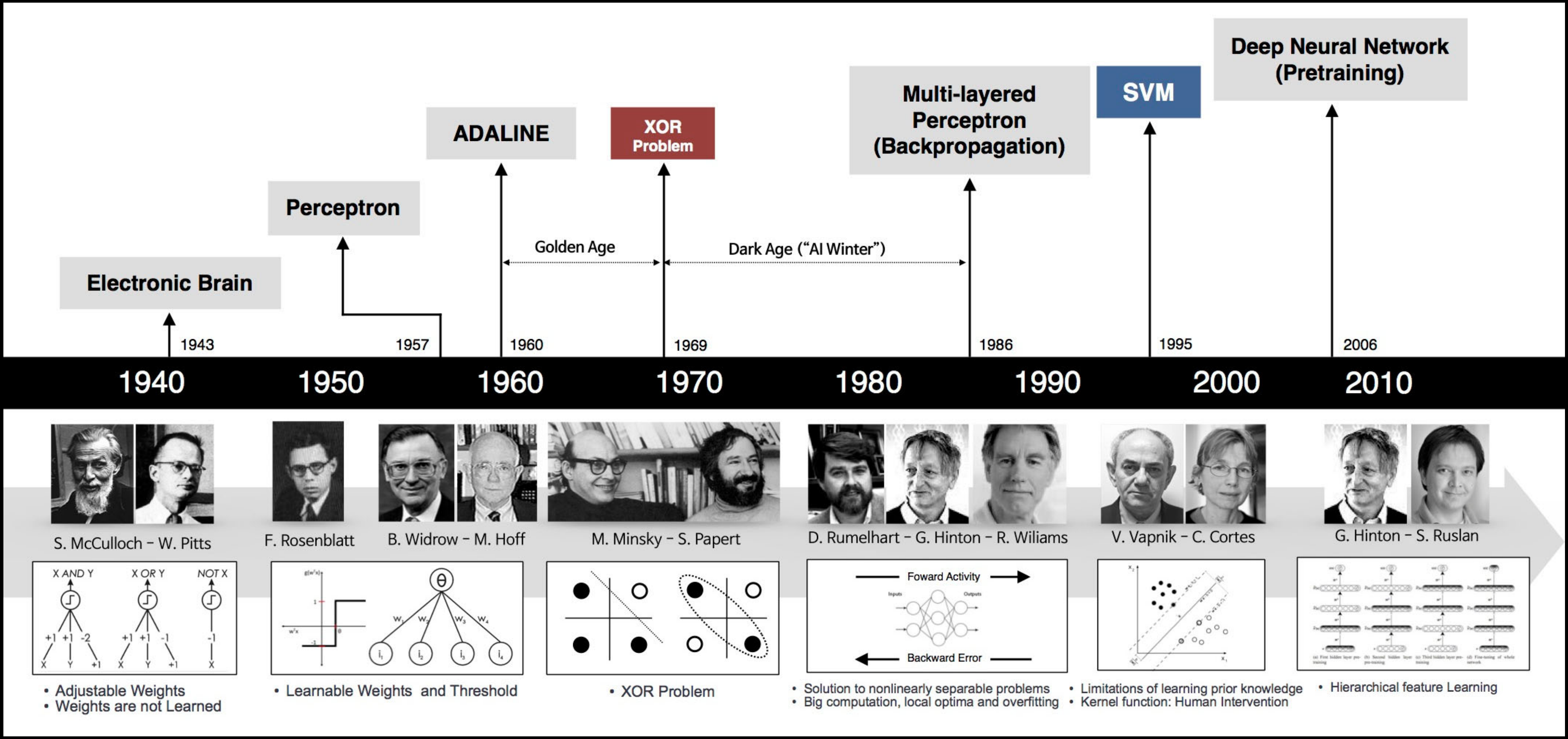
Deep Learning

COGS-Q355

Deep Learning – Overview

- More than 1 hidden layer, up to 50, 100, or more
- Makes data preprocessing unnecessary!
- Examples: Digit recognition, modeling the visual system

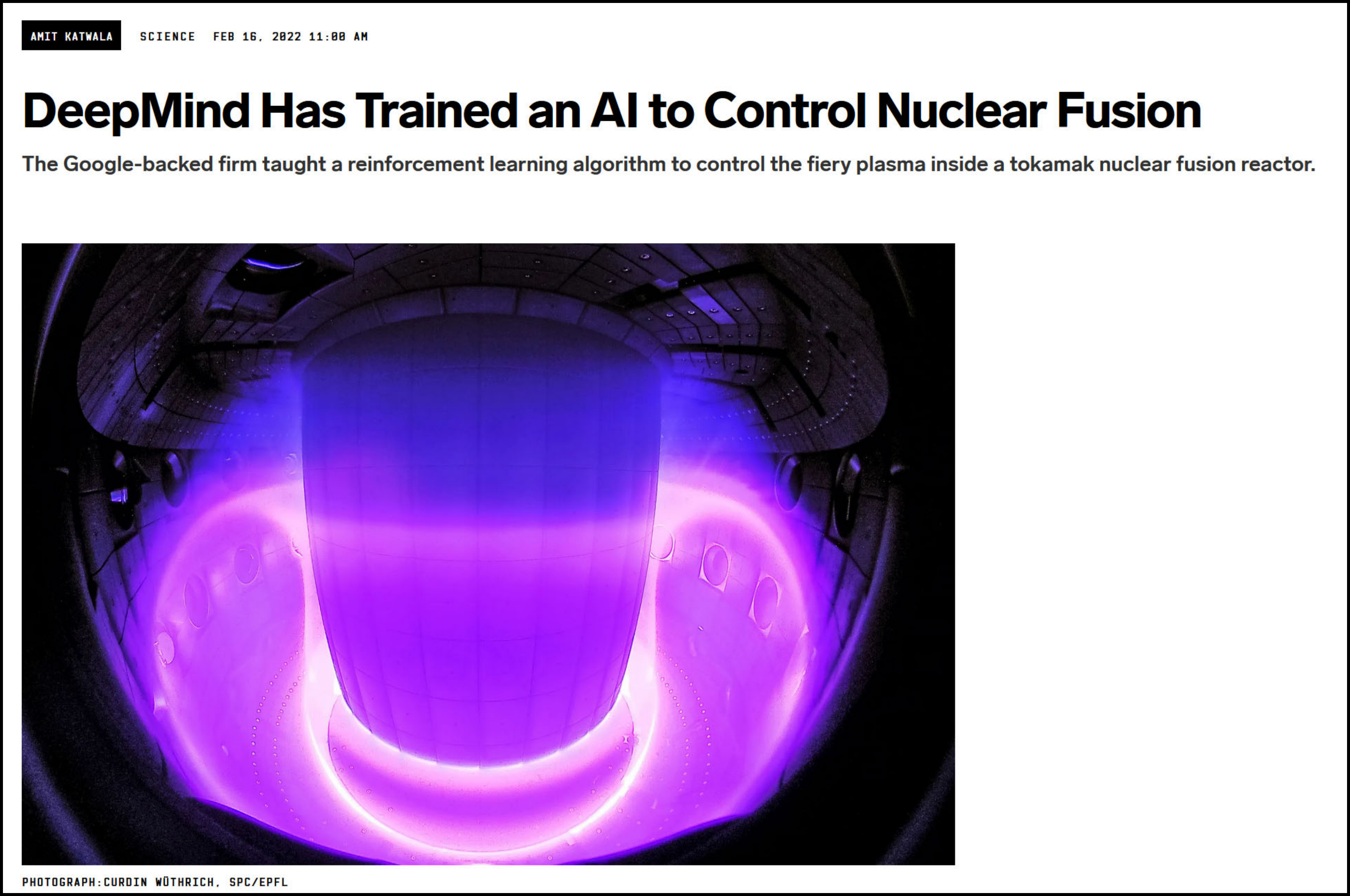
Deep Learning – History


Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

- 1940-1960 – No hidden layers
- 1986 – One hidden layer
- 2006 – Two or more hidden layers

https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

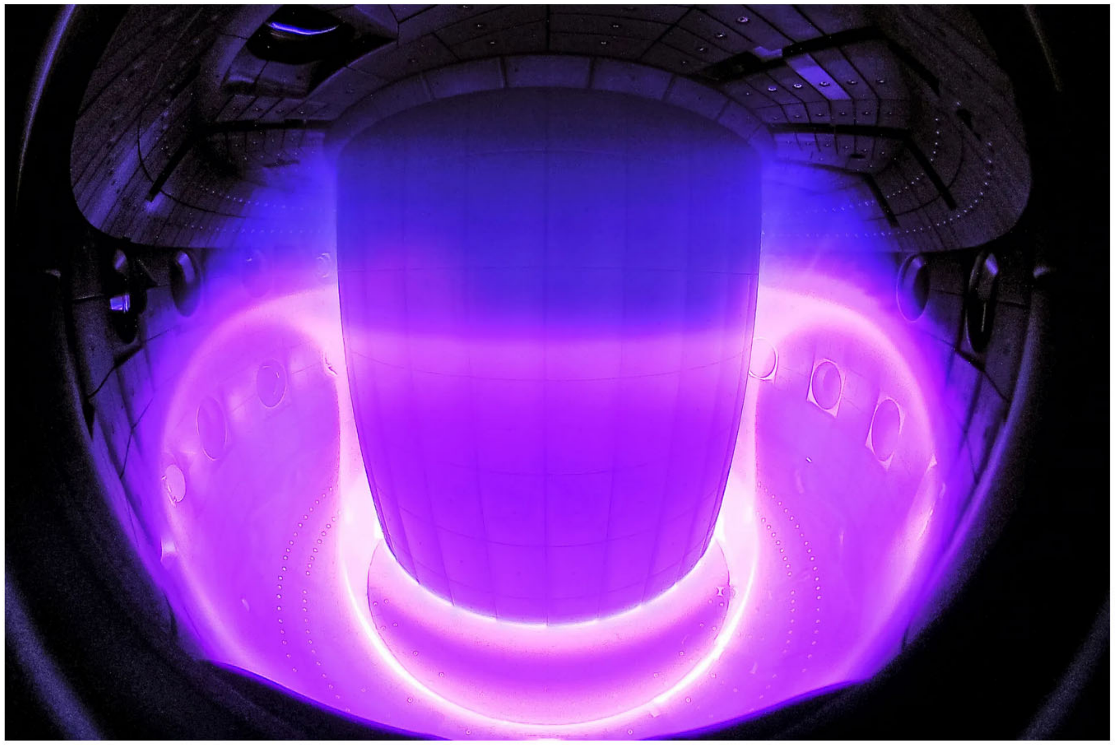
Deep Learning – Today



AMIT KATWALA SCIENCE FEB 16, 2022 11:00 AM

DeepMind Has Trained an AI to Control Nuclear Fusion

The Google-backed firm taught a reinforcement learning algorithm to control the fiery plasma inside a tokamak nuclear fusion reactor.

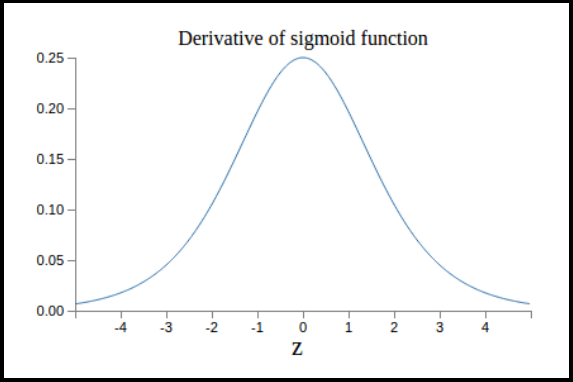


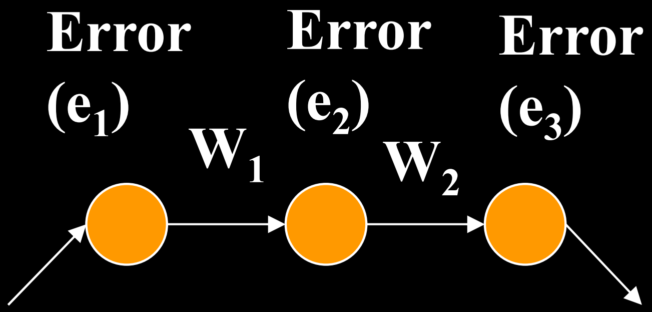
PHOTOGRAPH: CURDIN WÜTHRICH, SPC/EPFL

https://www.wired.com/story/deepmind-ai-nuclear-fusion/

Deep Learning – gradient problem

- vanishing gradient problem (or exploding gradient problem)
- Recall the learning laws:
  - Top layer: W(new) = w(old)+ $\alpha * error * f'(net) * y$
  - Earlier layers, error backpropagated: error(lower) = error(higher)*W*f'(net)
  - If f(net)=$\frac{1}{1+e^{-x}}$, then f'(net)=$\frac{e^{-x}}{(1+e^{-x})^2}$
- But this maxes out at 0.25!
- → Every time we propagate the error backward to an earlier layer, we multiply the error by the gradient of 0.25 or less.
- Eventually, the error becomes a very small number, and the learning rate becomes very slow in earlier layers! (0.25*0.25*… = small number!)





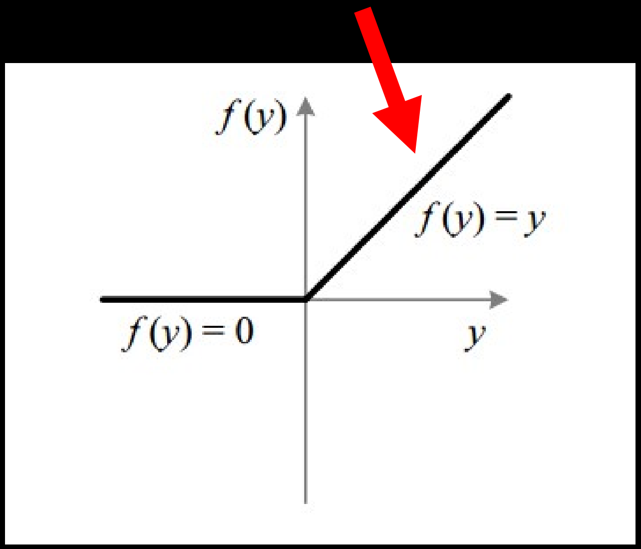
e_2=f'(net_2)*W_2*e_3

e_1=f'(net_1)*$W1_2$*e_2

Deep Learning – gradient problem

- How can we solve the vanishing gradient problem?
- First, choose a signal function that doesn’t have small gradients, like a “Rectified Linear Unit” (ReLU).
- People resisted this idea at first because f’(net) is undefined at zero (and f’(net)=0 for input < 0, but this turns out not to be a problem

F’(net) is 1, problem solved



Deep Learning – gradient problem

- Second, choose a cost function that doesn't involve the signal function derivative, like Cross Entropy
- Recall that originally, the loss function was E=$\sum_i e_i^2$, which led to an error function with the sigmoid derivative

- With Cross entropy, the cost function is
- $C = -\frac{1}{N}\sum_j \quad \sum_N [d \ln x + (1-d)\ln(1-x)]$
- The weight update value is $\frac{\partial C}{\partial w} = \frac{1}{N}\sum_x x_j(\sigma(x) - y)$, which has NO derivative of the sigmoid function. That helps at least at the top layer. (d = desired out, x = actual out)
- With the vanishing gradient problem solved, we can have *lots* of hidden layers without slowing down the learning!

http://neuralnetworksanddeeplearning.com/chap3.html

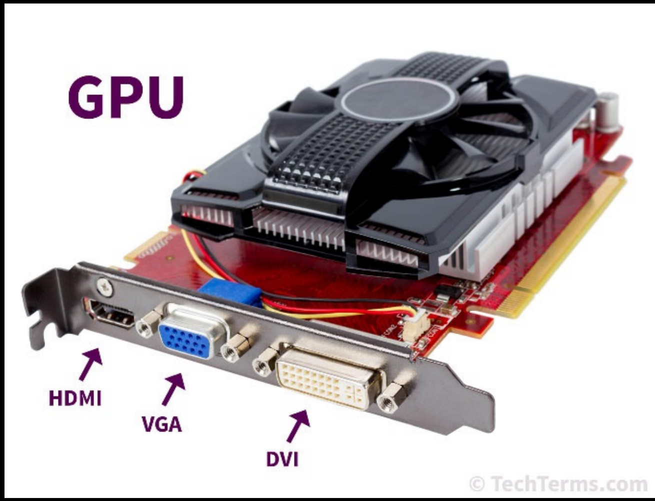
Deep learning history

- Before 2006, deep learning wasn’t really a thing – it was just backprop, which had been around since 1986.
- Geoff Hinton’s work in 2006 started the deep learning trend by pre-training lower layers to extract features automatically (previously this would have been tediously hand-tuned)
- In 2012, Alexnet (again, Geoff Hinton) used GPUs and got much improved performance on image classification

https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Deep learning history

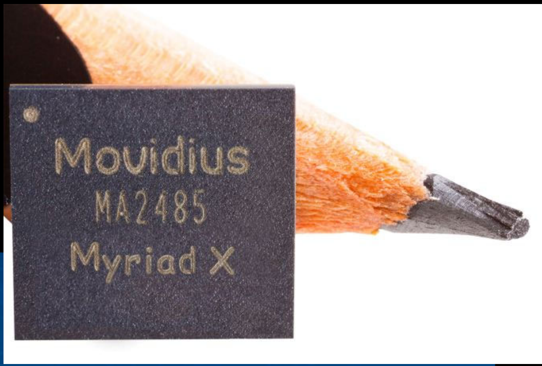
- With dramatic improvements in performance, big tech paid attention and started scooping up neural network talent and startups
- What else made deep learning a thing?
- GPUs – graphical processing units dramatically speeded up processing by 2-3 orders of magnitude (now TPUs…)
- Larger datasets for training/testing
- Software platforms – tensorflow, theano, etc.
- Regularization techniques – e.g. dropout
- Dramatic improvements since 2006

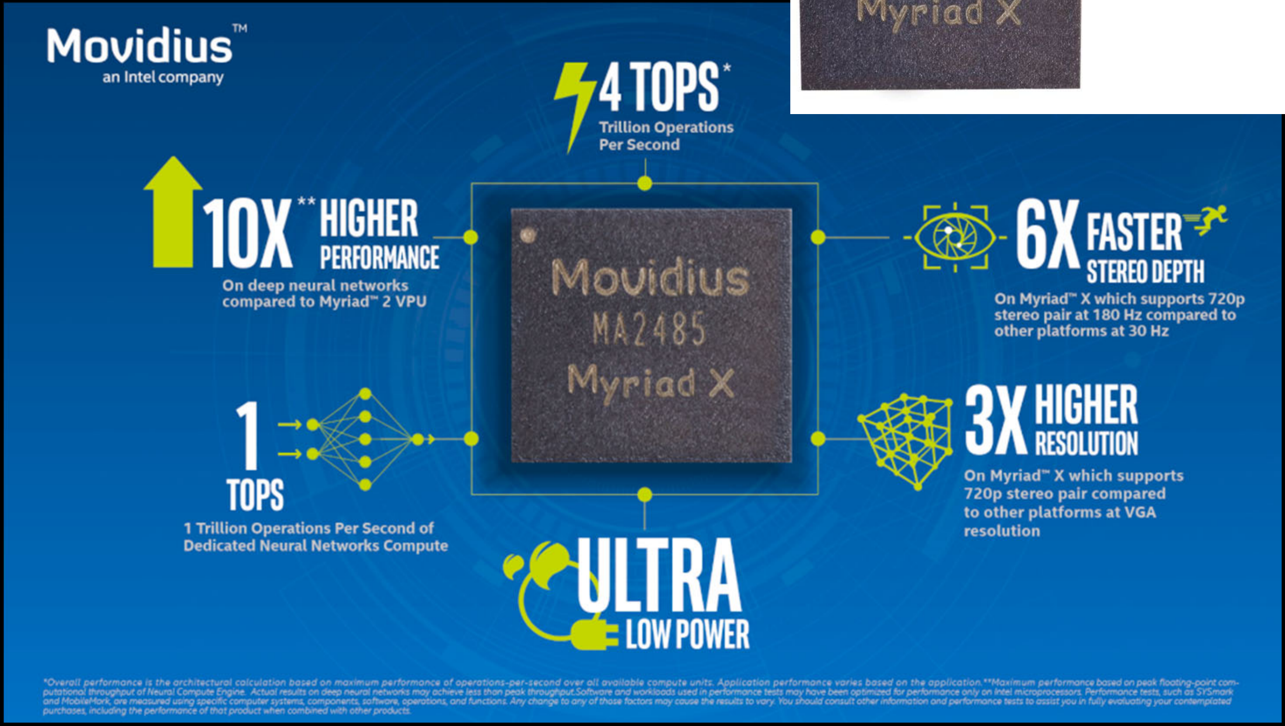


https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Deep learning tech – rapid growth

- Tensorflow – google's open source deep learning toolkit in python (also browser tensorflow.js)
- Theano (obsolescent)
- Hardware – Intel movidius / myriad x



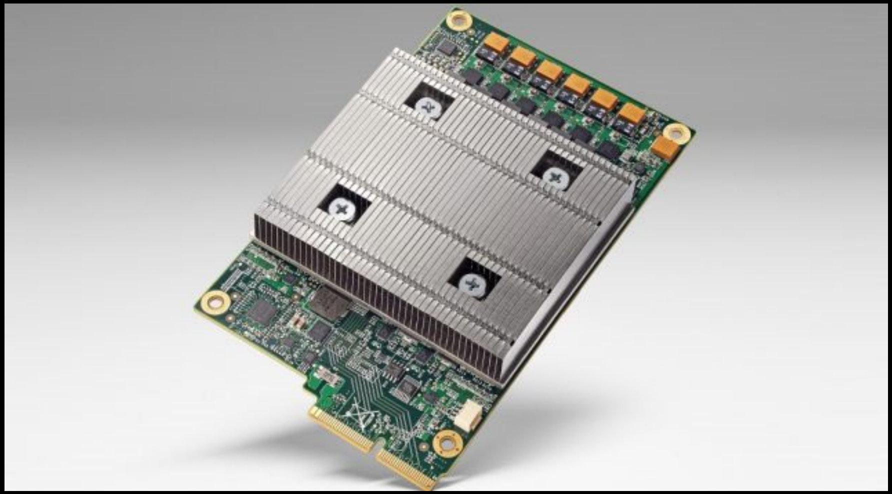

Movidius™
an Intel company
4 TOPS*
Trillion Operations Per Second
10X** HIGHER PERFORMANCE
On deep neural networks compared to Myriad™ 2 VPU
Movidius
MA2485
Myriad X
6X FASTER STEREO DEPTH
On Myriad™ X which supports 720p stereo pair at 180 Hz compared to other platforms at 30 Hz
1 TOPS
1 Trillion Operations Per Second of Dedicated Neural Networks Compute
3X HIGHER RESOLUTION
On Myriad™ X which supports 720p stereo pair compared to other platforms at VGA resolution
ULTRA LOW POWER

Movidius API only does feedforward networks, not recurrent networks currently – work in progress

Cf. Google Coral edge TPU

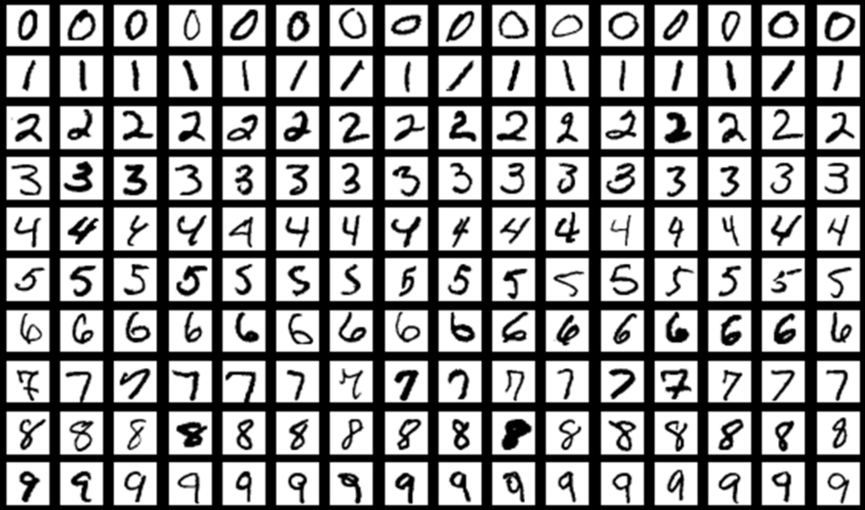
Deep learning Hardware

- CPU – Central Processing Unit
- GPU – Graphical Processing Unit (100-1000x faster)
- TPU – Tensor Processing Unit (a lot faster than GPU for neural networks) – low precision, like 8 bit, but massively parallel. Google's gen3 TPU (2022) – 420 Teraflops



Example deep learning: MNIST

- The MNIST image database (modified National Institute of Standards and Technology)
- 28x28 images of handwritten numbers 0-9, with known number values
- 60,000 training images, 10,000 test images
- Current best deep neural network gets only 23 wrong out of 10000 (error rate 0.23%, Cireşan et al. 2012)



MNIST model

- How does Ciresan et al. do it? Several features:
- Convolutional neural networks
- Shared weights
- Max pooling
- Dropout (and see recent paper on hexagonal knowledge structure)
- Perturbation

Ciresan et al. 2012

- DNN = Deep Neural Network
- MCDNN = Multi-column DNN

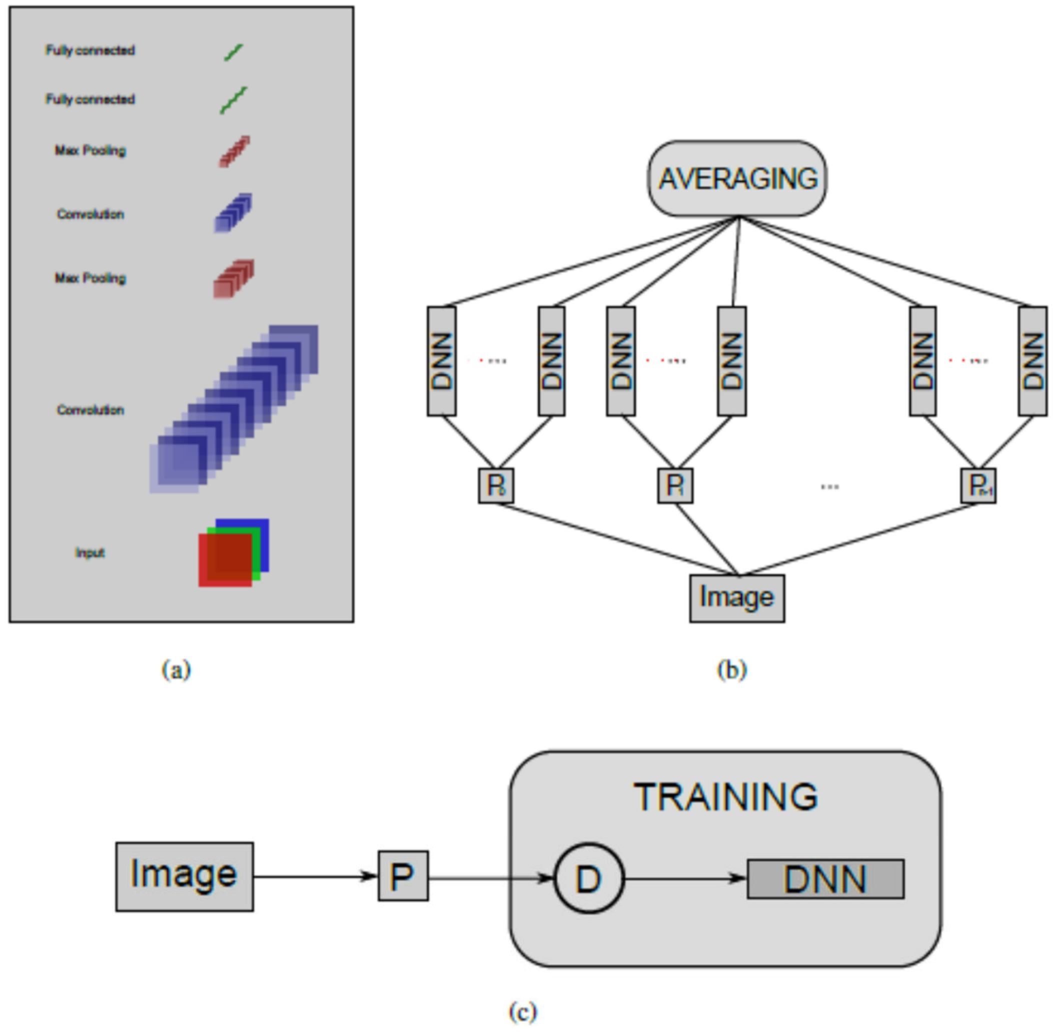
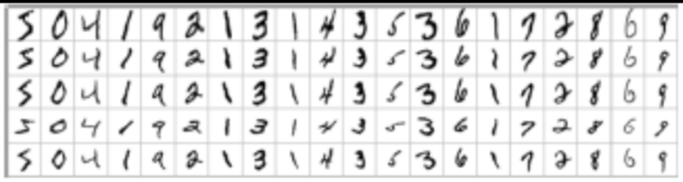
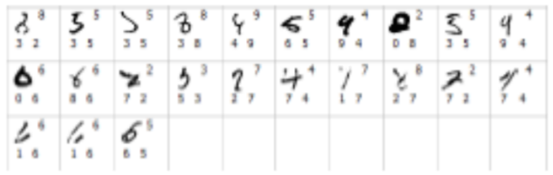


Figure 1: (a) DNN architecture. (b) MCDNN architecture. The input image can be preprocessed by $P_0 - P_{n-1}$ blocks. An arbitrary number of columns can be trained on inputs preprocessed in different ways. The final predictions are obtained by averaging individual predictions of each DNN. (c) Training a DNN. The dataset is preprocessed before training, then, at the beginning of every epoch, the images are distorted (D block). See text for more explanations.

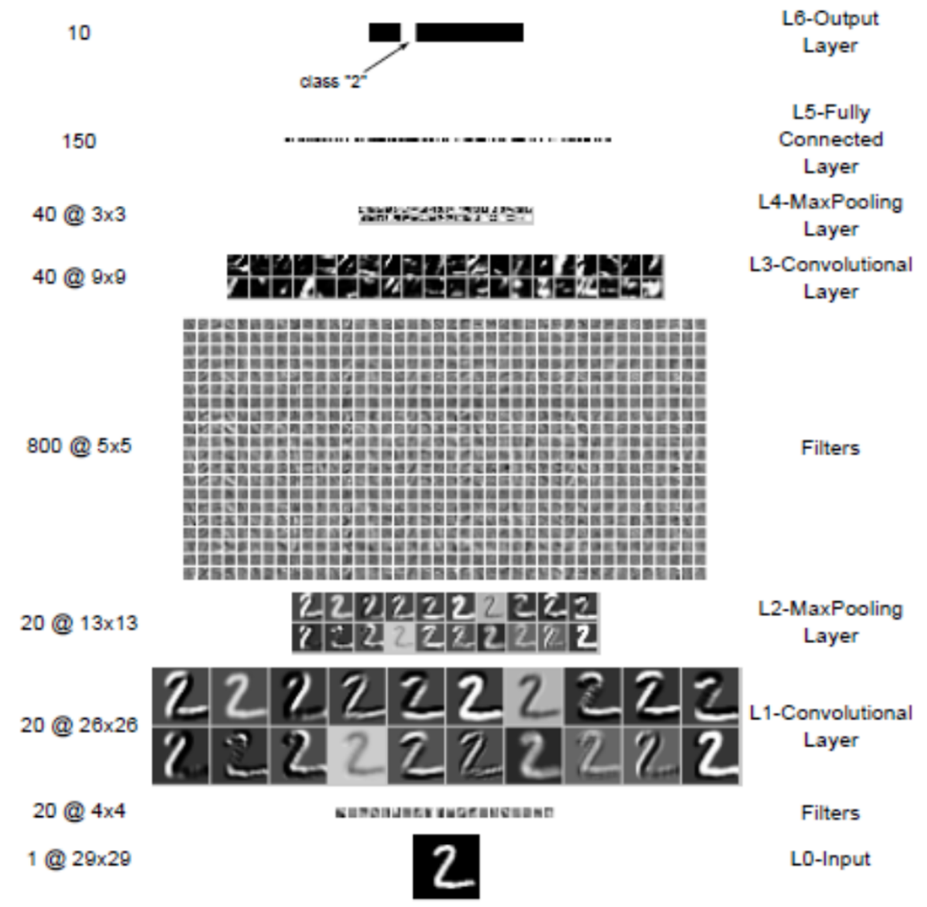
Ciresan et al. 2012



(a)



(b)



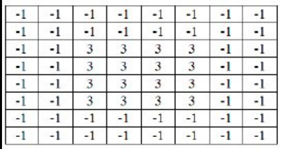
(c)

Figure 2: (a) Handwritten digits from the training set (top row) and their distorted versions after each epoch (second to fifth row). (b) The 23 errors of the MCDNN, with correct label (up right) and first and second best predictions (down left and right). (c) DNN architecture for MNIST. Output layer not drawn to scale; weights of fully connected layers not displayed.

Convolution

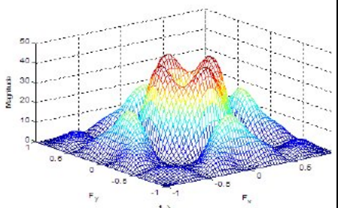
- Convolution basically means filtering. In the case of images, it is filtering with a 2D filter
- Example:

*



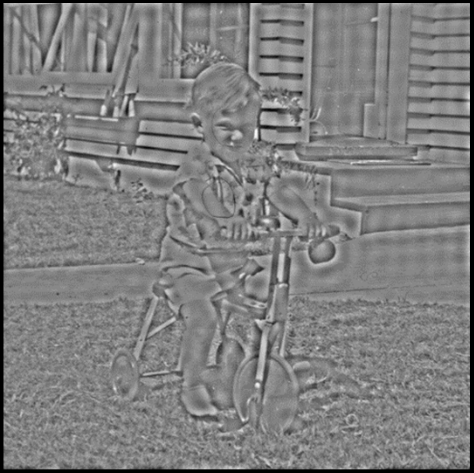
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	3	3	3	3	-1	-1
-1	-1	3	3	3	3	-1	-1
-1	-1	3	3	3	3	-1	-1
-1	-1	3	3	3	3	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

a)



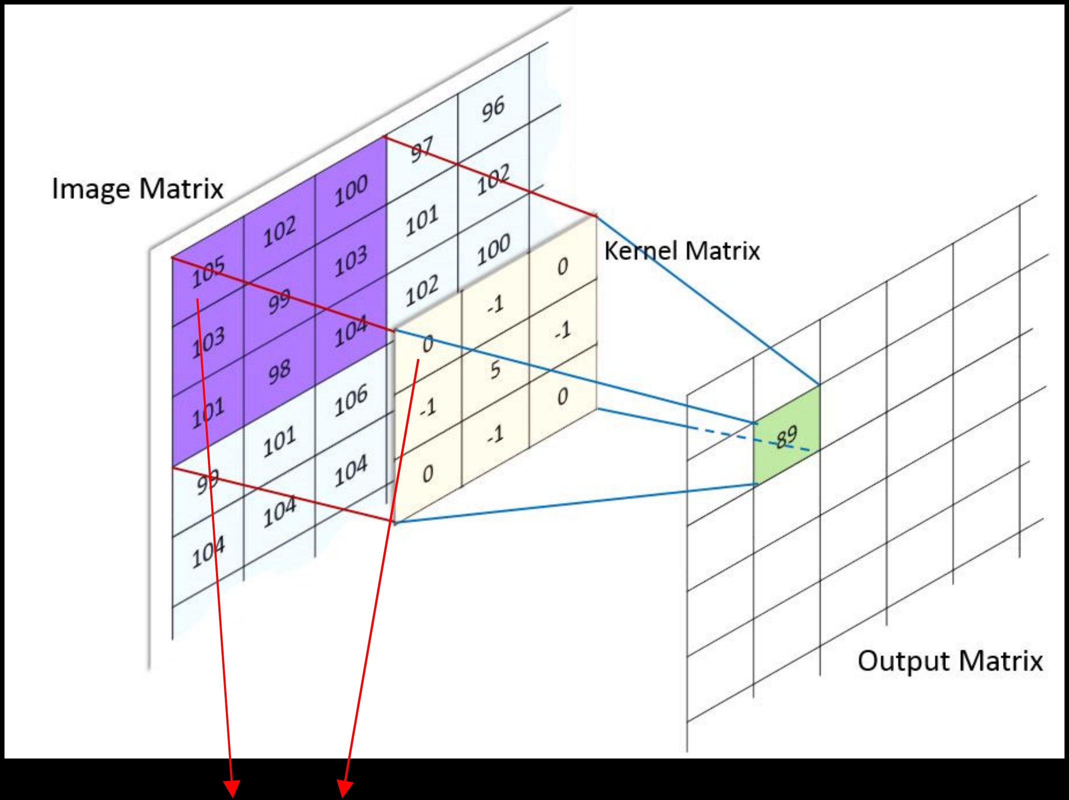
b)

=

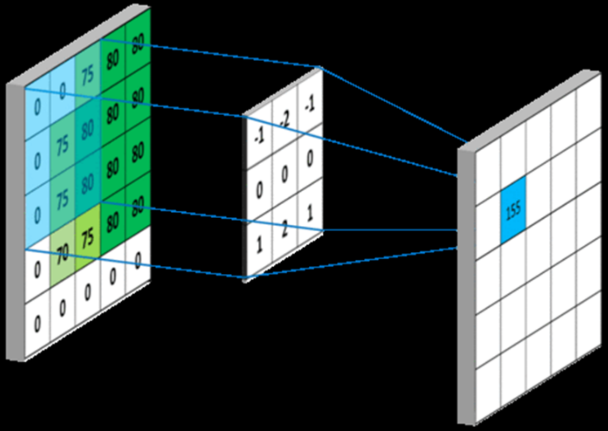


Convolutional Neural networks

- What is a convolution?
- Start with a 2D matrix of image intensities – each pixel is represented as a value (brighter = larger number)
- Use a "convolution kernel", which is multiplied element-by-element with the image pixel values, and these products are summed to generate the final convolution value
- Move the kernel to all possible locations in the image. The resulting image is the "convolved image"



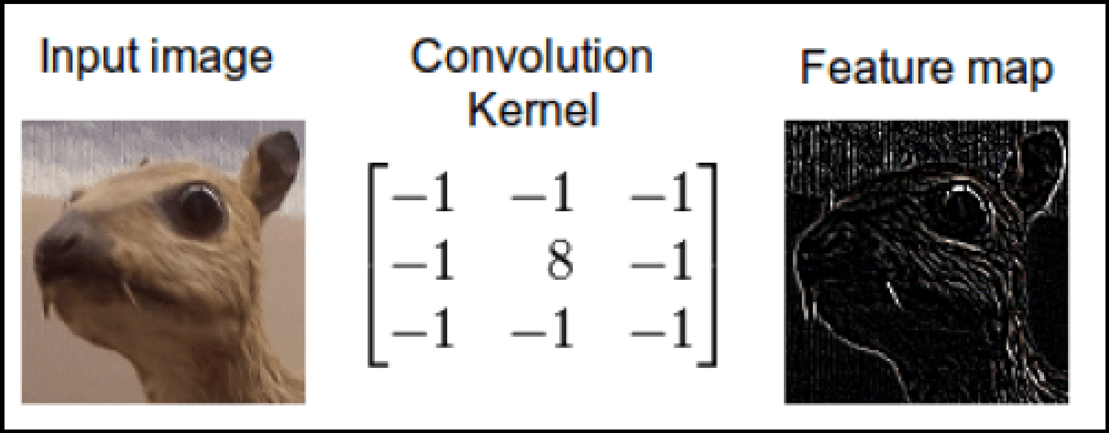
89 = 105*0 + 102*(-1) + 100 * 0 +…



http://machinelearningguru.com/computer_vision/basics/convolution/image_convolution_1.html
https://mlnotebook.github.io/post/CNN1/

Convolutional Neural networks

- The convolution kernels are like filters. The resulting convolved image will be bright where the local pattern matches the kernel pattern and dark otherwise.
- For example, the kernel here will identify edges
- The human visual system seems to first filter the incoming visual information to find edges of various orientations (more on that later)

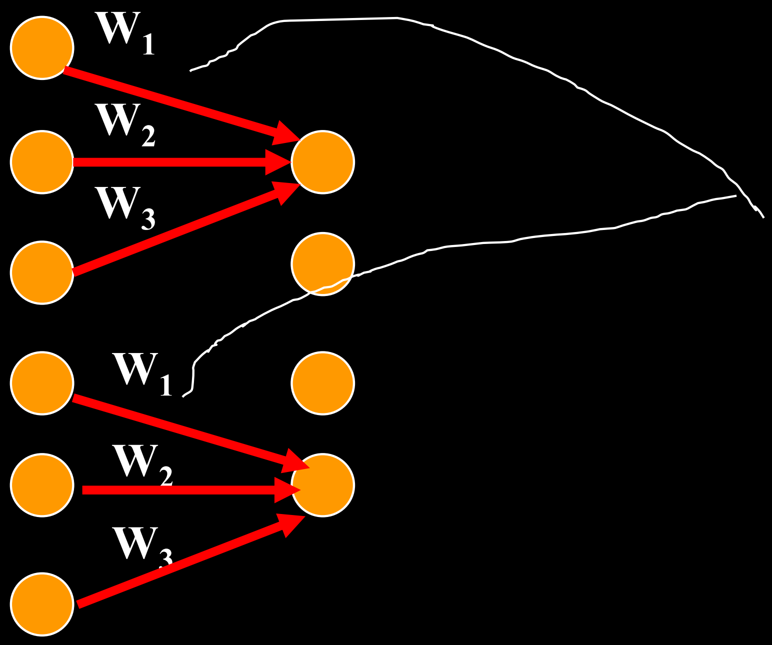


$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

http://timdettmers.com/2015/03/26/convolution-deep-learning/
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolutional Neural networks

- A good set of convolution kernels will extract the most useful information from the image. But how do we know what the best kernels are?
- Long ago, computer vision relied on hand-tuning the filters, e.g. using Gabor filters.
- A Convolutional Neural Network trains the kernels as part of learning!
- But how? Answer: shared weights with backpropagation



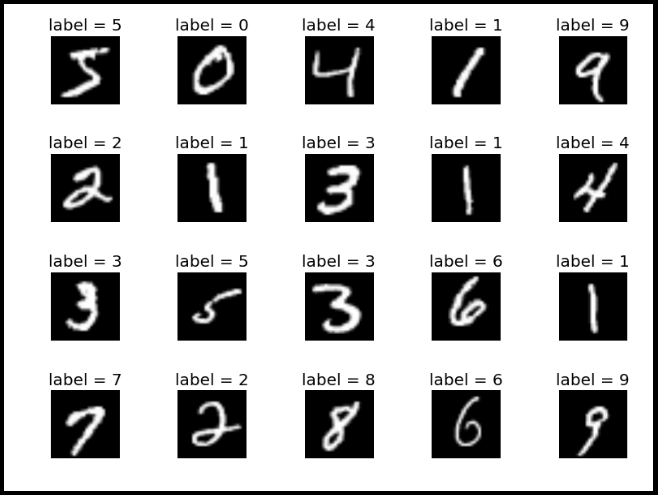
Weights are the same, so the neural network basically does a convolution on the image

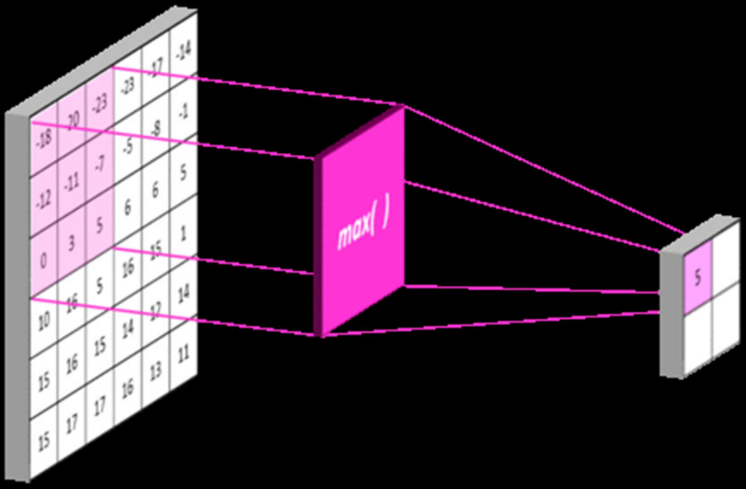
Backpropagation trains the weights to extract the most useful information to do the classification, with ReLU signal function

http://timdettmers.com/2015/03/26/convolution-deep-learning/

Max Pooling

- Problem: Sometimes the visual features are slightly shifted with respect to each other
- Solution: After convolution, take the maximum value in a local neighborhood



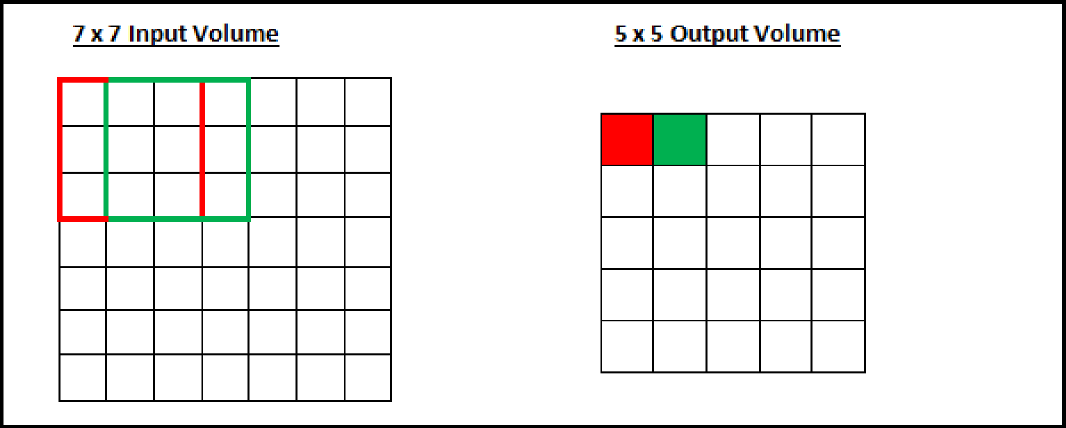


https://mlnotebook.github.io/post/CNN1/

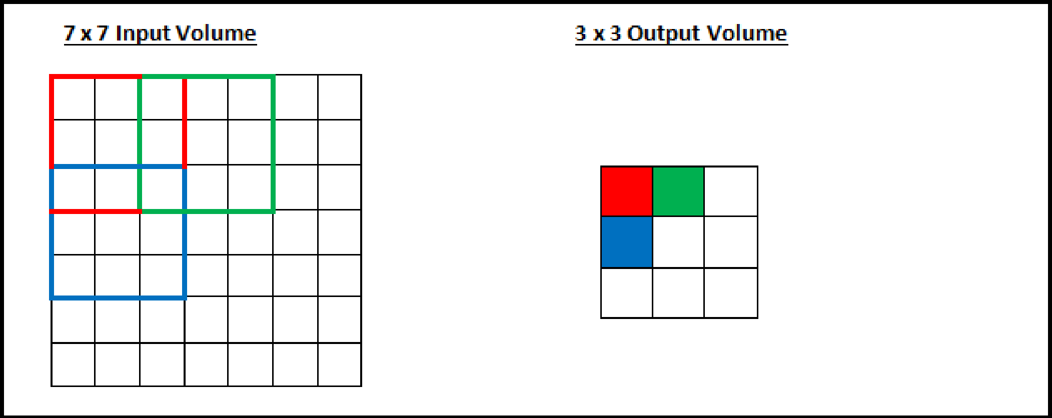
Max Pooling - Stride

- When doing max pooling, we have to choose how far we think the image features might randomly shift
- So how far do we shift the neighborhood at each step? That is the stride
- More feature movement → larger stride

Stride = 1



Stride = 2



https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Dropout

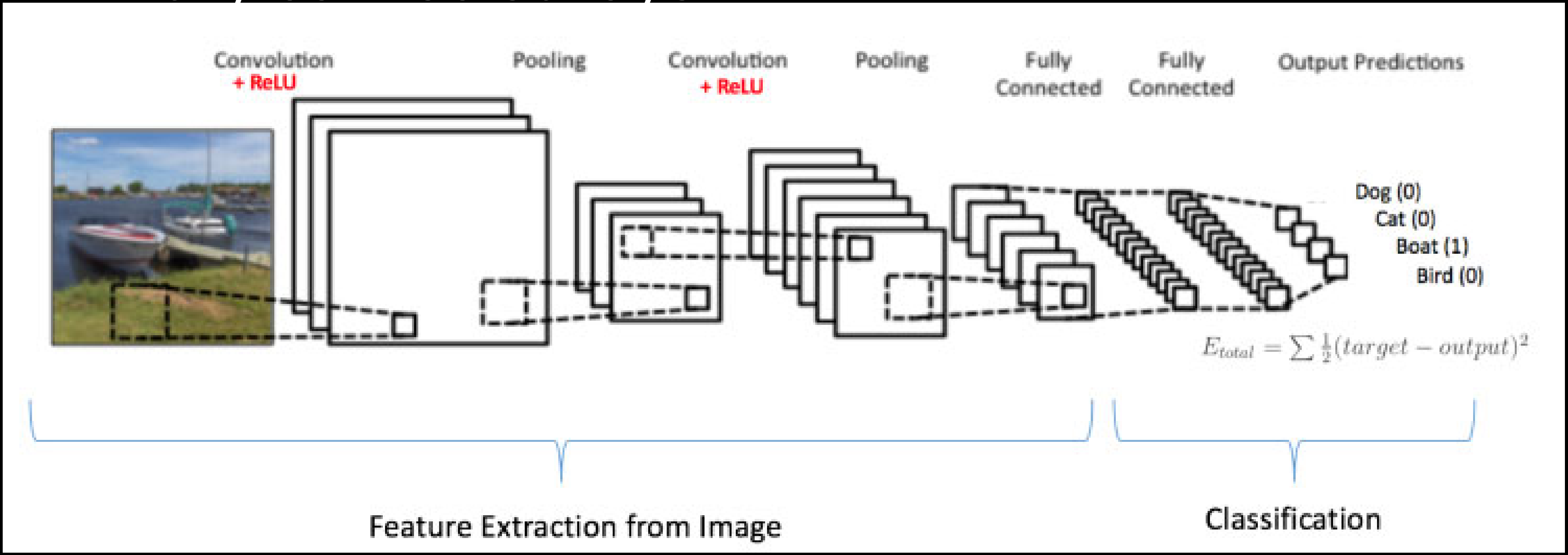
- Overfitting can be a problem, and one way to minimize it is to add noise to the training data
- The simplest way to introduce noise in training is to randomly set a fraction of a layer’s pixel values to zero.
- The probability of dropout in a layer is between 0 and 1, e.g. 0.2

Perturbation

- Another way to deal with overfitting is to augment the training set with perturbed exemplars
- Basically take an input image and shift it, stretch it, shear it, or otherwise vary it to capture some of the normal variation of input images expected in the test set

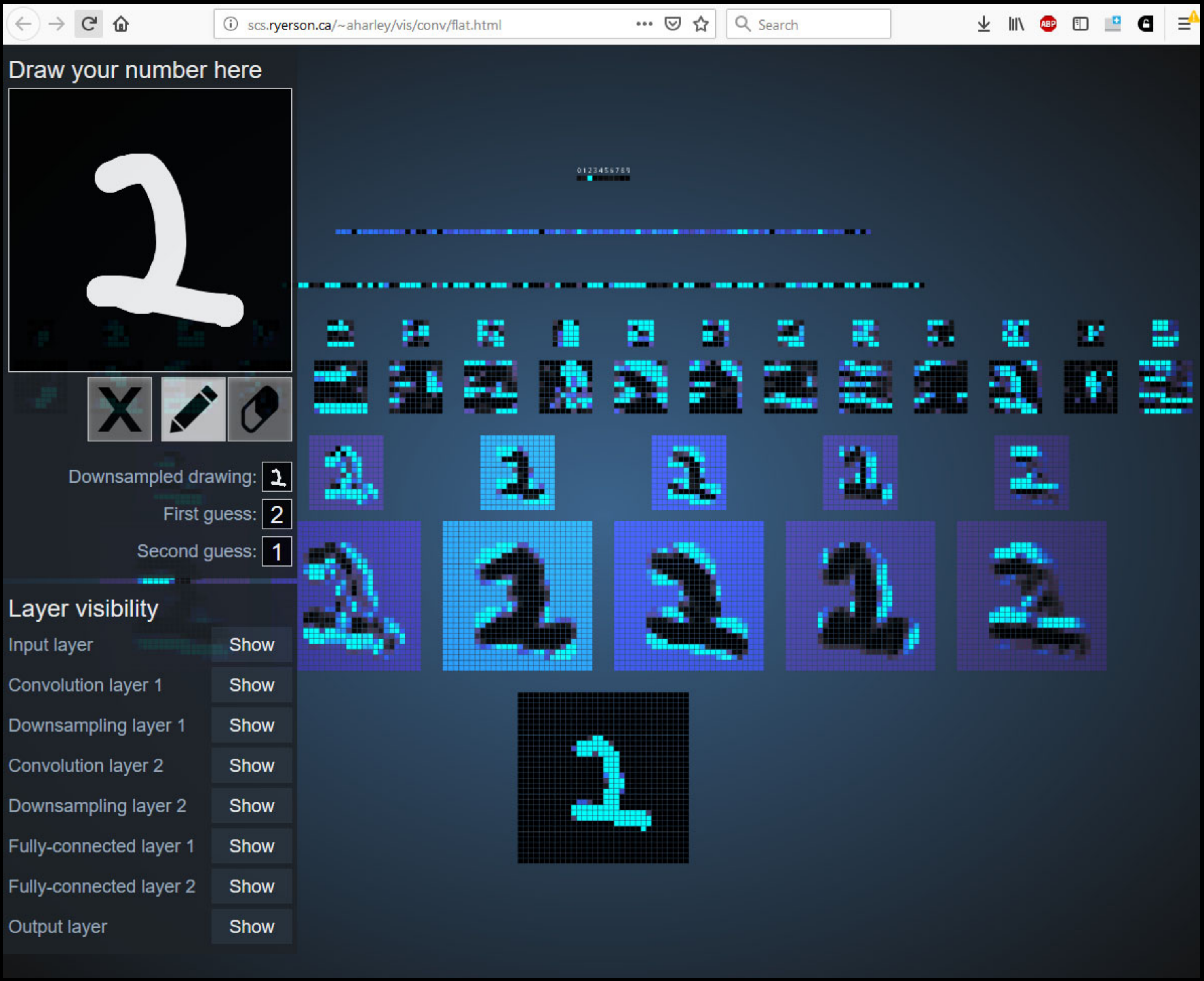
ConvNet model

- Typical image recognition model
- Convolution with ReLU
- Max Pooling, more convolution, more pooling, dropout…
- Fully connected layer



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

MNIST demo



scs.ryerson.ca/~aharley/vis/conv/flat.html

Draw your number here

Downsampled drawing: 2

First guess: 2

Second guess: 1

Layer visibility

Input layer	Show
Convolution layer 1	Show
Downsampling layer 1	Show
Convolution layer 2	Show
Downsampling layer 2	Show
Fully-connected layer 1	Show
Fully-connected layer 2	Show
Output layer	Show

http://scs.ryerson.ca/~aharley/vis/conv/flat.html

Tensorflow.org

- Tensorflow is an open source deep learning package made by Google
- Typically run as python library
- Can be run on colab.research.google.com
- Compatible with GPUs and TPUs
- Lots of powerful deep learning tools

Keras

- Keras (keras.io) – a high level neural networks API.
- It’s not a neural network simulator – instead, it’s a “front end” high level interface, making it easier to build deep learning models. It runs on Tensorflow and (obsolescent) Theano, among other libraries.

Keras - examples

- The “Sequential” model

```
from keras.models import Sequential
model=Sequential()
```

- Start with a model object, then add layers to it starting with the input layer (100 input units, 64 hidden, 10 output):

```
from keras.layers import Dense
model.add(Dense(units=64,
      activation='relu', input_dim=100))
model.add(Dense(units=10,
      activation='softmax'))
```

Keras - examples

- Once the model is specified, “compile” it by specifying the loss function, optimizer, and evaluation metric

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

- Train the model:

```
model.fit(x_train, y_train, epochs=5,
batch_size=32)
```

- Test the model:

```
loss_and_metrics = model.evaluate(x_test,
y_test, batch_size=128)
```

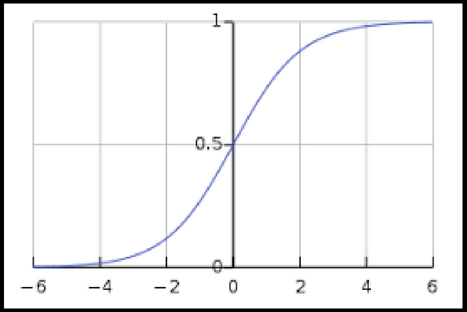
Tensorflow – MNIST with Keras API

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/_index.ipynb

How did that work?

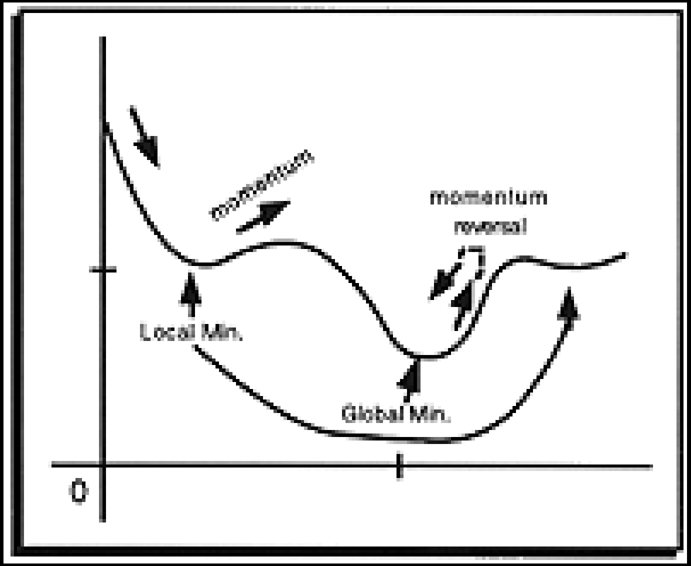
- What is softmax?
  - Activation function that restricts outputs to the range of [0, 1], and normalizes such that the outputs sum to 1
  - $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$



  - Basically run all the outputs through a logistic function, then normalize by dividing by sum, so sum is 1

How did that work?

- What is the "Adam" optimizer (Kingma et al., 2015)?
- First, understand momentum. If we ONLY descend the gradient of the loss function, then we might get stuck in a "local minimum", so we think the weights have converged, but there is a better solution if we keep going.
- Momentum allows the weight update rule to keep the weight change moving in the same direction for a while, even when the gradient is zero
- This allows the weight update rule to keep going through the local minimum to find a global minimum
- Also makes gradient descent smoother, faster



$$\Delta W_{t+1} = \underbrace{\gamma \Delta W_t}_{\gamma=\text{momentum}} + \underbrace{(1-\gamma) * \alpha * error * f'(net) * y}_{\text{Current error update}}$$

Adam optimizer (Kingma et al., 2015)

- Second, understand stochastic loss functions
- Problem: Sometimes the loss function varies stochastically from one evaluation to the next, perhaps because of intrinsic noise in the data, or because different data are used to evaluate it.
- Solution: The larger the noise in the loss function, the larger the steps we need to take in order to avoid getting stuck in a temporary local minimum due to noise
- Adam addresses both of these problems, including momentum AND scaling the learning rate by the variance of the gradient.