Week 03 Lecture 05
Artificial Neural Network

Wan Fang
Southern University of Science and Technology

Linear Regression

- $\mathbf{w}$
  - The ***weight*** determines the influence of each feature on our prediction, usually a vector form with w_i
- b
  - The ***bias*** says what value the predicted price should take when all features take 0
- Given a dataset, **our goal** is
  - To choose the weights $\mathbf{w}$ and bias b such that on average, the predictions made based on our model best fit the true prices observed in the data.

$$\hat{y} = w_1 \cdot x_1 + \ldots + w_d \cdot x_d + b \longrightarrow \hat{y} = \mathbf{w}^T \mathbf{x} + b.$$

Linear Regression

$$\hat{y} = w_1 \cdot x_1 + \ldots + w_d \cdot x_d + b \rightarrow \hat{y} = \mathbf{w}^T \mathbf{x} + b \rightarrow \hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

- Vectorization
  - All features into a vector **x** for a single data point
  - All weights into a vector **w**
  - Our entire dataset as the *design matrix* **X,** including one row for every example and one column for every feature

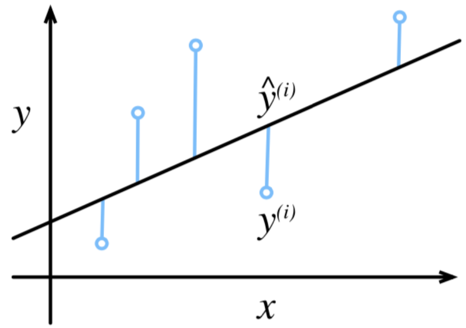
$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(i)} & \cdots & x_d^{(i)} \end{bmatrix}$$

one row for every example

one column for every feature

Loss Function

- To quantify the distance between the ***predicted*** and ***real*** value.
  - usually be a non-negative number where smaller values are better
  - perfect predictions incur a loss of 0
- The Sum of Squared Errors $l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$
  - the empirical error is only
    a function of the model parameters
- Loss Function as an averaged SSE

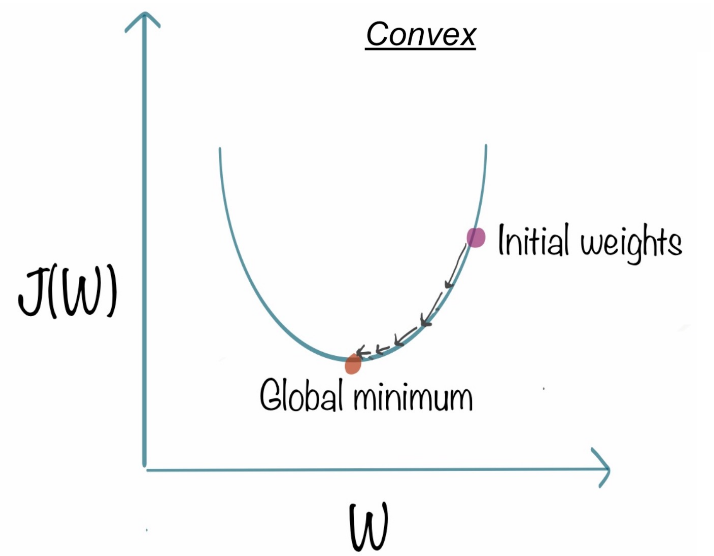


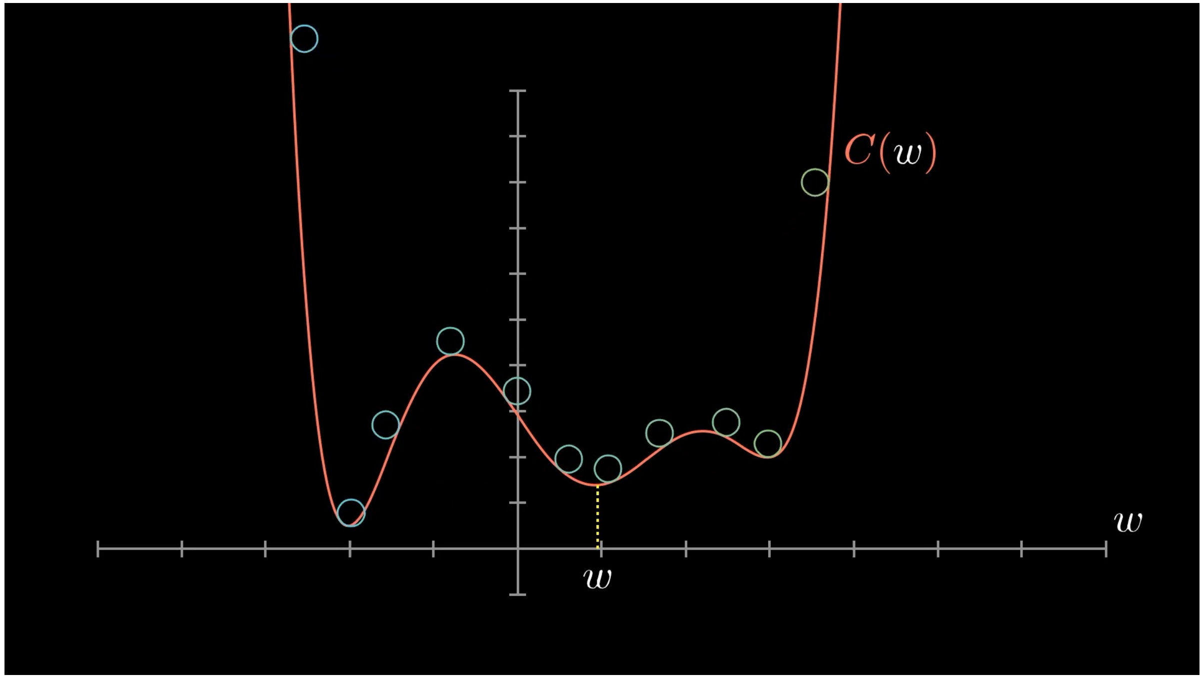
$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2$$

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\operatorname{argmin}}\ L(\mathbf{w}, b)$$

Gradient Descent

- **Iteratively reducing** the error by updating the parameters in the direction that incrementally lowers the loss function
  - On *convex* loss surfaces, it will eventually converge to a global minimum
  - For *nonconvex* surfaces, it will at least lead towards a (hopefully good) local minimum.



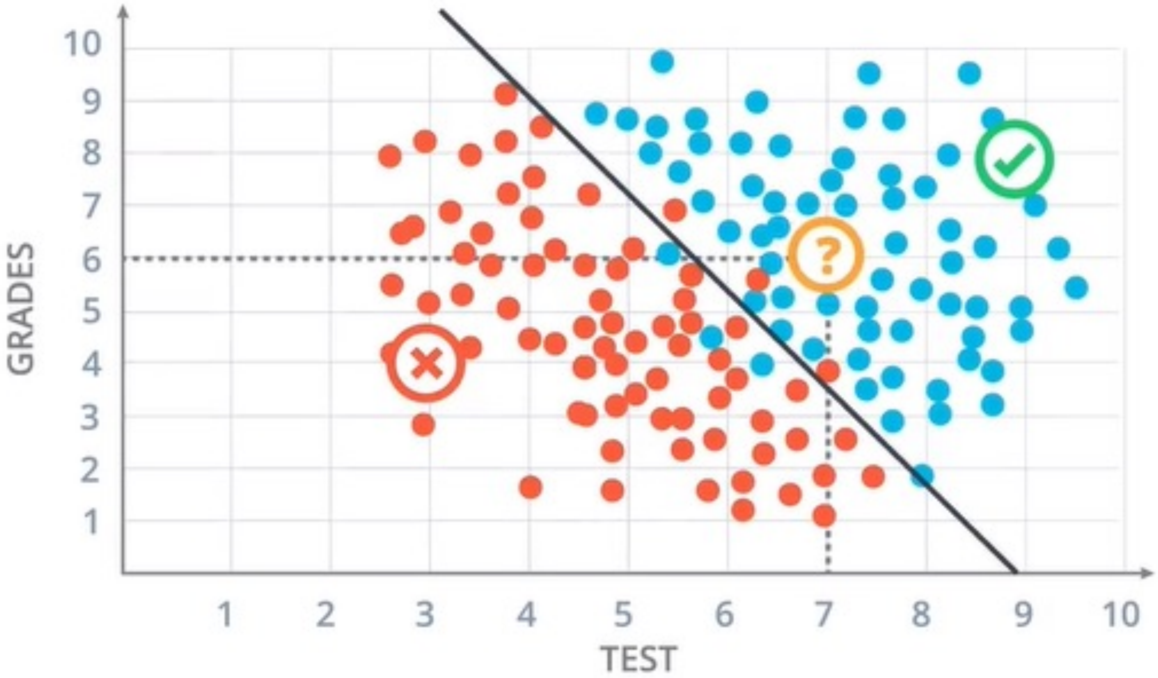


- The key technique for optimizing *nearly any* deep learning model

Linear Classification

- Hypothesis
  - Acceptance depending on Test and Grade
- Data
  - $\left(x^{(i)}, y^{(i)}\right)$
- Input
  - $x_1^{(i)}$ as test scores and $x_2^{(i)}$ as test scores
- Output
  - $\hat{y}^{(i)}$ as a threshold decision of Accept or Reject
- Model
  - A linear boundary line to separate the data
    - $w_1 x_1 + w_2 x_2 + b = 0$
  - A threshold to activate a decision against the line
    - > 0: Accept; < 0: Reject

An example of acceptance at a University

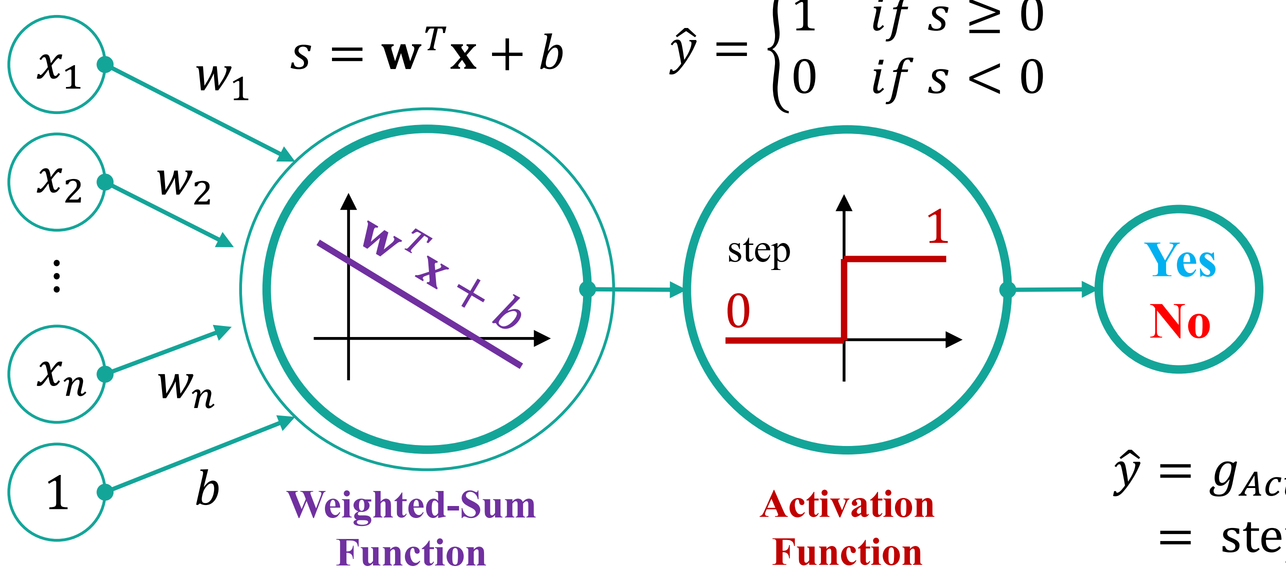


A Linear Boundary Line of $2x_1 + x_2 - 18 = 0$
as a decision criteria from regression to classification

Perceptron with an Activation Function

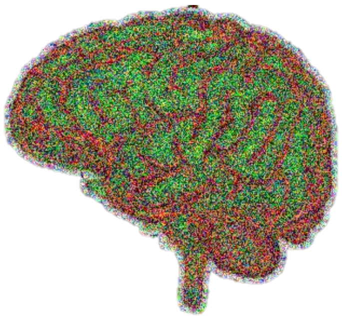
- An Artificial Neuron with two nodes
  - **Weighted-sum node**
    - Calculate a linear equation $s(x)$ with inputs on the weights plus bias
  - **Activation node**
    - Apply the step function to get the predicted result $\hat{y}(s)$

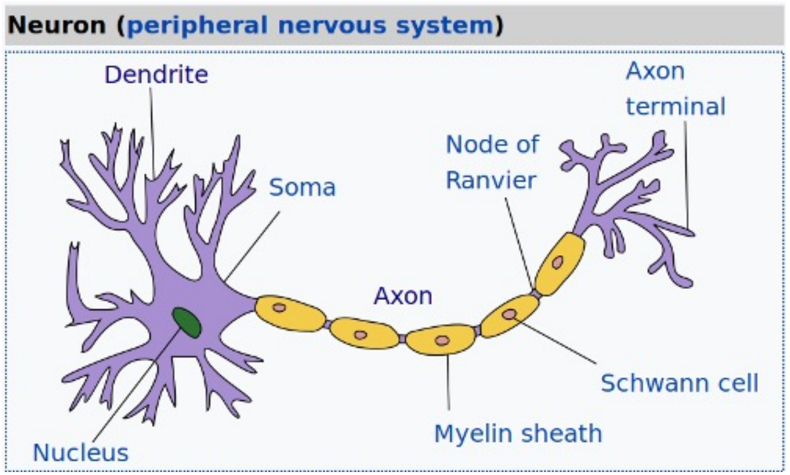
$$s = \mathbf{w}^T\mathbf{x} + b \qquad \hat{y} = \begin{cases} 1 & if\ s \geq 0 \\ 0 & if\ s < 0 \end{cases}$$

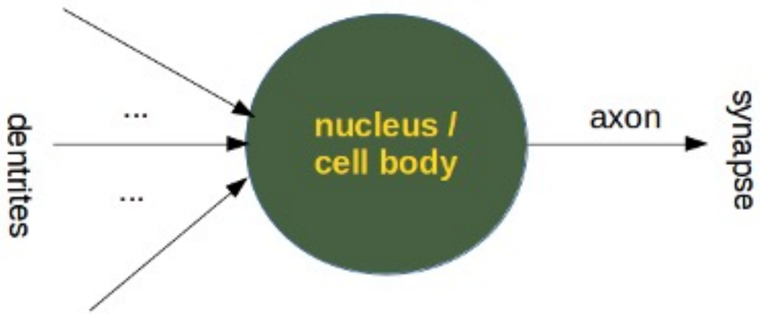


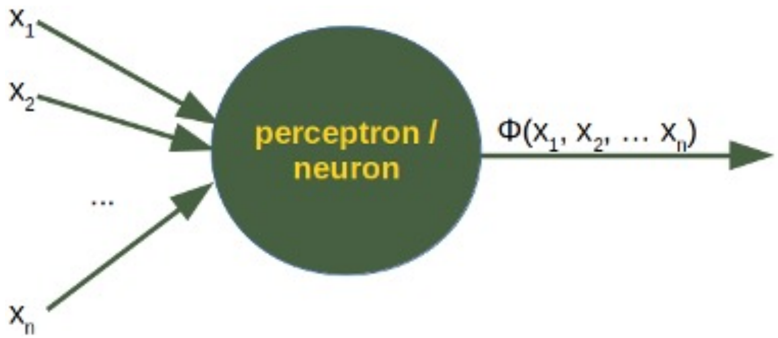
$$\hat{y} = g_{Activation}\left[f_{WeightedSum}(\mathbf{x})\right] = \text{step}(s, 0) = \text{step}(\mathbf{w}^T\mathbf{x} + b, 0)$$

A Perceptron as an Artificial Neuron





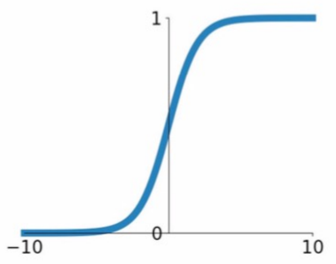




Activation Function

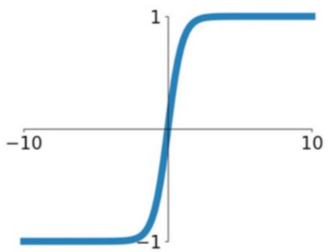
Sigmoid

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



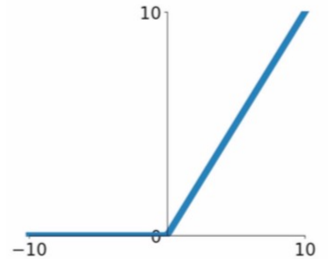
tanh

$\tanh(x)$



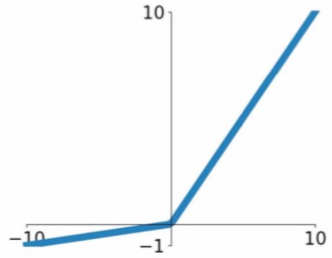
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

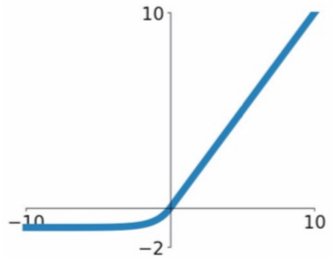


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

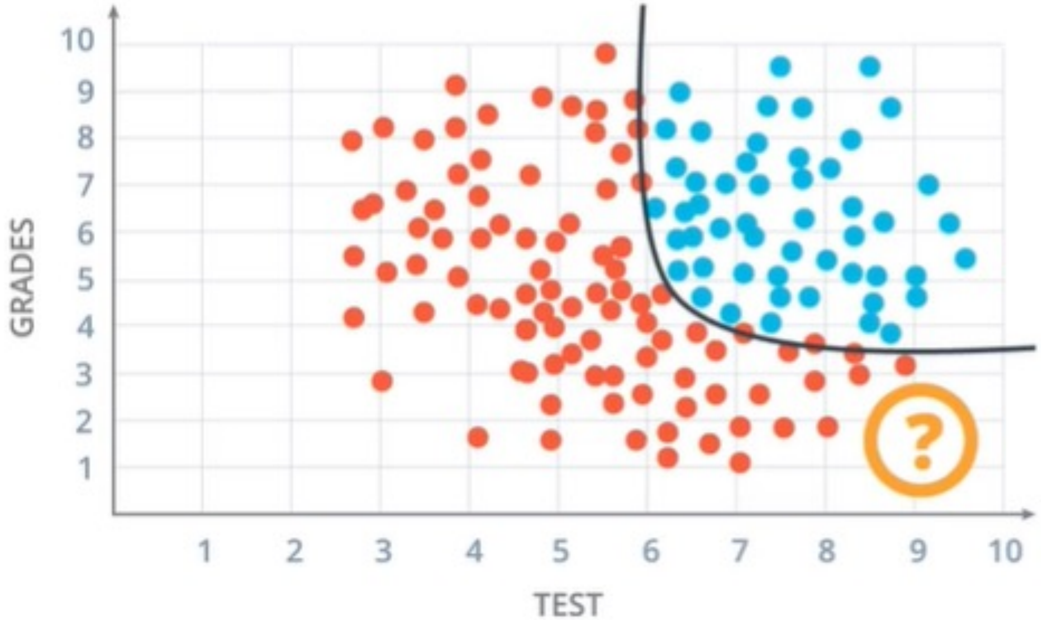
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



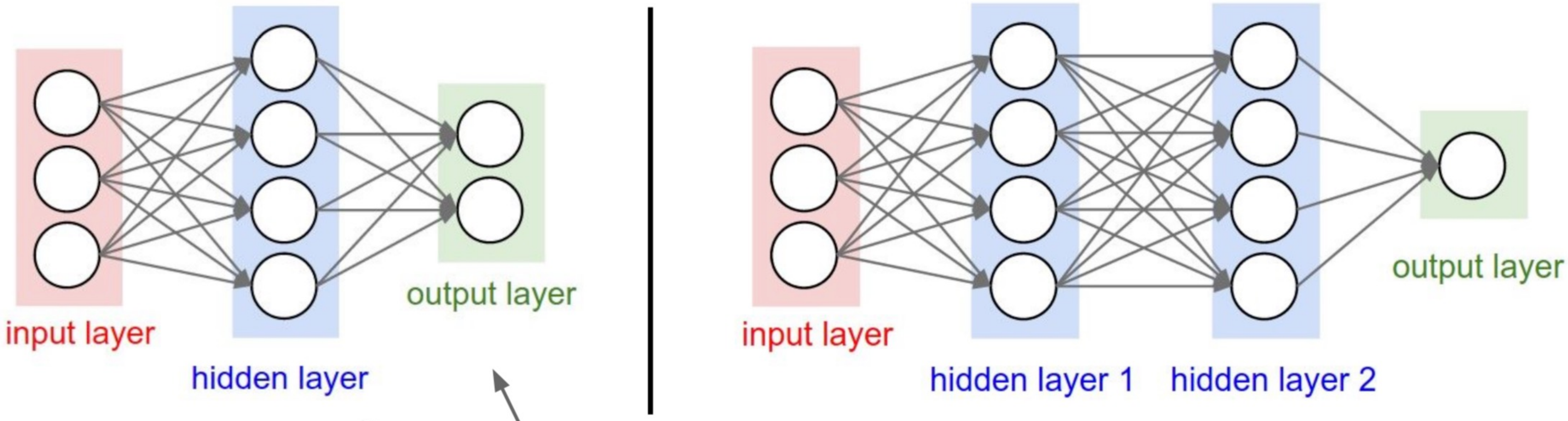
Limitation of Single Perceptron

What if the boundary line is non-linear?



- Unable to classify nonlinear scenarios

Multi-Layer Perceptrons



"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Exercise I

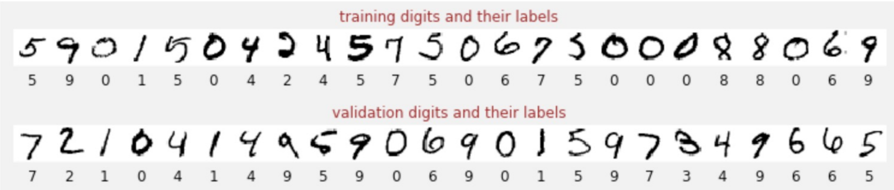
- Build your first neural networks on the website
  - https://playground.tensorflow.org/

- Play with different data types, features, network structures. Can Neural networks separate nonlinear features?
- How does nonlinearity come?
- How important is input features
- How important is number of neurons and layers of neurons?

Exercise II

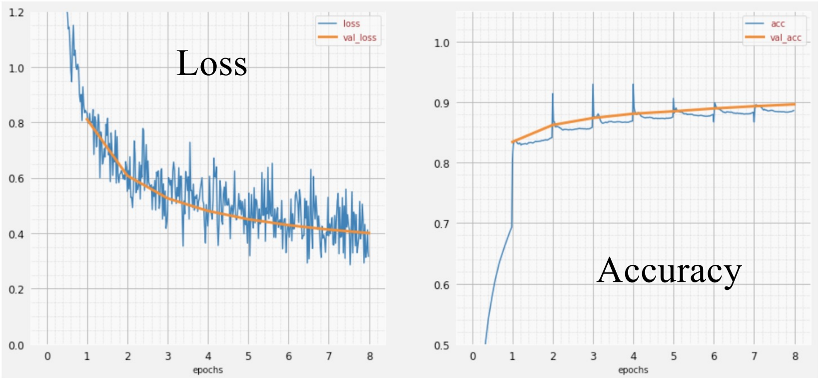
MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

A Toy Example of Training a Neural Network



Tensors as matrices storage of data

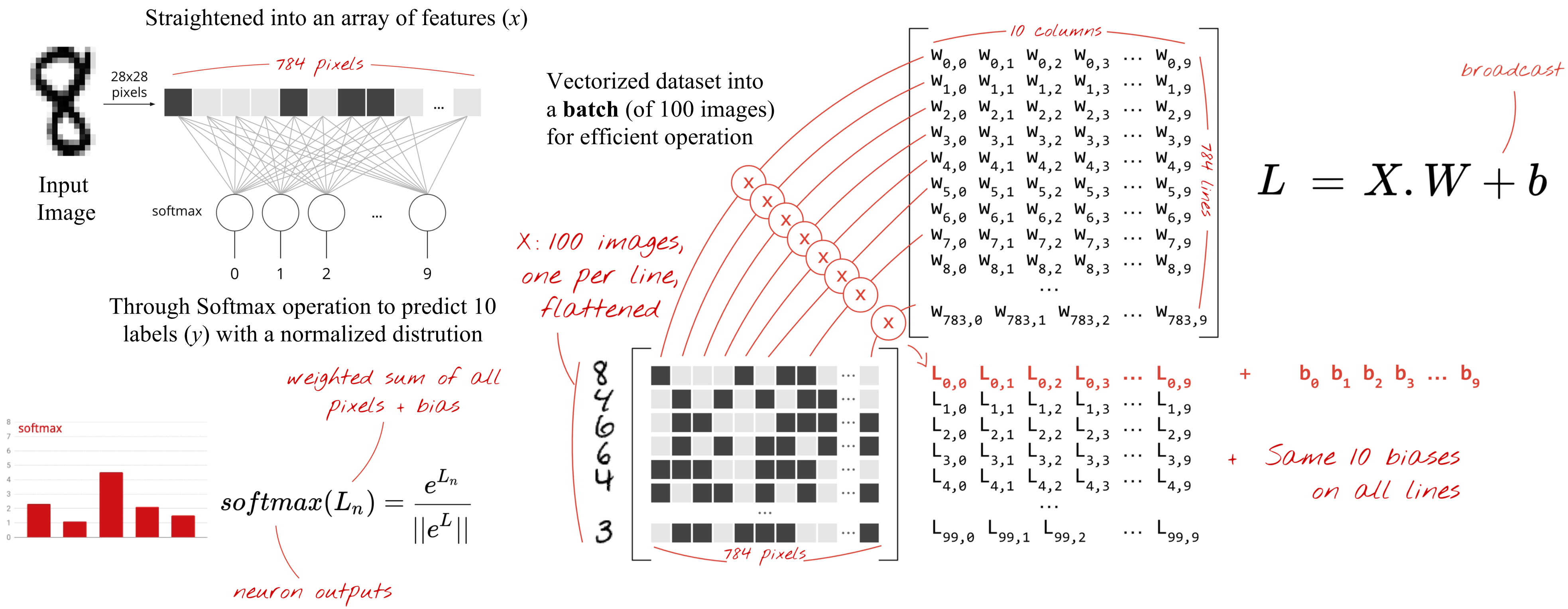
- [28, 28, 1]: A 28x28 pixel grayscale image (Gray)
- [128, 28, 28, 3]: A batch of 128 color images (RGB)



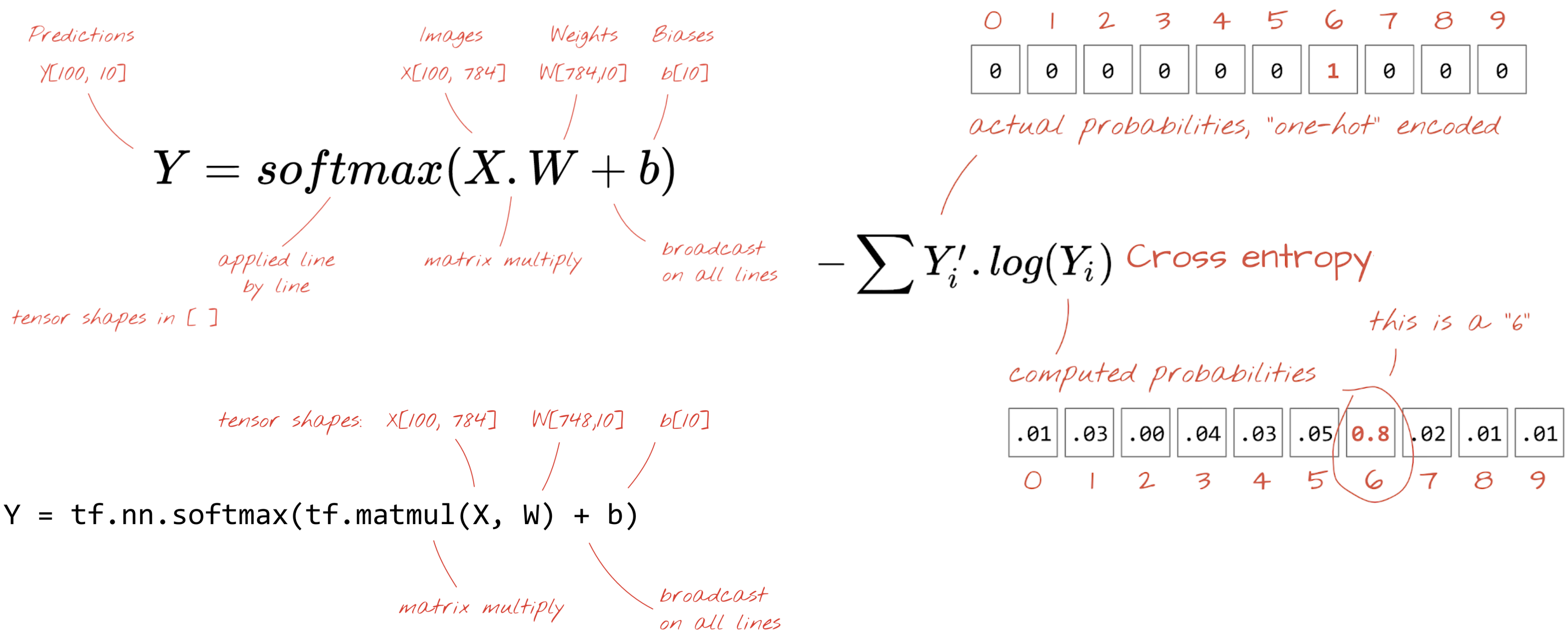

```
468/468 [==============================] - 2s 5ms/step - loss: 0.4290 - acc: 0.8862 - val_loss: 0.4003 - val_acc: 0.8963
Epoch 9/10
378/468 [=======================>......] - ETA: 0s - loss: 0.4230 - acc: 0.8866
```

60,000 training images
10,000 testing images

A Single-layer Network of Image Classification

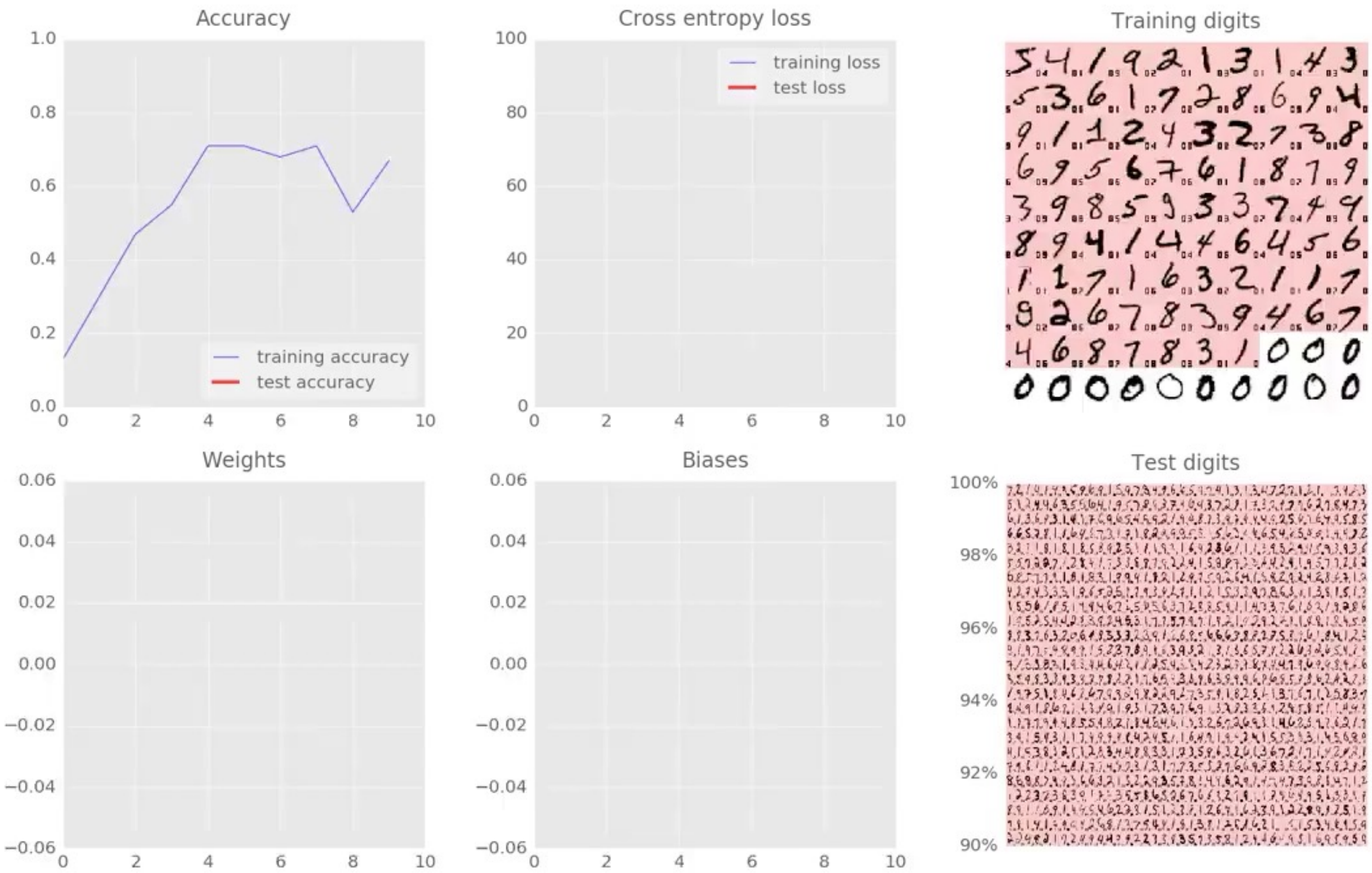


Softmax on a Batch of Images

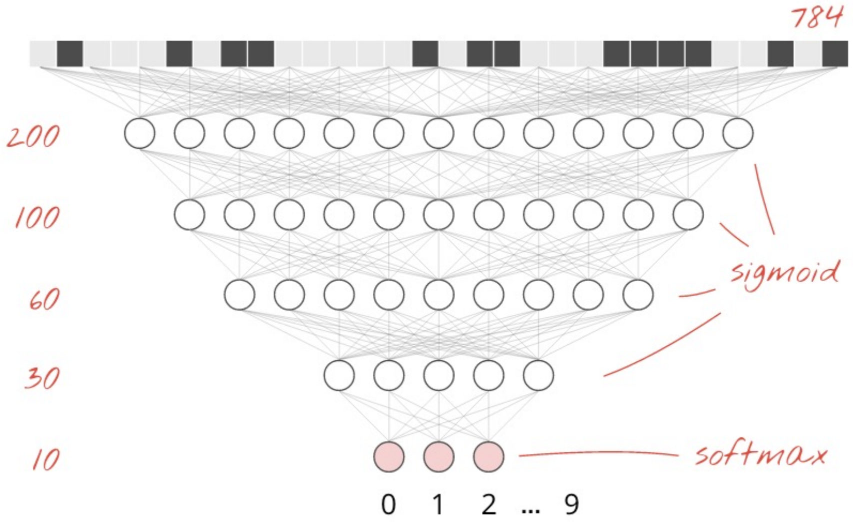


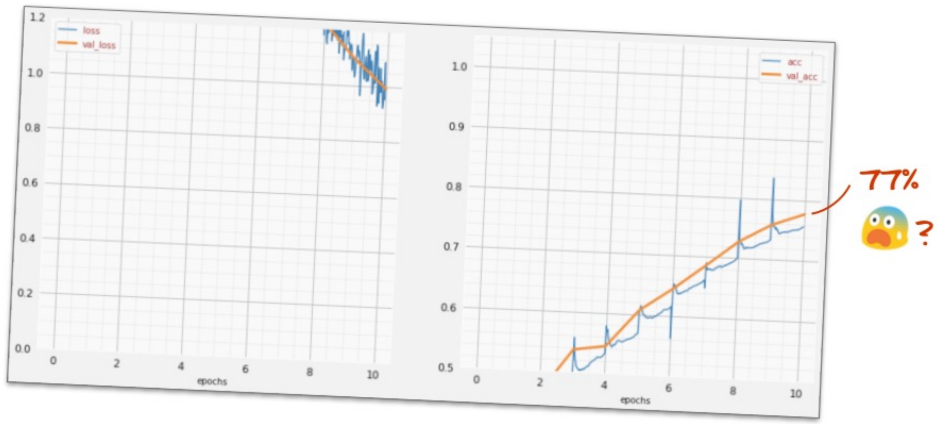
https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/blob/master/tensorflow-mnist-tutorial/keras_01_mnist.ipynb

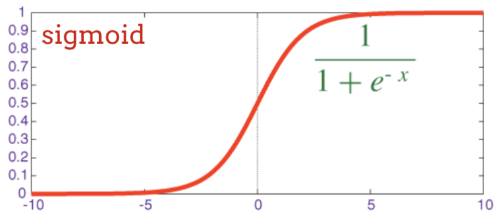
Training Process



Adding Layers





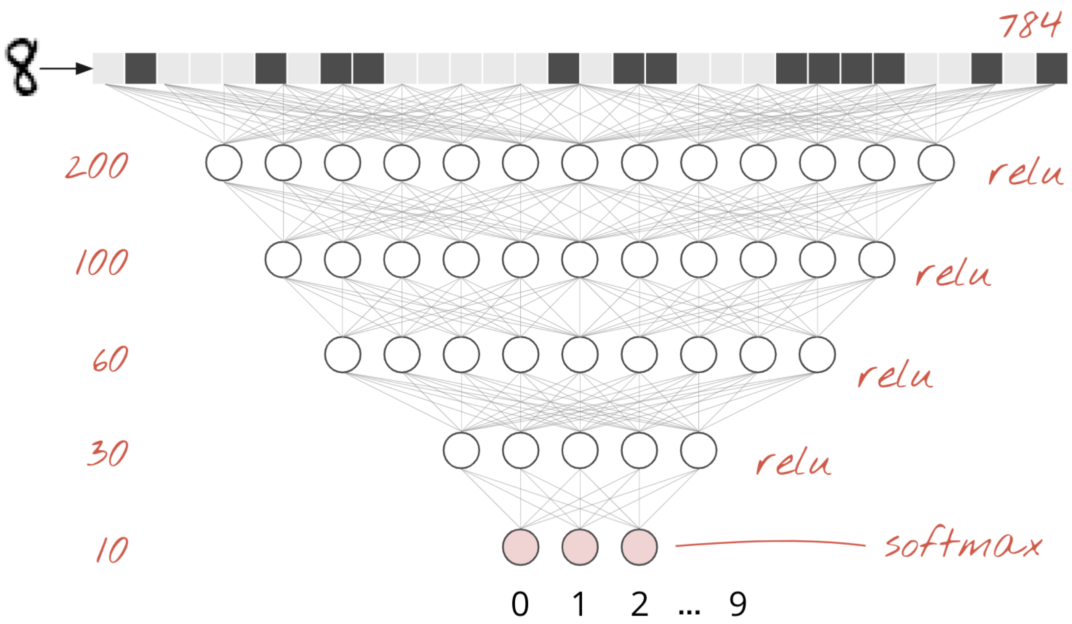


Getting flat

- The gradient can become very small and training get slower and slower.

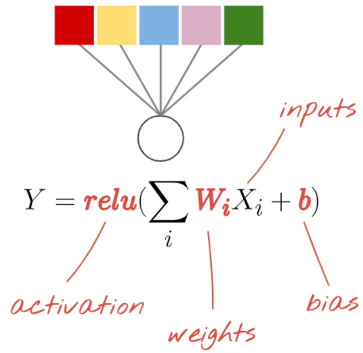
Simply adding more layers with sigmoid activations does not give us the expected results …

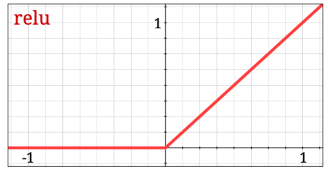
Special Care for Deep Networks

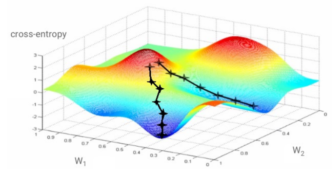


HANDS ON:

Replace all `activation='sigmoid'` with `activation='relu'` in your layers and train again.

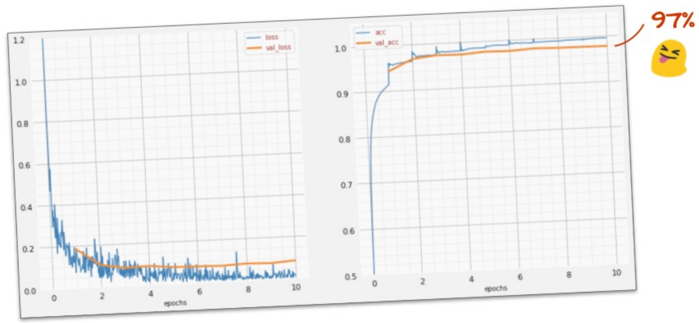






HANDS ON:

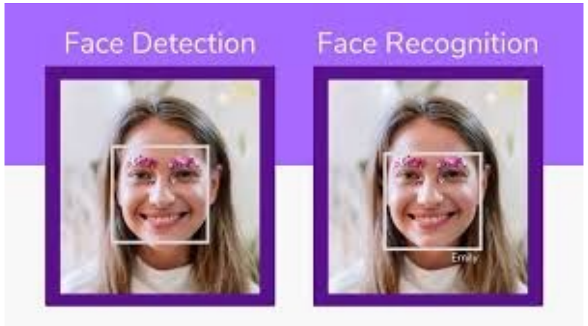
Replace the `'sgd'` optimizer with a better one, for example `'adam'` and train again.

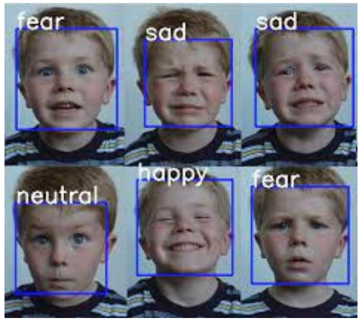


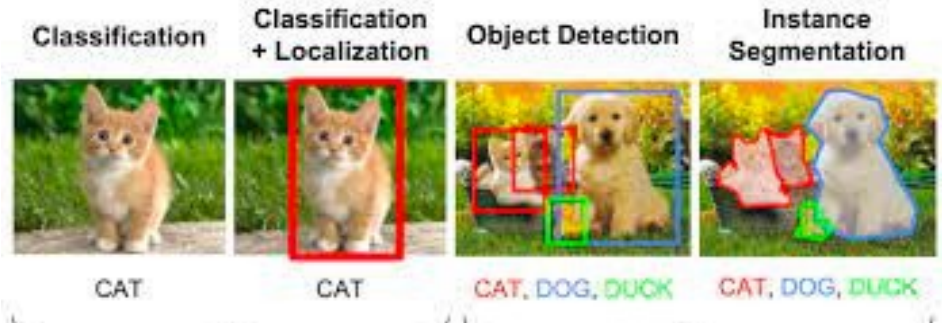
https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd/blob/master/tensorflow-mnist-tutorial/keras_02_mnist_dense.ipynb

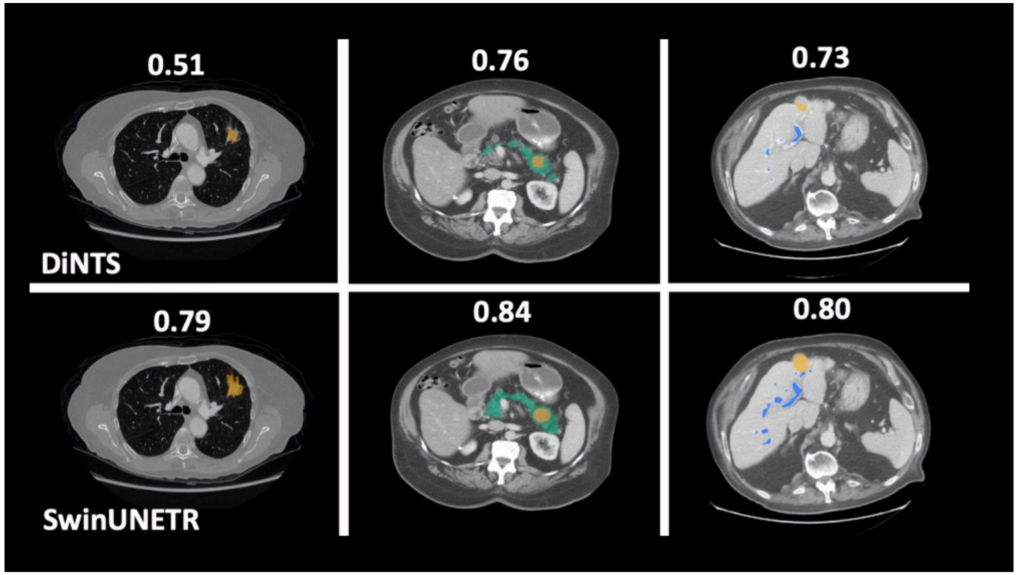
Convolutional Networks

Convolutional Networks Applications

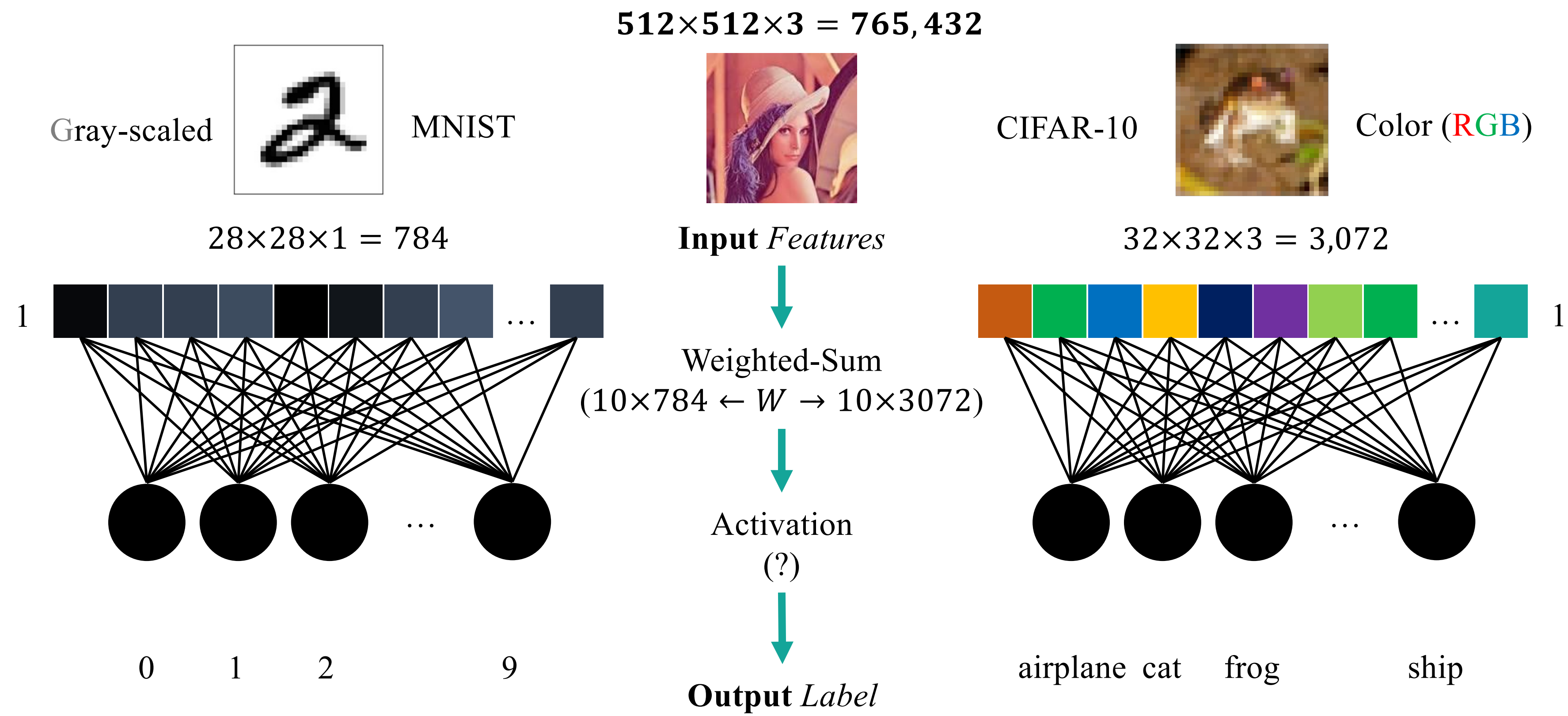






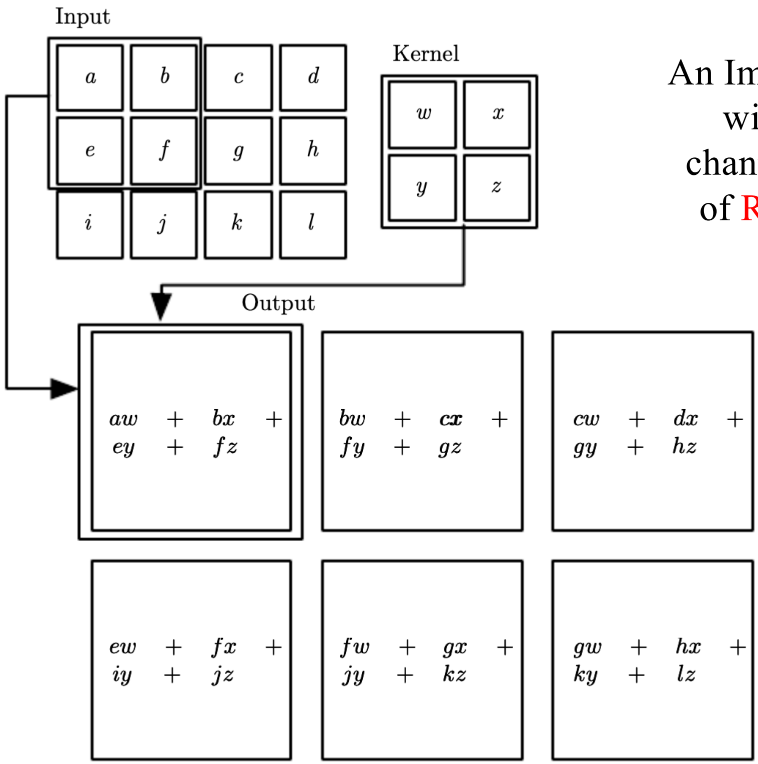


A Design Challenge with Increasing Dimensions

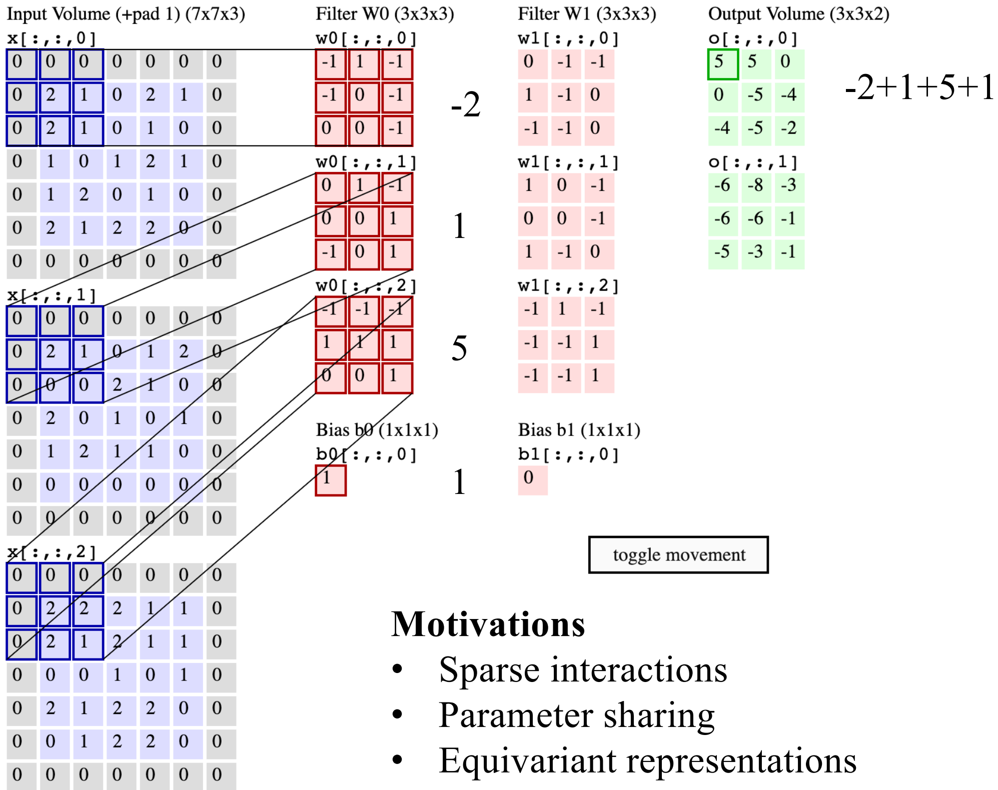


Regular Neural Nets don’t scale well to full images

Convolutional Operation



An Image with 3 channels of RGB



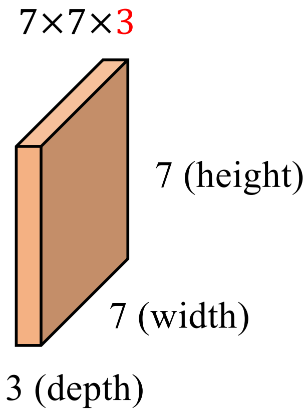
Motivations

- Sparse interactions
- Parameter sharing
- Equivariant representations

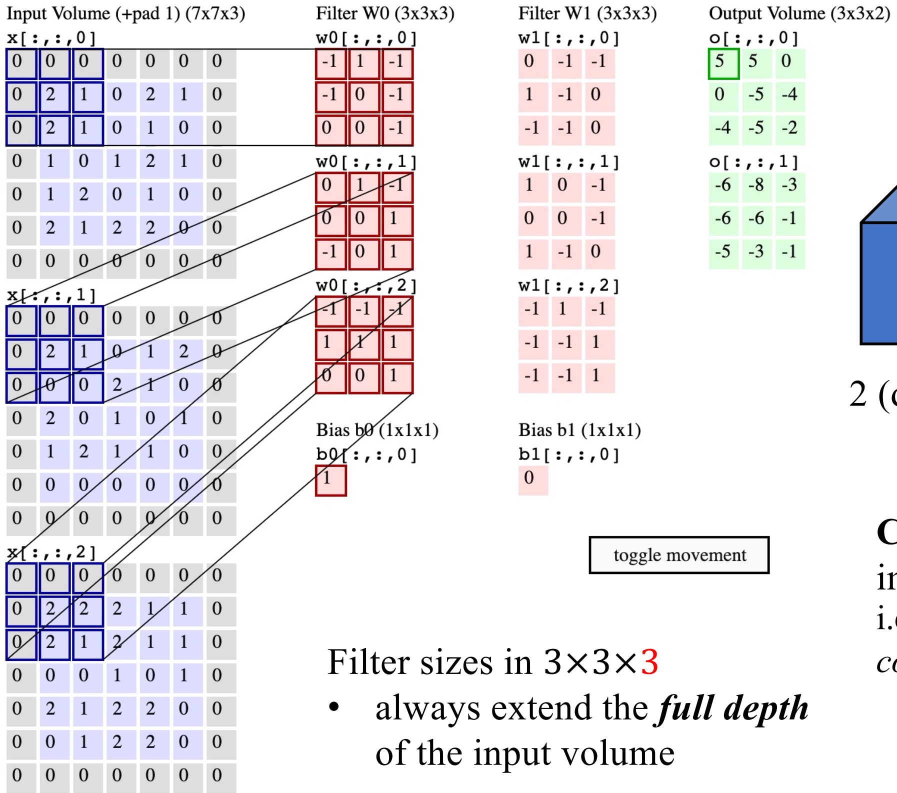
Convolution in 3D Volumes

Preserved spatial structure between the input and output volumes in width, height, number of channels

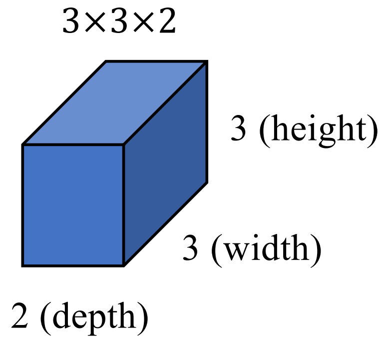
Input Volume in 3D



Layers in a ConvNet: Transform an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.



Filter sizes in 3×3×3
- always extend the ***full depth*** of the input volume

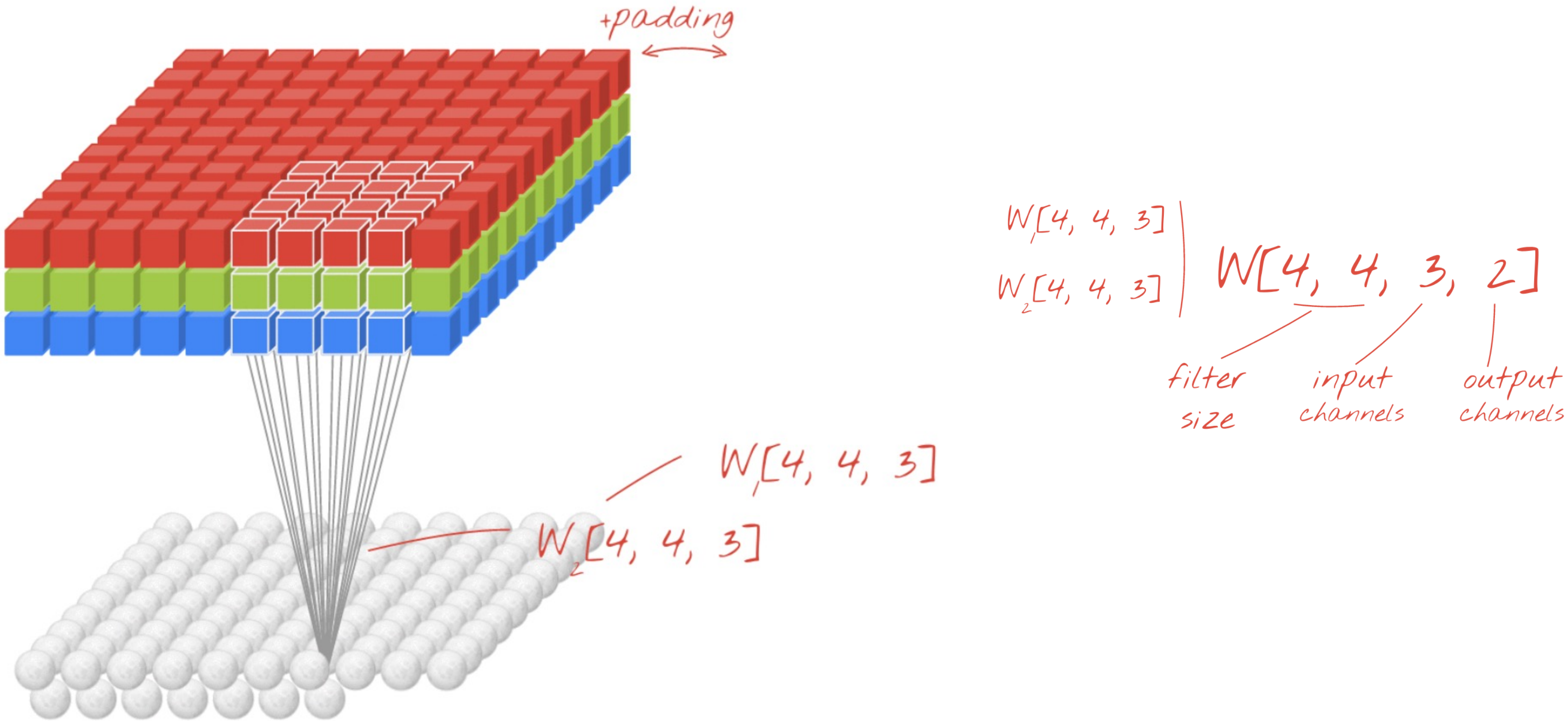


Output Volume in 3D

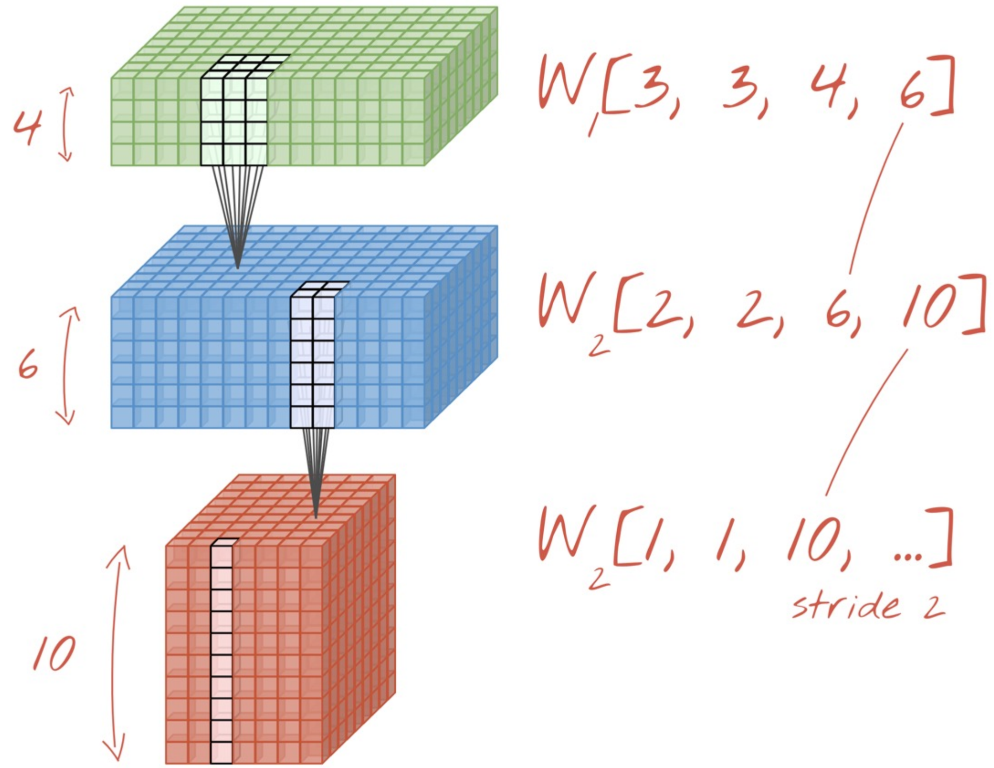
Convolve the filter with the image
i.e. "*slide over the image spatially, computing dot products*"

The Design of a Convolutional Layer

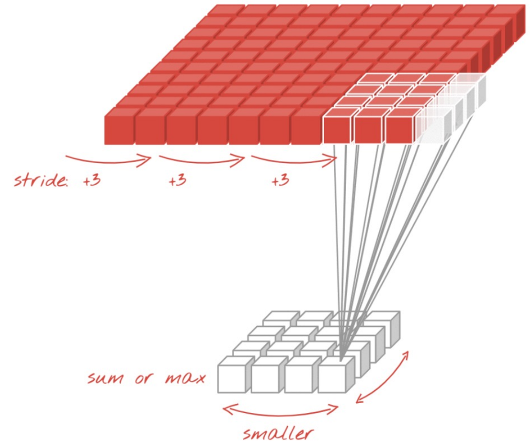
A convolutional layer is defined by the filter (or kernel) size, the number of filters applied and the stride



Output Volume Size



- Depth (number of channels):
  - *adjusted by using more or fewer filters*
- Width & Height:
  - *adjusted by using a stride >1*
  - *(or with a max-pooling operation)*



Defined by the filter (or kernel) size, the number of filters applied and the stride

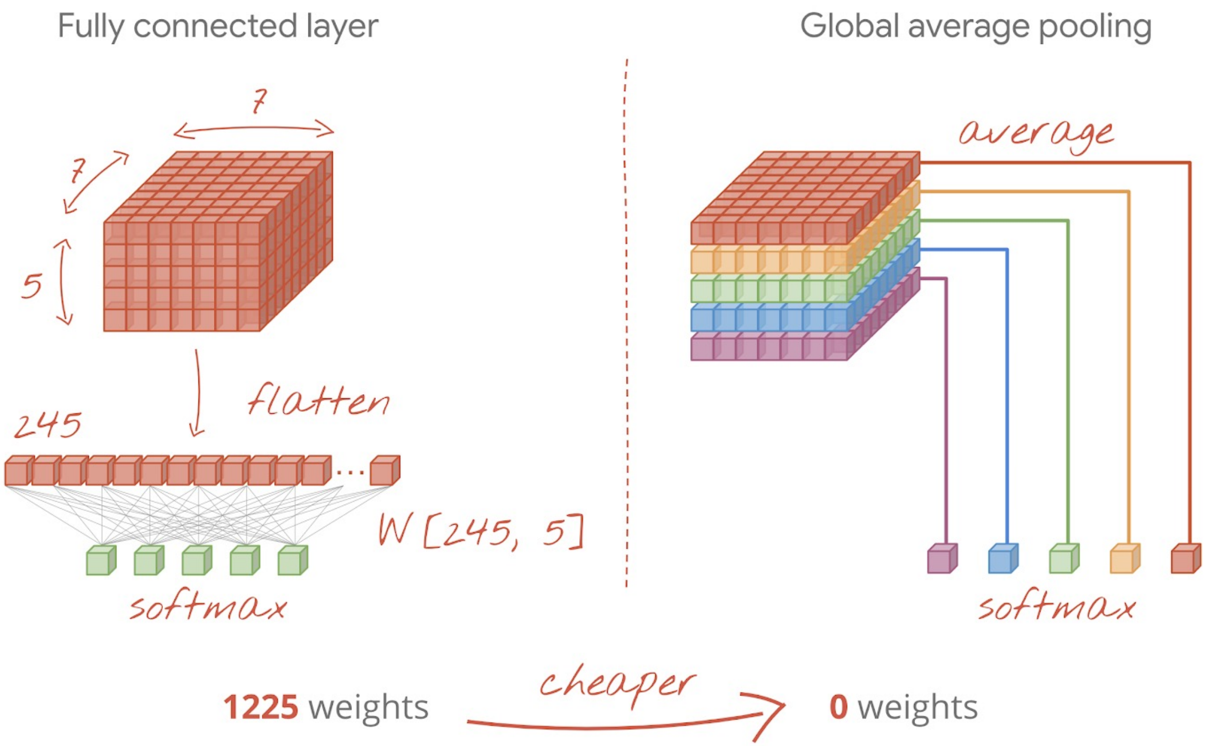
The Last Layer

From a Cubic Volume in 3D to predicted labels

Similar like a normal neural network

Expensive in #weights

But preseves the location data (*x*, *y*)

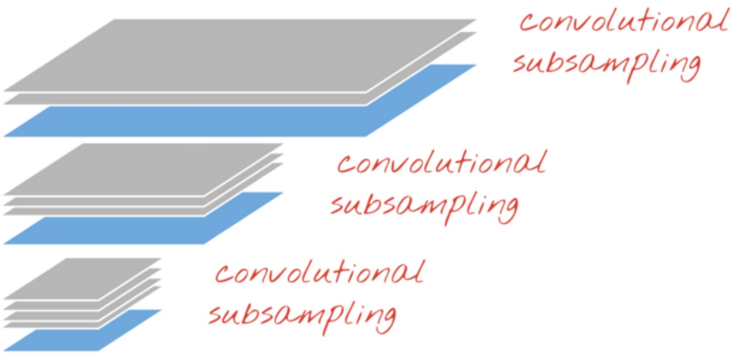


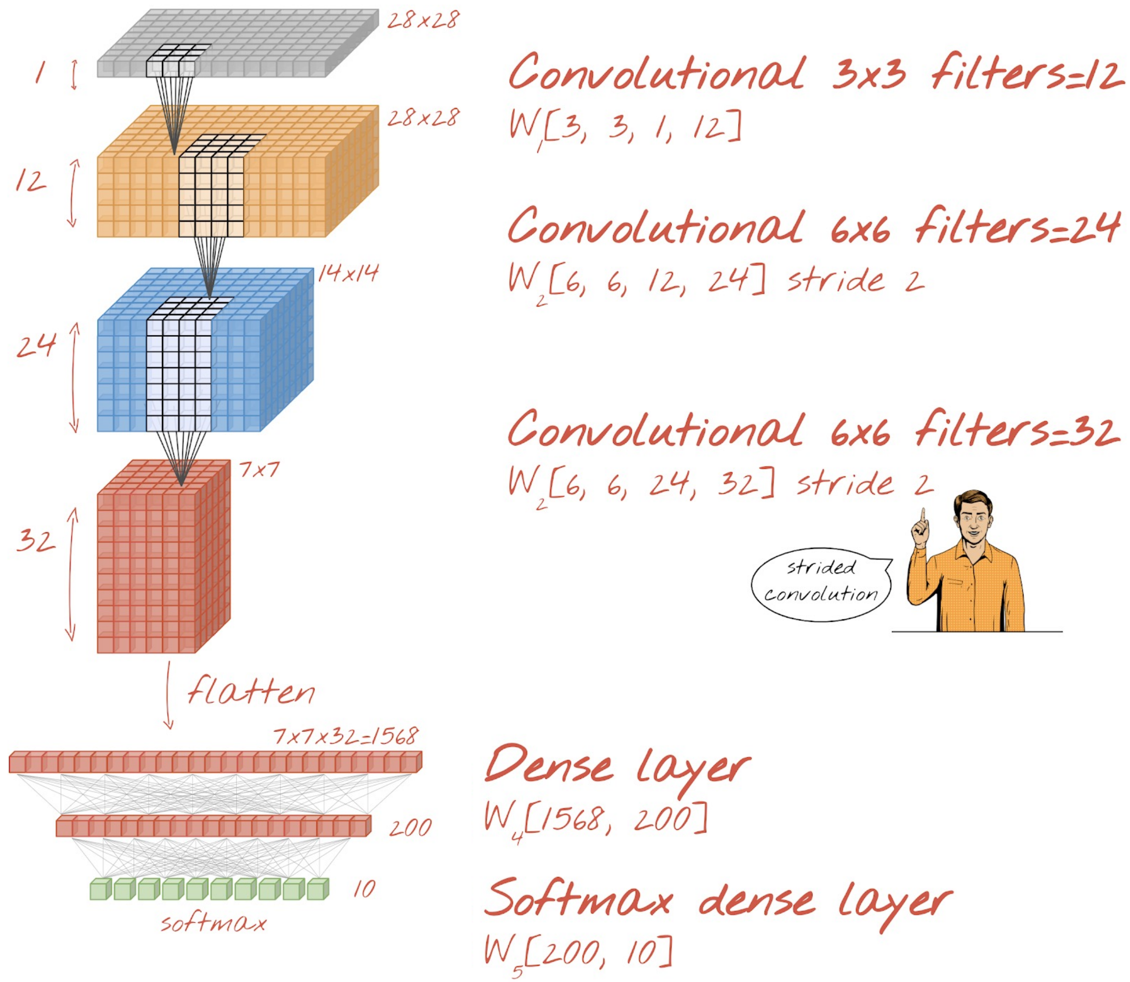
Much lighter in calculation

The average pooling explicitly discards all location data

Stacking Up a ConvNet

Layer-by-layer





Exercise: Handwritten digits classification

- MNIST forward Neural Network:
  - https://ml4a.github.io/demos/forward_pass_mnist/
- MNIST confusion matrix
  - https://ml4a.github.io/demos/confusion_mnist/
- MNIST Convolutional Neural Network: filters
  - https://ml4a.github.io/demos/convolution_all/
- crfm.stanford.edu/2023/03/13/alpaca.html
- MNIST TensorFlow.js playground: https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html
- Explainable AI Demos: https://lrpserver.hhi.fraunhofer.de/image-classification

Thank you~

Wan Fang
Southern University of Science and Technology