Deep Learning

The Revolution

Deep learning has provided **empirically much better** methods for:

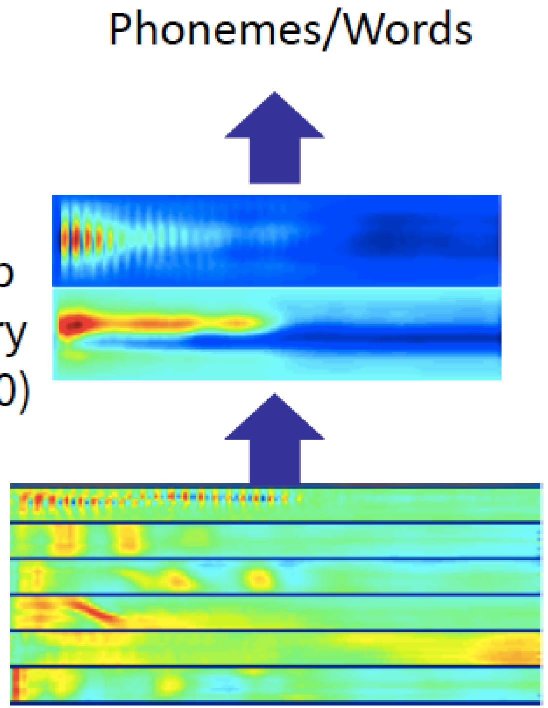
- Hard prediction problems
- Generative models of natural data distributions

Especially over high-dimensional data such as images, video, speech, text, and robotic control

Deep Learning for Speech Recognition

- The first breakthrough results of "deep learning" on large datasets happened in speech recognition
- Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition. Dahl et al. (2010)

Acoustic model Measuring WER	RT03S FSH	Hub5 SWB
Traditional (GMM) (2012)	27.4	23.6
Deep Learning (Dahl et al. 2012)	18.5 (−33%)	16.1 (−32%)
Xiong et al. (2017)		5.8



IMAGENET

www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities

Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

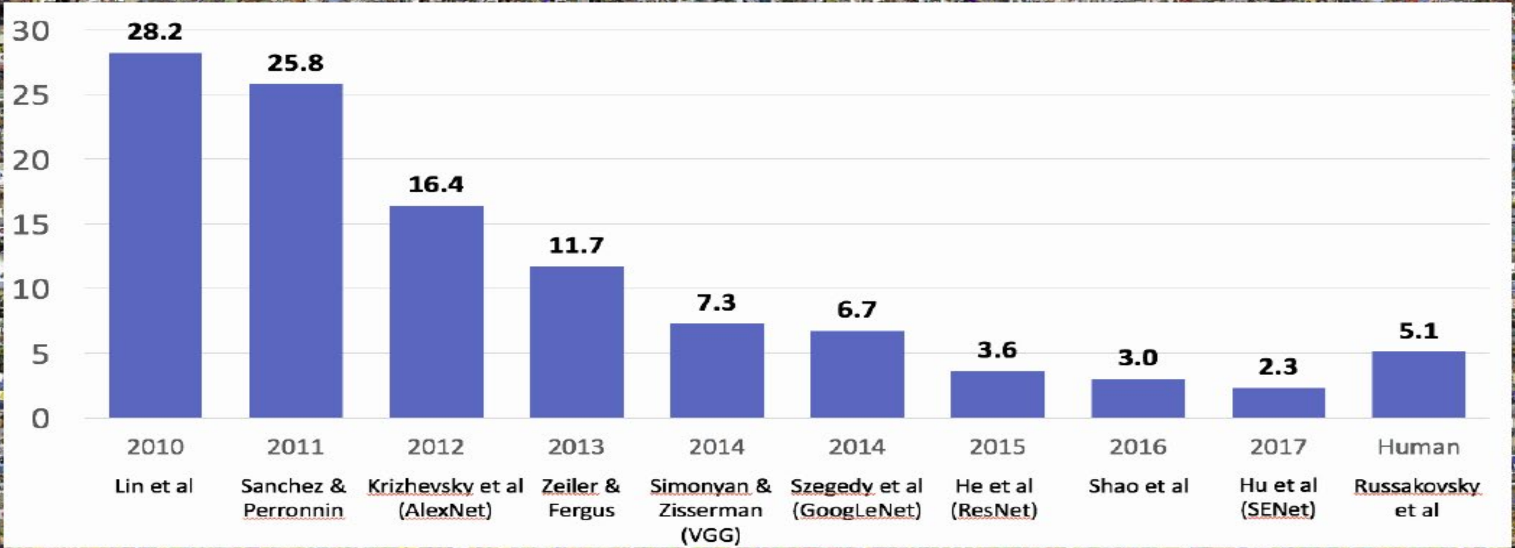
✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:
1,000 object classes
1,431,167 images

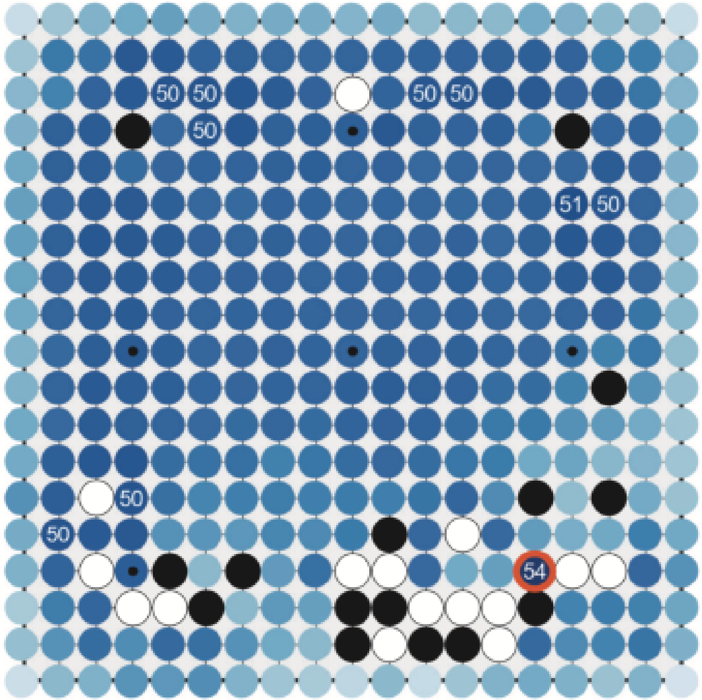


Russakovsky et al. IJCV 2015

Reinforcement Learning (AlphaGo)

The model works from a 319-dimensional input representing the board and uses a regression model to score potential next moves

Combined with Monte Carlo Tree Search, this “solved” Go much more quickly than anyone had been imagining



Recipe for winning Kaggle competitions

1. Careful data preprocessing, cleaning, augmentation, and feature engineering (this hasn't gone away to win Kaggle!)
2.
   a. **For classic, structured data tables: Gradient-boosted decision trees** (xgboost). Roughly, improved MART.
   b. **For "unstructured" text, images, video, speech: Neural networks**
3. Ensembling/stacking of models, with careful cross-validation testing to find best final configuration

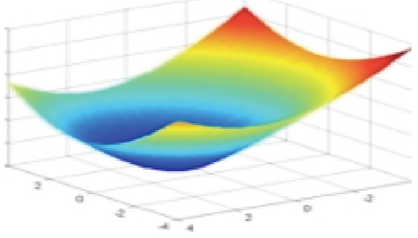
What is deep learning (DL)?

- **Deep learning** is a subfield of **machine learning (statistics?)**

- Most machine learning methods work well because of **human-designed input features or representations**
  - SIFT or HOG features for vision
  - MFCC or LPC features for speech
  - Features about words parts (suffix, capitalized?) for finding person or location names

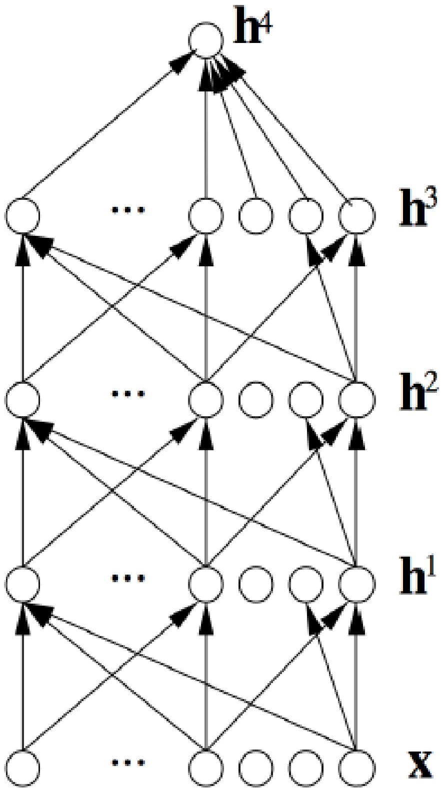
Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

- Machine learning becomes just optimizing weights to best make a final prediction

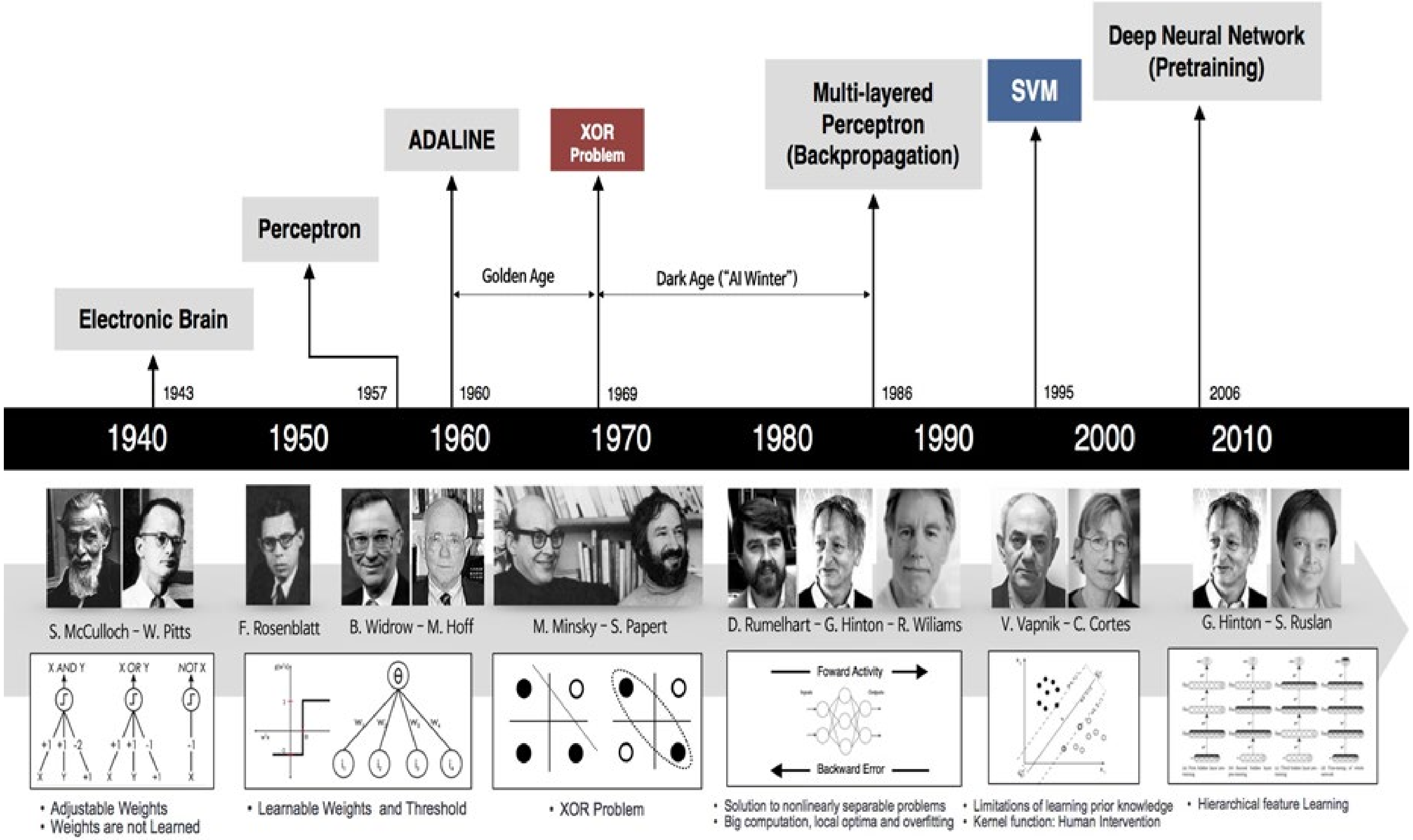


What is deep learning (DL)?

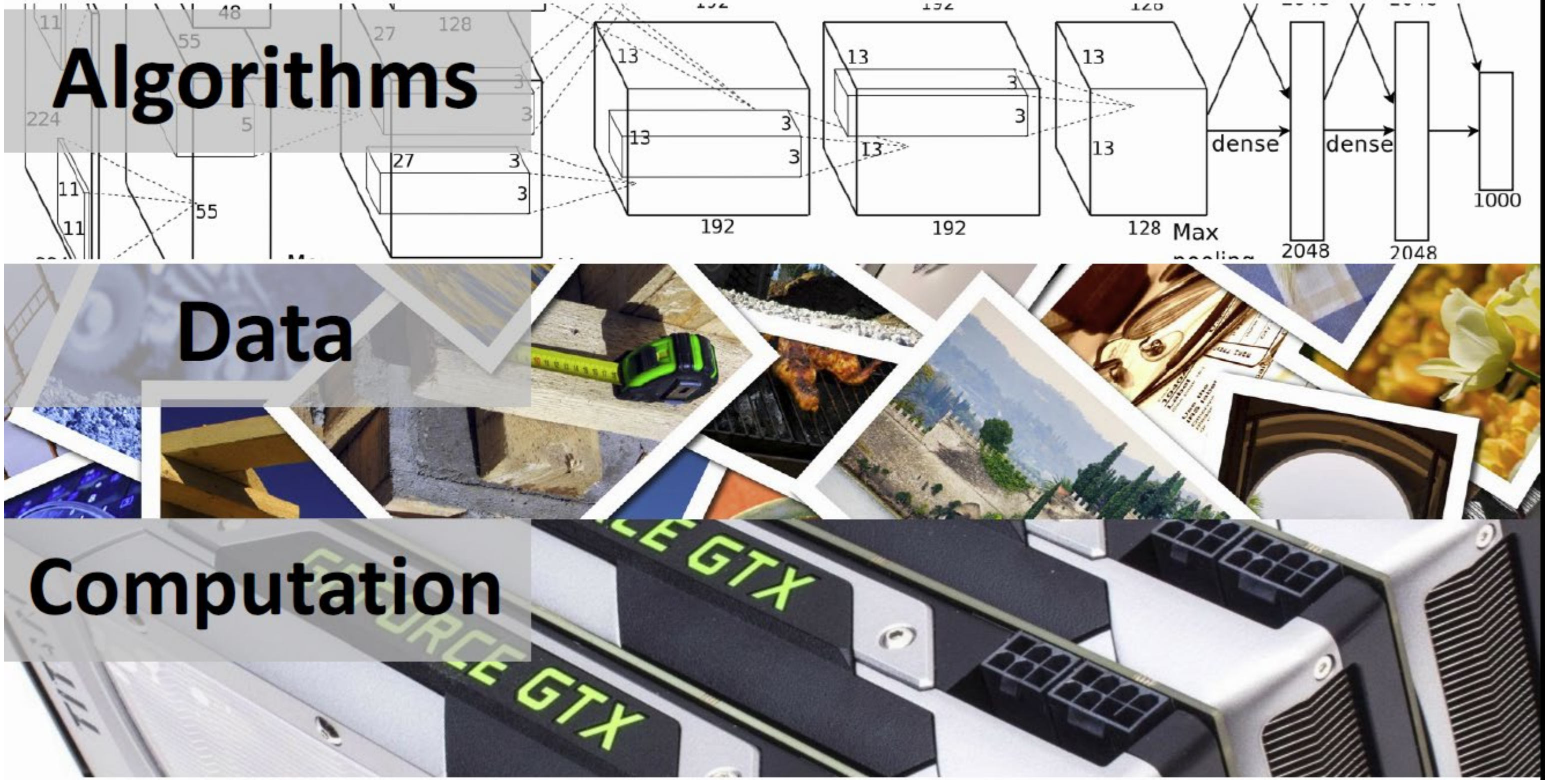
- In contrast to standard machine learning,
- Representation learning attempts to automatically learn good features or representations
- Deep learning algorithms learn multiple levels of representations (here: h^1, h^2, h^3) and an output (h^4)
- From "raw" inputs **x** (e.g. sound, pixels, characters, or words)
- Neural networks are the currently successful method for deep learning
- A.k.a. "Differentiable Programming"



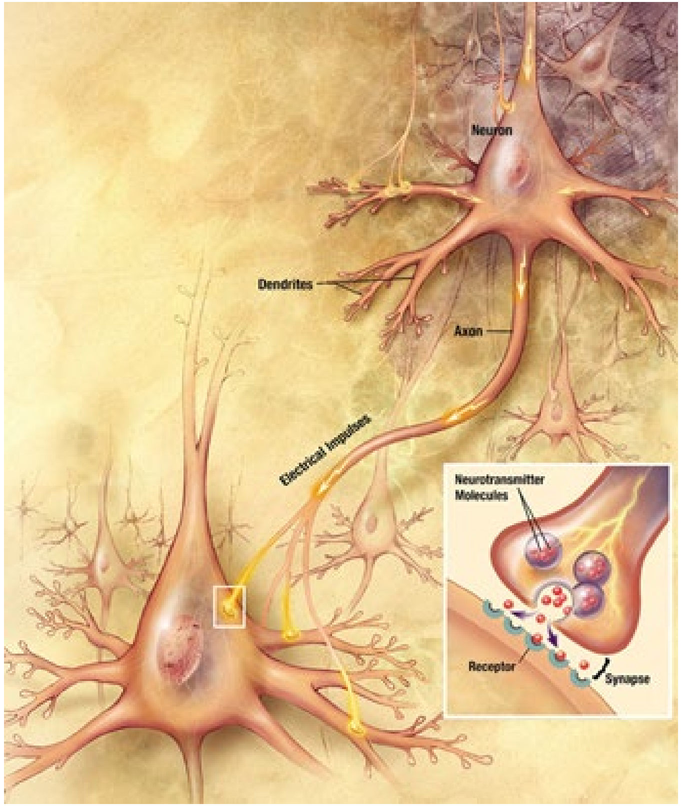
A Brief History of Neural Nets



Ingredients for Deep Learning



Neurons in the brain

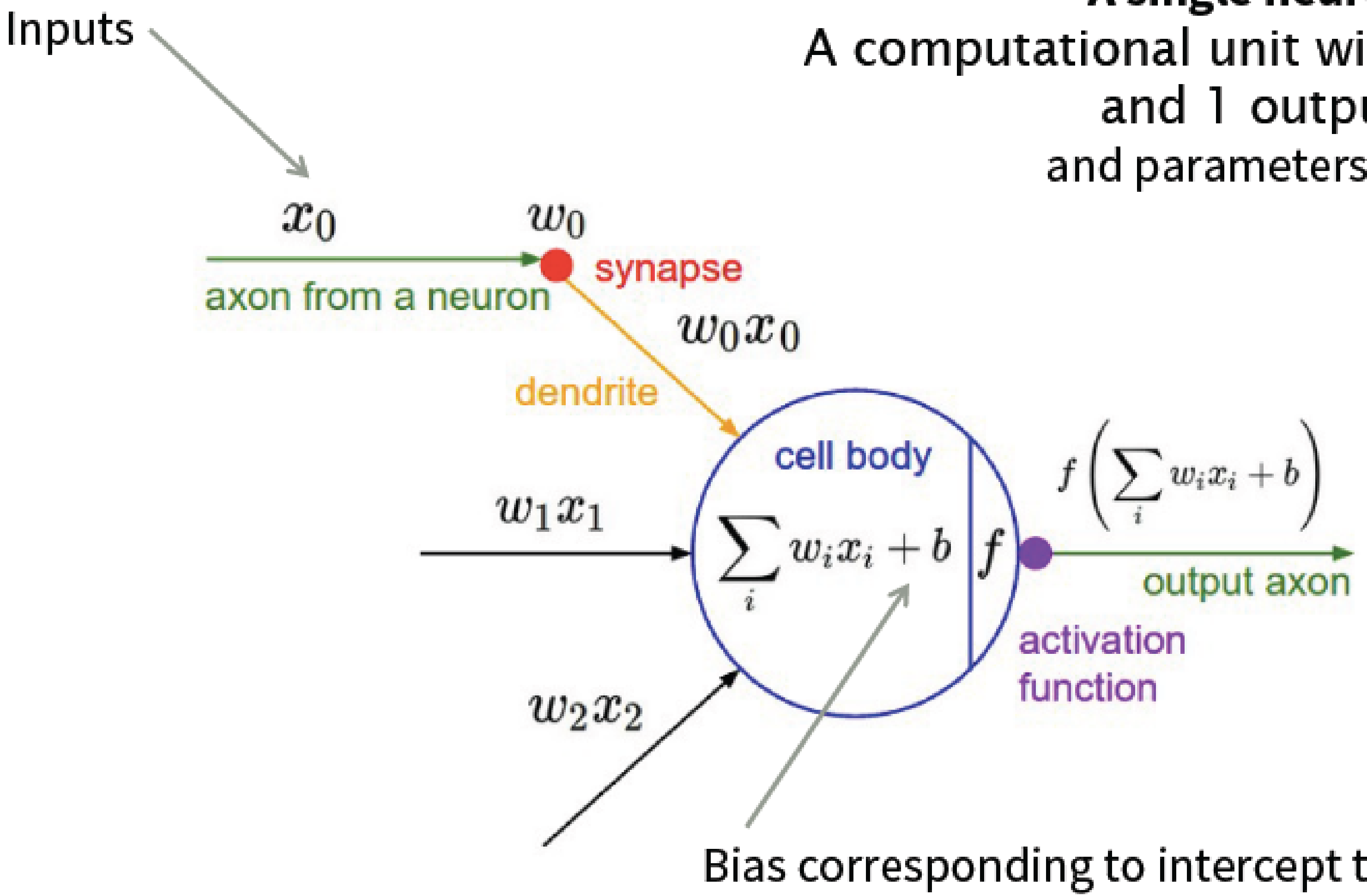


- Axons
- Dendrites
- Synapses
- Receptors

A single neuron

A computational unit with *n* (3) inputs and 1 output

and parameters *w*, *b*



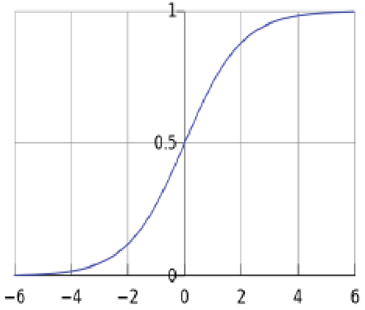
A neural is essentially a logistic regression unit

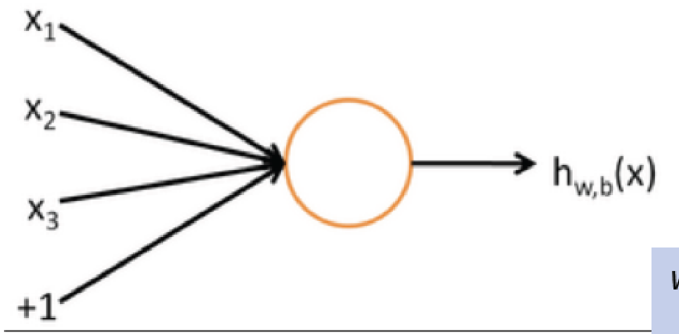
f = nonlinear activation function (e.g., logistic),
w = weights, b = bias, h = hidden, x = inputs

$$h_{w,b}(x) = f(w^{\mathsf{T}} x + b)$$

b: We can have an "always on" feature, which gives a class prior, or separate it out, as a bias term

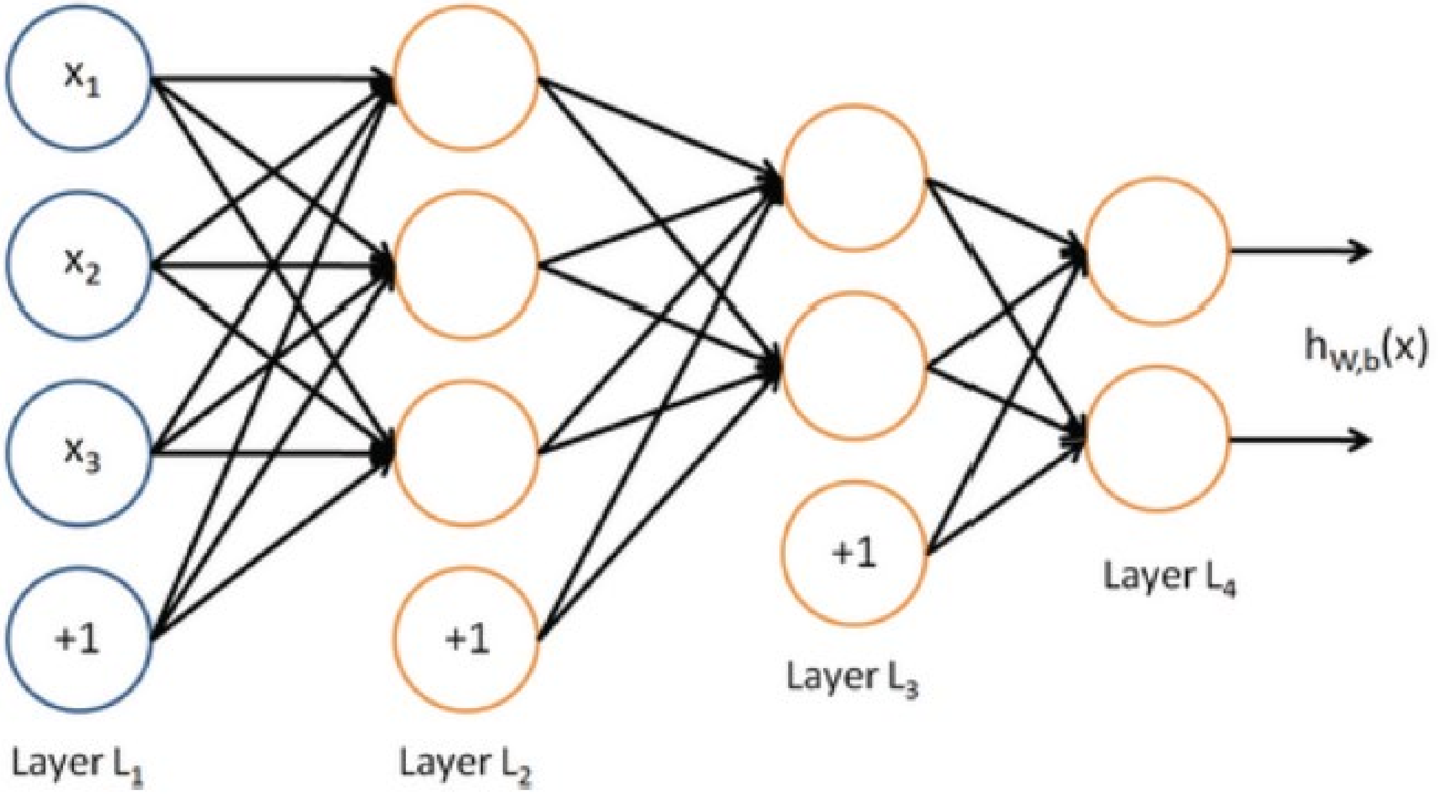
$$f(z) = \frac{1}{1 + e^{-z}}$$



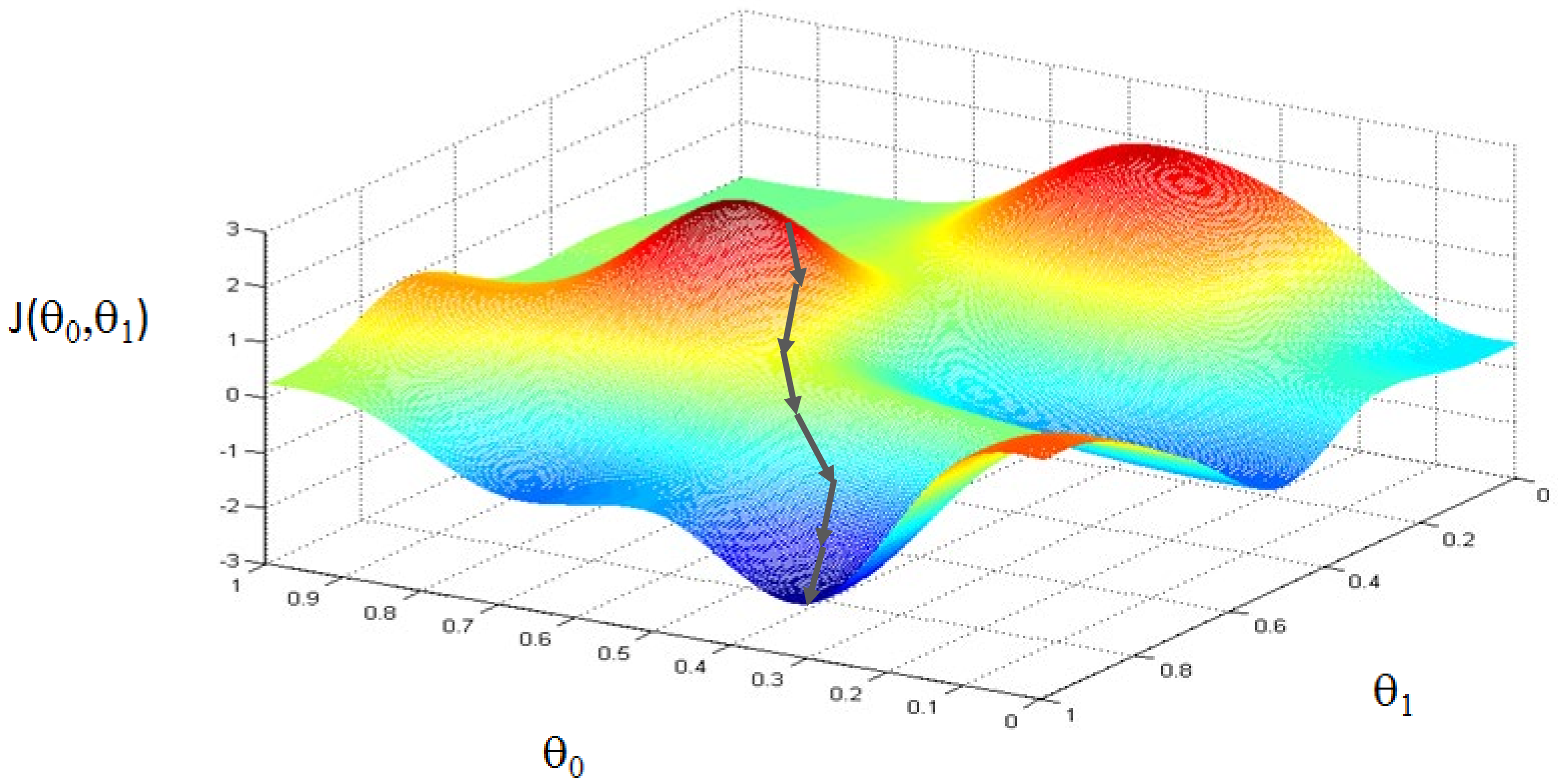


w, *b* are the parameters of this neuron i.e., this logistic regression model

A neural net = running many logistic regressions



Use gradient descent to learn weights and biases



Gradient descent

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$\}$$

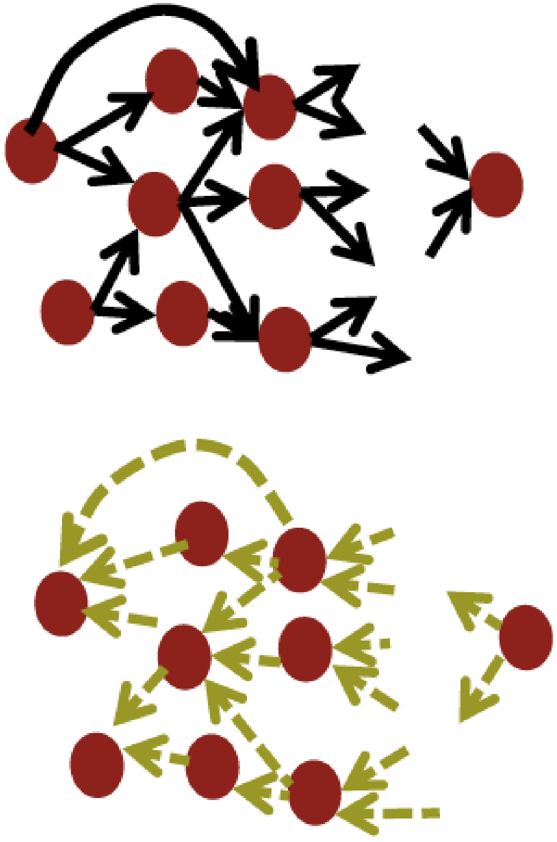
- Calculate the gradient of the loss function w.r.t. parameters
- Determine the learning rate
- Instead of batch gradient descent, we mostly use **stochastic gradient descent (SGD)**, which is much **faster**, easier to escape **local minima** and more stable.

Backpropagation algorithm

Input: Training set $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{m}$

1. Set $\Delta^{(l)} = 0$ for all l
2. **for** $i = 1$ **to** m
3. Set $\mathbf{a}^{(0)} = \mathbf{x}^{(i)}$
4. Perform forward propagation to calculate $\mathbf{a}^{(l)}$ for $l = 1, \cdots, L$
5. Using $y^{(i)}$, compute $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}^{(i)}$
6. Compute $\delta^{(L-1)}, \cdots, \delta^{(1)}$
7. $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l)}(\mathbf{a}^{(l-1)})^T$ for all l
8. $\Delta^{(l)} := \frac{1}{m}\Delta^{(l)}$ for all l
9. $W^{(l)} := W^{(l)} - \eta\Delta^{(l)}$ for all l

Automatic Differentiation



- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you via automatic differentiation

How to make deep learning work?

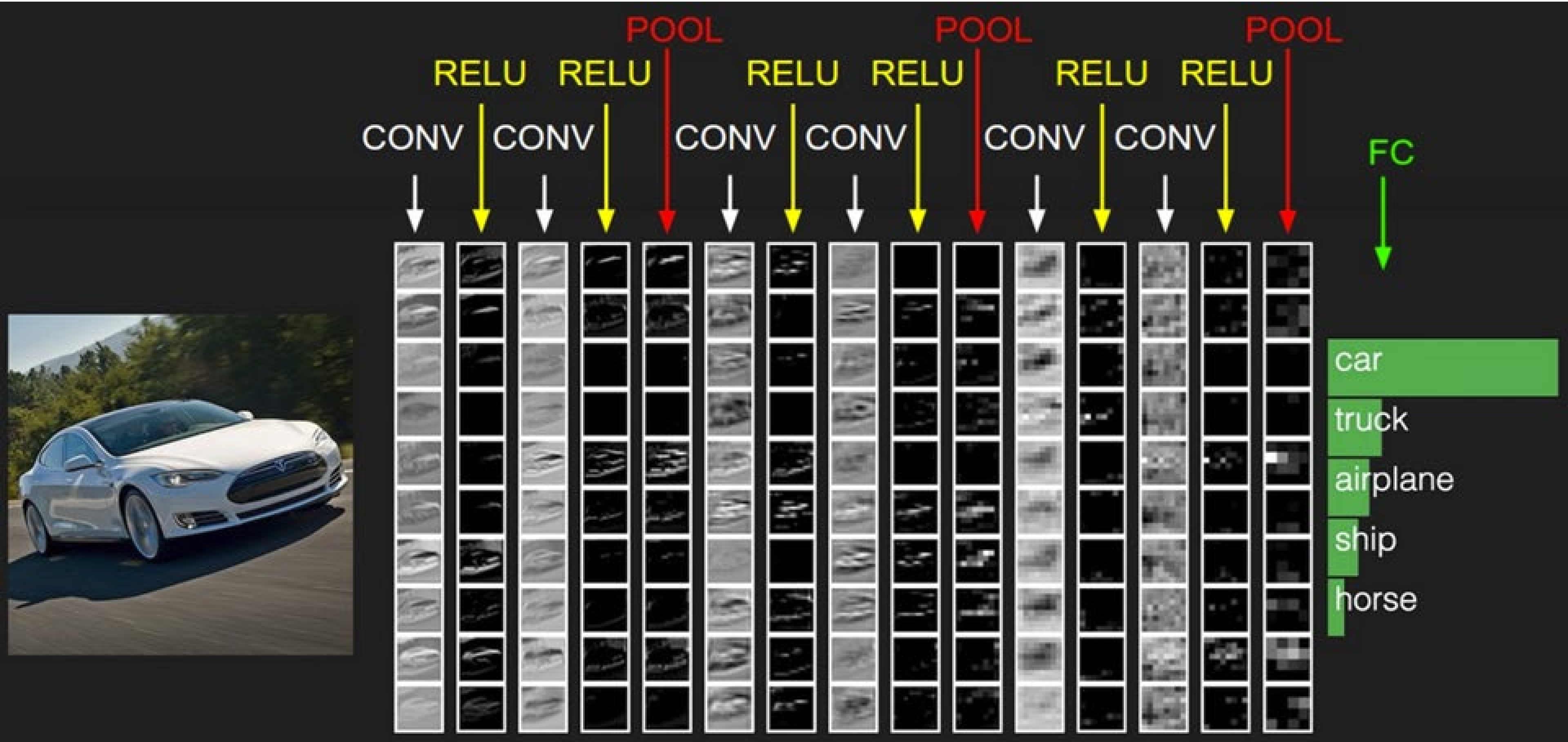
- Carefully designed network **architecture**
- Unsupervised pre-training can be done layer-wise
- Better training algorithms and heuristics such as **good initialization**, **batch normalization**, **weight clip**, **regularization**, etc
- **Drop-out**, **Early Stop** to control complexity, reduce overfitting
- **Transfer learning**, **meta learning**, etc

.

Structured Neural Models Underlying DL Revolution

- **Convolutional models**
- **Recurrent models**
- **Gated and residual connections**
- **Attention**

Convolutional Neural Nets

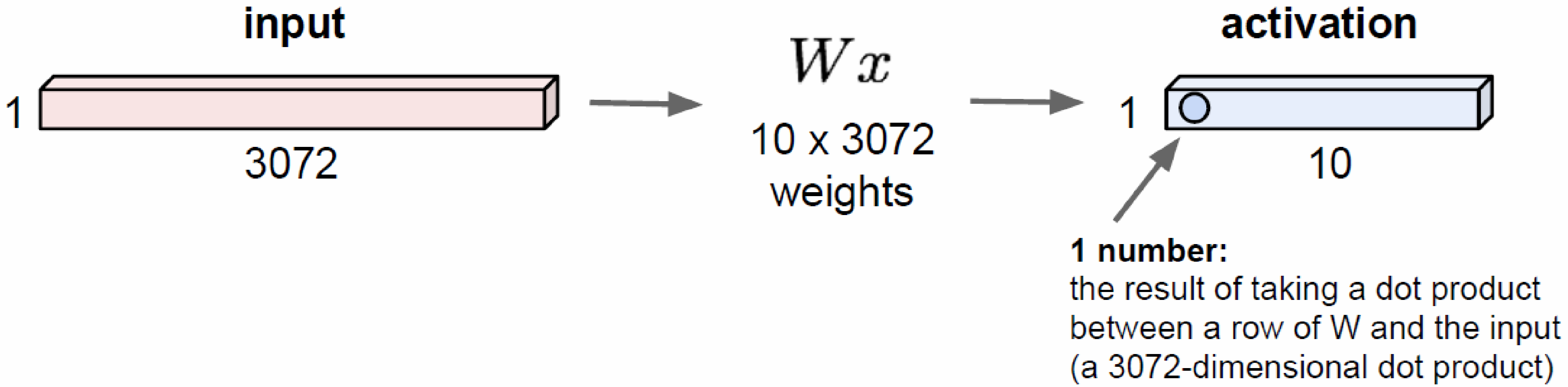


Vision: Convolutional Models

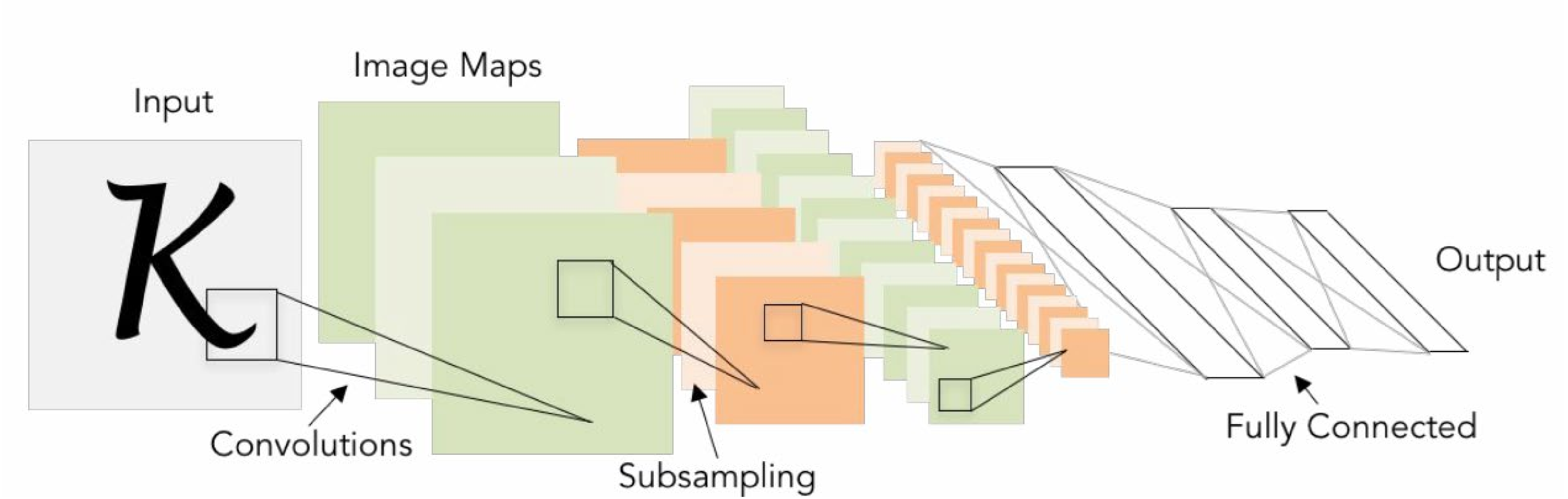
- For computer vision, a key property that we usually wish to capture is translation invariance
- We would like to have visual “feature detectors” that find something in an image regardless of precisely where it is located
- We do this with a convolutional layer

Fully Connected Layer

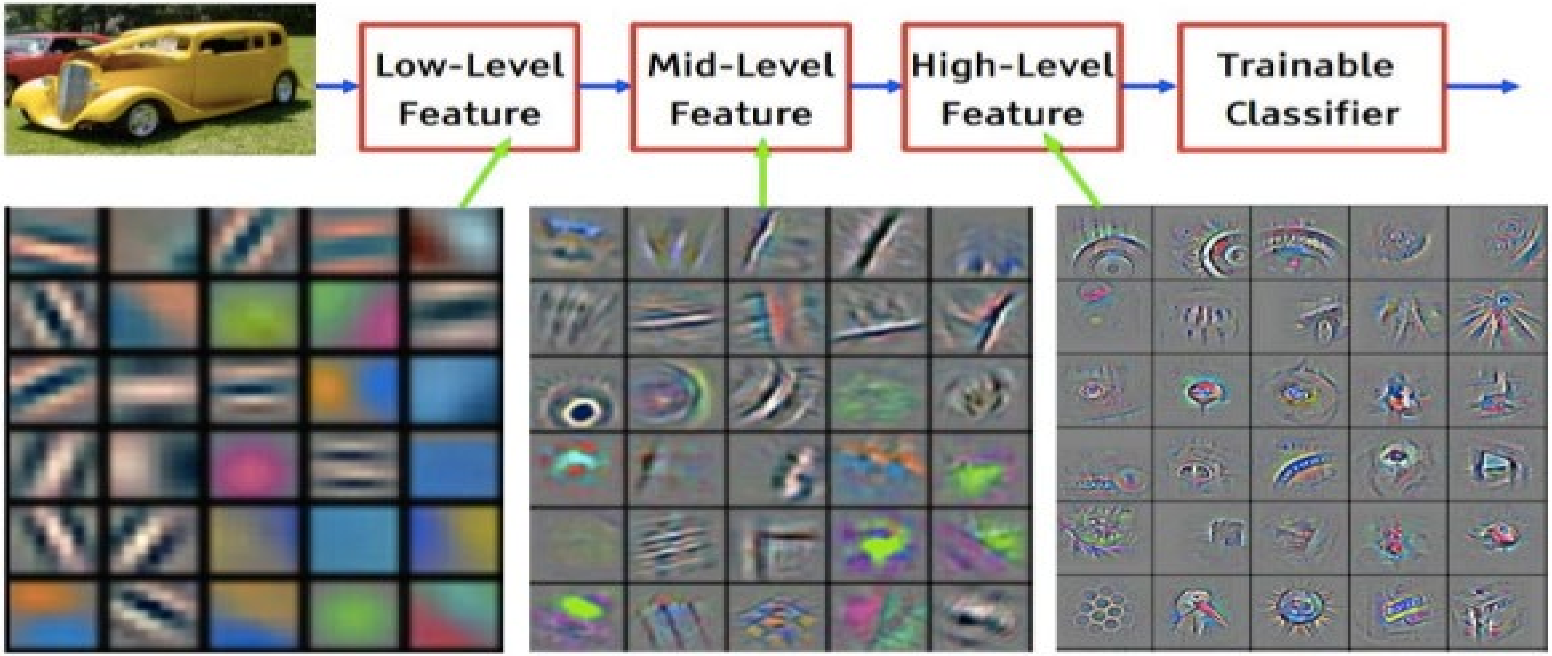
32x32x3 image -> stretch to 3072 x 1



Convolutional Neural Net



Features learned by CNN

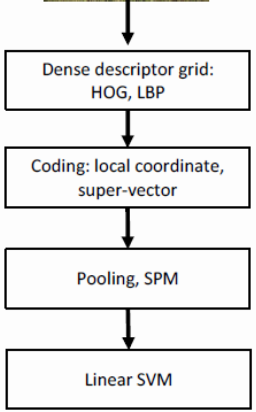


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

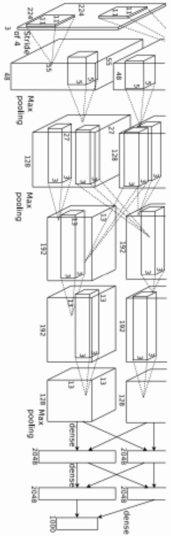
NEC-UIUC



[Lin CVPR 2011]

Year 2012

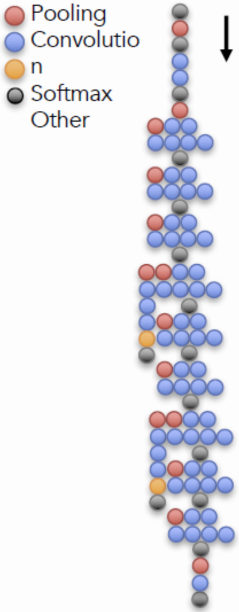
SuperVision



[Krizhevsky NIPS 2012]

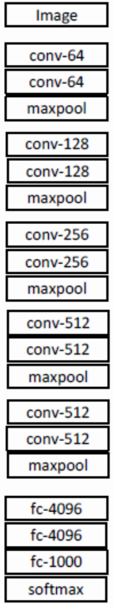
Year 2014

GoogLeNet



[Szegedy arxiv 2014]

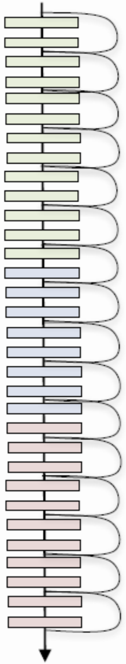
VGG



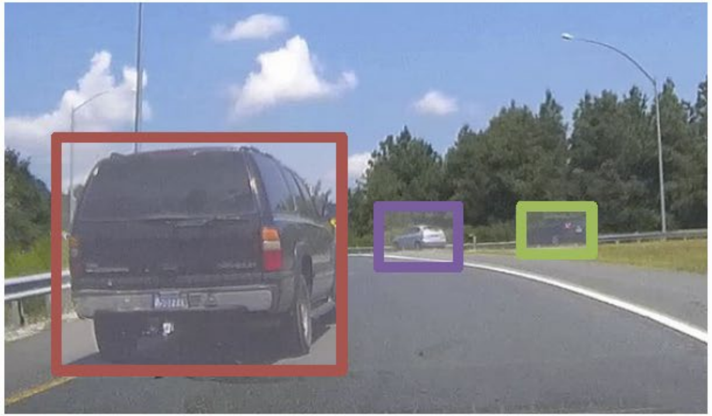
[Simonyan arxiv 2014]

Year 2015

MSRA



[He ICCV 2015]

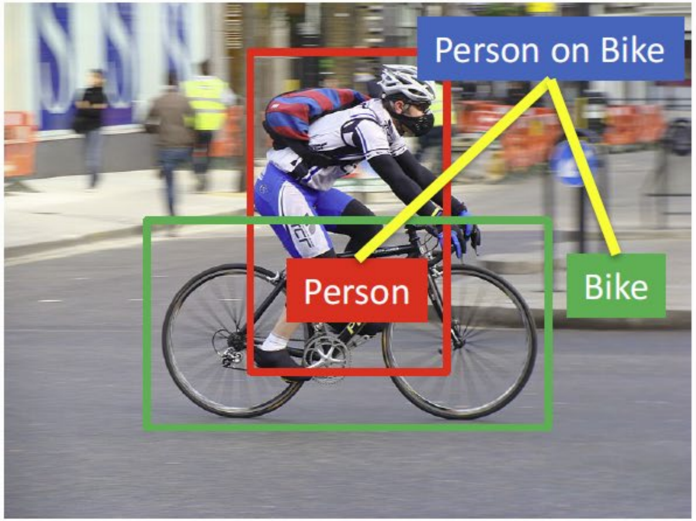


This image is licensed under CC BY-NC-SA 2.0; changes made

- Object detection
- Action classification
- Image captioning
- ...

Person

Hammer

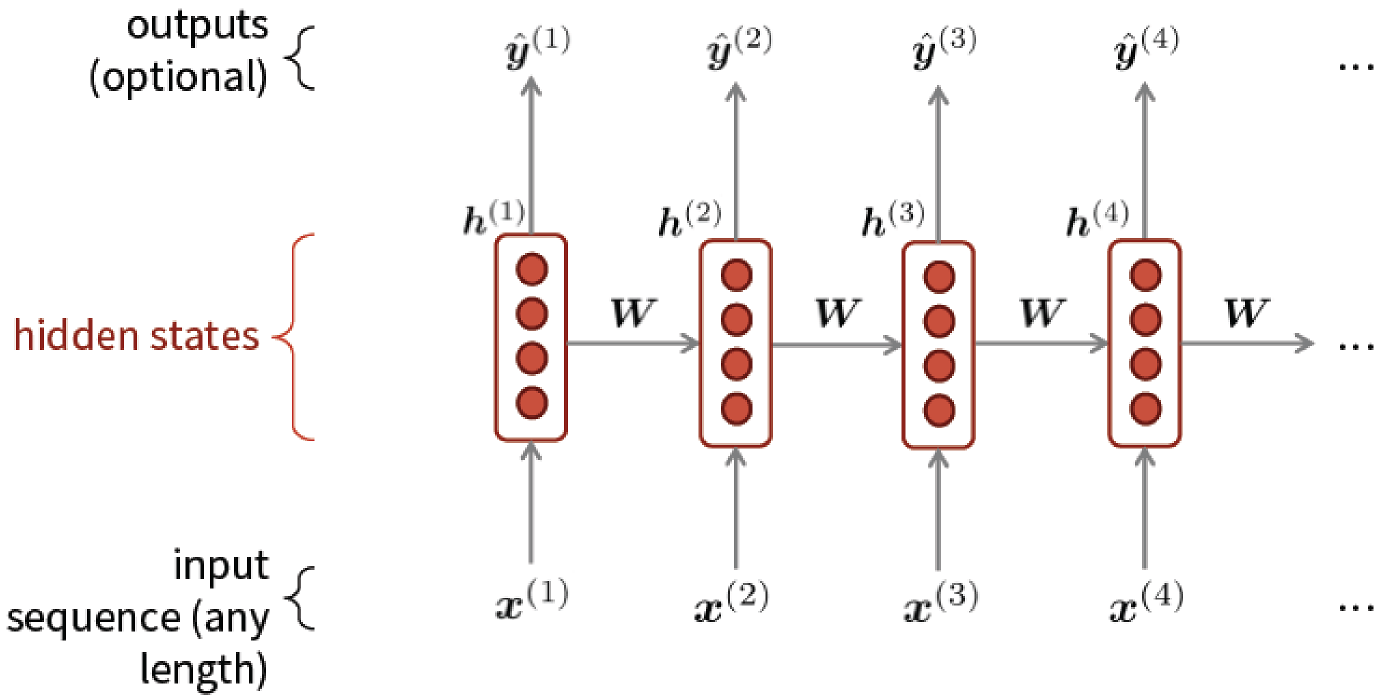


Language: Recurrent Models

- Until now, we've dealt with classifying/generating fixed-size objects.
  - We just resized images to our procrustean bed!
- How can we deal with variable-size inputs, such as the word sequences in human language text or bioinformatic gene sequences?
- We do this with a recurrent layer

Recurrent Neural Nets (RNN)

<u>Core idea:</u> Apply the same weights $\boldsymbol{W}$ *repeatedly*



RNN for Language Modeling

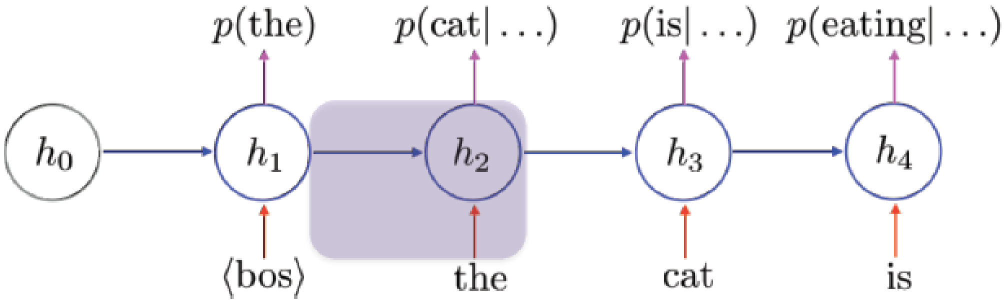
Transition Function $h_t = f(h_{t-1}, x_t)$

Inputs

i. Current word $x_t \in \{1, 2, \ldots, |V|\}$
ii. Previous state $h_{t-1} \in \mathbb{R}^d$

Parameters

i. Input weight matrix $W \in \mathbb{R}^{|V| \times d}$
ii. Transition weight matrix $U \in \mathbb{R}^{d \times d}$
iii. Bias vector $b \in \mathbb{R}^d$



Building a language model

Transition Function $h_t = f(h_{t-1}, x_t)$

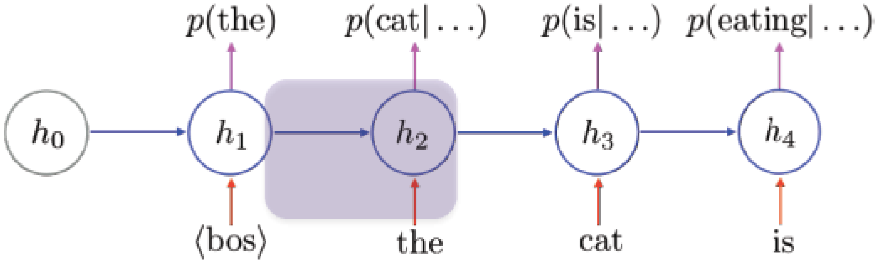
Naïve Transition Function

$$f(h_{t-1}, x_t) = \tanh(W[x_t] + Uh_{t-1} + b)$$

Element-wise nonlinear transformation

Trainable word vector

Linear transformation of previous state



Building a recurrent language model

Prediction Function $p(x_{t+1} = w | x_{\leq t}) = g_w(h_t)$

Inputs

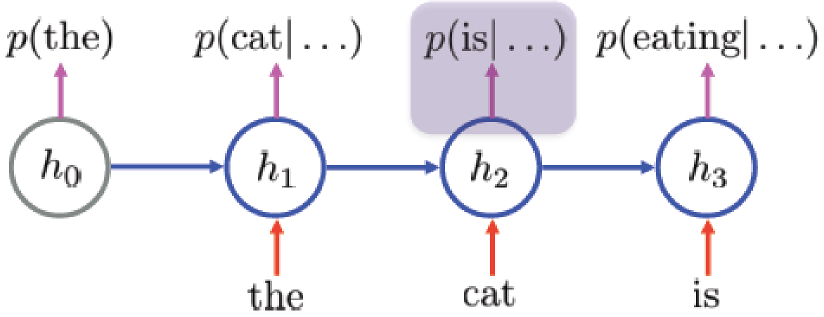
i. Current state $h_t \in \mathbb{R}^d$

Parameters

i. Softmax matrix $R \in \mathbb{R}^{|V| \times d}$

ii. Bias vector $c \in \mathbb{R}^{|V|}$

Softmax gives a probability distribution over next words. To generate text, we take the work with max prob., and use it as the input at the next time step



Some observations

- **Non-convexity** is no longer an issue. In fact, non-convex models seem to more powerful than convex models.

- We used to worry about bad **local minima**, but it turns out that bad local minima typically don’t exist and any different minima seem to be roughly equivalently good. No need to worry!

- Extremely deep models using residual connections (ResNets) have powered the progress in computer vision.   Need to rethink layered representations?

Some observations

- **Embedding**

Use of **distributed representations of categorical values** like words allows very effective modeling and sharing of dimensions of similarity

Some observations

Rethink overfitting

- Models with orders of magnitude more parameters than there are training data available, if trained with ample amounts of regularization (including new forms like dropout), will outperform simpler models – generalizing better to new data.

- **Practitioner's recipe**: build a sufficiently high-capacity model that it can be trained to 0% training error, and then increase its regularization until it doesn't overfit on dev.

Some observations

Architecture matters

- Identify the best architecture for your problem. RNN, CNN, GRU, LSTM ...

- Put in mechanisms to facilitate learning: Dropout, Batch normalization, Attention ...