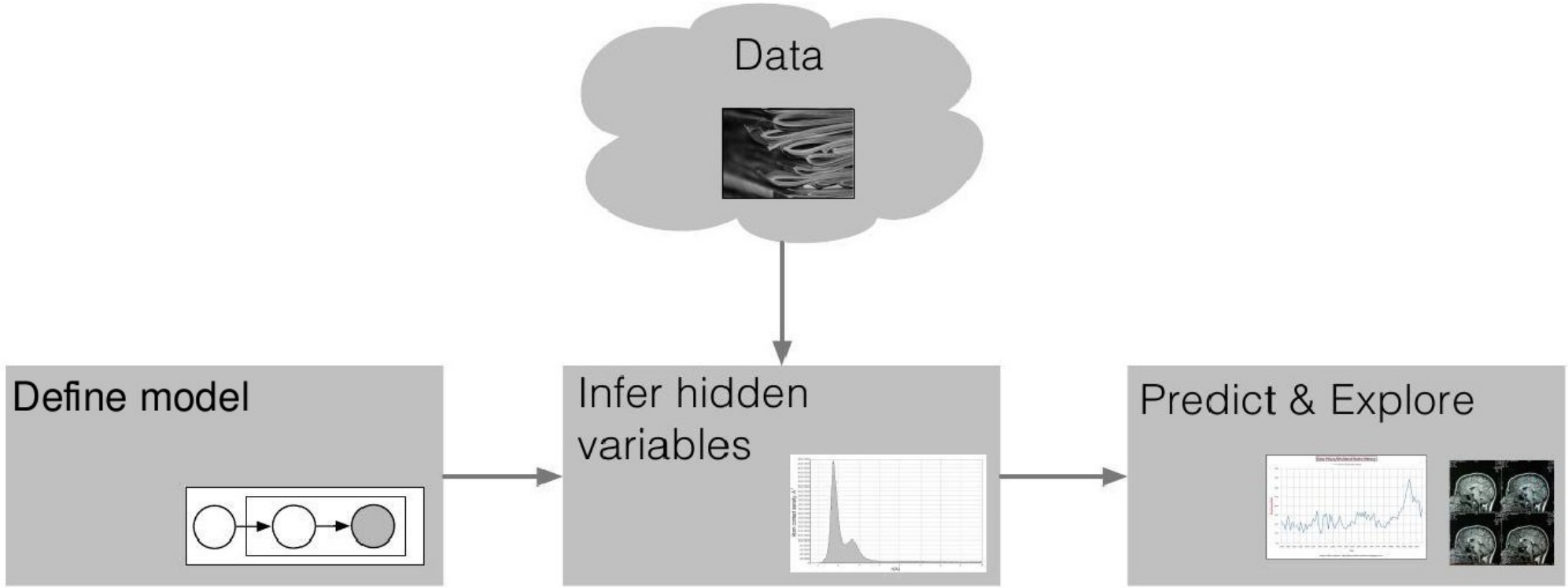
An Introduction to Deep Learning

A machine learning flowchart



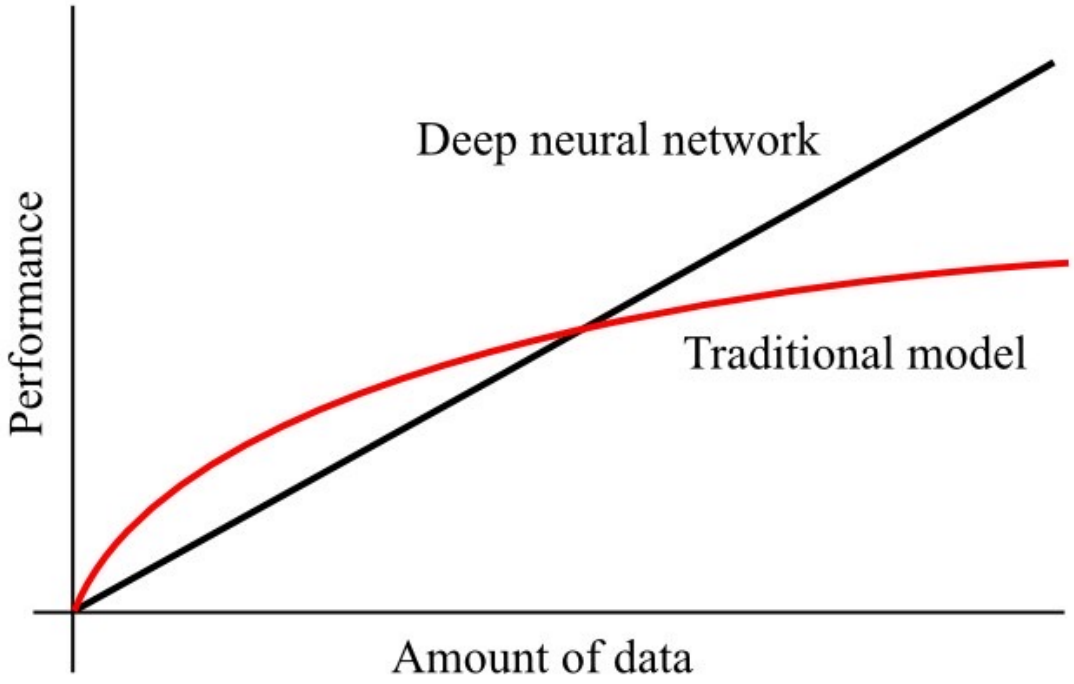
Deep learning: why now?

- Neural networks have been around since the 1980s.
- Scalability in two directions: computation and data
- Complex models learn tasks better, however...

✔ Need more data

✔ Need faster computation

✔ Need expressive models and optimization tricks



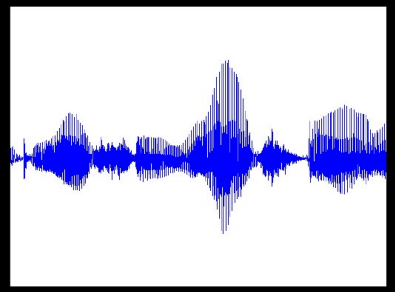
Some things to do with data...

AI is changing our world!

Images

→ Label image

Audio



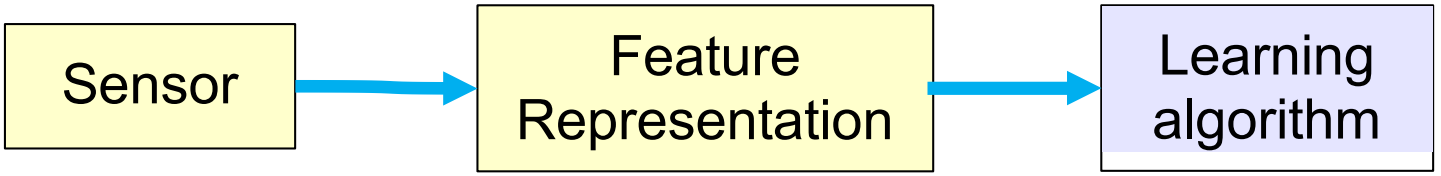
→ Speech recognition

Text

→ Web search

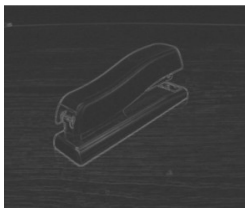
Sunrise Technology Autonomous Driving

But how do we do it?



Object detection

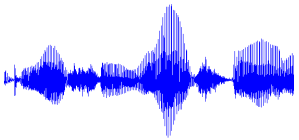
Image



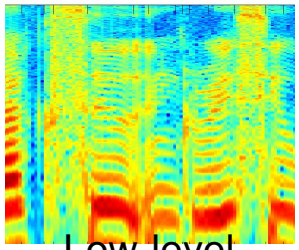
Low-level

Recognition

Audio classification



Audio

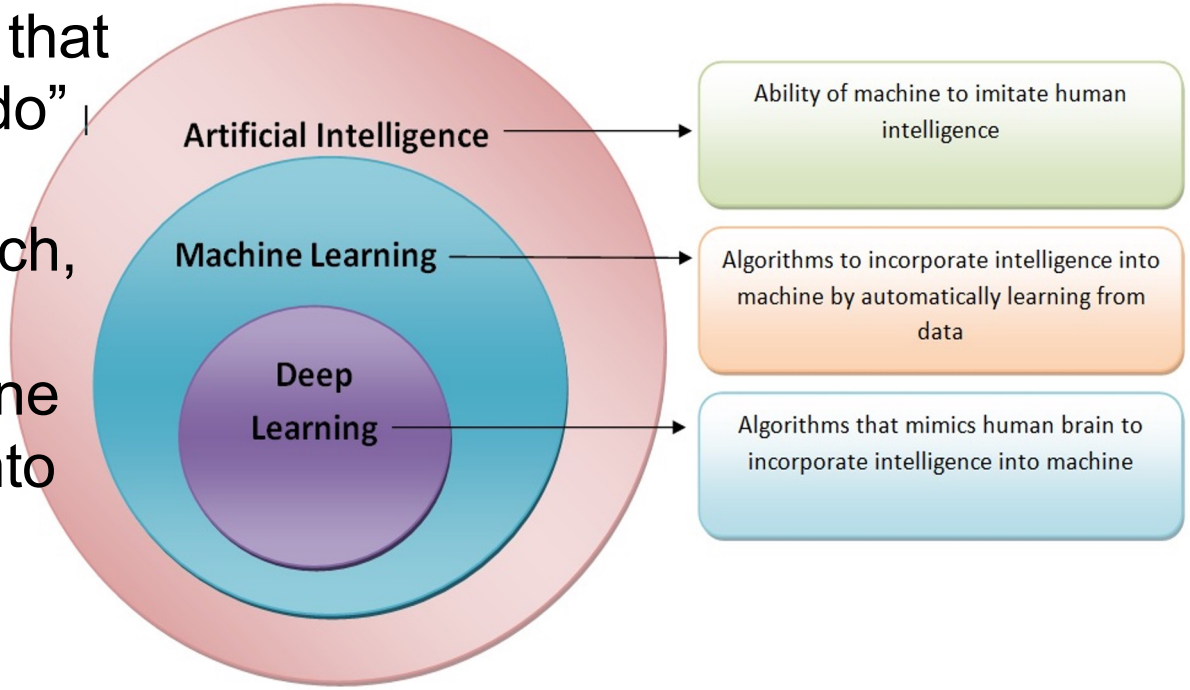


Low-level audio features

Speaker

What can deep learning do?

- **Short answer**: Nothing that other algorithms can't "do" already.
  But they often do it much, much better.
- **Longer answer**: Machine learning breaks down into supervised and unsupervised tasks

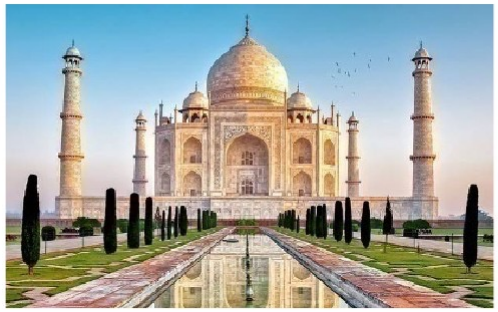


Supervised: Predict output corresponding to input.

Deep learning excels here. We will focus on this problem.

– Unsupervised: Learn structure from data.

- A newer research thrust. Many people focusing on this recently because there's still much room for improvement.



mosque, tower, building, cathedral, dome, castle

ski, skiing, skiers, skiiers, snowmobile

kitchen, stove, oven, refrigerator, microwave

bowl, cup, soup, cups, coffee

beach

snow

a car is parked in
the middle of nowhere .

a wooden table and chairs
arranged in a room .

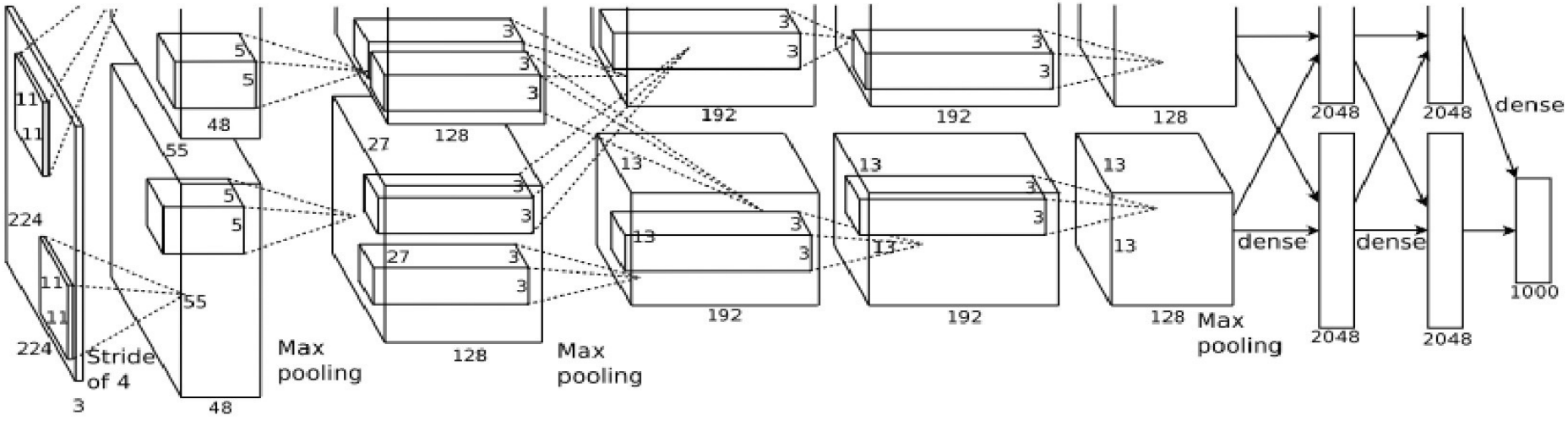
there is a cat sitting on a shelf .

a ferry boat on a marina
with a group of people .

a little boy with a bunch
of friends on the street .

ImageNet

- A difficult classification challenge dataset
  - Training data: 1.2M images, 1K categories
  - Testing: label 150K new images
- "AlexNet" shocked with 15.3% error rate versus 26%, which was best at that time

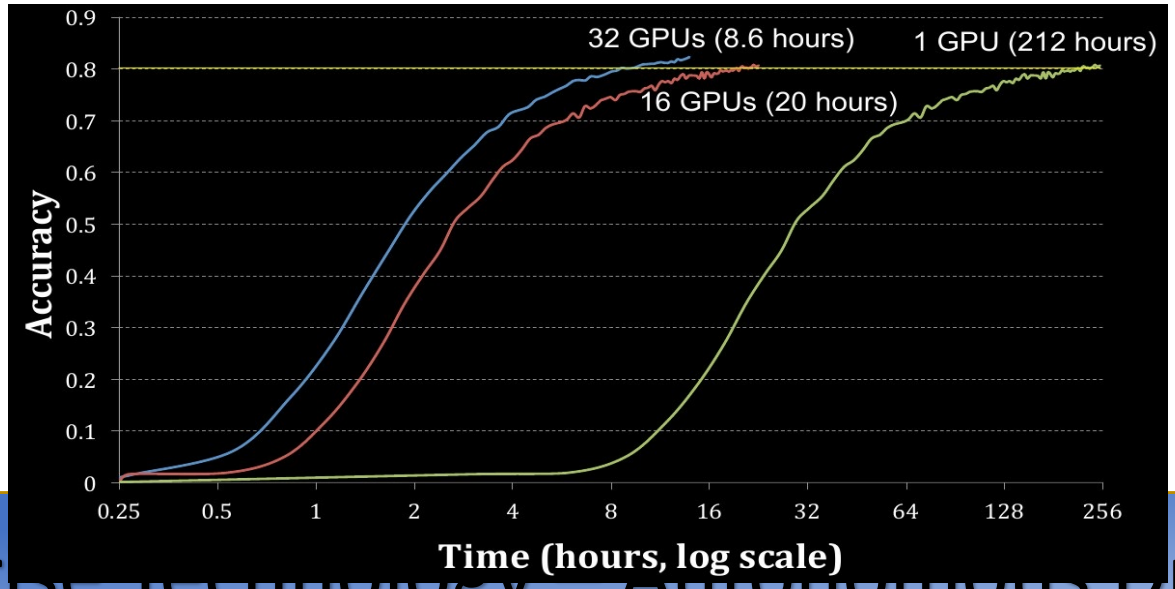


Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems,* 2012.

Public software (a selection)

- TensorFlow (Google)
- Caffe (UC Berkeley)
- Theano (University of Montreal)
- PyTorch (used by Facebook)
- MXNet (Amazon)

Implementation: GPUs are important for scaling



Building blocks of a deep learning model

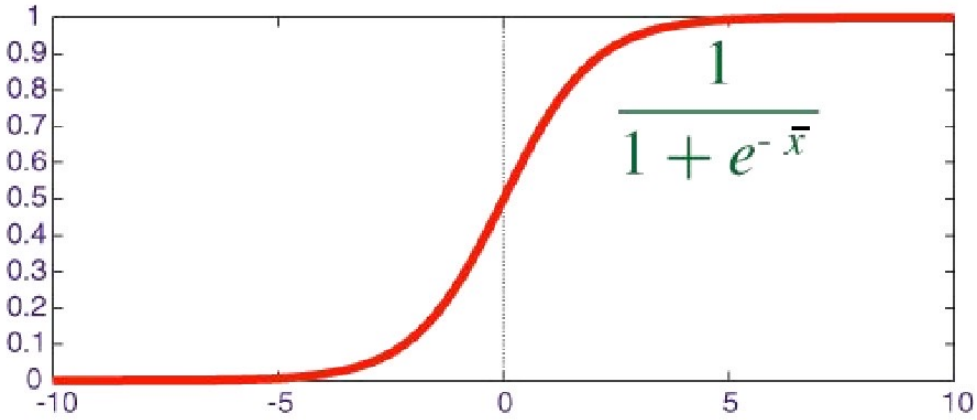
Part 1: Logistic regression (review)

- Logistic regression learns 2 classes

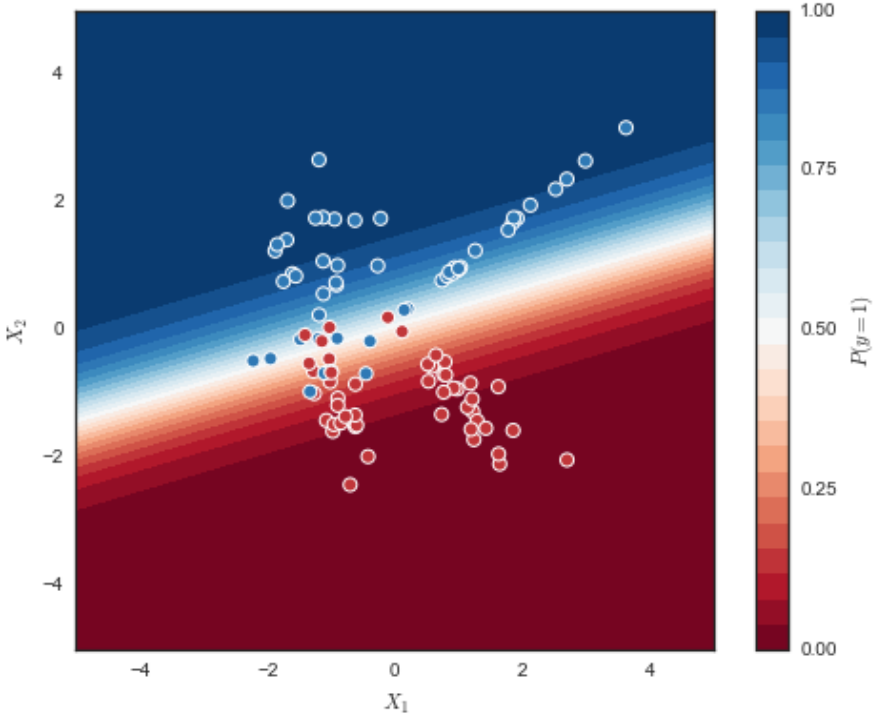
- The key is using the sigmoid function. This maps R → [0,1].

- A *dot-product* gives the value for sigmoid: $\bar{x} = w^T x + b$

Sigmoid function



One view of decision boundary

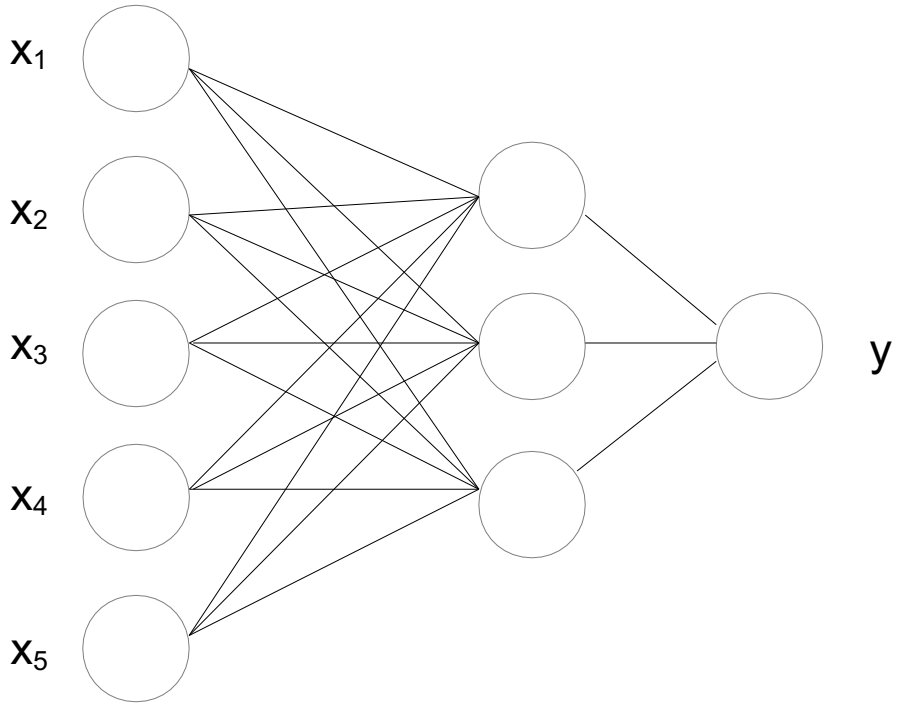


Multinomial logistic regression as a “neural network”

Imagine our data is 5-dimensional and we have three classes.

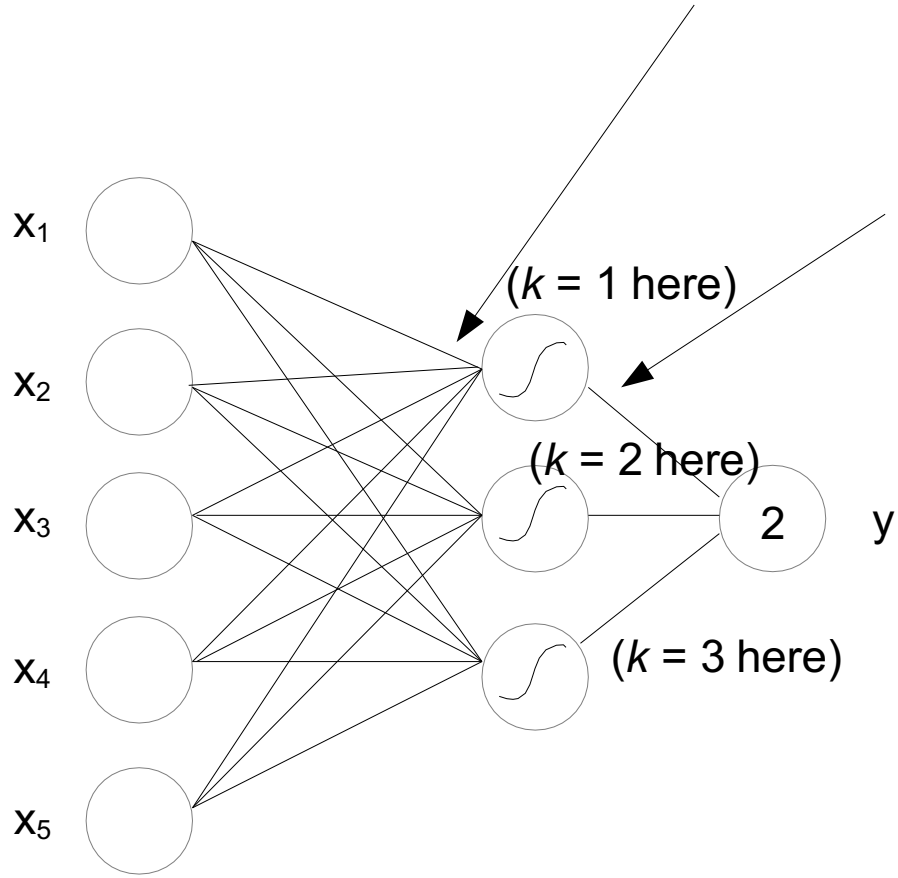
We can visualize this multinomial logistic regression problem graphically. This visualization is done to build NN intuitions.

This picture is for ***one*** input-output pair. The network operates by trying to map all inputs in the data to their outputs.



Multinomial logistic regression as a "neural network"

$$\hat{y}_k = W_{k,1}x_1 + \ldots + W_{k,5}x_5 + b_k$$



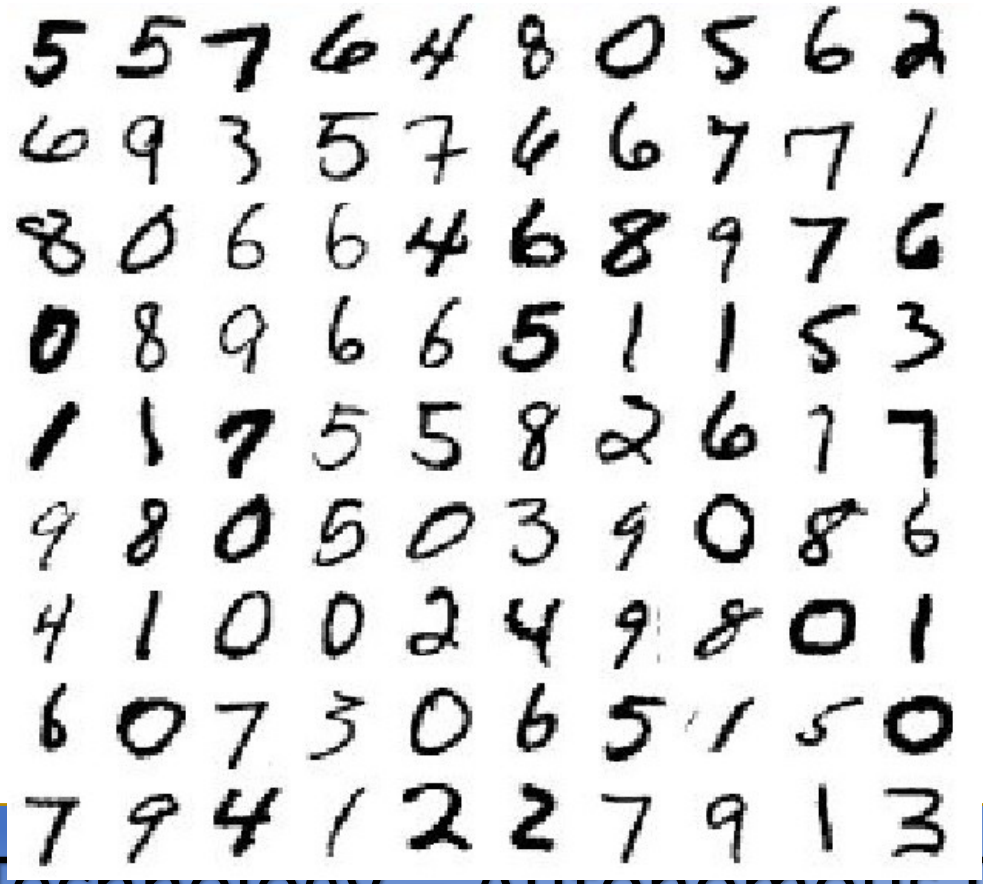
$$\Pr(\mathrm{y} = k|\hat{y}_k) = \sigma_k(\hat{y}_k)$$

This is repeated for all classes. Then a 3-sided dice is thrown to decide what y equals.

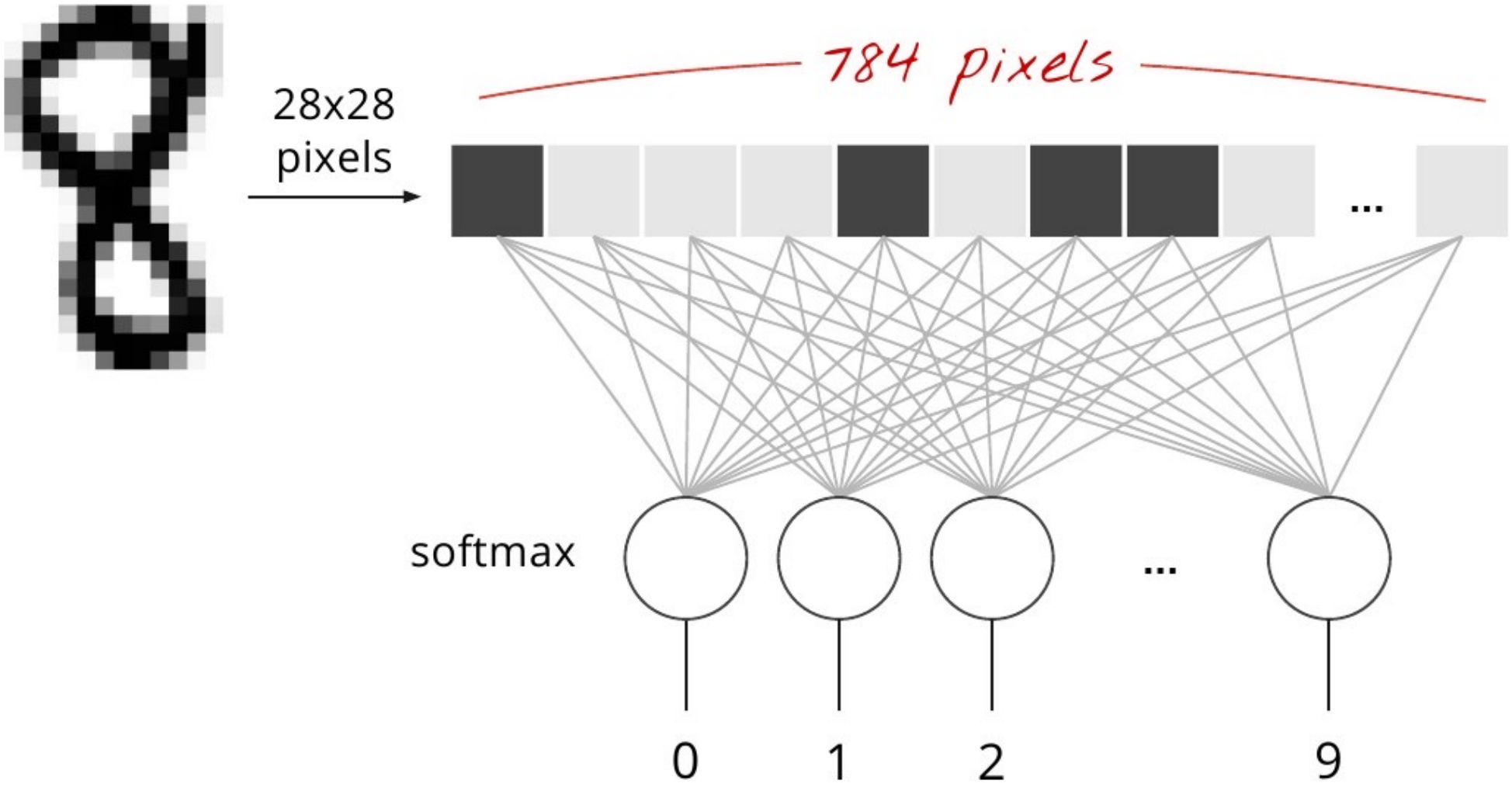
MNIST handwritten digits

A benchmark data set for many years. Good for illustrative purposes.

It contains 28 x 28 images, 60K to train the model, 10K to test.



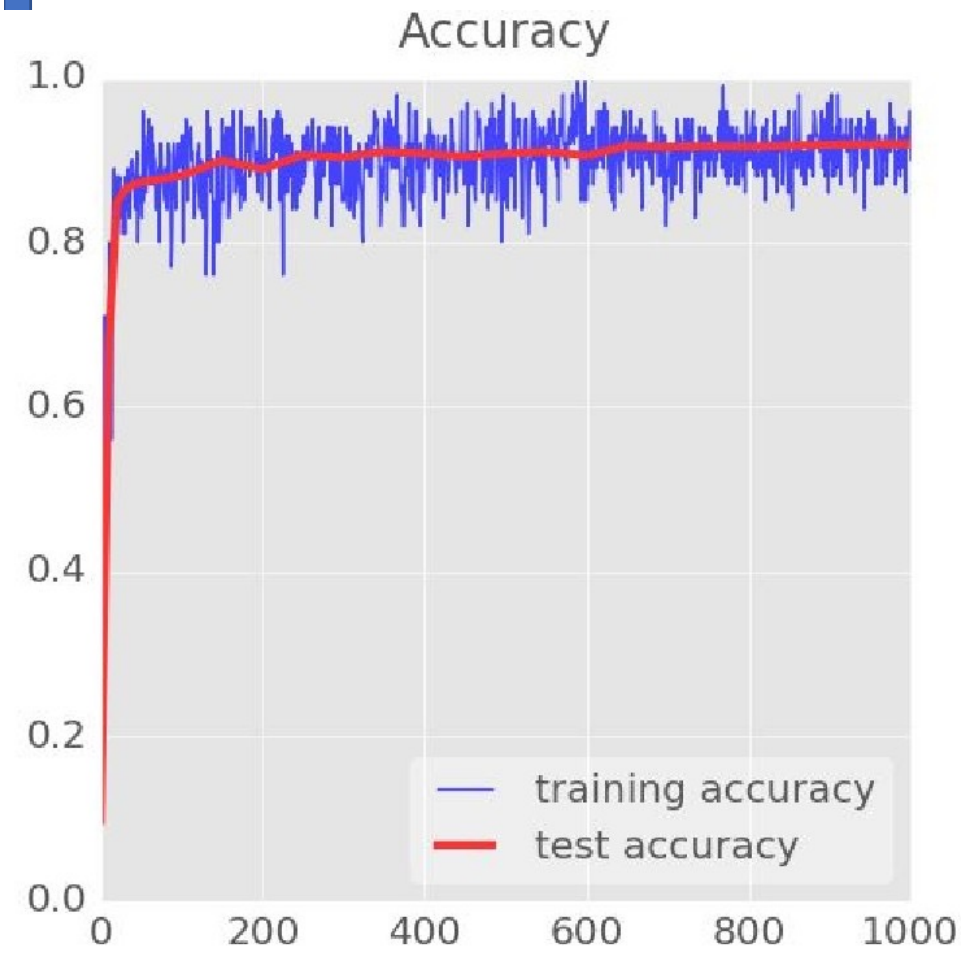
MNIST: a classification problem

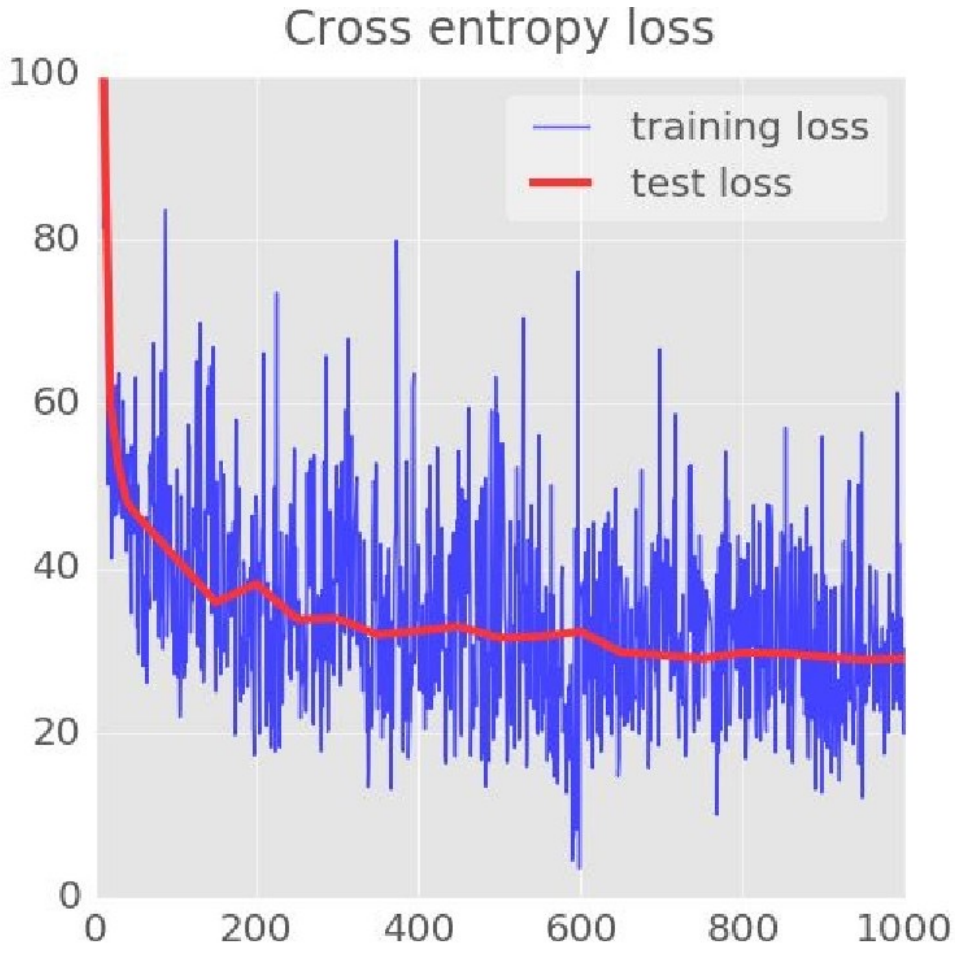


Learn 10 vectors (784 dims) and 10 biases to classify digits.

Left: The accuracy on new data (about 92%)

Right: The objective function we're minimizing



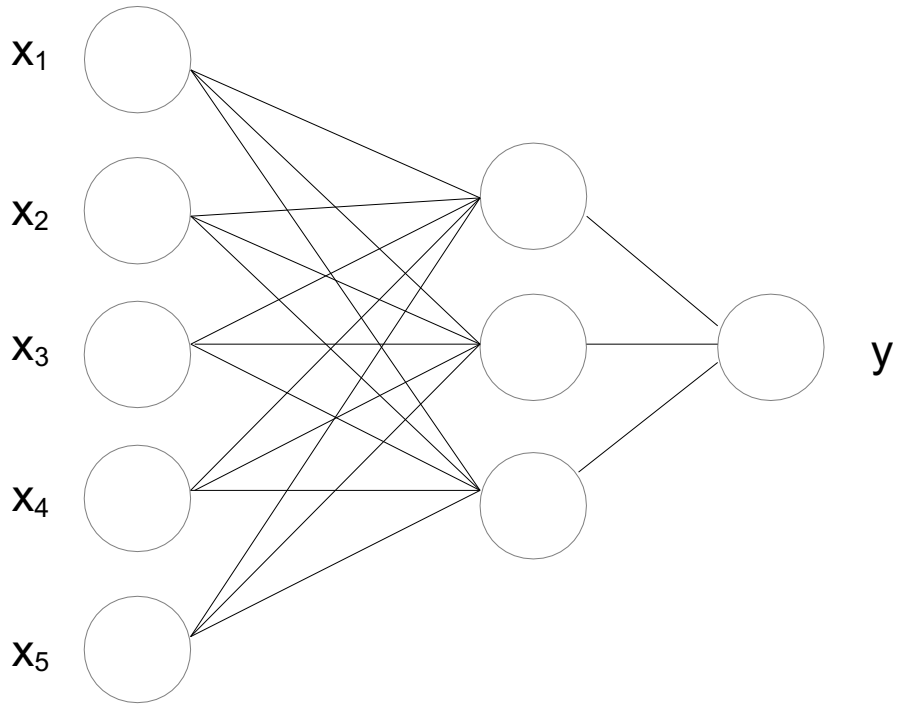


Building blocks of a deep learning model

Part 3: Multilayer perceptron

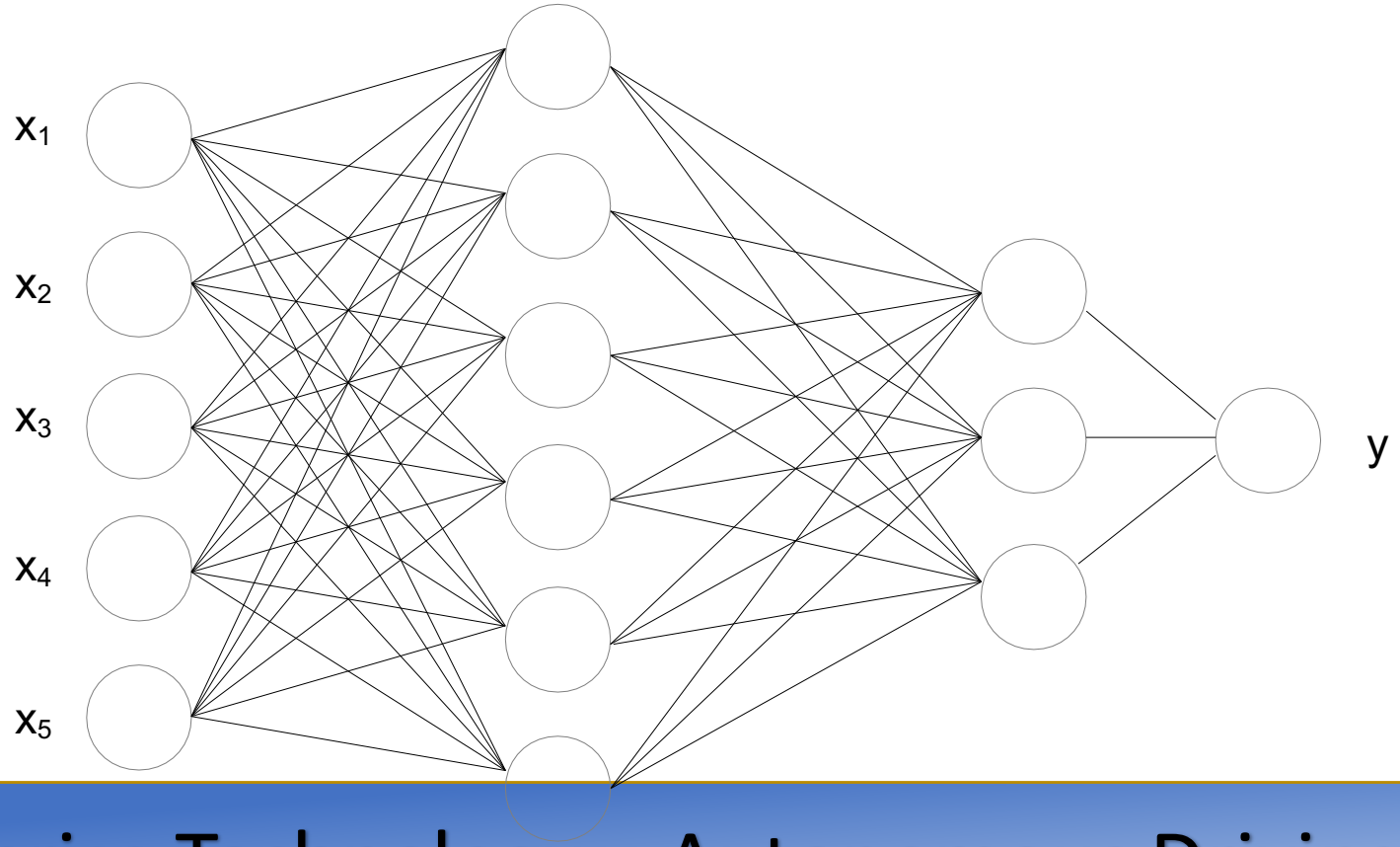
Making multinomial logistic regression a neural network

- The basic idea behind neural networks is to expand the multinomial logistic regression to include "**hidden layers**".

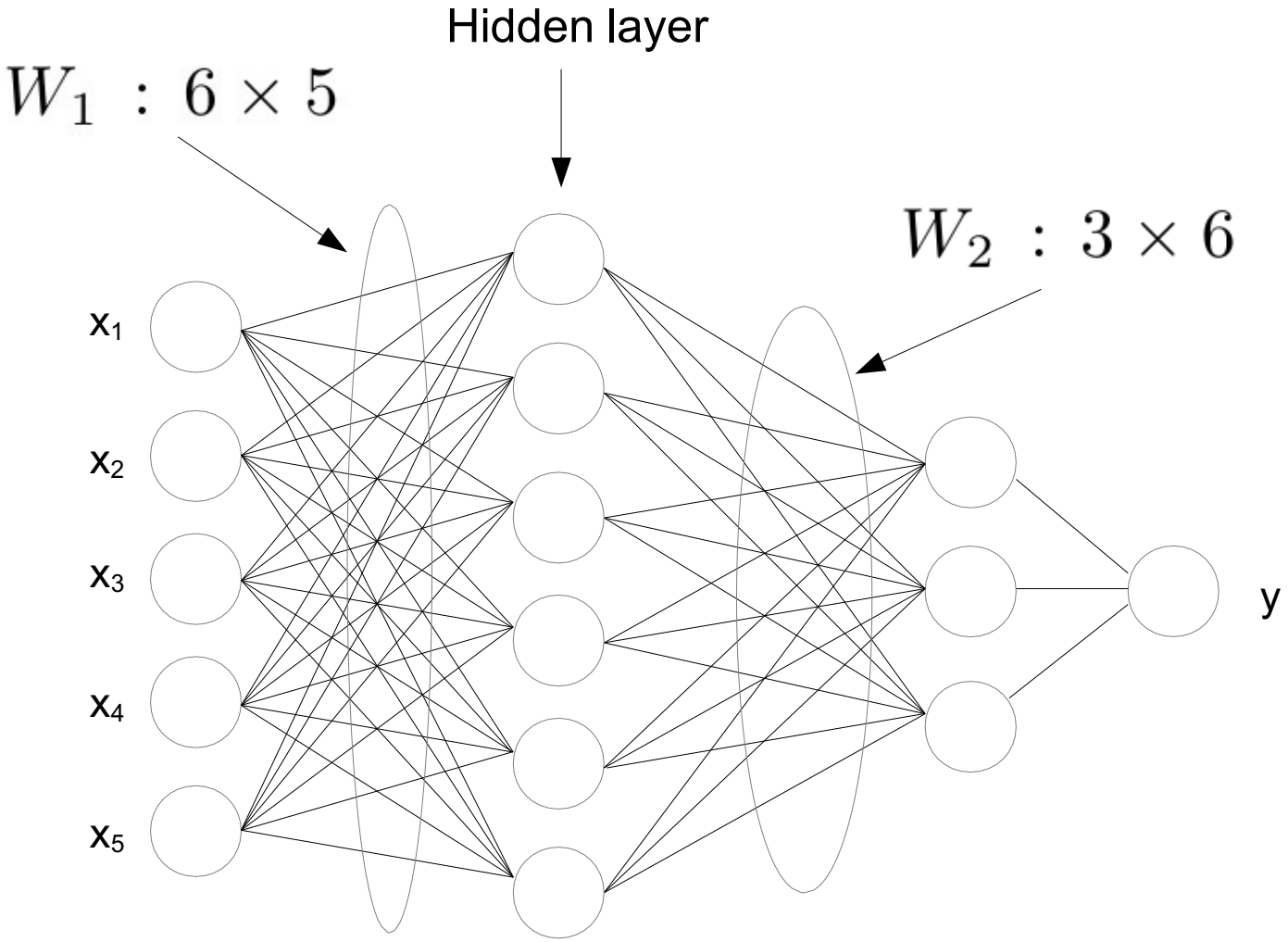


Making multinomial logistic regression a neural network

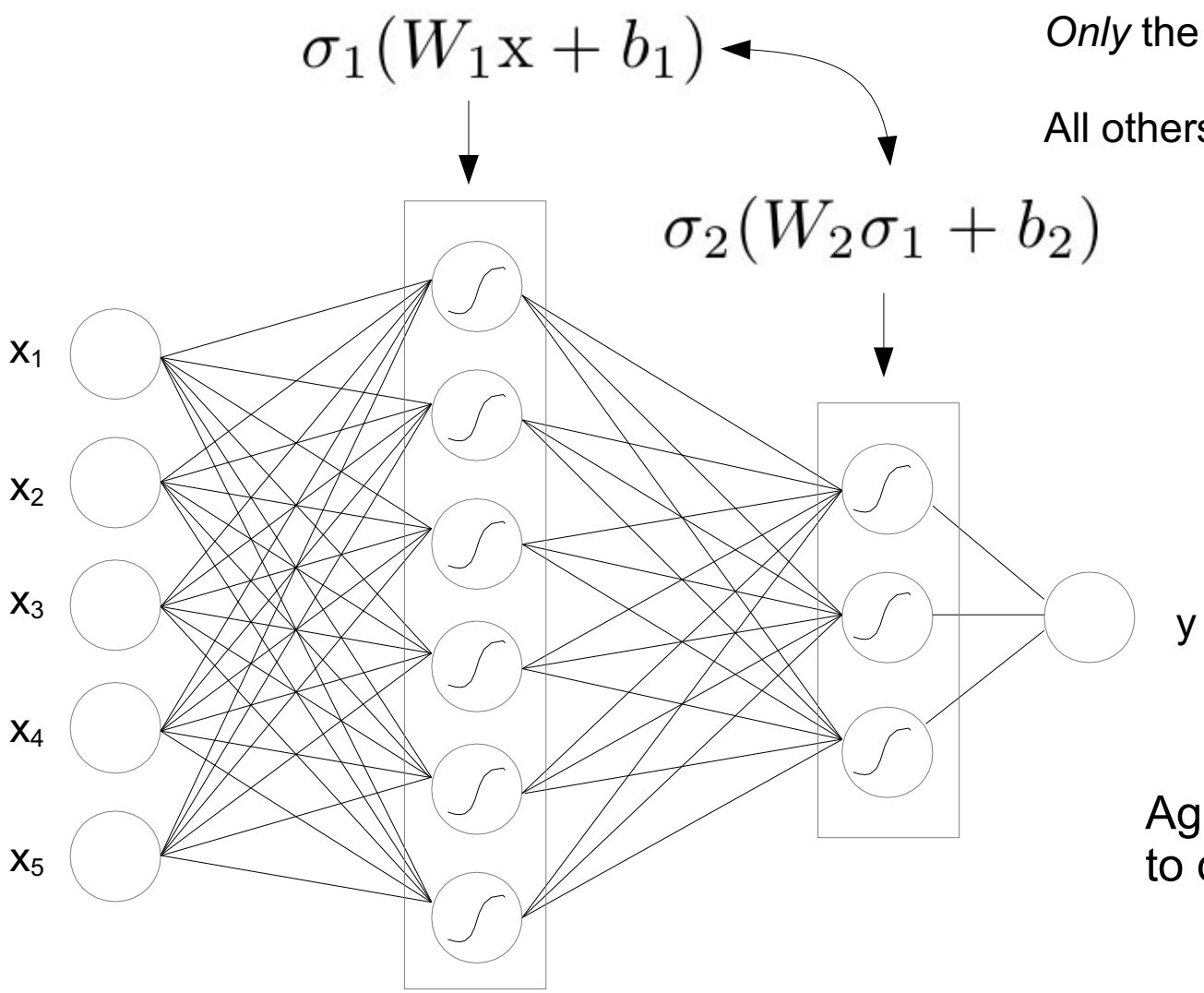
- The basic idea behind neural networks is to expand the multinomial logistic regression to include "**hidden layers**".



Basic neural networks (most subscripts now indicate layers)



Basic neural networks (most subscripts now indicate layers)



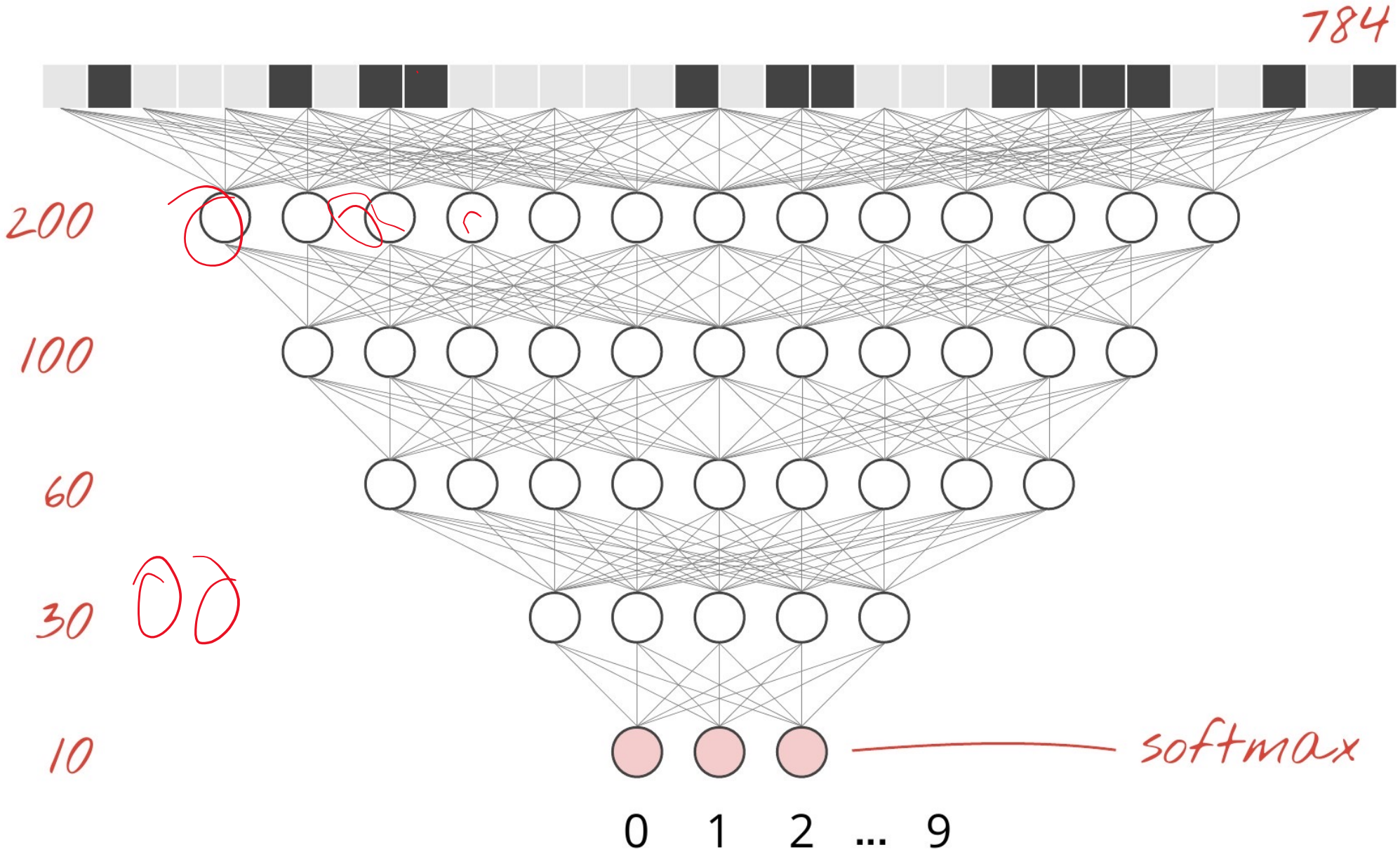
Alert: Notation overload!

Only the last layer is softmax.

All others are: $\sigma(a) = \dfrac{e^a}{1 + e^a}$

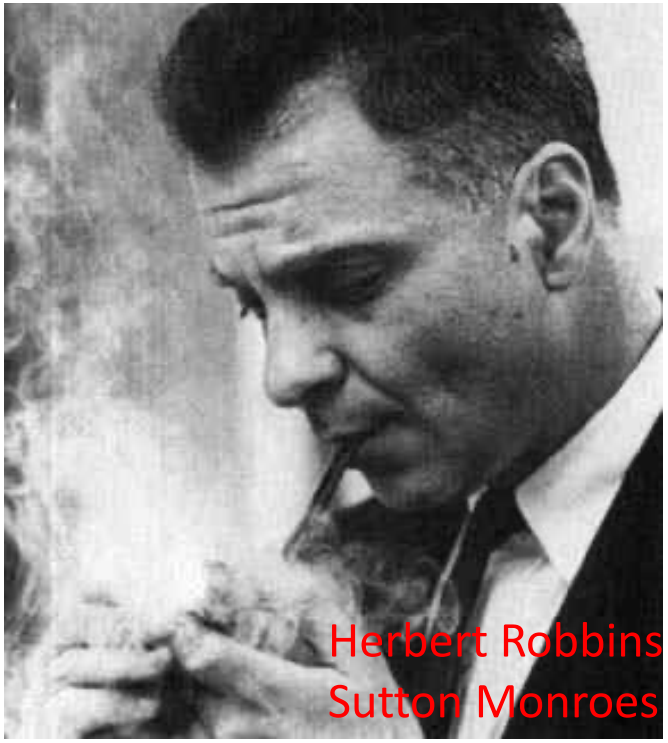
Again throw a 3-sided dice to decide what y equals.

MNIST revisited

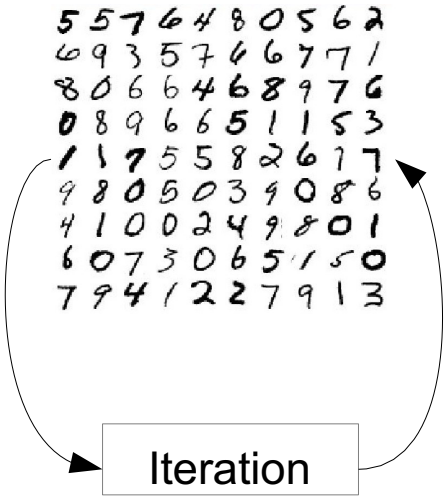


Technical problems: big data and step sizes

- There are several technical challenges. The first is scalability.
  - Computers are fast, but learning still slow when processing all data.
  - Fix: use **stochastic optimization** with a decreasing **step size**.
  - One pass through data = one "**epoch**". Make a few passes.

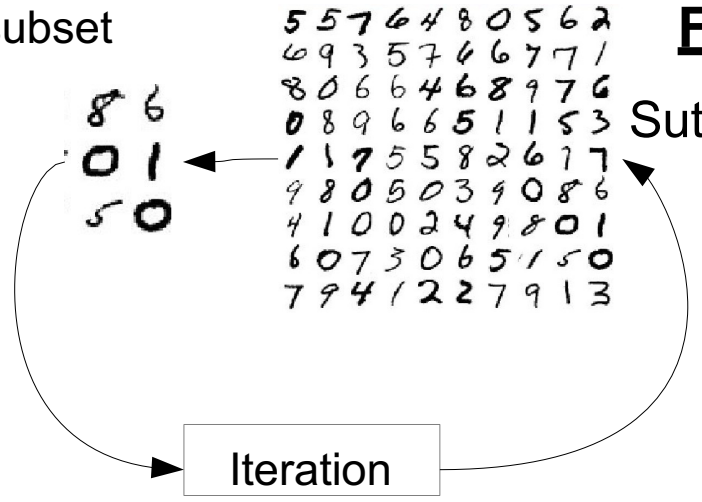


Slow



$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{D})$$

New subset



Fast

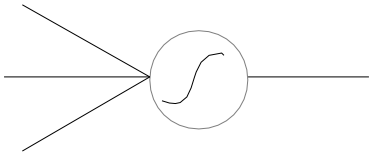
Sutton Monro

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{D}_{\text{subset}})$$

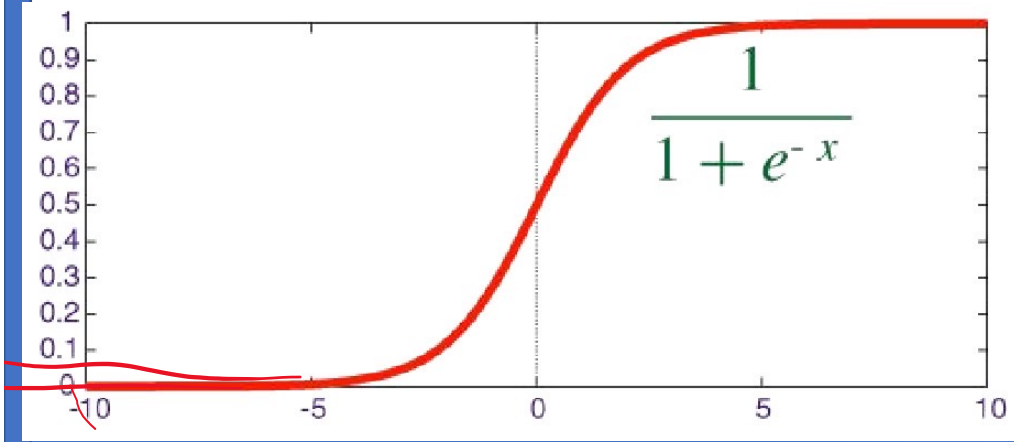
Technical problems: gradient vanishing

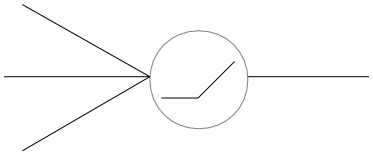
Sigmoid functions in models that are this deep lead to "**gradient vanishing**"" (The first derivatives for some parameters equal zero.)

The fix is "**ReLU**". Just replace internal sigmoids with ReLU as the activation function.

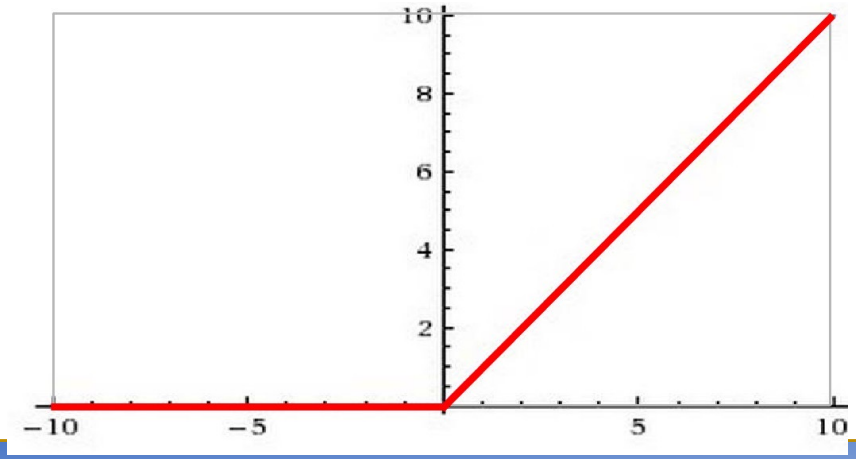


Sigmoid function





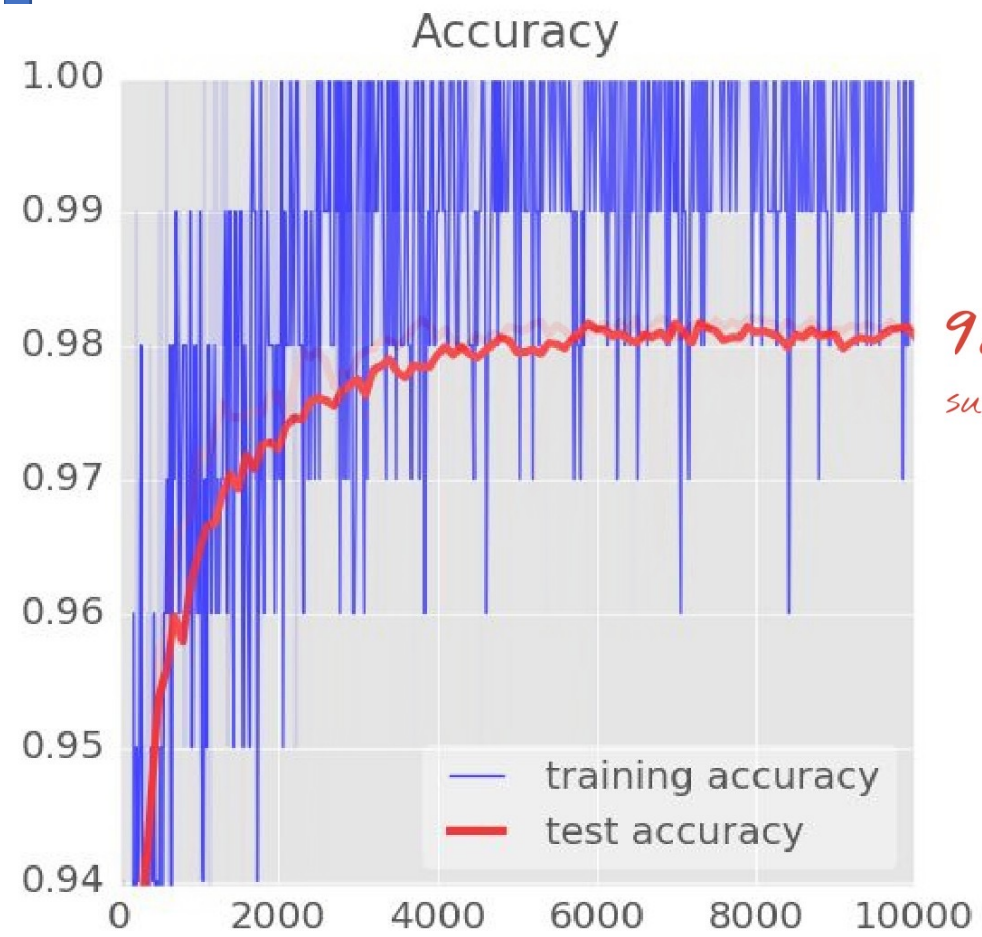
ReLU: Rectified Linear Unit

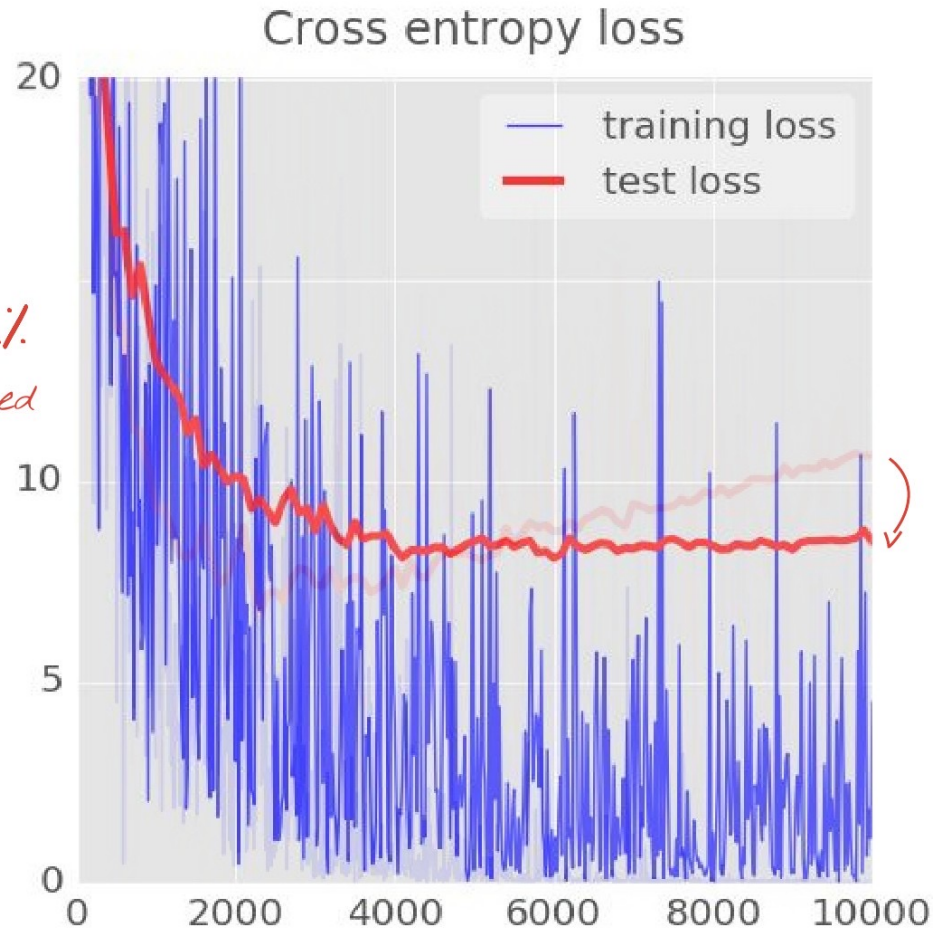


Technical problems:

- There are other problems that have been addressed.
- Other important techniques are:
- **Batch normalization**: Fixes data and parameter scaling issues
- **Back propagation**: The way for taking efficient derivatives in NN's
- **Early stopping**: Stop learning based on “validation set” performance **etc, etc, ...**

Specs: 4 hidden layers (200,100,60,30).
Roughly 34 million parameters
Left: The accuracy on new data (> 6% improvement)
Right: The objective function we're minimizing

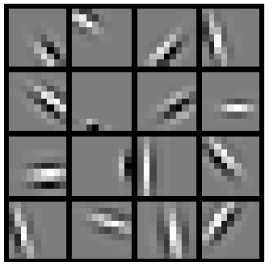




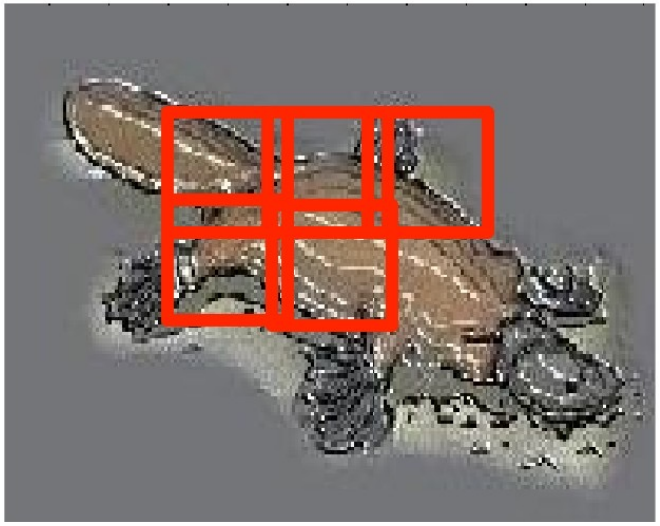
Building blocks of a deep learning model

Part 4: Convolutional neural networks (CNN)

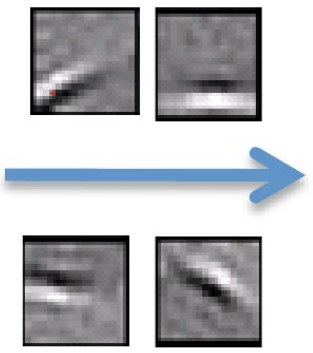
CNN: Deep learning with images



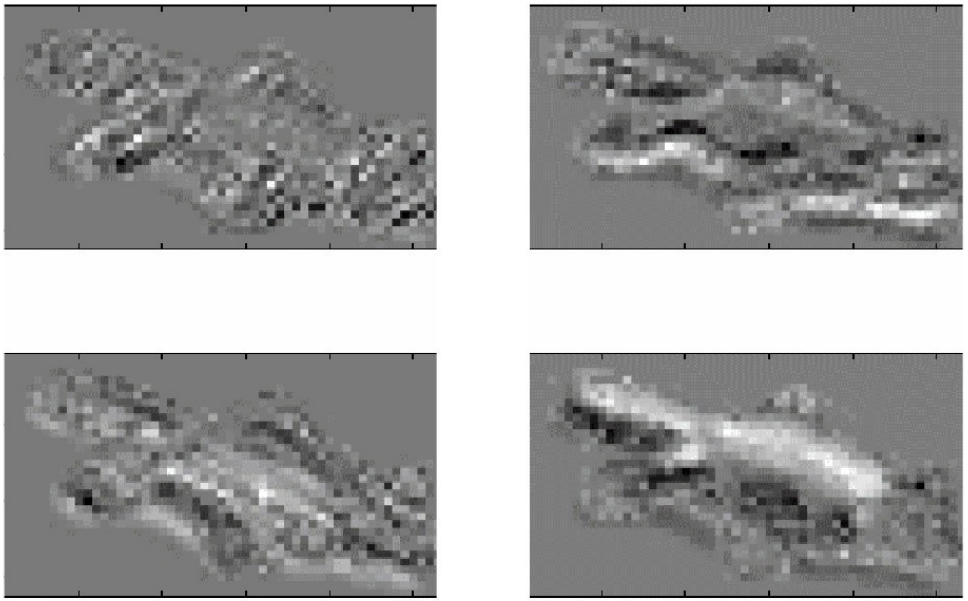
- A clear drawback of this approach with MNIST is that we removed structural information from the image (by vectorizing).
- **Convolutional neural networks** modify the previous framework by defining some layers to be convolutions using learned filters.



Input Image



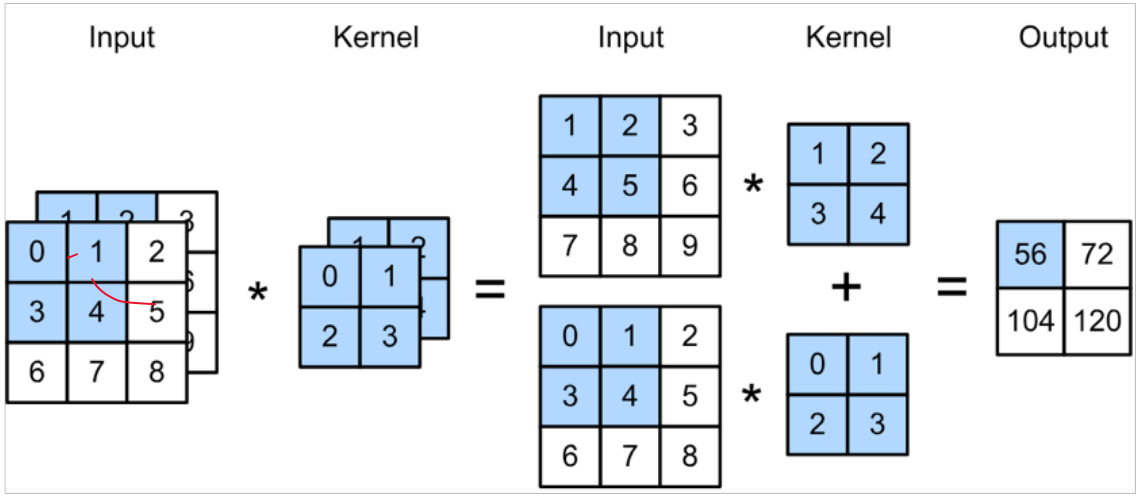
Learned bases



Features (coefficients)

Multiple Input Channels

- Have a kernel for each channel, and then sum results over channels



$$(1\times 1+2\times 2+4\times 3+5\times 4)$$
$$+(0\times 0+1\times 1+3\times 2+4\times 3)$$
$$=56$$

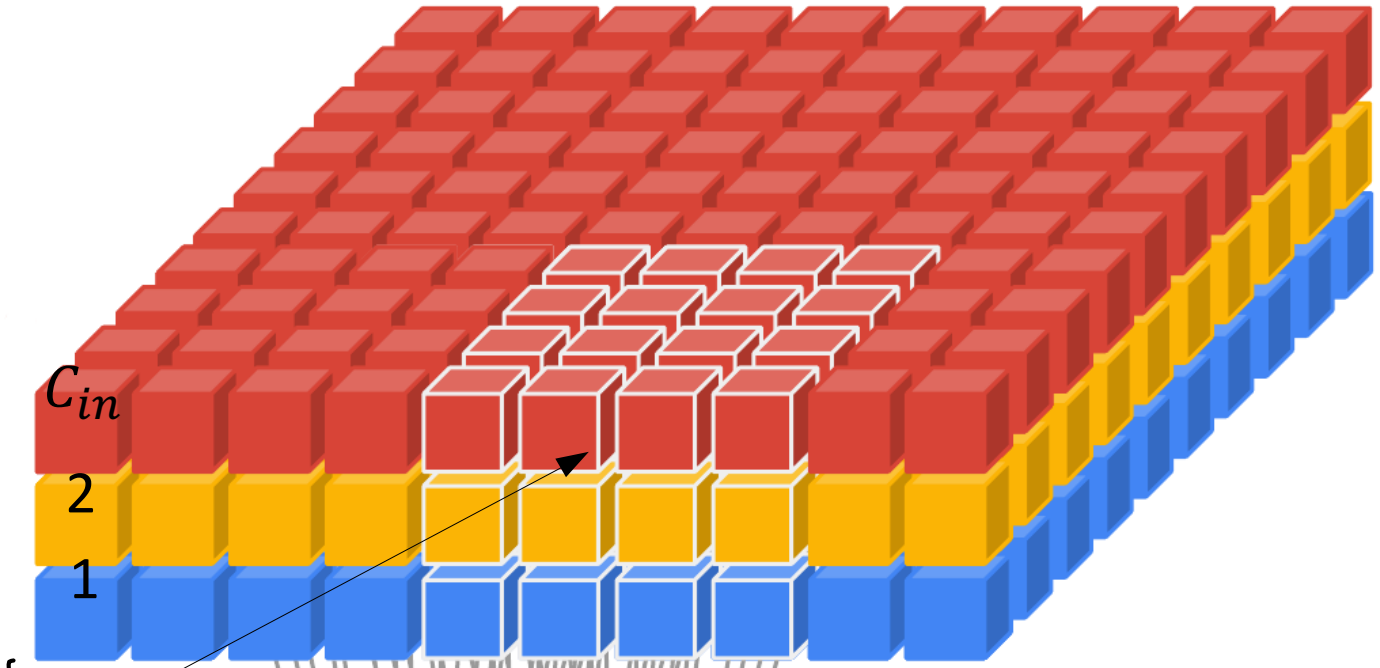
x

Multiple Input/output Channels

- No matter how many inputs channels, so far we always get single output channel
- We can have multiple 3-D kernels, each one generates a output channel
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

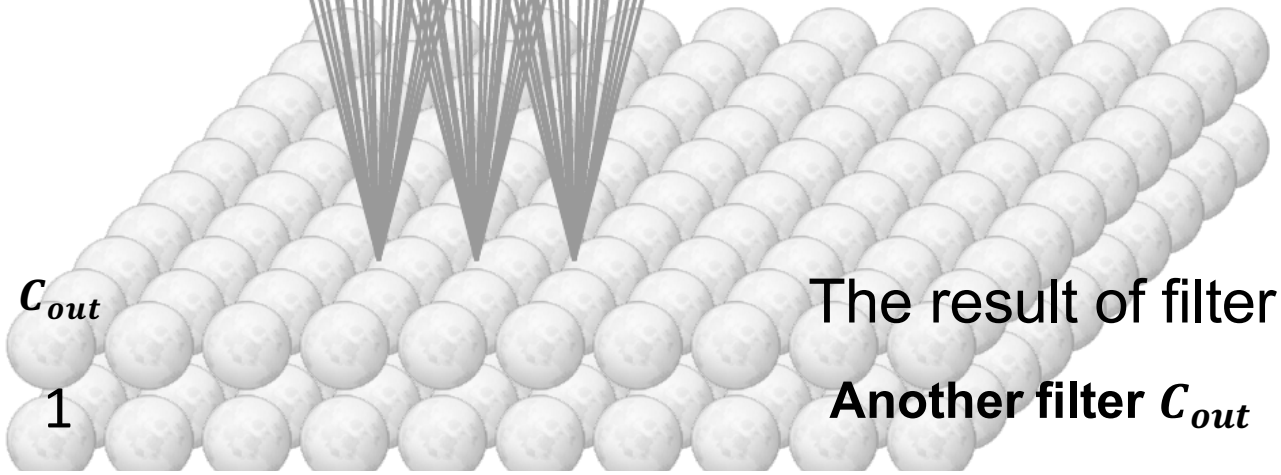
$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

for $i = 1, \ldots, c_o$



3x4x4X2 filter (2D)

3 Feature Maps

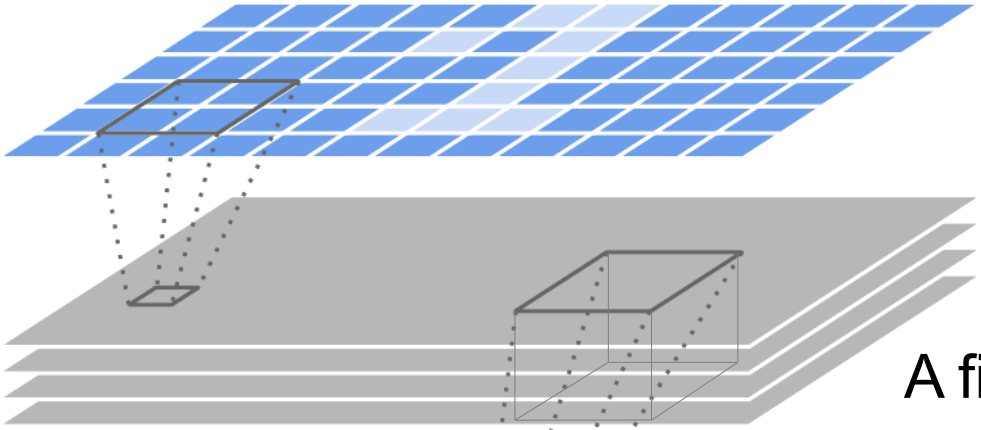


The result of filter 1

Another filter C_{out}

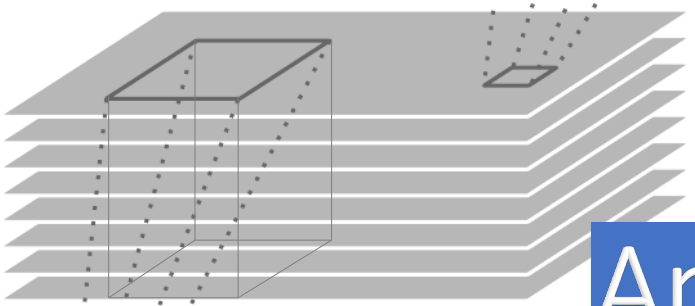
Sun

28x28x1



28x28x4

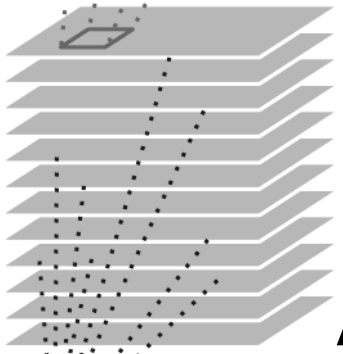
A first layer of filters



14x14x8

A second layer of filters

An MNIST image



7x7x12

A third layer of filters
(treat this as 1176-D vector)

A "fully connected" layer
(as before)

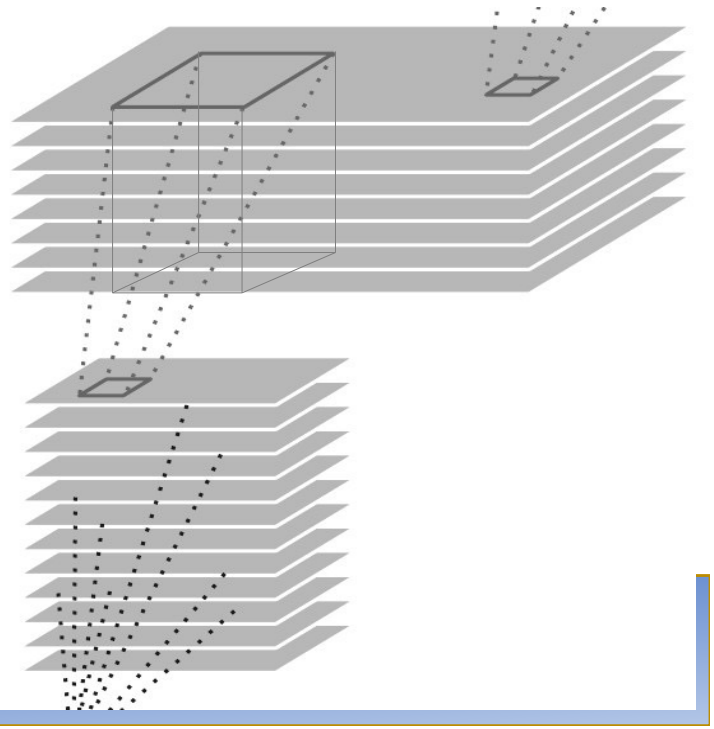
200

10

Softmax probability of label

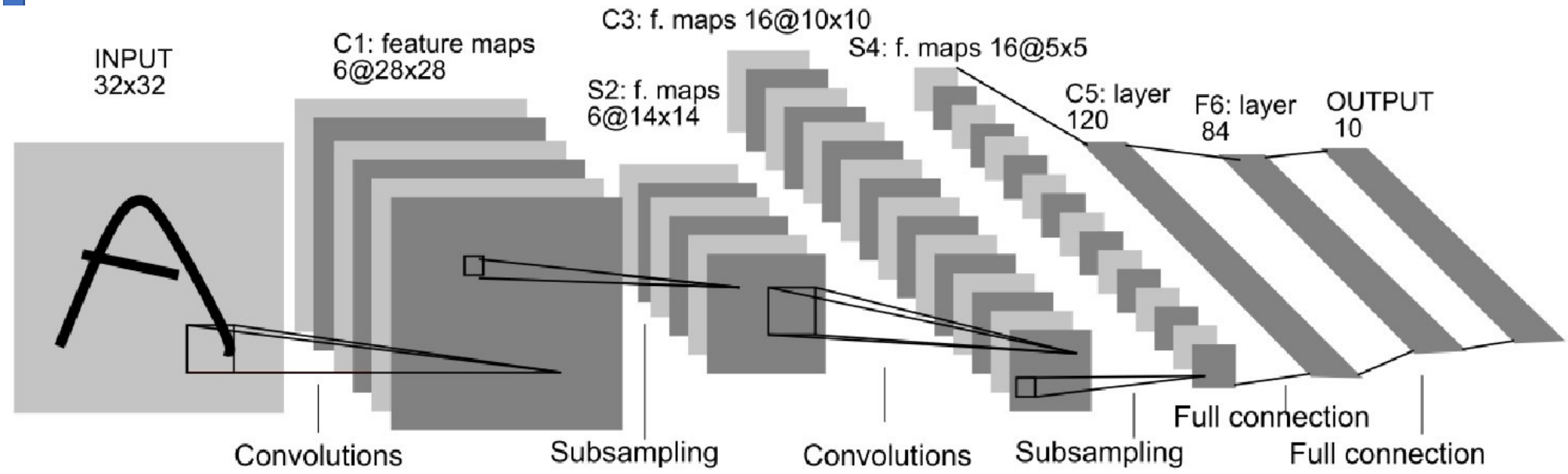
How did we go from 14x14x8 to 7x7x12?

- We used 12 different filters of size 4x4x8. But with correct padding, this should give 14x14x12 on the bottom row.
- Two common techniques are:
  - **Max-pooling:** Here, pick max of 2x2 windows
  - Change "**stride**": shift filters by 2



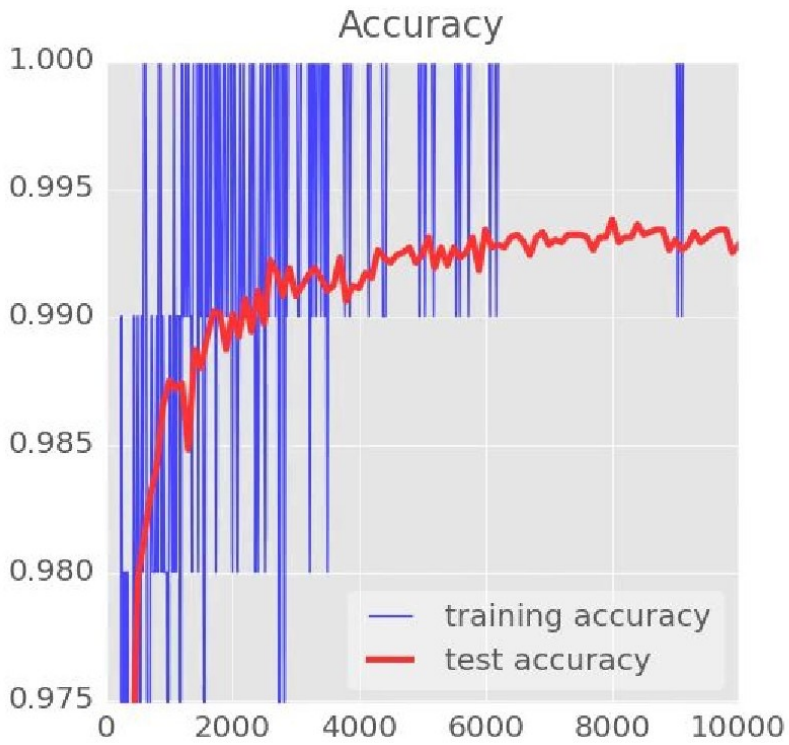
LeNet (Yann LeCun, 1998)

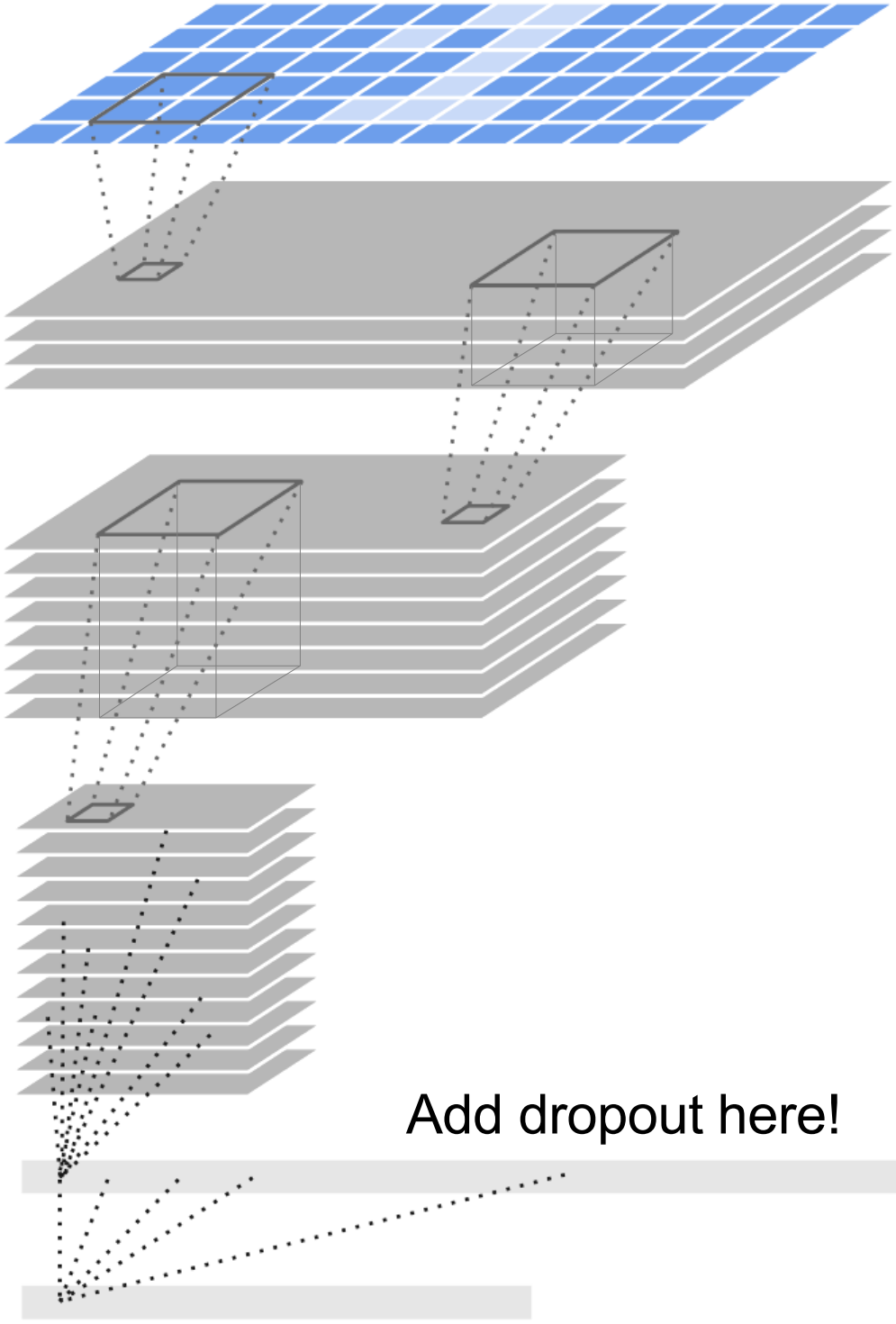
An early CNN
Obtained roughly 98.9% accuracy on MNIST.
Better than 98.2% with “fully connected” neural net of previous slide.



Using the whole bag of tricks

Accuracy 99.3%

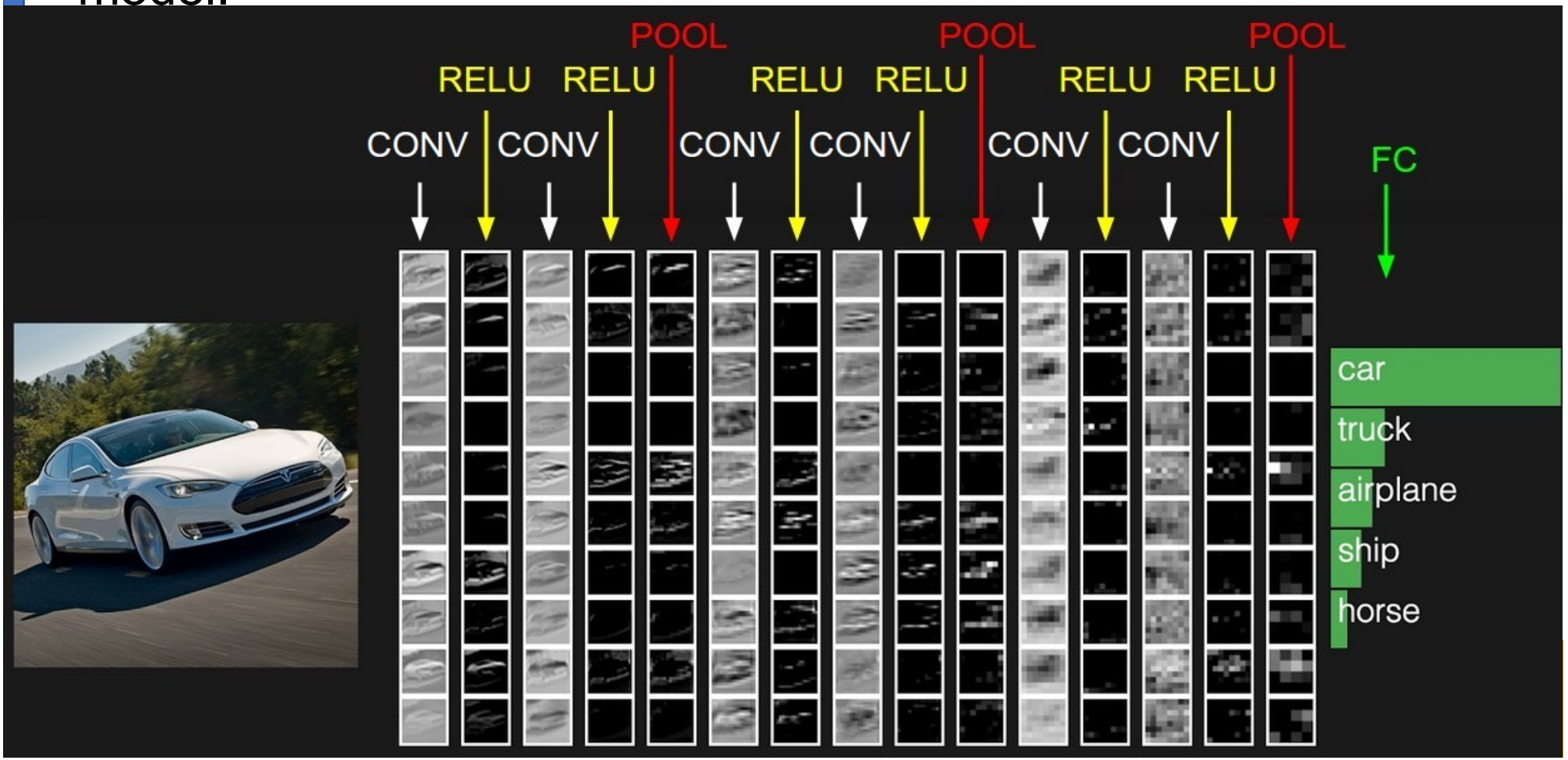




Add dropout here!

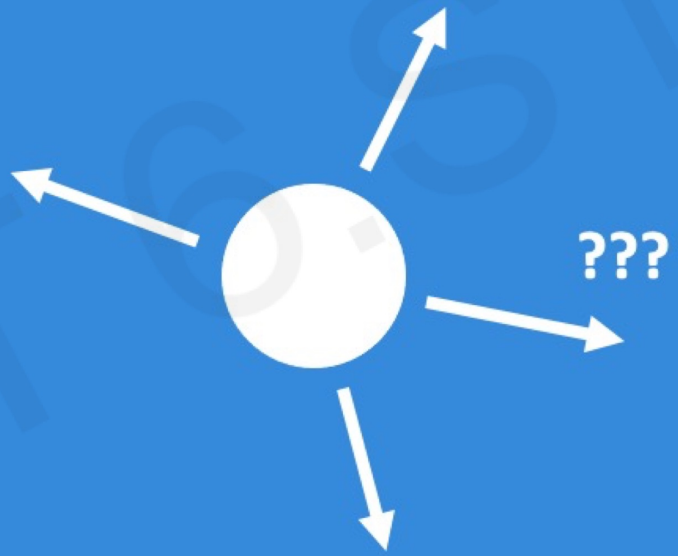
What is the CNN trying to do?

- The idea is to gradually “boil down” the visual information and transform input to intern representation for a fully-connected neural network, or perhaps a multinomial logistic regression model.



Given an image of a ball,
can you predict where it will go next?

Given an image of a ball,
can you predict where it will go next?

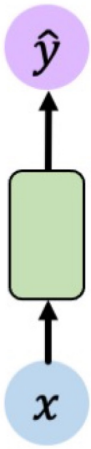


Given an image of a ball,
can you predict where it will go next?

Sequence Modeling Applications

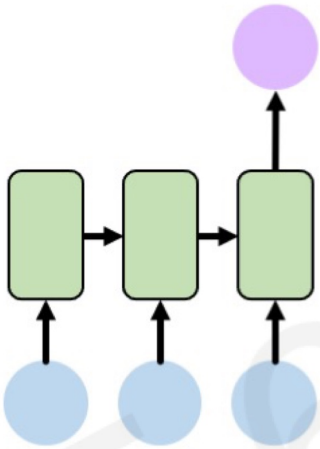
- Audio Signals

Sequence Modeling Applications

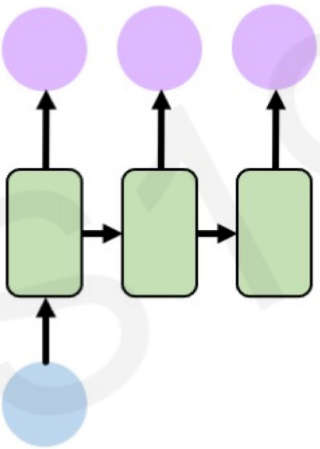


One to One
Binary Classification

"Will I pass this class?"
Student → Pass?

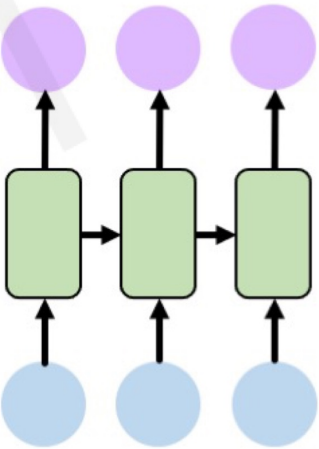


Many to One
Sentiment Classification



One to Many
Image Captioning

"A baseball player throws a ball."

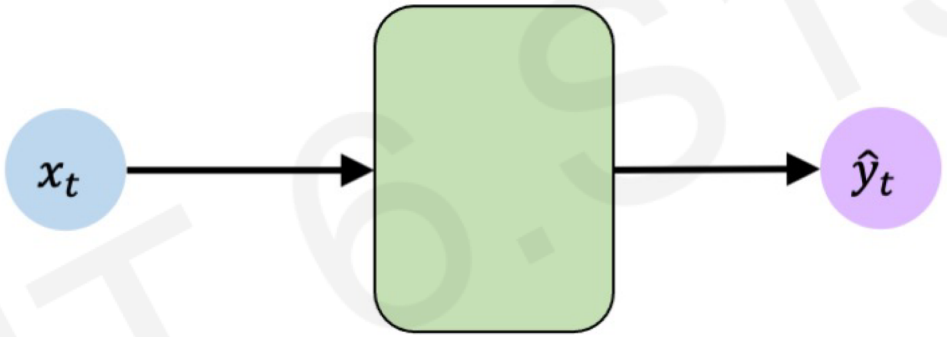


Many to Many
Machine Translation

Machine Translation
Google Translation

What is the difference between left and right?

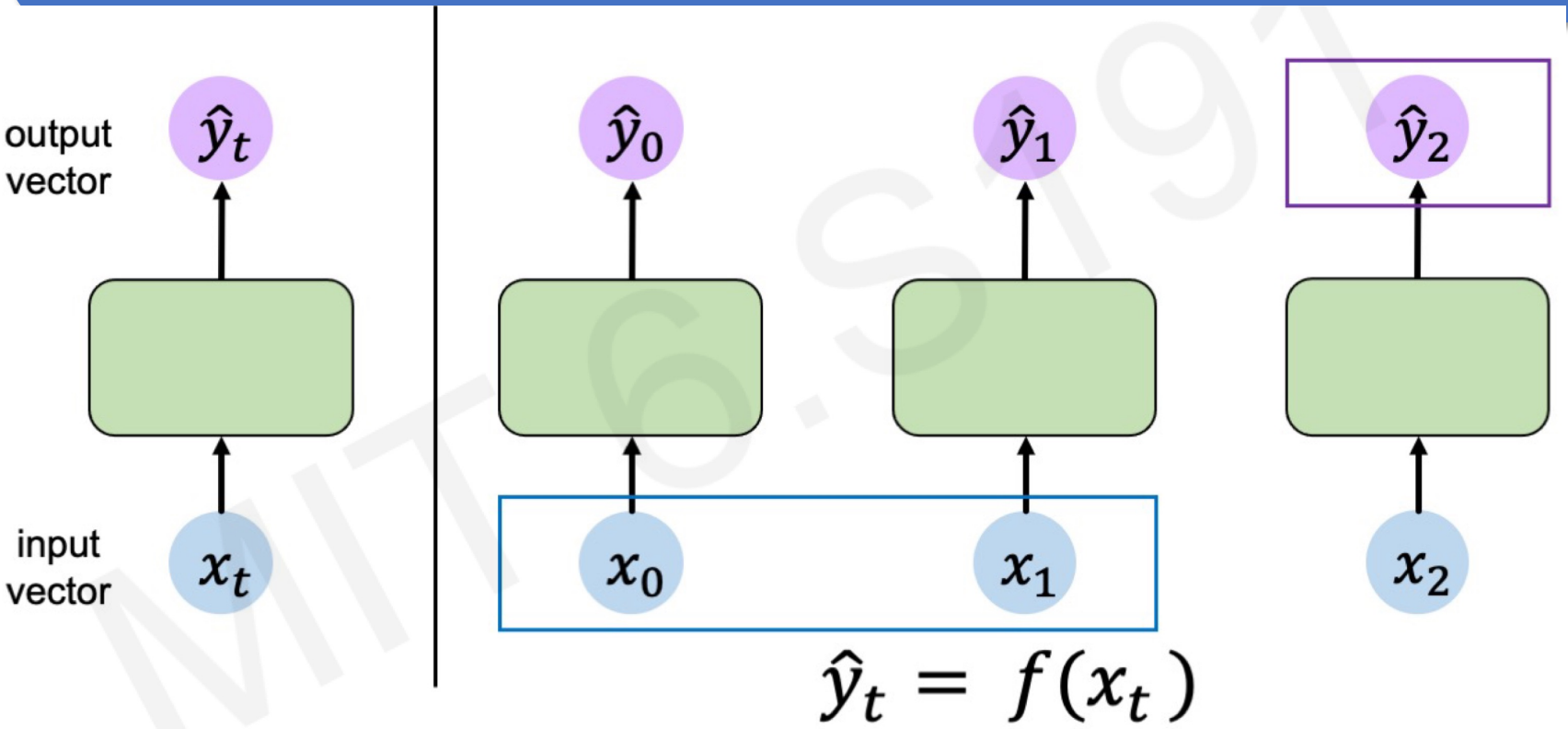
Feed-Forward Networks Revisited



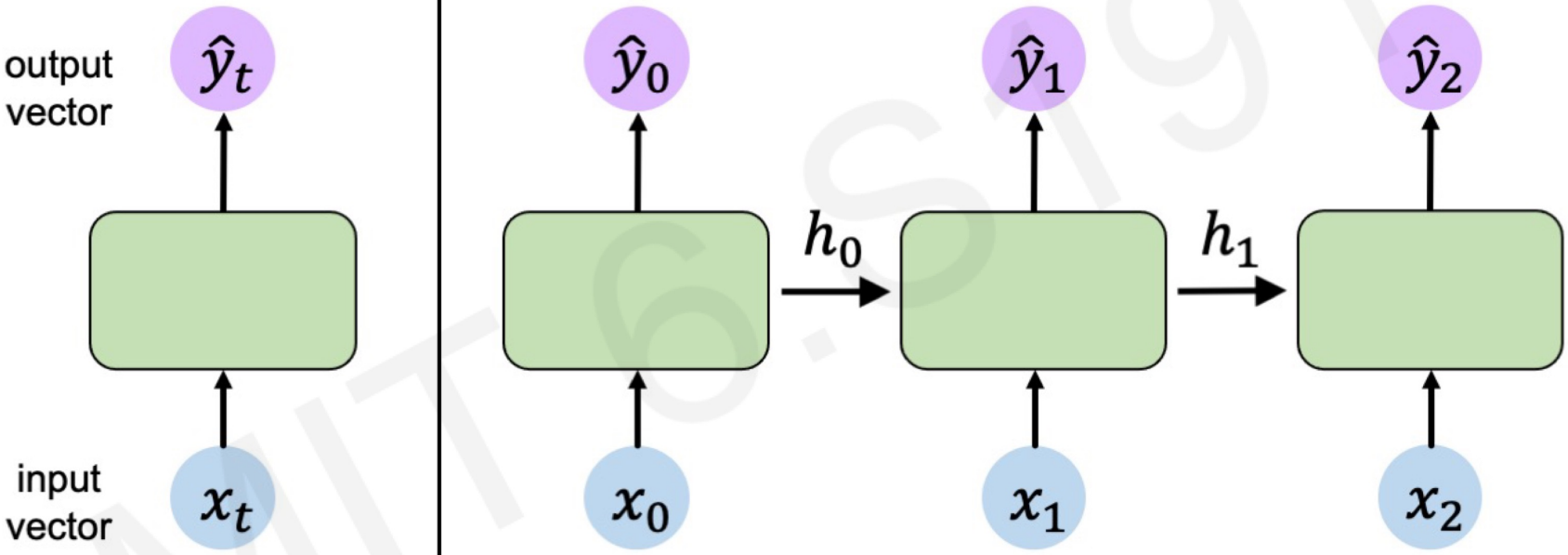
$\boldsymbol{x_t} \in \mathbb{R}^m$

$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

Handling Individual Time Steps

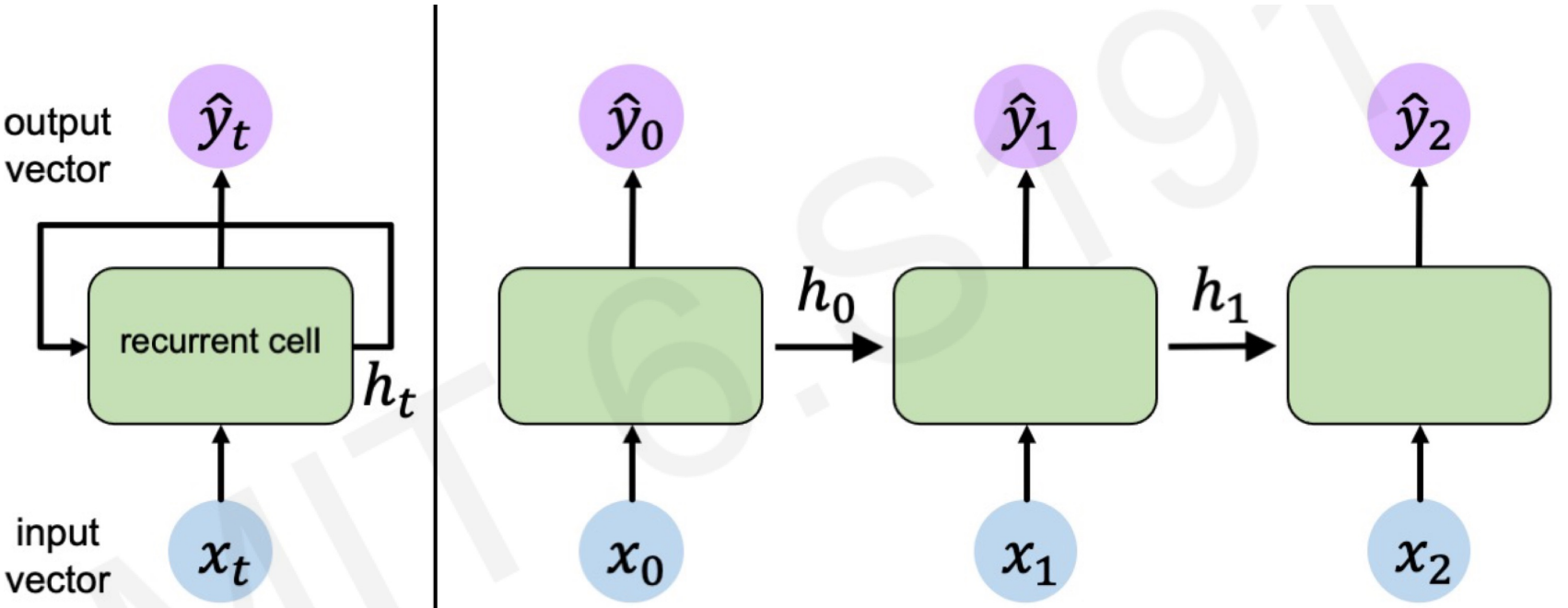


Neuron with Recurrence



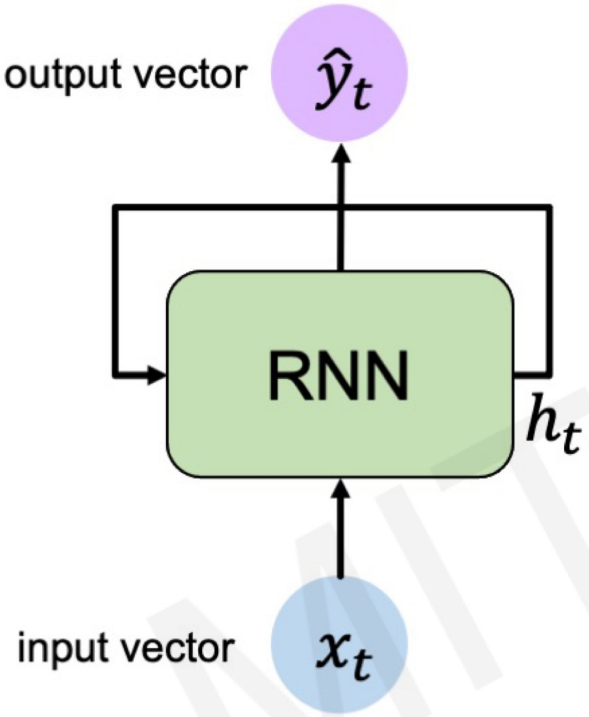
$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Neurons with Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks (RNN)



Apply a **recurrence relation** at every time step to process a sequence:

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

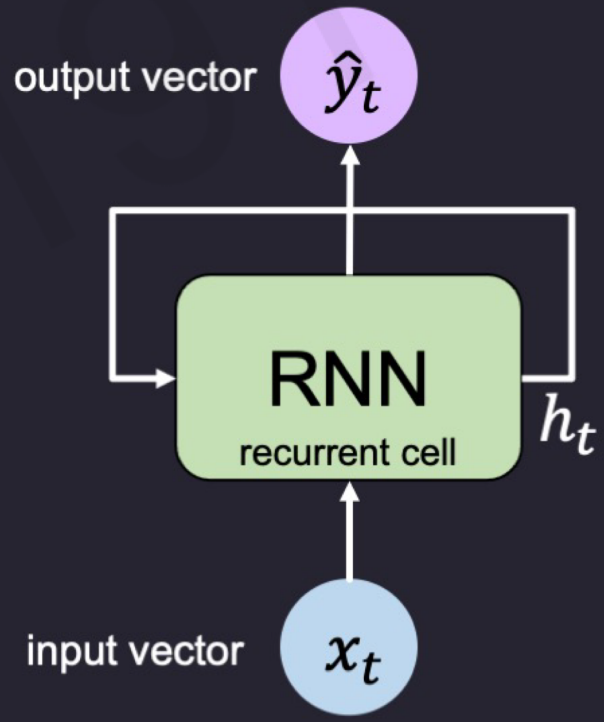
cell state | function with weights W | input | old state

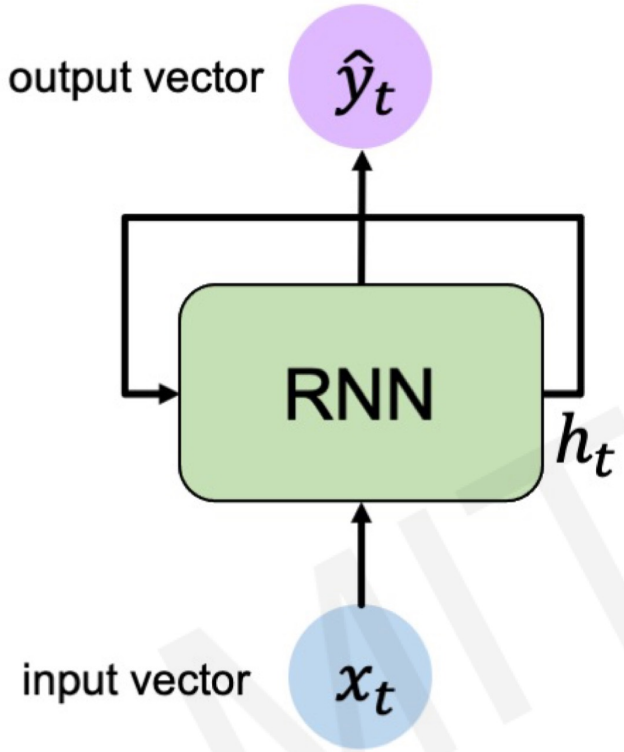
Note: the same function and set of parameters are used at every time step

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```





Output Vector

$$\hat{y}_t = \boldsymbol{W}_{hy}^{T} h_t$$

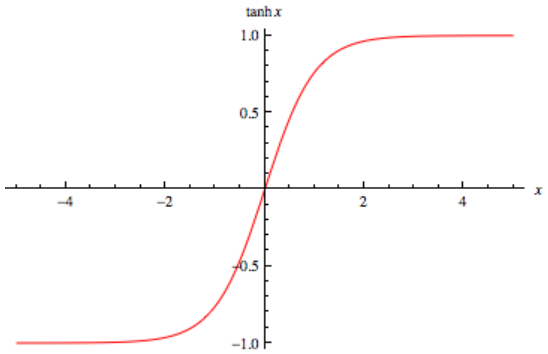
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^{T} h_{t-1} + \boldsymbol{W}_{xh}^{T} x_t)$$

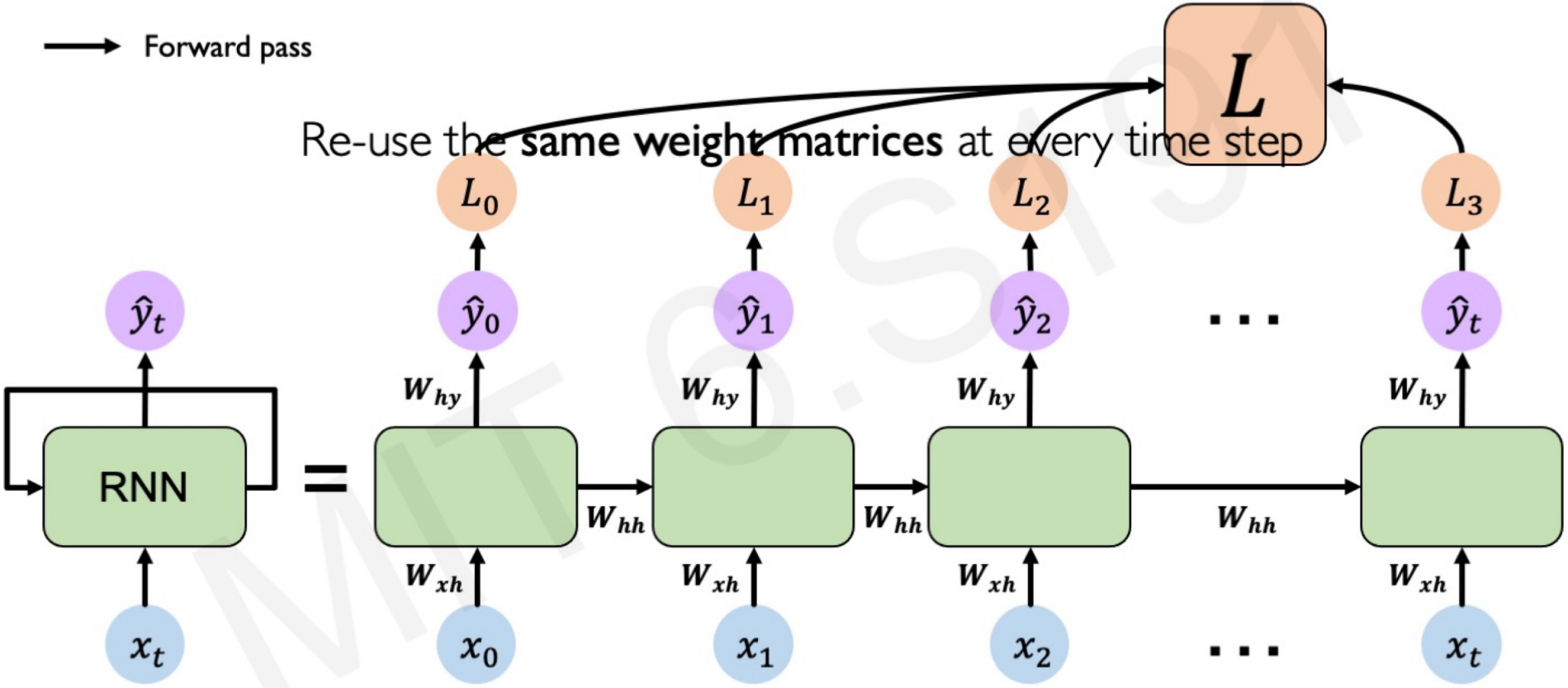
Input Vector

$$x_t$$

$$Tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

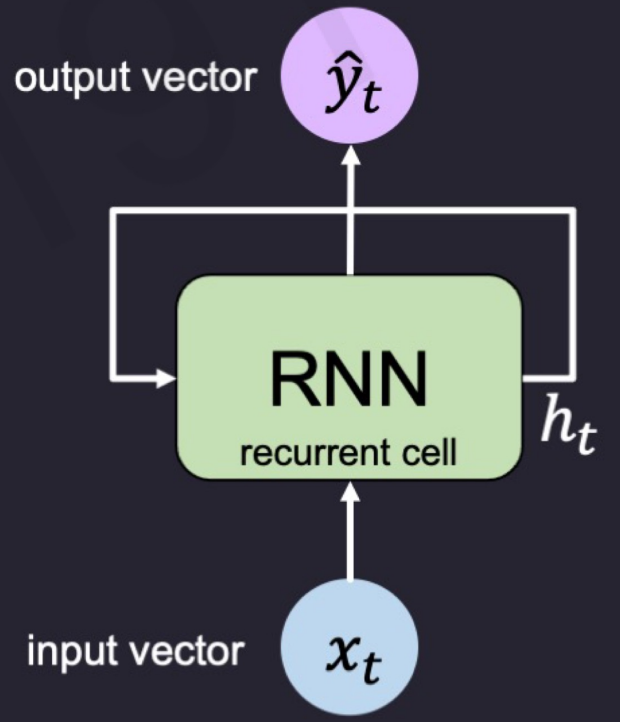


RNNs: Computational Graph Across Time



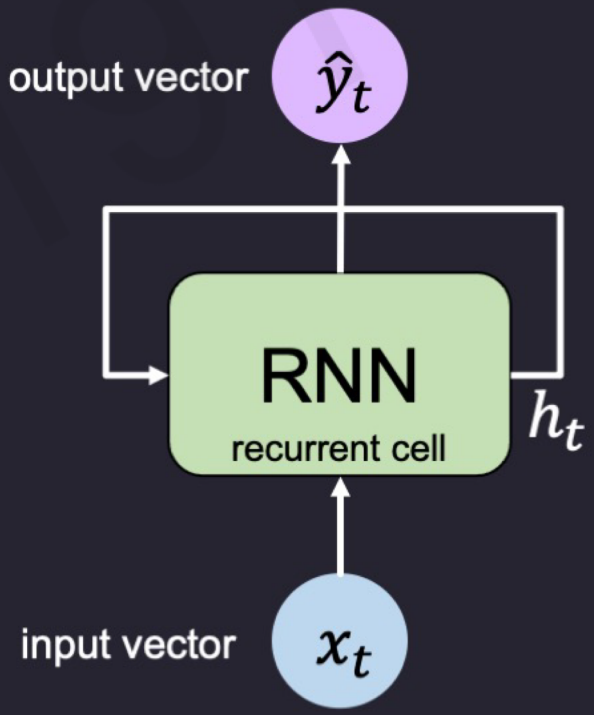
RNN Implementation

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```

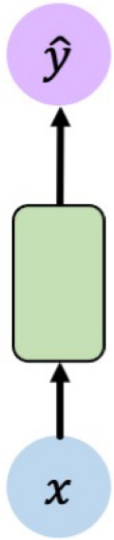


RNN Implementation in TensorFlow

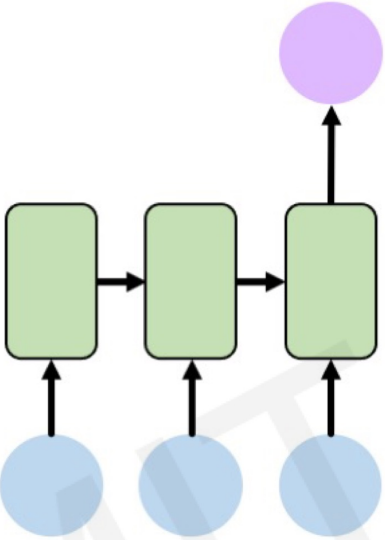
```
tf.keras.layers.SimpleRNN(rnn_units)
```



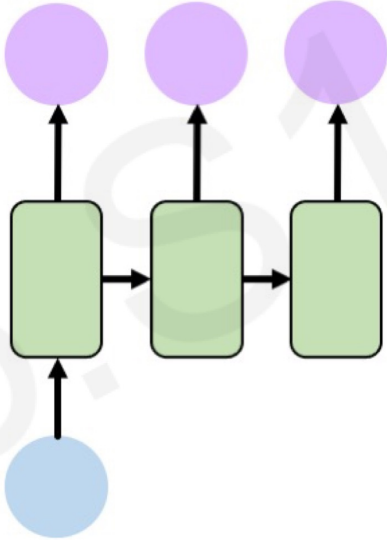
RNN for Sequence Modeling



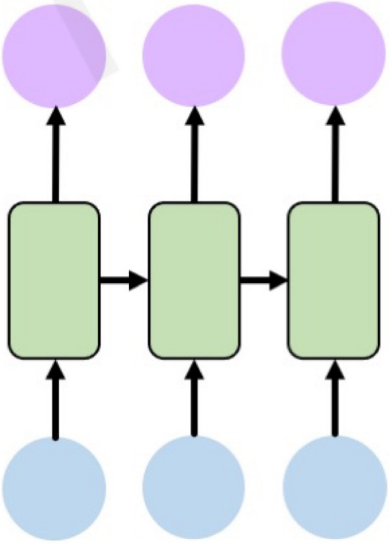
One to One
"Vanilla" NN
Binary classification



Many to One
Sentiment Classification



One to Many
Text Generation
Image Captioning

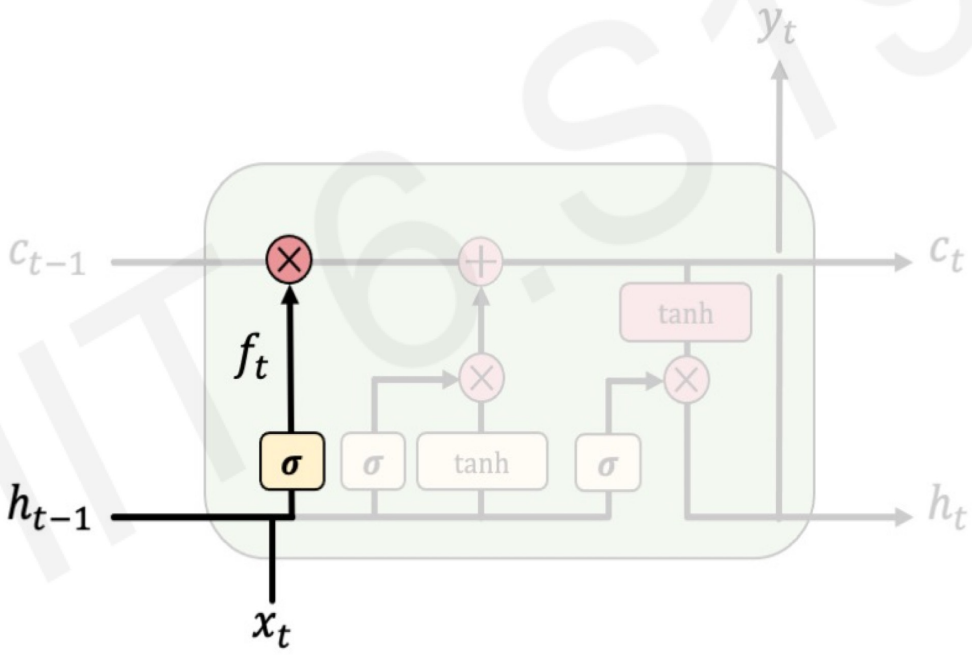


Many to Many
Translation & Forecasting
Music Generation

LSTM

1) Forget 2) Store 3) Update 4) Output

LSTMs **forget irrelevant** parts of the previous state



$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Take previous state information and forget some information from memory cell, it Modules how much should be forgot or kept in.

Other deep learning frameworks

- Autoencoders (AE)
  - unsupervised learning, representation learning
- Generative adversarial network (GAN)
  - try to “fool” a computer into thinking an algorithm is human
- Recurrent neural networks (RNN)
  - sequential data, especially language modeling and time series (Finance)

the two birds are trying to be seen in the water .

a giraffe is standing next to a fence in a field .

a parked car while driving down the road .

In general, a neural network performs better than more specially designed approaches.

Left: rainy image, Right: deep learning, Middle 2: other algorithms

- **Thanks!**