Computer Animation

Lecture 4

Deep Learning Introduction

Deep Learning Introduction

- Intro to Neural Networks
- Neural Network Basics
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)

What is deep learning?

- Deep learning allows **computational models** that are composed of **multiple processing layers** to **learn representations of data with multiple levels of abstraction**.

Famous deep learning Nature paper from 2015

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector

intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition[1–4] and speech recognition[5–7], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation[16,17].

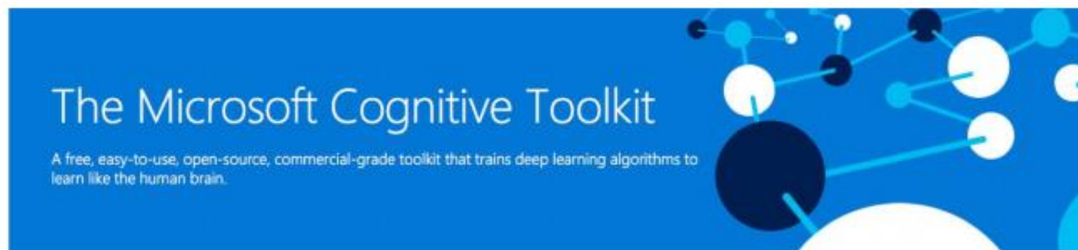
We think that deep learning will have many more successes in the near future because it requires very little engineering by hand, so it can easily take advantage of increases in the amount of available computation and data. New learning algorithms and architectures that are

Software

Caffe

Caffe2

MatConvNet



Large datasets of human motion

- CMU-Panoptic
  - http://domedb.perception.cs.cmu.edu/
- CMU-Mocap-DB
  - http://mocap.cs.cmu.edu/
- HUMAN 3.6M
  - http://vision.imar.ro/human3.6m/description.php
- HUMAN-EVA-I & II
  - http://humaneva.is.tue.mpg.de/datasets_human_1
  - http://humaneva.is.tue.mpg.de/datasets_human_2

Large datasets of human motion (Non-verbals)

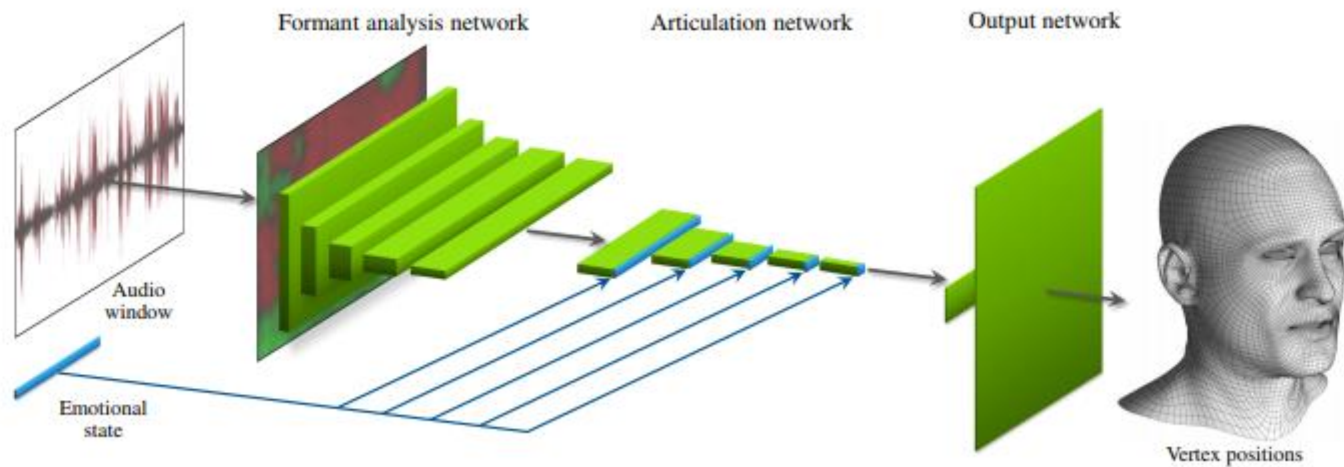
- Trinity Speech-Gesture Dataset: https://trinityspeechgesture.scss.tcd.ie/
- Talking With Hands 16.2 M: https://github.com/facebookresearch/TalkingWithHands32M
- IEMOCAP: https://link.springer.com/article/10.1007/s10579-008-9076-6
- VOCASET: https://voca.is.tue.mpg.de/
- BEAT: https://pantomatrix.github.io/BEAT
- (more are mentioned in papers)

https://paperswithcode.com/
https://paperswithcode.com/datasets

Applications

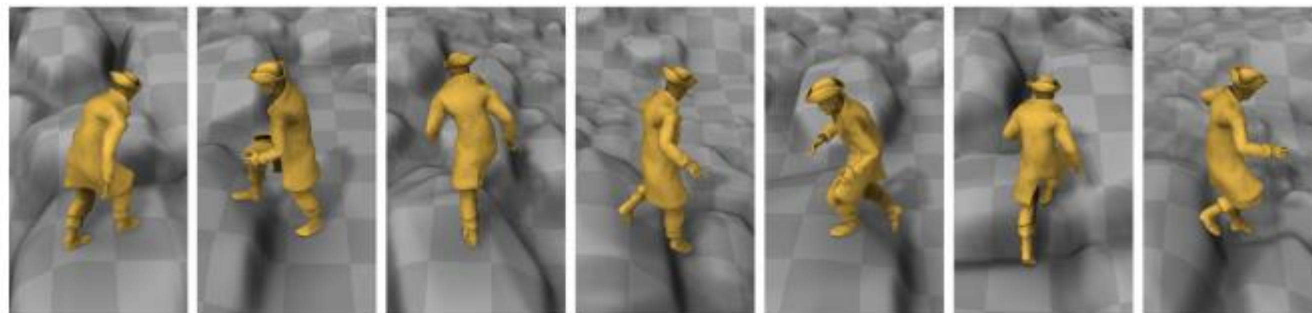
- Computer Vision
- Speech Recognition
- Natural Language Understanding/Generation
- Computer Games
- **Computer Animation**
- Human-robot interaction
- Self-driving cars
- Recommender systems
- …

Computer Animation



https://www.youtube.com/watch?v=lDzrfdpGqw4

Tero Karras, Timo Aila, Samuli Laine, Antti Herva, and Jaakko Lehtinen. 2017. Audio-driven facial animation by joint end-to-end learning of pose and emotion. ACM Trans. Graph. 36, 4, Article 94 (July 2017)



https://www.youtube.com/watch?v=Ul0Gilv5wvY

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. ACM Trans. Graph. 36, 4, Article 42 (July 2017)

Survey papers

- Bernhard Egger, William A. P. Smith, Ayush Tewari, Stefanie Wuhrer, Michael Zollhoefer, Thabo Beeler, Florian Bernard, Timo Bolkart, Adam Kortylewski, Sami Romdhani, Christian Theobalt, Volker Blanz, and Thomas Vetter. 2020. 3D Morphable Face Models—Past, Present, and Future. ACM Trans. Graph. 39, 5, Article 157 (October 2020), 38 pages.

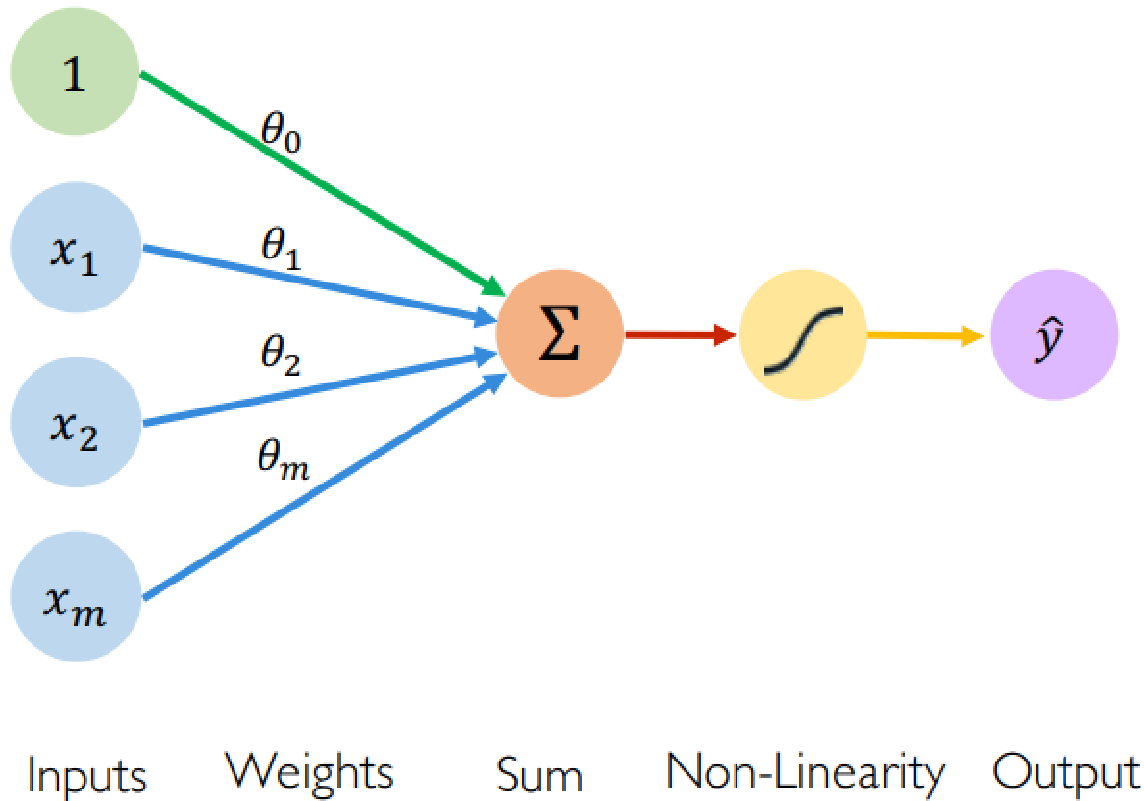
- L. Mourot, L. Hoyet, F. Le Clerc, F. Schnitzler, P. Hellier (2021) A Survey on Deep Learning for Skeleton-Based Human Animation Computer Graphics Forum

- Nyatsanga, S., Kucherenko, T., Ahuja, C., Henter, G.E. and Neff, M. (2023), A Comprehensive Review of Data-Driven Co-Speech Gesture Generation. Computer Graphics Forum, 42: 569-596.

- Nurziya Oralbayeva, Amir Aly, Anara Sandygulova, and Tony Belpaeme. 2023. Data-Driven Communicative Behaviour Generation: A Survey. J. Hum.-Robot Interact. Just Accepted (August 2023).

The Perceptron

- Building block of deep neural networks



Output

Linear combination of inputs

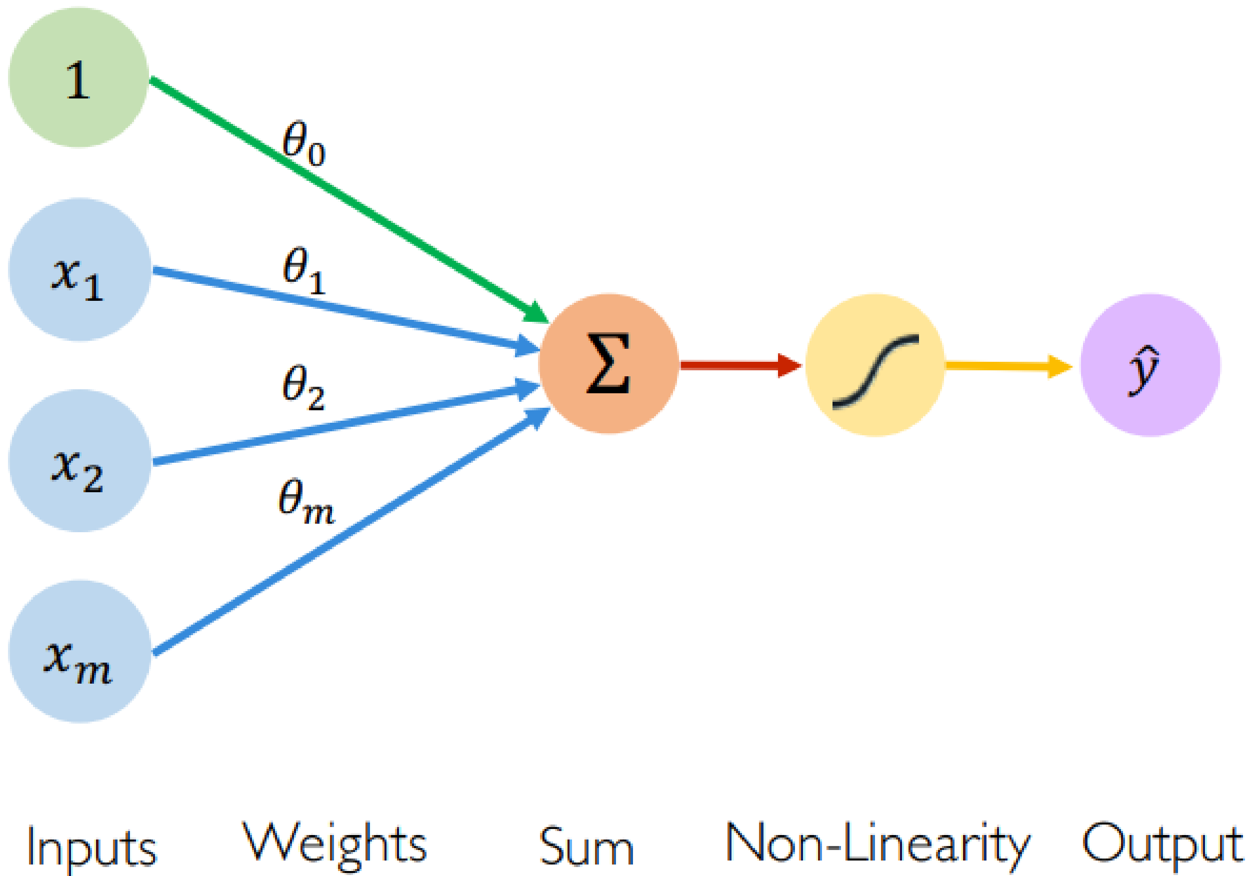
$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \, \theta_i\right)$$

Non-linear activation function

Bias

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

The Perceptron



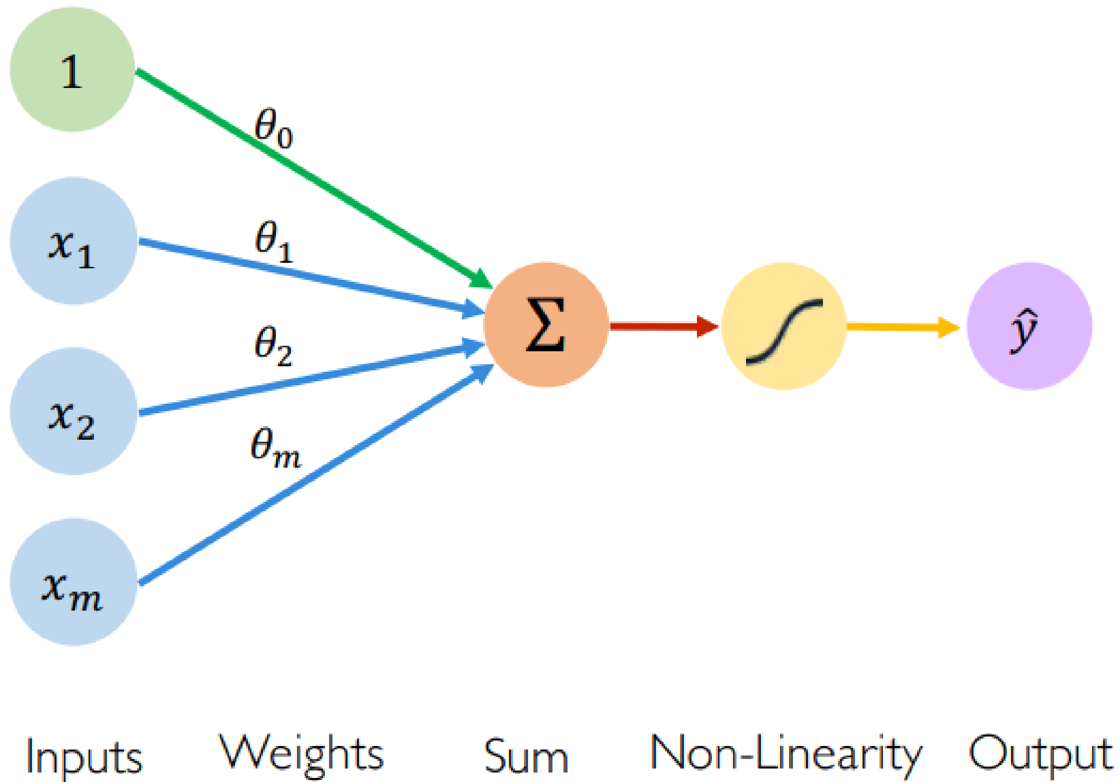
$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \theta_i\right)$$

$$\hat{y} = g\left(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

The Perceptron

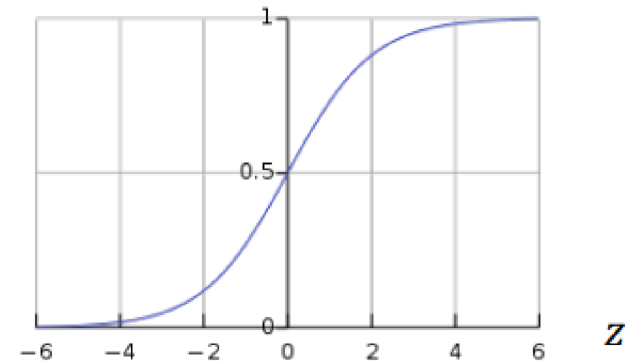


Activation Functions

$$\hat{y} = g\left(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta} \right)$$

- Example: sigmoid function

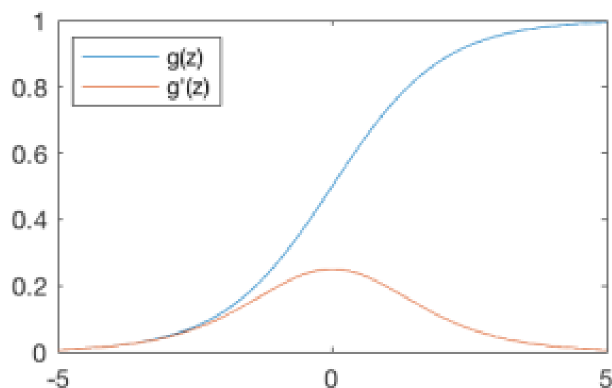
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

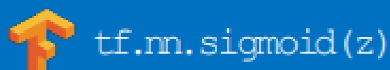
Common Activation Functions

Sigmoid Function

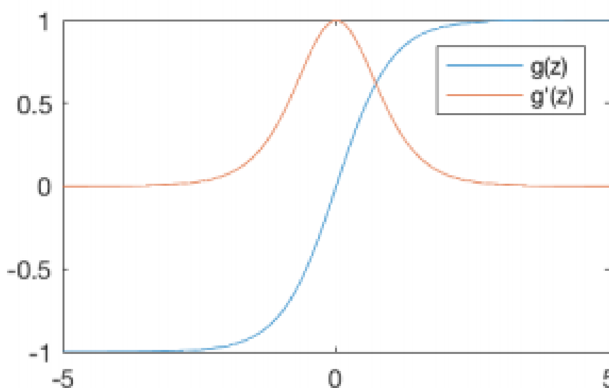


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$



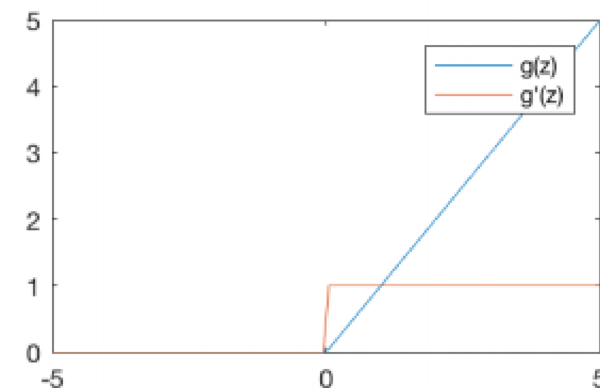
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

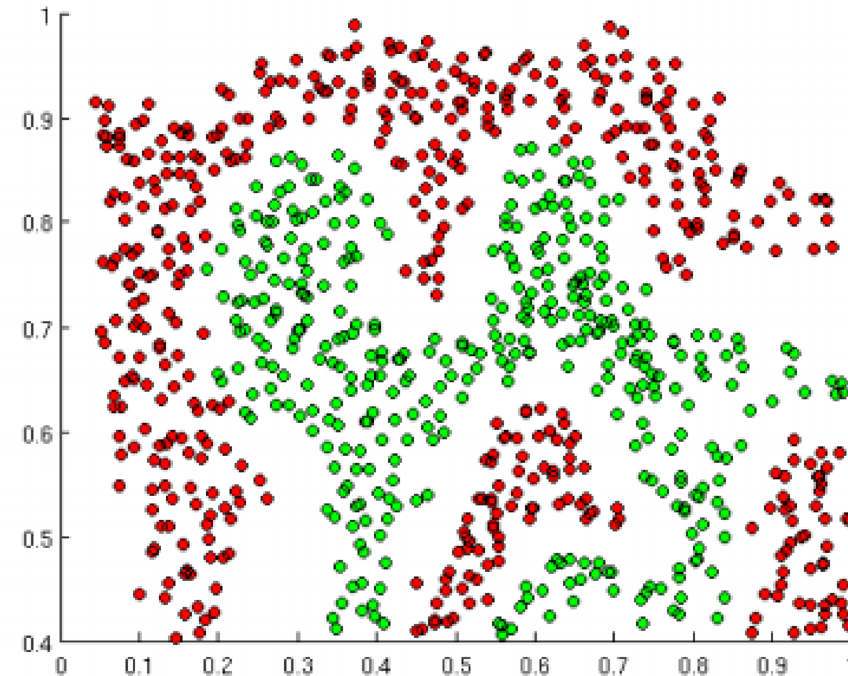
NOTE: All activation functions are non-linear

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Why do we need activation functions?

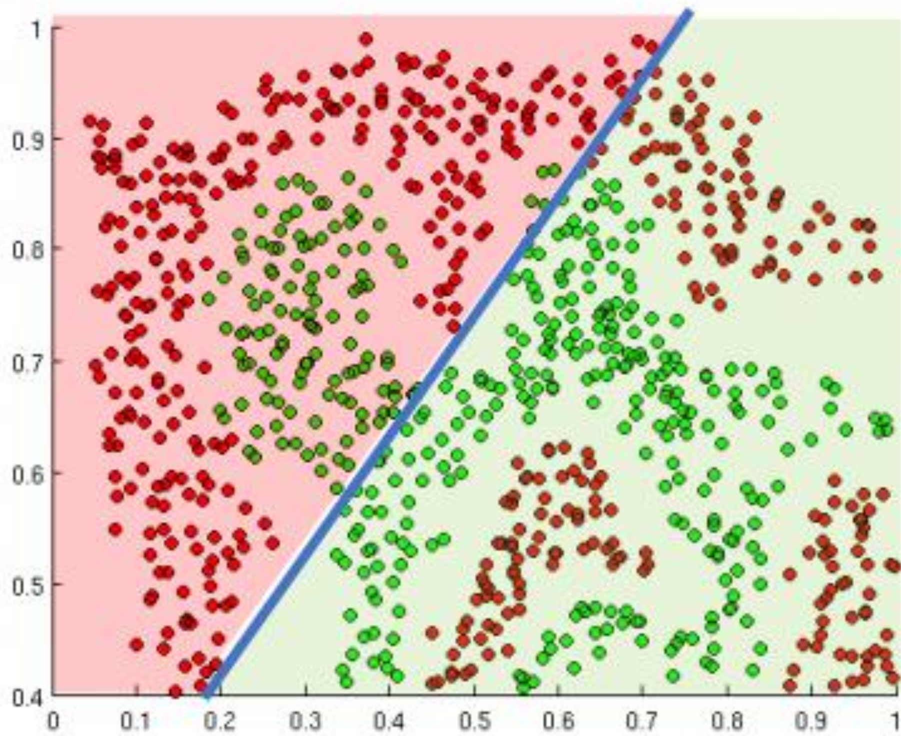
- To introduce non-linearities into the network

How to build a neural network to distinguish red and green points?

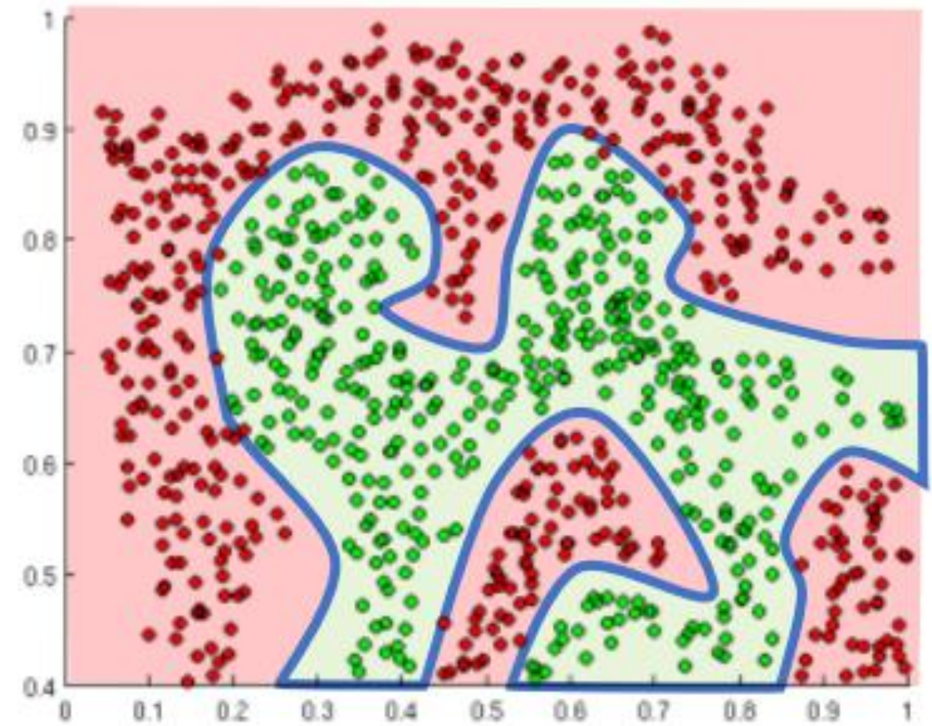


© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Linear vs non-linear activation functions



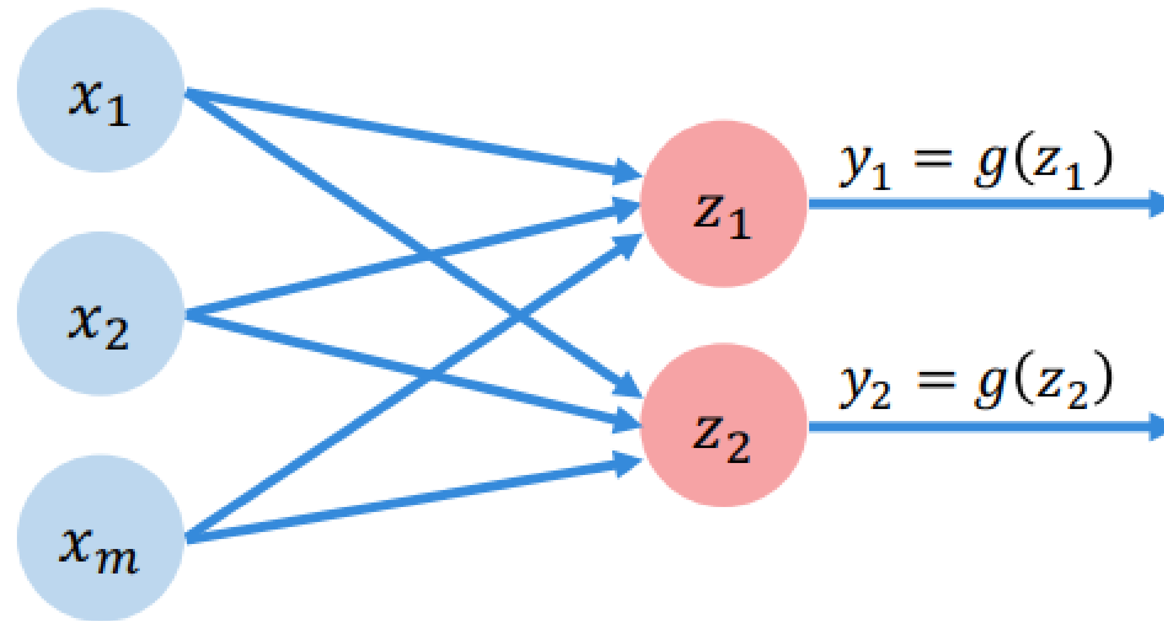
Linear activations produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

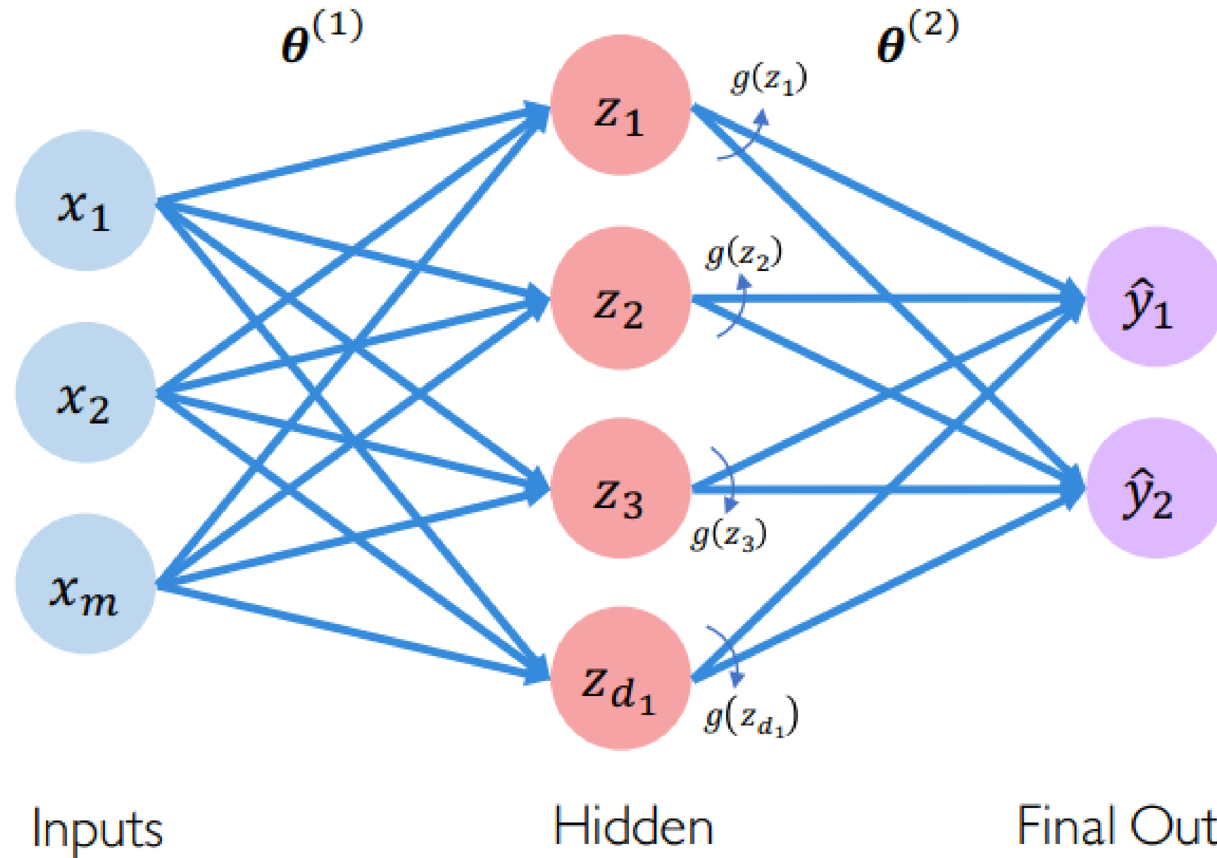
Multi-output perceptron



$$z_i = \theta_{0,i} + \sum_{j=1}^{m} x_j \, \theta_{j,i}$$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

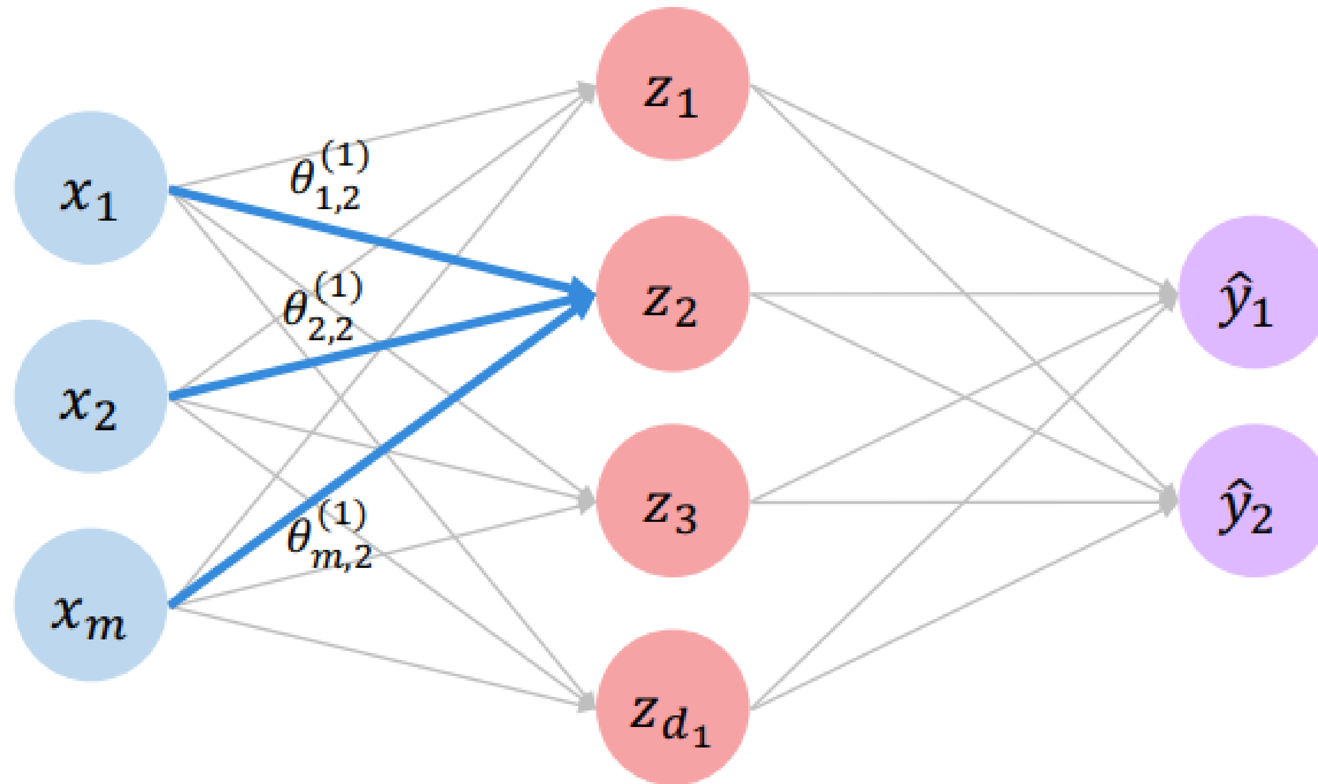
Single hidden layer neural network



$$z_i = \theta_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, \theta_{j,i}^{(1)} \qquad \hat{y}_i = \theta_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, \theta_{j,i}^{(2)}$$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Single hidden layer neural network

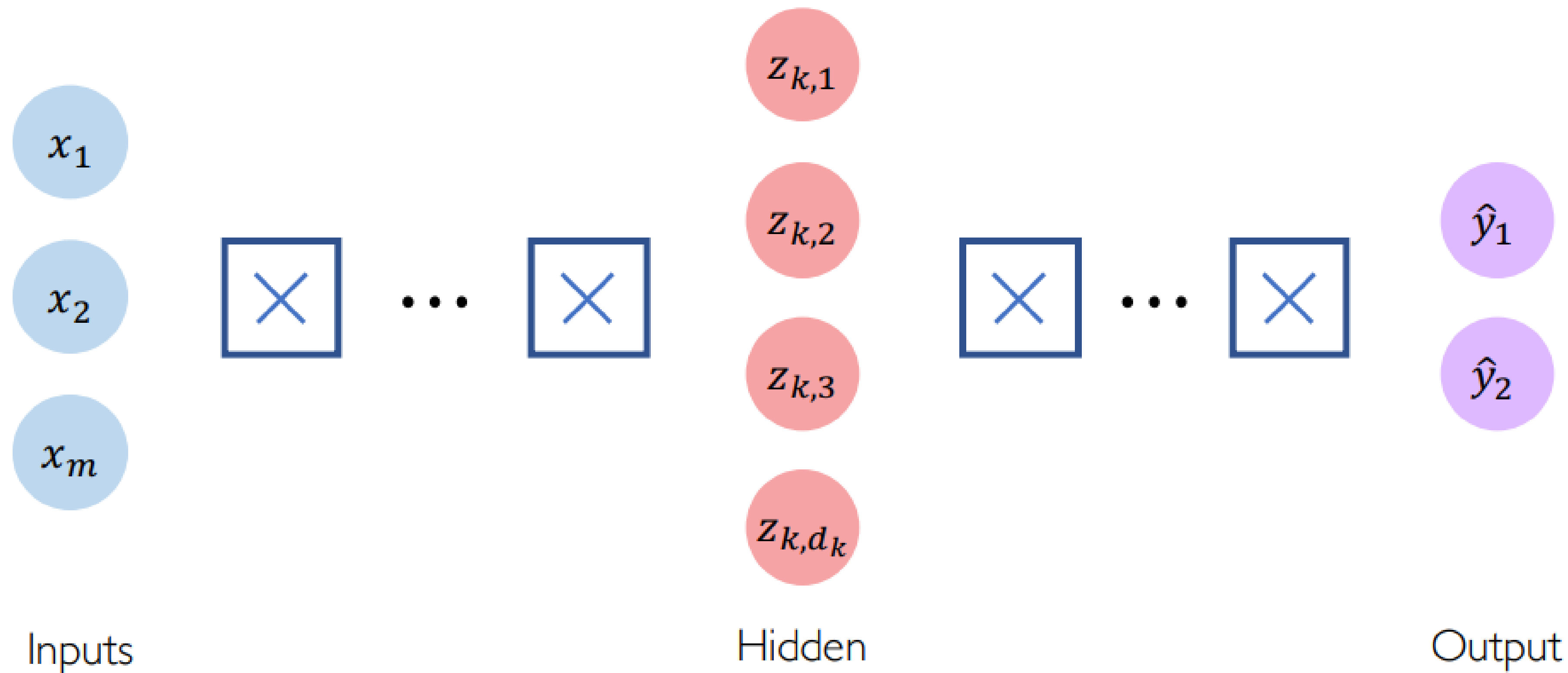


$$z_2 = \theta_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, \theta_{j,2}^{(1)}$$

$$= \theta_{0,2}^{(1)} + x_1 \, \theta_{1,2}^{(1)} + x_2 \, \theta_{2,2}^{(1)} + x_m \, \theta_{m,2}^{(1)}$$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Deep Neural Network



$$z_{k,i} = \theta_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, \theta_{j,i}^{(k)}$$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Example Problem

Will I pass this class?

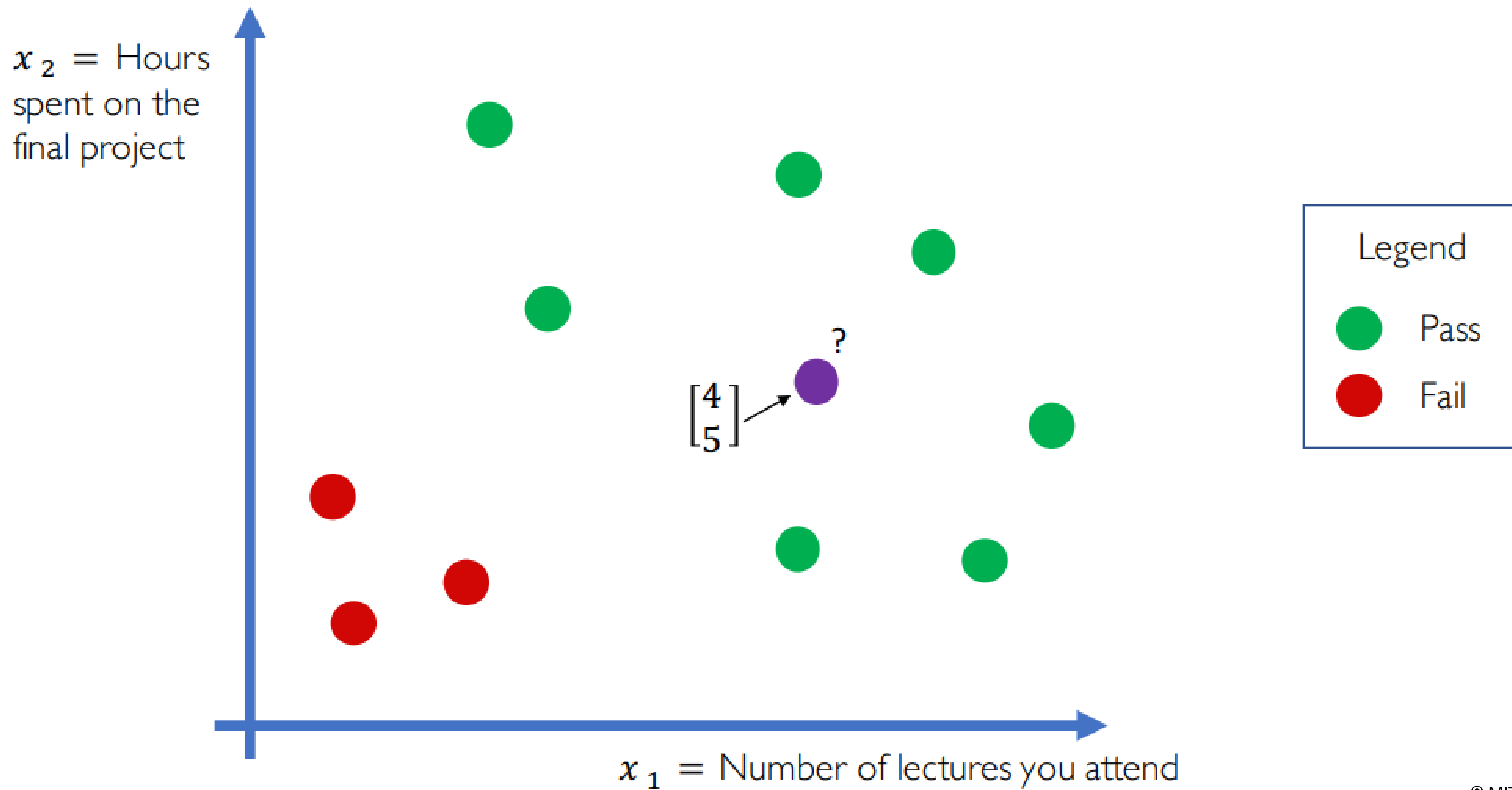
Let's start with a simple two feature model

x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

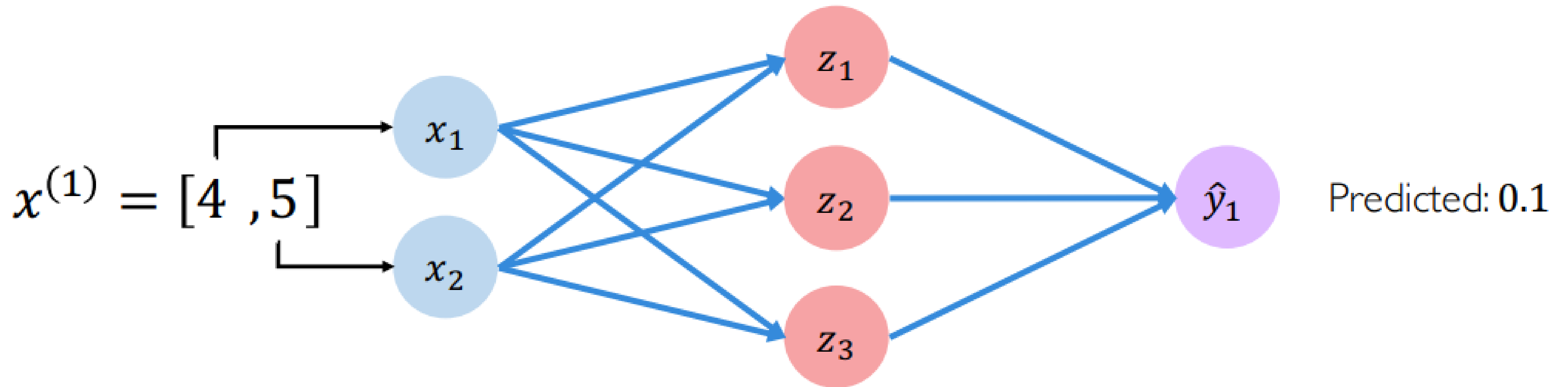
© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Example Problem



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

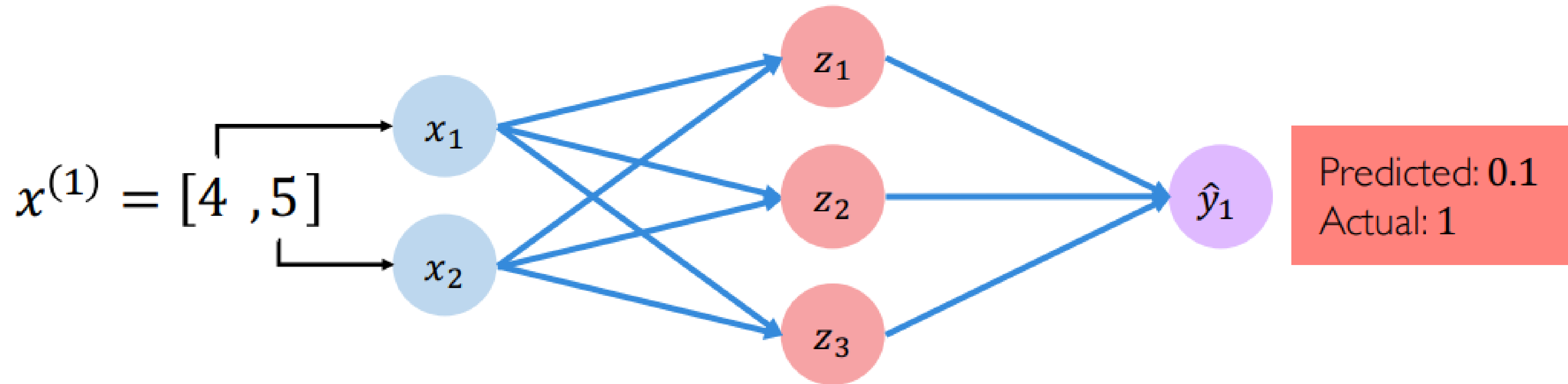
Example Problem



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Quantifying loss

*The **loss** of our network measures the cost incurred from incorrect predictions*



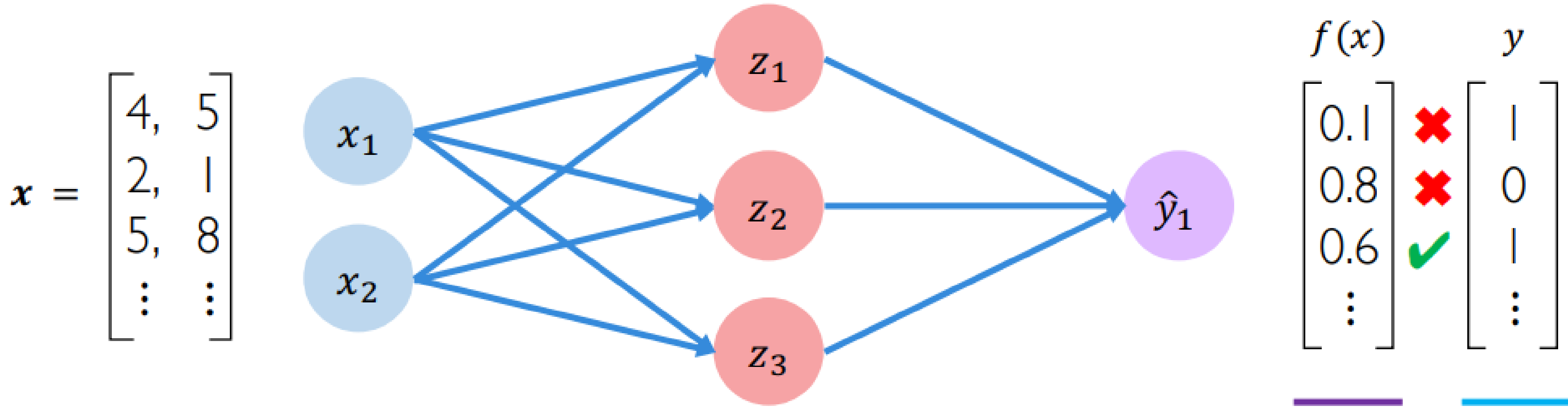
$$\mathcal{L}\left(f\left(x^{(i)};\boldsymbol{\theta}\right), y^{(i)}\right)$$

Predicted Actual

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Empirical Loss

The ***empirical loss*** *measures the total loss over our entire dataset*



Also known as:
- Objective function
- Cost function
- Empirical Risk

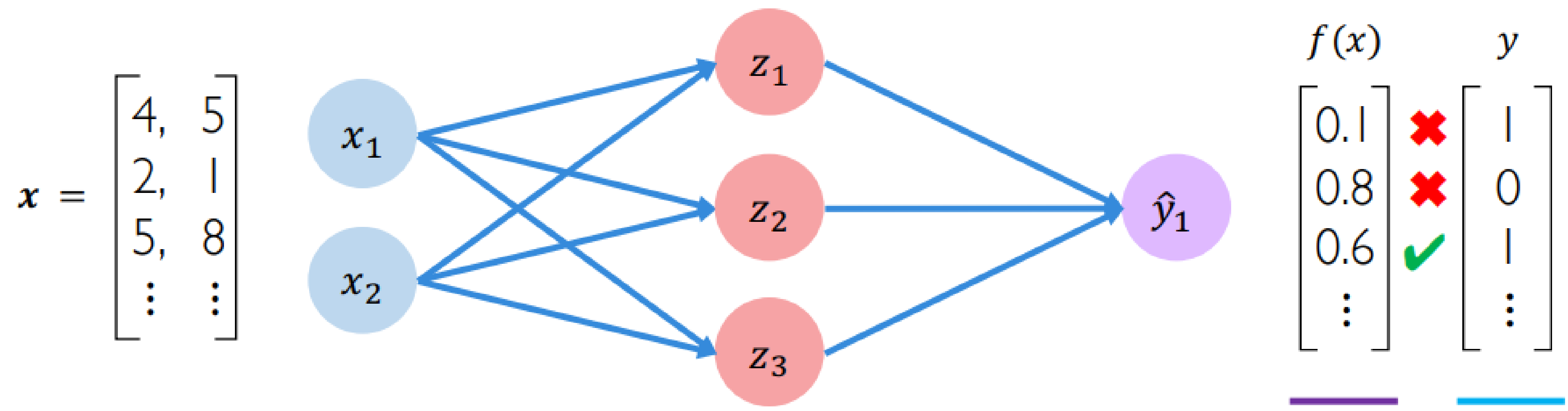
$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x^{(i)};\theta), y^{(i)})$$

Predicted Actual

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*



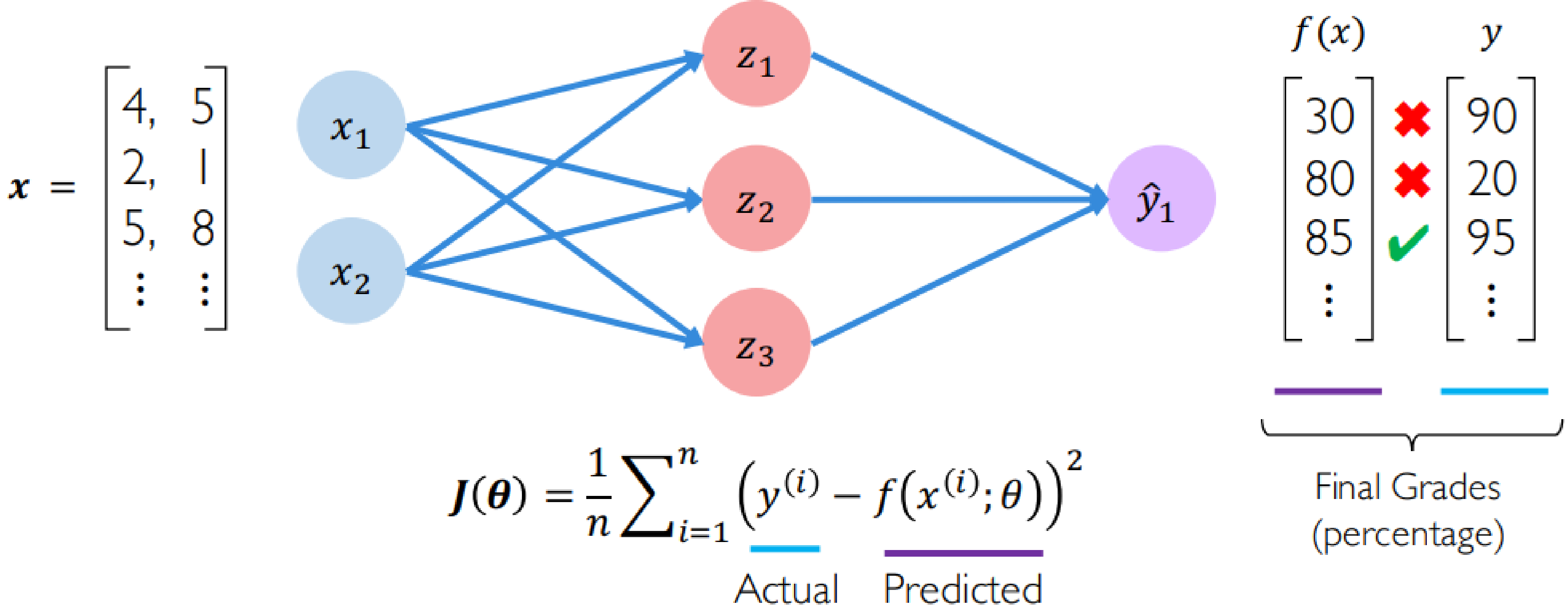
$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\theta\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)};\theta\big)}_{\text{Predicted}}\Big)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred) )
```

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*




```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$$

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

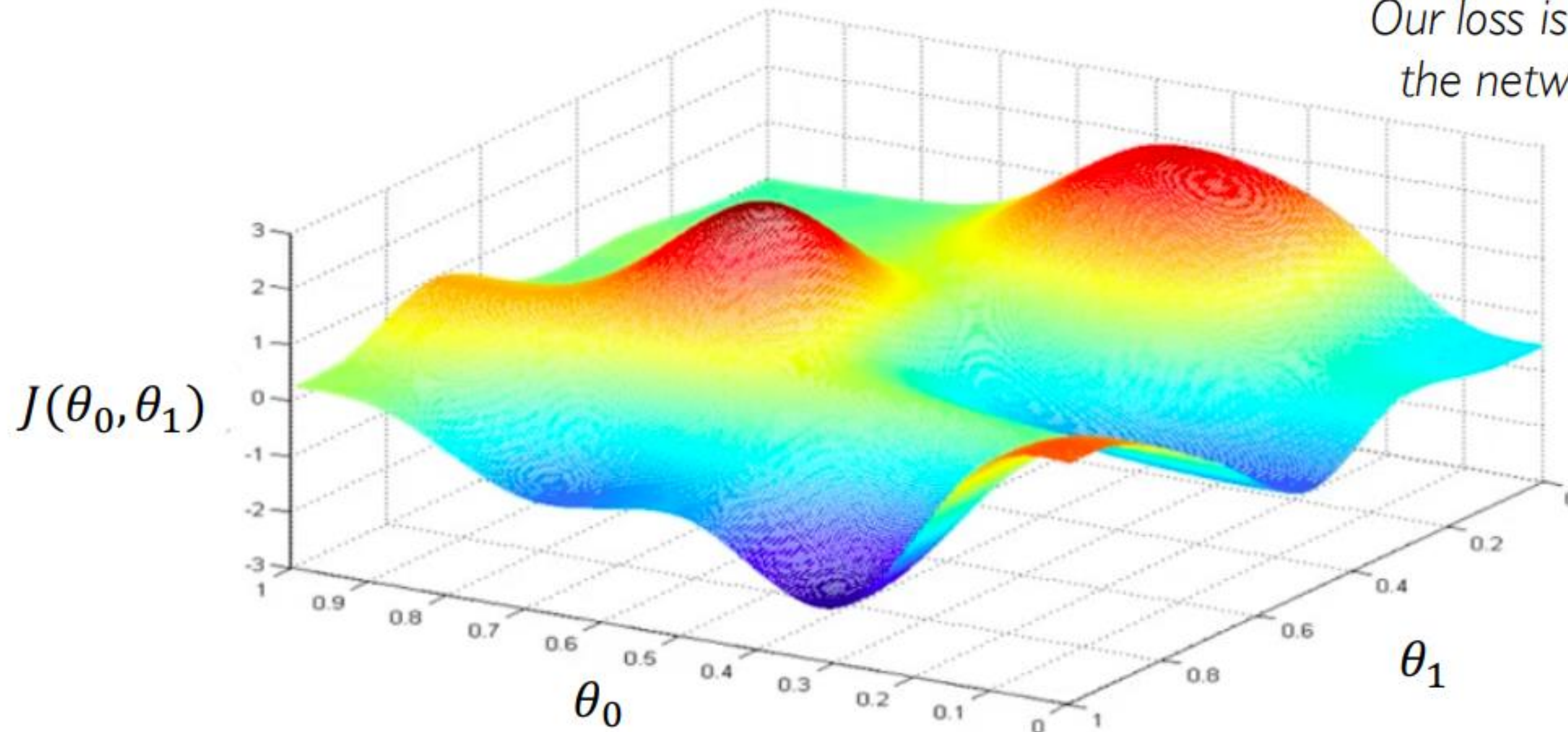
Remember:

$$\boldsymbol{\theta} = \{\theta^{(0)}, \theta^{(1)}, \cdots\}$$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Loss Optimization

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} J(\boldsymbol{\theta})$$

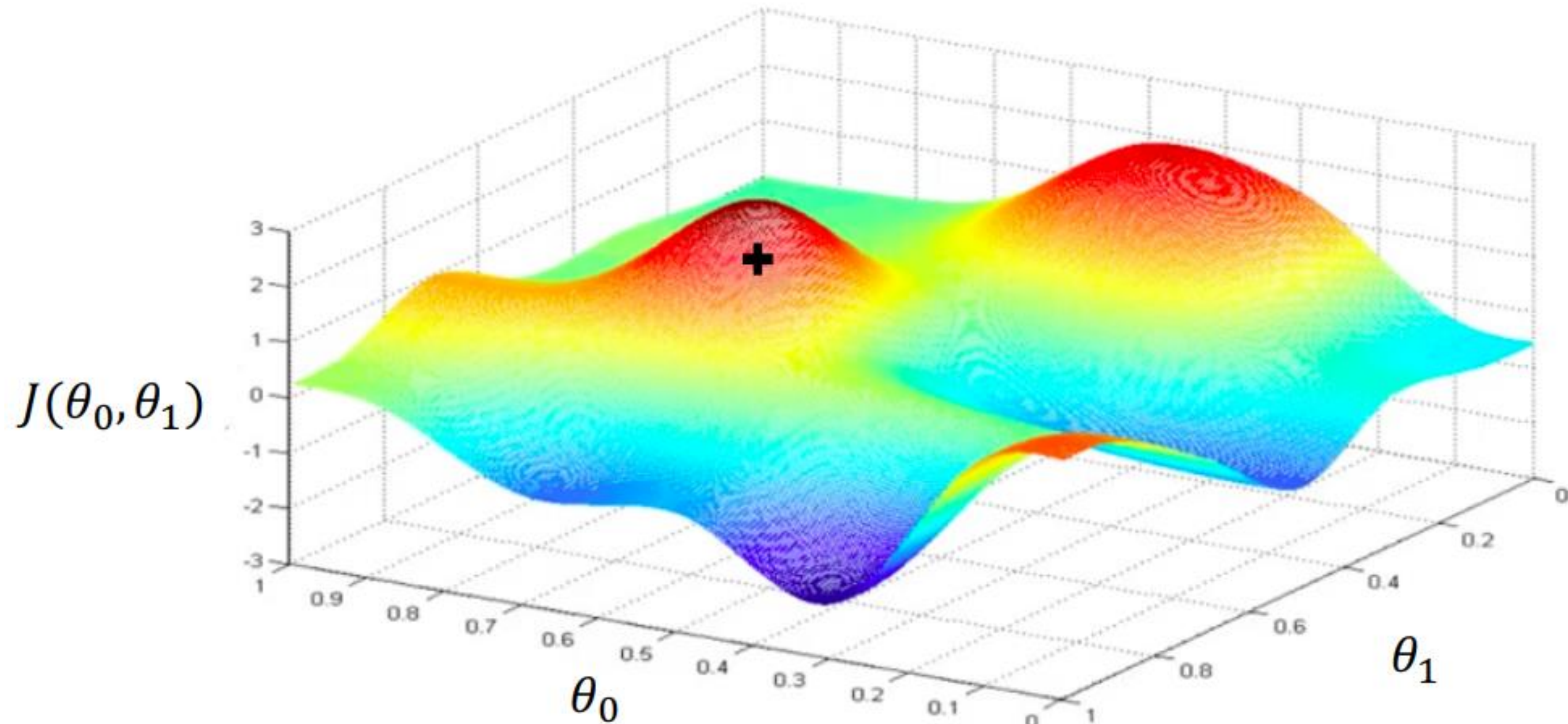


Remember:
Our loss is a function of the network weights!

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

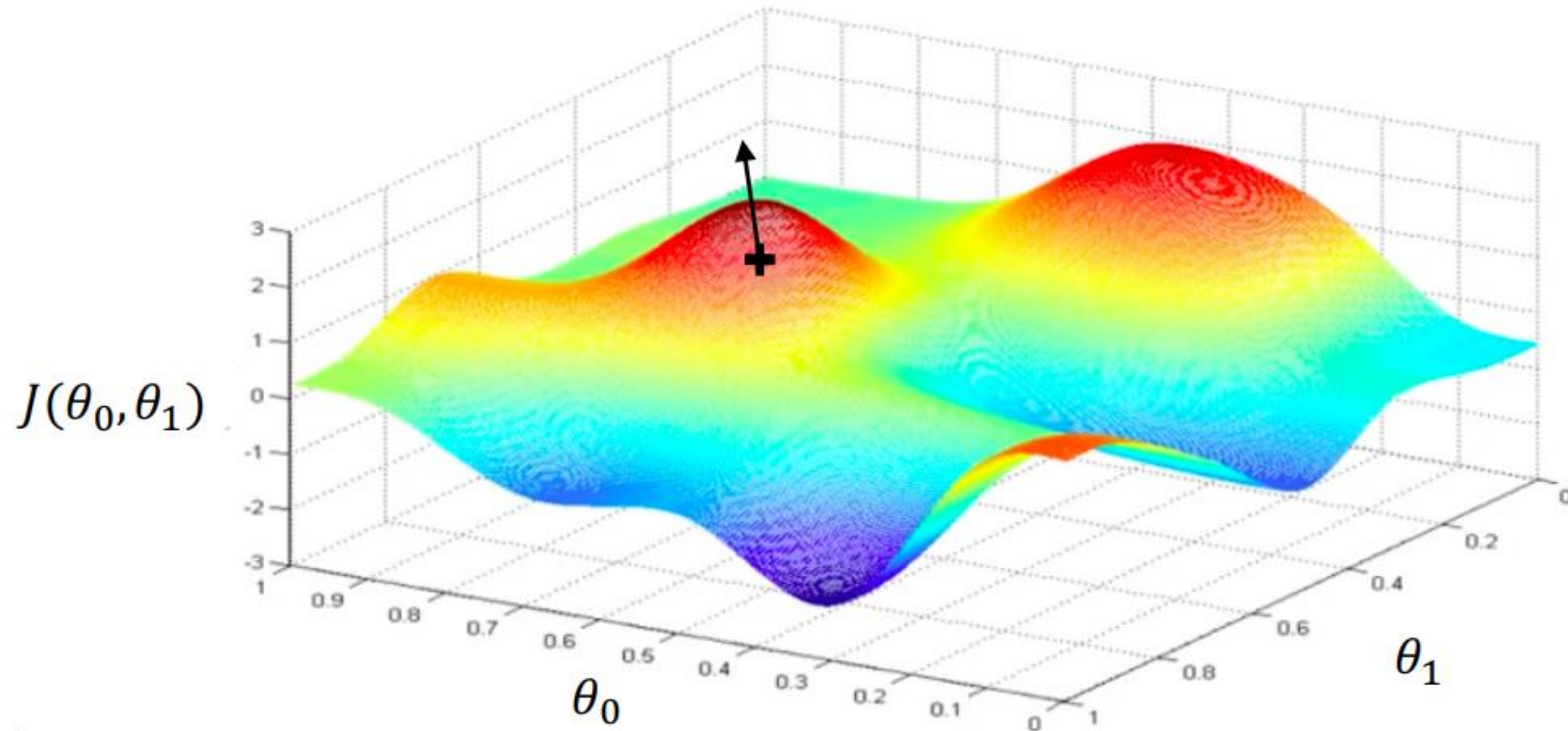
Loss Optimization

Randomly pick an initial (θ_0, θ_1)



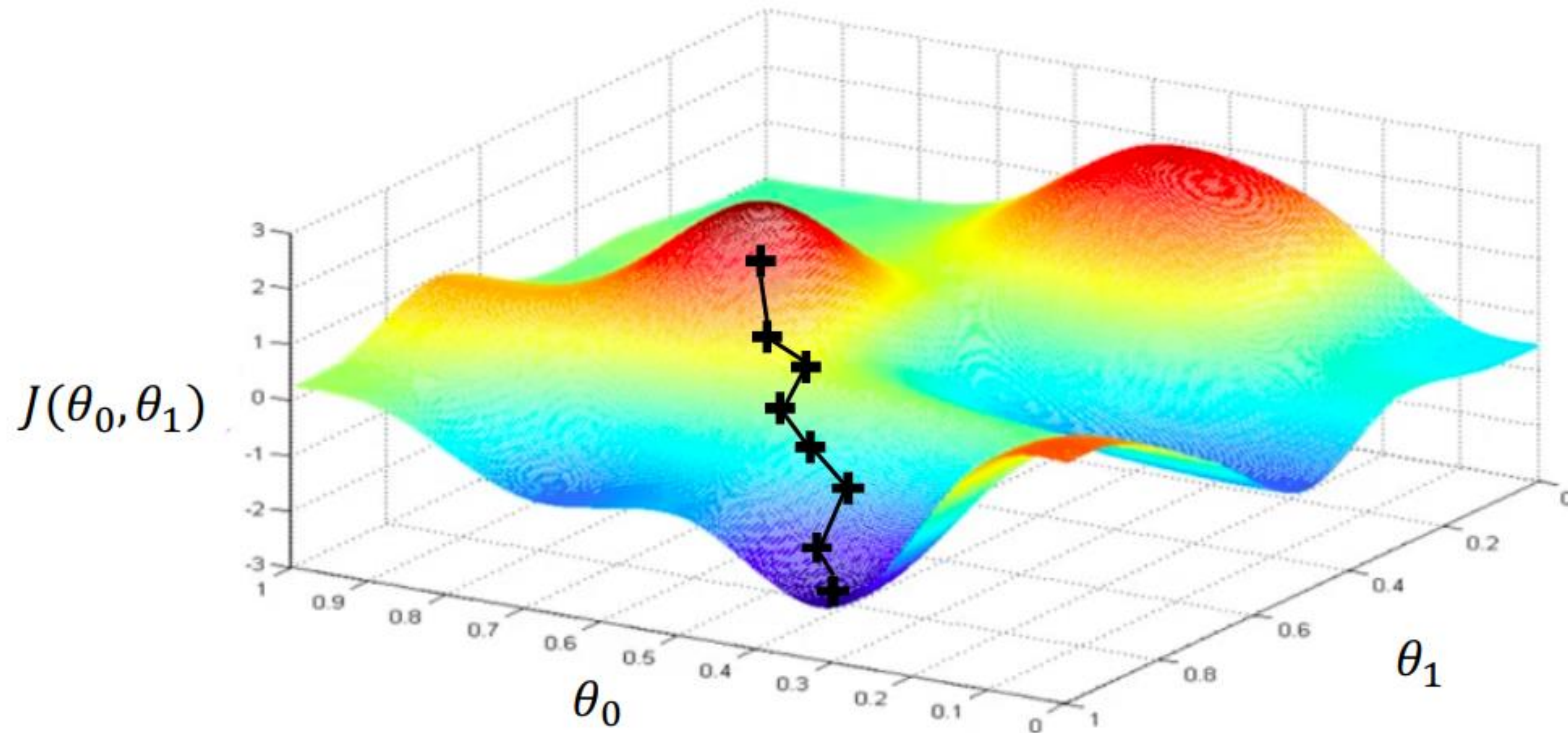
© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Loss Optimization



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Gradient Descent



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\theta)}{\partial \theta}$

4. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$

5. Return weights

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

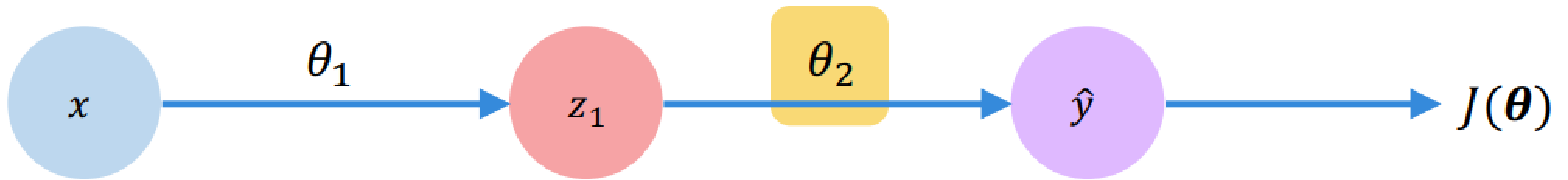
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\theta)}{\partial \theta}$

4. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$

5. Return weights

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

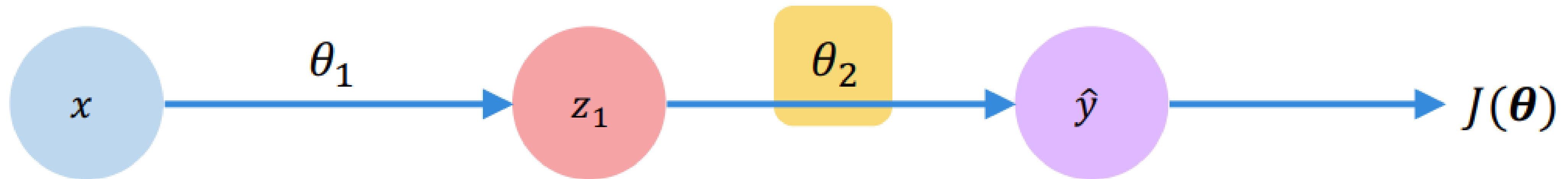
Computing Gradients: Backpropagation



How does a small change in one weight (ex. $\boldsymbol{\theta}_2$) affect the final loss $\boldsymbol{J}(\boldsymbol{\theta})$?

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Computing Gradients: Backpropagation

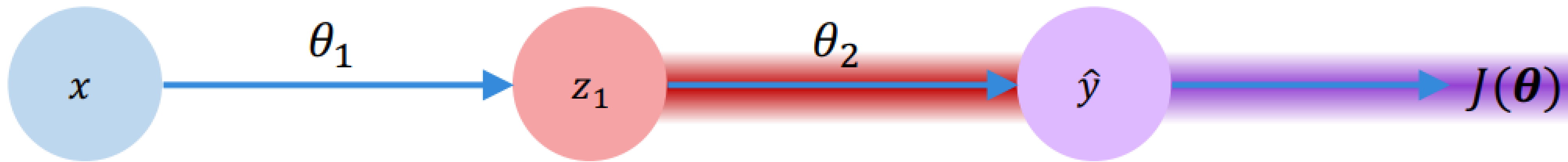


$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} =$$

Let's use the chain rule!

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

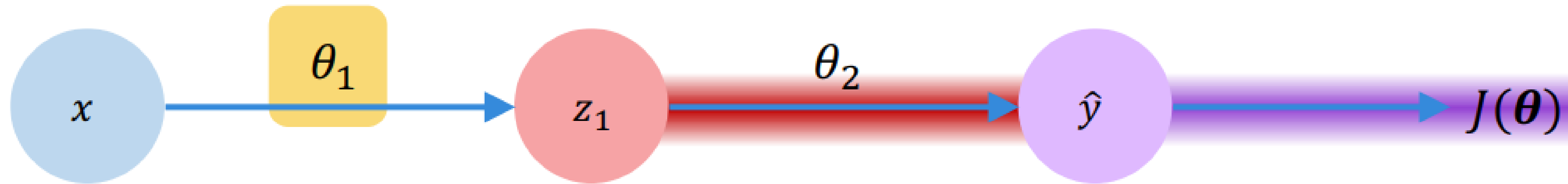
Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \theta_2}$$

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial \theta_1}$$

Apply chain rule!

Apply chain rule!

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Computing Gradients: Backpropagation

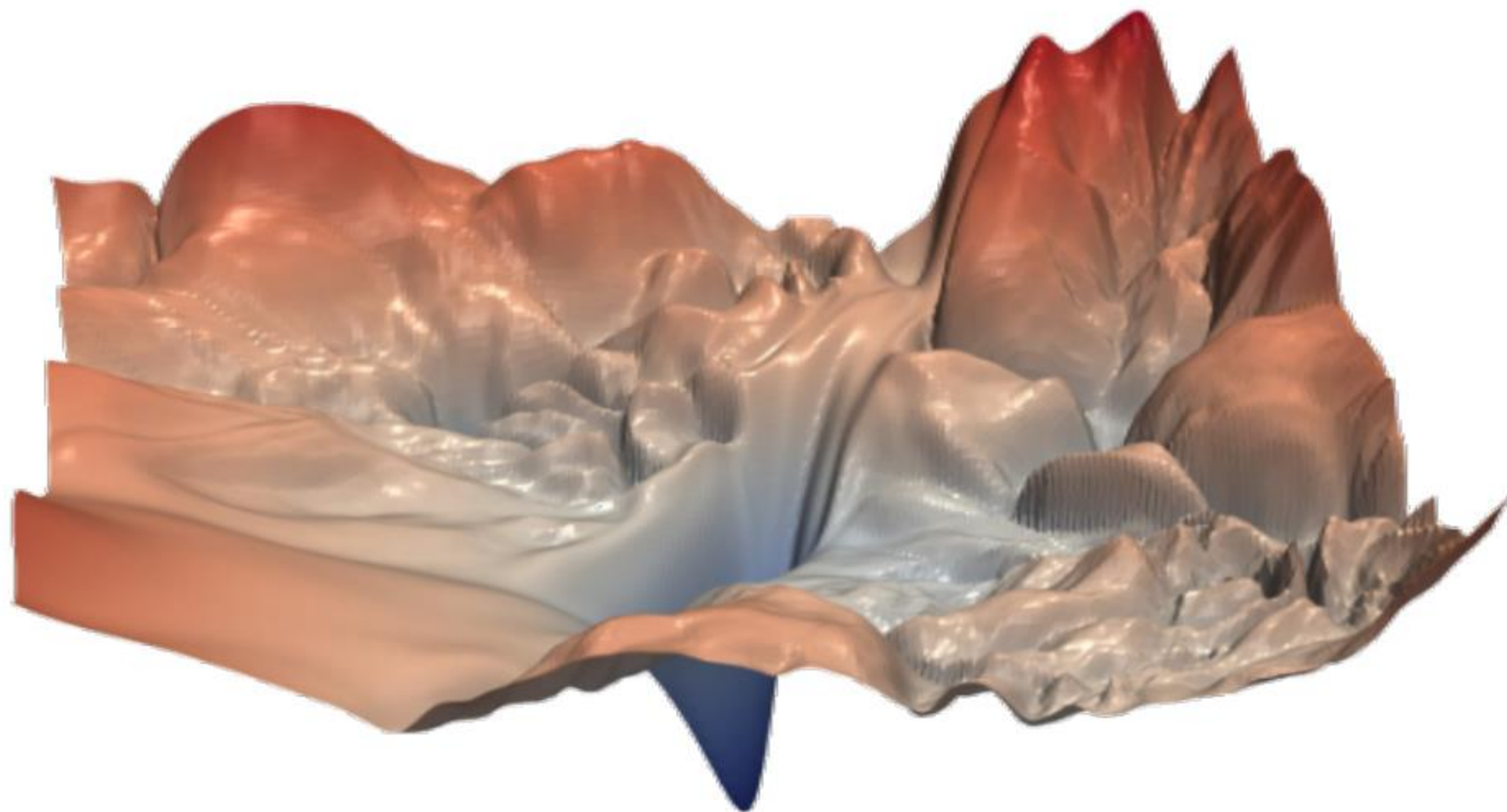
x —θ_1→ z_1 —θ_2→ $\hat{y}$ → $J(\boldsymbol{\theta})$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial \theta_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Training Neural Networks is Difficult



Hao Li, Zheng Xu, Gavin Taylor, Tom Goldstein, Visualizing the Loss Landscape of Neural Nets, 6th International Conference on Learning Representations, ICLR 2018

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Loss functions can be difficult to optimize

Remember:

Optimization through gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

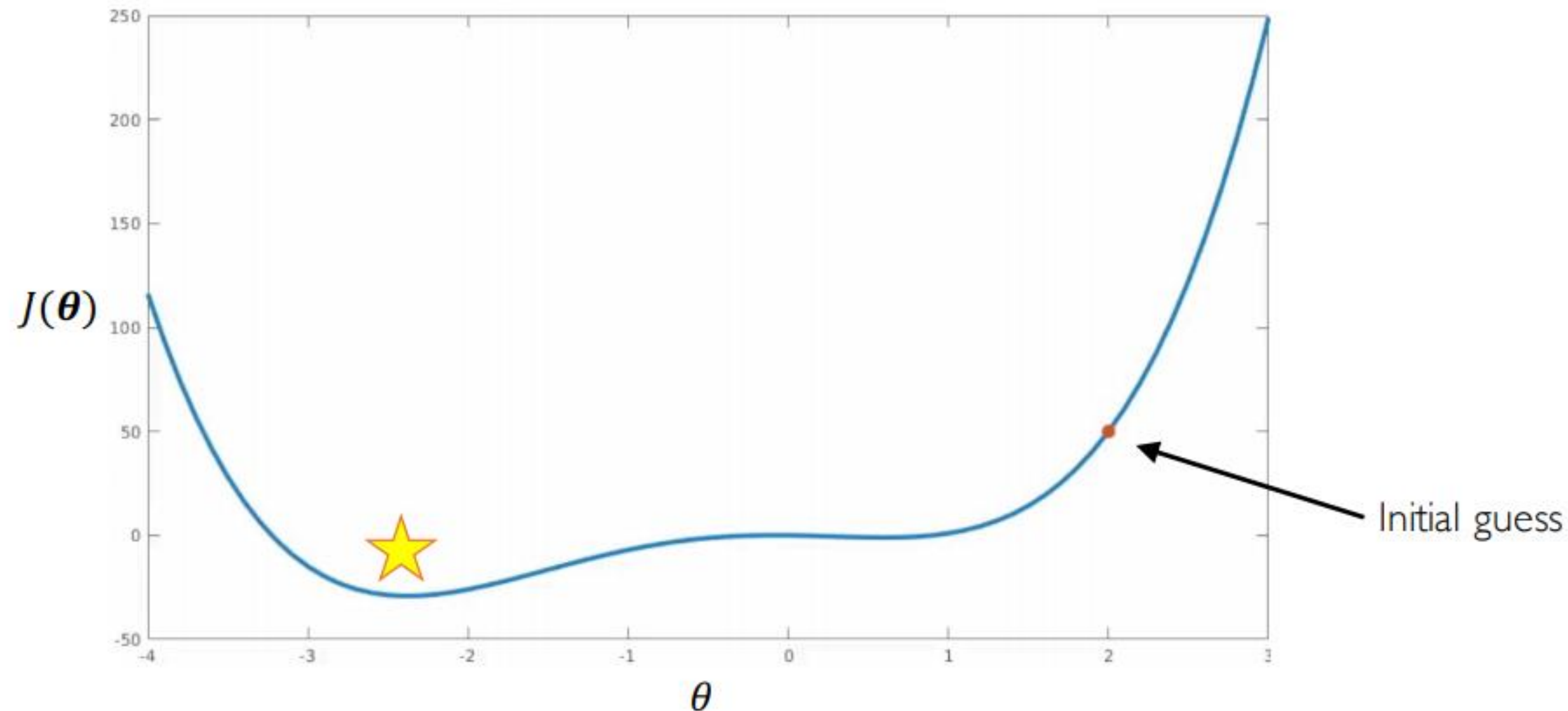
How can we set the learning rate?

Setting the learning rate

Small learning rates
converges slowly and gets stuck in false local minima

Large learning rates
overshoot, become unstable and diverge

Stable learning rates
converge smoothly and avoid local minima



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Adaptive Learning Rates

- Design an adaptive learning rate that adapts to the landscape

- Learning rates are no longer fixed

- Can be made larger or smaller depending on:
    - How large the gradient is
    - How fast learning is happening
    - Etc..

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Adaptive Learning Rate Algorithms

- Momentum
- Adagrad
- Adadelta
- Adam
- RMSProp

Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

Hinton's Coursera lecture (unpublished)

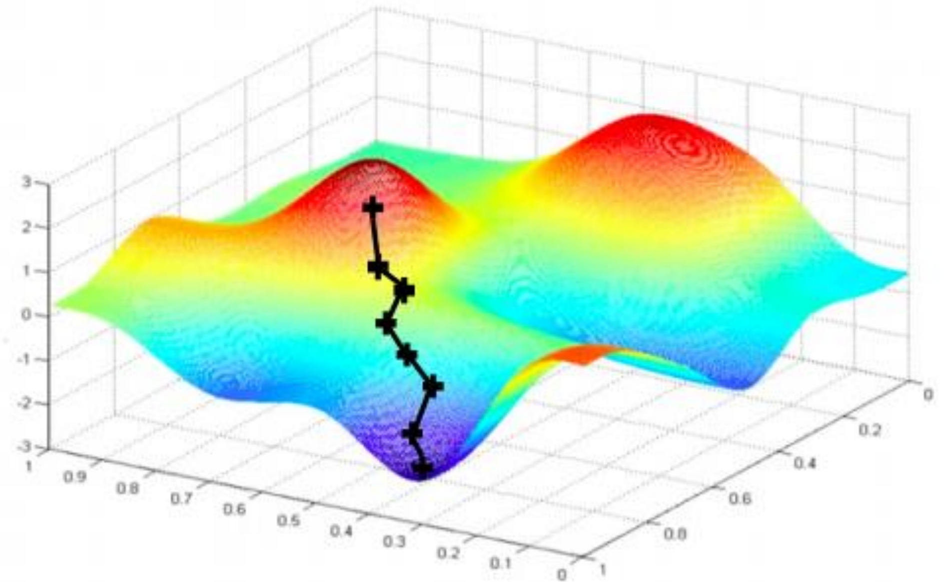
http://ruder.io/optimizing-gradient-descent/

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\theta)}{\partial \theta}$
4. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$
5. Return weights

Can be very computational to compute!



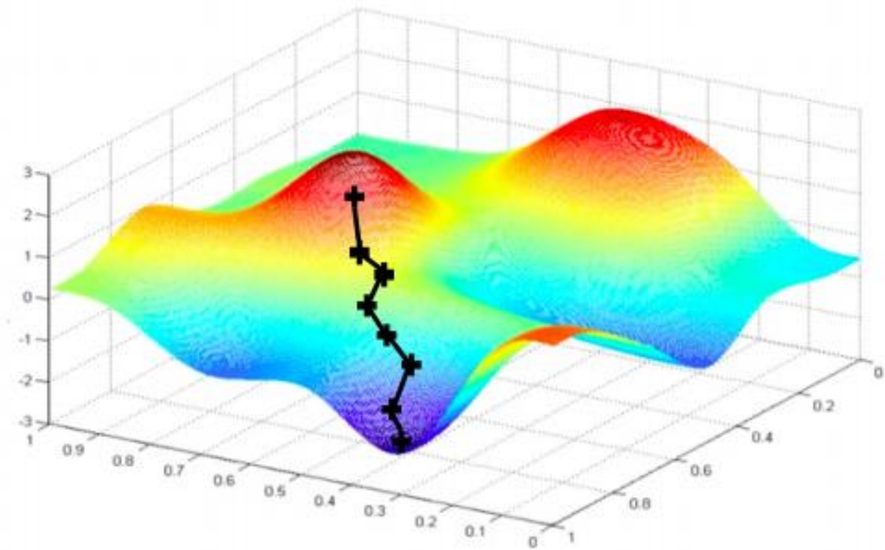
© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\theta)}{\partial \theta}$
5. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$
6. Return weights

Easy to compute but **very noisy** (stochastic)!

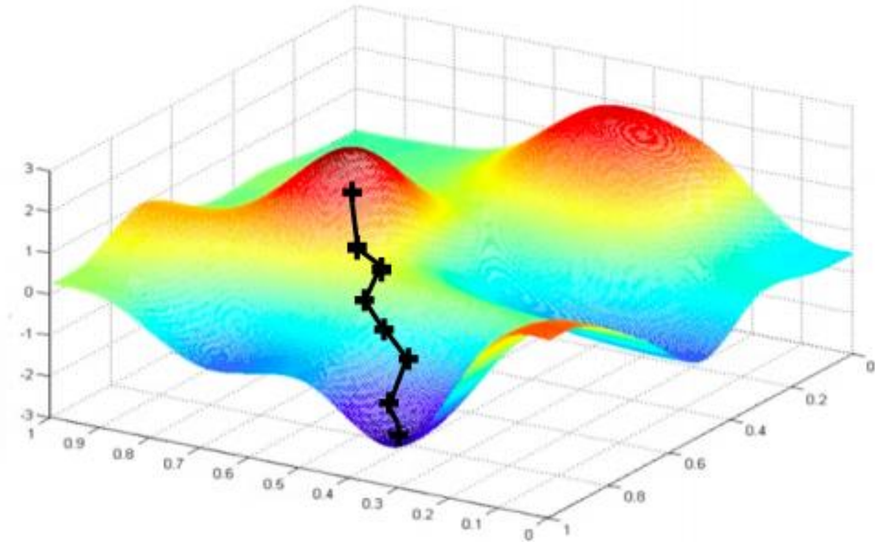


© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\theta)}{\partial \theta}$
5. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Mini-batches

- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Mini-batches lead to fast training
  - Can parallelize computation + achieve significant speed increases on GPU's

© MIT 6.S191: Introduction to Deep Learning

Terminology

- **Number of iterations**: The number of times the gradient is estimated and the parameters of the neural network are updated using a batch of training instances
- **Batch size:** Number of training instances used in one iteration
- **Mini-batch:** When the total number of training instances N is large, a small number of training instances B<<N which constitute a mini-batch can be used in one iteration to estimate the gradient of the loss function and update the parameters of the network
- **Epoch:** It takes n = N/B iterations to use the entire training data once. That is called an epoch. The total number of times the parameters get updates is (N/B)*E, where E is the number of epochs.

https://www.quora.com/What-are-the-meanings-of-batch-size-mini-batch-iterations-and-epoch-in-neural-networks

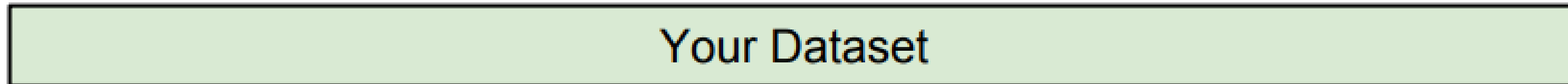
Three modes of gradient descent

- Batch mode: $N=B$, one epoch is same as one iteration.
- Mini-batch mode: $1<B<N$, one epoch consists of N/B iterations.
- Stochastic mode: $B=1$, one epoch takes N iterations.

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: always works perfectly on training data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

Setting Hyperparameters

Your Dataset

Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results

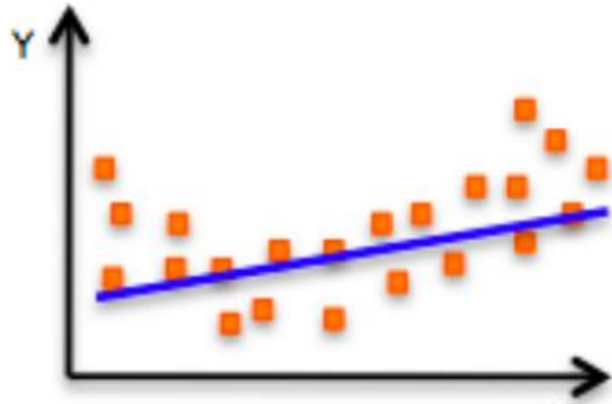
fold 1	fold 2	fold 3	fold 4	fold 5	test

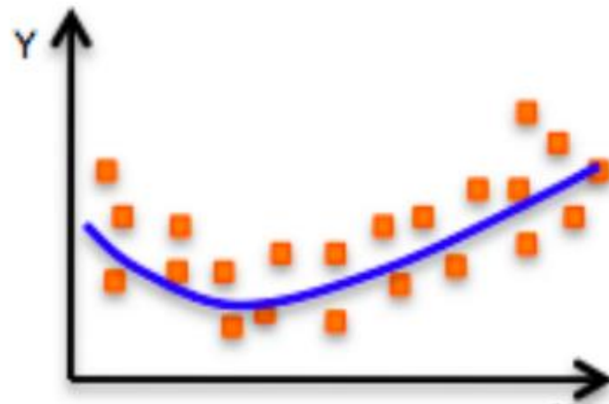
fold 1	fold 2	fold 3	fold 4	fold 5	test

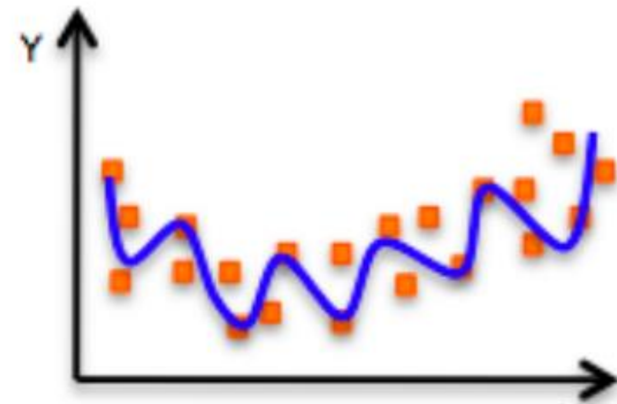
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but not used too frequently in deep learning

The Problem of Overfitting







Underfitting
Model does not have capacity to fully learn the data

High bias

← **Ideal fit** →

Overfitting
Too complex, extra parameters, does not generalize well

High variance

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

High Bias vs High Variance

- High Bias (high training set error)
  - Use a bigger network
  - Try different optimization algorithms
  - Train longer
  - Try different architecture

- High Variance (high validation set error)
  - Collect more data
  - Use regularization
  - Try different NN architecture

Regularization

- What is it?
  - Technique that constrains our optimization problem to discourage complex models
- Why do we need it?
  - Improve generalization of our model on unseen data

© MIT 6.S191: Introduction to Deep Learning

Regularization 1: Penalizing weights

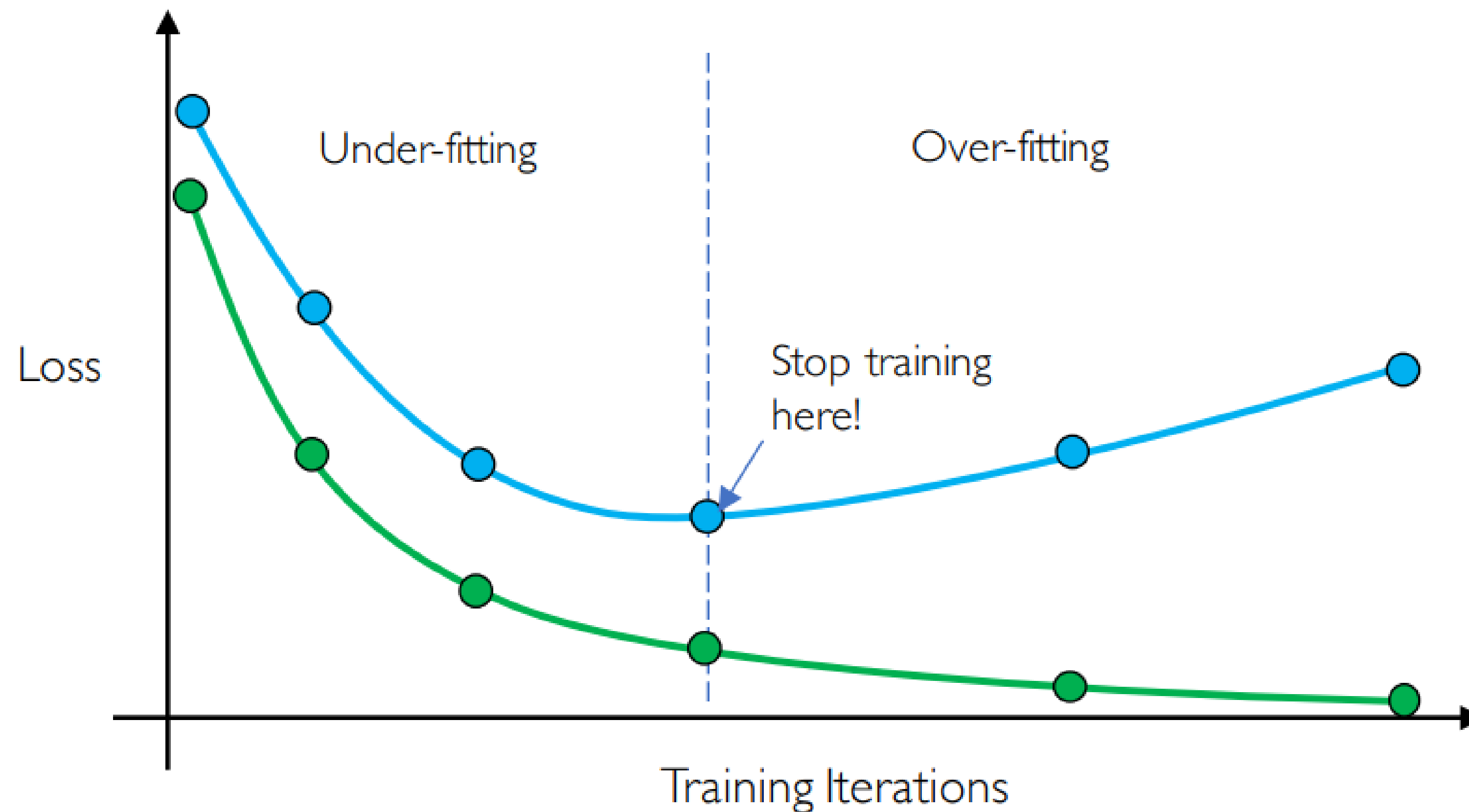
- Penalize large weights using penalties: constraints on their squared values (L2 penalty) or absolute values (L1 penalty)

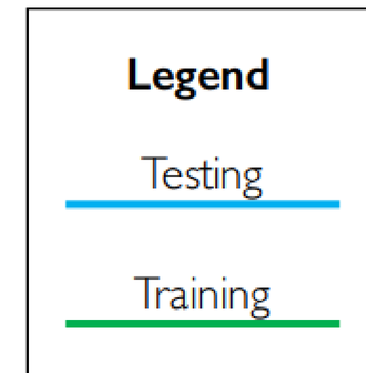
- Neural networks have thousands (or millions of parameters)
  - Danger of overfitting

$$\theta^* \leftarrow \arg\min_{\theta} \sum_{(x,y)\subseteq(X,Y)} \ell(y, a_L(x;\theta_{1,\ldots,L})) + \lambda\Omega(\theta)$$

Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

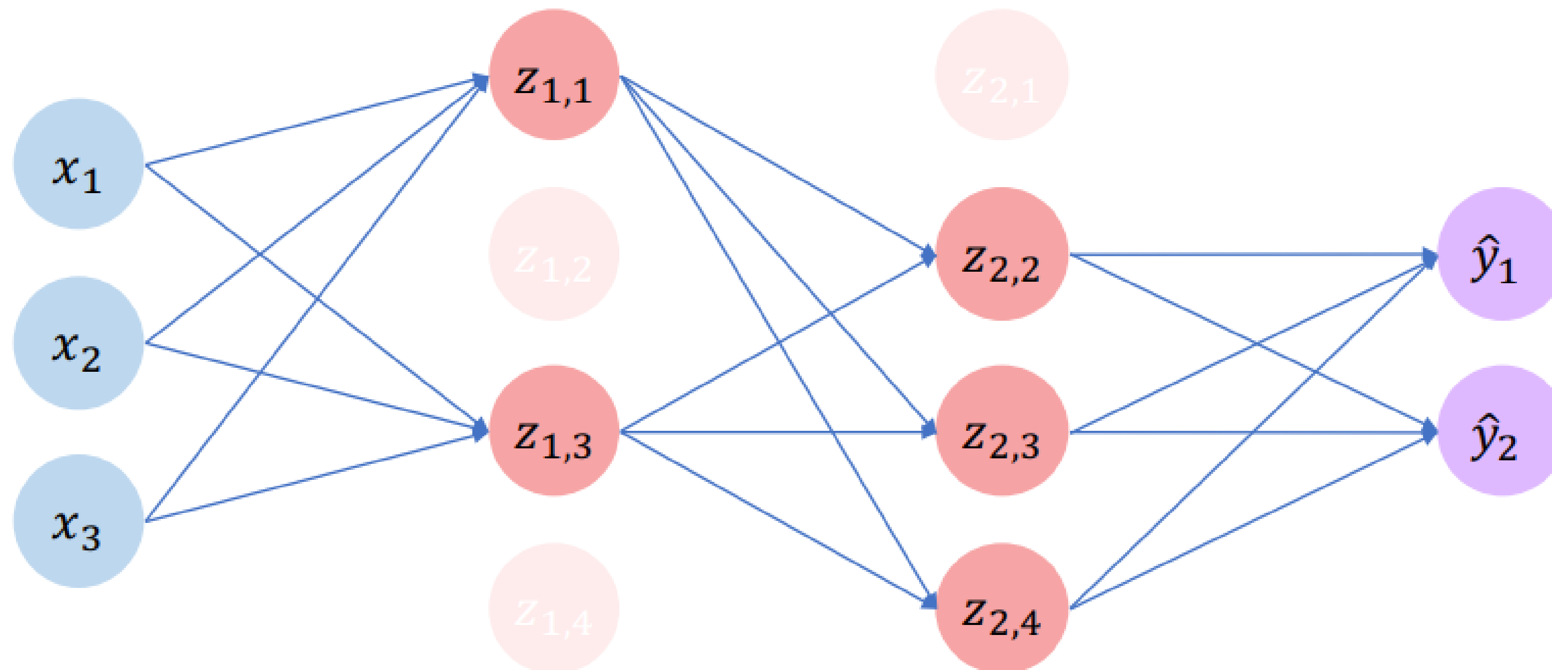




Regularization 3: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

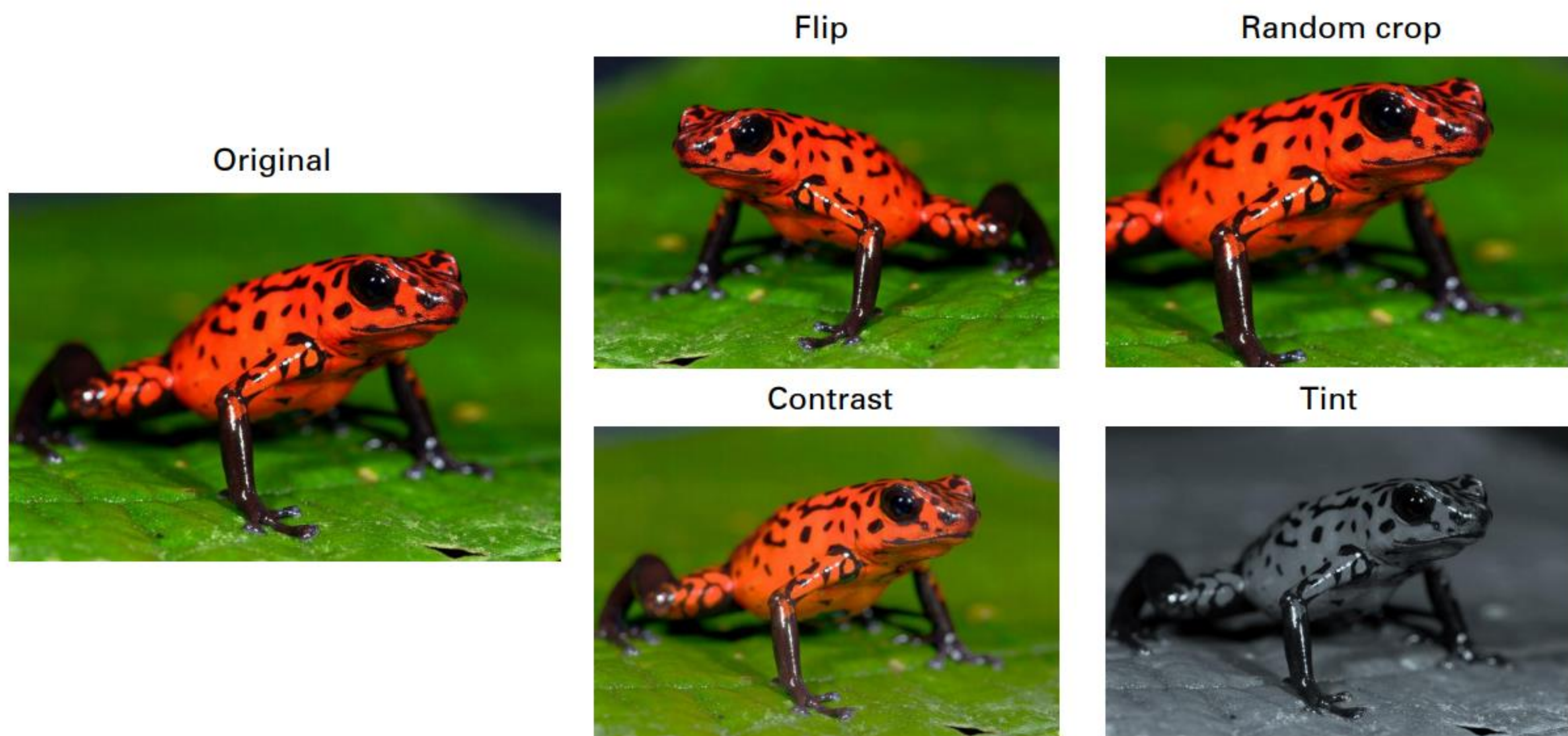
```
tf.nn.dropout(hiddenLayer, p=0.5)
```



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Regularization 4: Data Augmentation

- Adding more data reduces overfitting
- Data collection and labelling is expensive
- Solution: Synthetically increase training dataset



Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, 2012

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

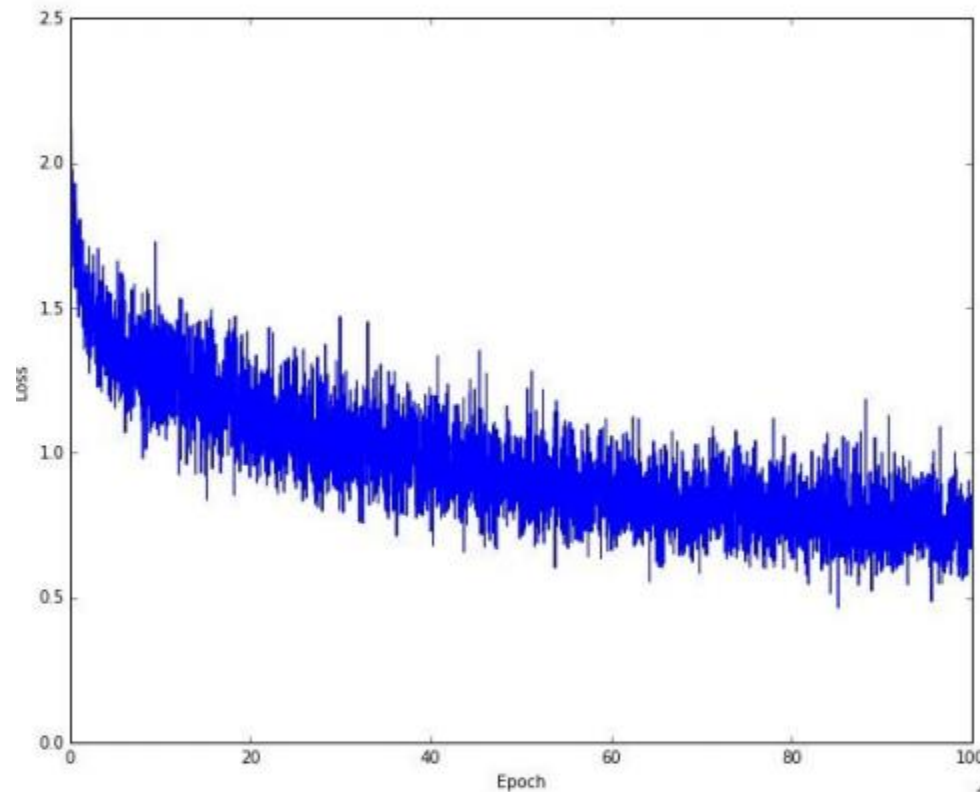
Hyperparameters tuning

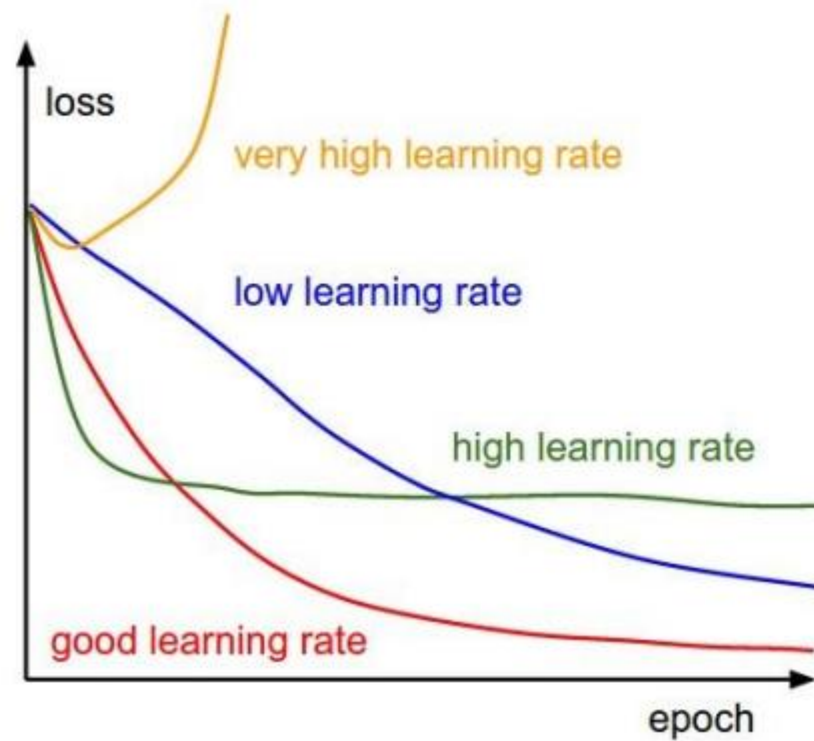
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

neural networks practitioner
music = loss function

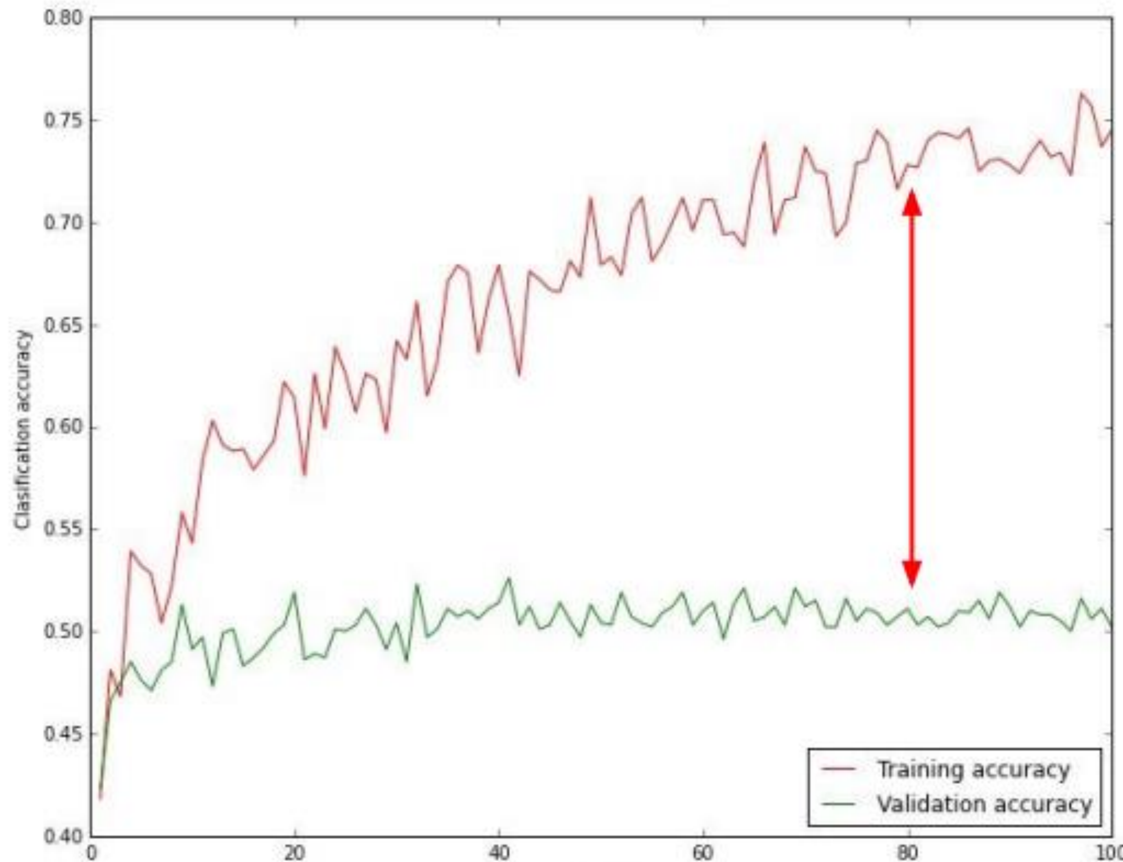
This image by Paolo Guereta is licensed under CC-BY 2.0

Monitor and visualize the loss curve





Monitor and visualize the accuracy



big gap = overfitting

=> increase regularization strength?

no gap

=> increase model capacity?

Convolutional Neural Networks (CNNs):

Input Image

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel Representation

classification

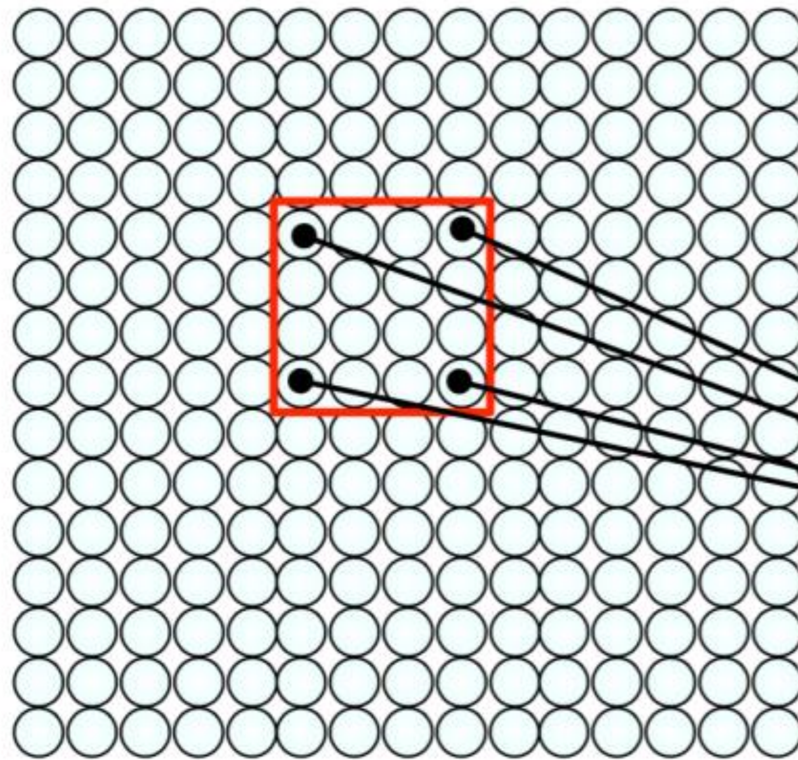
Lincoln	0.8
Washington	0.1
Jefferson	0.05
Obama	0.05

- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Using Spatial Structure

Input: 2D image.
Array of pixel values

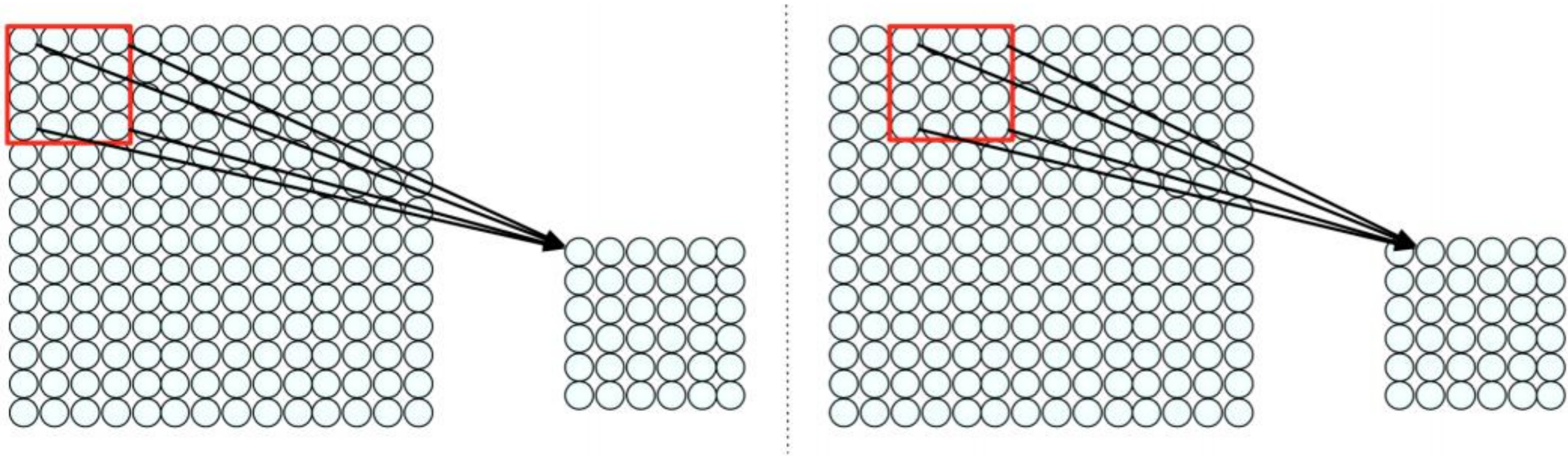


Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Using Spatial Structure



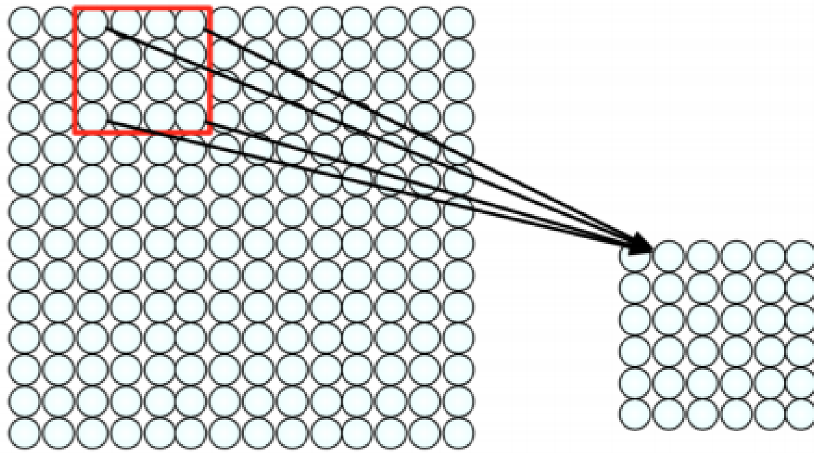
Connect patch in input layer to a single neuron in subsequent layer.

Use a sliding window to define connections.

How can we **weight** *the patch to detect particular features?*

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

© MIT 6.S191: Introduction to Deep Learning

Feature Extraction and Convolution

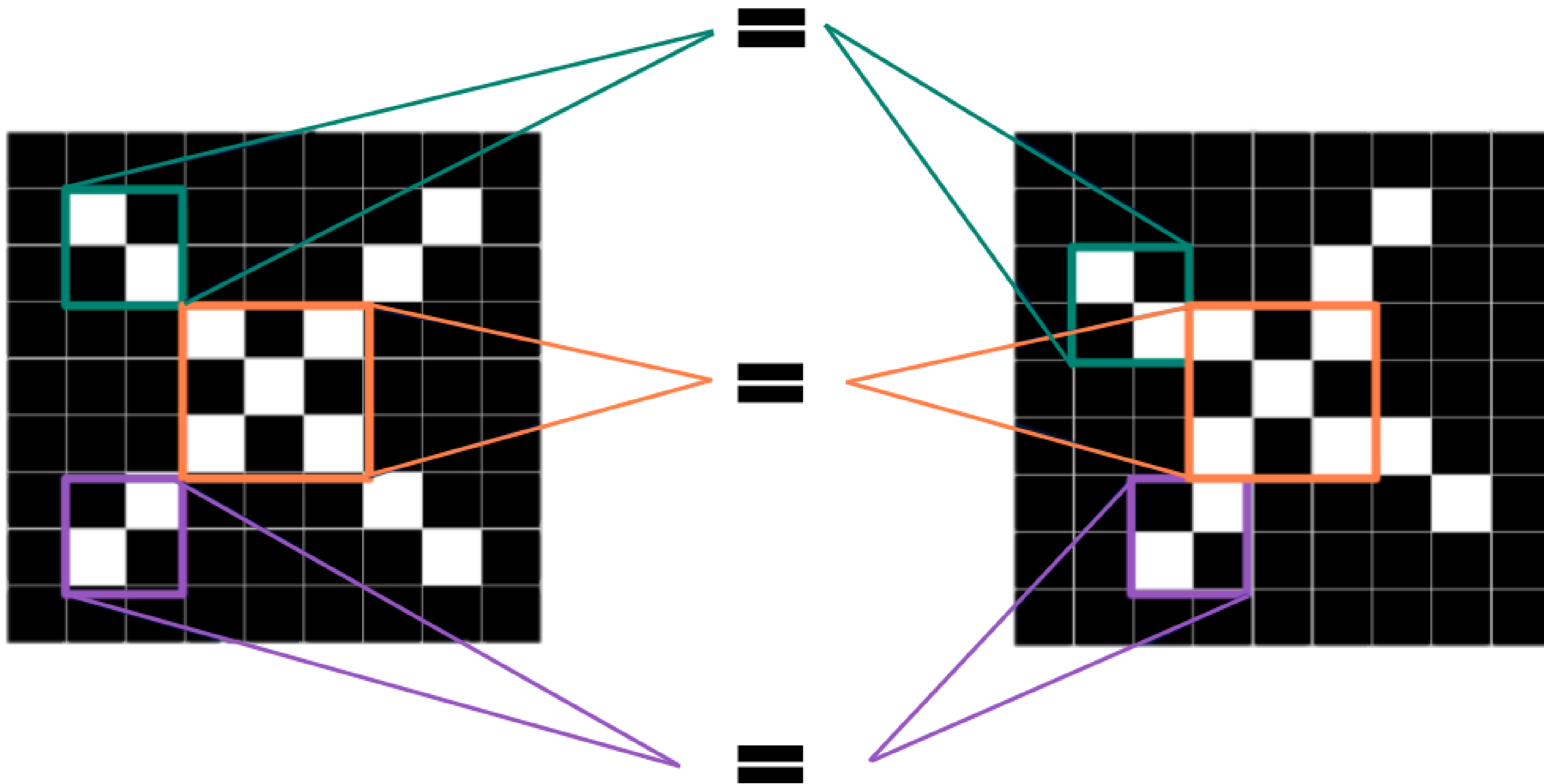
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	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	-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	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image is represented as matrix of pixel values... and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Features of X



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

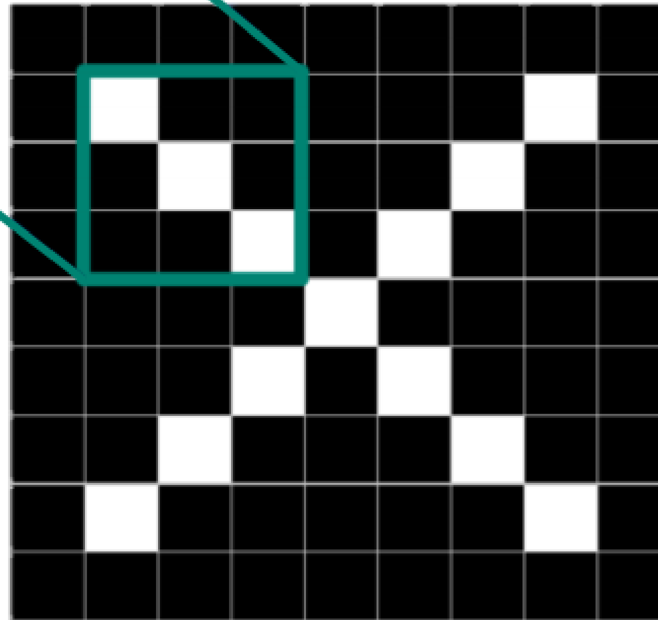
Filters to Detect X Features

filters

1	-1	-1
-1	1	-1
-1	-1	1

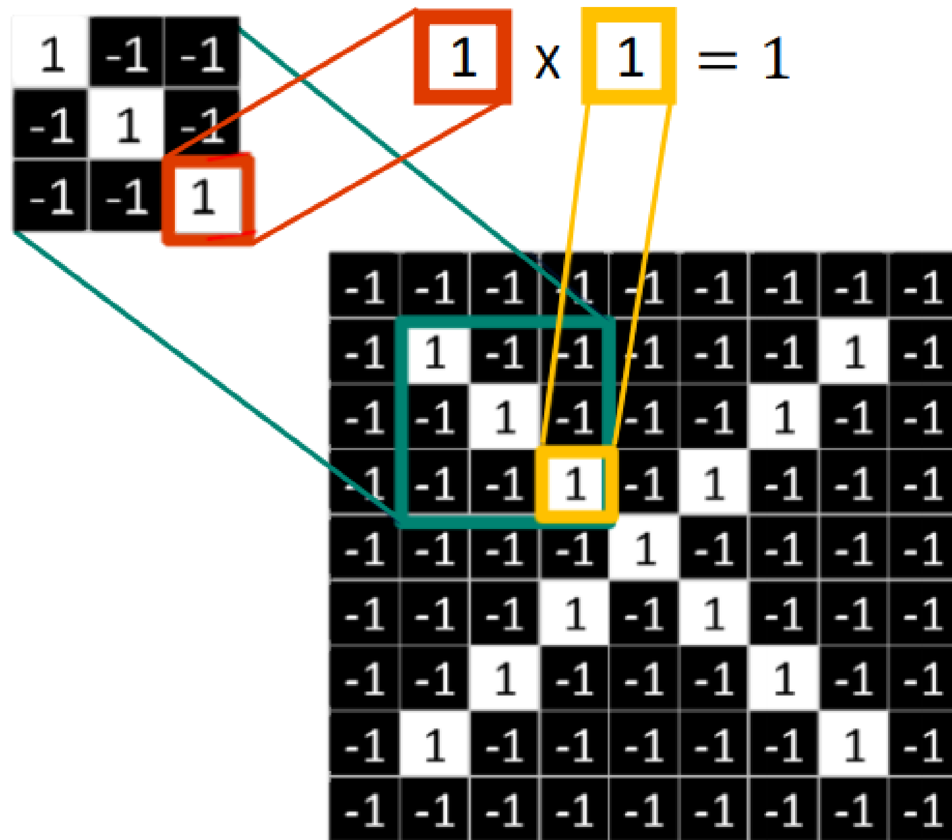
1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1

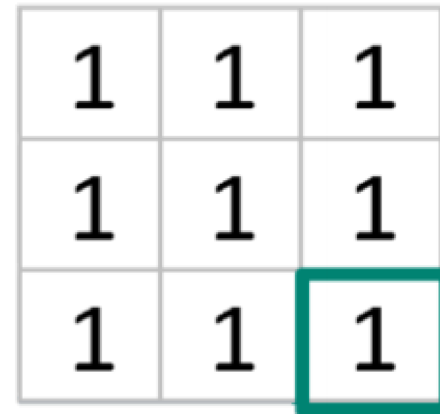


© MIT 6.S191: Introduction to Deep Learning

The Convolution Operation



$\odot$



= 9

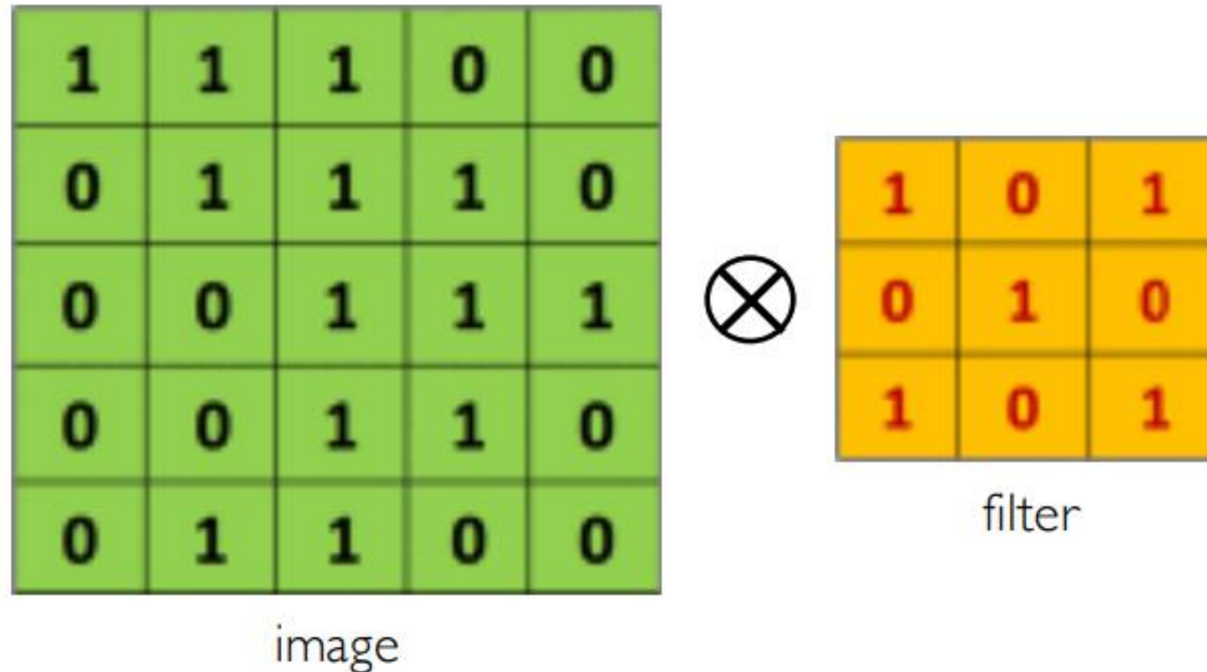
Element wise multiply

Add outputs

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

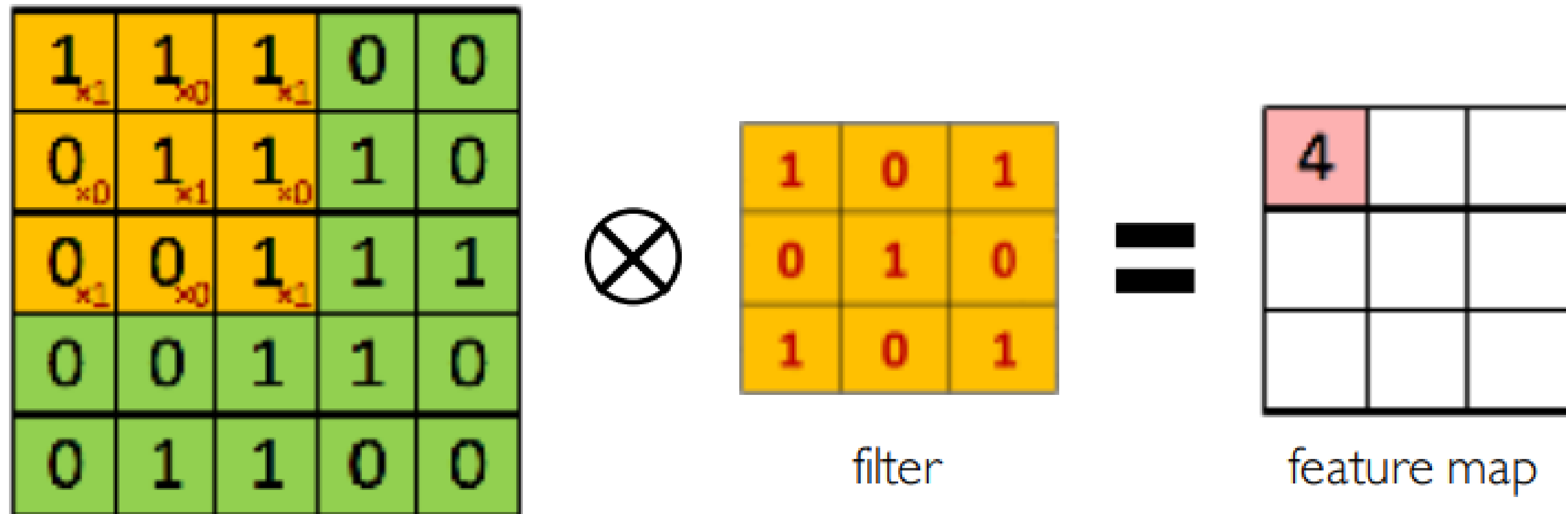


We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

The Convolution Operation

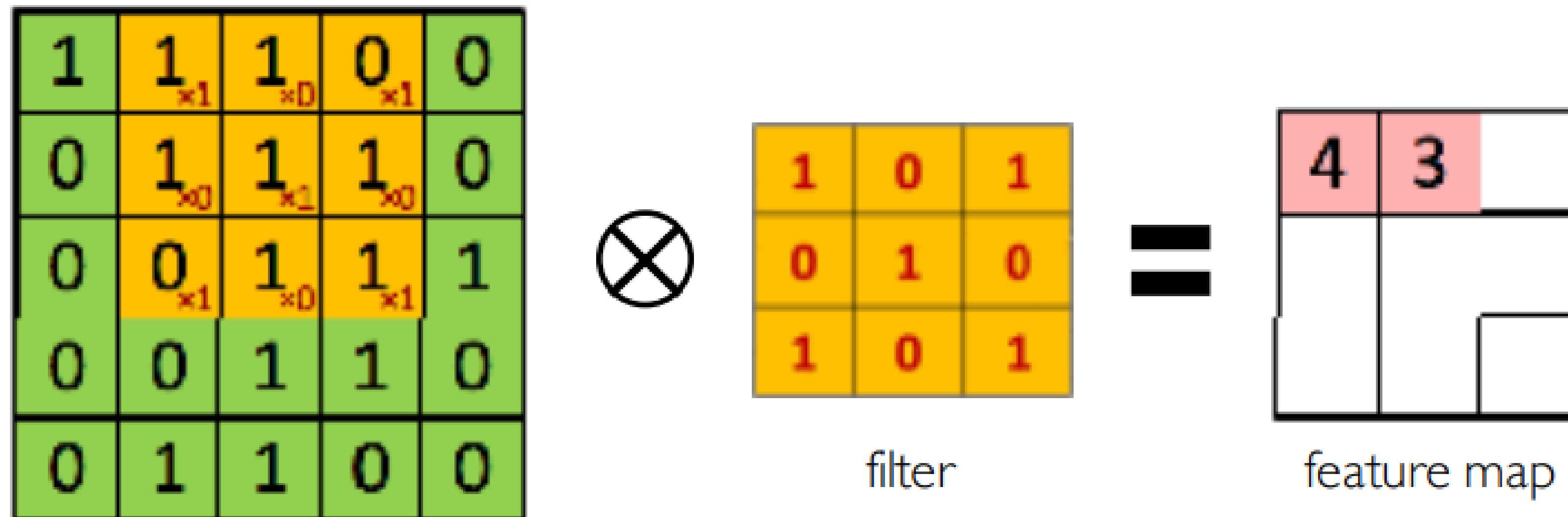
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

The Convolution Operation

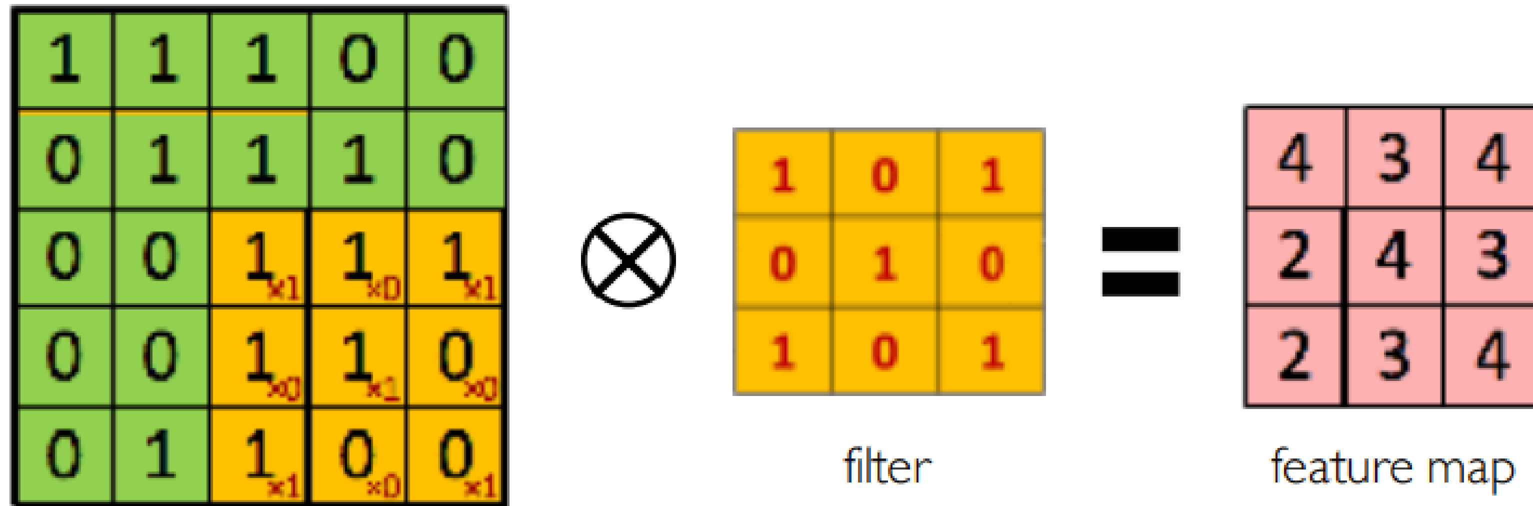
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Producing Feature Maps

Original

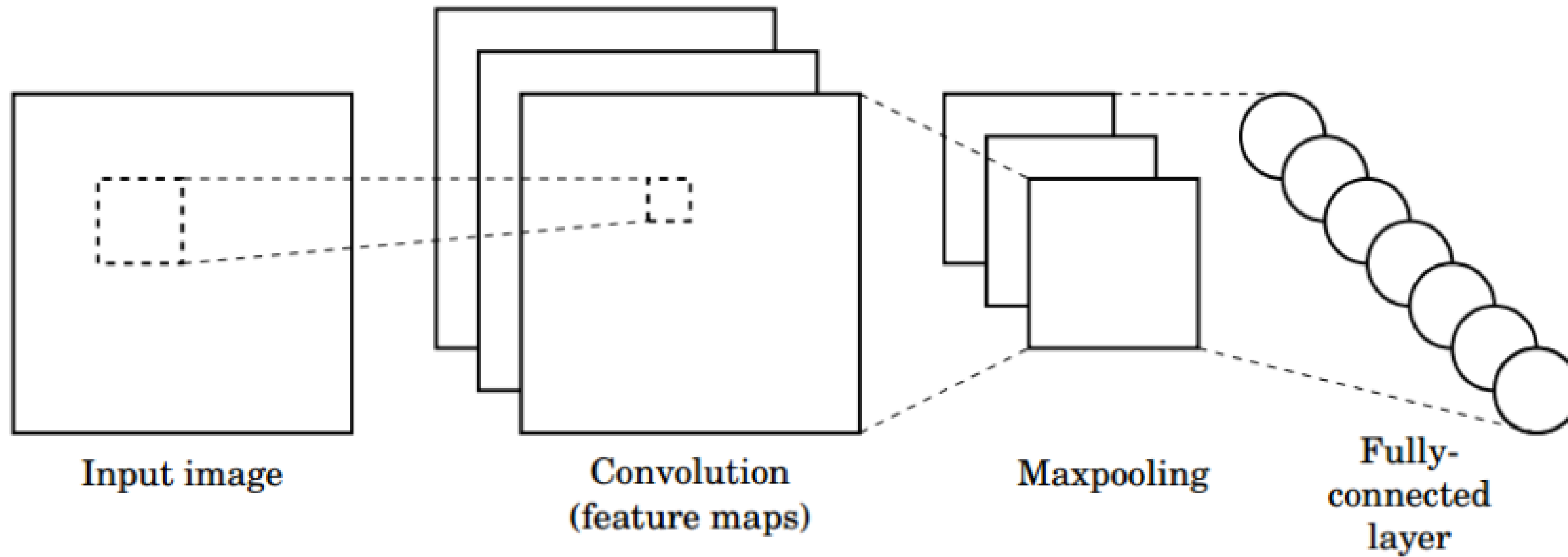
Sharpen

Edge Detect

"Strong" Edge Detect

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

CNNs for Classification



1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

Train model with image data.
Learn weights of filters in convolutional layers.

Pooling

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool with 2x2 filters
and stride 2

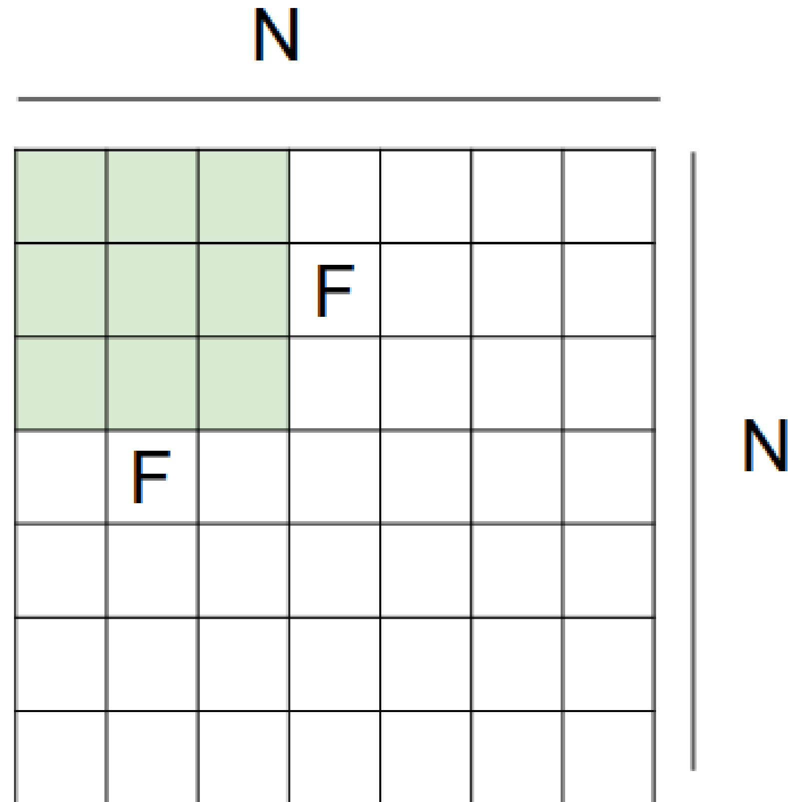
6	8
3	4

1) Reduced dimensionality
2) Spatial invariance

How else can we downsample and preserve spatial invariance?

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

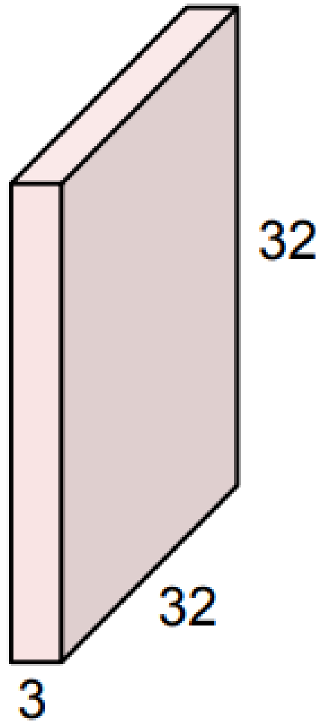
Stride

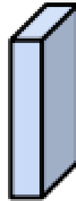


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

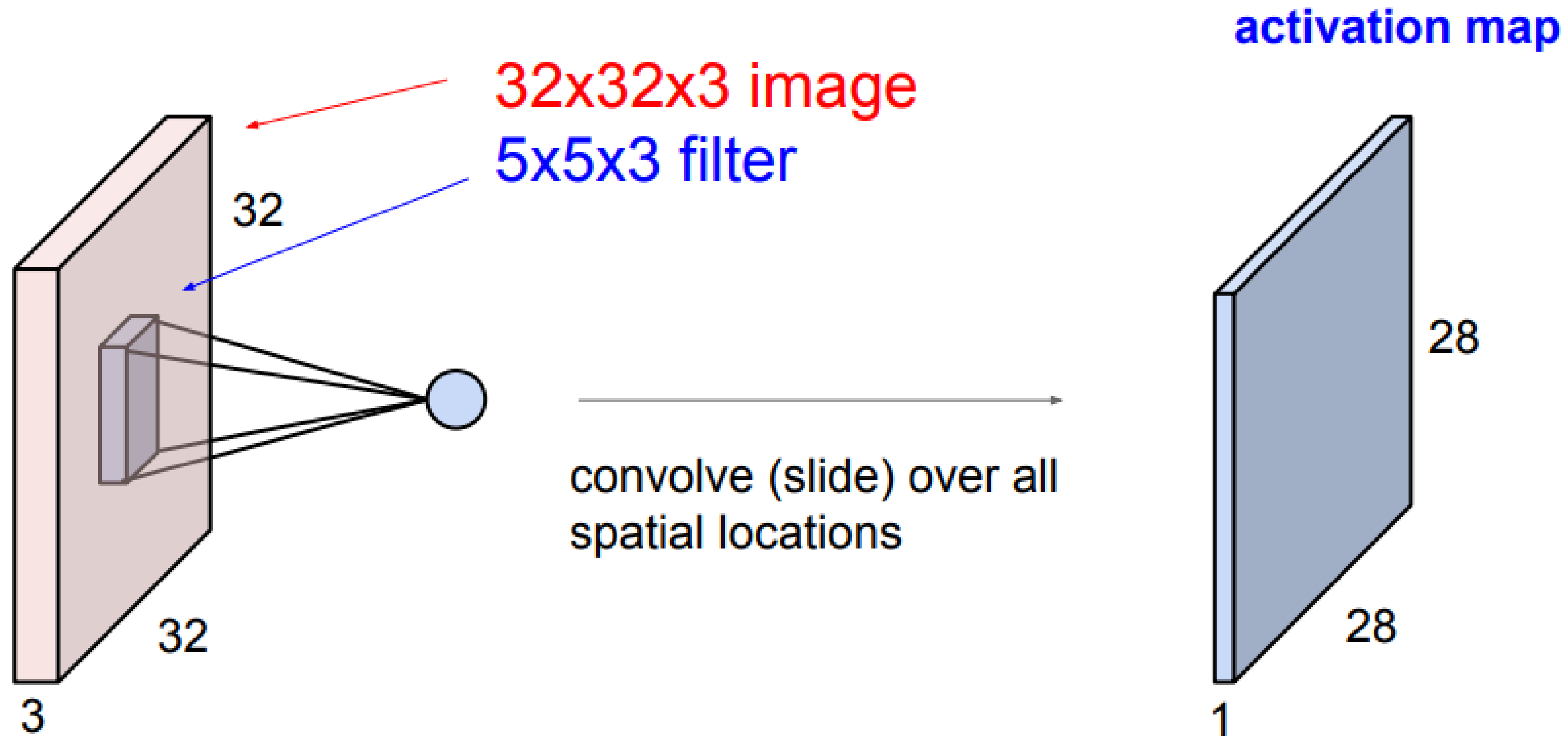
Convolution Layer



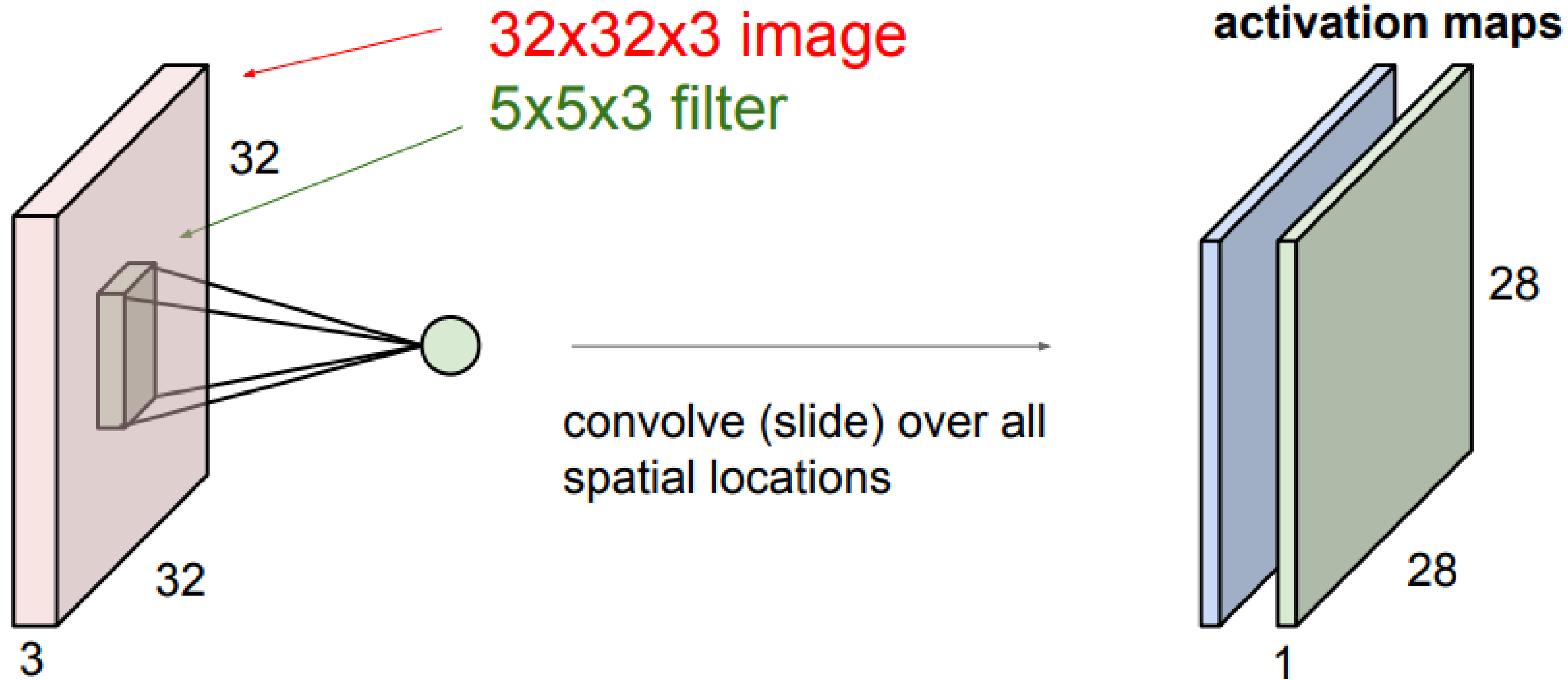


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Convolution Layer

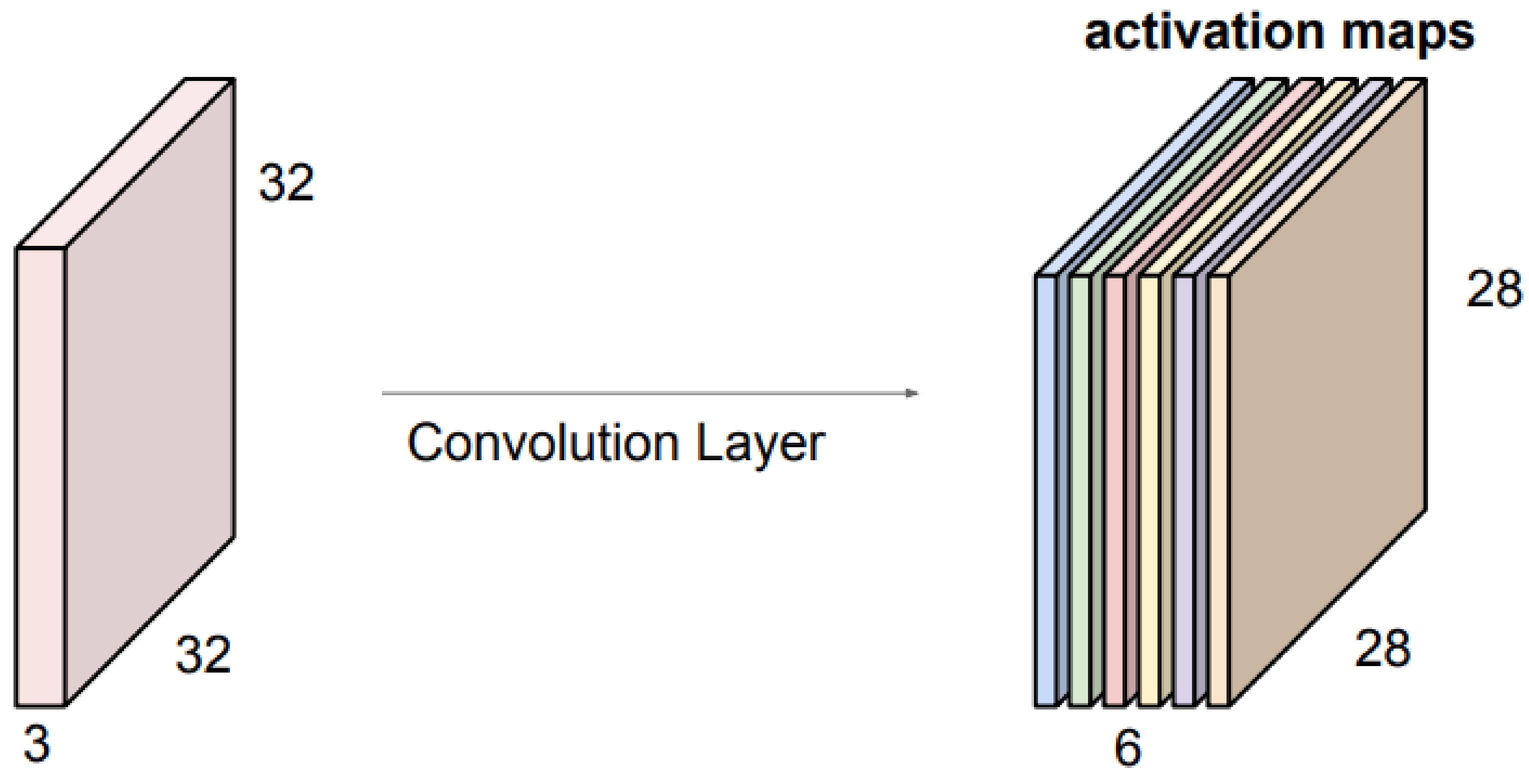


Convolution Layer



Convolution Layer

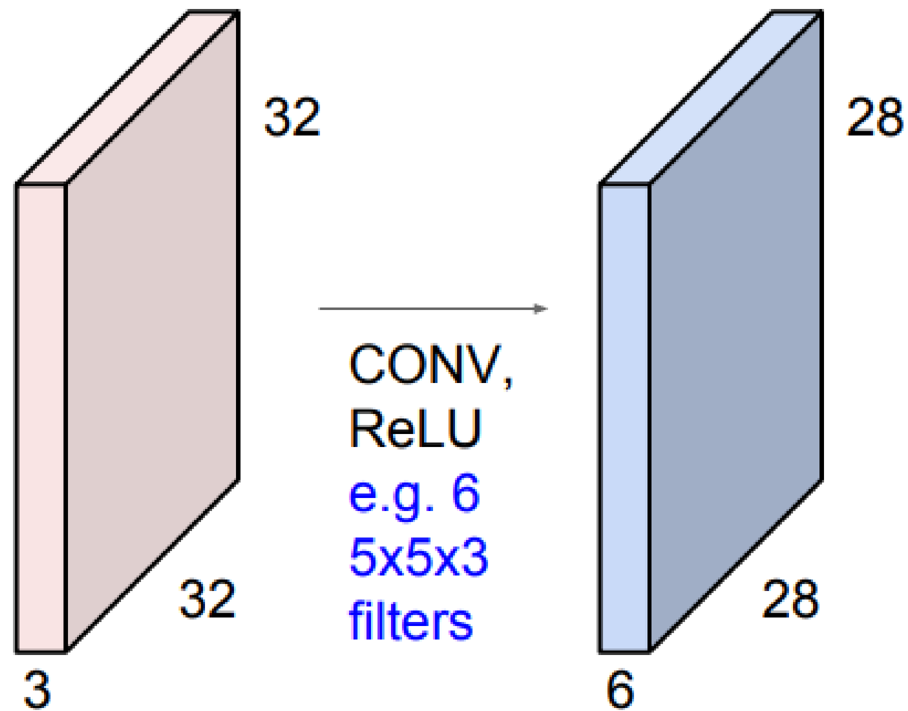
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



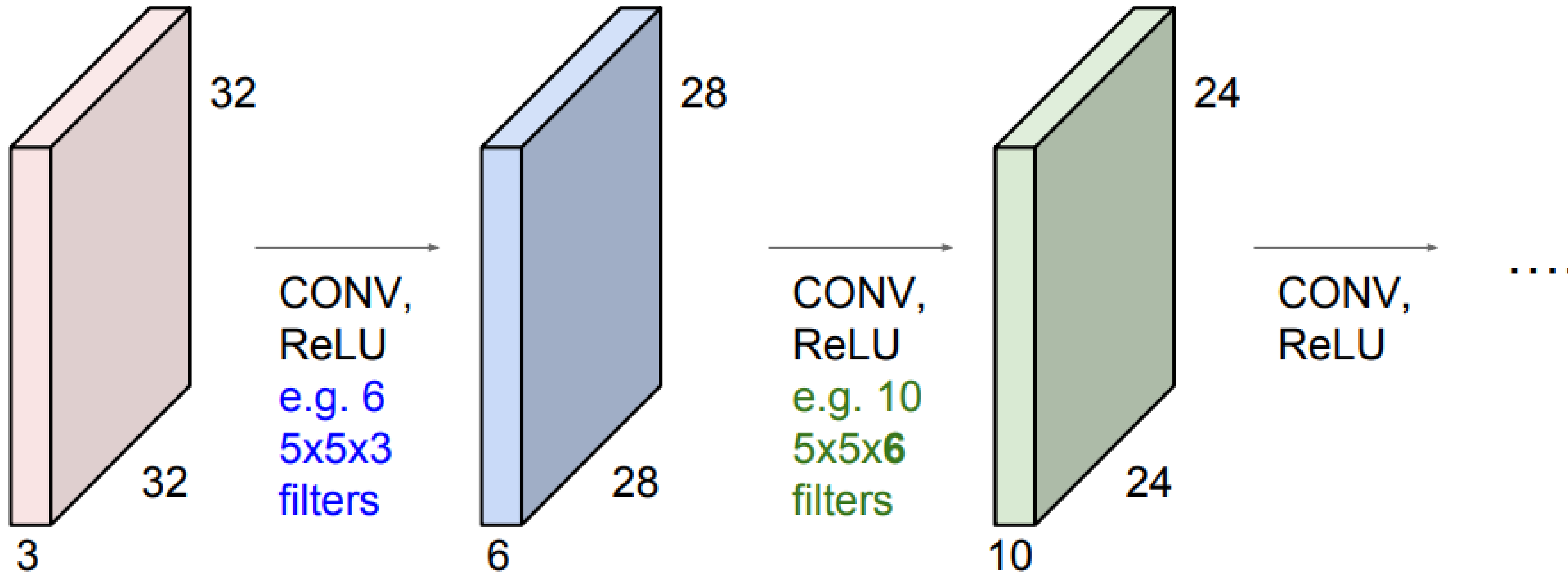
We stack these up to get a "new image" of size 28x28x6!

Convolution Layer

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolution Layer



Human Pose Estimation

Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

Johnson and Everingham, "Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation", BMVC 2010
Toshev and Szegedy, “DeepPose: Human Pose Estimation via Deep Neural Networks”, CVPR 2014

Human Pose Estimation

Vector:
4096

Correct left
foot: (x', y')

Left foot: (x, y) → L2 loss

Right foot: (x, y) → L2 loss

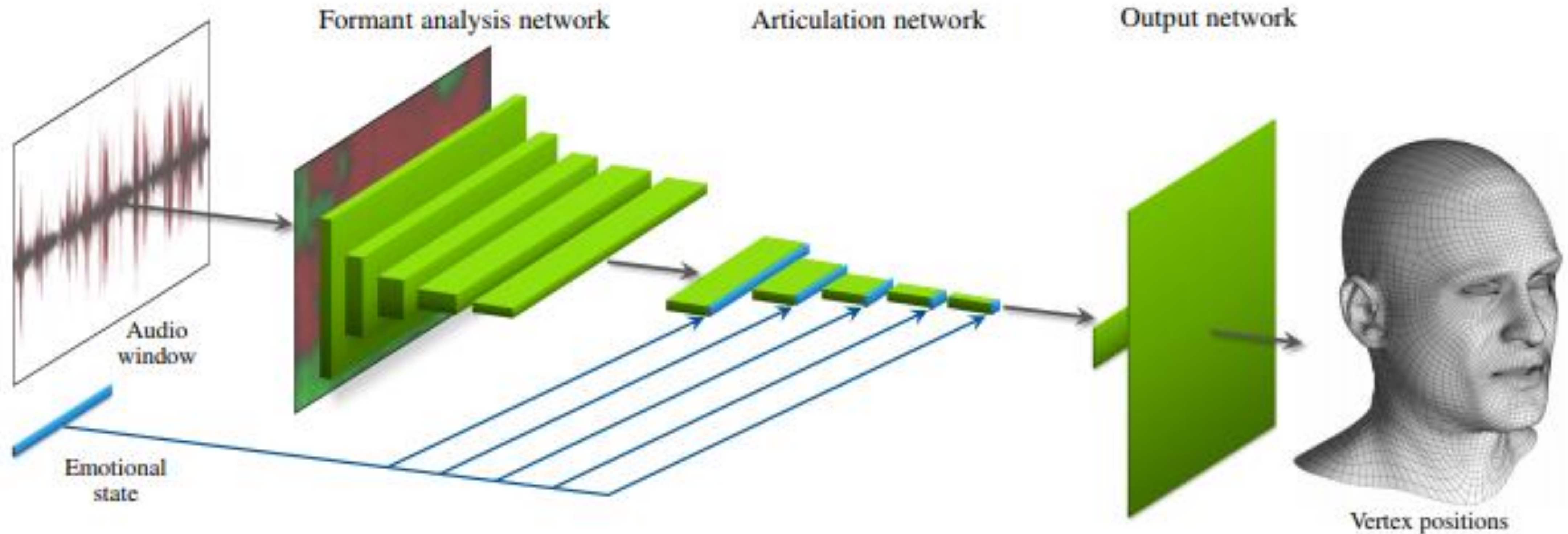
...

Head top: (x, y) → L2 loss

Correct head
top: (x', y')

\+ → **Loss**

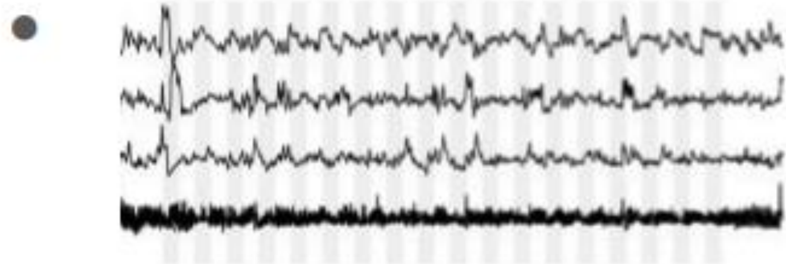
Animation using Convolutions



https://www.youtube.com/watch?v=lDzrfdpGqw4

Tero Karras, Timo Aila, Samuli Laine, Antti Herva, and Jaakko Lehtinen. 2017. Audio-driven facial animation by joint end-to-end learning of pose and emotion. ACM Trans. Graph. 36, 4, Article 94 (July 2017)

Recurrent Neural Networks (RNNs)

- "This morning I took the dog for a walk." sentence
- 
  medical signals
-
  speech waveform

© MIT 6.S191: Introduction to Deep Learning

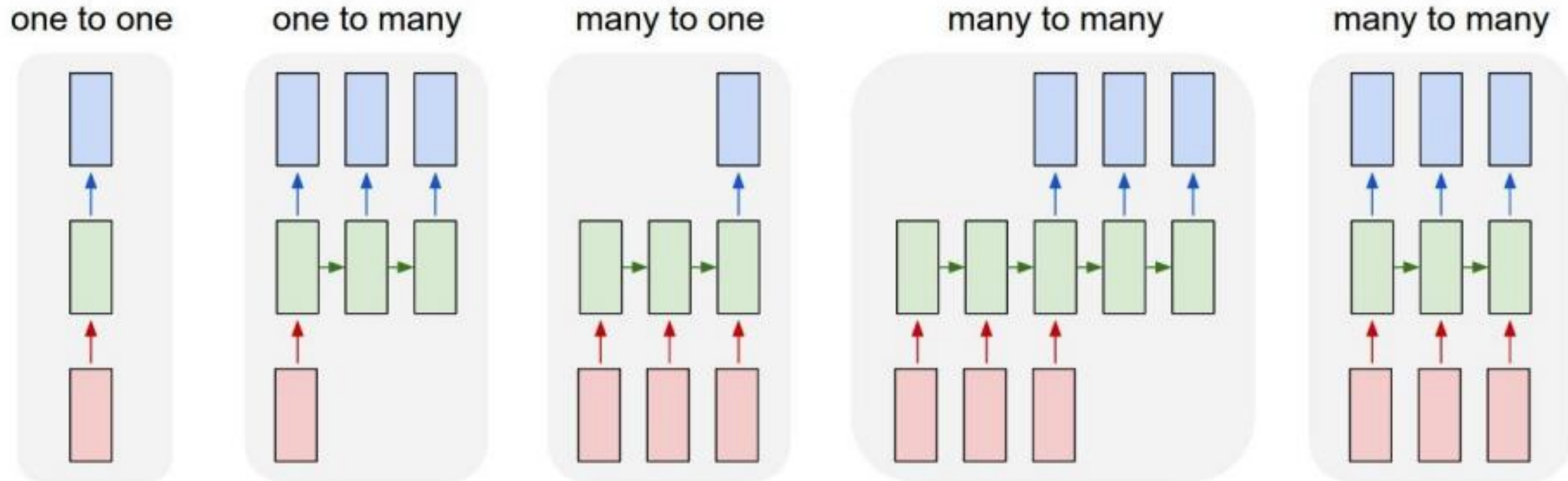
Why sequence models?

to model sequences, we need:

1. to deal with variable-length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

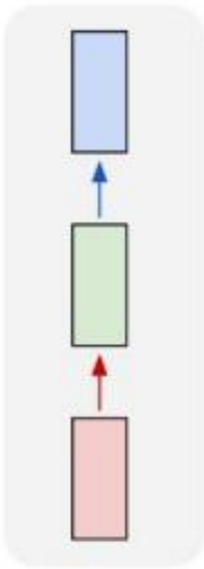
© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Recurrent Neural Networks: Process Sequences

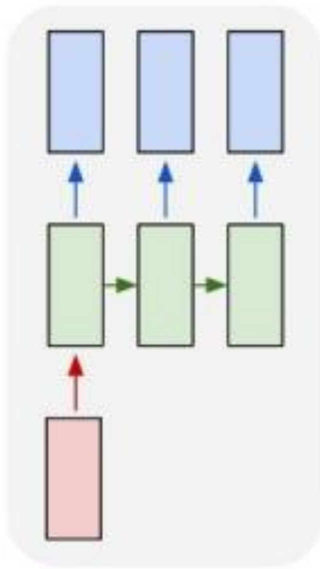


Recurrent Neural Networks: Process Sequences

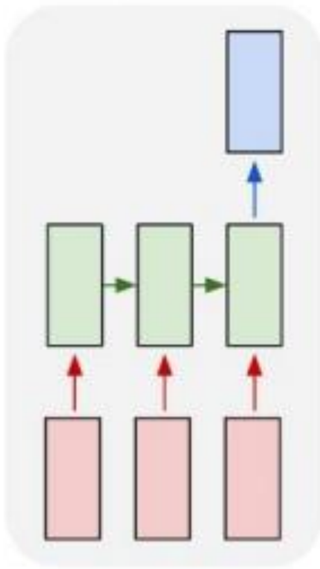
one to one



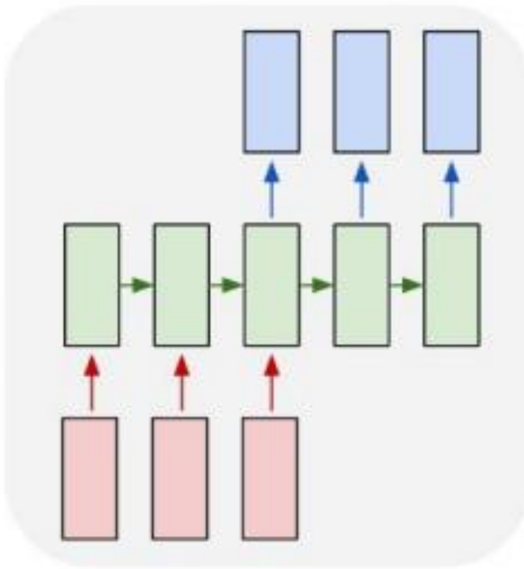
one to many



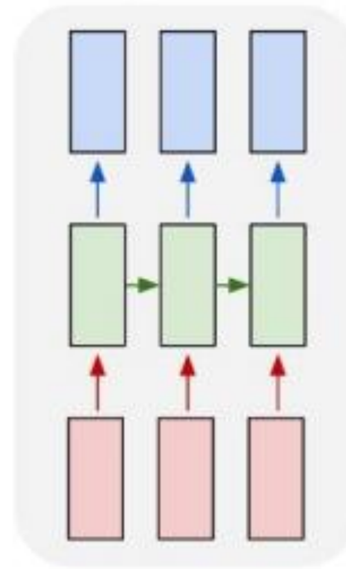
many to one



many to many

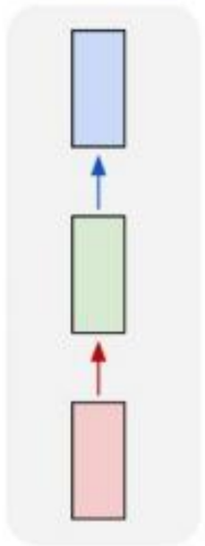


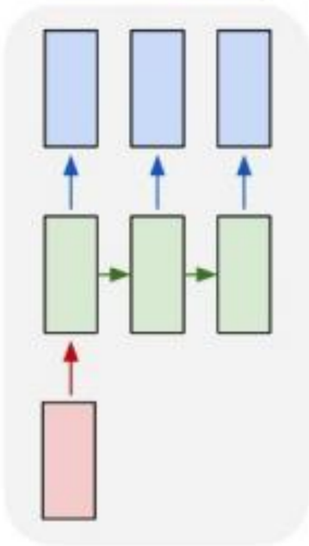
many to many

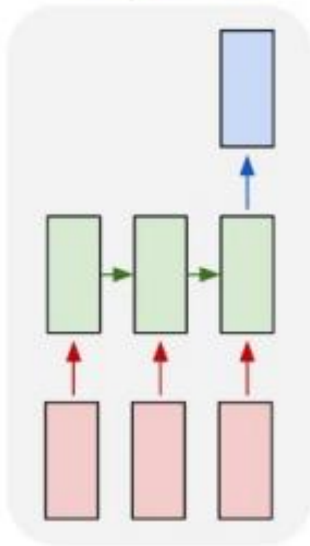


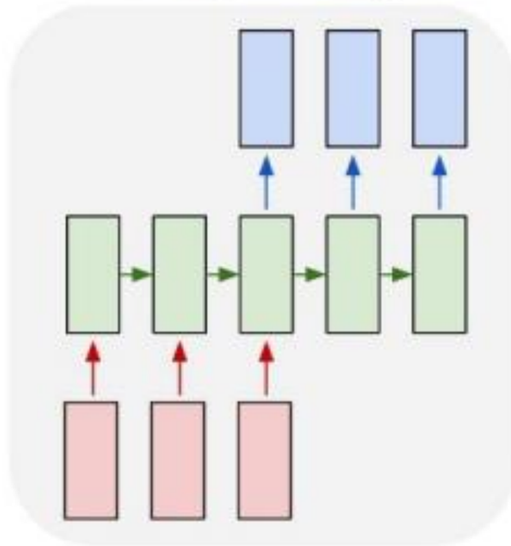
e.g. **Sentiment Classification**
sequence of words -> sentiment

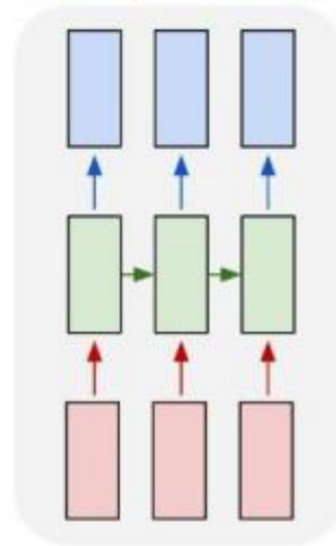
Recurrent Neural Networks: Process Sequences





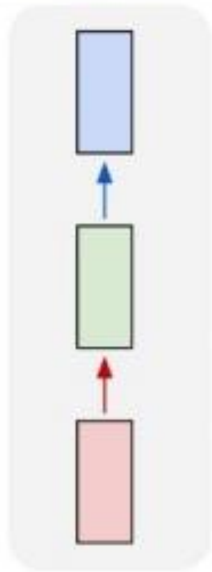


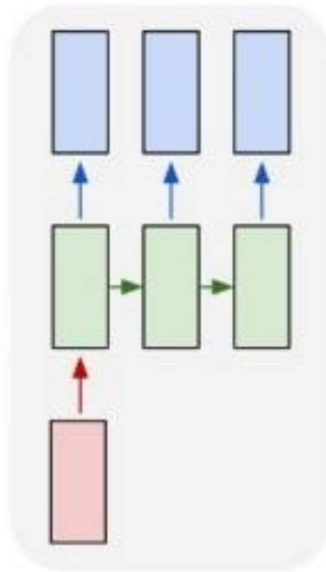


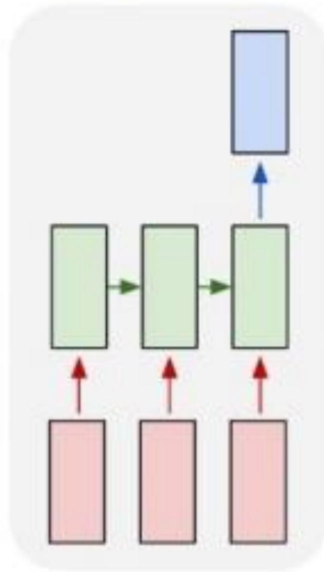


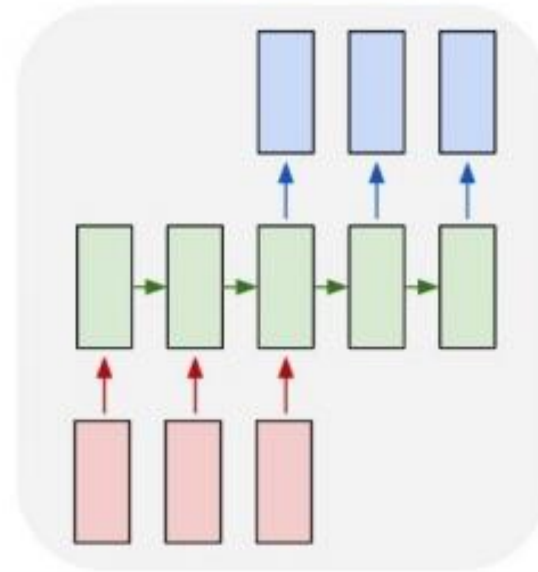
e.g. **Machine Translation**
seq of words -> seq of words

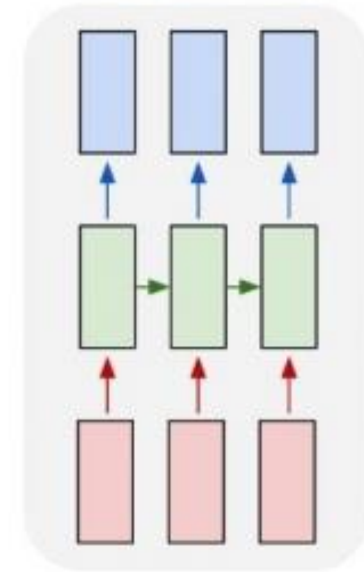
Recurrent Neural Networks: Process Sequences





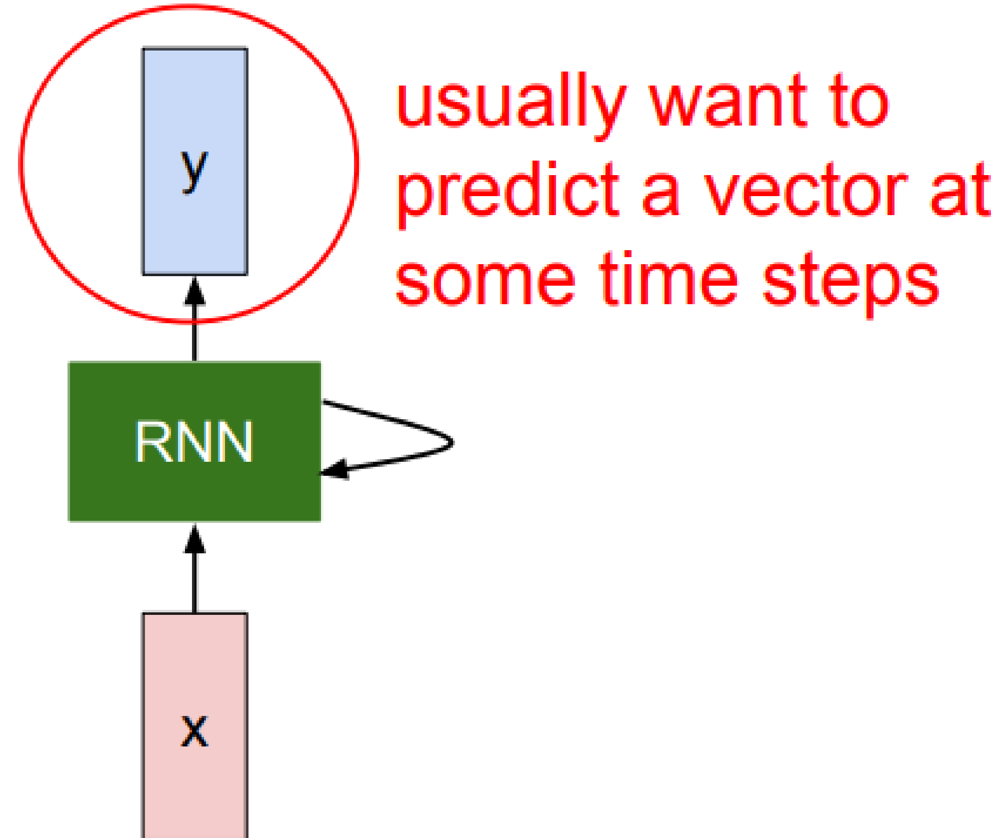






e.g. **Video classification on frame level**

Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

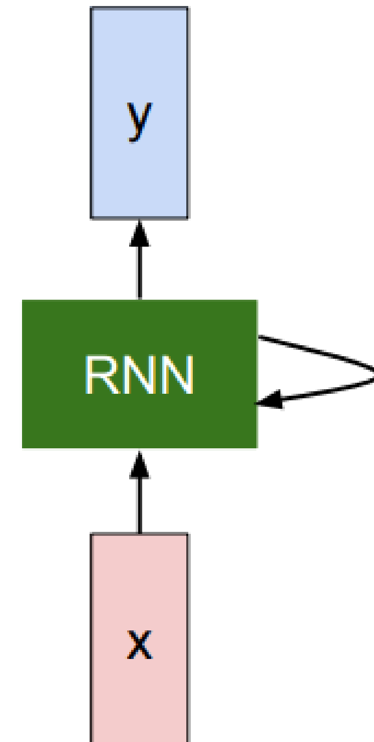
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step

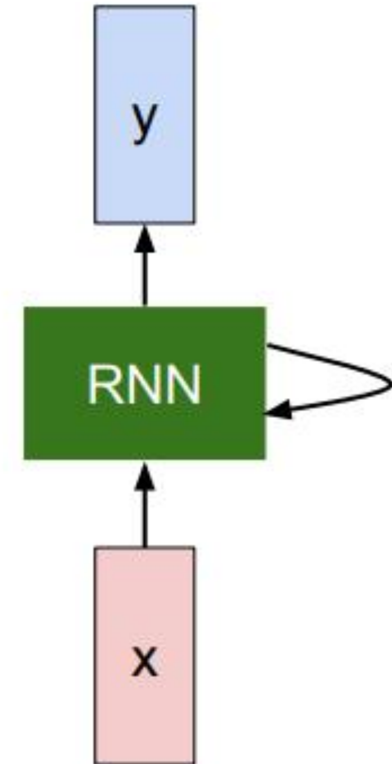


Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

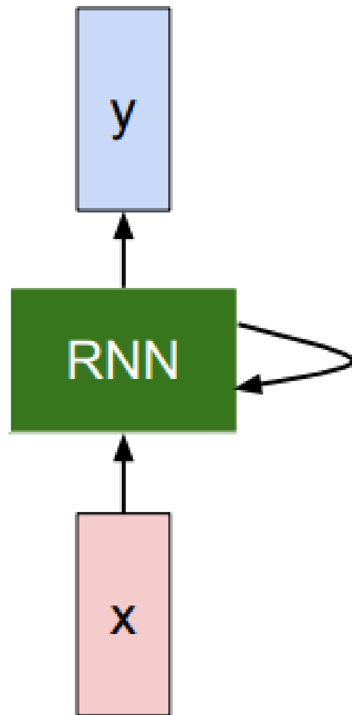
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:

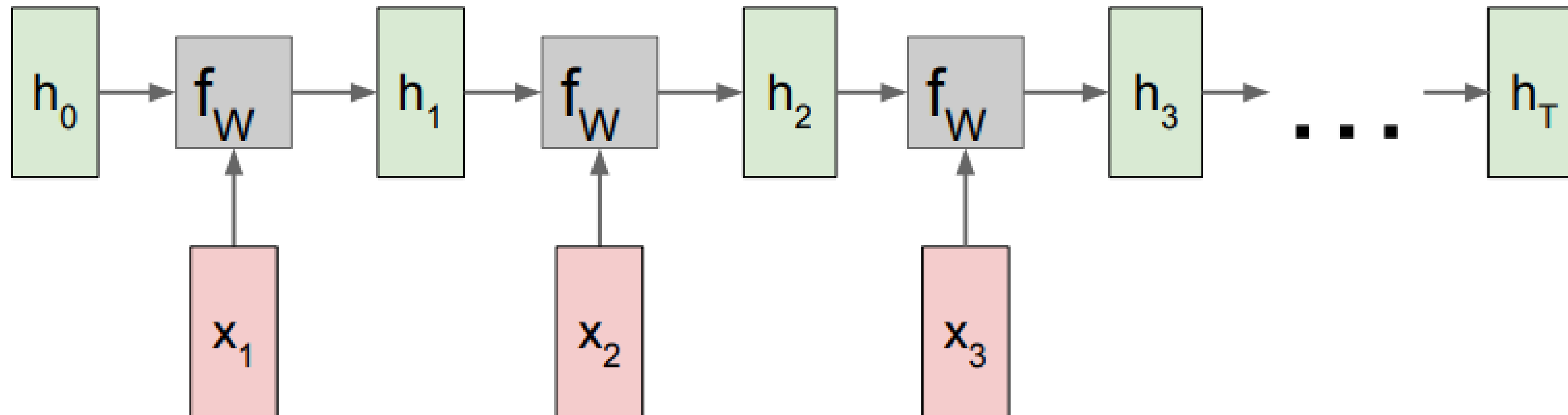


$$h_t = f_W(h_{t-1}, x_t)$$

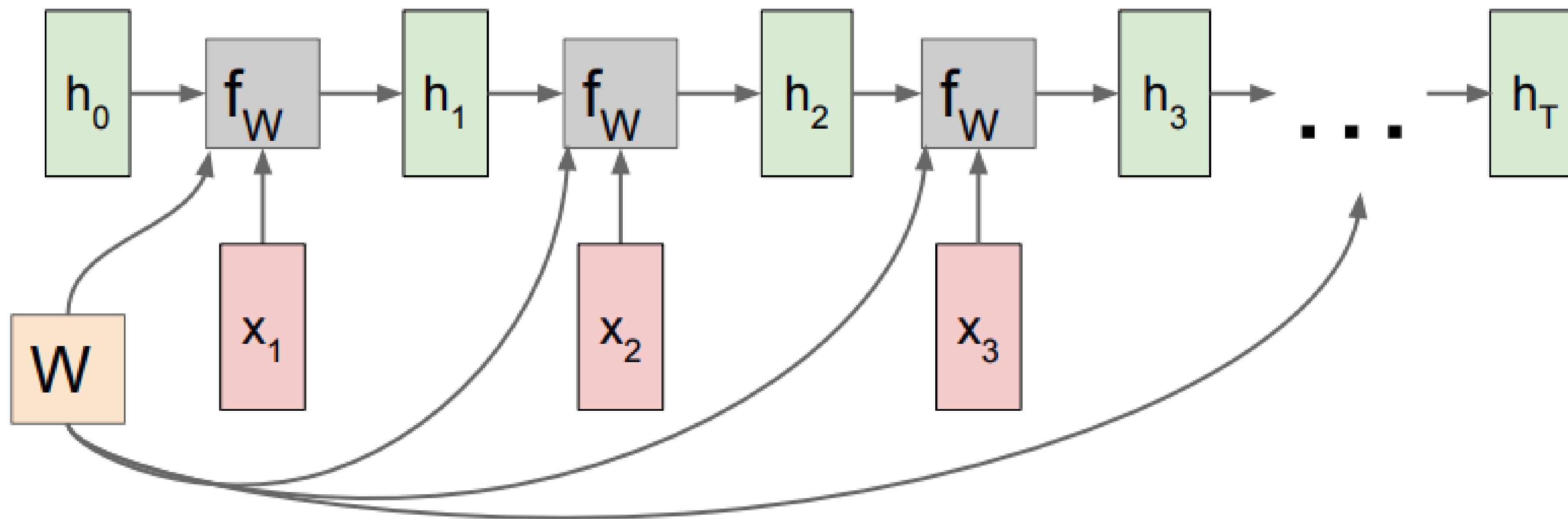
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

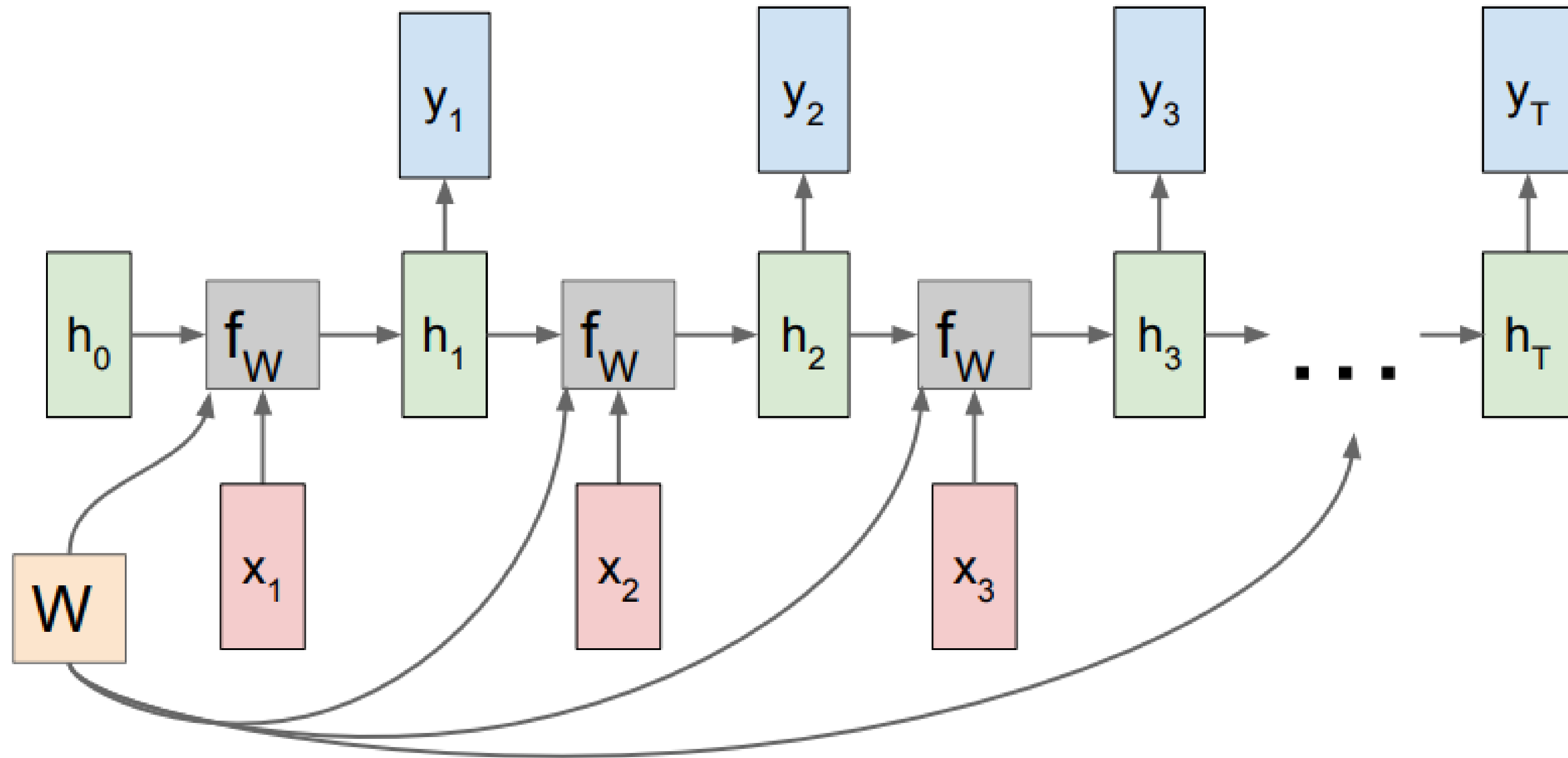
RNN: Computational Graph



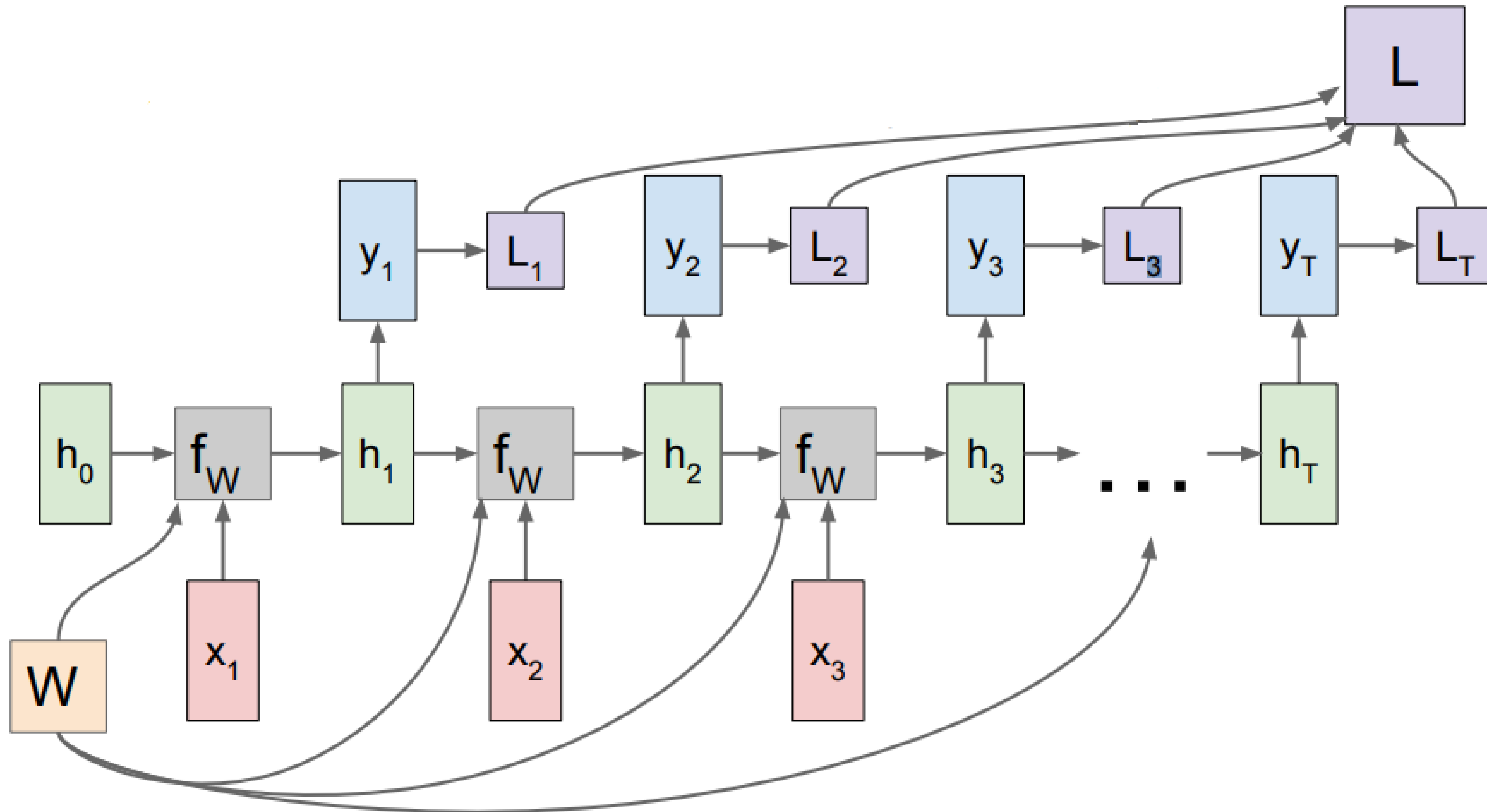
RNN: Computational Graph



RNN: Computational Graph: Many to Many

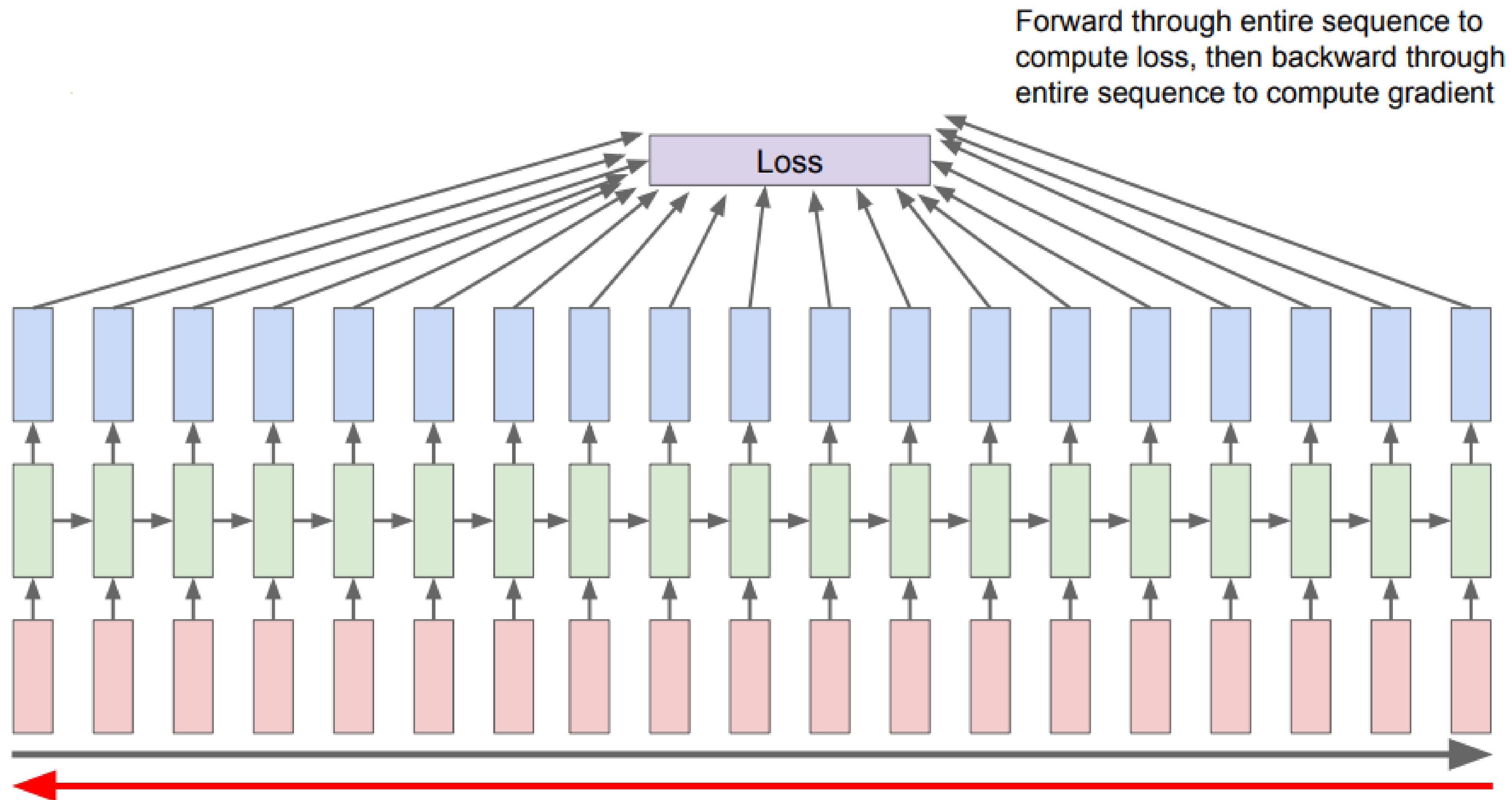


RNN: Computational Graph: Many to Many



Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



how do we **train** an RNN?

backpropagation!

(through time)

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

remember: **backpropagation**

1. take the derivative (gradient) of the loss with respect to each parameter

2. shift parameters in the opposite direction in order to minimize loss

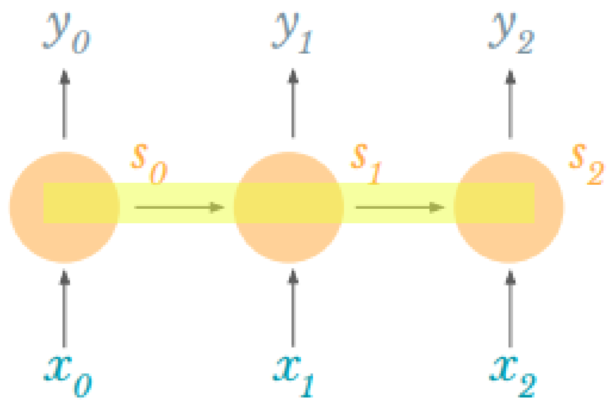
© MIT 6.S191: Introduction to Deep Learning

why are RNNs **hard to train**?

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

problem: vanishing gradient

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$



at $k = 0$: $\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$

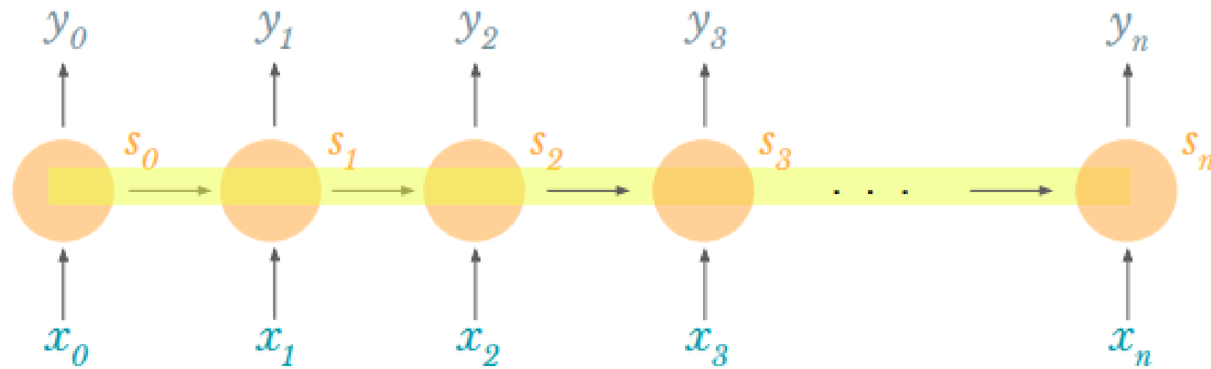
© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

problem: vanishing gradient

$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

as the gap between timesteps gets bigger, this product gets longer and longer!



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

we’re multiplying a lot of **small numbers** together.

so what?

errors due to further back timesteps have increasingly **smaller gradients.**

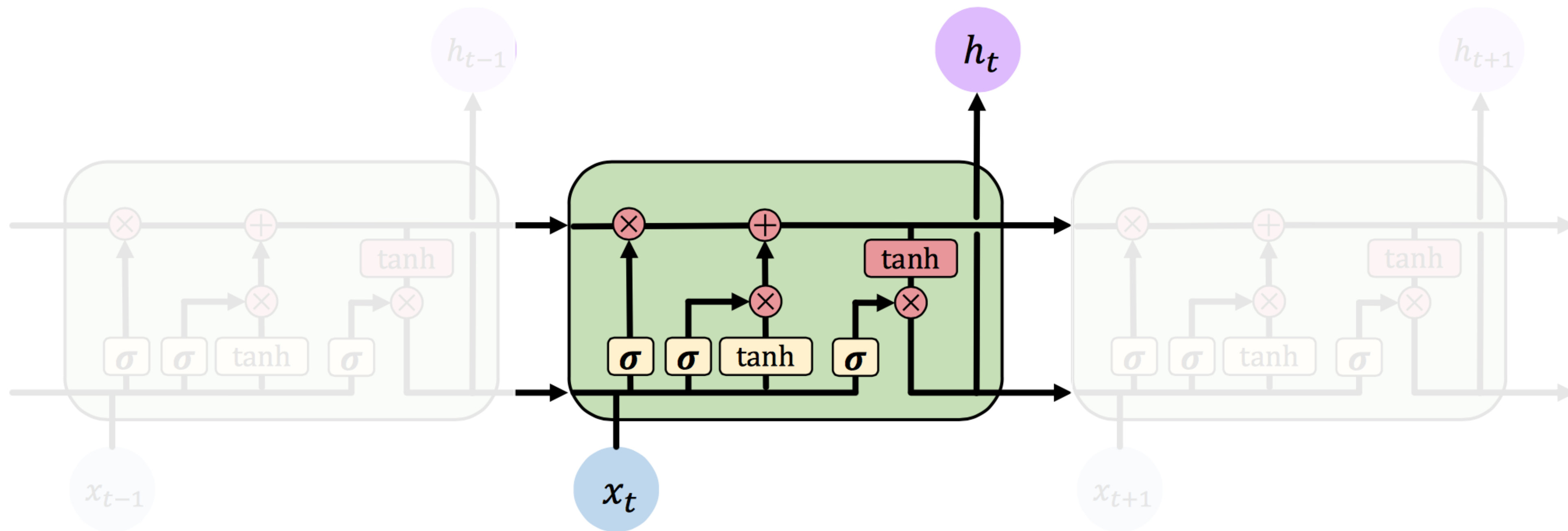
so what?

parameters become biased to **capture shorter-term** dependencies.

© MIT 6.S191: Introduction to Deep Learning

Long Short Term Memory (LSTMs)

LSTM repeating modules contain **interacting layers** that **control information flow**



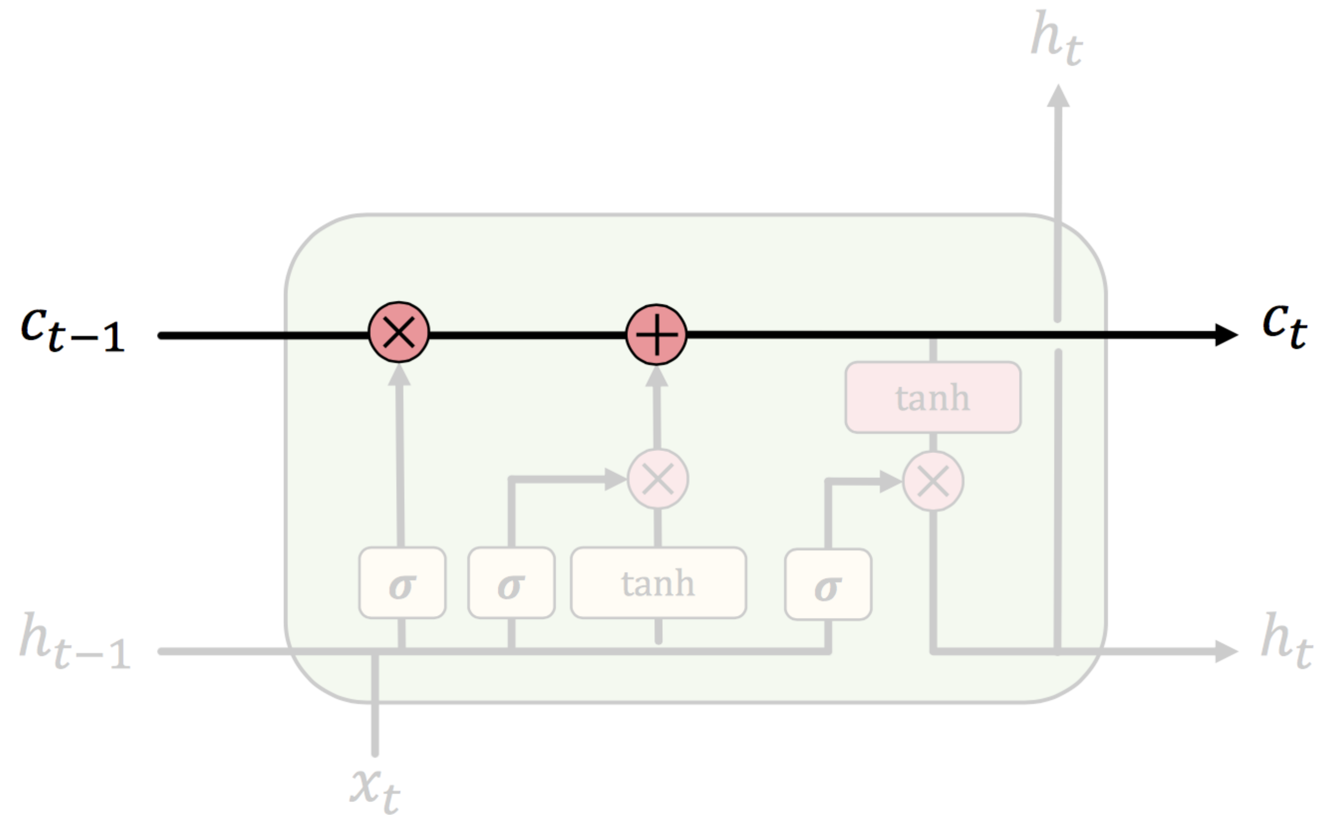
LSTM cells are able to track information throughout many timesteps

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 1997), 1735-1780

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Long Short Term Memory (LSTMs)

LSTMs maintain a **cell state** c_t where it's easy for information to flow

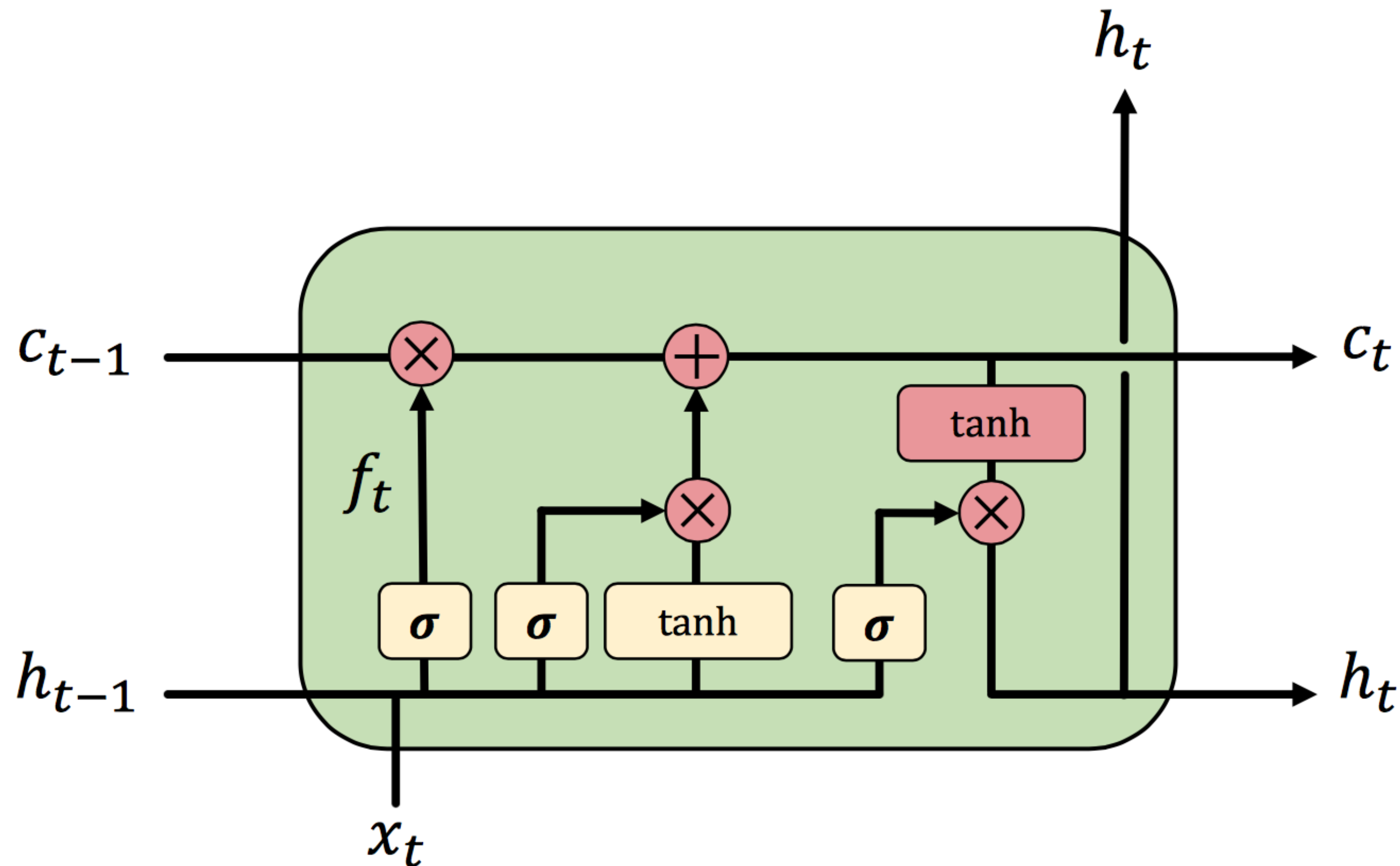


© MIT 6.S191: Introduction to Deep Learning

Long Short Term Memory (LSTMs)

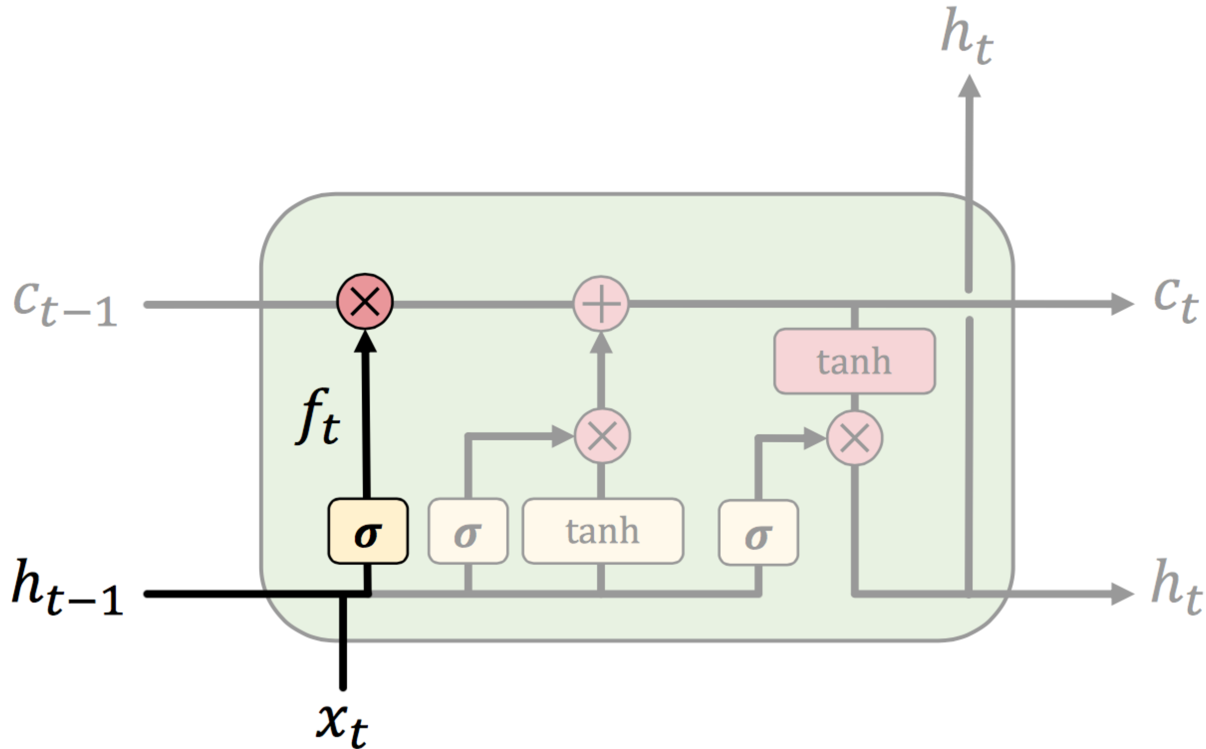
How do LSTMs work?

1) Forget 2) Update 3) Output



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTMs: forget irrelevant information



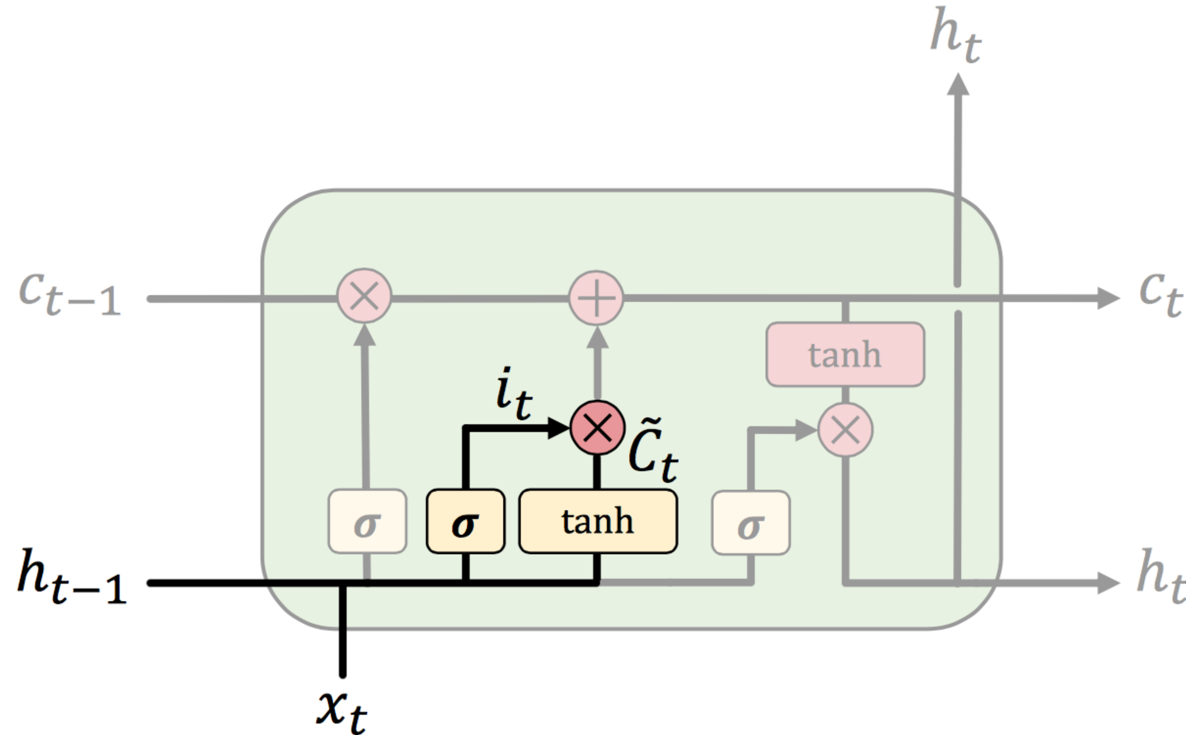
$$f_t = \sigma(\boldsymbol{W_i}[\, h_{t-1}, x_t \,] + b_f)$$

- Use previous cell output and input
- Sigmoid: value 0 and 1 – "completely forget" vs. "completely keep"

ex: Forget the gender pronoun of previous subject in sentence.

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTMs: identify new information to be stored



$$i_t = \sigma(\boldsymbol{W_i}[\, h_{t-1}, x_t \,] + b_i)$$

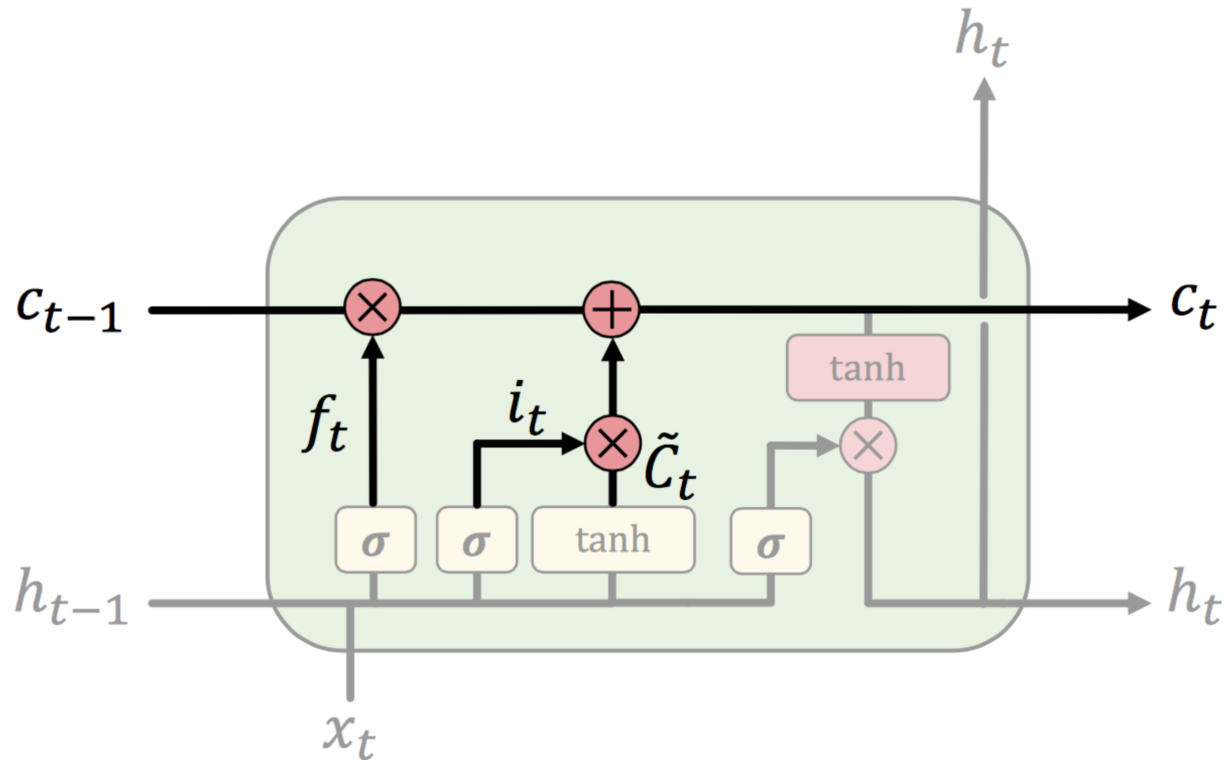
$$\tilde{C}_t = \tanh(\boldsymbol{W_C}[\, h_{t-1}, x_t \,] + b_C)$$

- Sigmoid layer: decide what values to update
- Tanh layer: generate new vector of "candidate values" that could be added to the state

ex: Add gender of new subject to replace that of old subject.

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTMs: update cell state



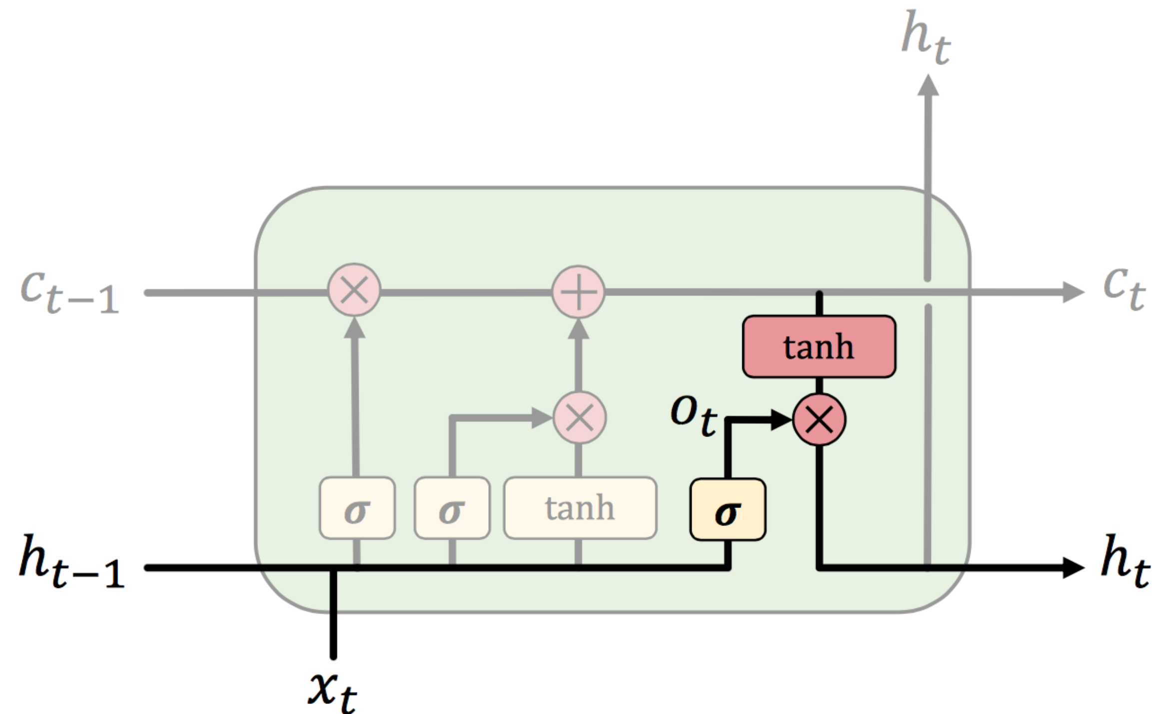
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Apply forget operation to previous internal cell state: $f_t * C_{t-1}$
- Add new candidate values, scaled by how much we decided to update: $i_t * \tilde{C}_t$

ex: *Actually drop old information and add new information about subject's gender.*

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTMs: output filtered version of cell state



$$o_t = \sigma(\boldsymbol{W_o}[\, h_{t-1}, x_t \,] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

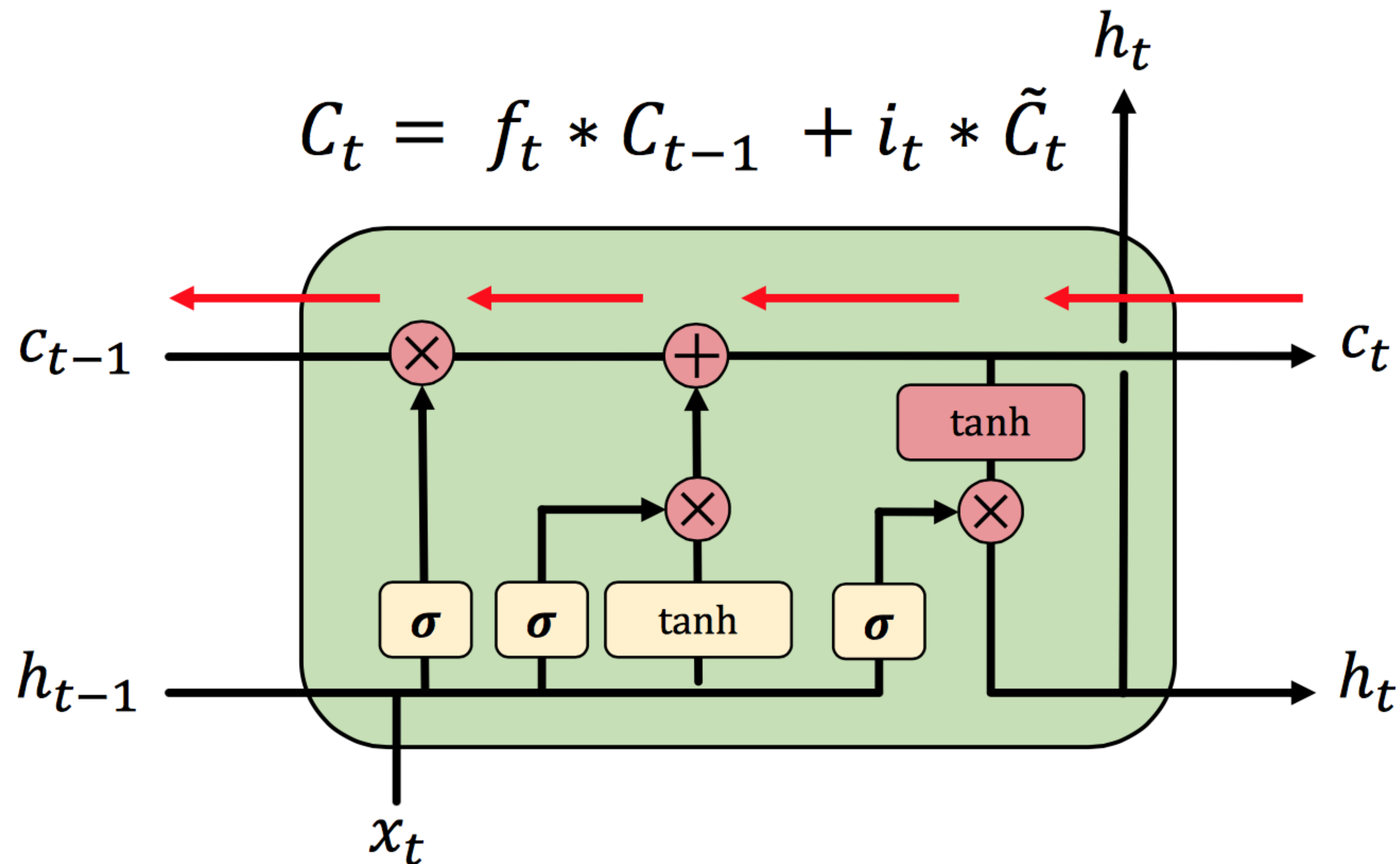
- Sigmoid layer: decide what parts of state to output
- Tanh layer: squash values between -1 and 1
- $o_t * \tanh(C_t)$: output filtered version of cell state

ex: Having seen a subject, may output information relating to a verb.

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTM gradient flow

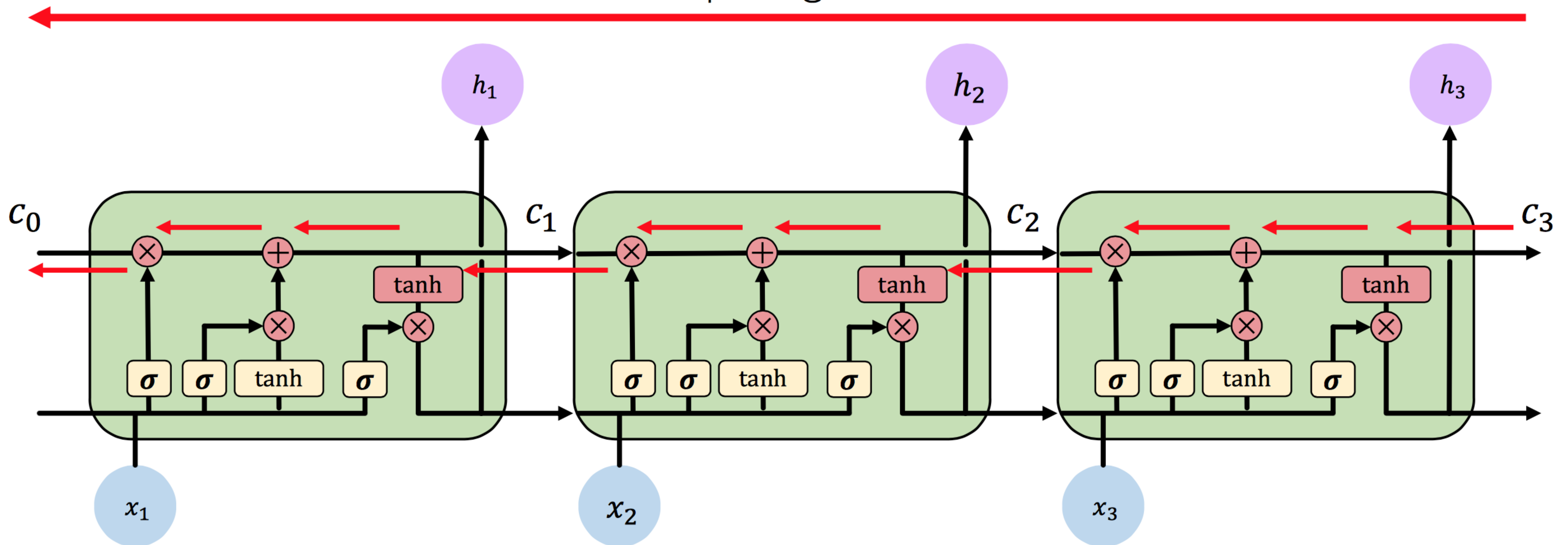
Backpropagation from C_t to C_{t-1} requires only elementwise multiplication!
No matrix multiplication → avoid vanishing gradient problem.



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTM gradient flow

Uninterrupted gradient flow!



© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

LSTMs: key concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - Forget gate gets rid of irrelevant information
   - Selectively update cell state
   - Output gate returns a filtered version of the cell state
3. Backpropagation from c_t to c_{t-1} doesn't require matrix multiplication: **uninterrupted gradient flow**

© MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

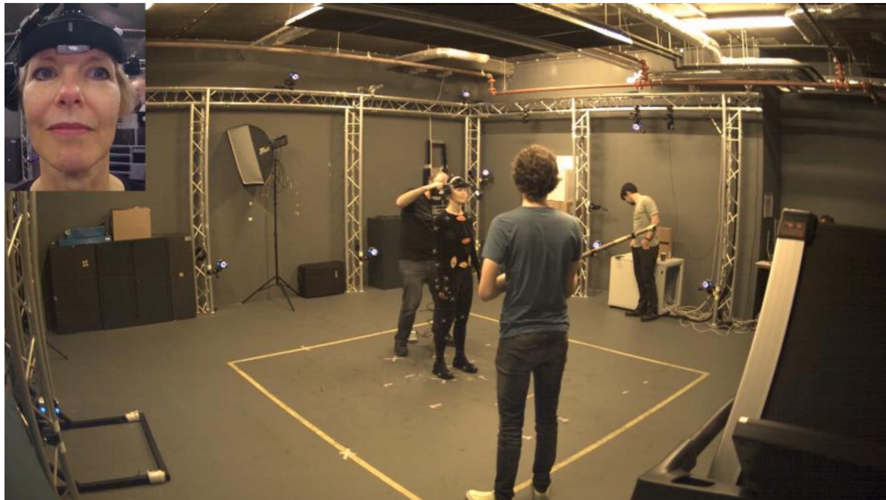
Other RNN Variants

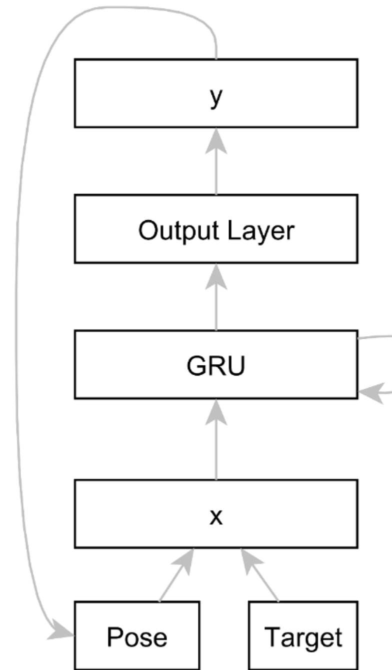
GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

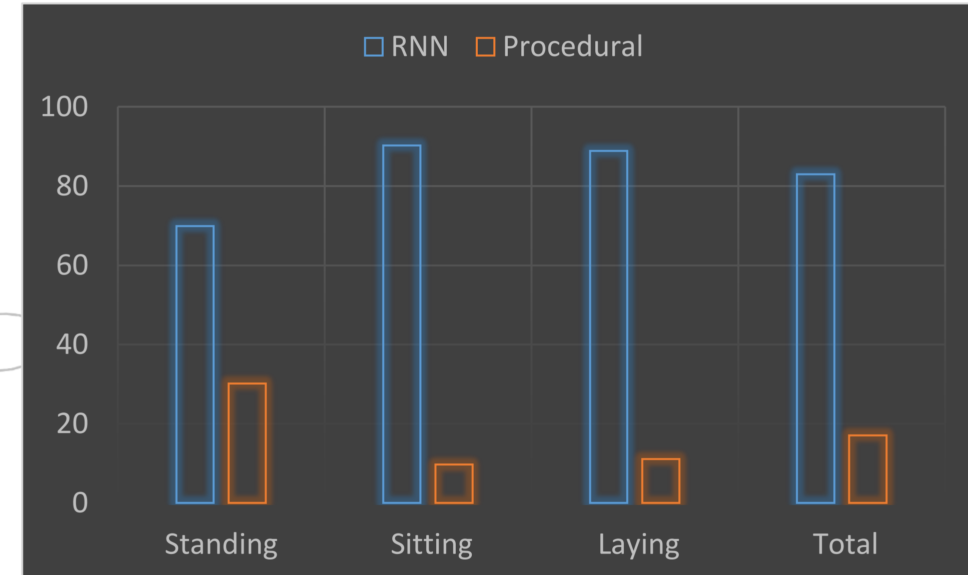
$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

Computer Animation

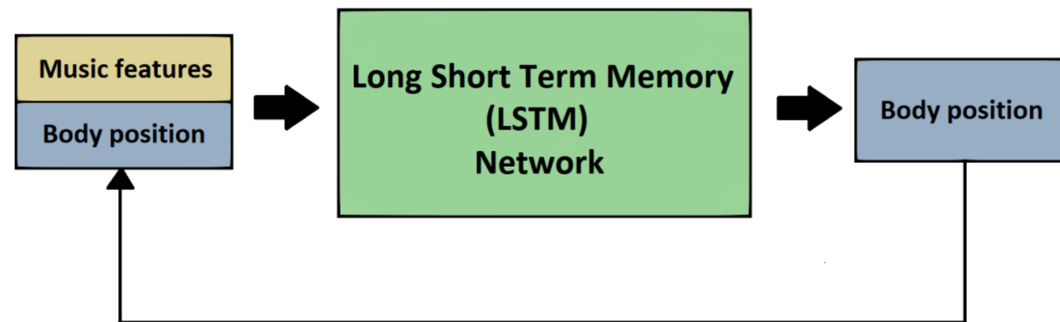






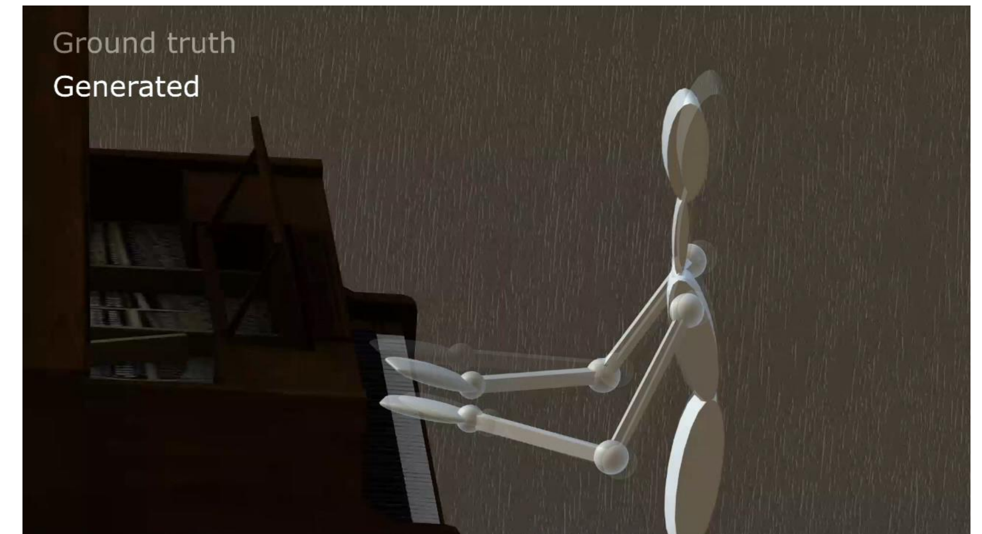
A. Klein, Z. Yumak, A. Beij and A. F. van der Stappen. Data-driven Gaze Animation using Recurrent Neural Networks. ACM Siggraph Conference on Motion, Interaction and Games, October 2019 (Best Student Paper Award)

Music-driven Expressive Gestures



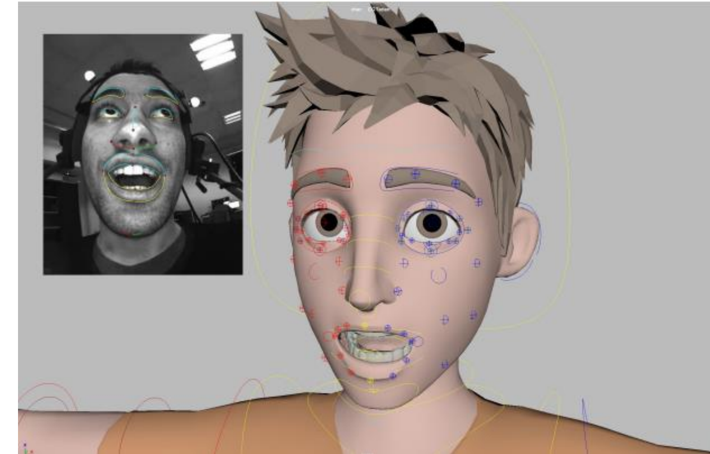
Features/Metric	MSE	#epochs	APE		Acceleration		Jerk	
			μ	σ	μ	σ	μ	σ
Pitch	2763.754	37	0.15126	0.03007	-2.19118	0.90039	18.59880	10.68545
Pitch+Beat	2837.244	22	0.15226	0.03064	-2.18774	0.89687	20.51324	14.38038
Pitch+RMS	2478.616	50	0.14495	0.04101	-2.13702	0.87939	11.22615	10.39768
Pitch+Beat+RMS	2675.961	35	0.14857	0.03512	-2.16969	0.87687	7.99591	9.56305
MFCC	2111.649	50	0.11749	0.03671	-2.14330	0.88659	12.32060	12.52234
All	2153.945	50	0.11790	0.03481	-2.12027	0.88340	1.18094	7.19398
Ground Truth	-	-	-	-	-2.12803	0.74442	2.62996	3.07753

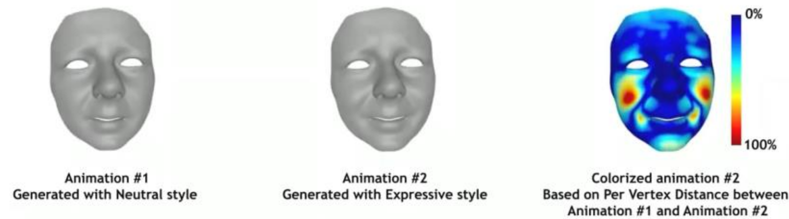
Table 1: The MSE, APE, acceleration and jerk of each condition over all generated animations.

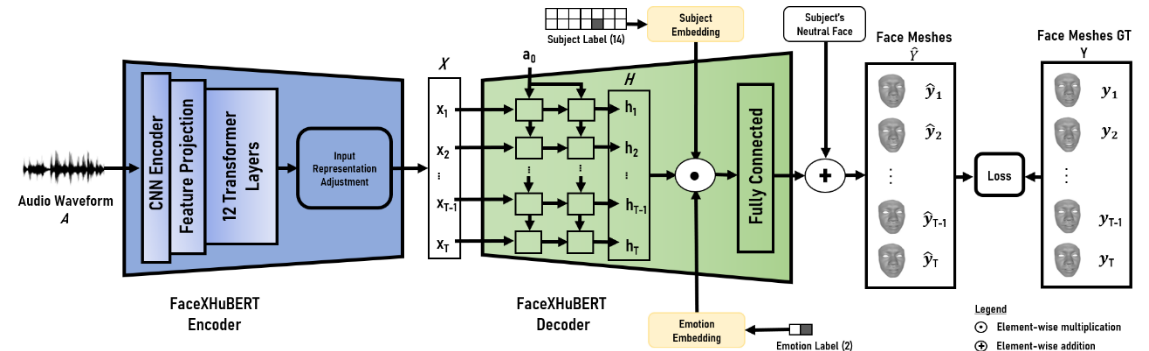


A. Bogaers, Z. Yumak, A. Volk. **Music-Driven Animation Generation of Expressive Musical Gestures**. ACM International Conference on Multimodal Interaction (ICMI 2020)

Data-driven Facial Animation







FaceXHuBERT: Text-less Speech-driven E(X)pressive 3D Facial Animation Synthesis Using Self-Supervised Speech Representation Learning, Kazi Injamamul Haque and Zerrin Yumak, 25th ACM International Conference on Multimodal Interaction, ACM ICMI 2023, Paris France (https://galib360.github.io/FaceXHuBERT)

FaceDiffuser: Speech Driven 3D Facial Animation Synthesis using Diffusion, Stefan Stan, Kazi Injamamul Haque and Zerrin Yumak, ACM Siggraph Conference on Motion, Interaction and Games, MIG 2023, Rennes, France (https://uuembodiedsocialai.github.io/FaceDiffuser/)

References and Supplementary Material

- Deep Learning book: https://www.deeplearningbook.org/

- Coursera Deeplearning.ai on YouTube:
- https://www.deeplearning.ai/courses/deep-learning-specialization/
- https://www.deeplearning.ai/courses/generative-adversarial-networks-gans-specialization/
- https://www.deeplearning.ai/courses/generative-ai-for-everyone/

- CS231N: Convolutional Neural Networks, Stanford University
- http://cs231n.stanford.edu/schedule.html

- 6.S191: Introduction to Deep Learning, MIT
- http://introtodeeplearning.com/

- UU Pattern Recognition and Deep Learning
- UvA Deep Learning Course

- Next lectures:
  - Motion Capture Tutorial
  - Advanced Facial Animation and Deep Learning Tutorial

- Intermediary report deadline: 4th December 23:59