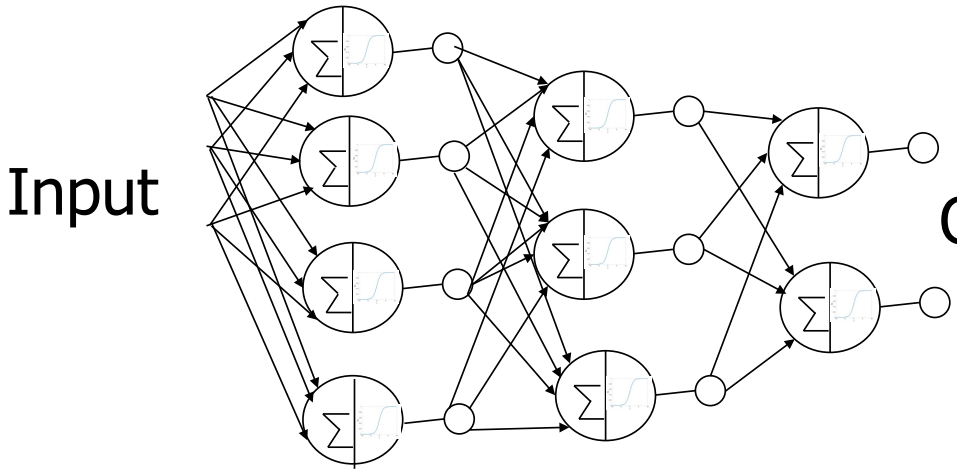
Deep neural architecture and applications (part I) (Week 12: Lecture 55)

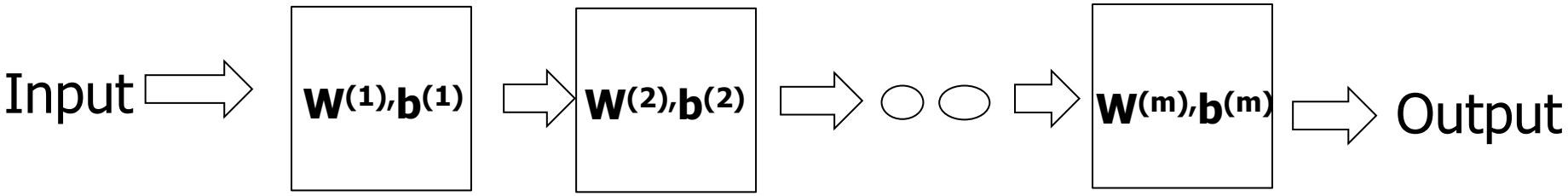
Jayanta Mukhopadhyay
Dept. of Computer Science and Engg.

Courtesy: K. Sairam

Classical vs. Deep Architecture



Classical ANN:
Only a few
hidden layers.



Deep architecture:
Could have more than
hundred hidden layers.

Application: Image Classification

ImageNet Large Scale Visual Recognition Challenge.

- 1000 object classes and 1.4M Images in the dataset.
- Major algorithms submitted deep features based.

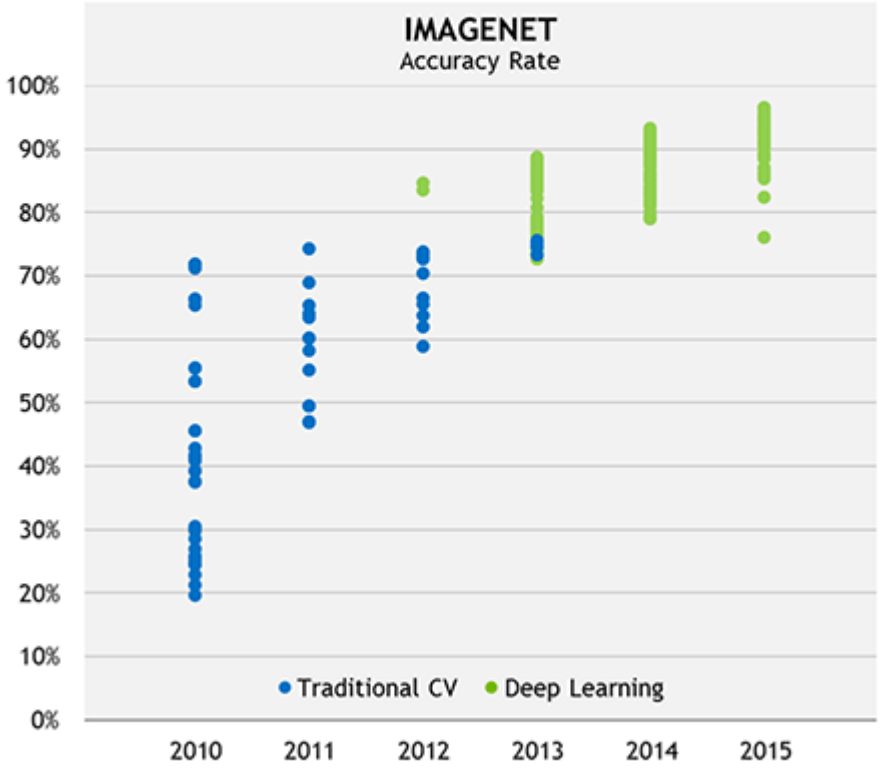
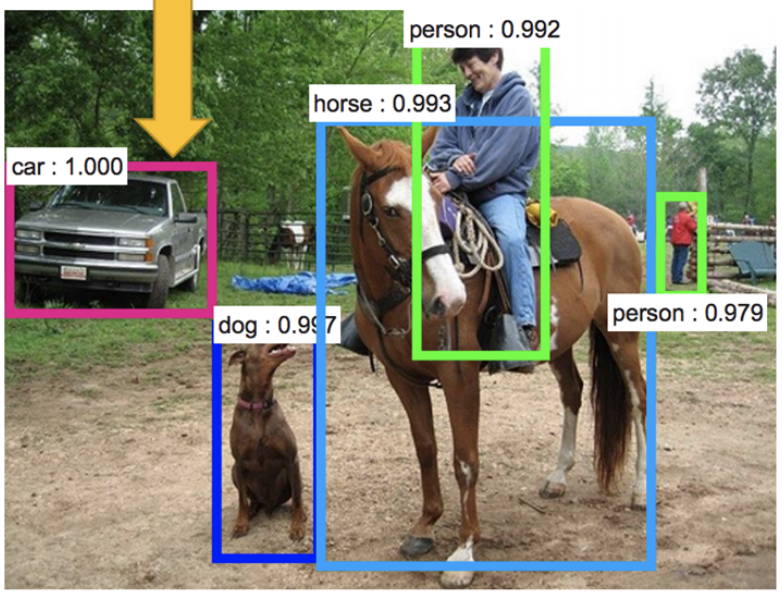


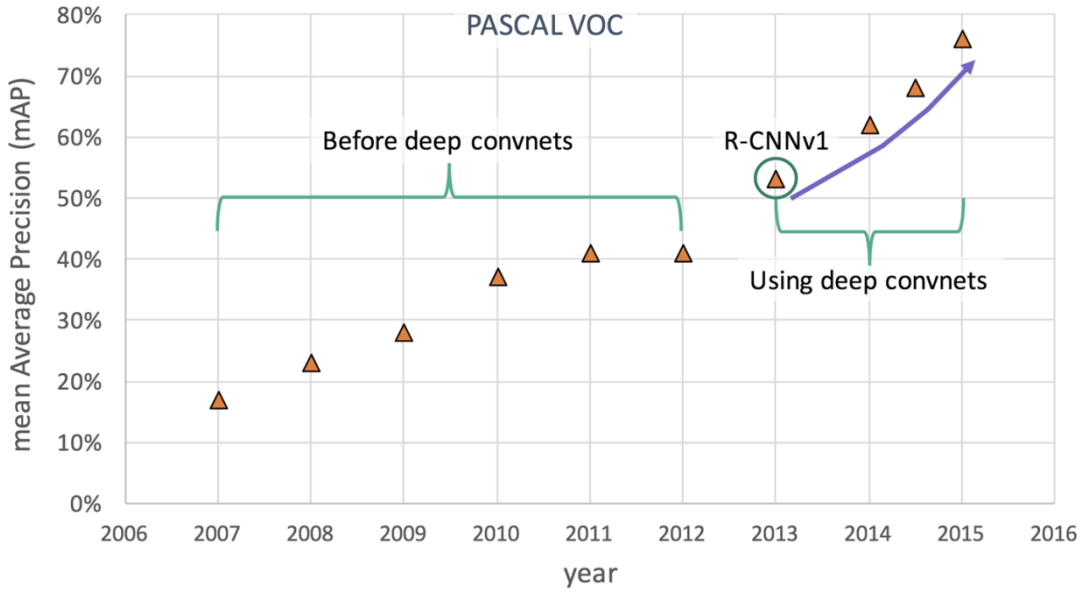
Image: https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus

Application: Object Recognition and Localization

Localization
Where?

Recognition
What?



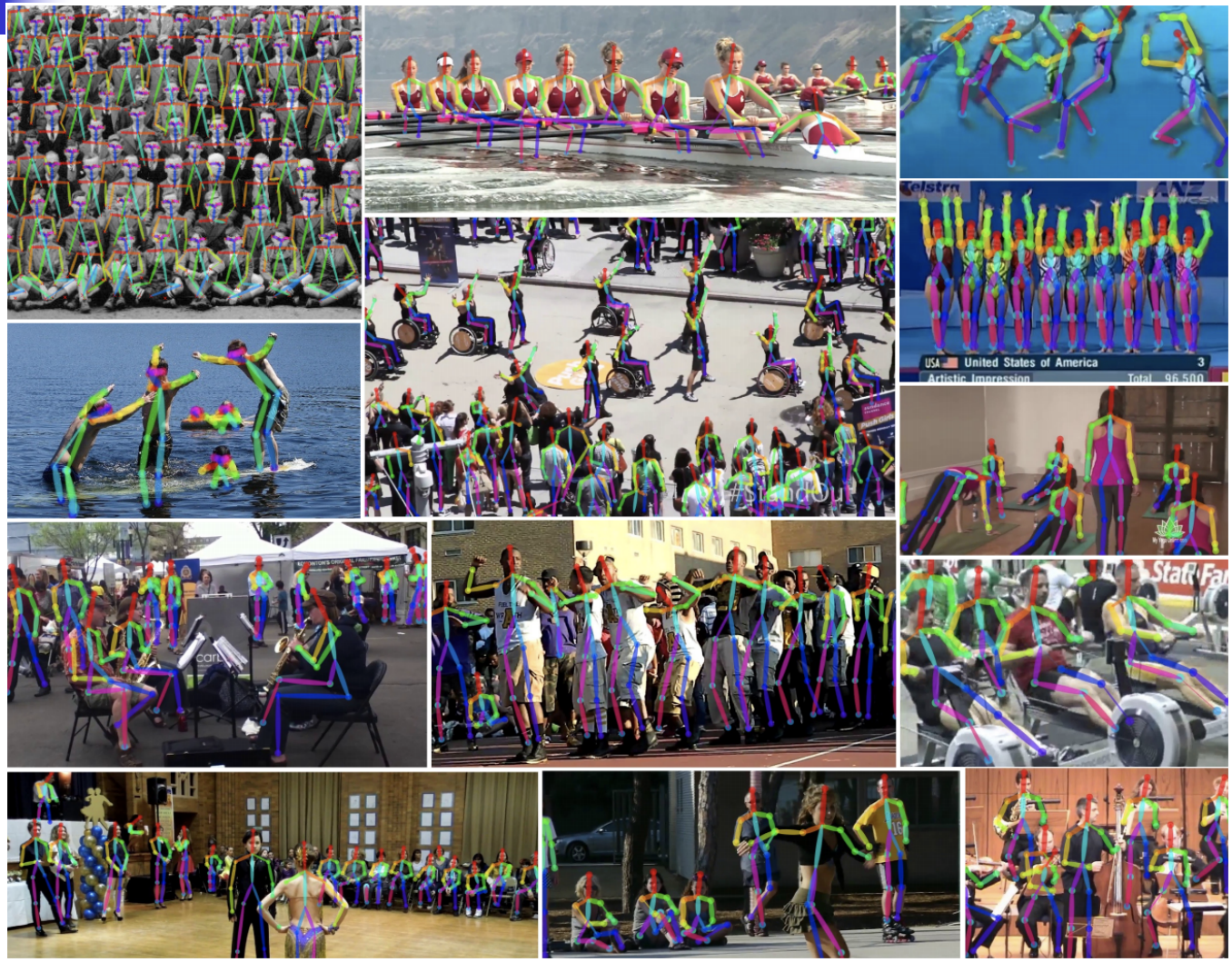


Slide credit: Ross Girshick, ICCV 2015

Application: Object Segmentation

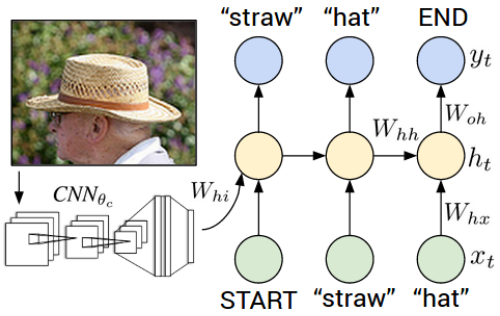
*Dai, He, and Sun, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, CVPR 2016

Application: Pose Estimation



Cao et al, "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", arXiv 2016

Application: Image Captioning



"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

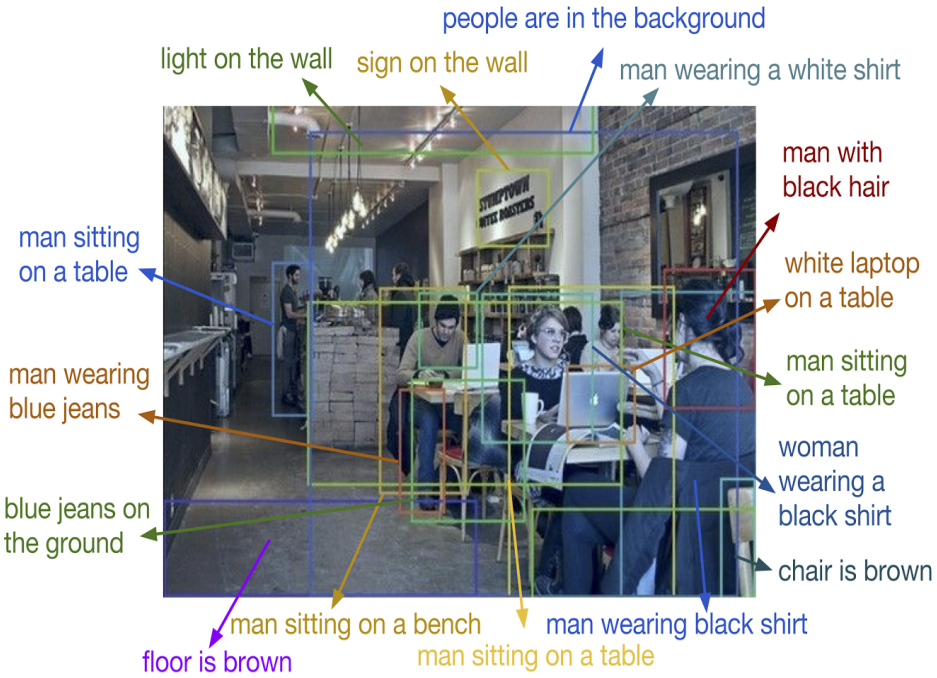
"black and white dog jumps over bar."

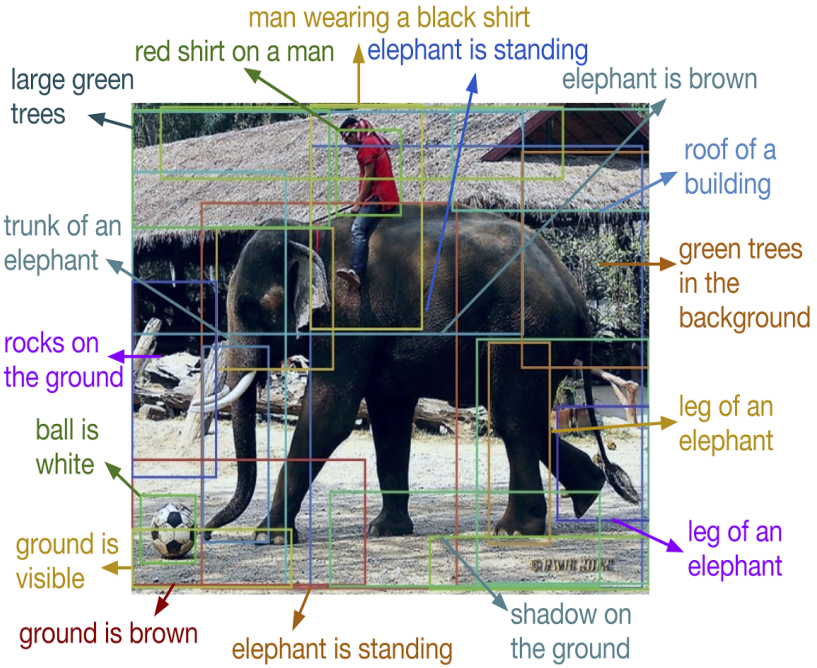
"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

* Karpathy and Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions”, CVPR 2015

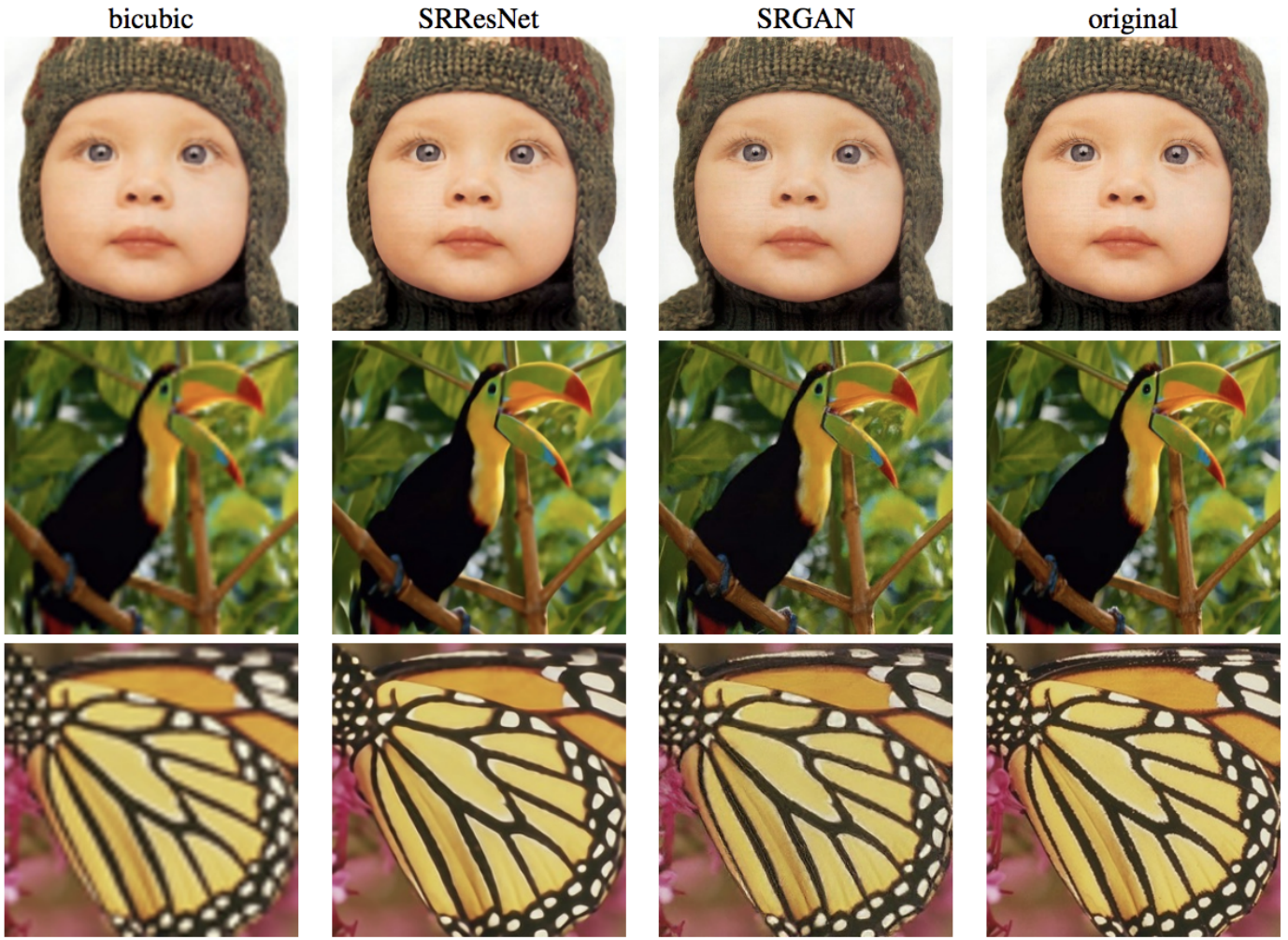
Application: Dense Image Captioning





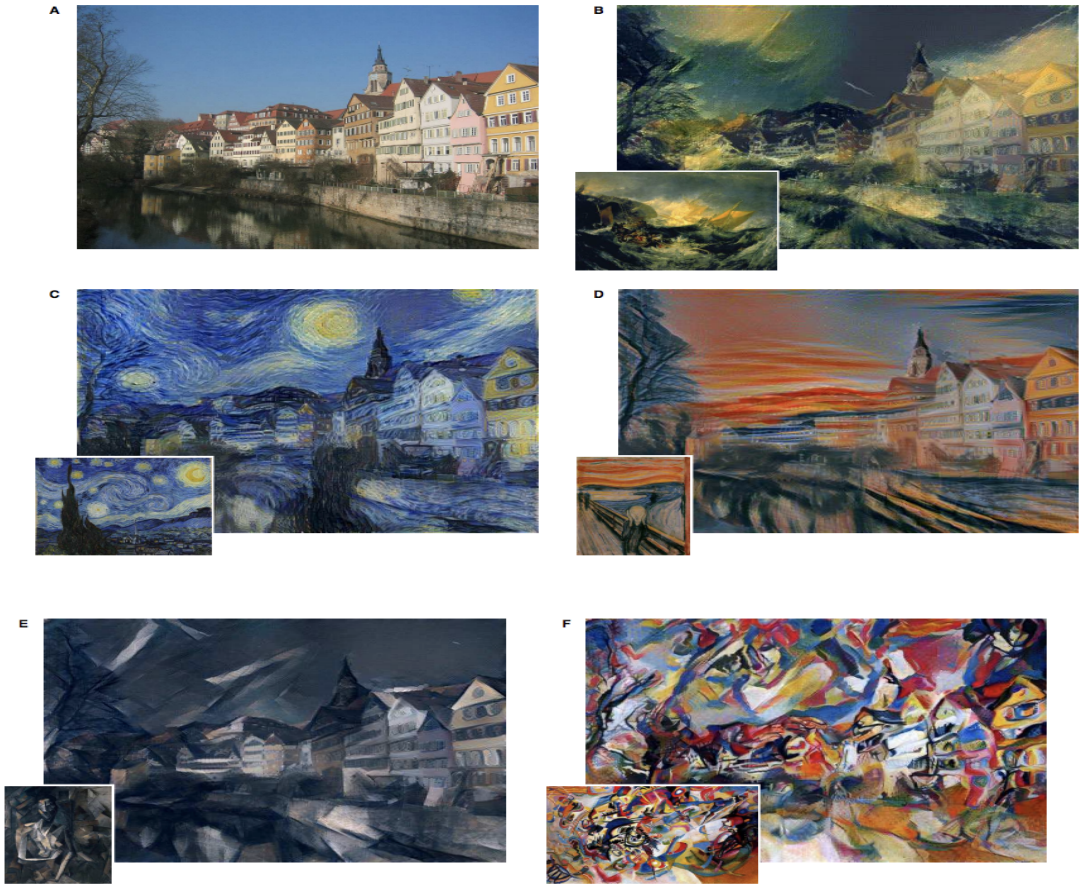
*Johnson, Karpathy, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016

Application: Super Resolution

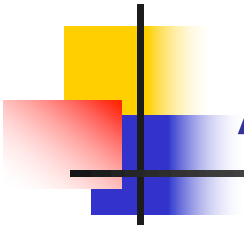


*Ledig et al, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, arXiv 2016

Application: Image Art Generate & Transfer



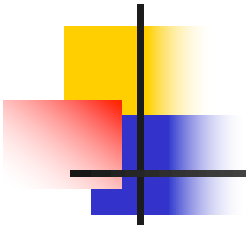
*Gatys, Ecker, and Bethge, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016



Application: Outside Computer Vision

- Machine Translation & Text Synthesis
- Speech Recognition & Synthesis
- Navigating Autonomous Vehicles
- Playing Atari like Games
- Alpha Go

Deep architecture: Why so late?



- Concepts introduced in 80's.
- Basic principles remain the same.
- Two major reasons.
  - Availability of large scale annotated data.
    - Penetration of internet and smart phones.
    - Wide spread of social networking.
    - Online shopping, etc.
  - Advancement of computing power.
    - High throughput GPU computing.

Classical Image Classification

hand-crafted feature extractor

Classifier Algorithm

output

Tiger?

Cat?

Lion?

- Edges
- SIFT/SURF key Point
- HOG Regional Features
- Motion Features, etc.

- Bayesian
- LDA
- SVM
- KNN

Classification Challenges

- Very tedious and costly to develop hand-crafted features to handle various challenges.

View Point variation

Deformation

Occlusion

Intraclass Variation

Illumination

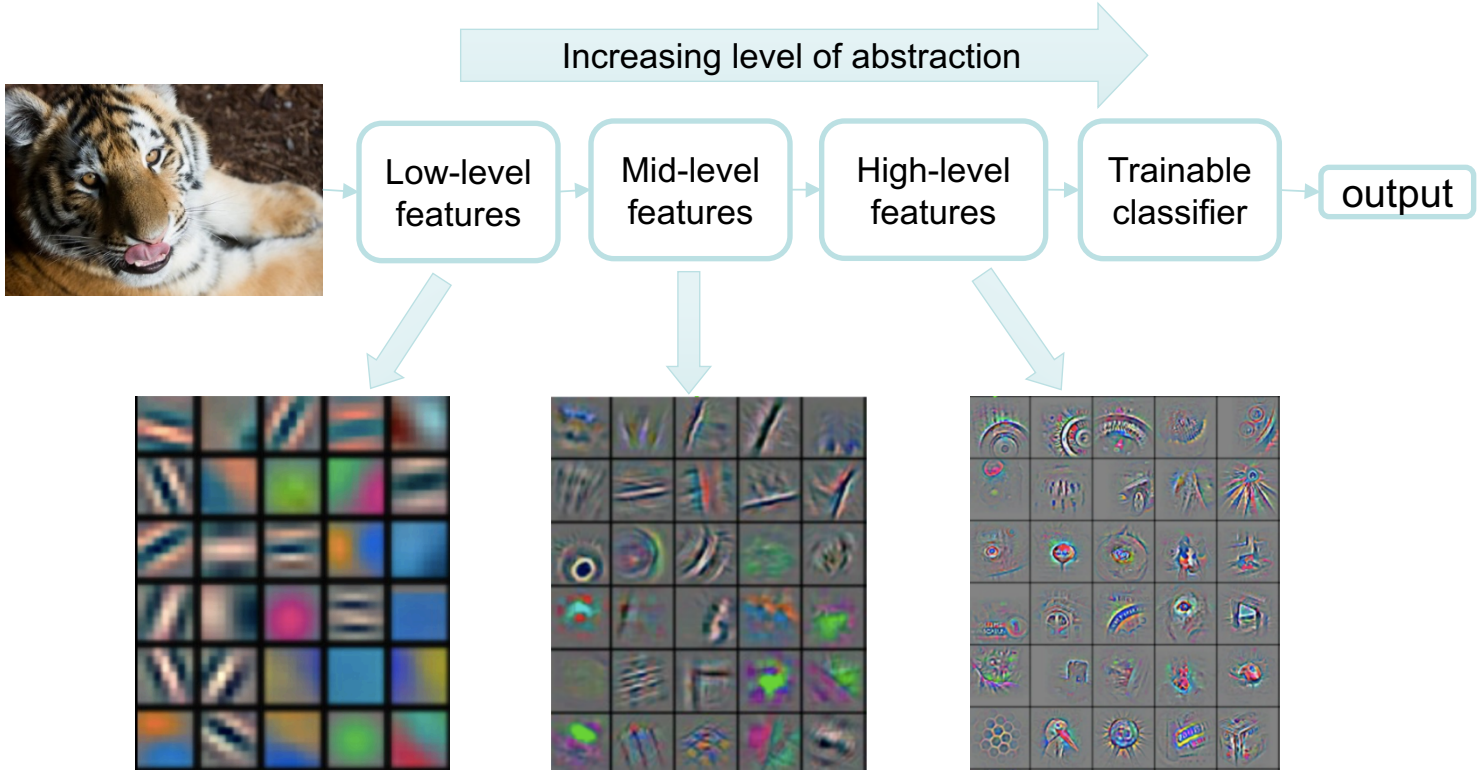
Clutter

Instances

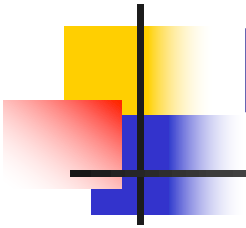
Scale

Highly dependent on one application, and not transferable easily to other applications.

Deep Features



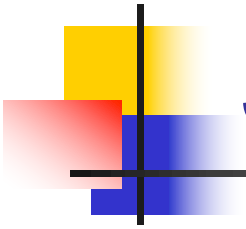
*Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)



Deep Features

- Utilize large amount of training data to learn features.
- Rich hierarchical representations are learnt fast through multiple stage of feature learning process.
- Learned features are easy to adapt.

*Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)



Supervised Learning

Data: (x, y) where x is data, y is label

Goal: Learn a function f to map $x \rightarrow y$

Examples: Classification, Regression, Object detection, Semantic Segmentation, Image Captioning, etc.

Image Classification

What class that image belongs to? How to classify?

Classify

Tiger

Cat

Lion

Learn a parametric function f composed by weight parameters w model to classify Image x as class label y.

Supervised Learning

Data Driven Approach to learn the model in three steps:

Step 1: Define Model

$$\hat{y} = f(x, w)$$

Predicted output (image label)

Model structure

Input data (Image pixels)

Model weights

Supervised Learning

Step 2: Collect data.

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Training input

True output

Supervised Learning

Step 3: Learn the model.

Total Loss = Data Loss + Regularization Loss

Predicted output

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)$$

Learned weights

Minimize average loss over training set

Loss function: Measures "badness" of prediction

Regularizer: Penalizes complex models

Loss

- A **loss function** tells how good our current classifier is.
- **Data loss**: Model predictions should match training data
  - **Multiclass SVM loss** (Hinge Loss)**:**

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1)$$

  - **Softmax Loss** (Multinomial Logistic Regression):

$$L_i = -\log P(Y = y_i|X = x_i) \qquad L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

Cross-entropy Loss

- Another form of softmax loss.
  - **2-class entropy:**
    - $-(y \log(p)+(1-y) \log(1-p))$; p: *Prob.* $(y=1|o)$
  - **Multiclass:**

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Estimated Prob. of o belonging to c

Binary indicator (1 if o belongs to c, else 0).

True Prob. of o belonging to c

More general:

$$-\sum_{c=1}^{M} q_{o,c} \log(p_{o,c})$$

Regularization Loss

i

- **Regularization Loss**: Model should be "simple", so it works on test data as "W" is not unique with just data loss.
  - L_2 Regularization (Weight Decay) $R(W) = \sum_k \sum_l W_{k,l}^2$
  - L_1 Regularization $R(W) = \sum_k \sum_l |W_{k,l}|$
  - Elastic net (L_1 + L_2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

How to find best weights w^* ?

$$w^* = \arg\min_w \frac{1}{N} \overbrace{\sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)}^{g(w)}$$

$$= \arg\min_w g(w)$$

- Random Search
  - Inefficient in higher dimensions
  - Gradient descent
    - Back propagation algorithm

Gradient Descent

How to update weights?

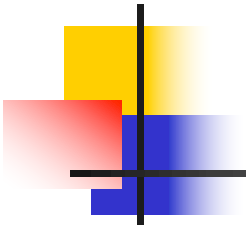
Initialize w randomly
While true:
    Compute gradient $\nabla g(w)$ at current point

    Move downhill a little bit: $w = w - \alpha \nabla g(w)$

updating the weights at each iteration

Learning rate: How big each step should be

For more details on brief details, ref to http://ruder.io/optimizing-gradient-descent/



Back Propagation

- **Forward pass**:
  - Run graph "forward" to compute loss
- **Backward pass**:
  - Run graph "backward" to compute gradients with respect to loss
- Efficient to compute gradients for big, complex models.

Supervised Learning: Linear regression

$x_i \in \mathbb{R}^{D_{in}}$ $y_i \in \mathbb{R}^{D_{out}}$

Input and output are vectors

$$\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2$$

Loss is Euclidean distance

Linear Regression

$$f(x, W) = Wx$$

$W \in \mathbb{R}^{D_{out} \times D_{in}}$

Model is just a matrix multiply

$$R(W) = \lambda \|W\|_{fro}^2$$

Regularizer is Frobenius norm of matrix (sum of squares of entries)

Learning Problem

$$W^* = \arg\min_W \frac{1}{2N} \sum_{i=1}^{N} \|Wx_i - y\|_2^2 + \lambda \|W\|_{fro}^2$$

Supervised Learning: Neural Network

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

$$f(x, W_1, W_2) = W_2 W_1 x$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

$$W = W_2 W_1$$

Model is **two** matrix multiplies
which is again as Linear Regression

Supervised Learning: Neural Network

$$x_i \in \mathbb{R}^{D_{in}} \qquad y_i \in \mathbb{R}^{D_{out}}$$

Input and output are vectors

Linear Regression

$$f(x, W) = Wx$$

$$W \in \mathbb{R}^{D_{out} \times D_{in}}$$

Model is just a matrix multiply

New Model

~~$f(x, W_1, W_2) = W_2 W_1 x$~~

$$f(x, W_1, W_2) = W_2 \sigma(W_1 x)$$

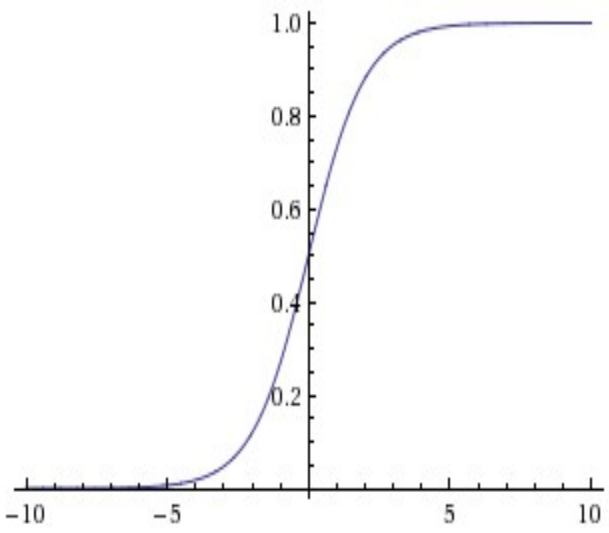
$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

Model is **two** matrix multiplies, with an **elementwise nonlinearity**

$$\sigma : \mathbb{R}^H \to \mathbb{R}^H$$

Non Linearity: Activation Functions

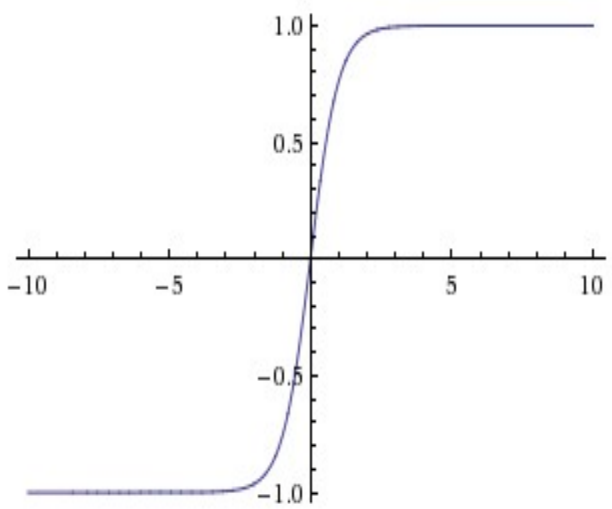


Sigmoid

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

- Squashes numbers to range [0,1] – can kill gradients.
- Best for learning "logical" functions – i.e. functions on binary inputs.
- Not as good for image networks
- Not zero-centered

Activation Function

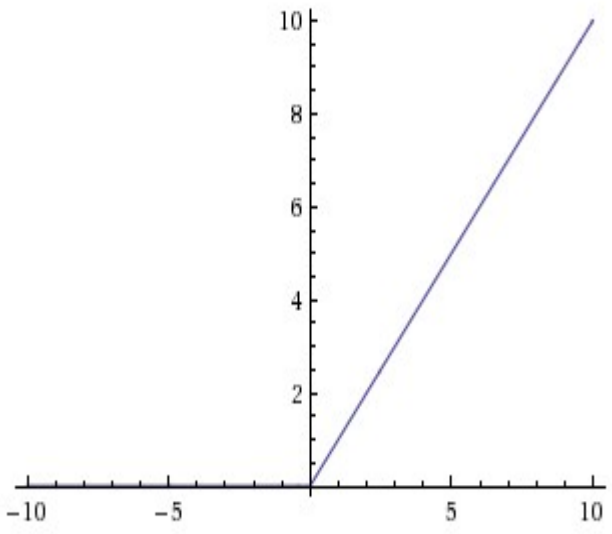


tanh(x)

- Squashes numbers to range [-1,1]
- Zero centered (desirable)
- Still kills gradients when saturated
- Not as good for binary functions

*LeCun et al., 1991

Activation Function

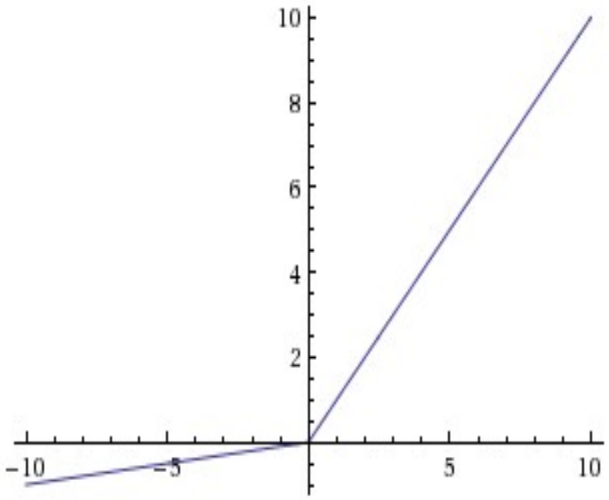


ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**
- Does not saturate (in +region)
- Converges faster than sigmoid/tanh on image data (e.g. 6 times).
- Very computationally efficient
- Not suitable for logical functions
- Not for control in recurrent nets
- Not zero-centered output
- Dead ReLU never activates as gradient is 0 for x < 0. So, no filter update!

*Krizhevsky et al., 2012

Non Linearity: Activation Functions



- Does not saturate
- Converges faster than sigmoid/tanh on image data(e.g. 6 times.)
- **will not "die".**

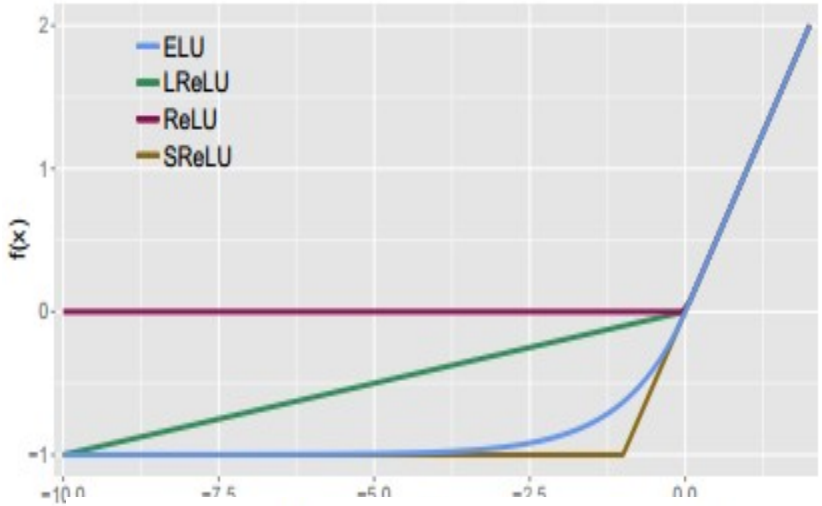
Leaky ReLU

$$f(x) = \max(0.01x, x)$$

*Mass et al., 2013

Non Linearity: Activation Functions

Exponential Linear Units (ELU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

- All benefits of ReLU
- Does not die
- Closer to zero mean outputs

*Clevert et al., 2015

Non Linearity: Activation Functions

MaxOut Neuron

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

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Does not saturate!
- Does not die!
- doubles the number of parameters / neuron

*Goodfellow et al., 2013

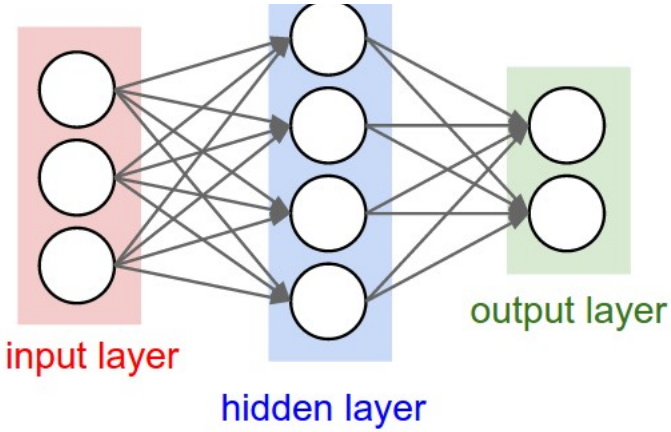
Neural Network

Two Layer Neual network / One hidden layer Neural Network

$$f(x, W_1, W_2) = W_2\sigma(W_1 x)$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$



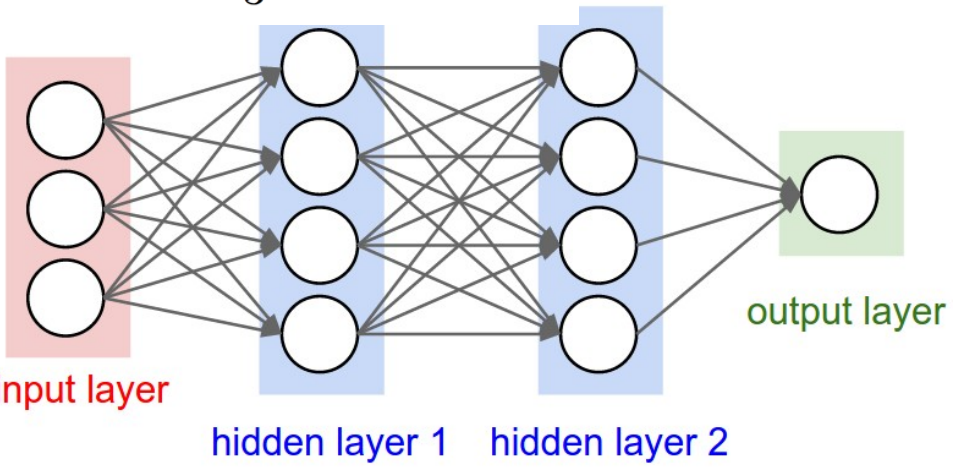
Three Layer Neural Network / Two hidden layer Neural Network

$$f(x, W_1, W_2, W_3) = W_3\sigma(W_2(\sigma(W_1 x)))$$

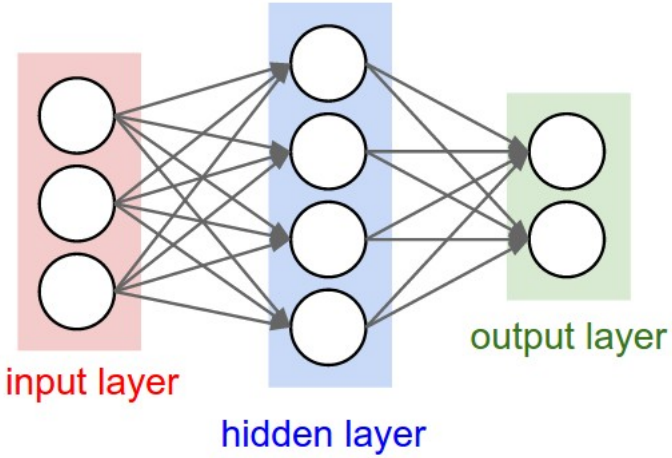
$$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$$

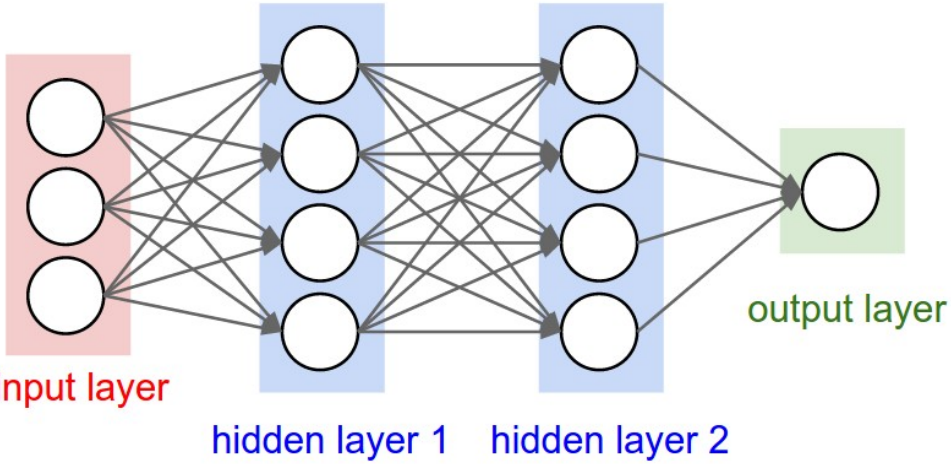
$$W_2 \in \mathbb{R}^{H_2 \times H_1}$$

$$W_3 \in \mathbb{R}^{D_{out} \times H_2}$$



Neural Network





Sometimes Multilayer called "Fully-Connected Network" or "Perceptron"

Hidden layers are **learned feature representations** of the input! These are **Deep Features!**