

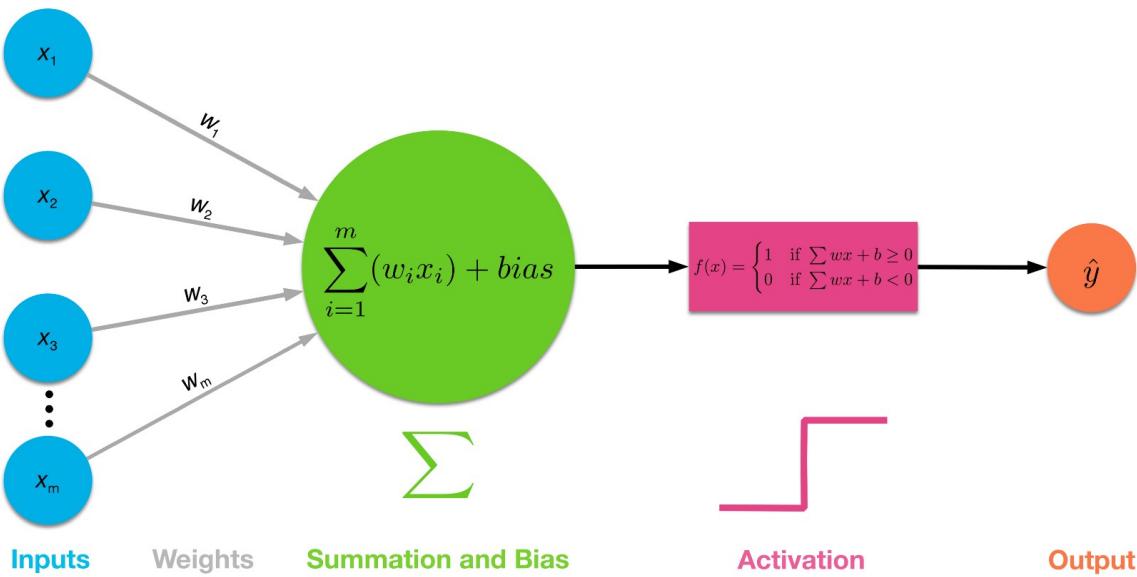
Neural Networks (DNNs and CNNs)

Sharib Ali, Ph.D.

Institute of Biomedical Engineering (IBME)
Big Data Institute (BDI)

Perceptron

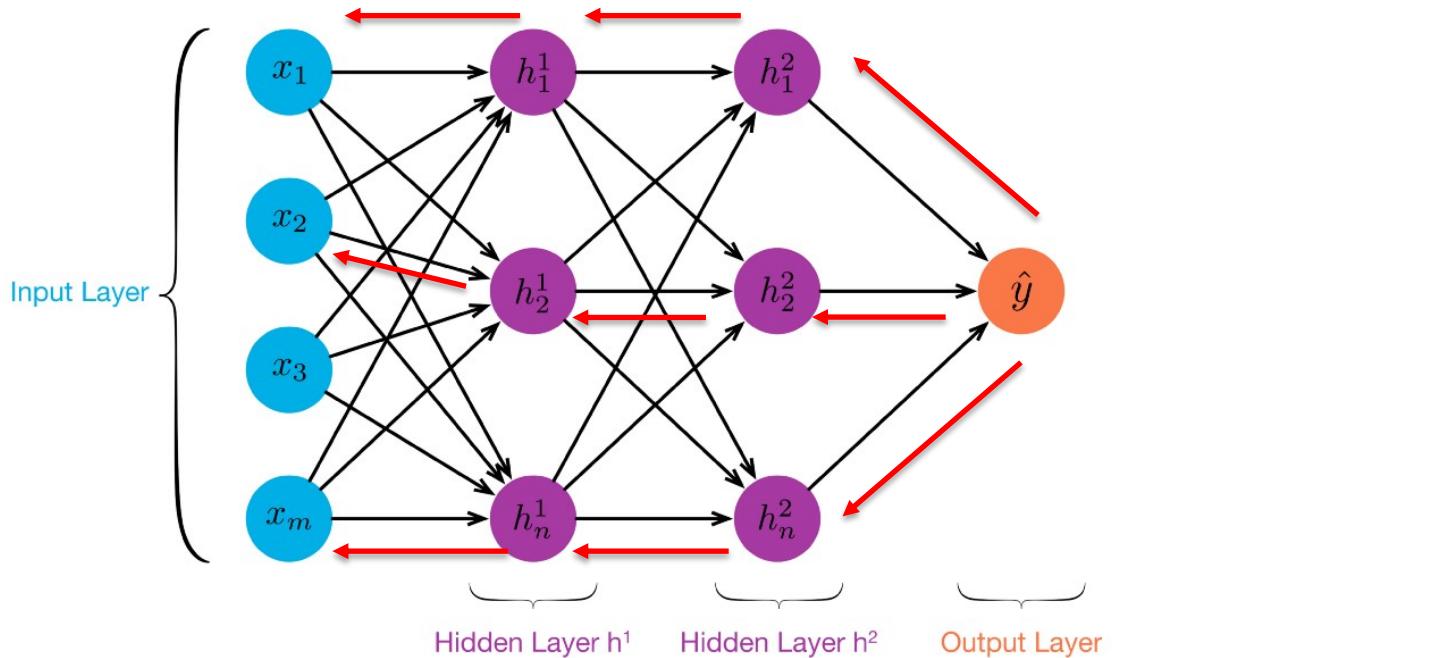
- Simple model of a neuron (1958, 1969, 80's...)
- Linear classifier
- Finds a hyperplane that separates 2 classes of points (if exists)



Single-layer Perceptron Network

Multi Layer Perceptron

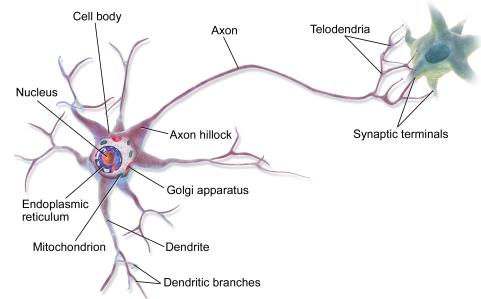
- Two main discoveries in 1960s
 - Backpropagation, a procedure to *repeatedly adjust the weights*
 - Hidden Layers, which are *neuron nodes stacked in between inputs and outputs*, allowing neural networks to learn more complicated features



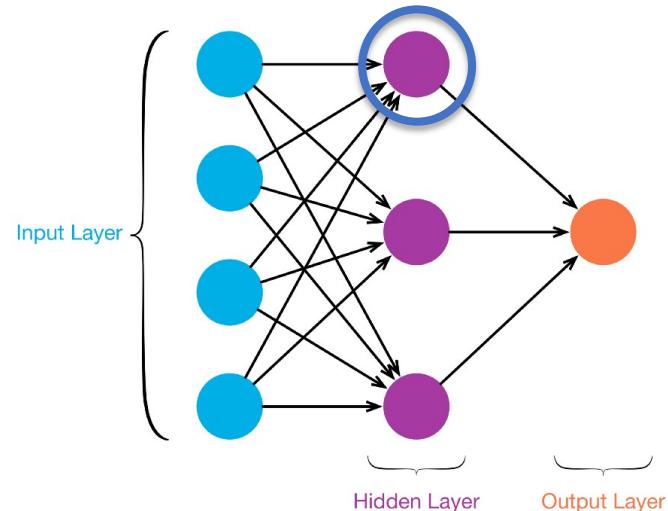
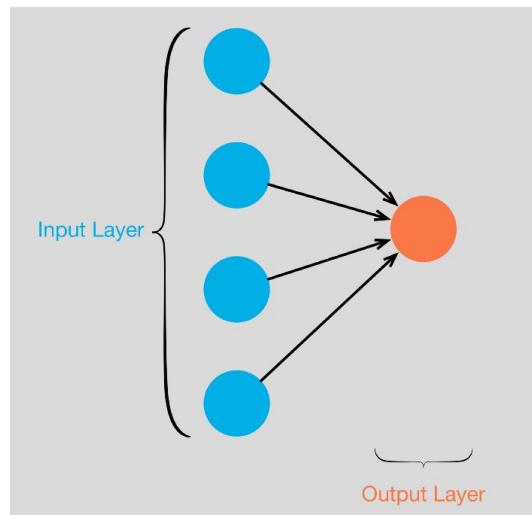
Effects of neurons



UNIVERSITY OF
OXFORD

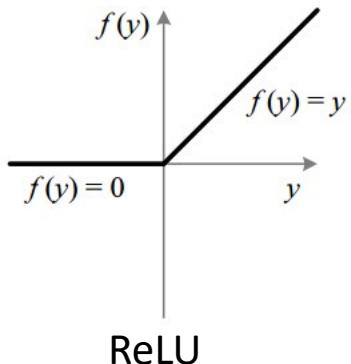
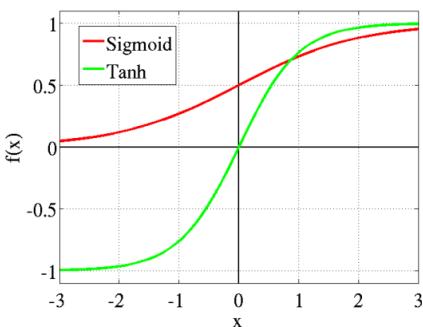


Source: wikipedia

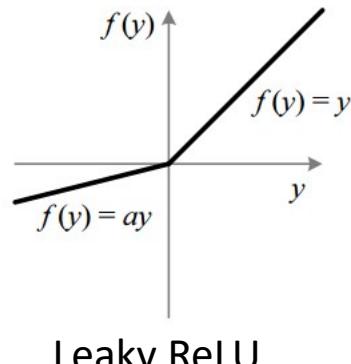


Multi-layer Perceptron Network

Live demo
Demo-NonLinearClassification



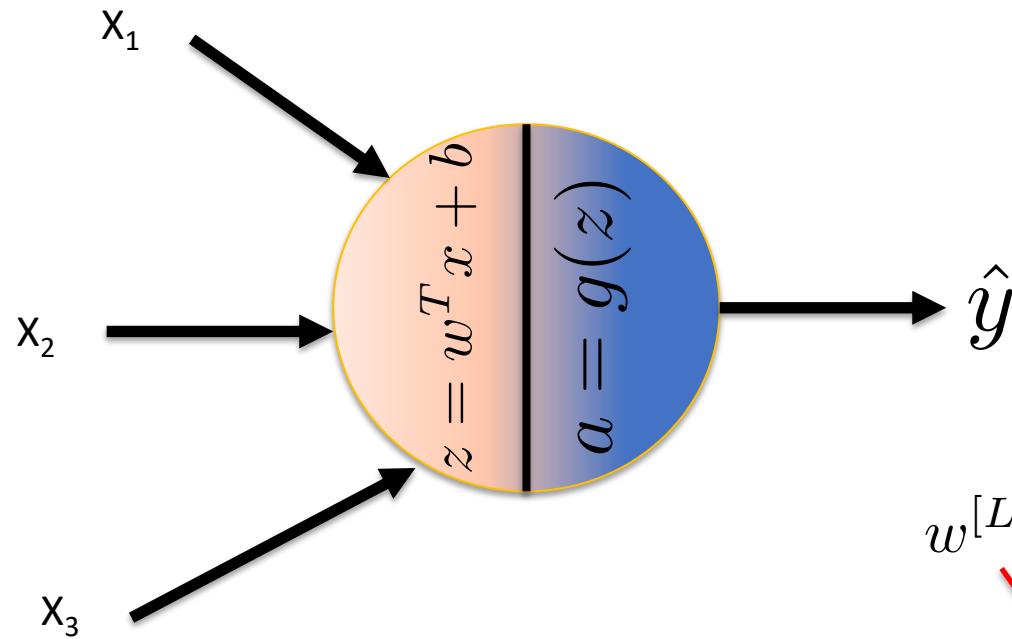
ReLU



Leaky ReLU



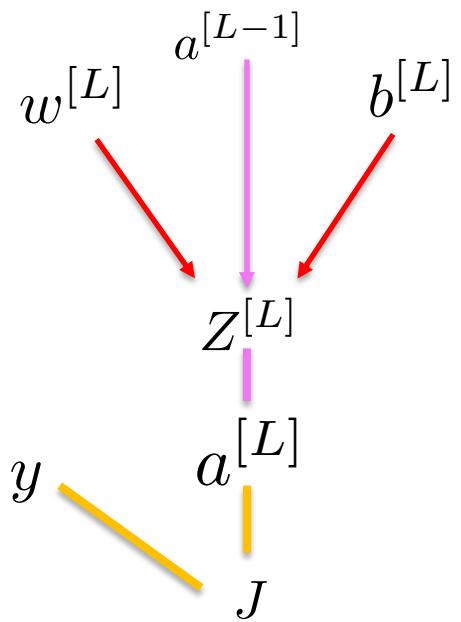
A single neuron



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

$$g(z) = \max(z, 0) \quad (\text{ReLU})$$

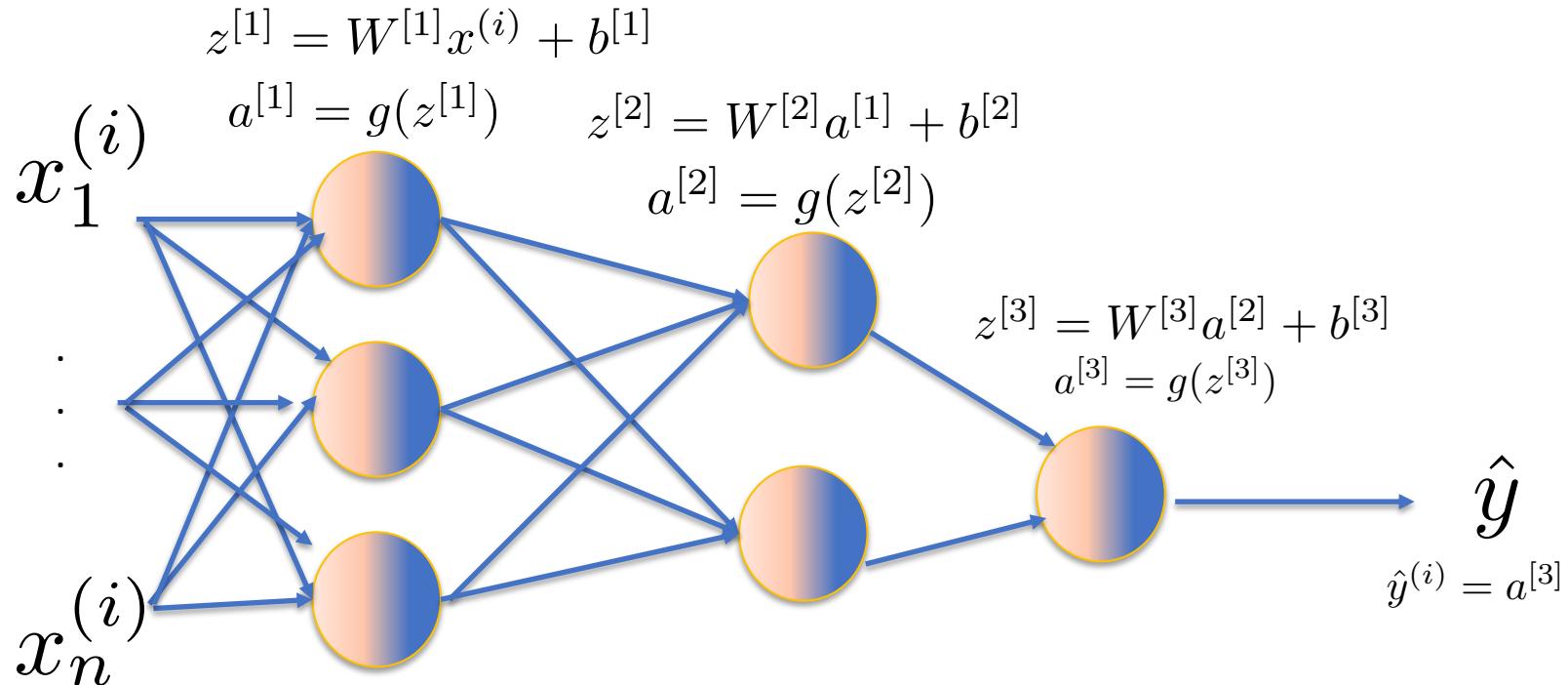
$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (\tanh)$$





2-layer neural network

Forward propagation



parameters: $3n + 3$ $2 * 3 + 2$ $2 + 1$

Sizes: $W^{[1]} \in \mathbb{R}^{3 \times n}$ $W^{[2]} \in \mathbb{R}^{2 \times 3}$ $W^{[3]} \in \mathbb{R}^{1 \times 2}$

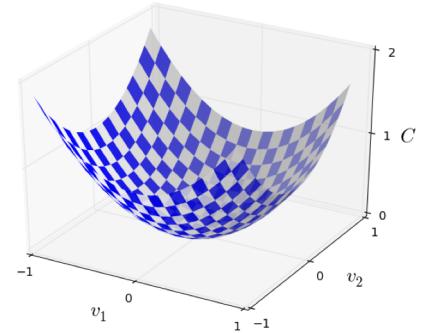
$b?$



2-layer neural network

Loss function

$$L(\hat{y}, y) = -[(1 - y) \log(1 - \hat{y}) + y \log \hat{y}]$$



Cost function

$$J / C_0 = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)}))$$

$$W^{[L]} := W^{[L]} - \alpha \frac{\partial J}{\partial W^{[L]}}$$

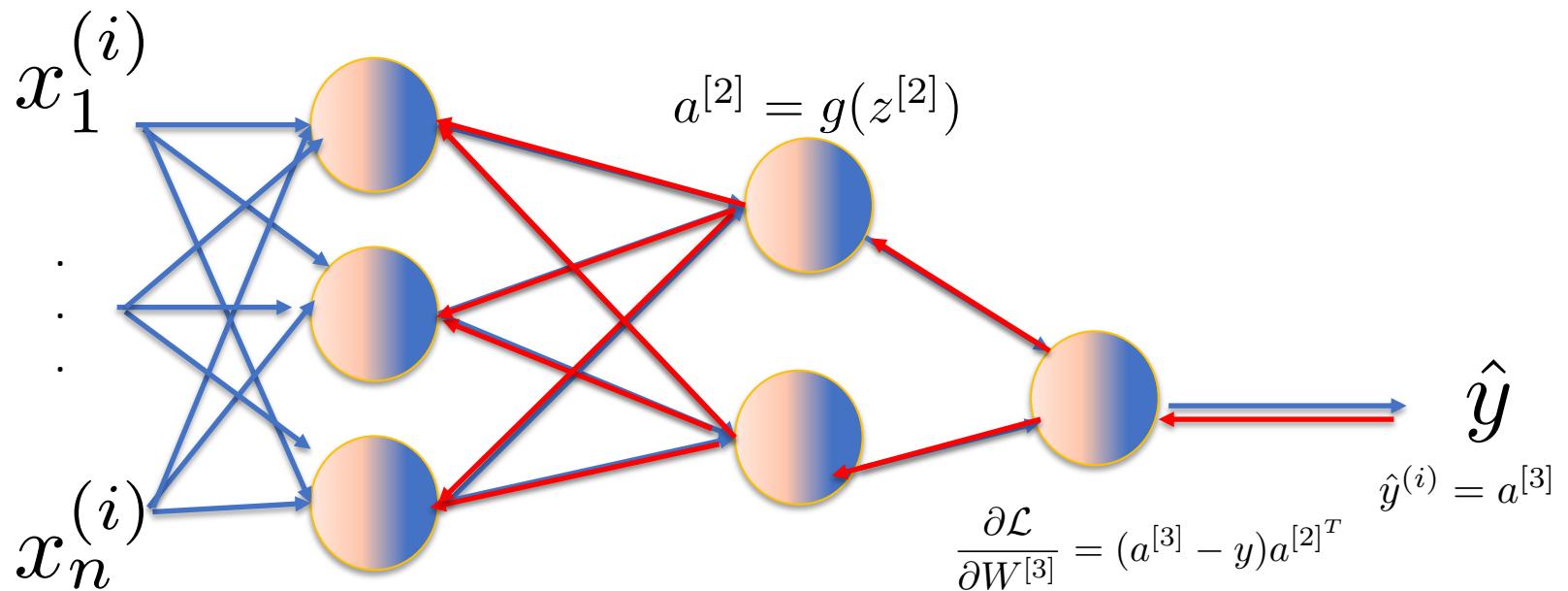
$$b^{[L]} := b^{[L]} - \alpha \frac{\partial J}{\partial b^{[L]}}$$

Backpropagation



2-layer neural network

Backward propagation



*Here we assume Stochastic gradient descent.

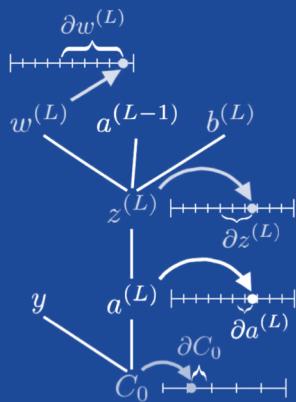


2-layer neural network

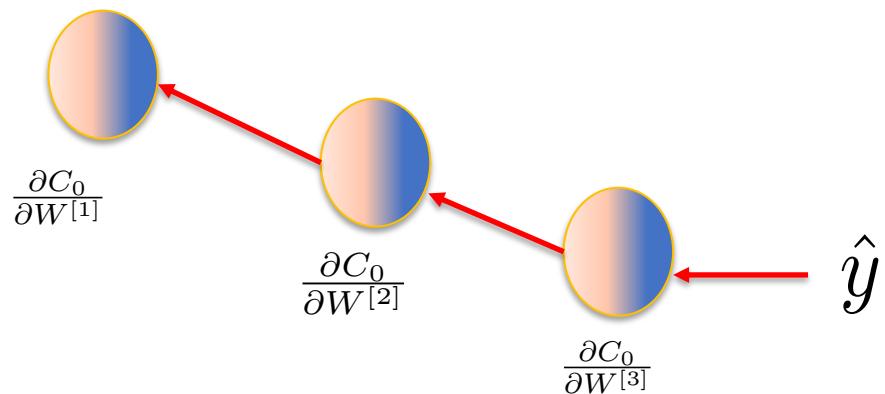
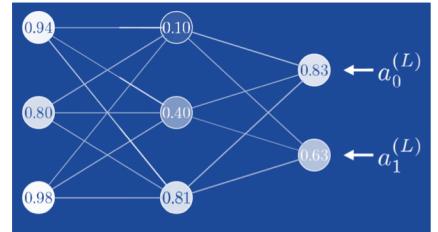
Backward propagation and updates

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

Chain rule



$$W^{[L]} := W^{[L]} - \alpha \frac{\partial J}{\partial W^{[L]}}$$
$$b^{[L]} := b^{[L]} - \alpha \frac{\partial J}{\partial b^{[L]}}$$



Assuming, gradient descent



2-layer neural network

Parameter initialization

- In practice, small random values are used to initialize ‘W’ and b is set to 0
- Best practice, use Xavier/He initialization

Optimization

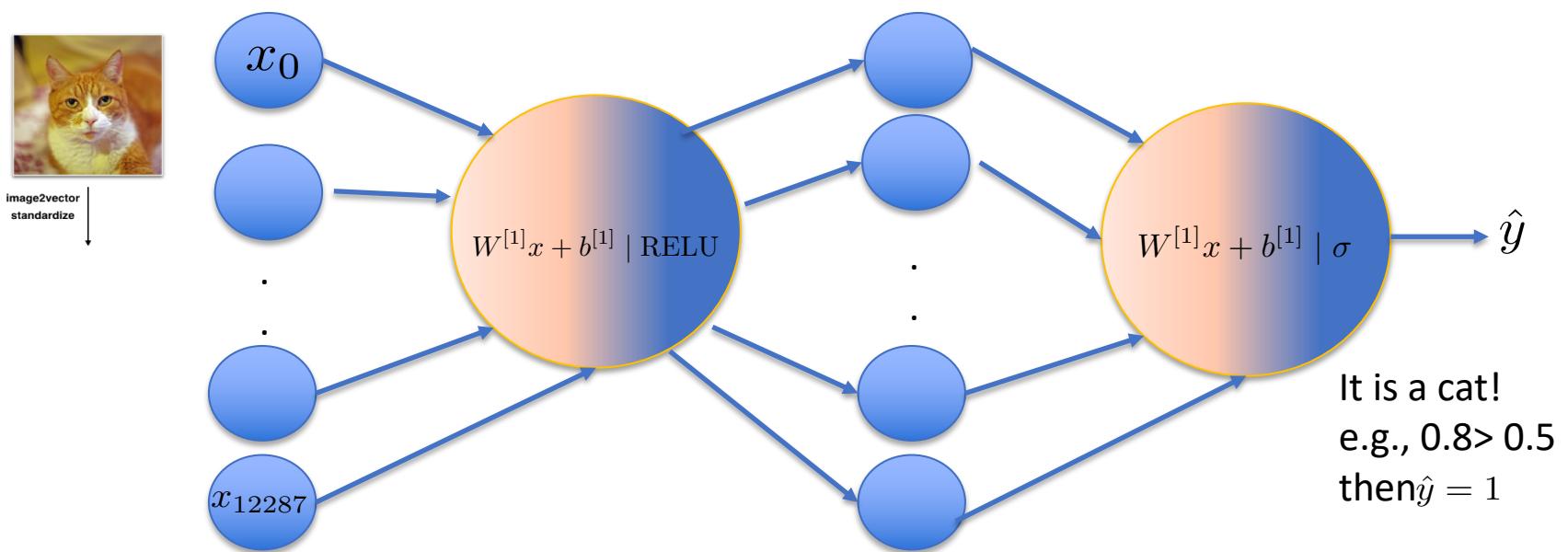
- Gradient descent, Stochastic gradient descent (SGD) or Adam
- Other optimization techniques: momentum, RMSprop, minibatch gradient descent



2-layer neural network

Exercises

- First create a single neuron and perform logistic regression classification
- Design a 2-layer neural network to classify cat vs non-cat image



Convolutional neural network

Drawbacks of DNN

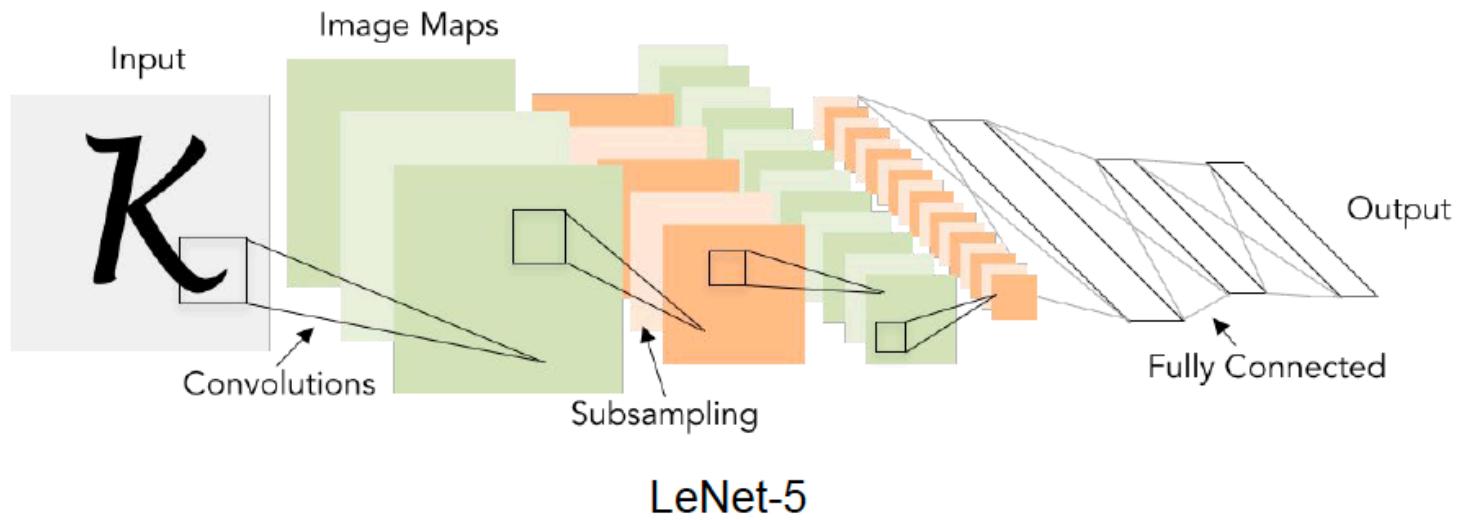
- Number of parameters are large (overfitting, computationally slower)
- Does not incorporate local information (like edges, corners in images)

Convolutions in NNs

- No. of parameters are reduced
- Incorporates local information
- Translation invariant

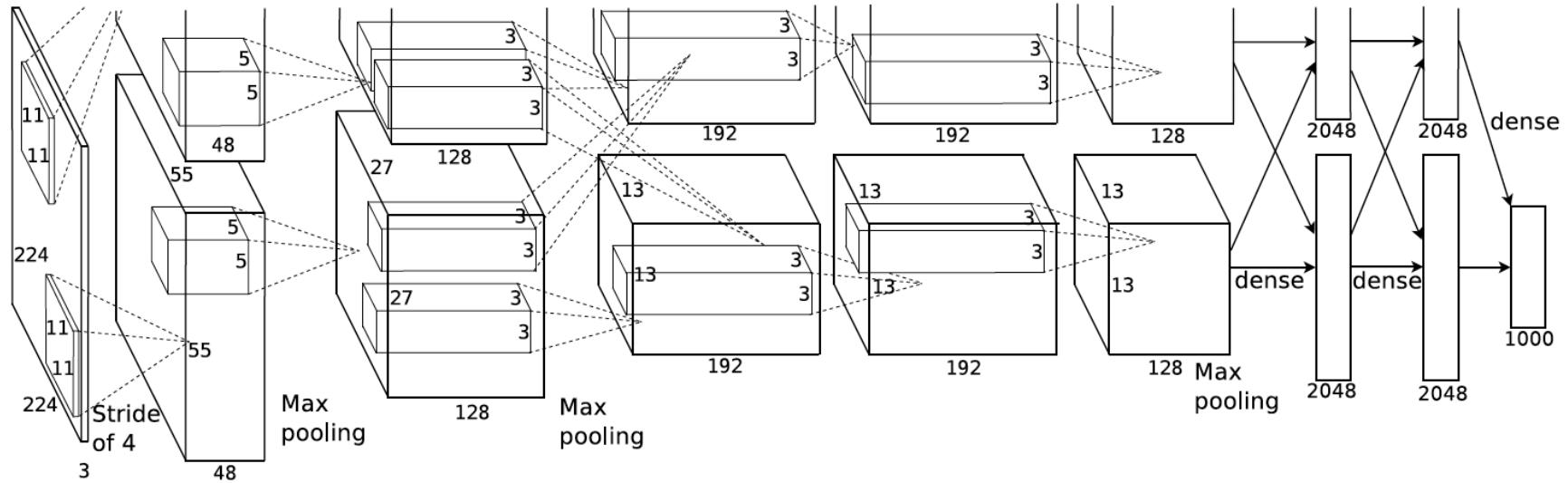
Concept: Use convolution filters to extract feature maps

Brief history: CNN



Gradient-based learning applied to document recognition, LeCun, Bottou, Bengio, Haffner, 1998

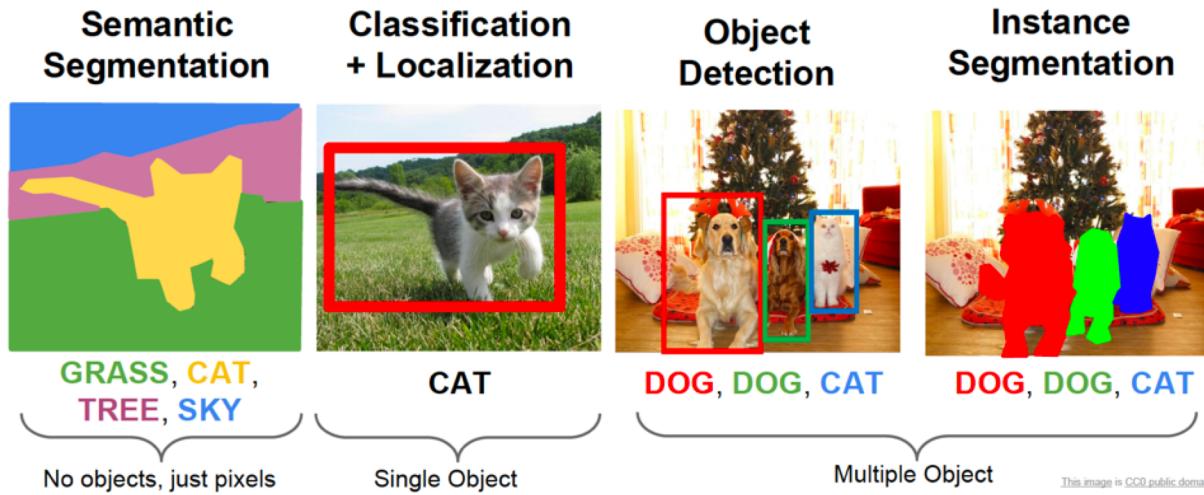
Brief history: CNN (Revolutionary)



ImageNet classification with Deep Convolutional Neural Networks,
 Krizhevsky, Sutskever, Hinton, 2012

Today: CNN

It's everywhere: from image classification and localization to image retrieval; from object detection to image segmentation; image generation; and many more...

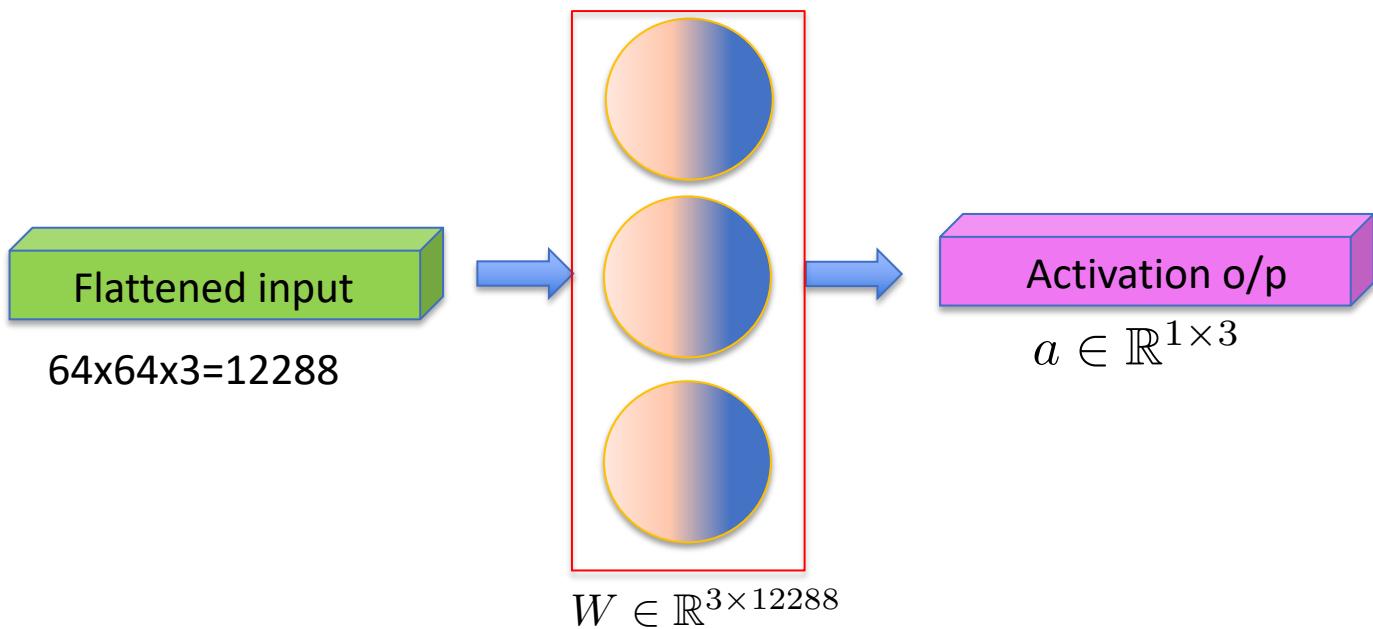


This has to do a lot with advanced GPUs

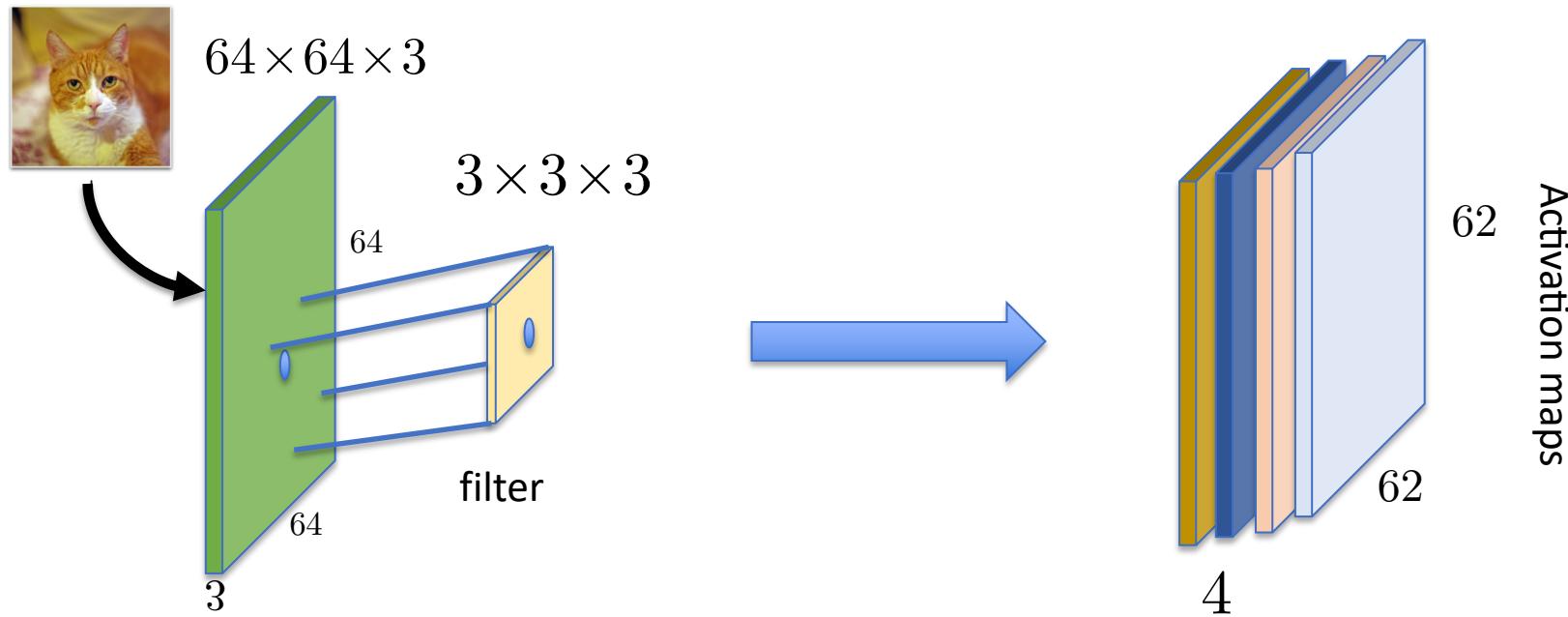


CNN

Recall, fully connected layer

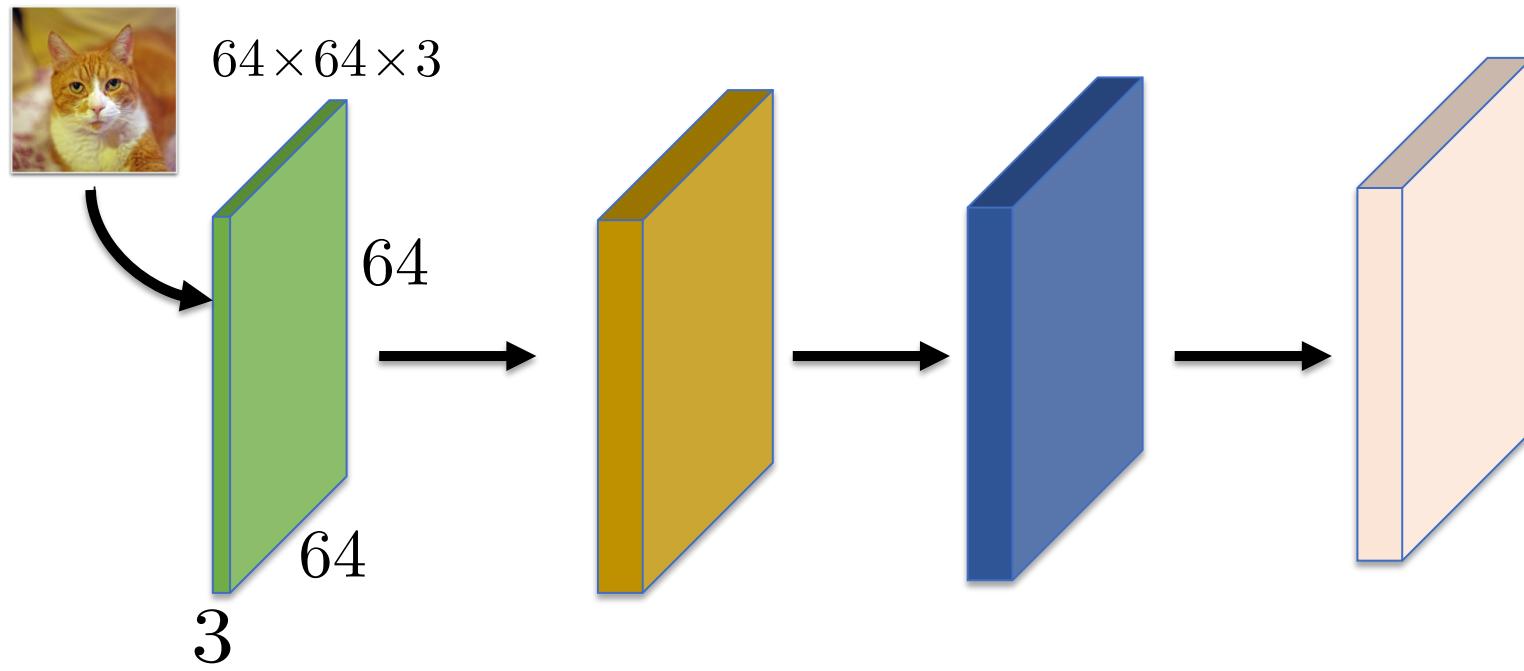


CNN



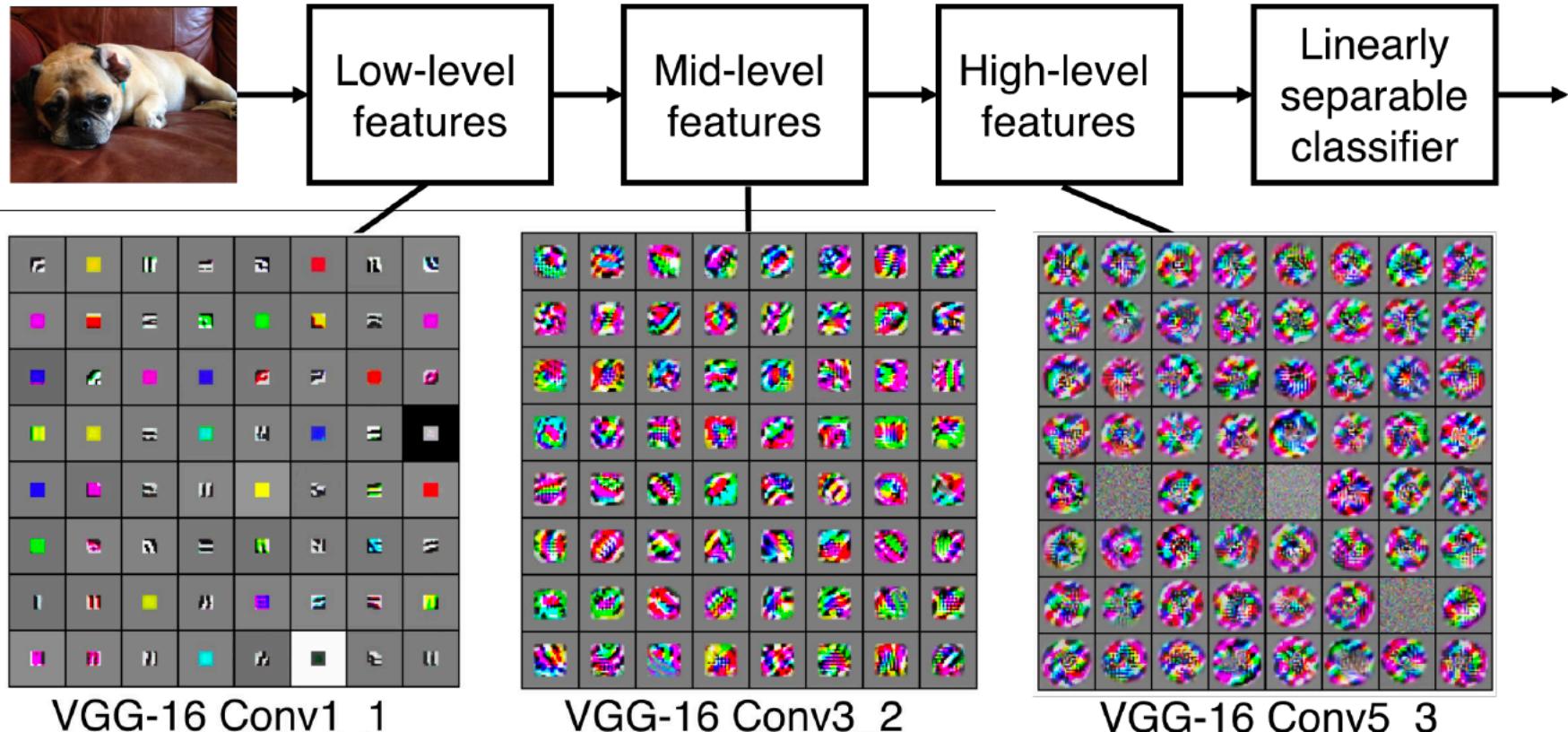
- Convolve the filter with the images, that is, slide your filter over the images spatially
- No. of parameters = #no. of filters * ($27 + 1$ (bias))

CNN: Model?



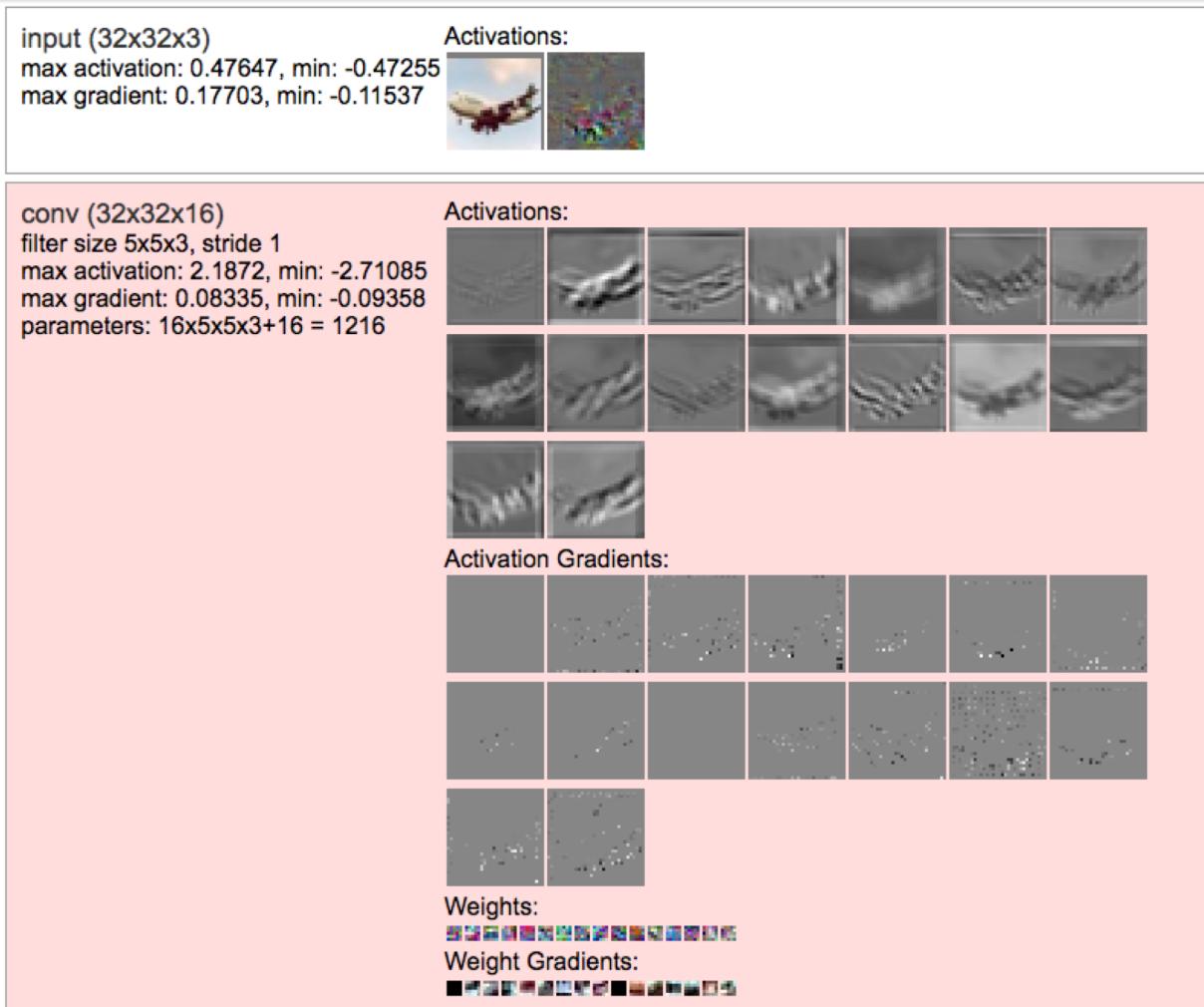
Suppose you want to convolve above image with **5x5x3** filters and you want to have **6-10-10** maps. Compute:
a) No. of parameters and b) dimension of your activation maps

CNN: What do they learn?



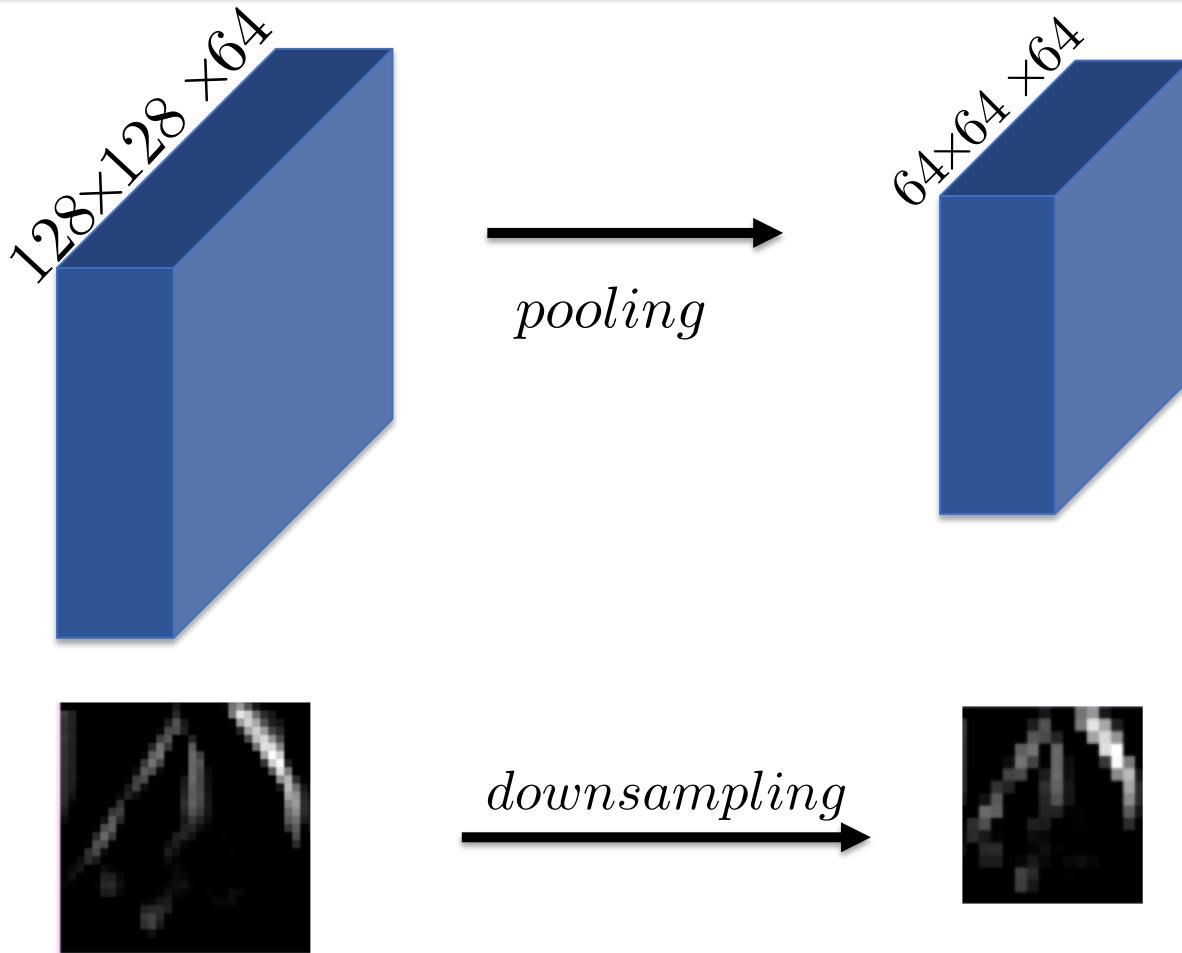


CNN: What do they learn?



Play here: [cnnsWithCifar10:onlineDemo_karpathy](#)

CNN: Pooling

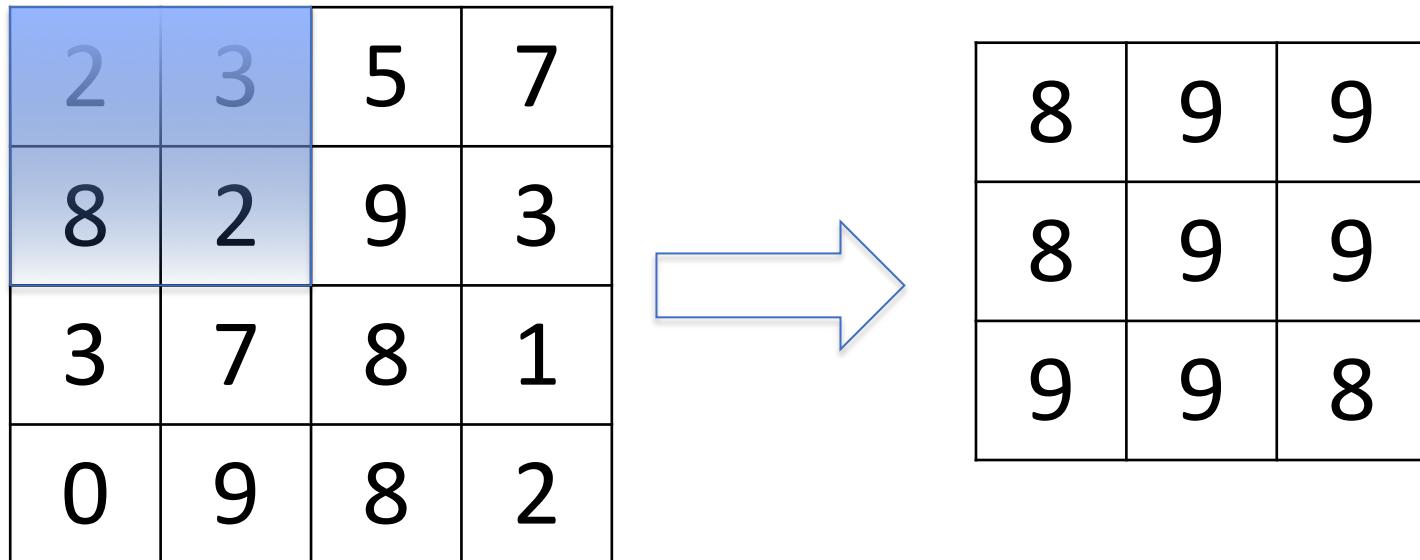


Note: Each channel in activation map is down-sampled independently

CNN: Max pooling

Concept

- Take maximum value in a given filter and stride e.g.
2x2 filter with stride 1



- Try with 2x2 filter with stride 2 and stride 3



CNN: FC layer

- Last layer always contains neurons that connect to the entire input volume
- Common practice is to add several pooling layers before the last layer
- Typical architecture is:
 - $[(\text{Conv-RELU})..(\text{Conv-RELU}) * \text{POOLING}]^* -- (\text{FC-RELU})^* K$,
Softmax
 - But, not only limited to above

CNN: Dropout

Concept

- Randomly knocking out units in your neural network
- That is, using smaller network
- So, can't rely on any one feature so the weights get spread out
- This you can do it at a particular layer/vary your dropout differently at different layers
- This effect in regularization and avoid from over-fitting

CNN: Data augmentation

Concept

- Increase your training data by performing transformations: e.g., color transformation, scaling, rotations, translations, shear etc
- You can do this on fly during your training (<https://keras.io/preprocessing/image/>)
- This allows regularization as well and you might get better accuracy

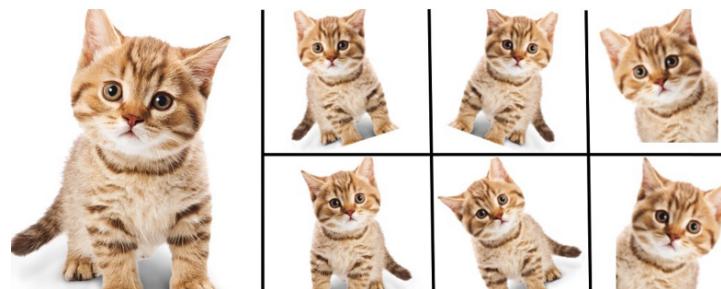


Image source: <https://medium.com/nanoneats>

- Widely used (<https://keras.io>)
 - Easy to construct your network
 - Allows easy and fast prototyping
 - Runs both on CPU and GPU
 - High-level neural network API and tensorflow backend



Keras: How easy?

- Design your model

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=input_shape))
model.add(BatchNormalization(momentum=0.95))
model.add(Activation('relu'))

model.add(Conv2D(64, kernel_size=(3, 3)))
model.add(BatchNormalization(momentum=0.95))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3)))
model.add(BatchNormalization(momentum=0.95))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3)))
model.add(BatchNormalization(momentum=0.95))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# model.add(Flatten()) # we can flatten or use a global pooling scheme.
model.add(GlobalAveragePooling2D())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

- Compile your model

```
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
```

- Fit your model

```
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))
```

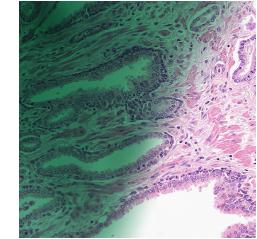
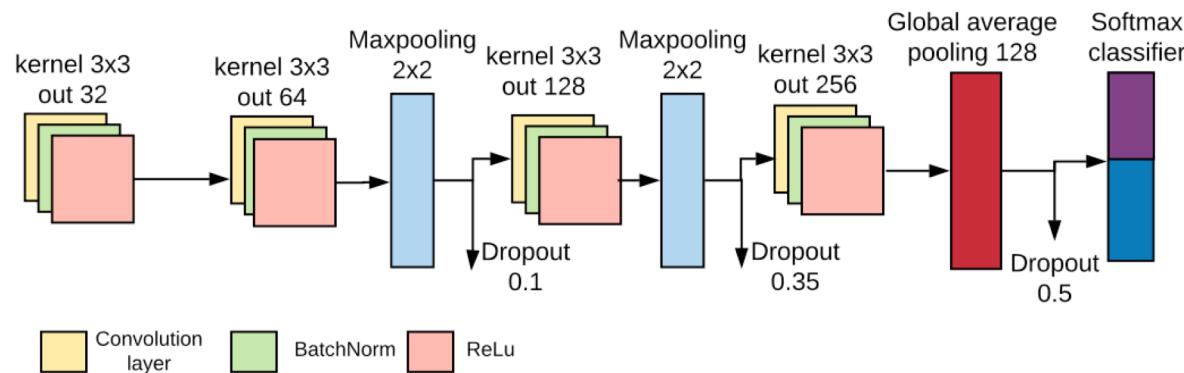
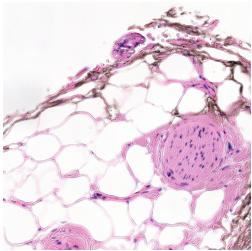
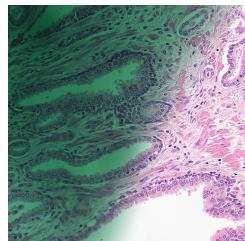
Tensorflow and PyTorch



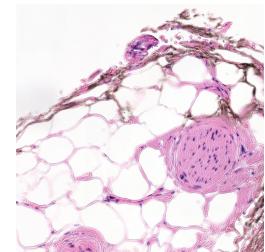
- Worth learning for development
 - Both allows you to develop each component of your network
 - Tensorflow is better for deployment especially for real-time applications
- We will show you how to get started with tensorflow as well

Example application

- Ink Removal in histopathology data



Ink:1



no-Ink:0

Recap

- Perceptron
- Deep neural networks
- Convolutional neural networks
- Applications
 - Cat vs non-cat
 - MNIST digit dataset (classify 10 different digits)
 - Binary classification of marker ink vs non-ink in histopathology data

Exercises

- Now, lets look at L-layer neuron network and see how this improves test accuracy
- Lets implement using Keras and TensorFlow libraries
- Lets discover the limitation of small size data and see advantage of large size data

Datasets can be downloaded here:

<https://s3.amazonaws.com/teachingv3.0/datasets.zip>

You can also try ink-no-ink classification (complimentary)

https://s3.amazonaws.com/teachingv3.0/ink_removal_data.zip



K Keras



UNIVERSITY OF
OXFORD

Practical

Project: You will try to find publicly available medical dataset for multi-class or binary classification or use any one of below:

a) Grand-challenge datasets

<https://grand-challenge.org/challenges/>

<https://ead2019.grand-challenge.org> (multi-class problem and bounding box detection)

b) Kaggle datasets (you can even win cash prizes here!!!)

<https://www.kaggle.com/c/histopathologic-cancer-detection>

<https://www.kaggle.com/c/diabetic-retinopathy-detection/data>

<https://www.kaggle.com/c/rsna-pneumonia-detection-challenge> (multi-class problem and bounding box detection)

c) Malaria dataset released by Dr. Stefan Jaeger (<https://ceb.nlm.nih.gov/repositories/malaria-datasets/>). We have provided you data which contains downscaled 64x64x3 h5 compressed 27,558 images. Tasks are detailed in the provided ipython notebook.

*Note: Beware of computation power that you might need to work with these data. For learning, you will scale down all data that you intend to use.

Good reads!

- “*Hands-On Machine Learning with Scikit-Learn and TensorFlow*” by Aurélien Géron
- “*Deep learning*” by Ian Goodfellow, Yoshua Bengio and Aaron Courville

You can look more lists here:

<https://www.bonkersabouttech.com/ai/best-deep-learning-books/494>