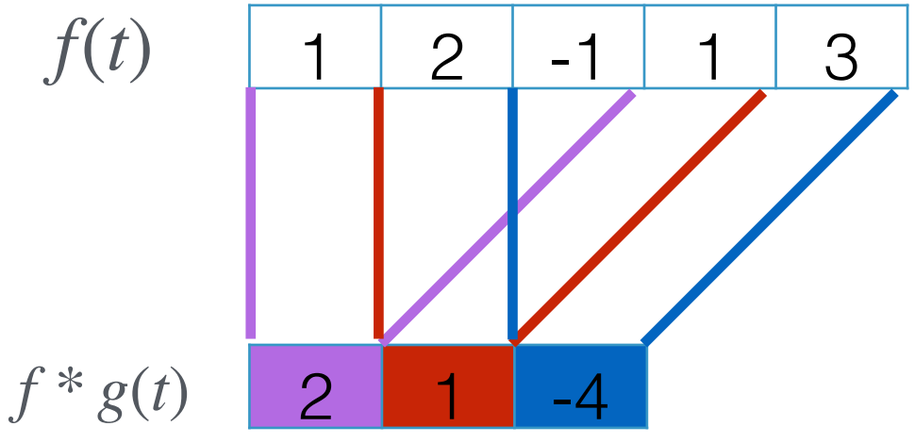


Figure from https://cedar.buffalo.edu/~srihari/CSE676/

The Convolution Operation (1D): Discrete

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau)$$

(Hyper)Parameters of convolution operation

Convolution kernel size (K)

Convolution stride (S)

Input padding (P)

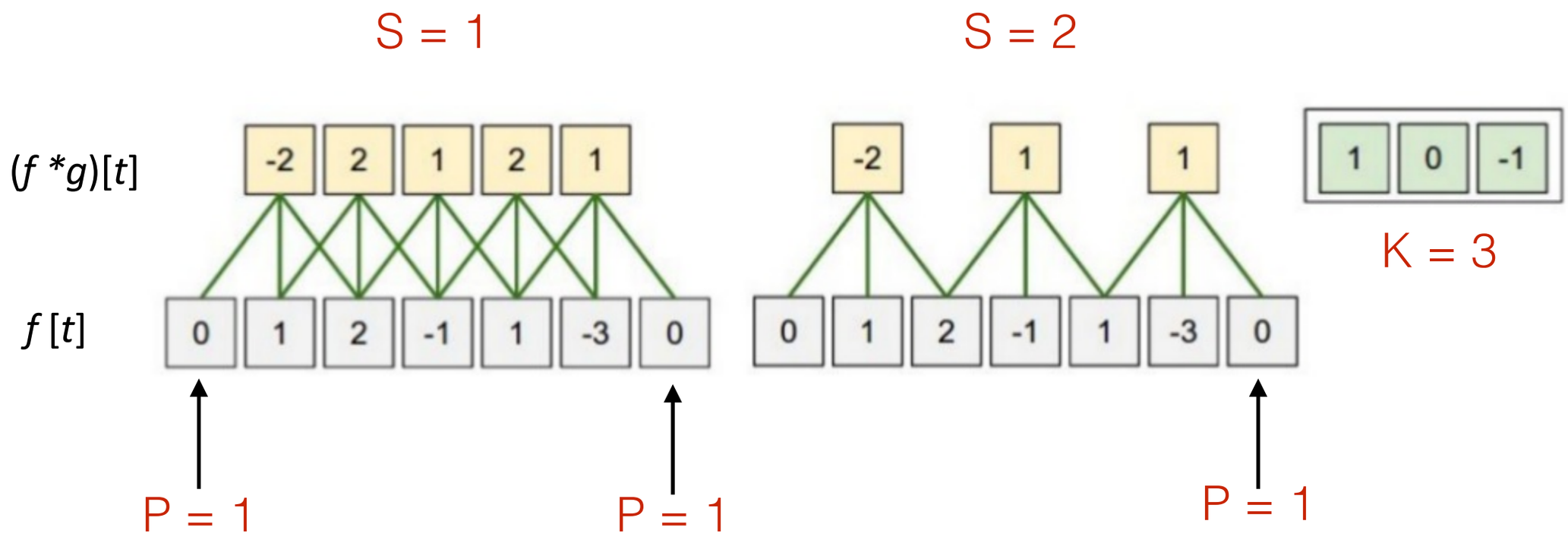


Figure from https://cedar.buffalo.edu/~srihari/CSE676/

The Convolution Operation (2D): Discrete

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau)$$

S = 1 and P = 0

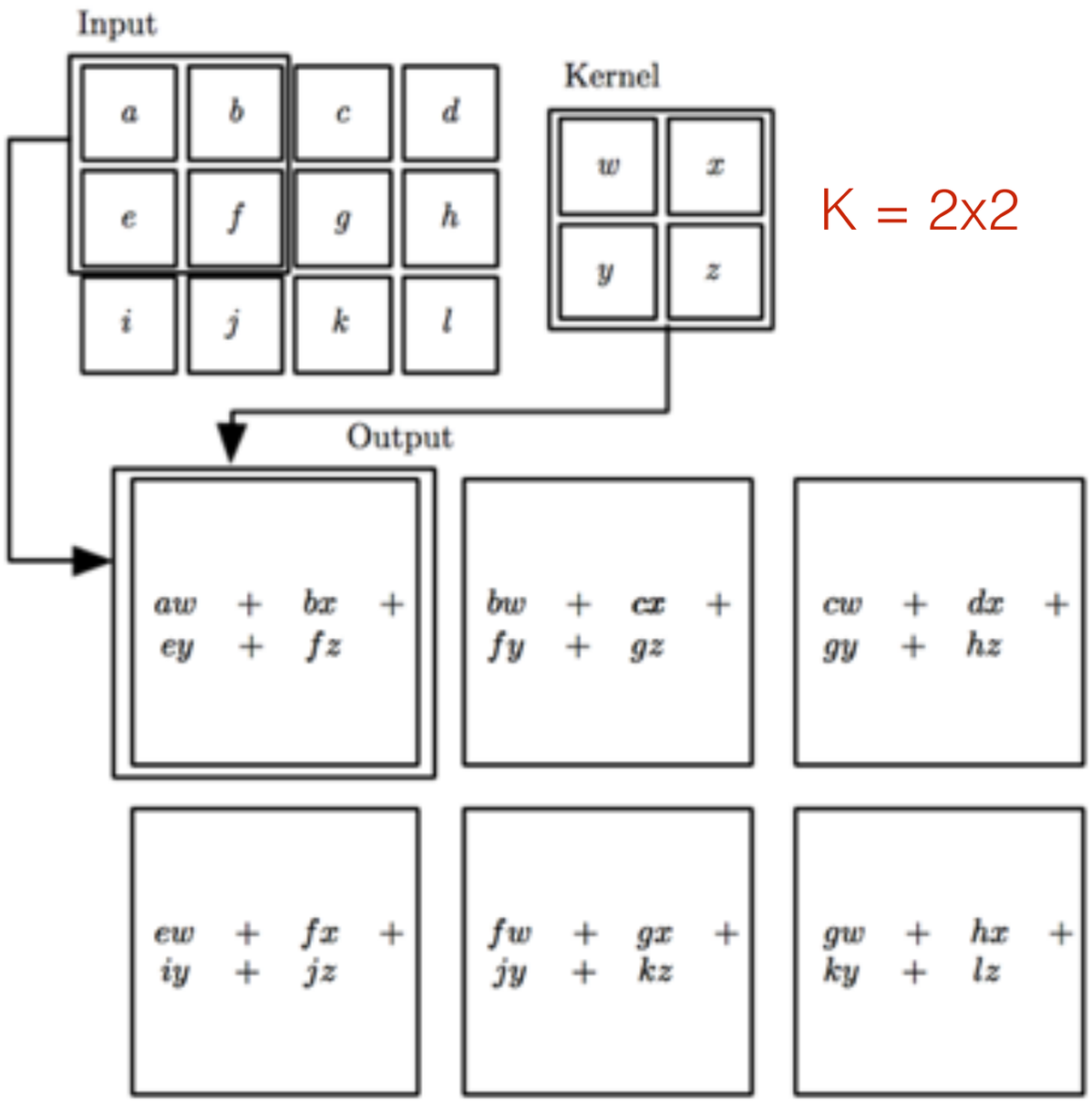


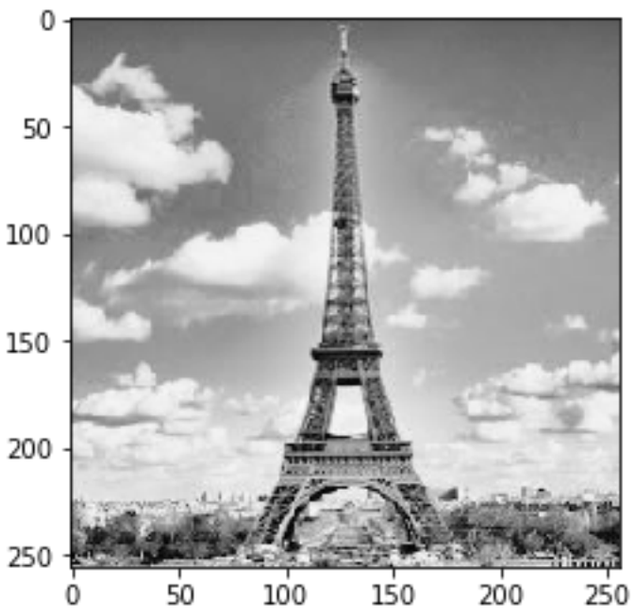
Figure from https://cedar.buffalo.edu/~srihari/CSE676/

What Do These Kernels Mean?

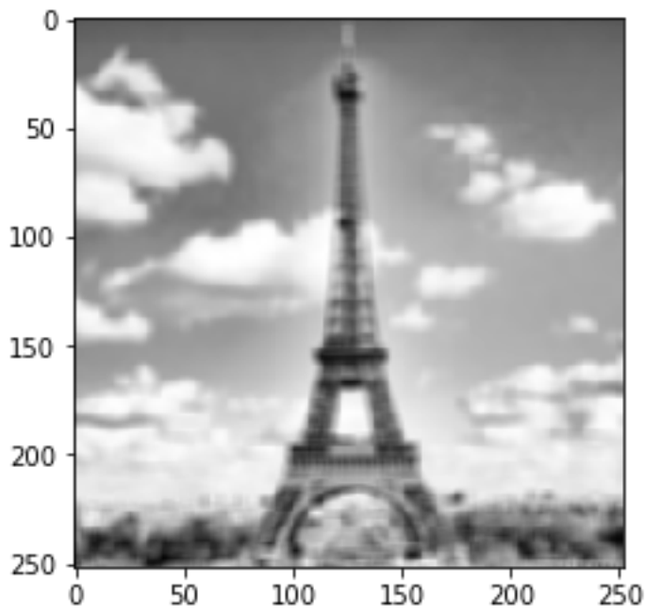
$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau)g(t - \tau)$$

What Do These Kernels Mean?

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau)$$



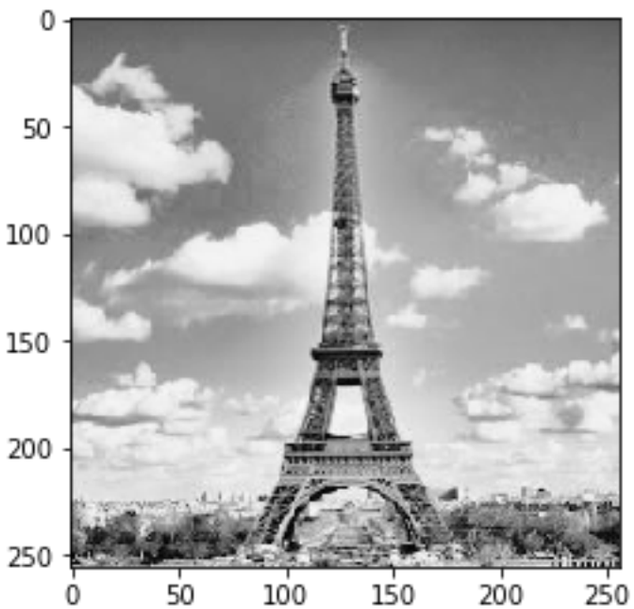
$$* \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$



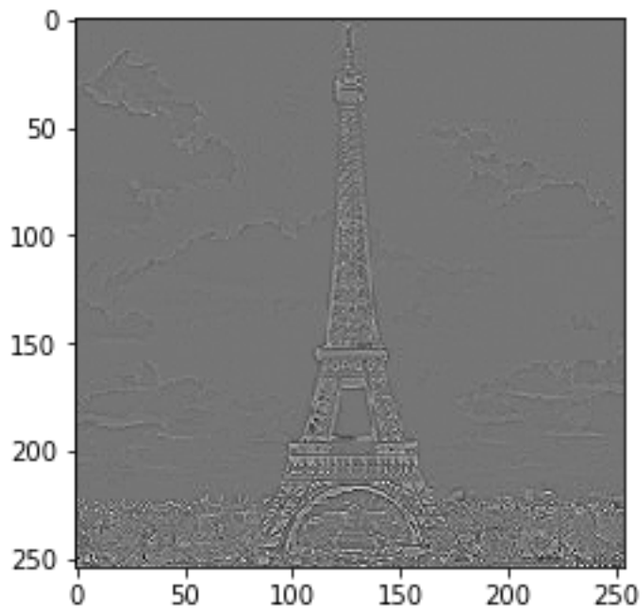
Blurring

What Do These Kernels Mean?

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau)g(t-\tau)$$



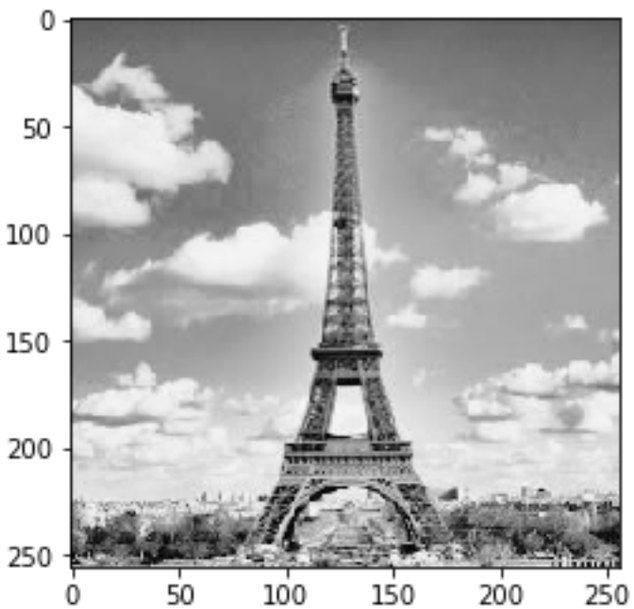
$$* \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$



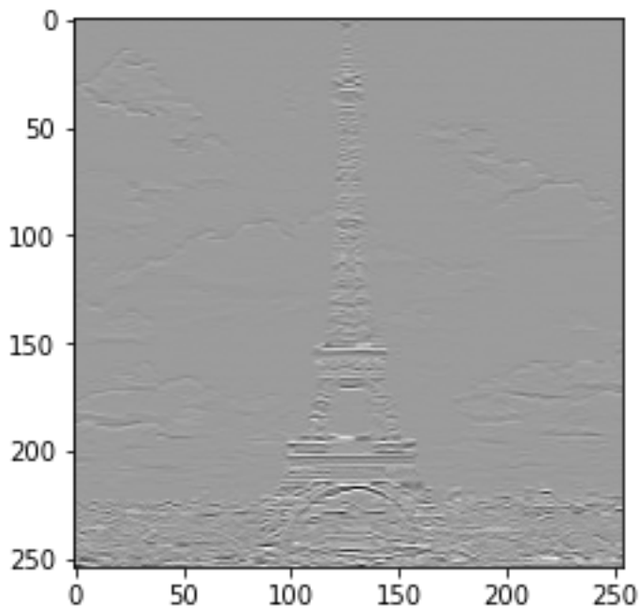
Edge detection

What Do These Kernels Mean?

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau)g(t-\tau)$$



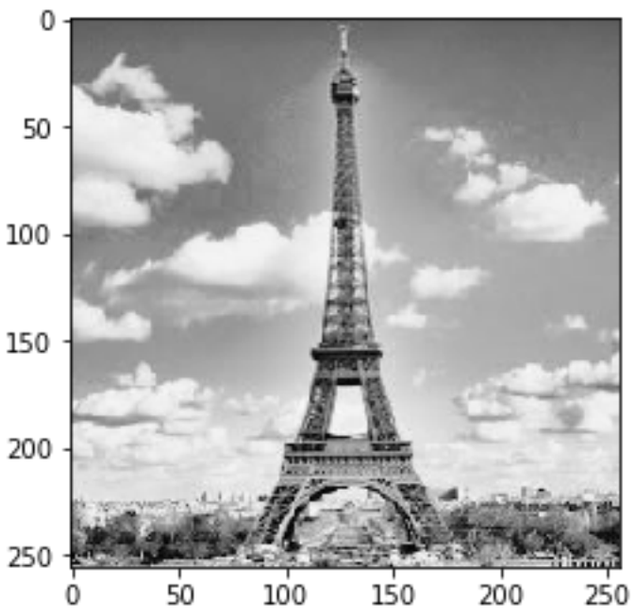
$$* \begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix}$$



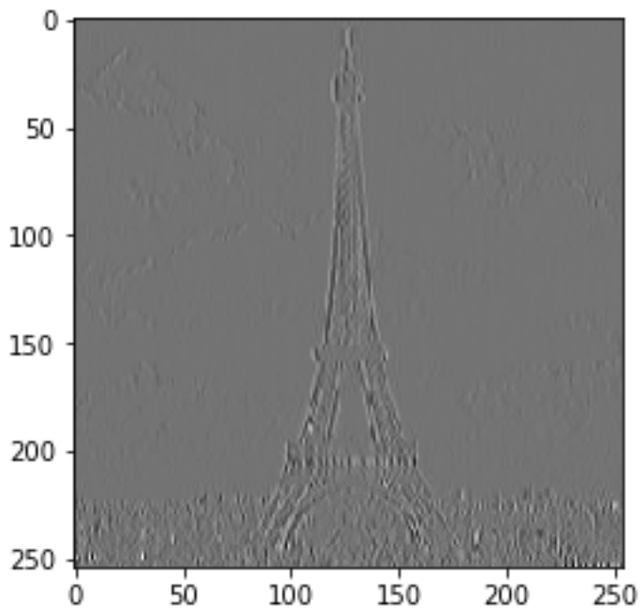
Horizontal line detection

What Do These Kernels Mean?

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau)$$



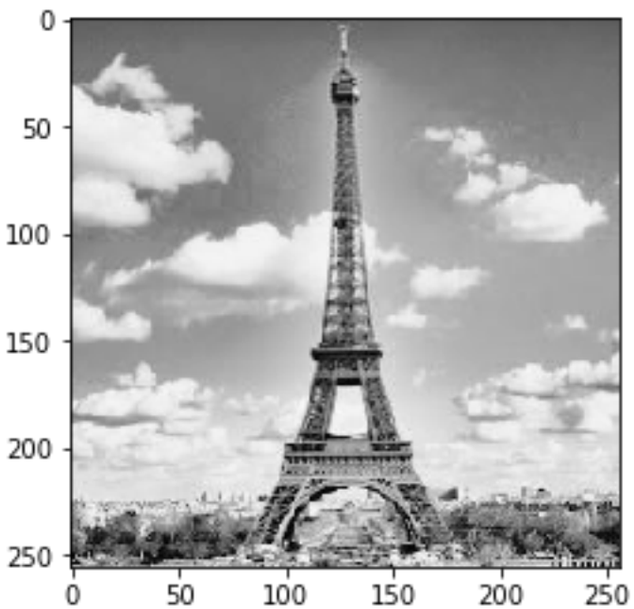
$$* \begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}$$



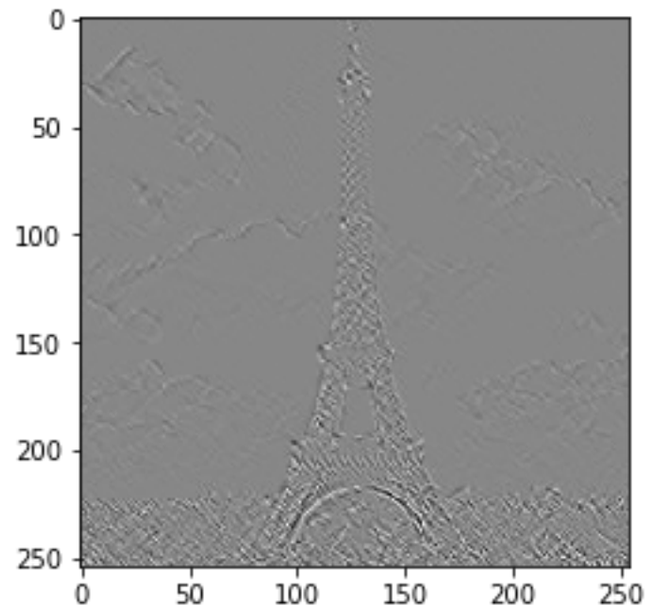
Vertical line detection

What Do These Kernels Mean?

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau)g(t-\tau)$$



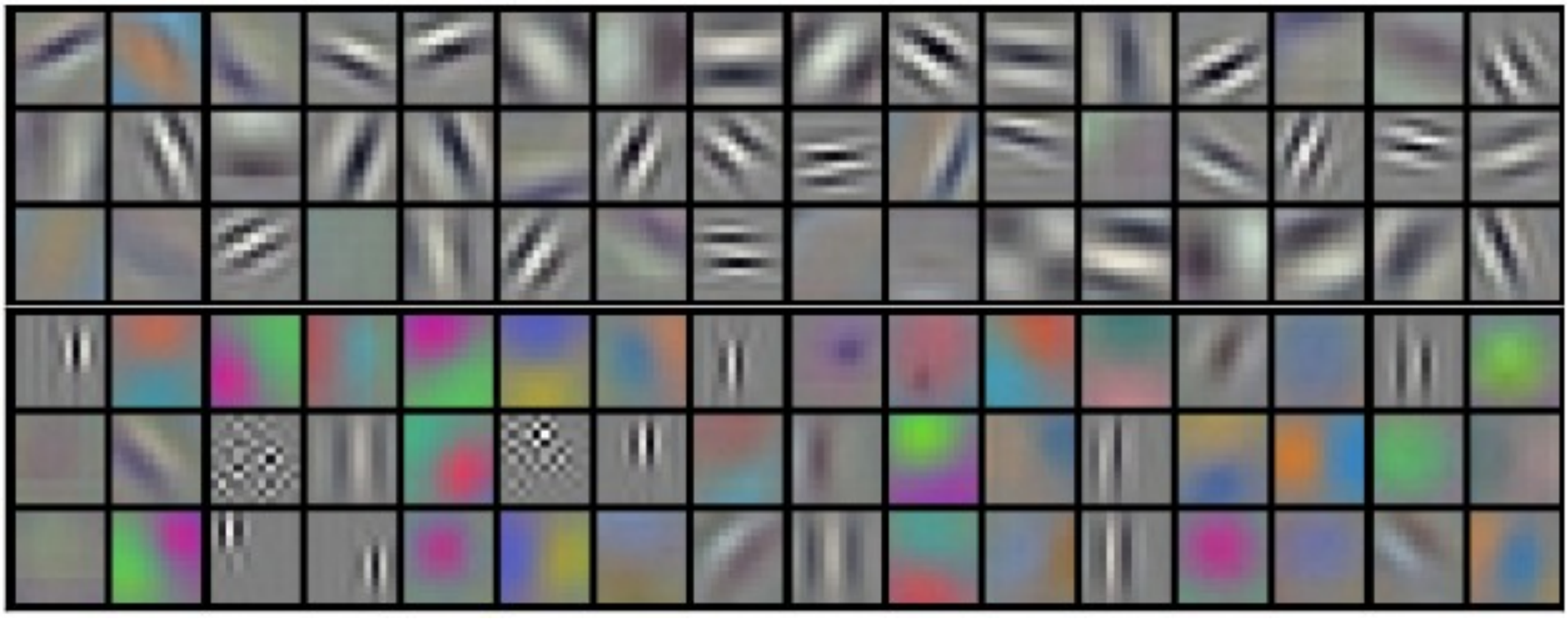
$$* \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$



135^o line detection

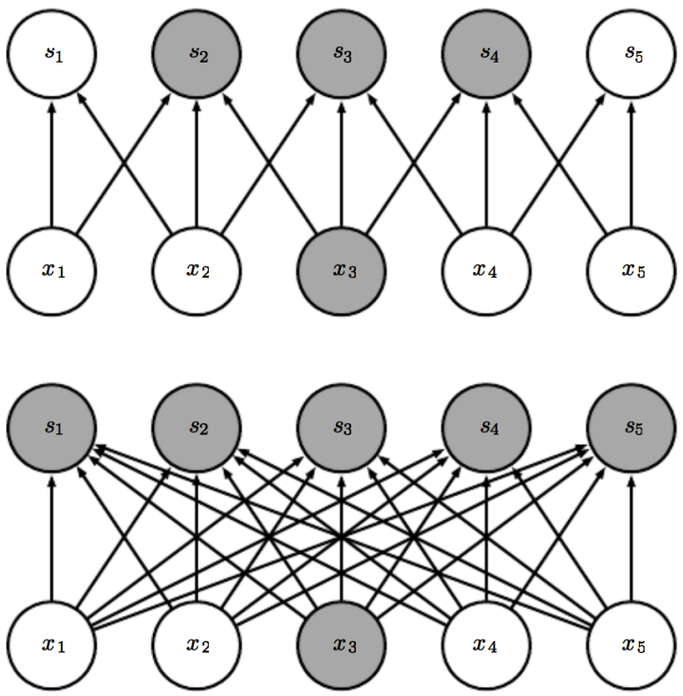
What Do These Kernels Mean?

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau)$$



Sparsity in Convolution Operation

Sparsity from below



Each input x_i feeds all the outputs $s_j's$ in a fully connected operation

Each input x_i only feeds a subset of outputs $s_j's$ in a convolutional operation

Sparsity from top

Each output s_i is influenced by all inputs $x_j's$ in a fully connected operation

Each output s_i is influenced by only a subset of inputs $x_j's$ in convolutional operation

Convolution Layer and Weight Sharing

The trainable parameters are the parameters of the kernels

Each feature map is generated using the same kernel

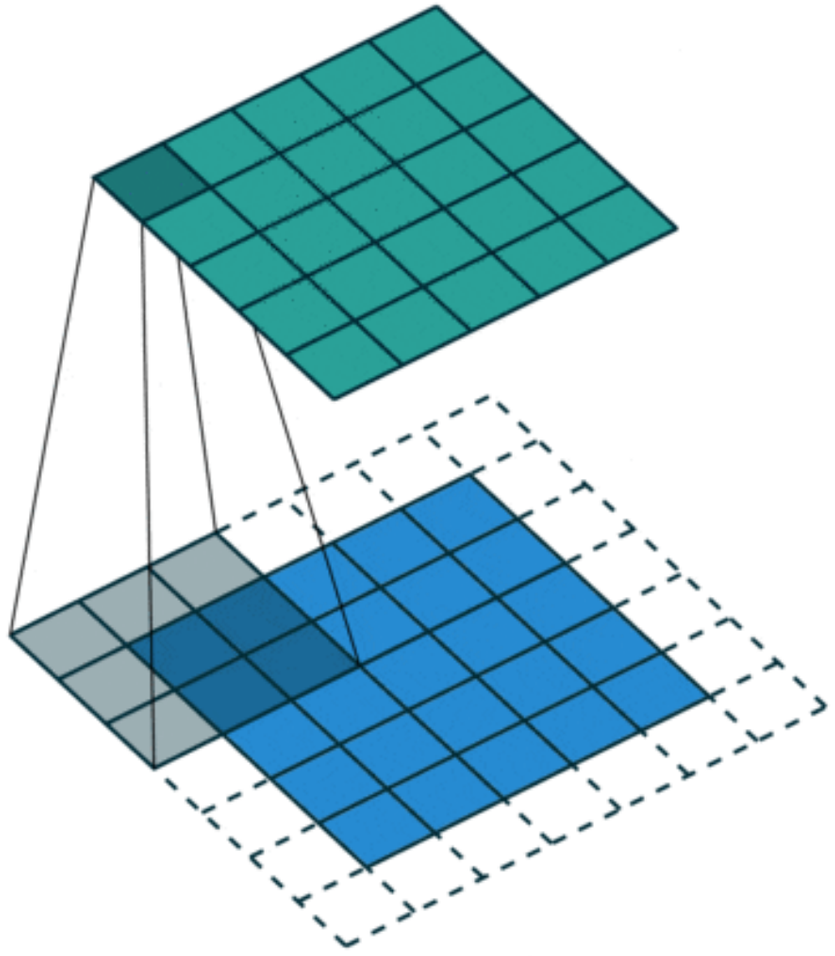
Weights are shared across input

Typically you have multiple feature maps

Number of feature maps dictate the number of weights in a convolutional layer

Feature Maps

Inputs



Pooling/Subsampling Layer

A key aspect of CNNs

Typically applied after the Convolutional layer

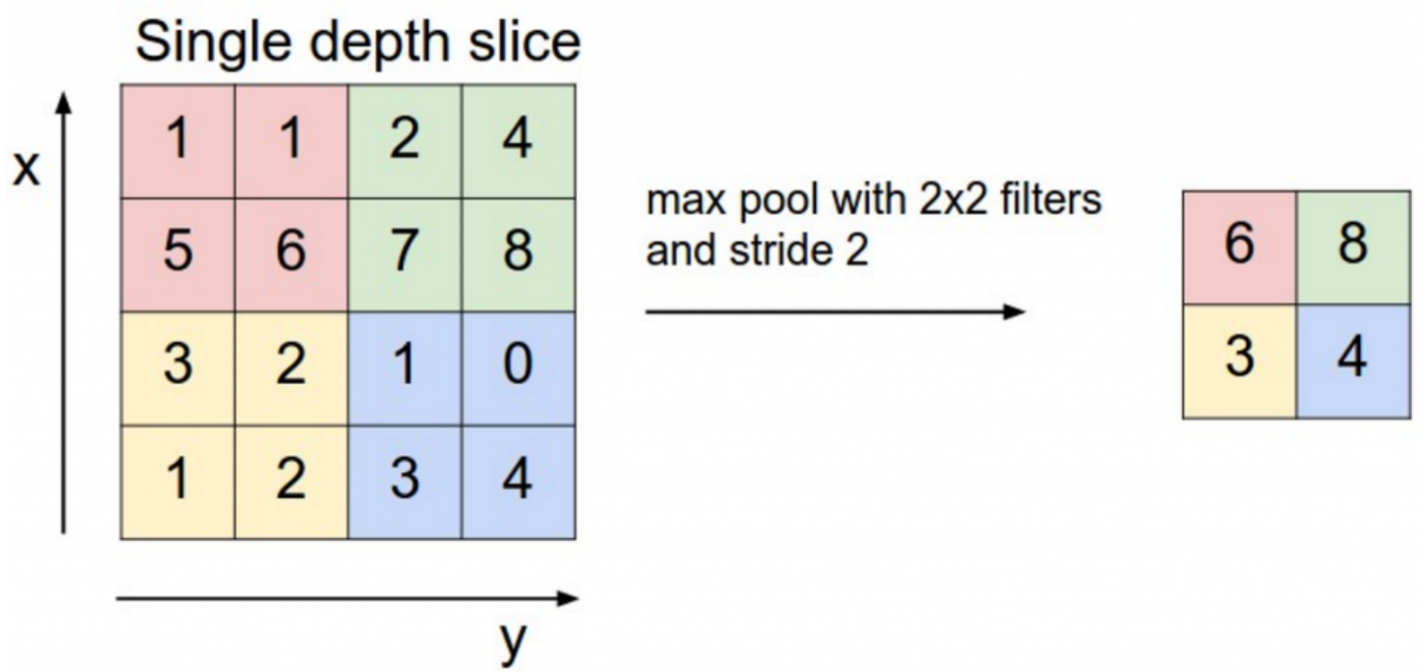
A pooling function replaces the output of a network at a certain location with the summary statistics of the nearby inputs

Pooling layer is responsible for making the network invariant to translation and rotation

It also allows us to convert a variable sized input into a fixed size output

Types of Pooling

Max Pooling



Average pooling

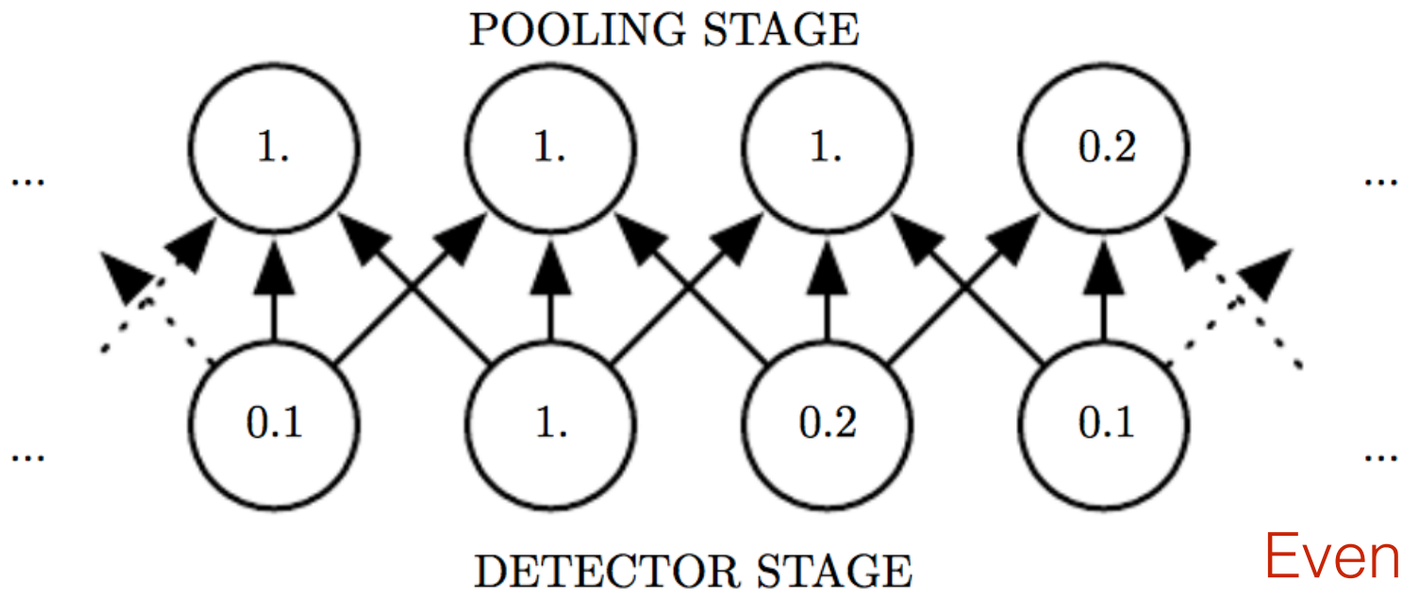
L^2 norm of the rectangular neighborhood

Weighted average based on the distance from the center pixel

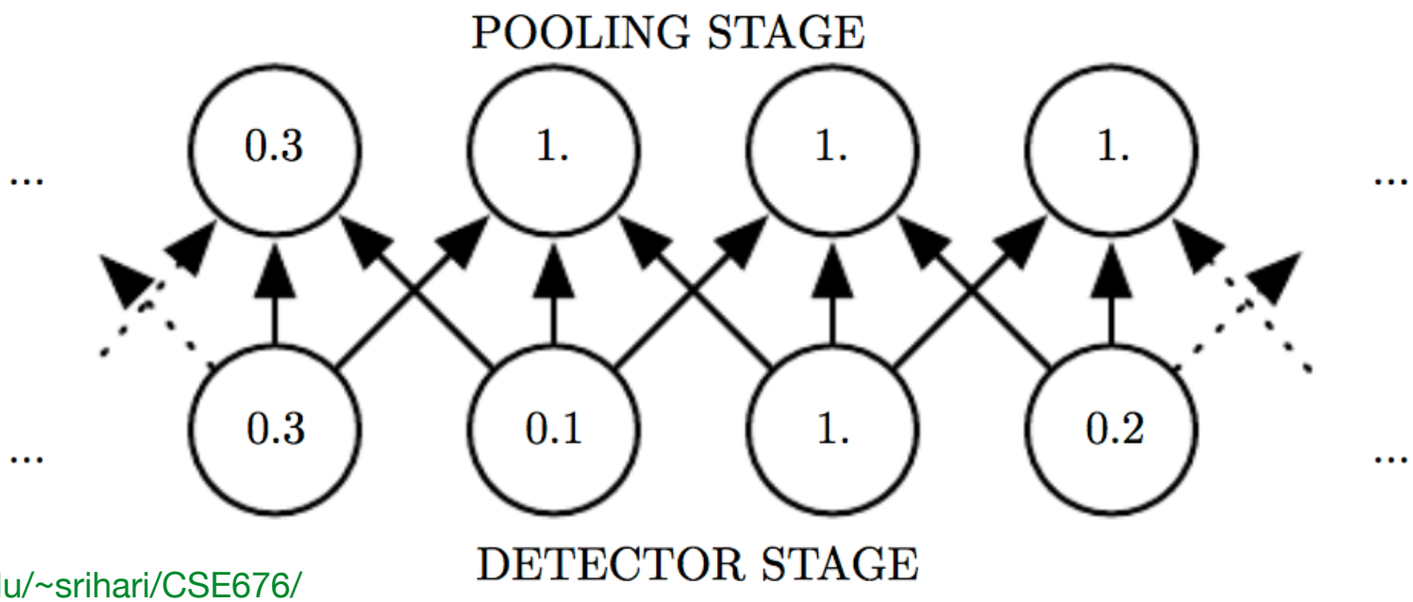
Figure from https://cedar.buffalo.edu/~srihari/CSE676/

Invariance using Pooling

Original Input and Max Pooling



Shifted Input and Max Pooling



Even though all inputs are changed the outputs remain fairly robust with only two changes

Figure from https://cedar.buffalo.edu/~srihari/CSE676/

Backpropagation in Convolution Layer

Input: $x = [x_1, x_2, x_3, x_4]$

Weights: $w = [w_1, w_2], b$

Output: $y = [y_1, y_2, y_3]$

Then the forward convolution operation says

$$y_1 = x_1 w_1 + x_2 w_2 + b$$

$$y_2 = x_2 w_1 + x_3 w_2 + b$$

$$y_3 = x_3 w_1 + x_4 w_2 + b$$

Now during the back propagation phase, assume that we know

$$\delta_y = [\delta_{y_1}, \delta_{y_2}, \delta_{y_3}] = \left[\frac{dL}{dy_1}, \frac{dL}{dy_2}, \frac{dL}{dy_3}\right]$$

Our goal is to find

$$\frac{dL}{dx} \text{ and } \frac{dL}{dw}$$

Backpropagation in Convolution Layer

By chain rule we have $\frac{dL}{dw} = \frac{dL}{dy} \cdot \frac{dy}{dw} = \delta_y \cdot \frac{dy}{dw}$

And $\frac{dy}{dw} = \begin{pmatrix} \frac{dy_1}{dw_1} & \frac{dy_1}{dw_2} \\ \frac{dy_2}{dw_1} & \frac{dy_2}{dw_2} \\ \frac{dy_3}{dw_1} & \frac{dy_3}{dw_2} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \\ x_2 & x_3 \\ x_3 & x_4 \end{pmatrix}$

Thus

same space between

$$\frac{dL}{dw} = \begin{pmatrix} \delta_{w_1} \\ \delta_{w_2} \end{pmatrix} = [\delta_{y_1} \quad \delta_{y_2} \quad \delta_{y_3}] \cdot \begin{pmatrix} x_1 x_2 \\ x_2 x_3 \\ x_3 x_4 \end{pmatrix}$$

$$= \begin{pmatrix} x_1\delta_{y_1} + x_2\delta_{y2} + x_3\delta_{y_3} \\ x_2\delta_{y_1} + x_3\delta_{y2} + x_4\delta_{y_3} \end{pmatrix}$$

$$= \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} * \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Its also a convolution operation

Backpropagation in Pooling Layer

White board

Full Convolutional Neural Network Architecture

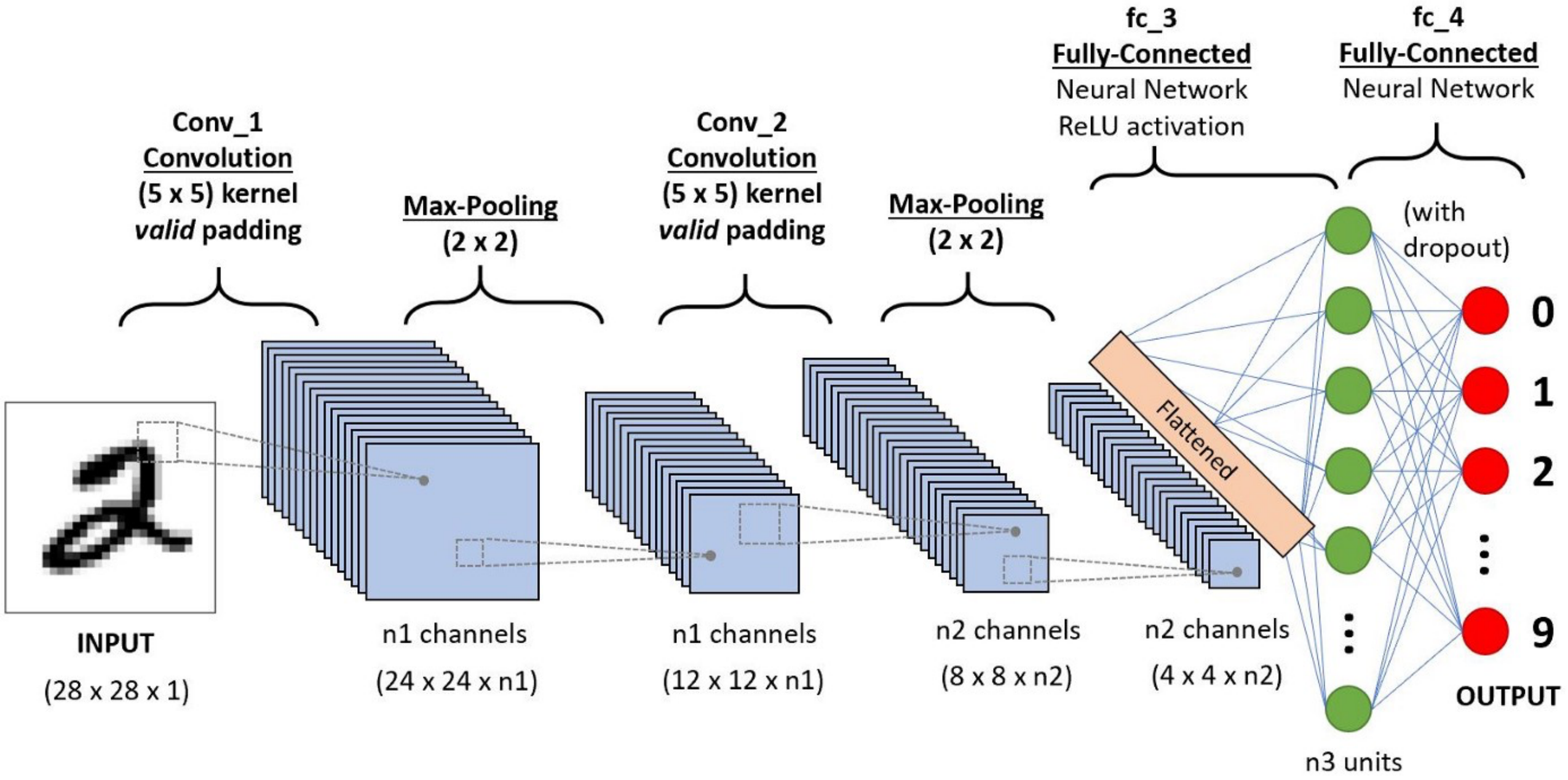


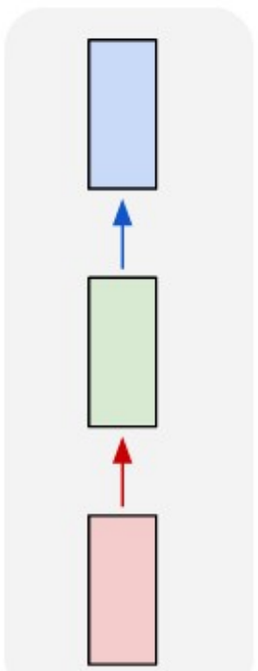
Figure from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Recurrent Neural Networks

Recurrent Neural Networks

Output

one to one



Input

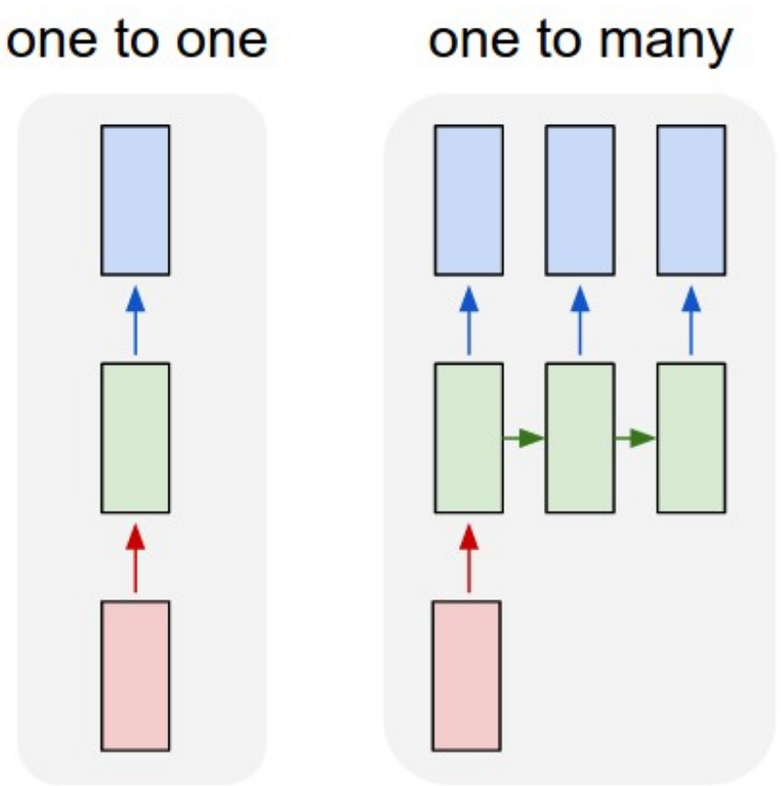
Fully connected and to an extent convolutional networks fall in this category

Input is an image —> Output is a label

Figure from https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture12.pdf

Recurrent Neural Networks

Output



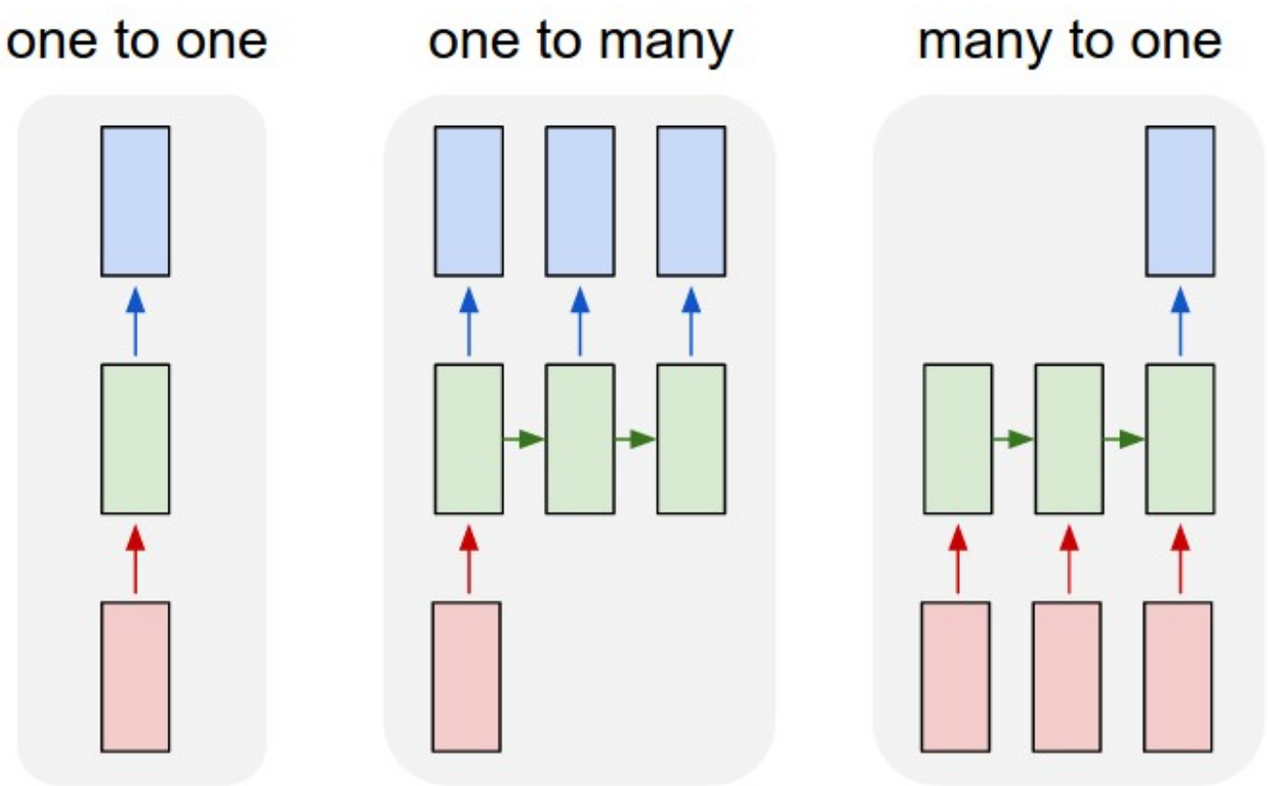
Input

What if, given an image, we want to generate a caption?

Figure from https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture12.pdf

Recurrent Neural Networks

Output

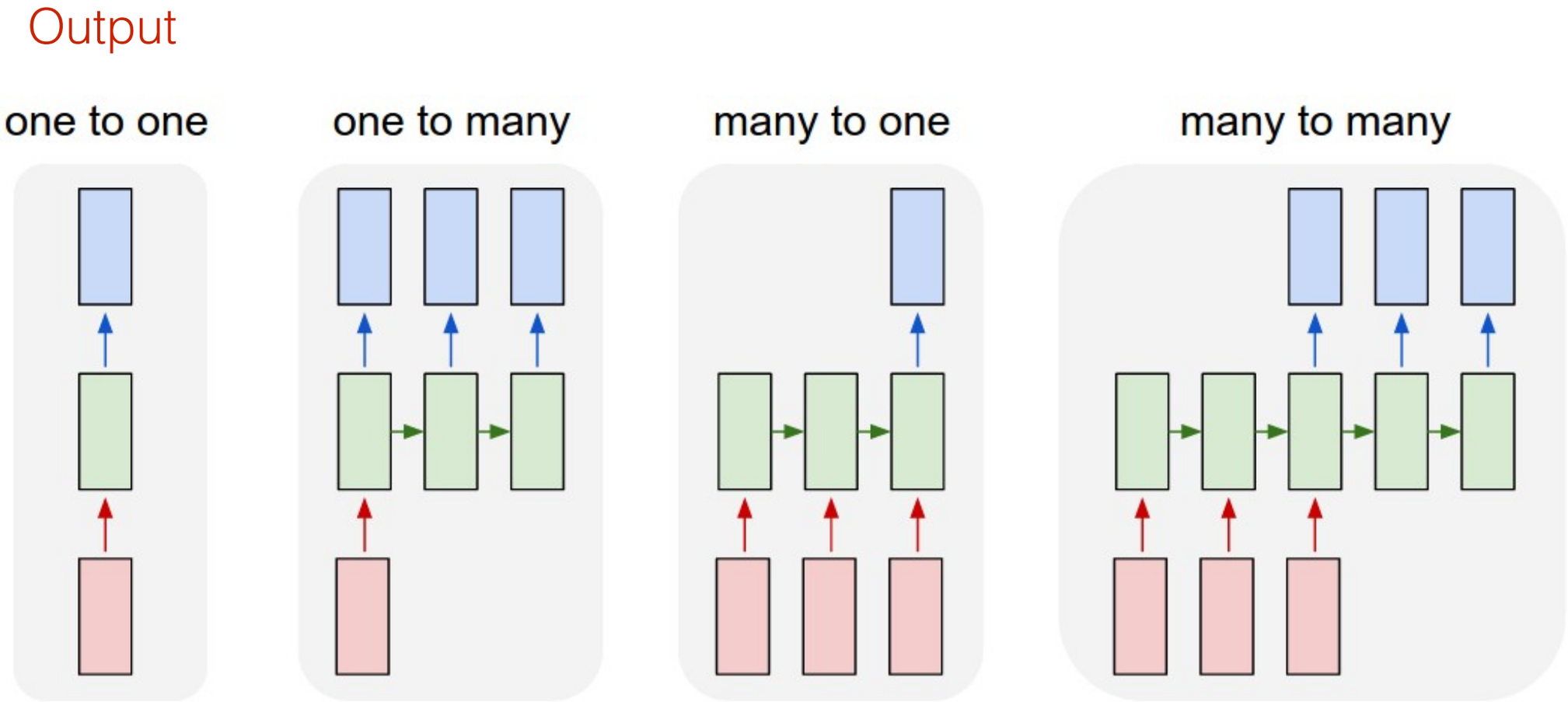


Input

What if, given a review of a movie written in English (a sentence of variable length) we want to infer whether the sentiment is positive or negative?

Figure from https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture12.pdf

Recurrent Neural Networks

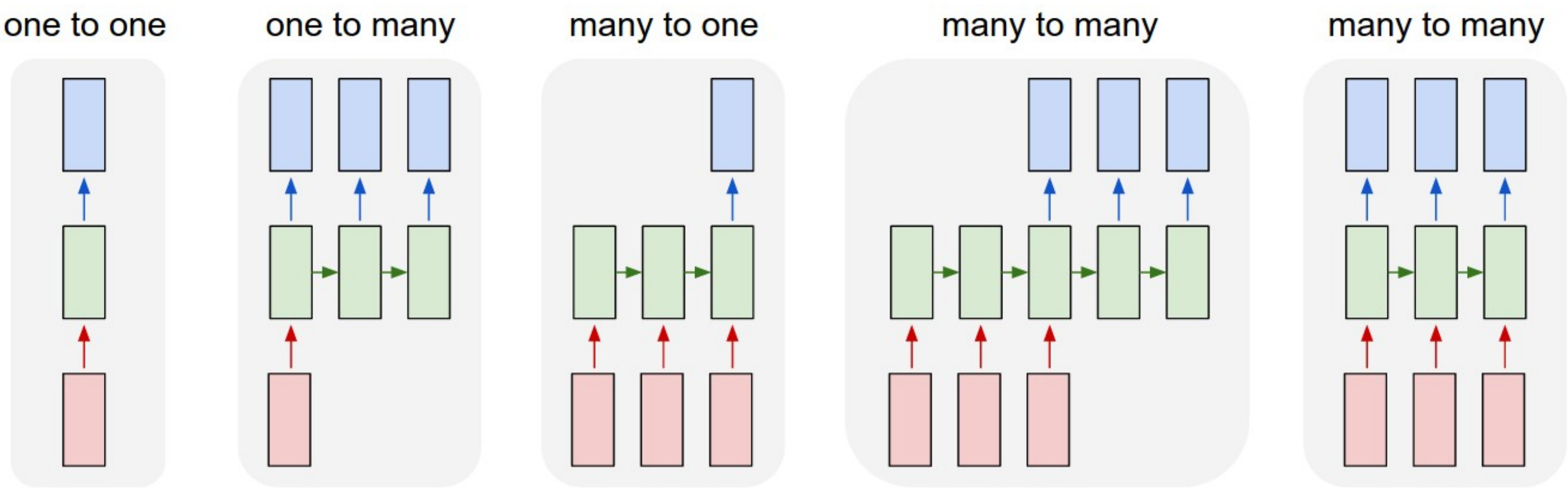


What if, given a sentence (of variable length) written in French, we want to translate it into English?

Figure from https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture12.pdf

Recurrent Neural Networks

Output

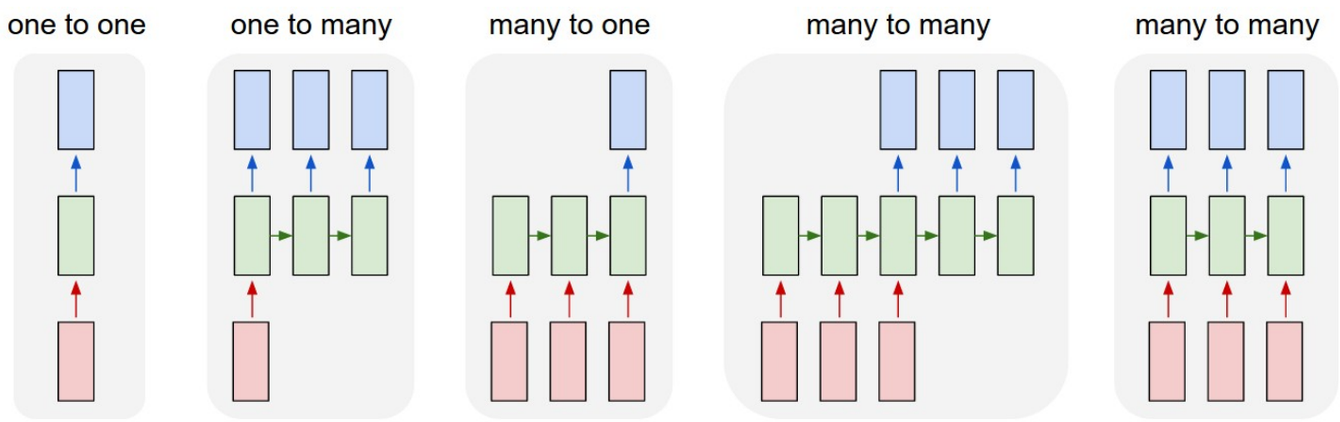


Input

What if, given a video of variable number of frames, we want to classify each frame?

Figure from https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture12.pdf

Recurrent Neural Networks



RNNs are the architecture of choice for such problems

Able to handle variable size inputs

Able to handle variable size outputs

Able to handle sequential data

The key idea behind RNNs is that they maintain an internal memory state which captures the data they have seen thus far. This state gets updated as new information is processed

Figure from https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture12.pdf

Recurrent Neural Networks

Language Modeling Task

Given a sequence of words, predict the next word

The cat drank the [_____]

The cat drank the cow: unlikely

The cat drank the milk: likely

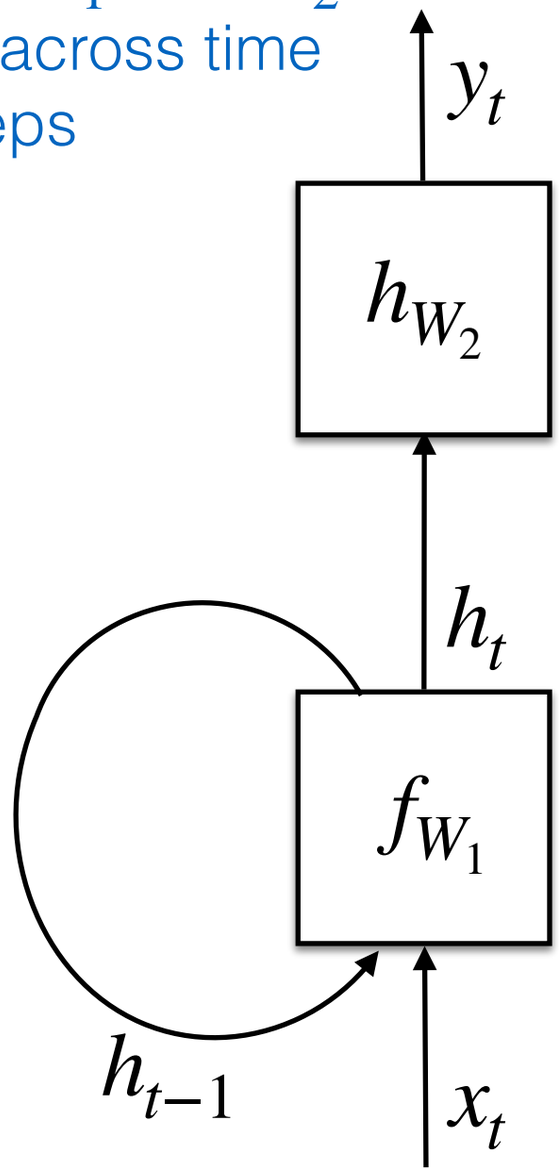
The cat drank the whiskey: kind of unlikely

This is achieved by training a model on a large corpus of text in a way so that given any window of text, it predicts the next word.

Recurrent Neural Networks

The same function is applied recursively to the entire sequence

The weights W_1 and W_2 are shared across time steps



Recurrent Unit

Current output

New state

$$y_t = h_{W_2}(h_t)$$

New state

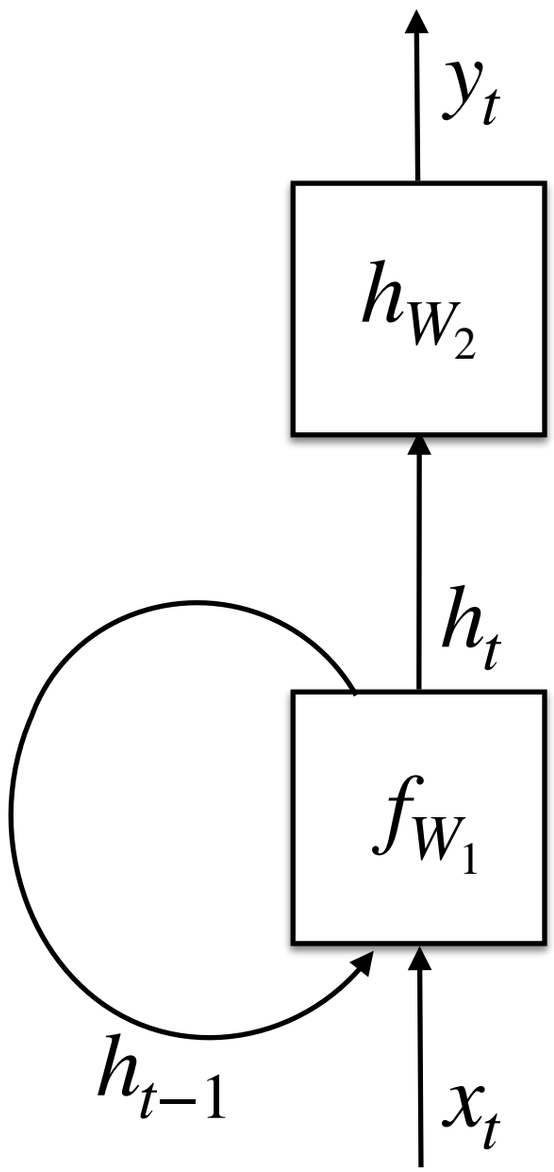
Current input

$$h_t = f_{W_1}(h_{t-1}, x_t)$$

Old state

Recurrent Neural Networks

Elman Networks



Current output

New state

$$y_t = \sigma(W_{hy} h_t)$$

New state

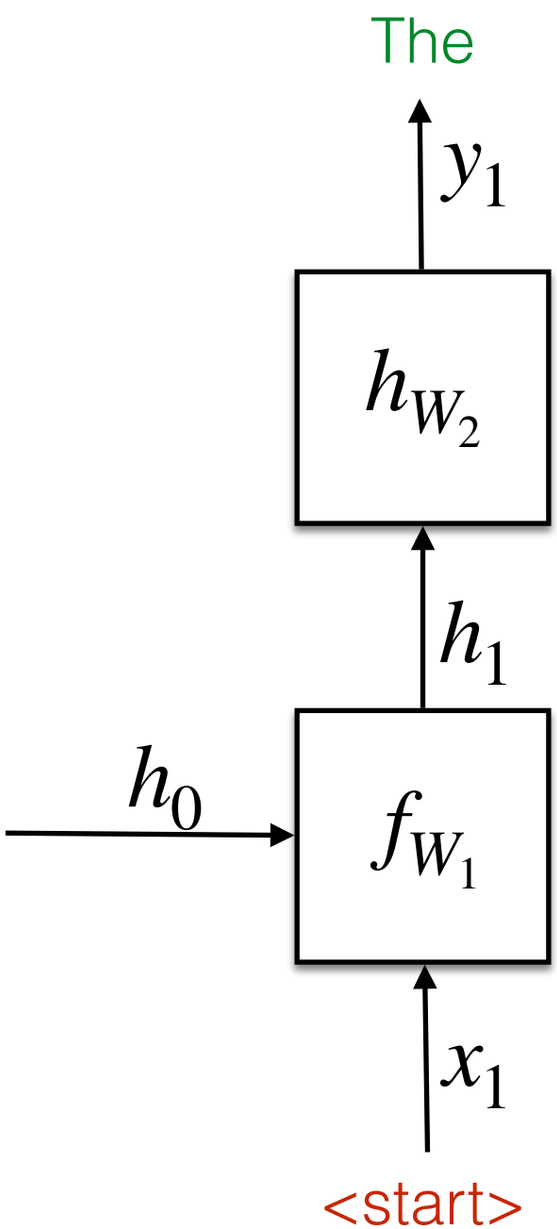
Current input

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

Old state

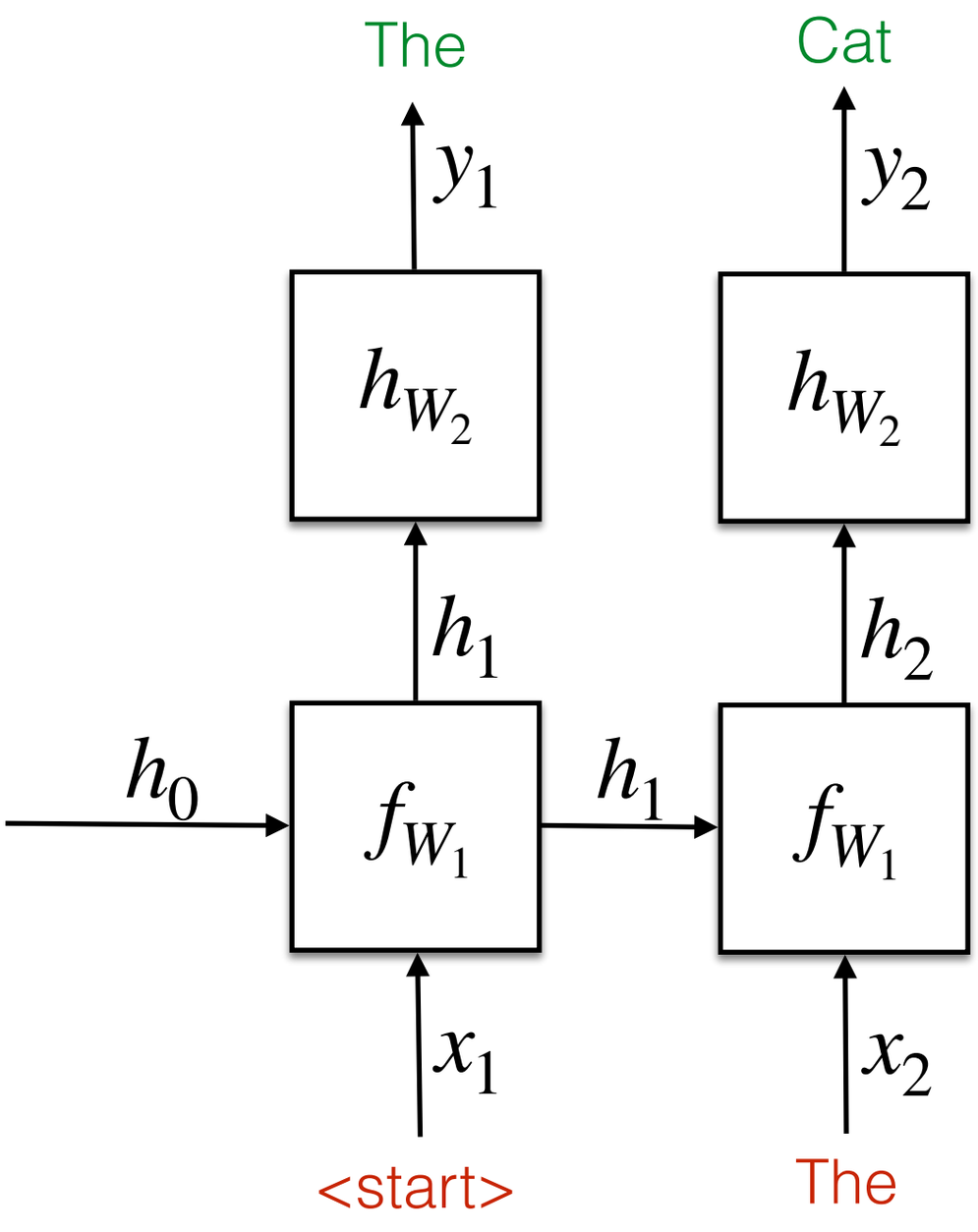
Recurrent Neural Networks

Language Modeling

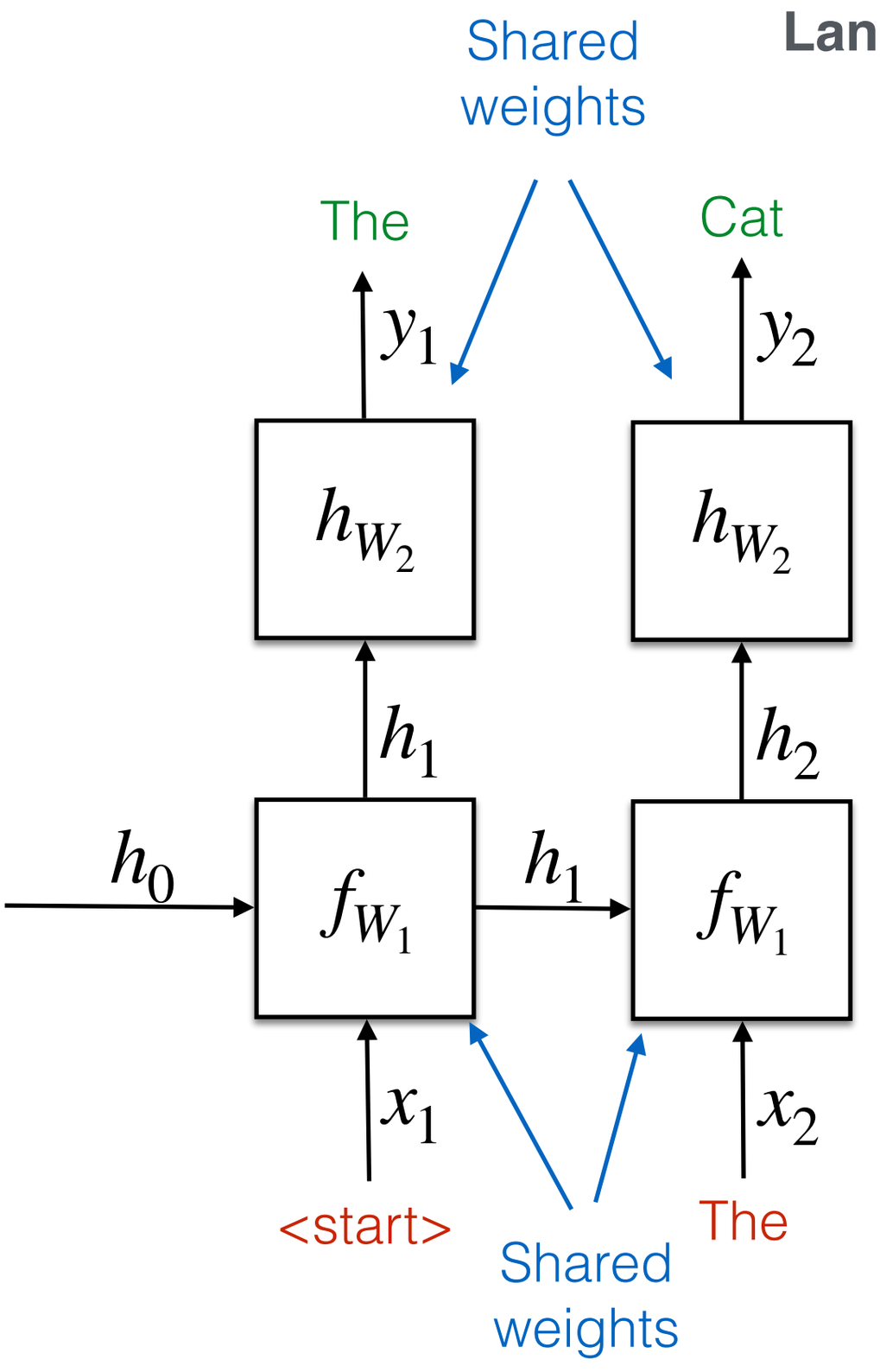


Recurrent Neural Networks

Language Modeling

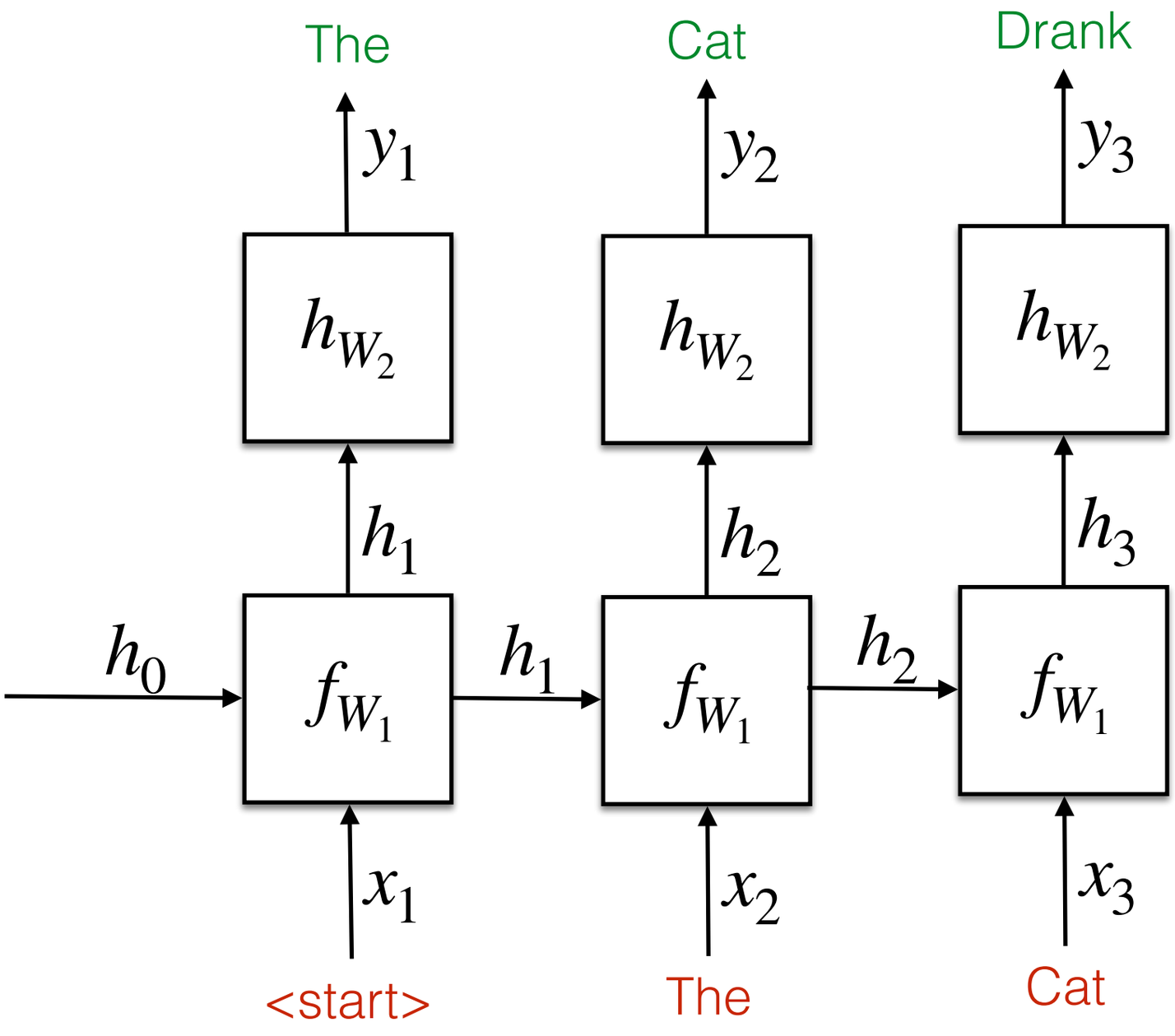


Recurrent Neural Networks



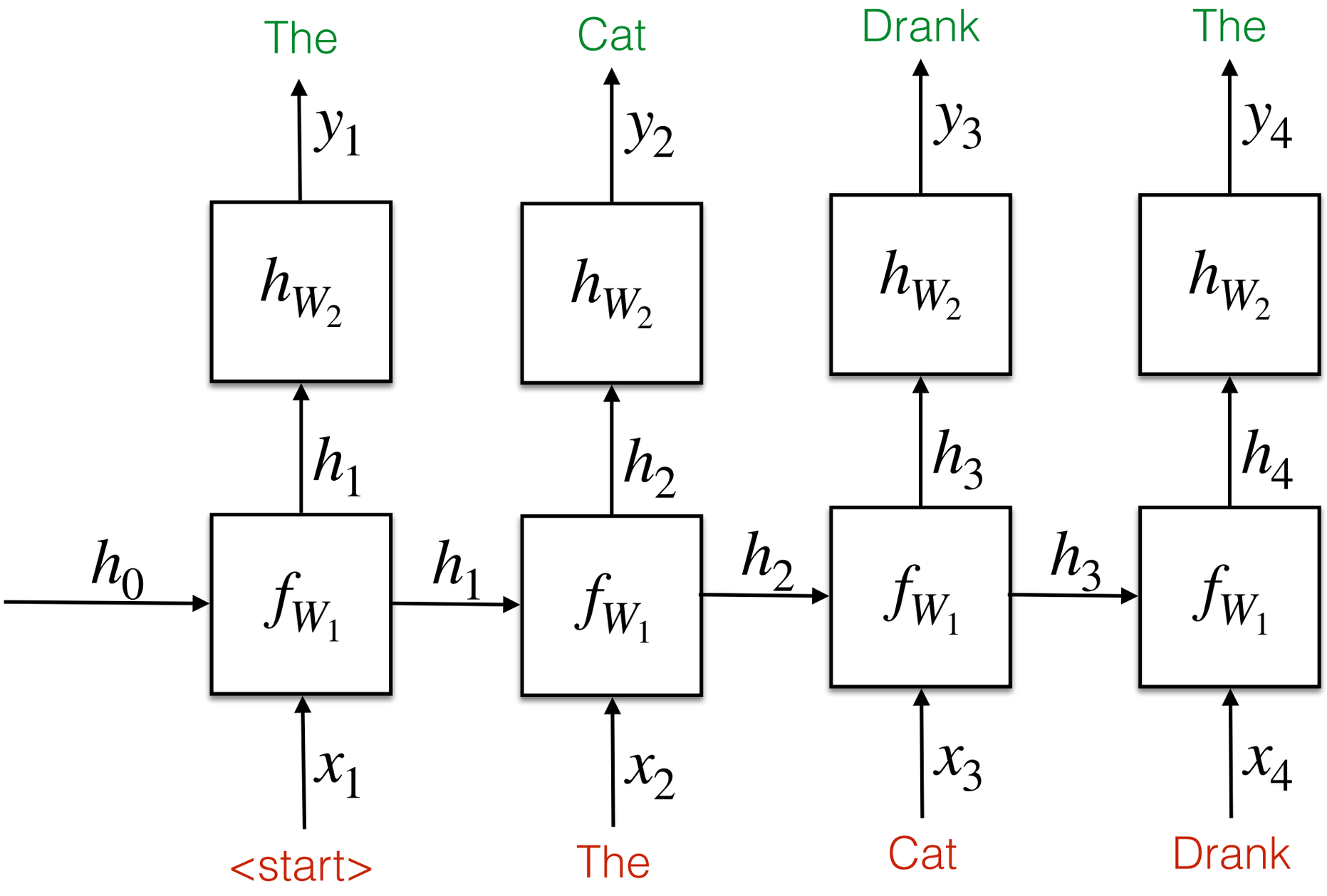
Recurrent Neural Networks

Language Modeling



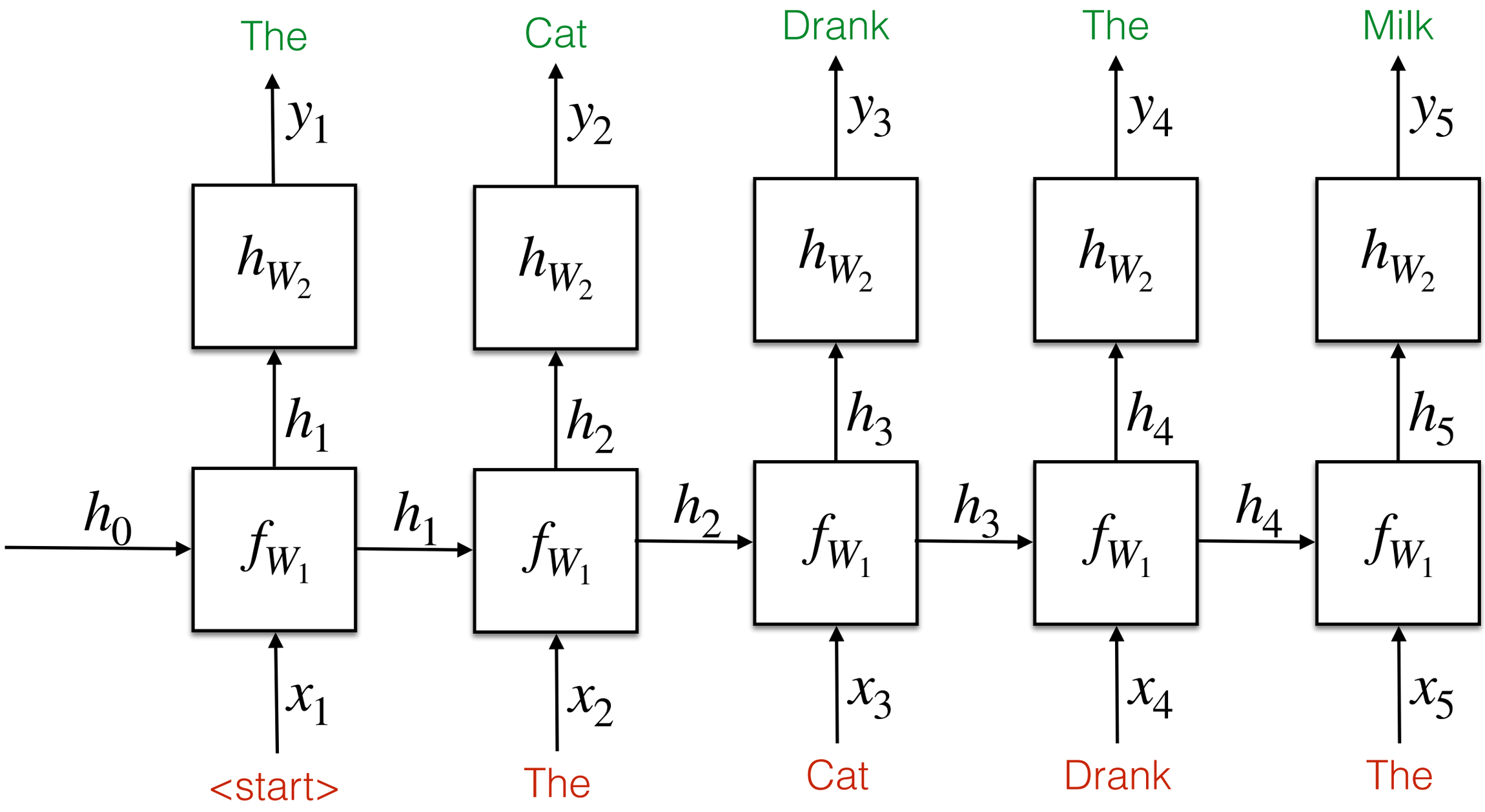
Recurrent Neural Networks

Language Modeling

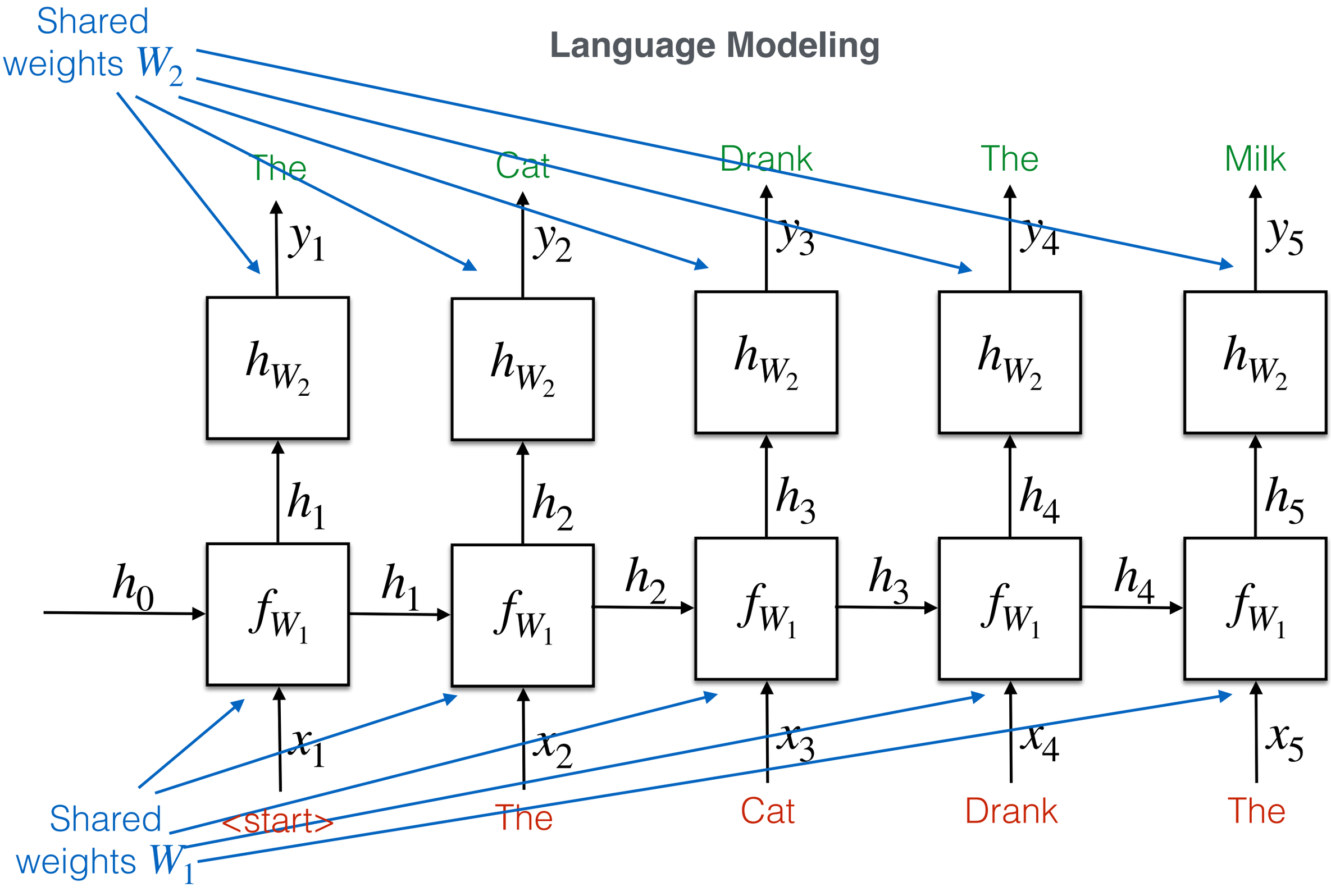


Recurrent Neural Networks

Language Modeling



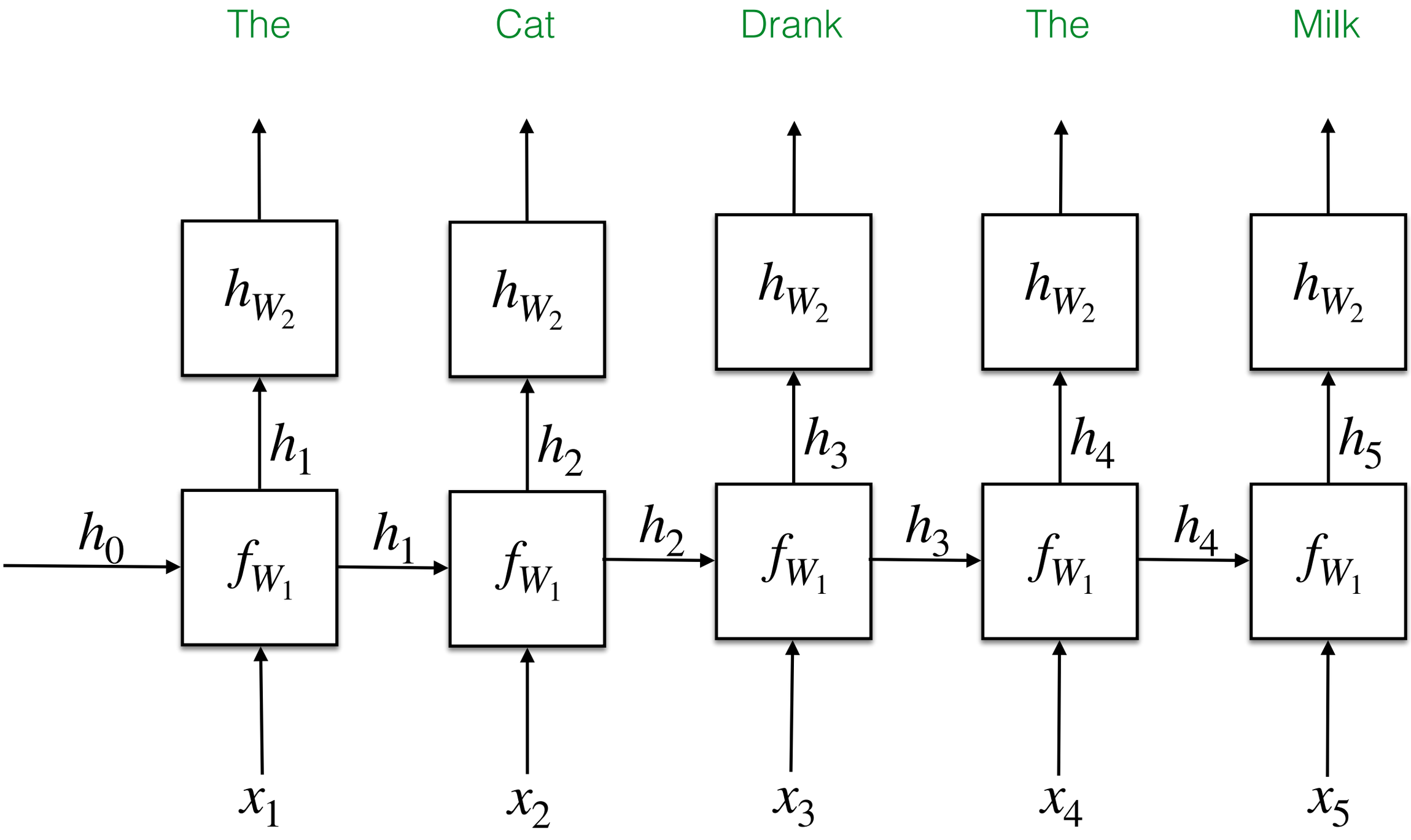
Recurrent Neural Networks



Recurrent Neural Networks Training

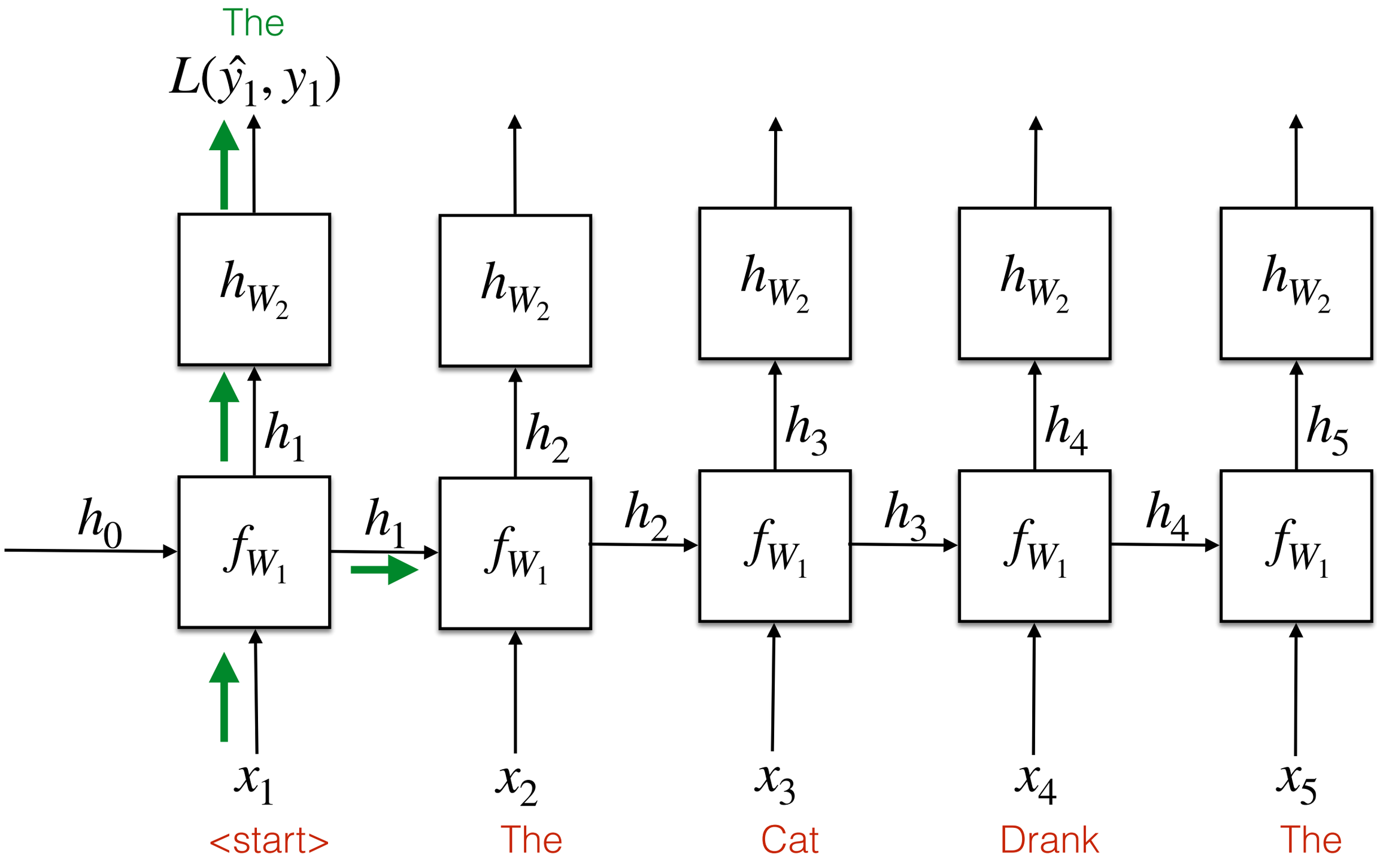
$$Loss = L(\hat{y}_i, y_i)$$

Back Propagation Through Time



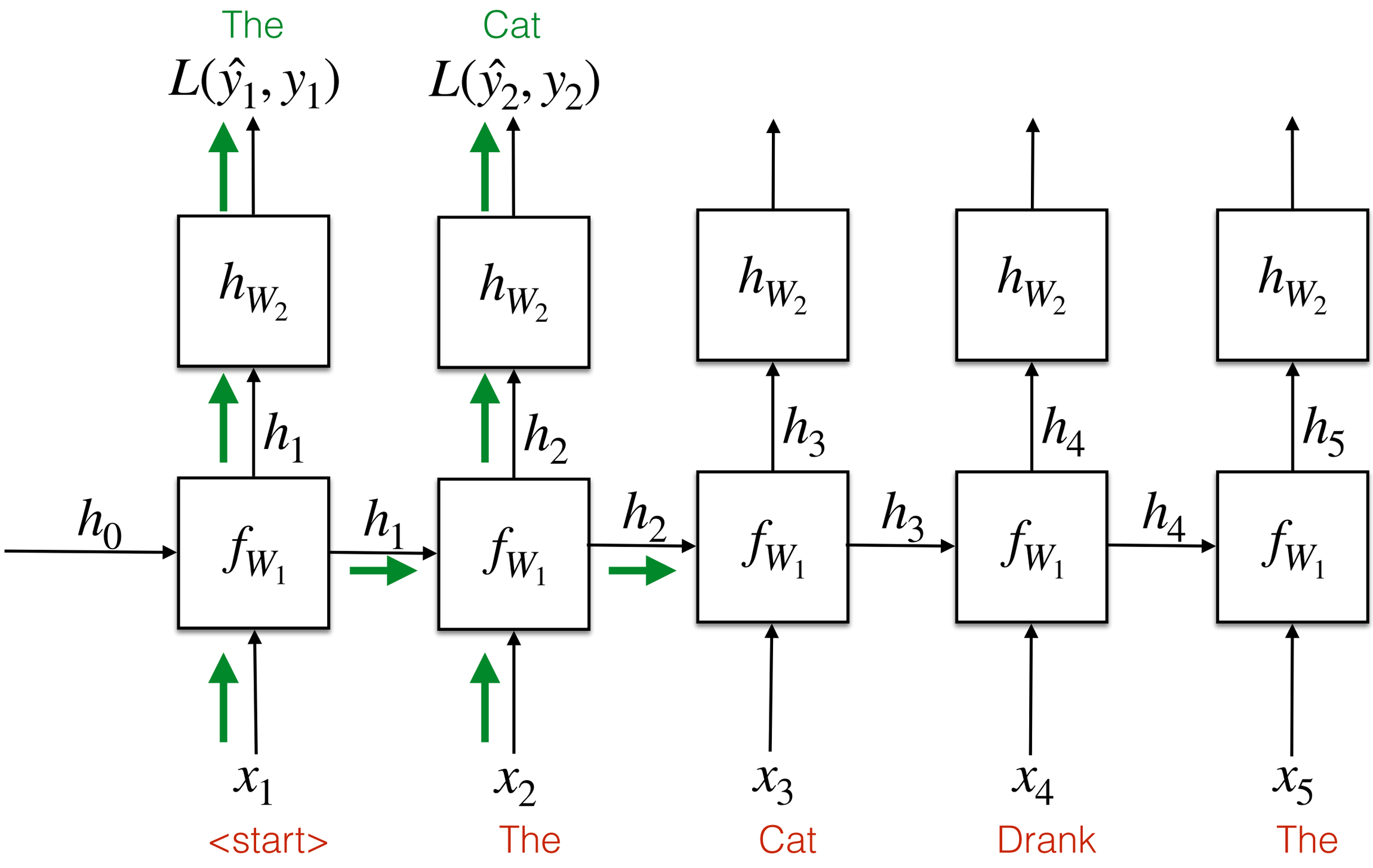
Recurrent Neural Networks Training

Back Propagation Through Time: Forward Pass



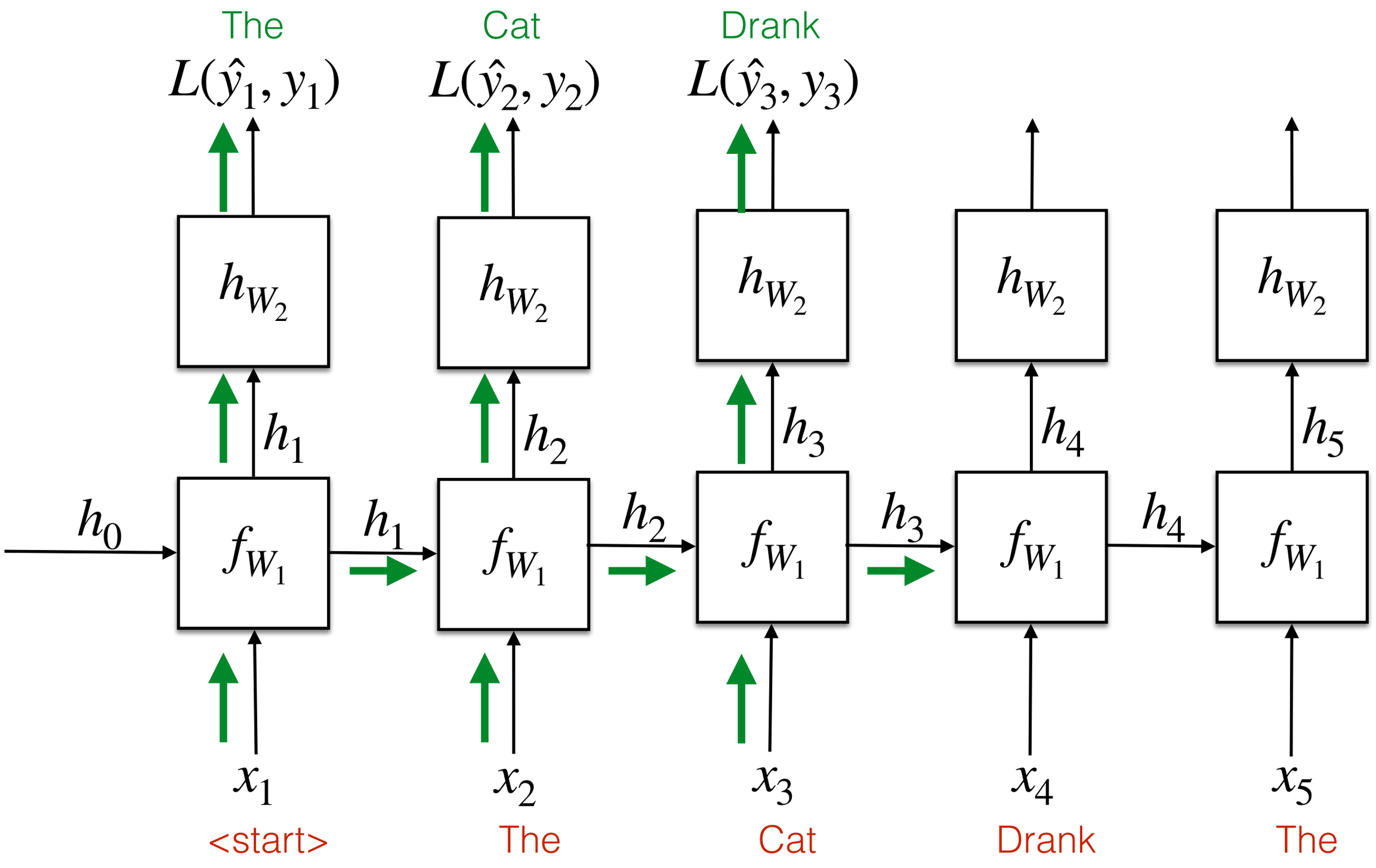
Recurrent Neural Networks Training

Back Propagation Through Time: Forward Pass



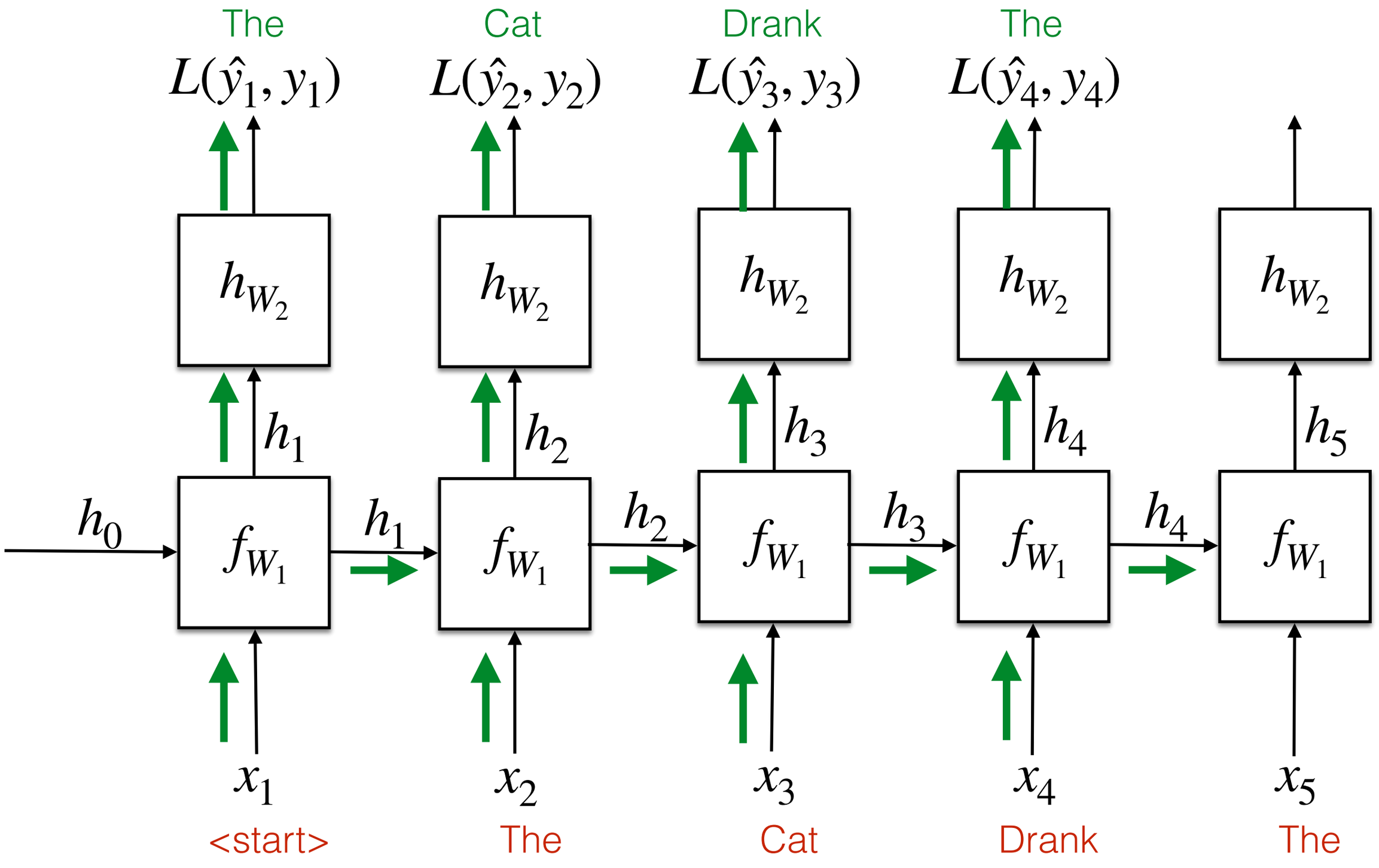
Recurrent Neural Networks Training

Back Propagation Through Time: Forward Pass



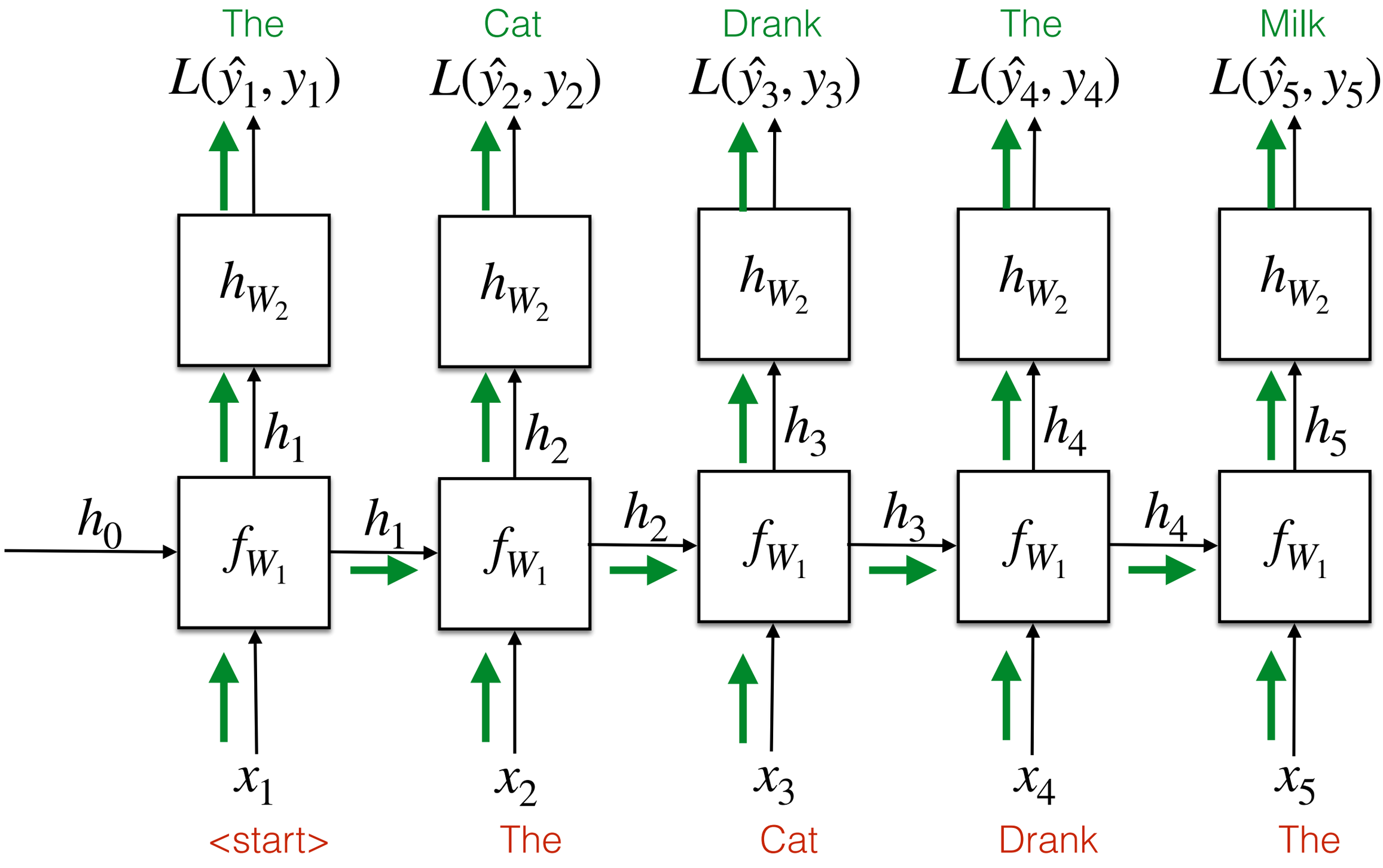
Recurrent Neural Networks Training

Back Propagation Through Time: Forward Pass



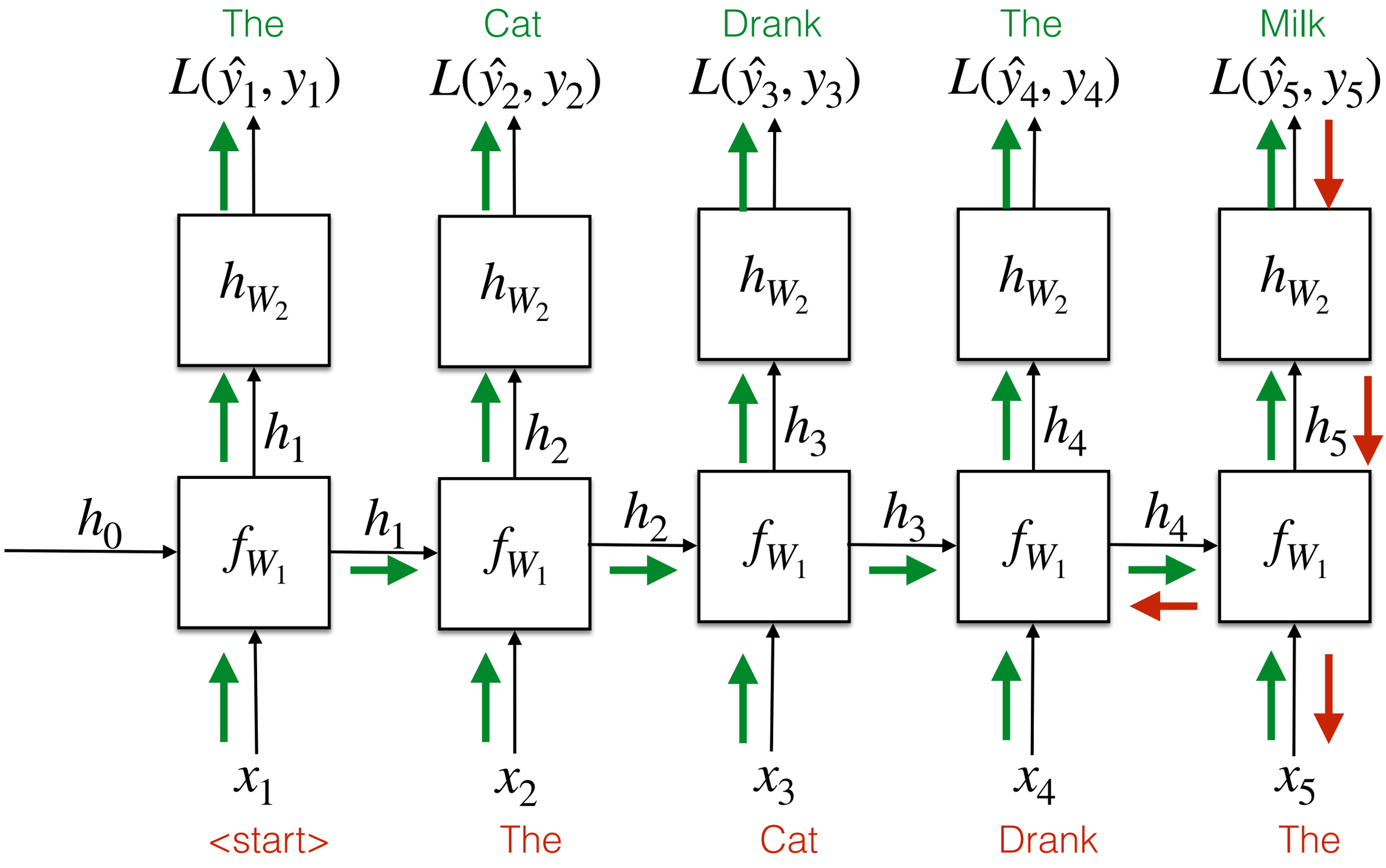
Recurrent Neural Networks Training

Back Propagation Through Time: Forward Pass



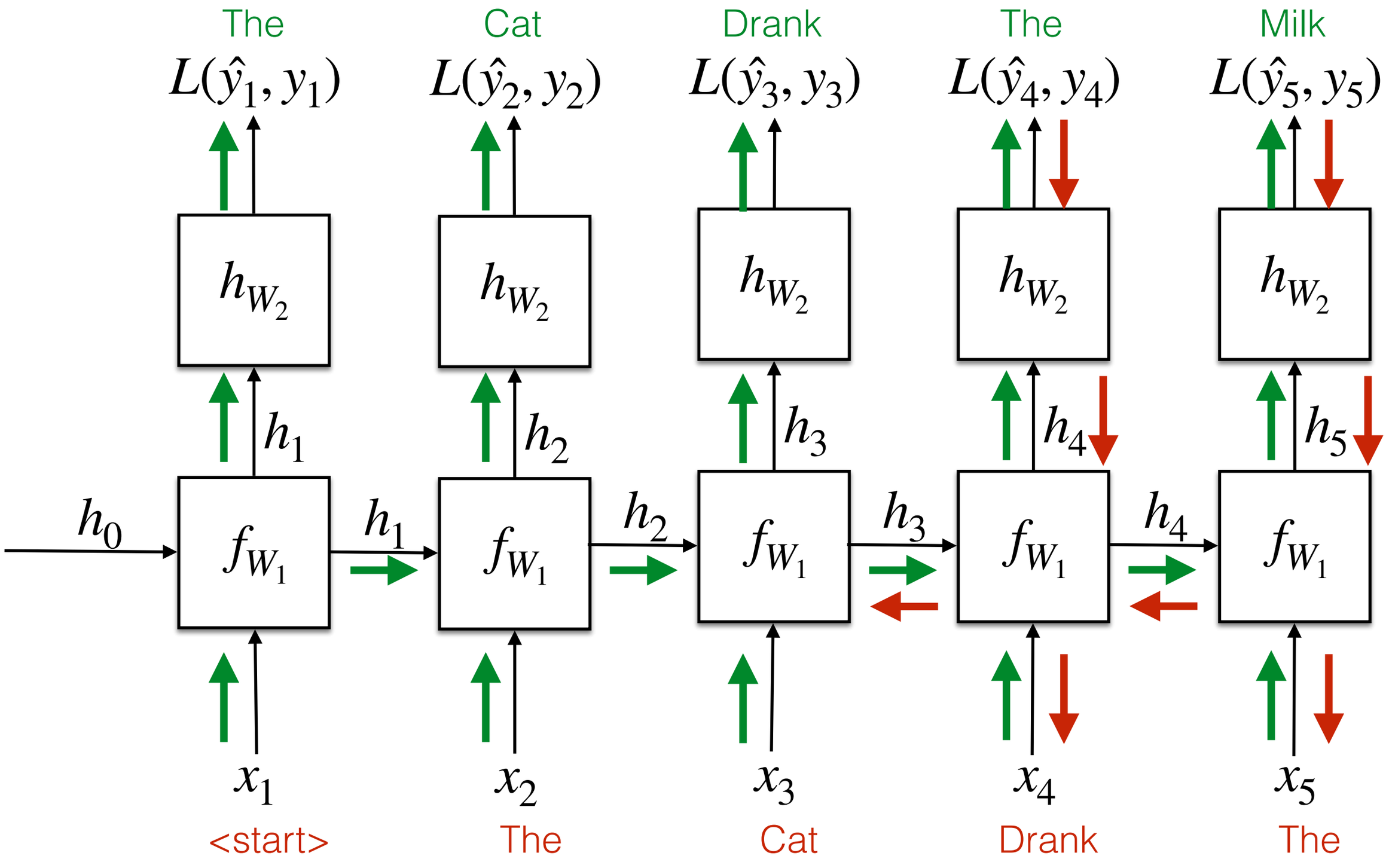
Recurrent Neural Networks Training

Back Propagation Through Time: Backward Pass



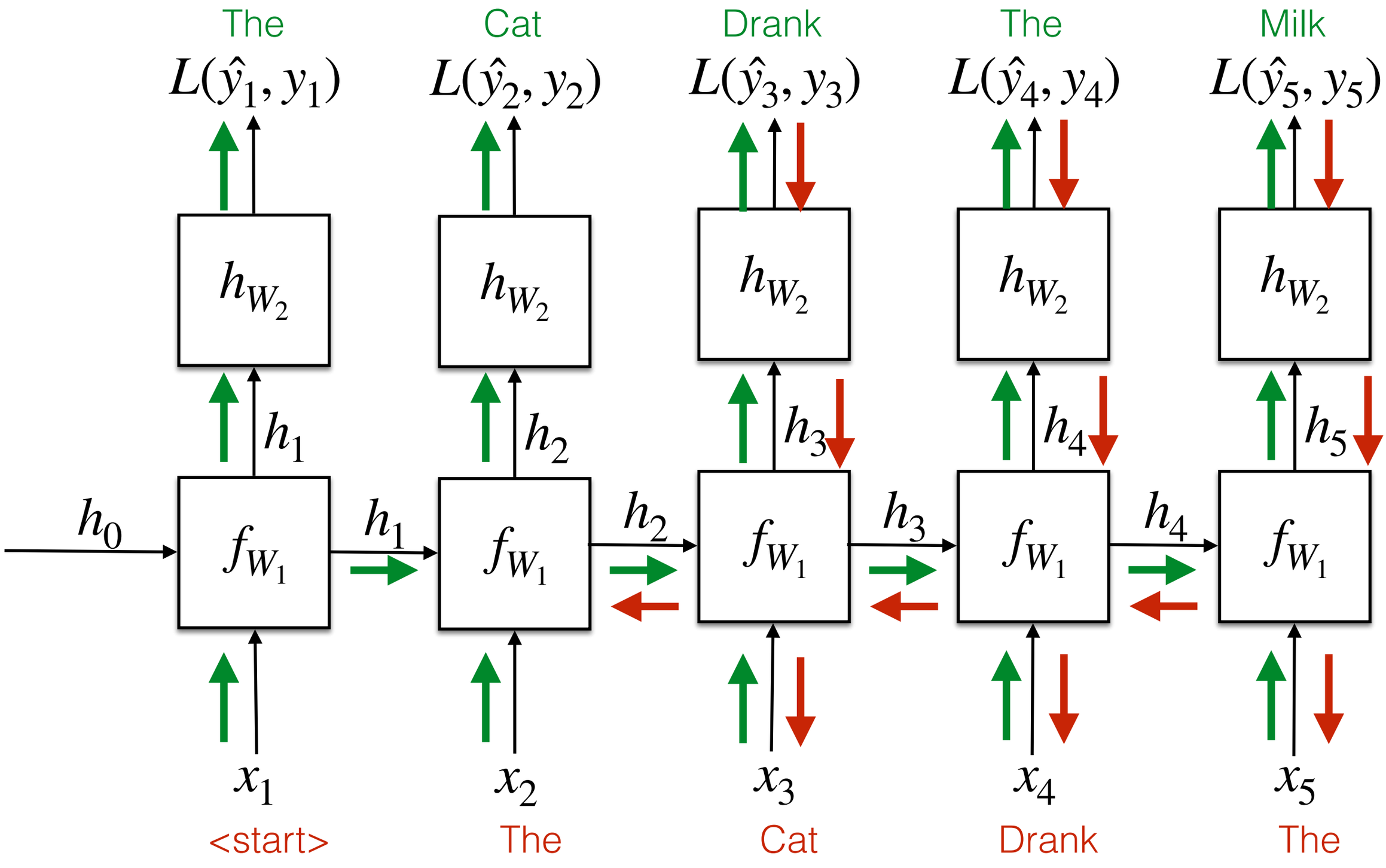
Recurrent Neural Networks Training

Back Propagation Through Time: Backward Pass



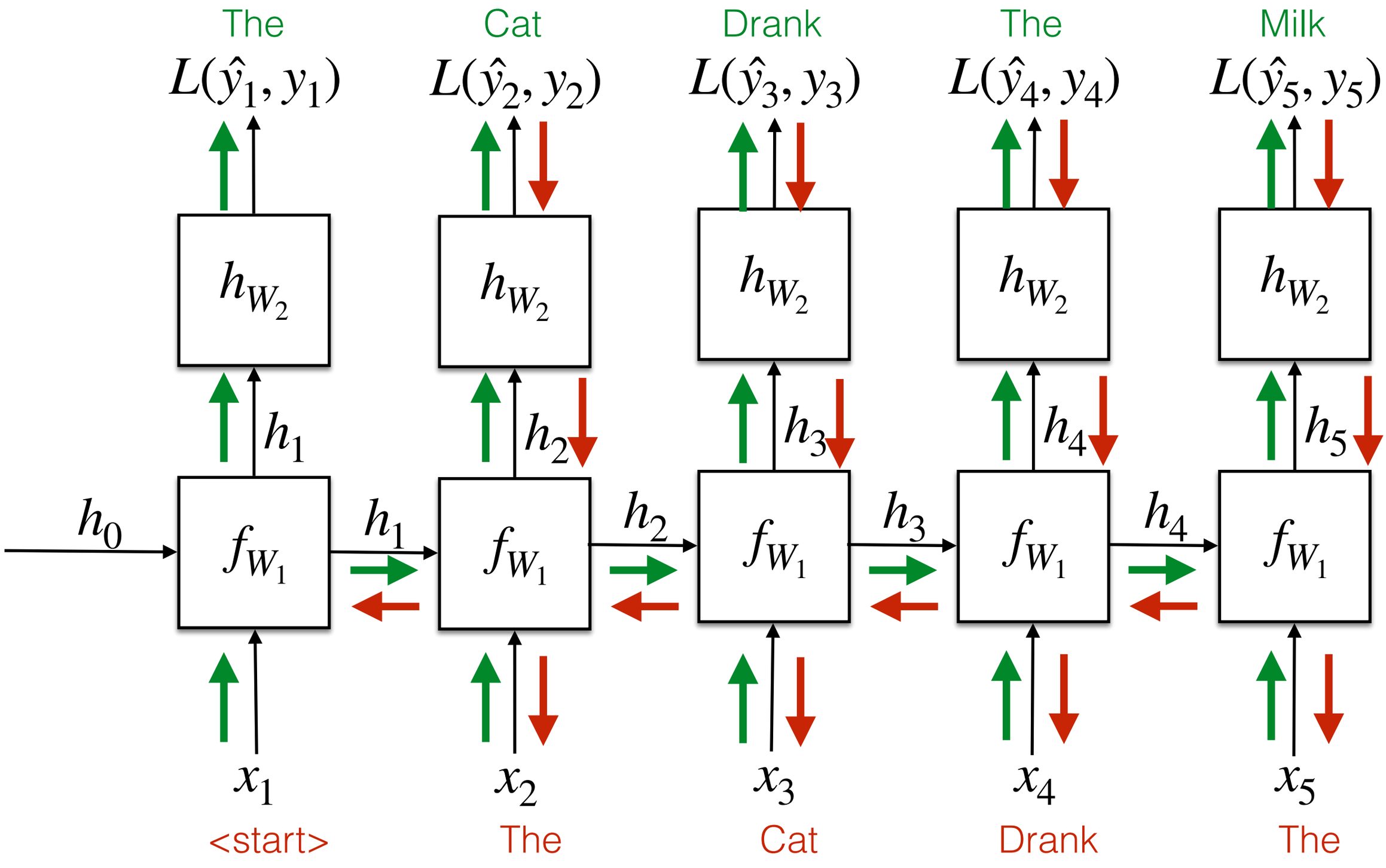
Recurrent Neural Networks Training

Back Propagation Through Time: Backward Pass



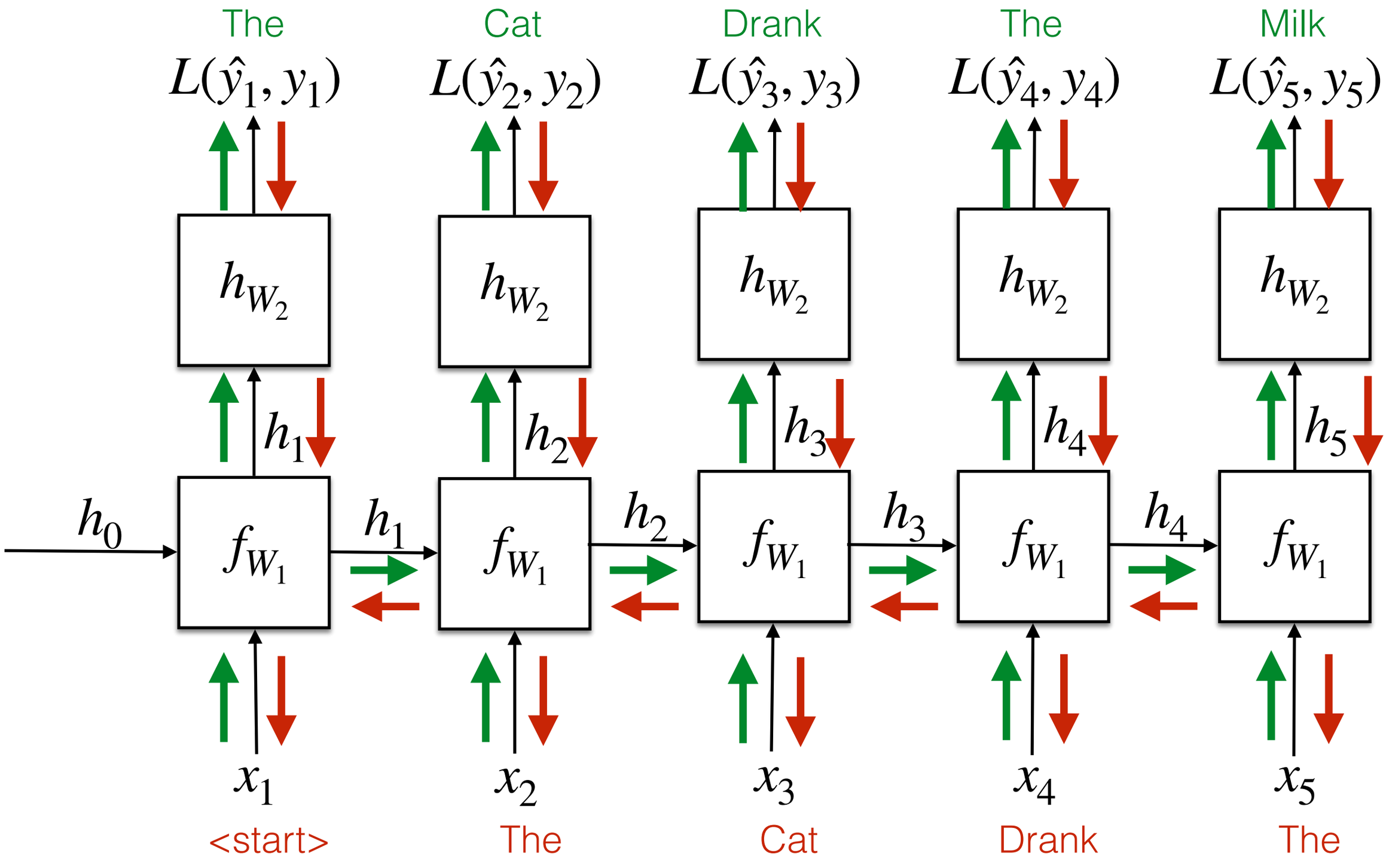
Recurrent Neural Networks Training

Back Propagation Through Time: Backward Pass



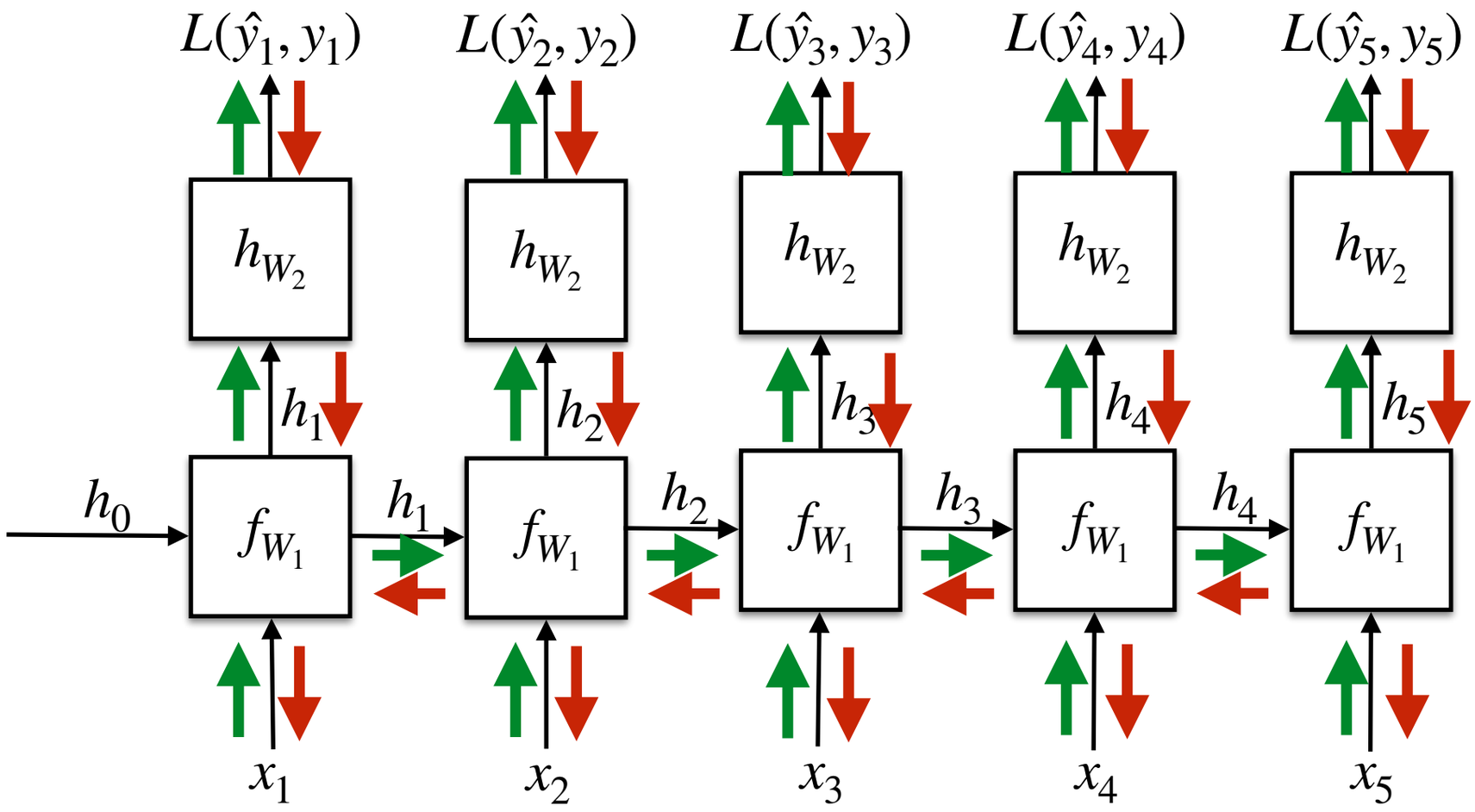
Recurrent Neural Networks Training

Back Propagation Through Time: Backward Pass



Recurrent Neural Networks Training

Back Propagation Through Time



Exploding gradients

Vanishing gradients

End of Lecture 09