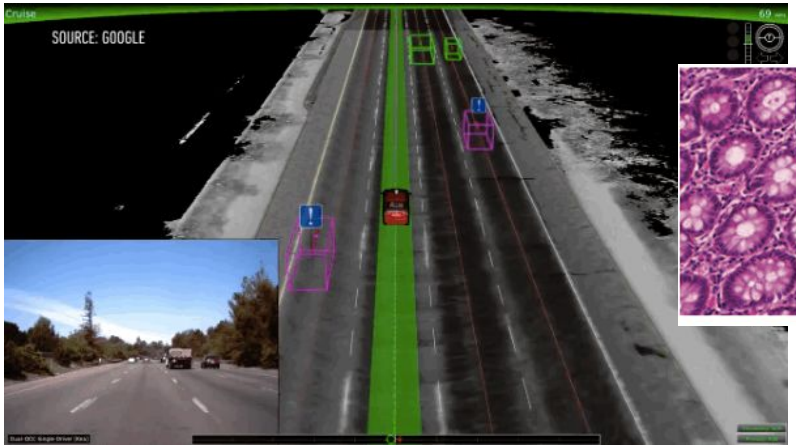
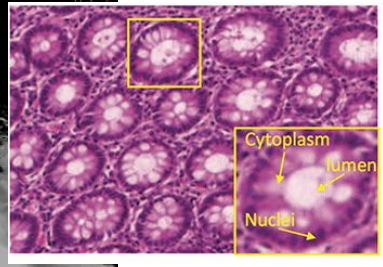
Deep Learning

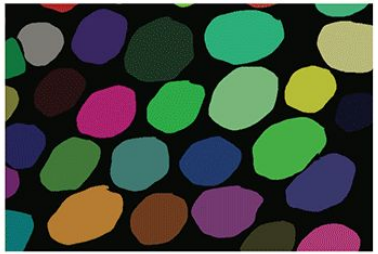
Martin Schrimpf & Jon Gauthier
MIT BCS Peer Lectures

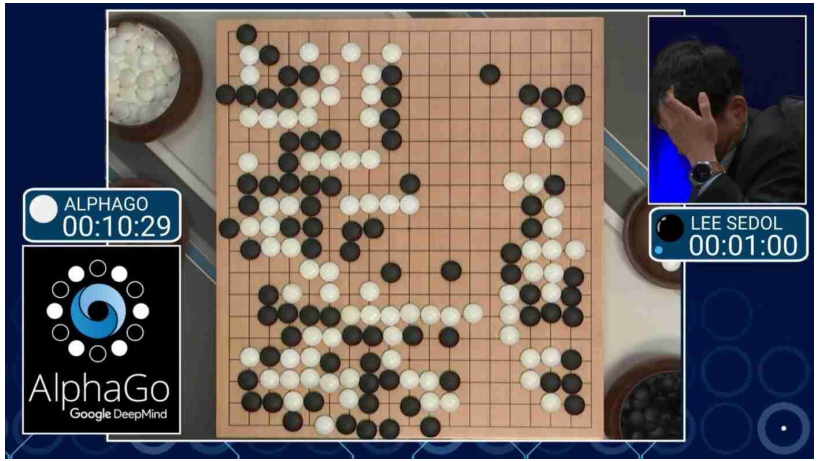
Motivation

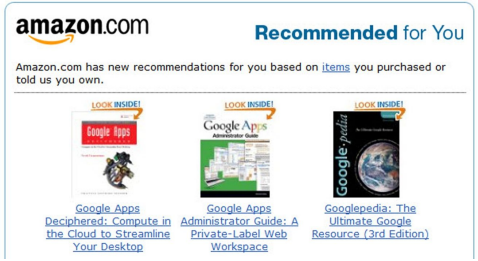
Google Translate

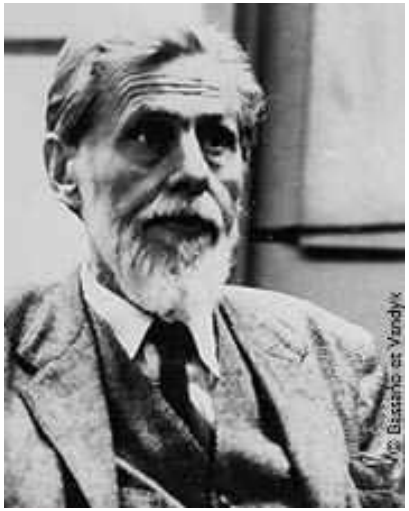












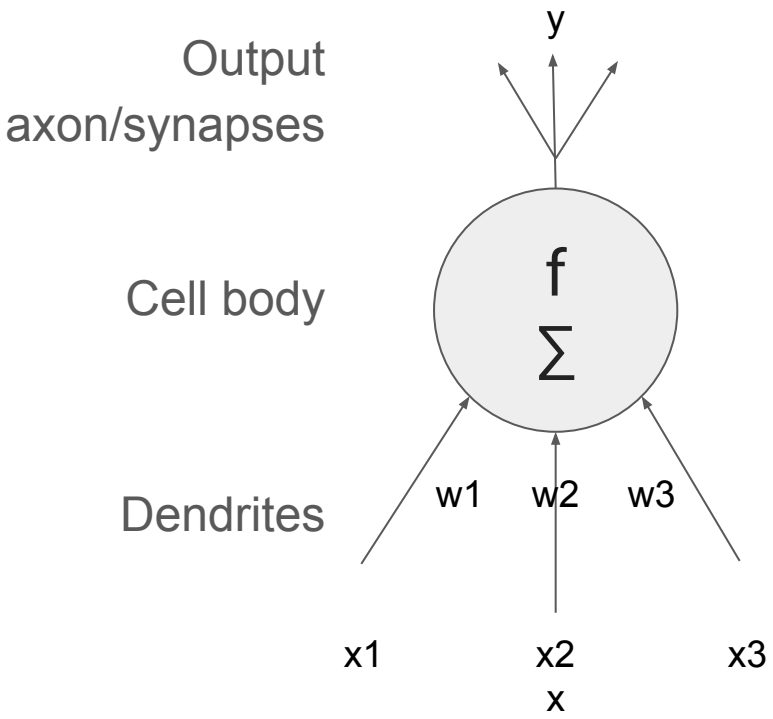
BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

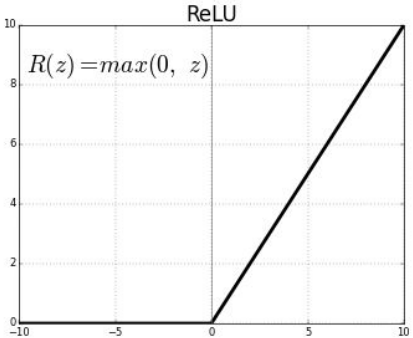
FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

A neuron

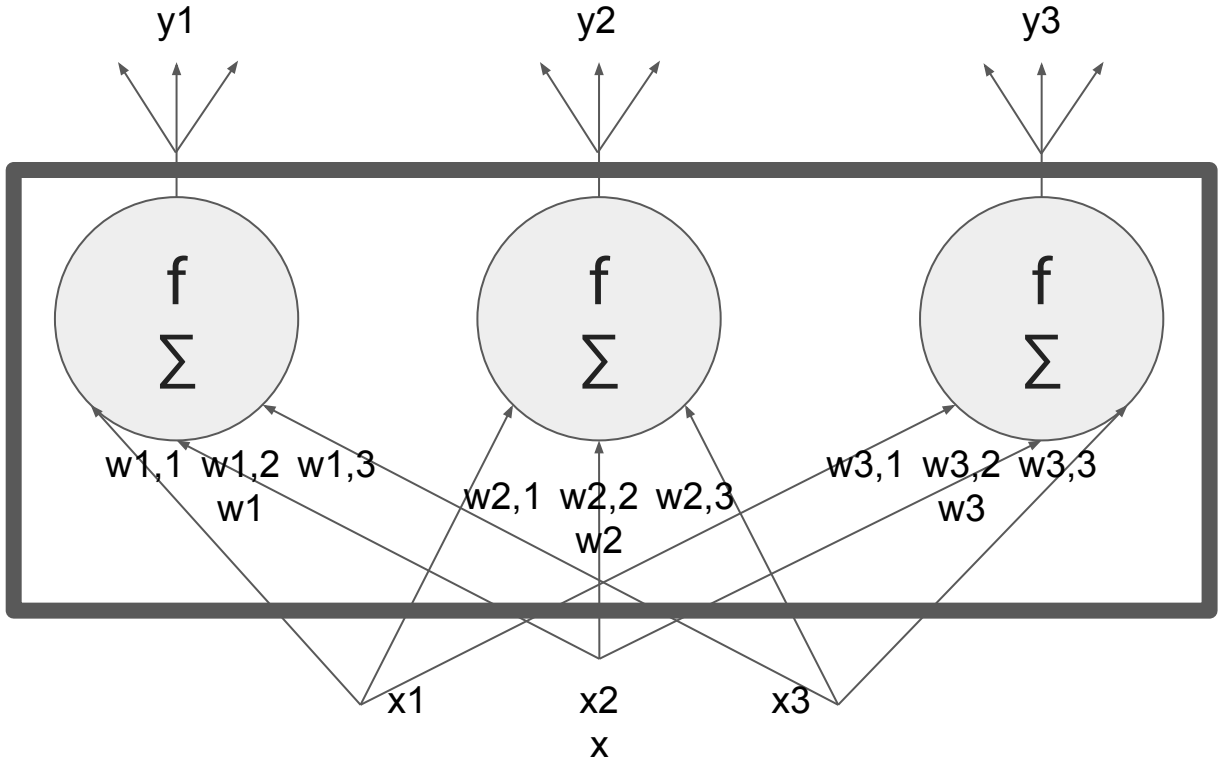


$y = f(w1 \cdot x1 + w2 \cdot x2 + w3 \cdot x3)$
$= f(w \bullet x)$

x: input vector
w: weight vector
∑: summation
f: activation function
y: output vector



A few neurons (a layer)



1 layer

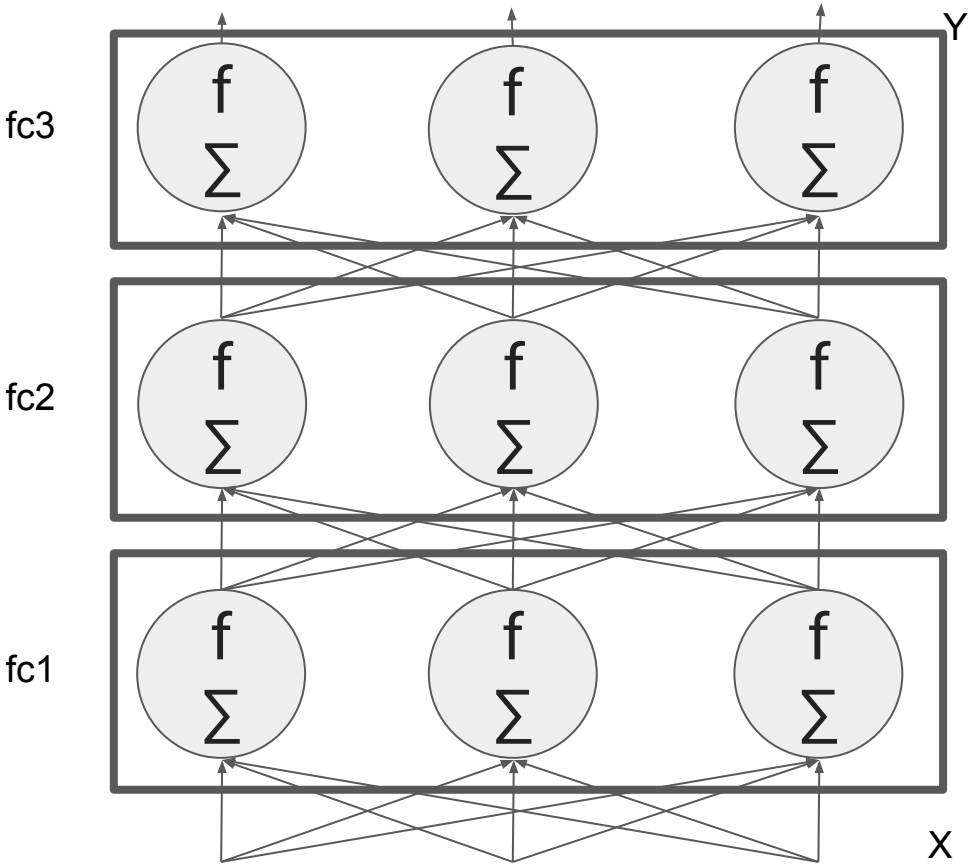
"fully-connected"

$y1 = f(w1 \cdot X)$
$y2 = f(w2 \cdot X)$
$y3 = f(w3 \cdot X)$

$W = [w1, w2, w3]$
$\mathbf{y = f(W \cdot X)}$

W is now a weight **matrix**

A few stacked layers

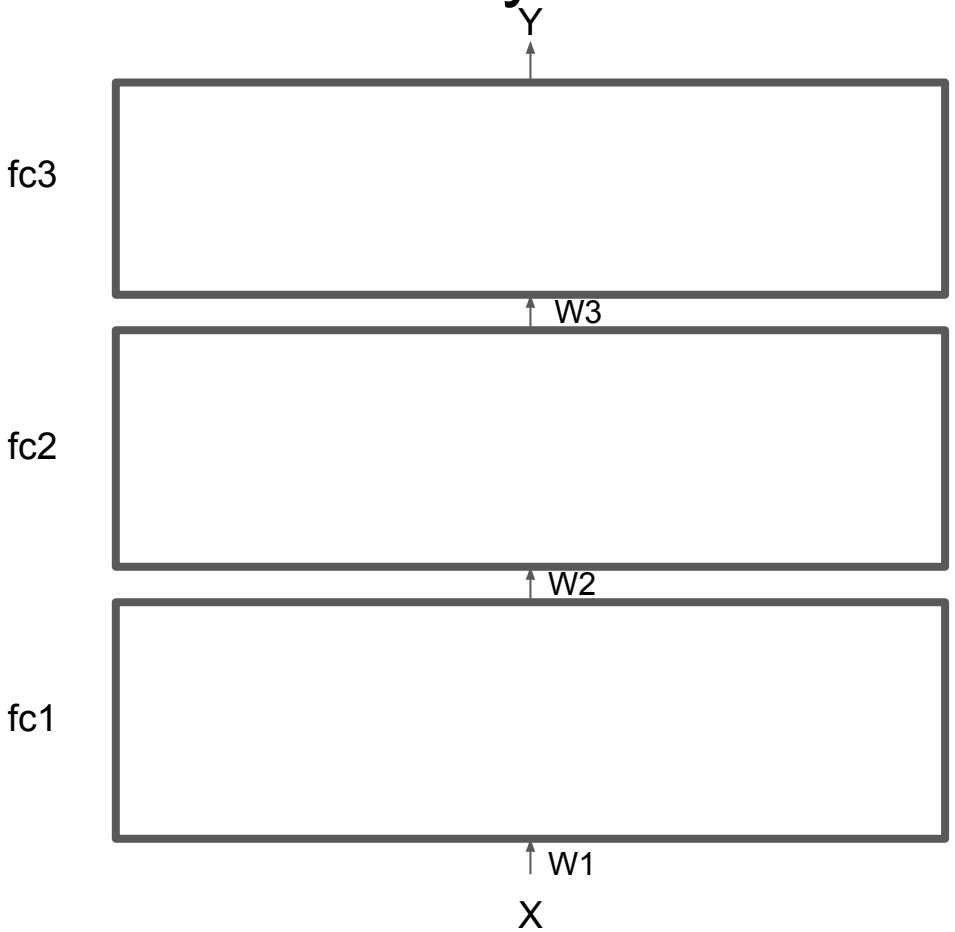


Y = fc3 = **f(W3 • fc2)** =
f(W3 · f(W2 · f(W1 · X)))
fc2 = **f(W2 • fc1)** =
f(W2 · f(W1 · X))
fc1 = **f(W1 • X)**

Each layer usually contains thousands to hundred thousands of neurons

Combine multiple input vectors x1, x2, …, xn to input matrix X which yields an output matrix Y.

A few stacked layers



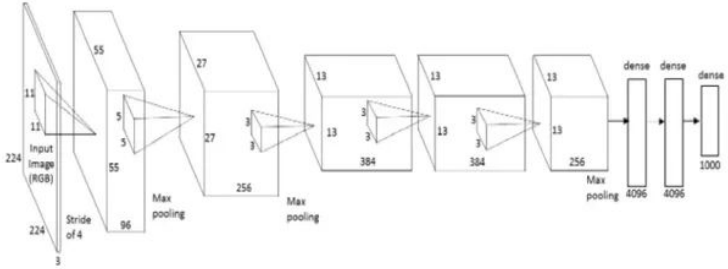
$$Y = fc3 = f(W3 \cdot fc2) = f(W3 \cdot f(W2 \cdot f(W1 \cdot X)))$$
$$fc2 = f(W2 \cdot fc1) = f(W2 \cdot f(W1 \cdot X))$$
$$fc1 = f(W1 \cdot X)$$

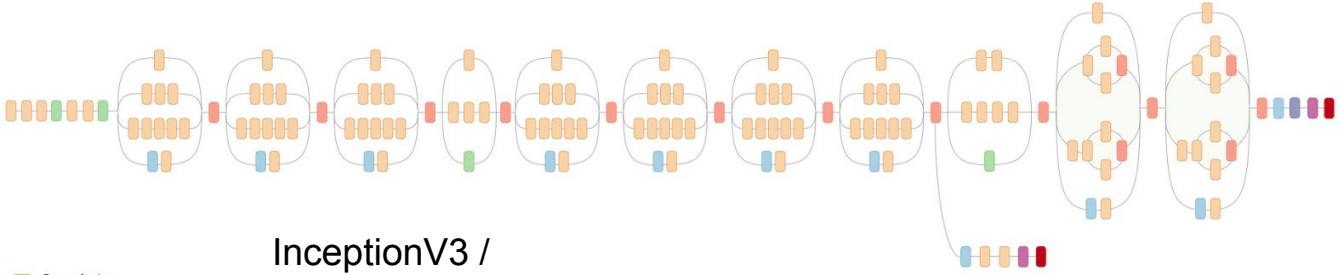
Each layer usually contains thousands to hundred thousands of neurons

Combine multiple input vectors x1, x2, …, xn to input matrix X which yields an output matrix Y.

A few more (the real world)

Alexnet



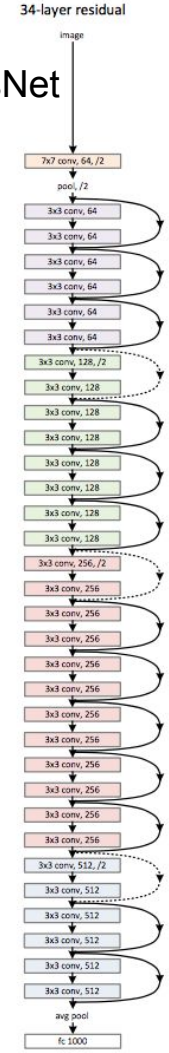


InceptionV3 /
GoogLeNet

Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax

VGG16

ResNet



How to train W? 1. Loss functions

Exchangeable terms: *Loss function = cost function = -(objective function)*

In supervised learning, loss functions describe **deviation** between predictions Ŷ and ground-truth outputs Y.
Pick an objective which best describes prediction error for your task!

Regression: L2 loss / mean squared error

$$L = \frac{1}{N} \sum_{1 \leq i \leq N} (y_i - f(x_i))^2$$

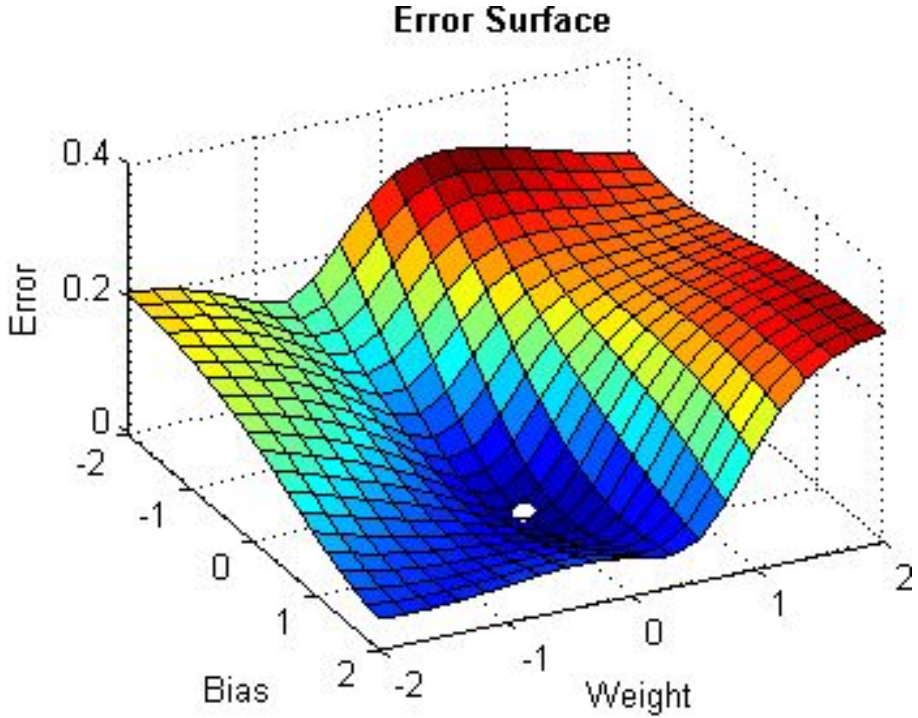
Classification: relative entropy

$$L = \frac{1}{N} \sum_{1 \leq i \leq N} D_{KL}(\delta(y_i) || f(x_i))$$

How to train W? 2. Follow the gradient

- If our network is completely linear, we can maximize the objective using basic calculus. (This yields linear regression or logistic regression.)
- Nonlinearities create **non-convexity** (saddle points and local optima) in the objective
- But **gradient descent**, a popular convex optimization tool, still works!

Gradient descent



$$w_i := w_i - \eta \frac{\partial L(\mathbf{x}, \mathbf{y})}{\partial w_i}$$

$$\mathbf{w} := \mathbf{w} - \eta \nabla L(\mathbf{x}, \mathbf{y})$$

How to train W with many layers? 3. Backpropagation!

[Werbos+ 1982; Parker+ 1985; LeCun+ 1985; Rumelhart, Hinton+ 1986; …]

Considering a single weight j at neuron i (w_ij):
given error E, the neuron's output o_i, and learning rate η, the weight change is:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j$$

minimize

gradient

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j)o_j(1-o_j) & \text{if } j \text{ is an output neuron,} \\ (\sum_{\ell \in L} \delta_\ell w_{j\ell})o_j(1-o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

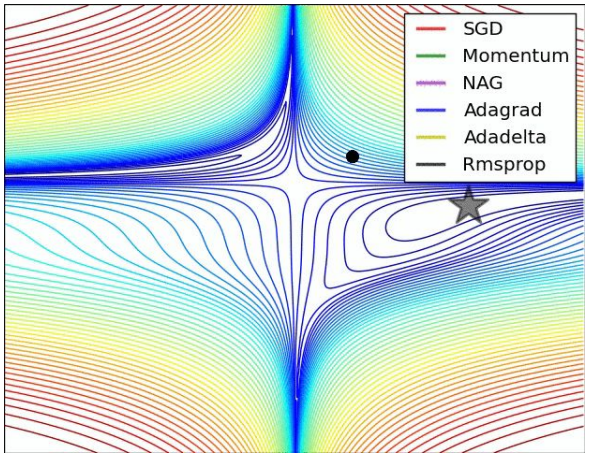
this is the backprop part

The Wikipedia article on this is pretty good! https://en.wikipedia.org/wiki/Backpropagation

Oodles of ways to follow the gradient

Big ideas:

1. Use the gradient to specify an acceleration rather than a velocity (Momentum, NAG, RMSProp)
2. Optimize with a memory of past updates (Adagrad, Adadelta)



Initialization: where do the weights come from?

What happens if we initialize the weights to zero?

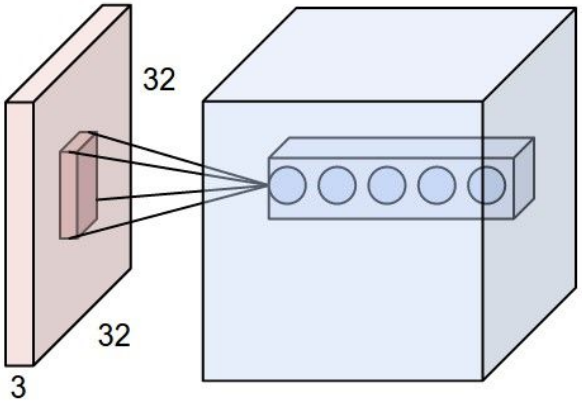
What happens if we initialize the weights to the same number?

What happens if we initialize the weights to numbers of different magnitude?

Convolutions (ConvNets/CNNs)

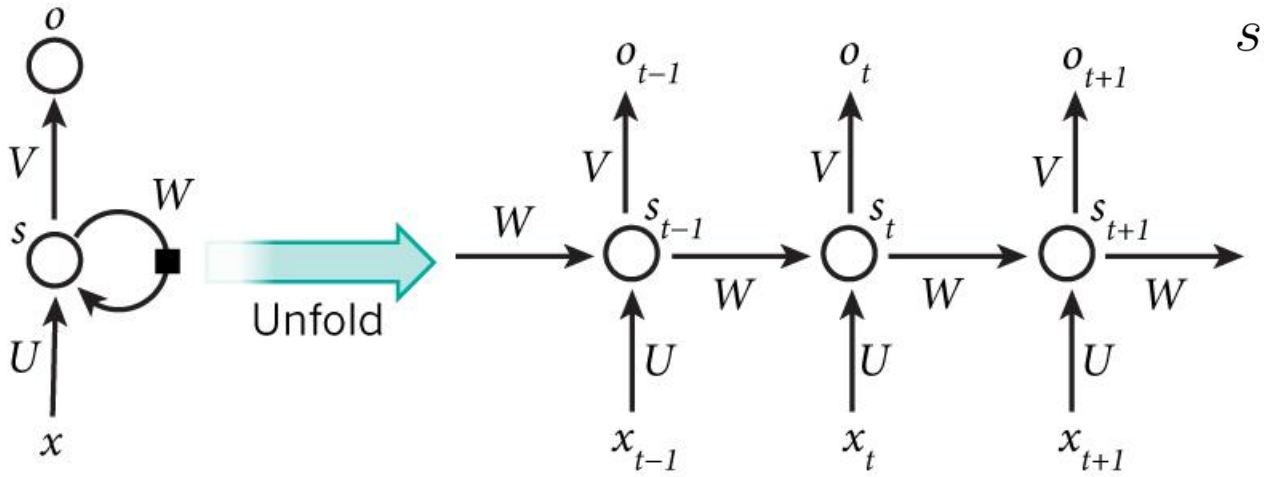
Fully-connected layers can be hard to train due to the number of weights

Idea: if a feature (e.g. edge) at the bottom left of the image is important, it is also important at all other locations
→ apply the same weights (“filters”) across space



Recurrency (RNNs)

Use to handle variable-length input (e.g. text in Natural Language Processing)



$$s_t = f(Ux_t + Ws_{t-1})$$

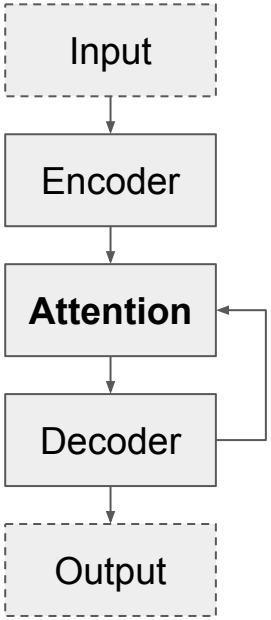
Basically fully-connected layers with weights shared over time

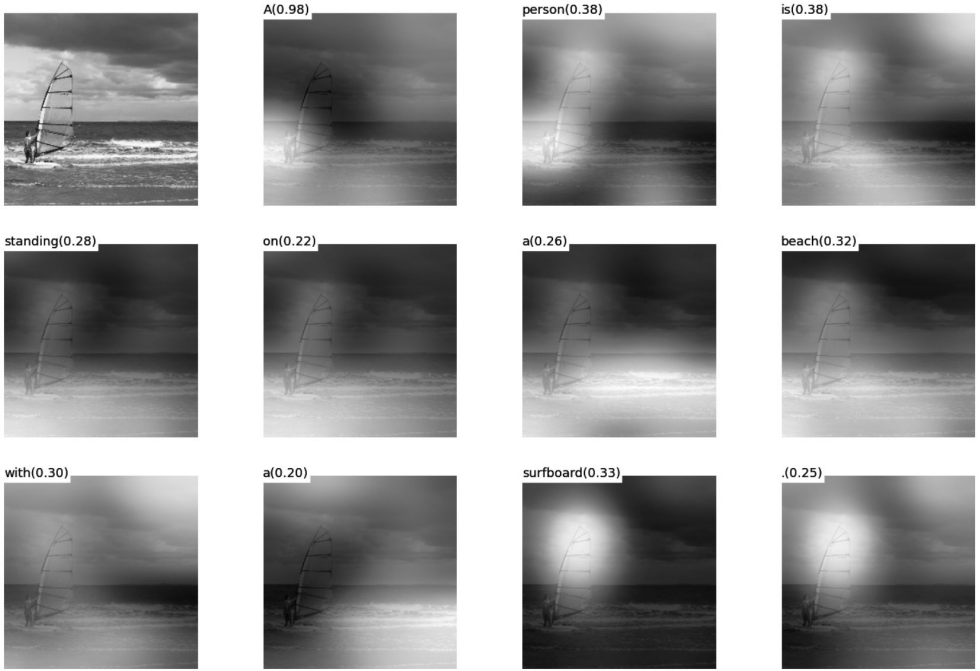
More advanced RNNs with explicit memory: LSTM, GRU

Attention

Shift focus by weighing encoded input with recurrent attention matrix

Relevant for images, text, ...

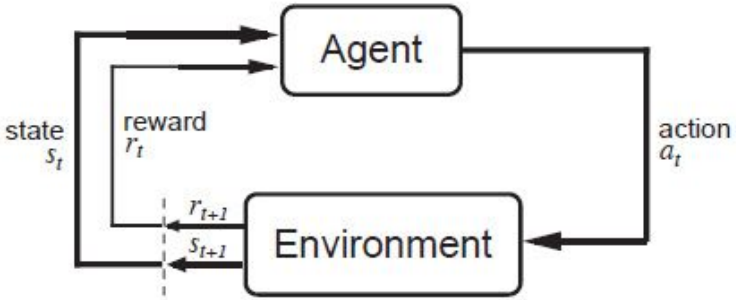


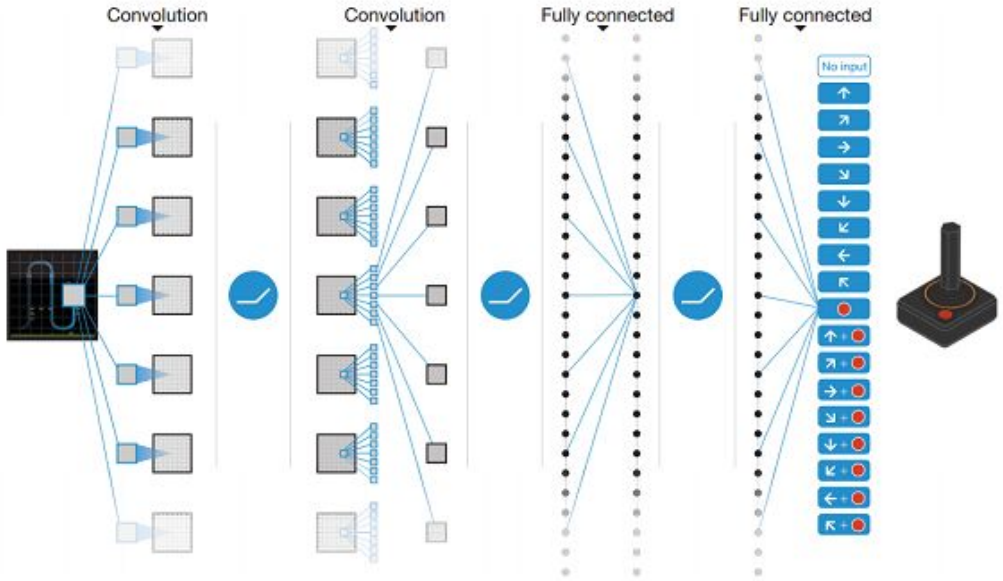


(b) A person is standing on a beach with a surfboard.

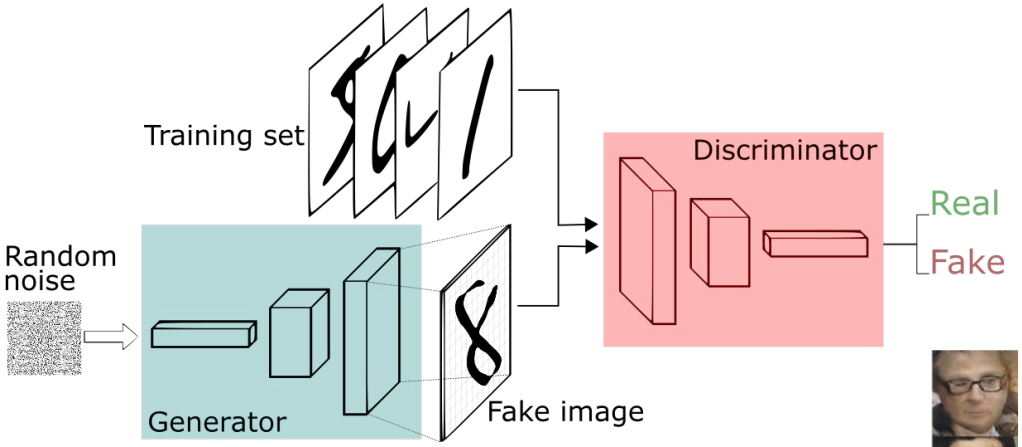
Deep Reinforcement Learning

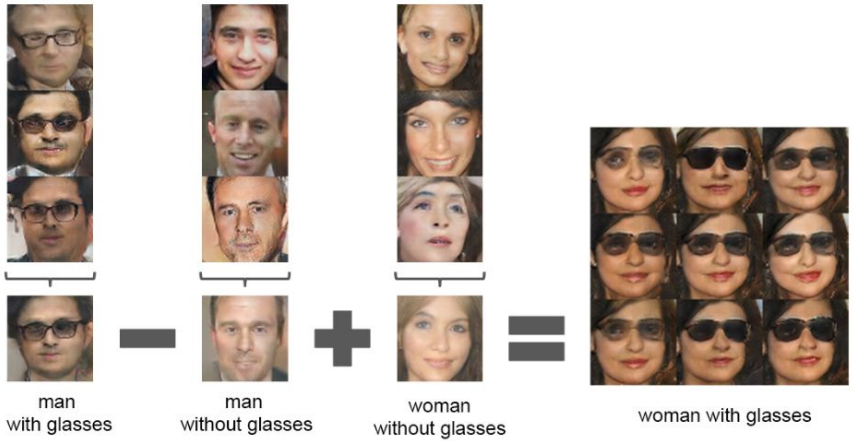
Train deep nets with e.g. REINFORCE → deep RL





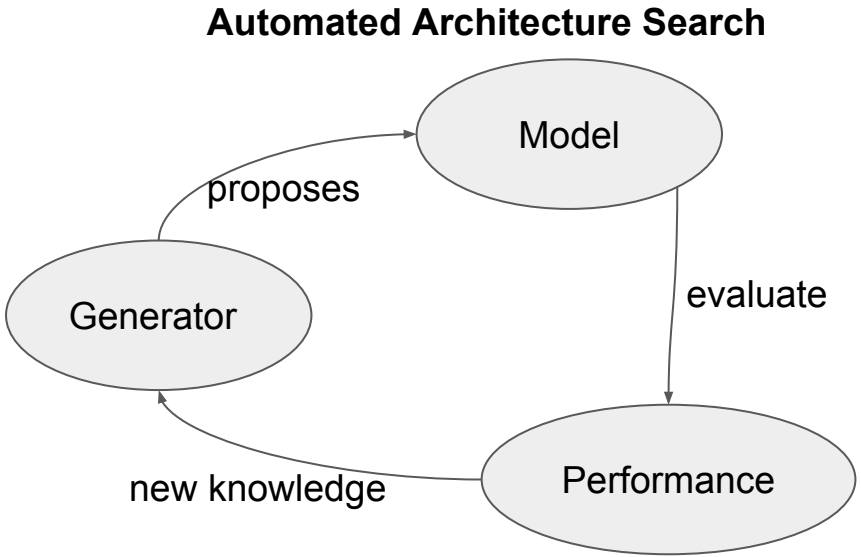
Generative Adversarial Networks (GANs)





Hyperparameters

- Architecture (so many!)
- Optimizer
- Training details (so many!)



Frameworks

Recommended (highly dependent on use-case!)

- Matlab (very high-level, low flexibility to develop new models, but perhaps suited for quick feature computation)
- Keras (high-level, some flexibility; better flexibility when coding in TensorFlow backend)
- PyTorch (medium-level, full flexibility, good debugging)

Others: TensorFlow, Torch, Caffe, CNTK, MXNet, ~~Theano~~

Open Questions

1. Few-shot learning: DL needs millions of examples to train successfully - humans can learn from often just a single example and generalize like crazy
2. Unsupervised learning
3. Multi-sensory/-domain integration
4. Memory?
5. Do we need feedback loops?
6. Importance of “sleep”
7. How to reason about decisions? What is actual “understanding”?
8. How can we intuitively understand millions of weights?
9. Is backpropagation/SGD the right learning rule?
10. Does the brain do backprop? Maybe.
11. Rethink everything and use capsules?

    ...

If you have neuroscience-y ideas for any of these, let’s chat! There’s likely a project in it

Reading recommendations

Pattern Recognition and Machine Learning, Chris Bishop. 2006
(the ML classic book, good to understand the maths)

Deep learning, Yann LeCun and Yoshua Bengio and Geoffrey Hinton. Nature 2015 (short overview)

Deep Learning in Neural Networks: An Overview, Jürgen Schmidhuber. Neural Networks, 2015 (historical overview)

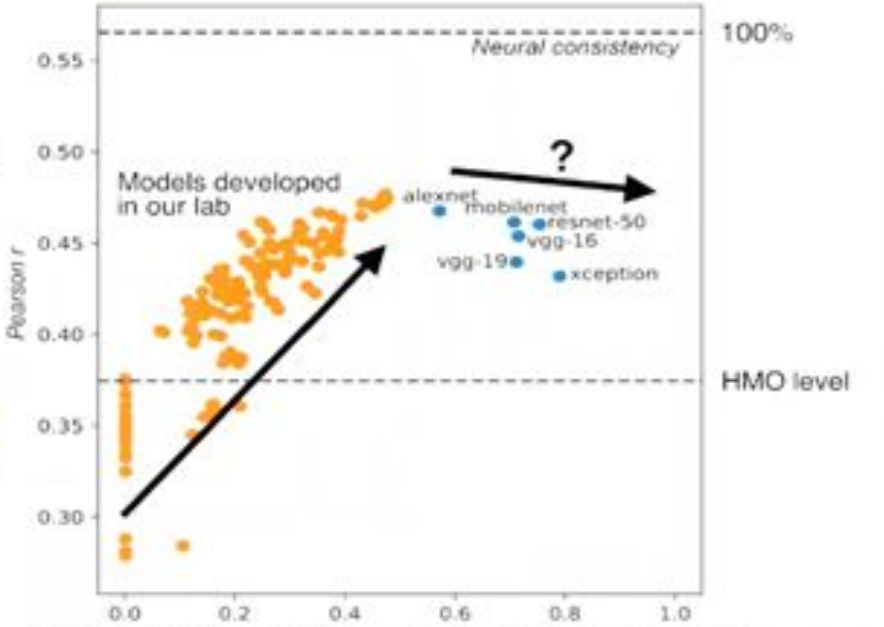
Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville. 2016
(many people recommend this book, fully available online)

Backup

Can neuroscience just sit back and wait for even better answers?

Ability of last hidden layer of the deep CNN model to explain IT responses*

A neuroscience goal



A computer vision goal

(*90-110 msec time window)

DiCarlo, unpublished

Some training tips

Split data into dev (80%) and test (20%)

Split dev into

- dev-train (60%)
- dev-validation (20%) ← avoid overfitting on dev-train
- dev-test (20%) ← test generalization to optimize model

Test ← test actual generalization of model (we optimize on dev-test by hand)