CS7GV1: Computer Vision

Neural networks

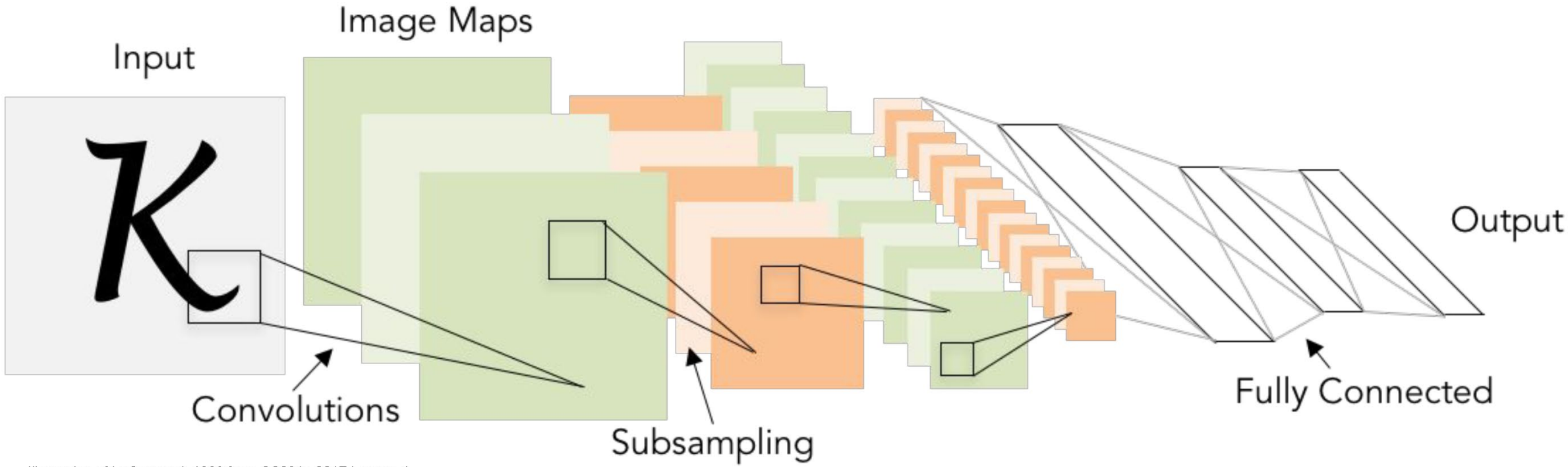


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Credits: Some Slides from Noah Snavely (Cornell)
and Fei-Fei Li, Justin Johnson, Serena Yeung (Stanford)
http://vision.stanford.edu/teaching/cs231n/

Readings

- Neural networks
  - http://cs231n.github.io/neural-networks-1/
  - http://cs231n.github.io/neural-networks-2/
  - http://cs231n.github.io/neural-networks-3/
  - http://cs231n.github.io/neural-networks-case-study/

- Convolutional neural networks
  - http://cs231n.github.io/convolutional-networks/

Image Classification: a core task in computer vision

- Assume given set of discrete labels, e.g.

  {cat, dog, cow, apple, tomato, truck, … }

f() = “apple”

f() = “tomato”

f() = “cow”

Dataset: ETH-80, by B. Leibe Slide credit: L. Lazebnik

Recap: linear classification

- Have score function and loss function
  - Score function maps an input data instance (e.g., an image) to a vector of scores, one for each category
  - Last time, our score function is based on linear classifier

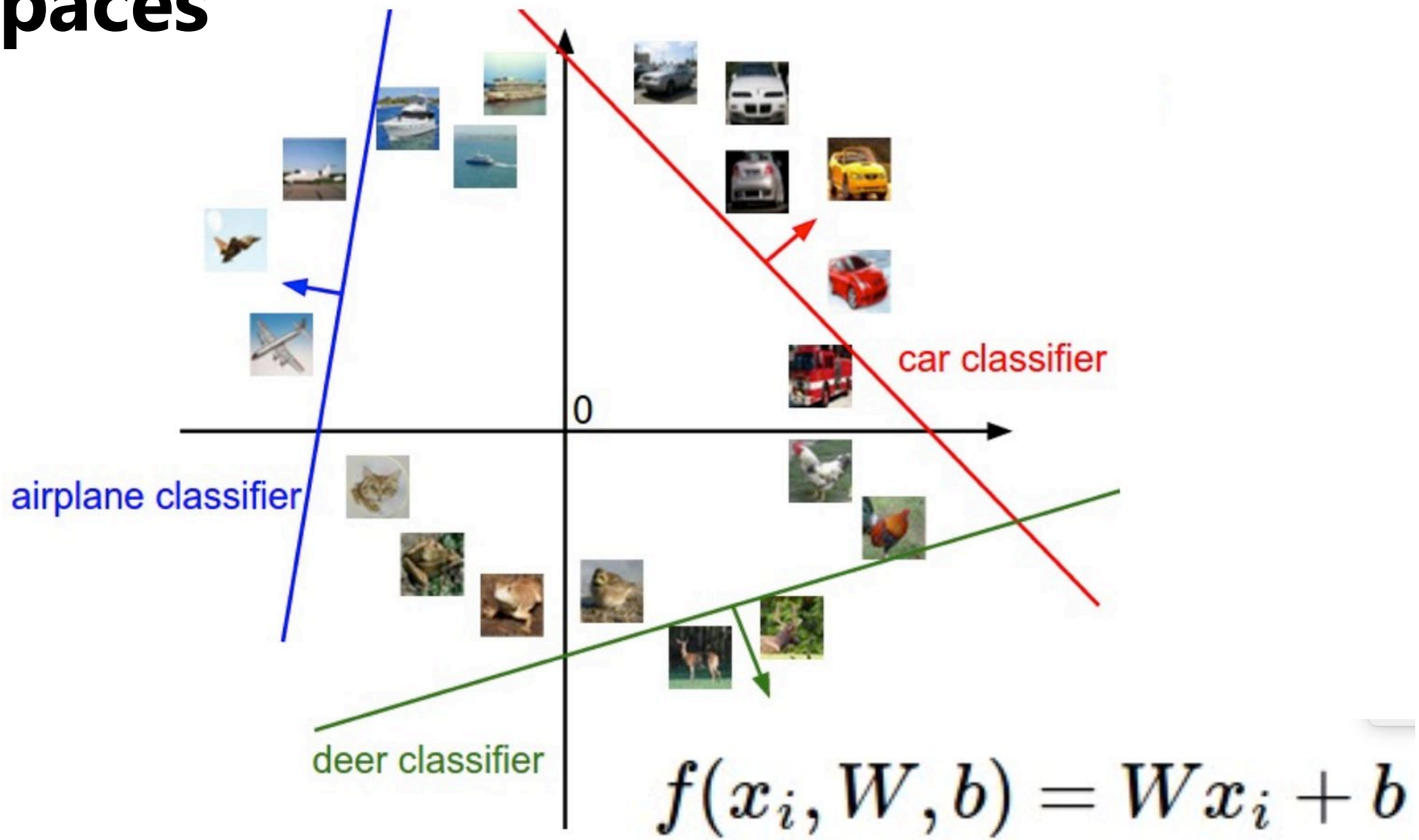
$$f(x,W) = Wx + b$$

f: score function
x: input instance
W, b: parameters of a linear (actually affine) function

- Find **W** and **b** to minimize a *loss*, e.g. cross-entropy loss

$$L = \frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Linear classifiers separate features space into half-spaces



$$f(x_i, W, b) = Wx_i + b$$

Neural networks

(**Before**) Linear score function: $f = Wx$

Neural networks

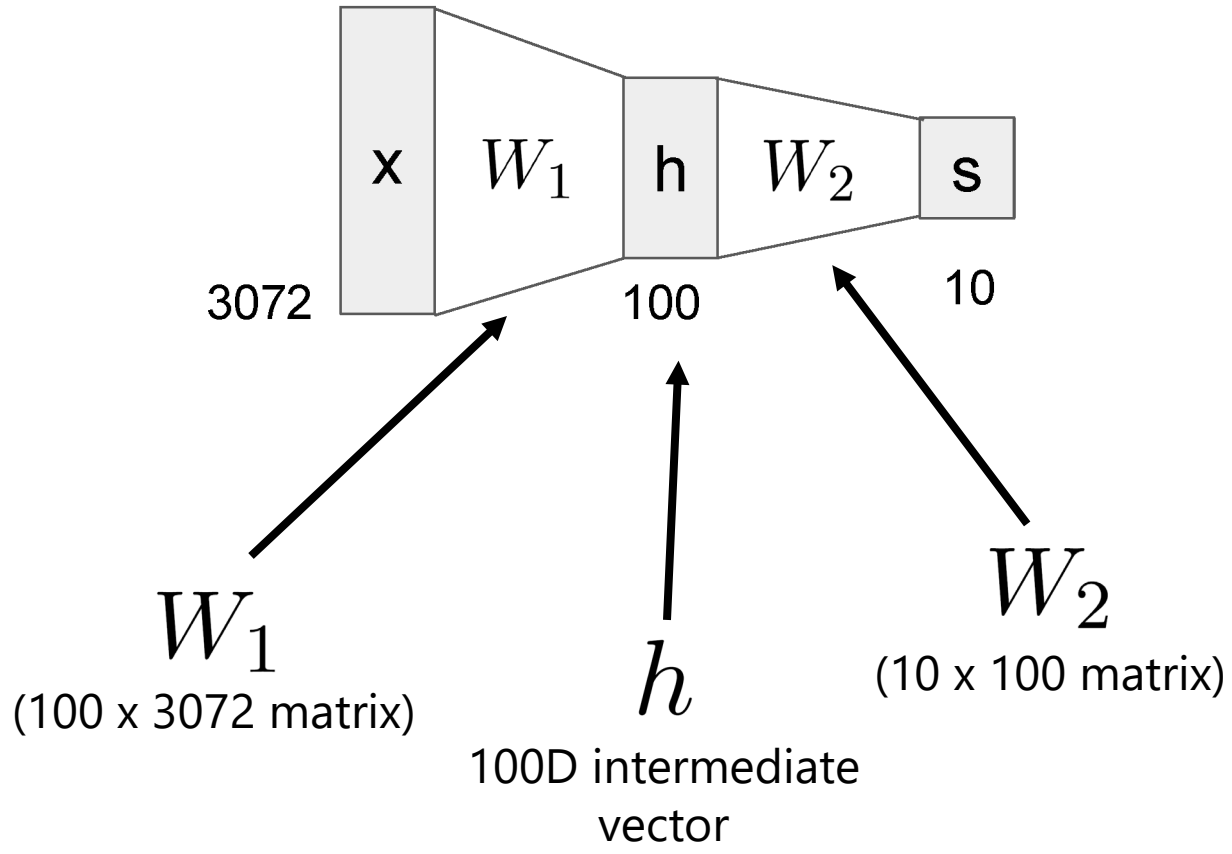
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

Neural networks

(**Before**) Linear score function: $f = Wx$

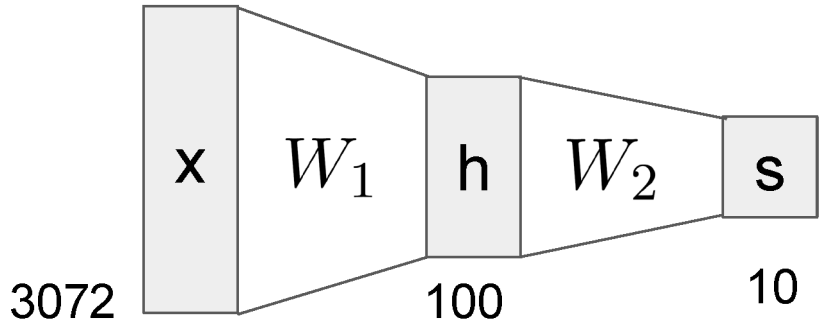
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



- Total number of weights to learn:

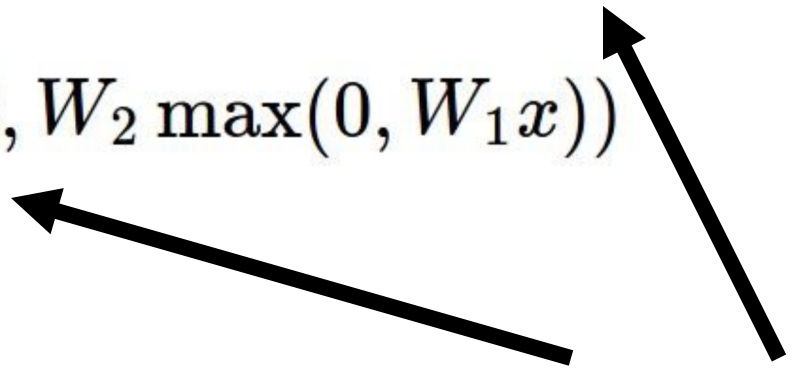
  3,072 x 100 + 100 x 10 = 308,200

Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$



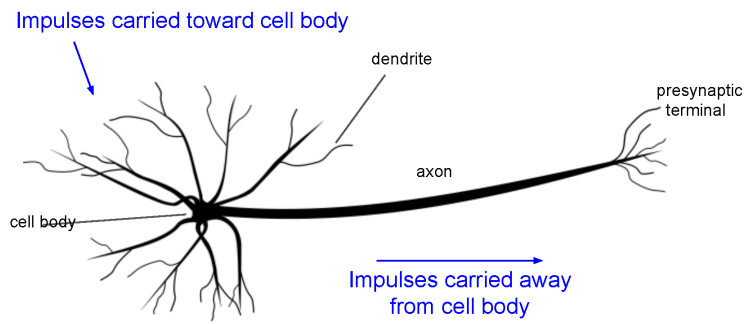
also called “Multi-Layer Perceptrons” (MLPs)

Neural networks

- Very coarse generalization of neural networks:
  - Linear functions chained together and separated by non-linearities (*activation functions*), e.g. "max"

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

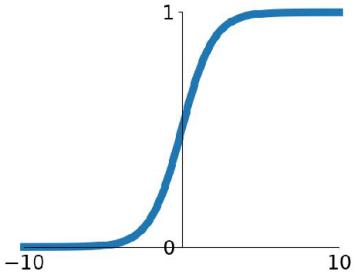
  - Why separate linear functions with non-linear functions?
  - *Very roughly* inspired by real neurons



Activation functions

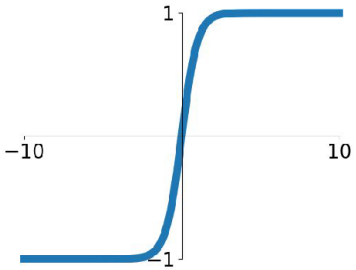
Sigmoid

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



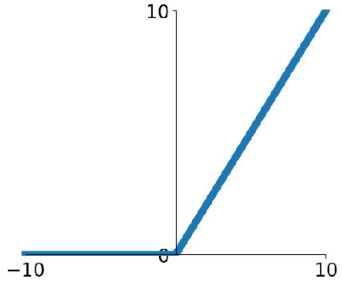
tanh

$$\tanh(x)$$



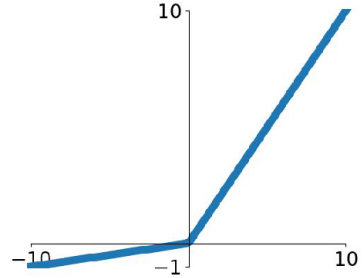
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

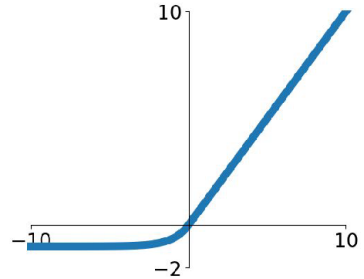


Maxout

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

ELU

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

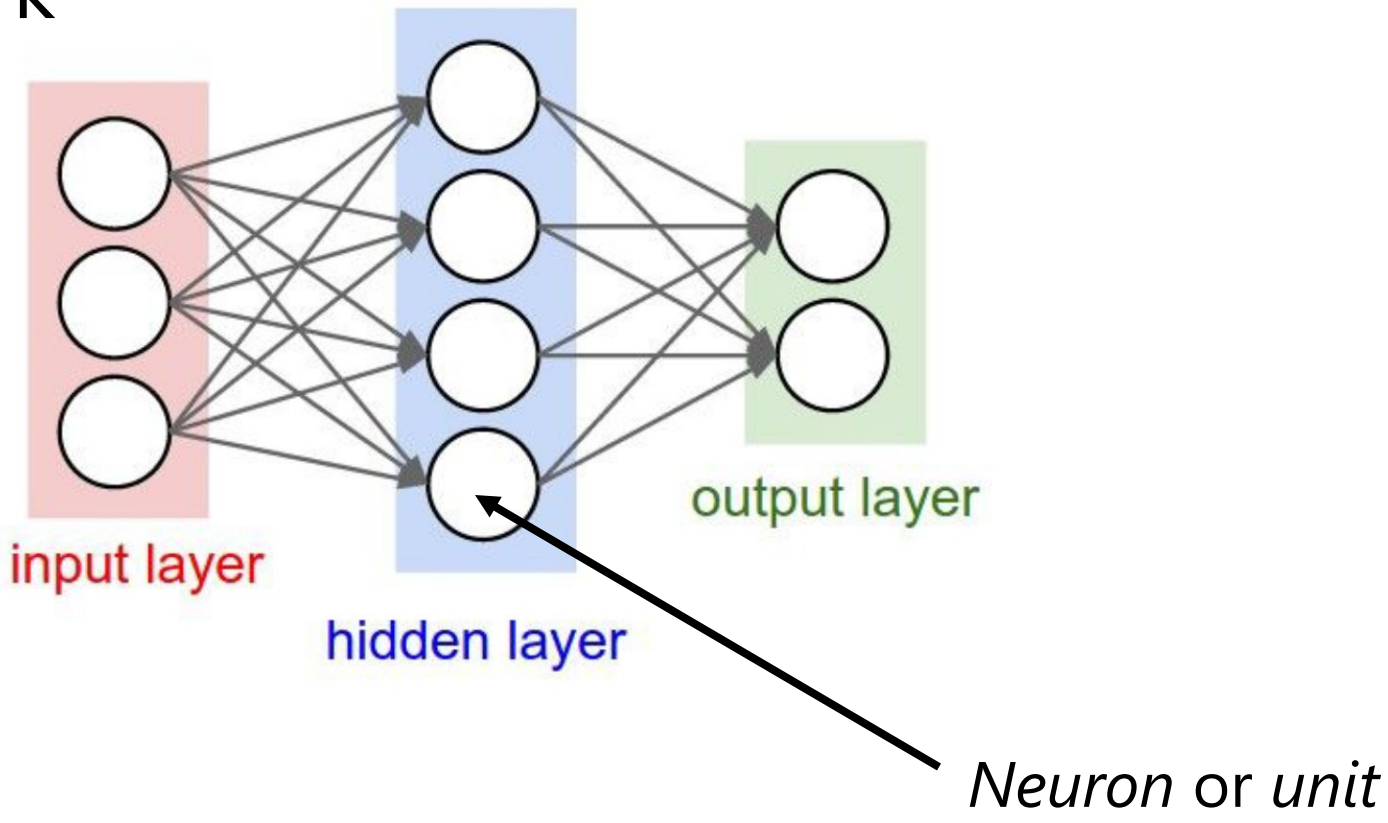


ReLU = rectified linear activation function;
Leaky ReLU = allows small negative values when input <0

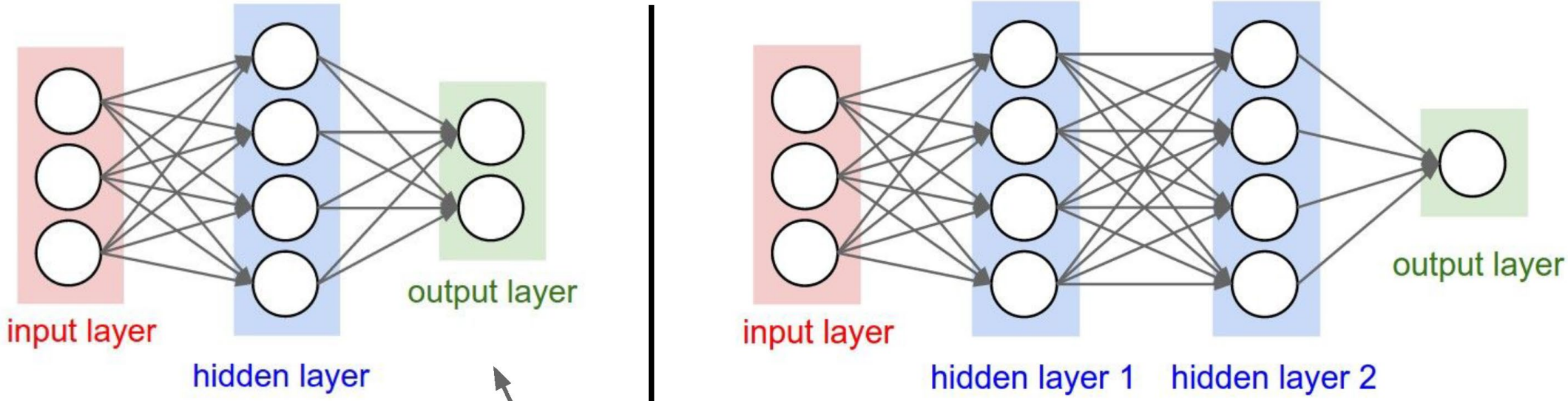
Maxout = max of several linear functions
ELU = Exponential Linear Unit

Neural network architecture

- Computation graph for a 2-layer neural network



Neural networks: Architectures



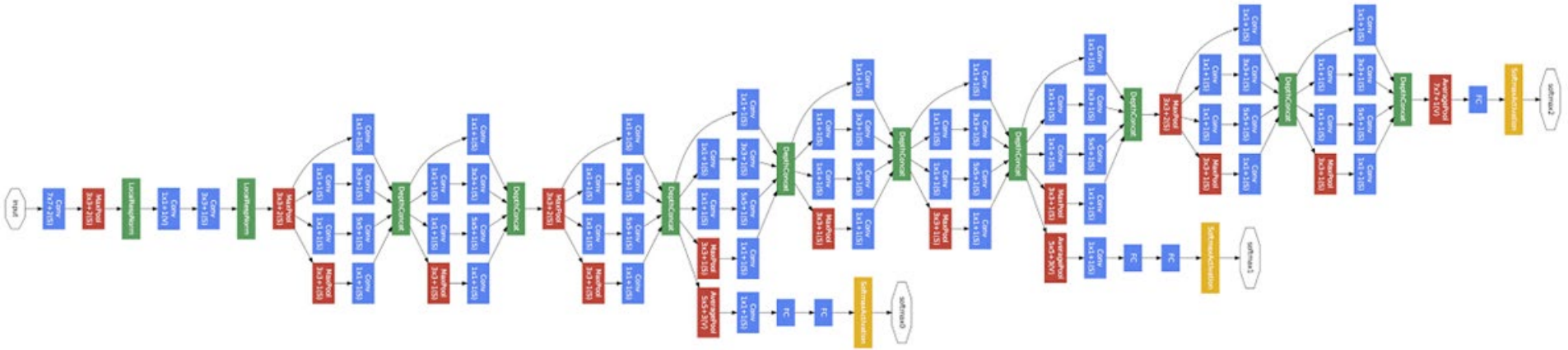
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

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

"Fully-connected" layers

- **Deep** networks typically have many layers and potentially millions of parameters

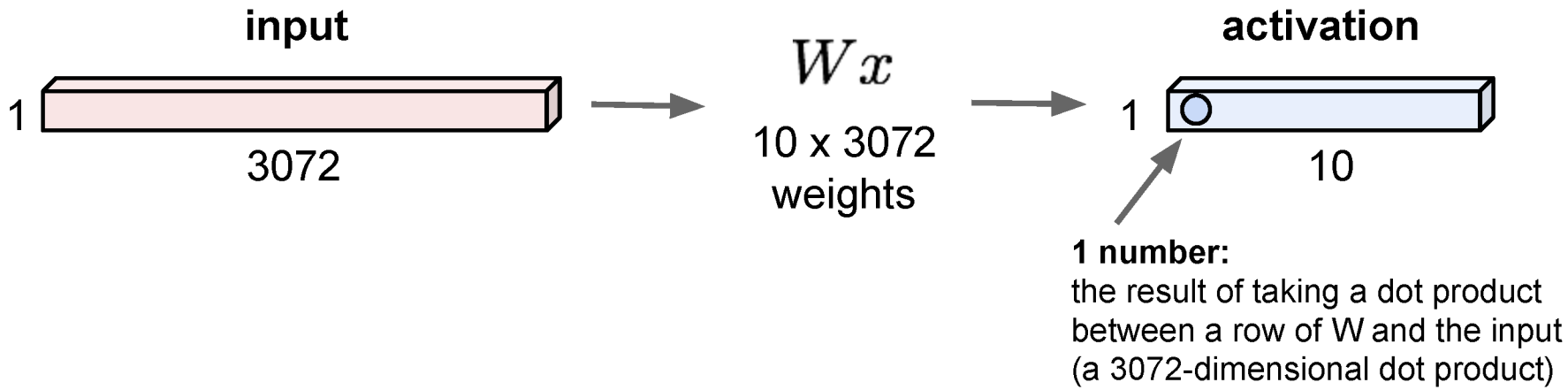
Deep neural network



- *Inception* network (Szegedy et al, 2015)
- 22 layers

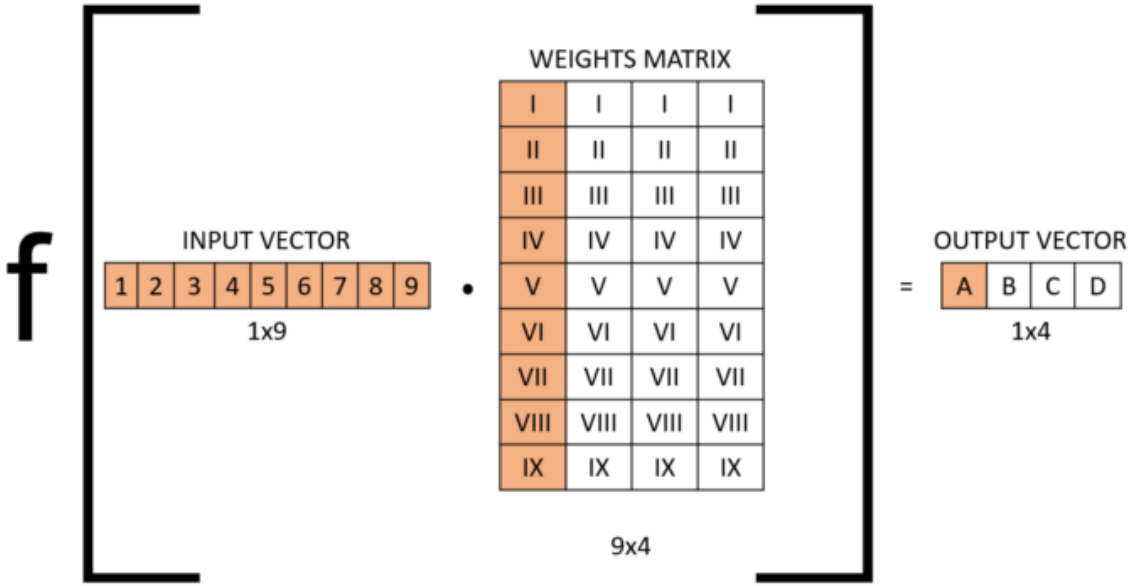
Fully Connected Layer

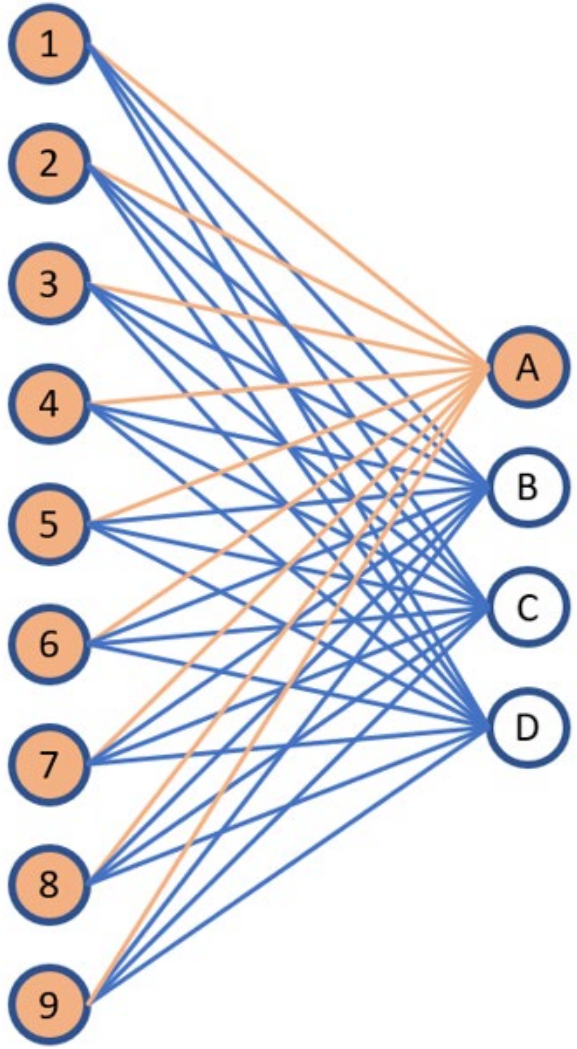
32x32x3 image -> stretch to 3072 x 1



- Just like a linear classifier – but in this case, just one layer of a larger *network*

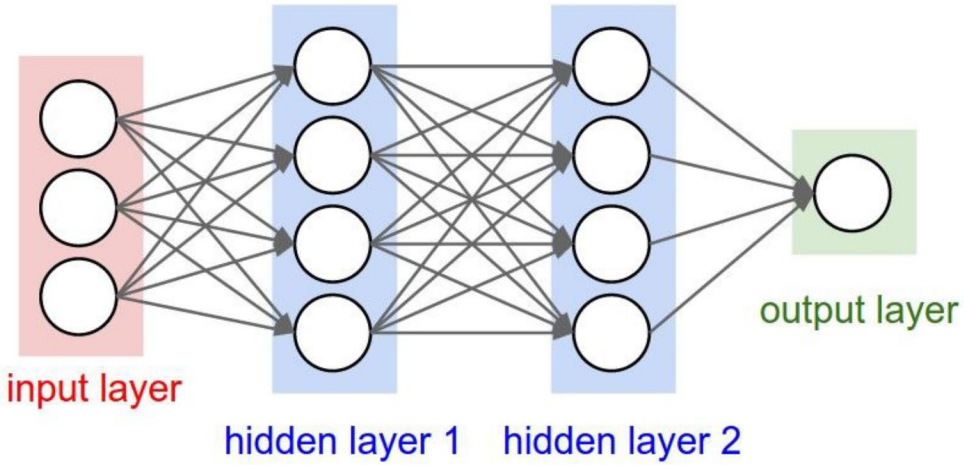
Why Fully Connected?





https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b

Example feed-forward computation of a neural network



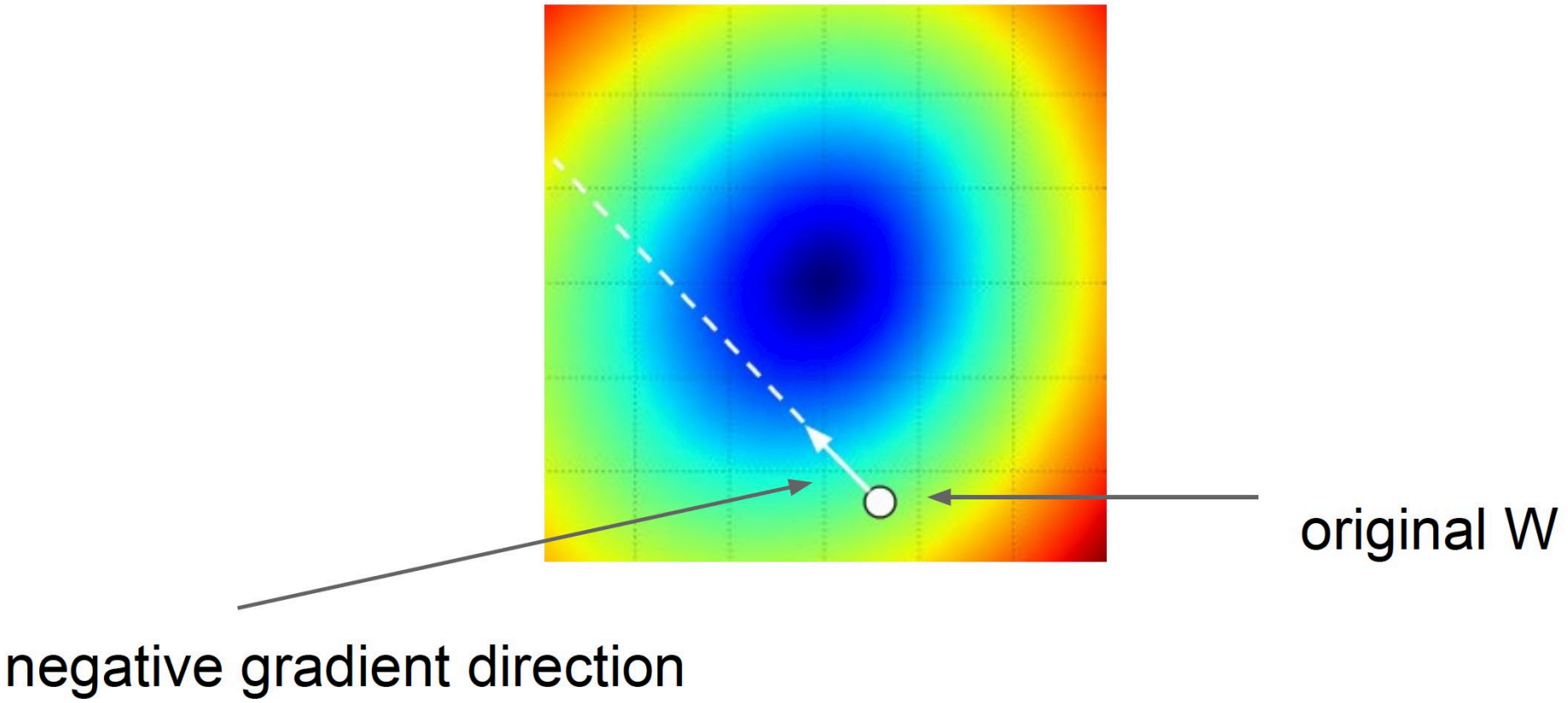

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Summary

- We arrange neurons into fully-connected layers
- The abstraction of a **layer** has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- Neural networks are not really *neural*

Optimizing parameters with gradient descent

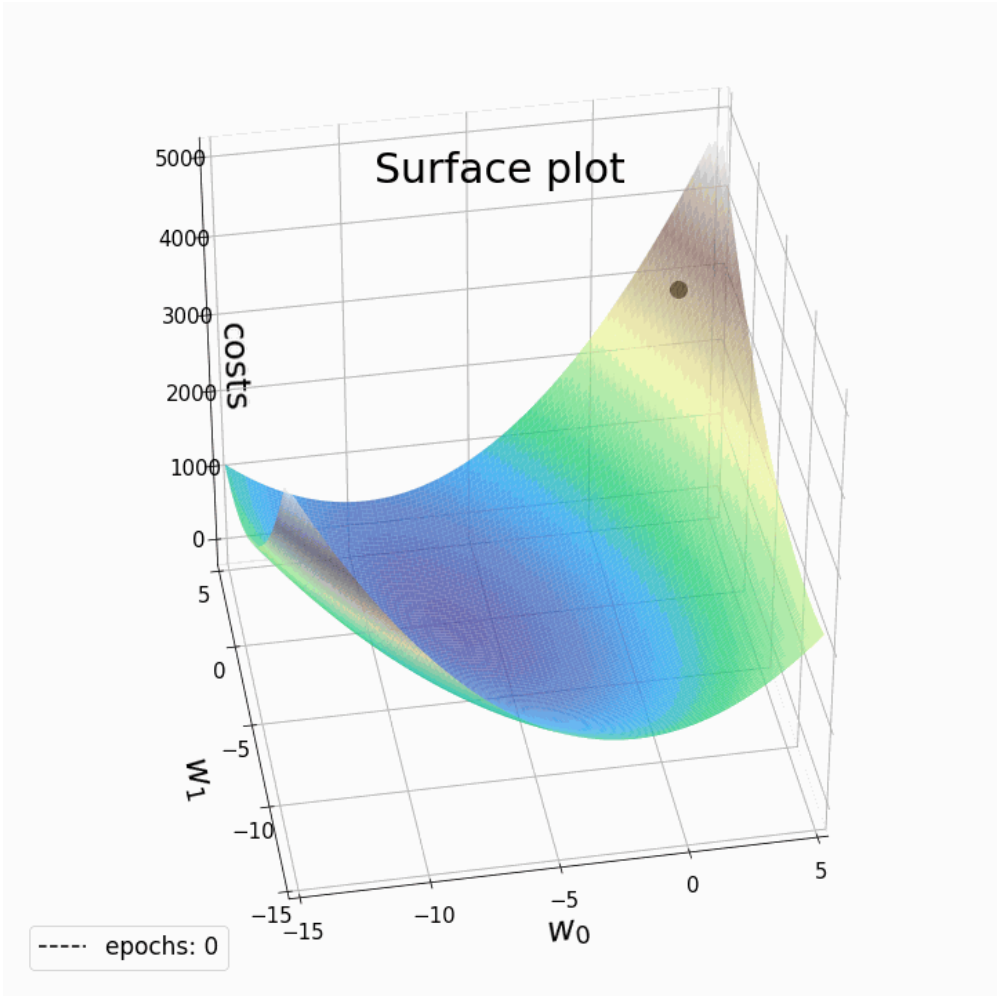
- How do we find the best **W** and **b** parameters?
- In general: *gradient descent*
  1. Start with a guess of a good **W** and **b** (or randomly initialize them)
  2. Compute the loss function for this initial guess and the *gradient* of the loss function
  3. Step some distance in the negative gradient direction (direction of steepest descent)
  4. Repeat steps 2 & 3
- Note: efficiently performing step 2 for deep networks is called *backpropagation*



Gradient descent: walk in the direction opposite gradient

- **Q**: How far?
- **A**: Step size: *learning rate*
- Too big: will miss the minimum
- Too small: slow convergence

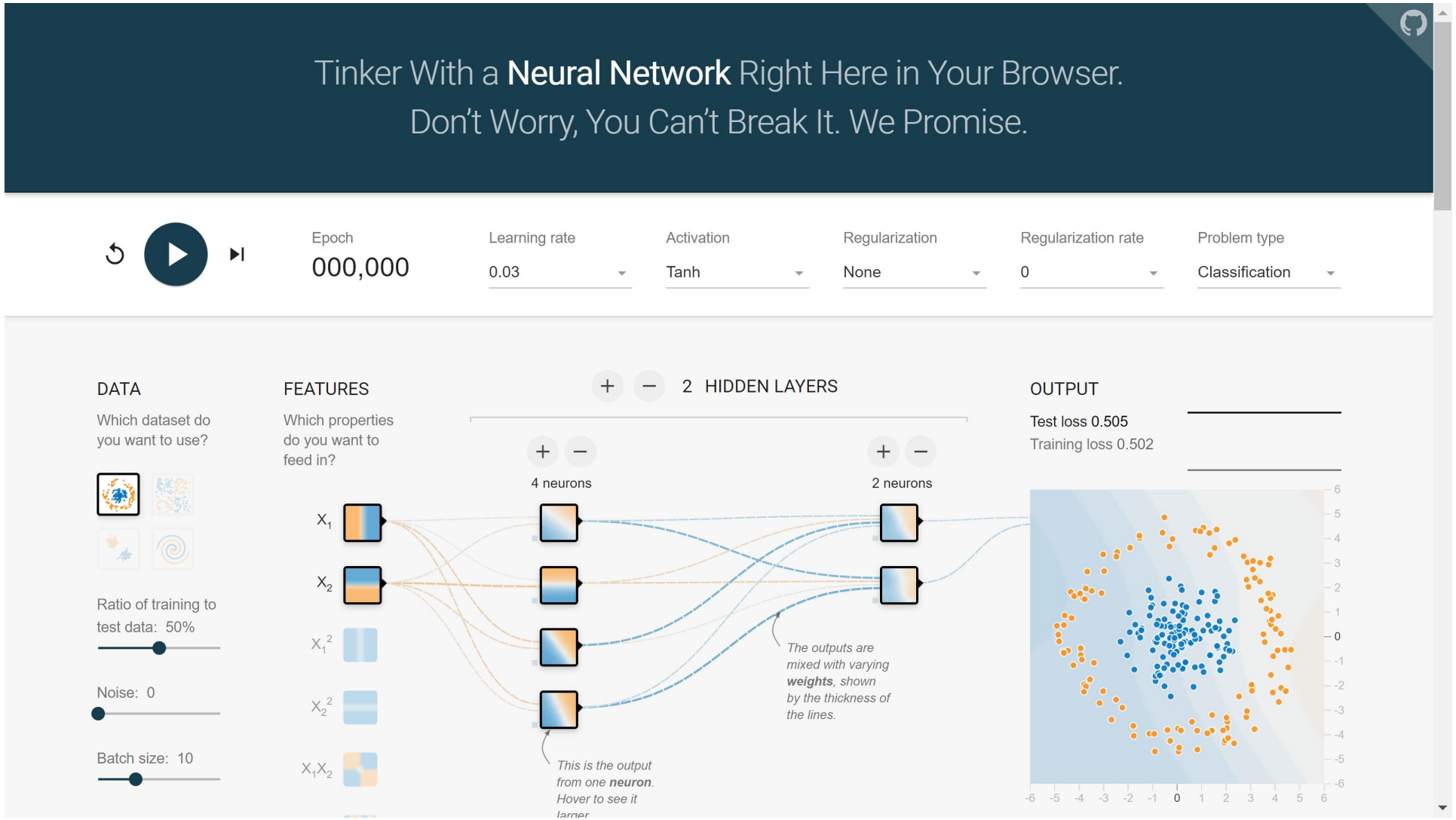
2D example of gradient descent



- In reality, in deep learning we are optimizing a highly complex loss function with millions of variables (or more)

https://laptrinhx.com/gradient-descent-animation-2-multiple-linear-regression-3070246823/

2D example: TensorFlow Playground



https://playground.tensorflow.org

Convolutional neural networks

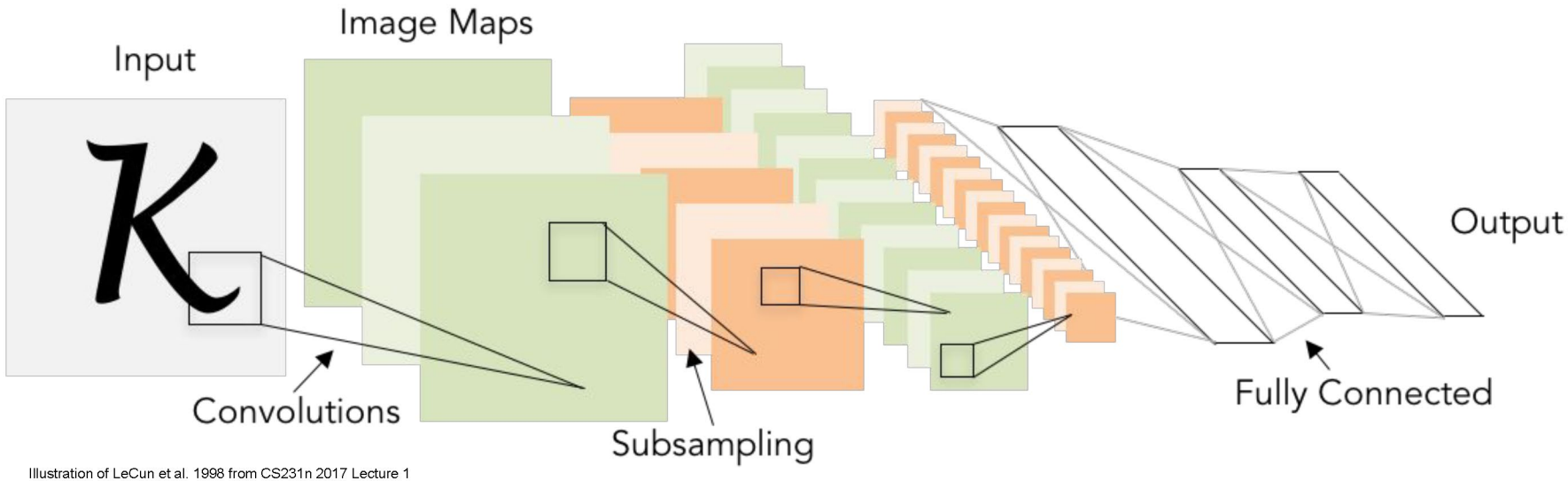


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

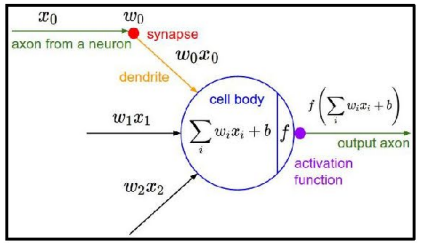
A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



update rule:

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

Frank Rosenblatt, ~1957: Perceptron

A bit of history…

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

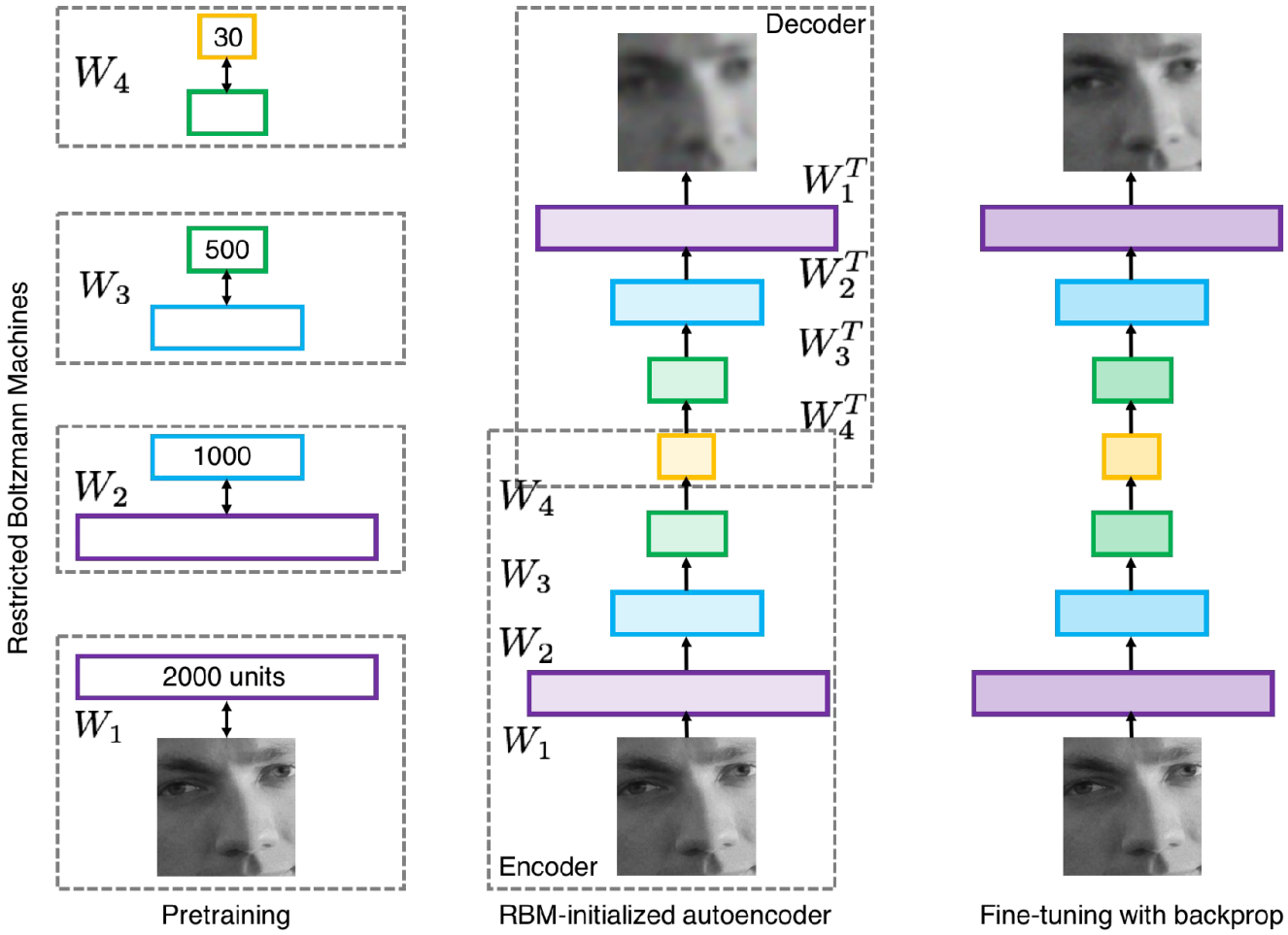


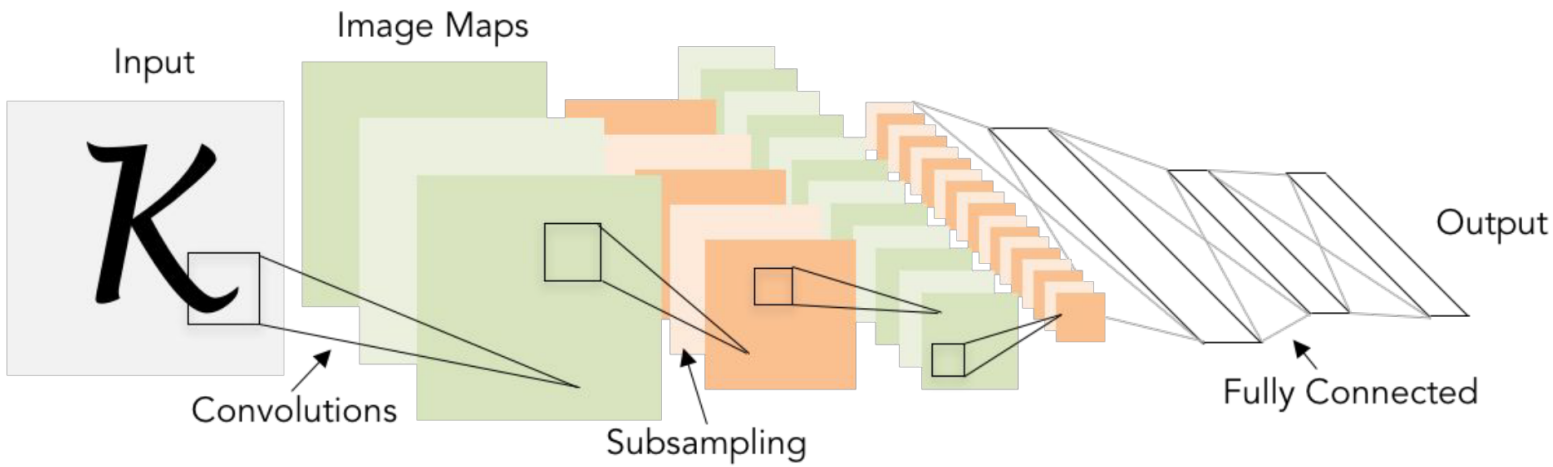
Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Hinton and Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 2016.

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

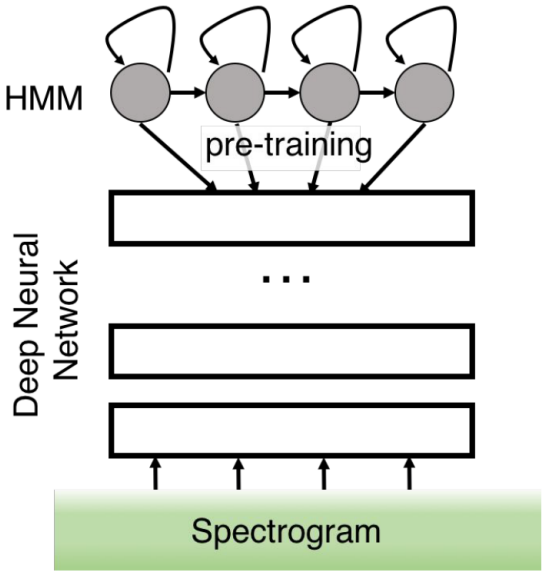
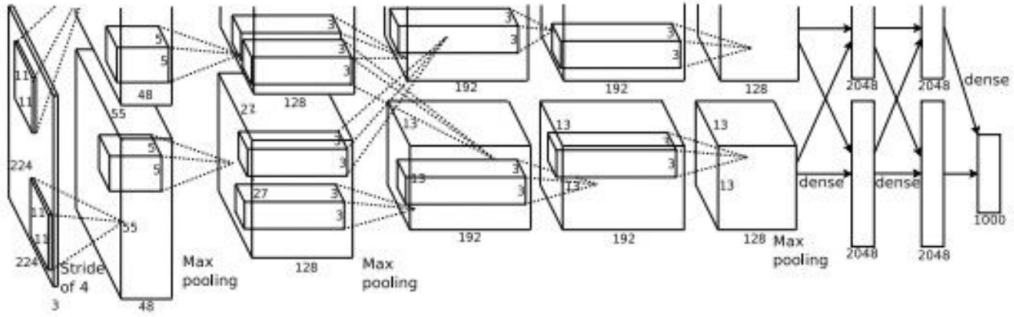
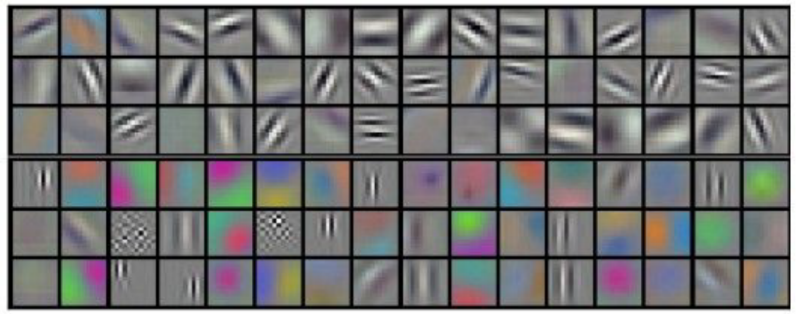


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

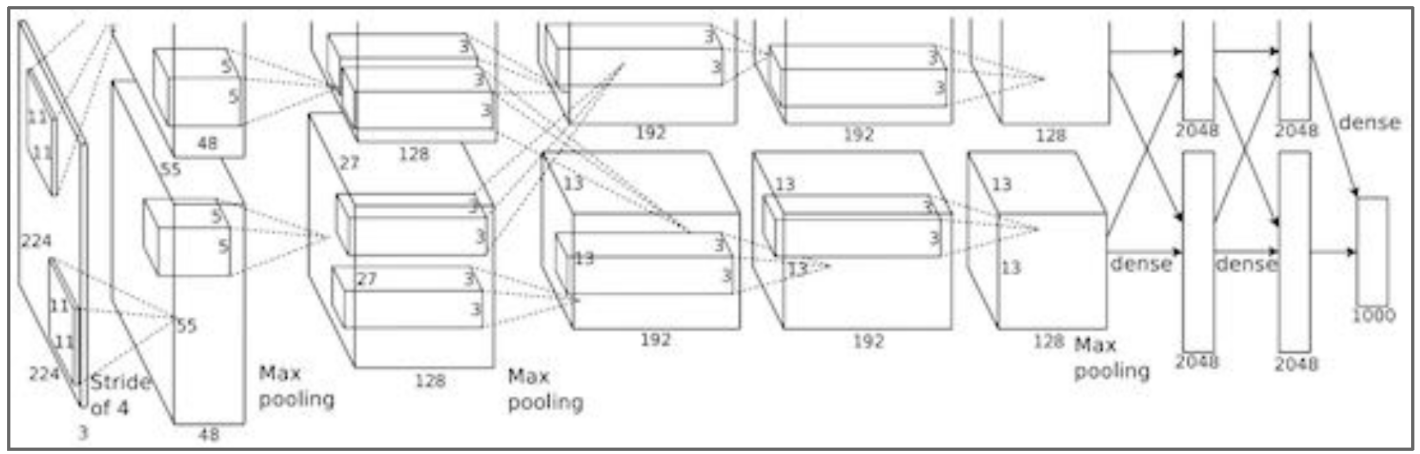


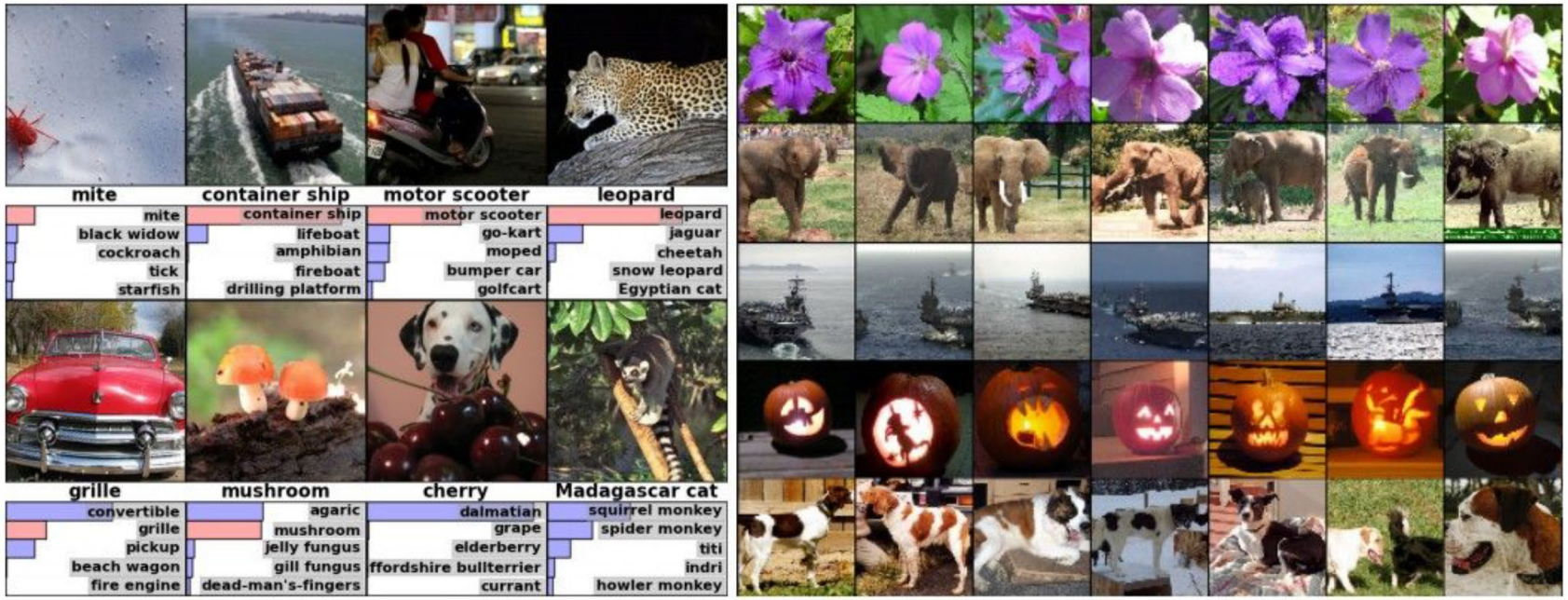
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

Fast-forward to today: ConvNets are everywhere

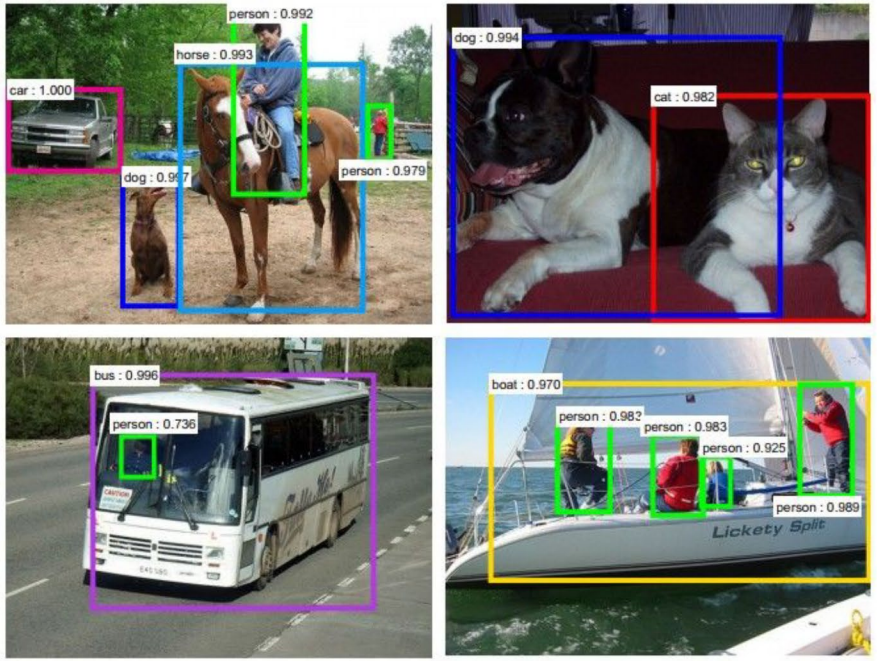
Classification

Retrieval



Fast-forward to today: ConvNets are everywhere

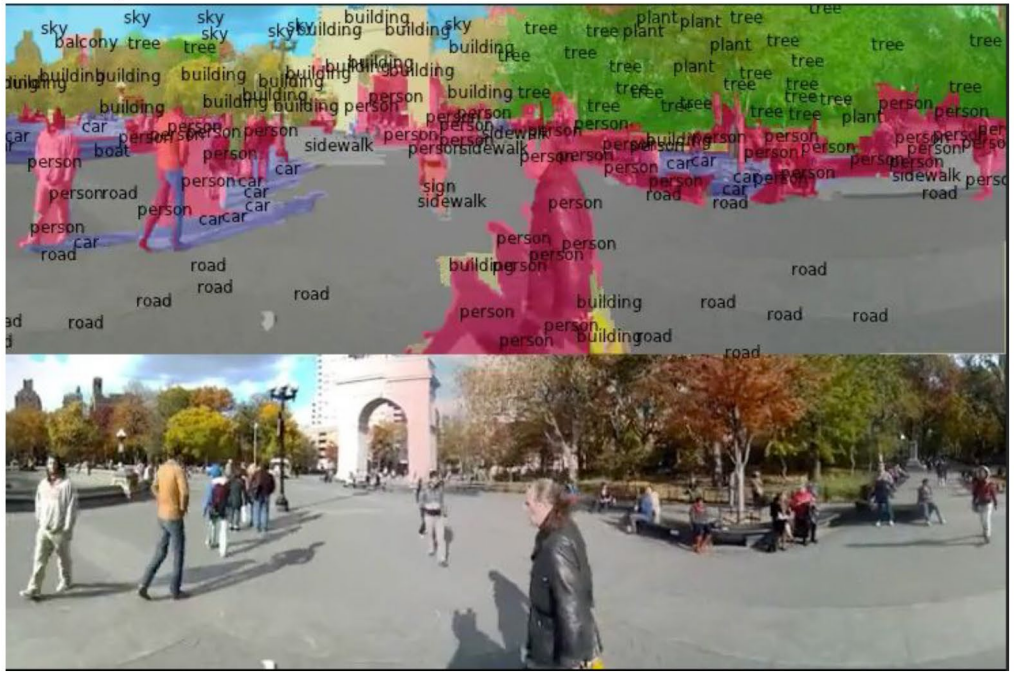
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



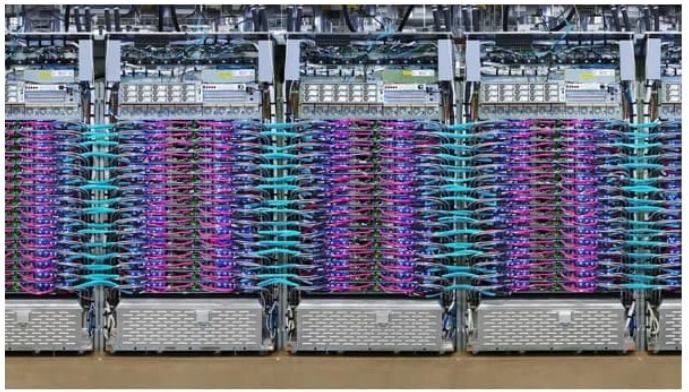
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

Self-driving cars (video courtesy Tesla)
https://www.tesla.com/AI

NVIDIA Tesla V100 GPU



Cloud TPU v3 Pod

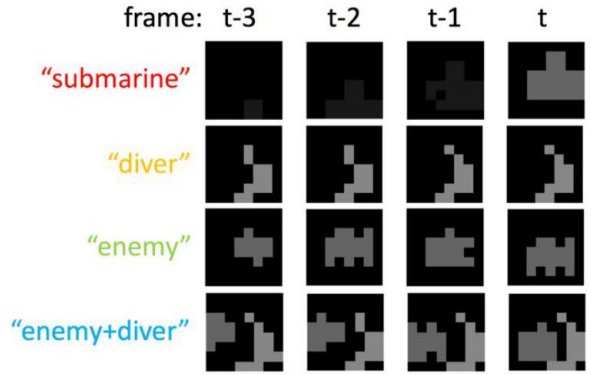
100+ petaflops

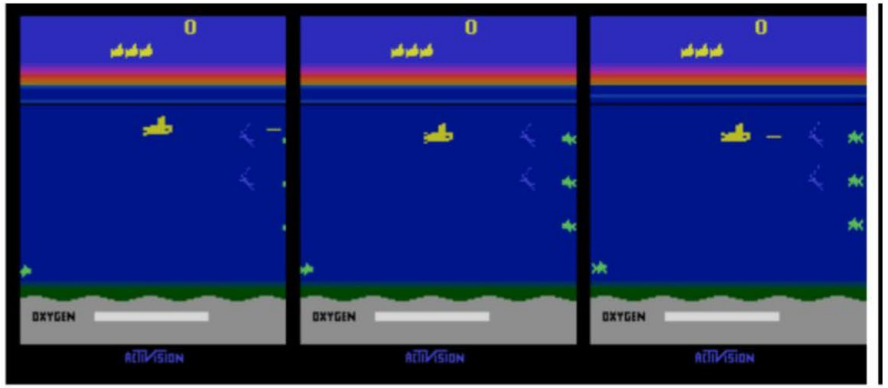
https://cloud.google.com/tpu/

Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

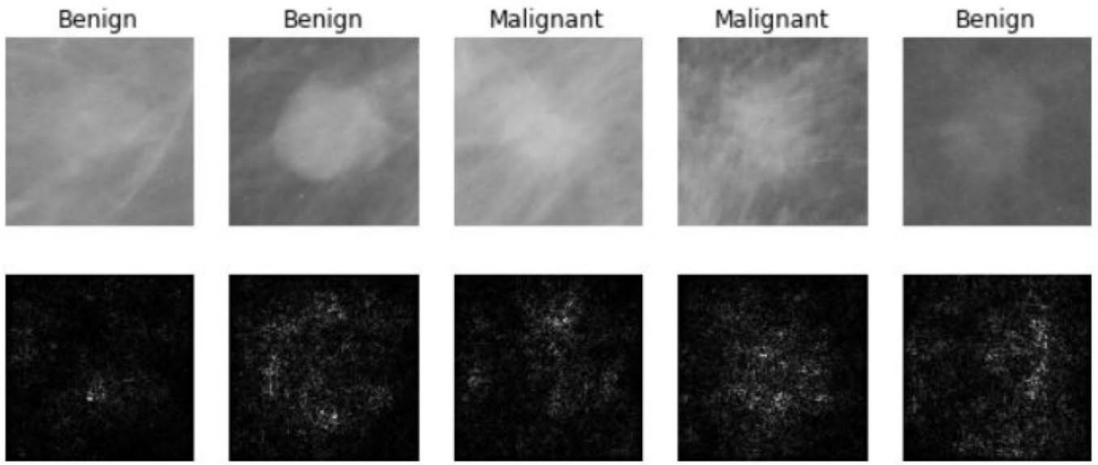




[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Sermanet et al. 2011]
[Ciresan et al.]

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2

Caption-to-image

TEXT PROMPT

an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

Edit prompt or view more images↓

TEXT PROMPT

an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES

Edit prompt or view more images↓

DALL·E: Creating Images from Text, OpenAI
https://openai.com/blog/dall-e/

TEXT PROMPT

a store front that has the word ‘openai’ written on it [...]

AI-GENERATED IMAGES

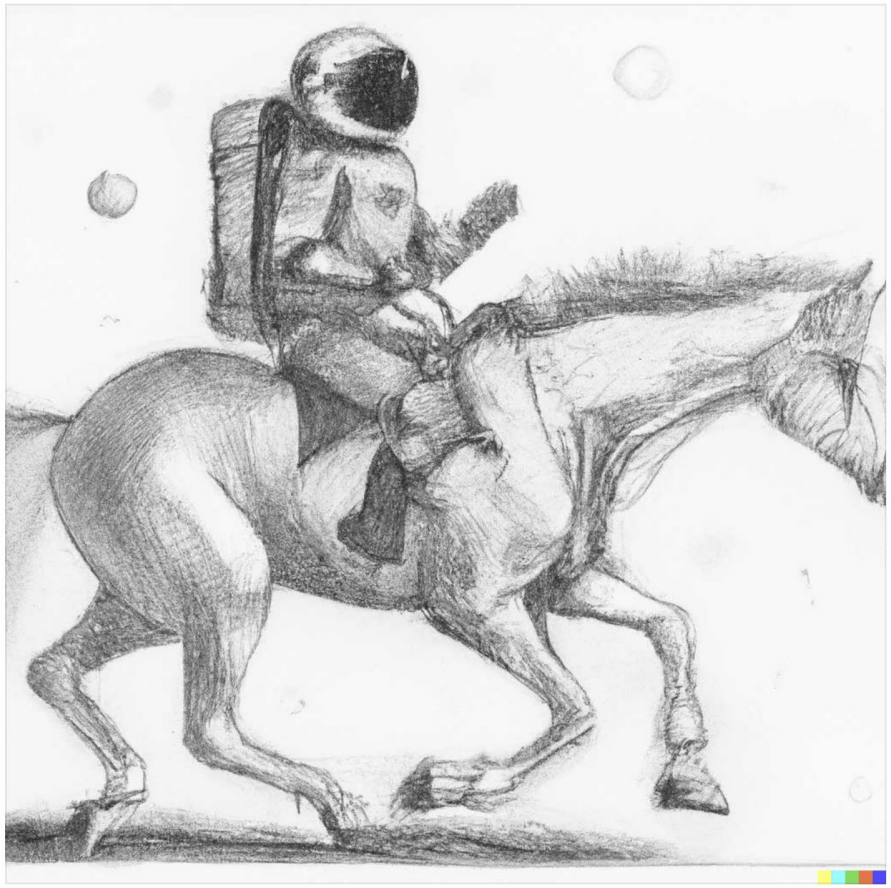
Caption-to-image

An astronaut Teddy bears A bowl of soup

riding a horse lounging in a tropical resort in space playing basketball with cats in space

in a photorealistic style in the style of Andy Warhol as a pencil drawing

→



https://openai.com/dall-e-2/