
# Deep Learning for Environmental Sciences

## Perspective, Methods, Applications

Nathaniel Newlands and Etienne Lord
*Science and Technology, AAFC/Government of Canada*

August 16-17, 2019
Sasana Kijang, Kuala Lumpur, Malaysia

# About Nathaniel

Sustainable Ecosystems (Food, Water, Energy)

Predictive Analytics, Data Science

- Research Scientist
  Summerland (British Columbia
  Agriculture and Agri-Food Canada)
- Adjunct Assoc. Professor 2014-
  (Geography, University of Victoria/UVic)
- Adjunct Professor 2008-2016
  (Statistics, University of British Columbia/UBC)

# About Etienne

Digital Agronomy, Bioinformatics, Data Science

- Research Scientist
  Saint-Jean-sur-Richelieu (Quebec),
  Agriculture and Agri-Food Canada

# Overview

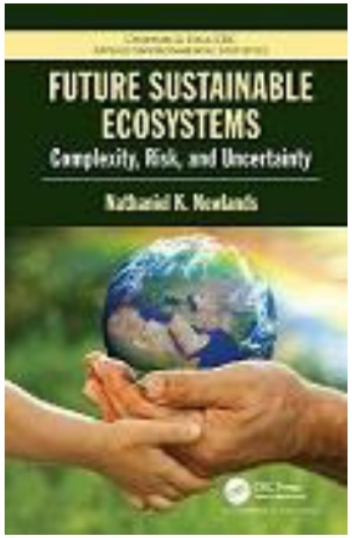
**Friday 16 August**

| Time | Description | Duration |
|---|---|---|
| 9:00 - 10:00 am | **Introduction: Perspective, Background, Model Optimization** | 1h |
| 10:00 - 10:15 am | *Health break* | 15 min |
| 10:15 - 10:45 am | **The DL community: Overview of software tools, coding libraries, databases** | 30 min |
| 10:45 - 11 pm | Practical - Installation of software packages and libraries (R/R Studio and Anaconda Python) | 15 min |
| 11 - 12:30 pm | **Image Analysis: Convolutional neural networks (CNN)** | 1.5h |
| 12:30 - 1:30 pm | *Lunch* | 1h |
| 1:30 - 2:30 pm | Practical examples - Classification of images (CNN): fruit and landcover/landuse satellite remote-sensing imagery | 1h |
| 2:30 - 3:30pm | **Environmental time series/regression: Recursive neural networks (RNN), Long-Short Term Memory (LSTM), Deep belief networks (DBN), Reinforcement Q-learning (DQN)** | 1 h |
| 3:30 - 3:45pm | *Health Break* | 15 min |
| 3:45 - 4:30pm | Practical examples - RNN, DBN, DQN, pretraining, hyperparameter search, wine-quality, crop yield prediction | 45 min |

**Saturday 17 August**

| Time | Description | Duration |
|---|---|---|
| 9:00 - 10:30 pm | Practical examples continued... | 1.5h |
| 10:30 - 10:45 am | Health break | 15 min |
| 10:45 -12:30 am | ***Evaluating models (sensitivity and validation):Bias-variance tradeoff, performance metrics, hyperparameter search*** | 1h 45min |

# Bridging the gap: Improving our dialogue on sustainability and climate change




*"Future Sustainable Ecosystems: Complexity, Risk, Uncertainty" (Taylor and Francis)*

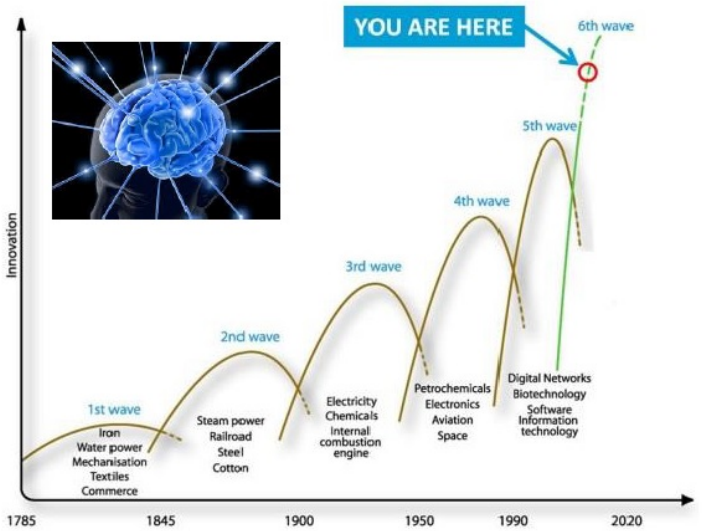
Polarized perspectives, cultural cognition

Integrated risk concepts, models, frameworks, urgent need for broader open dialogue

Evidence-based decision making, knowledge integration

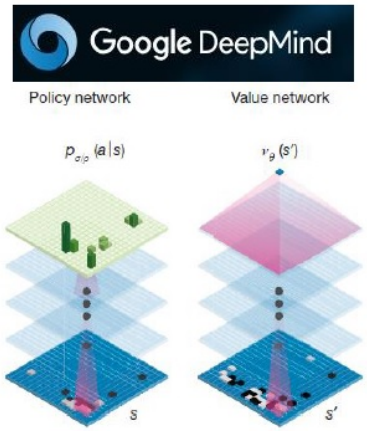
Rapid learning: human and artificial/machine intelligence (monitoring, assessment, forecasting)

# Harnessing the predictive power of "Big data"




1990s, knowledge doubled every century
1940s, doubled every 25 yrs
Currently doubling every 13 months
Soon, every 12 hrs?

*"Transition from linear to exponential growth of human knowledge has taken place"*

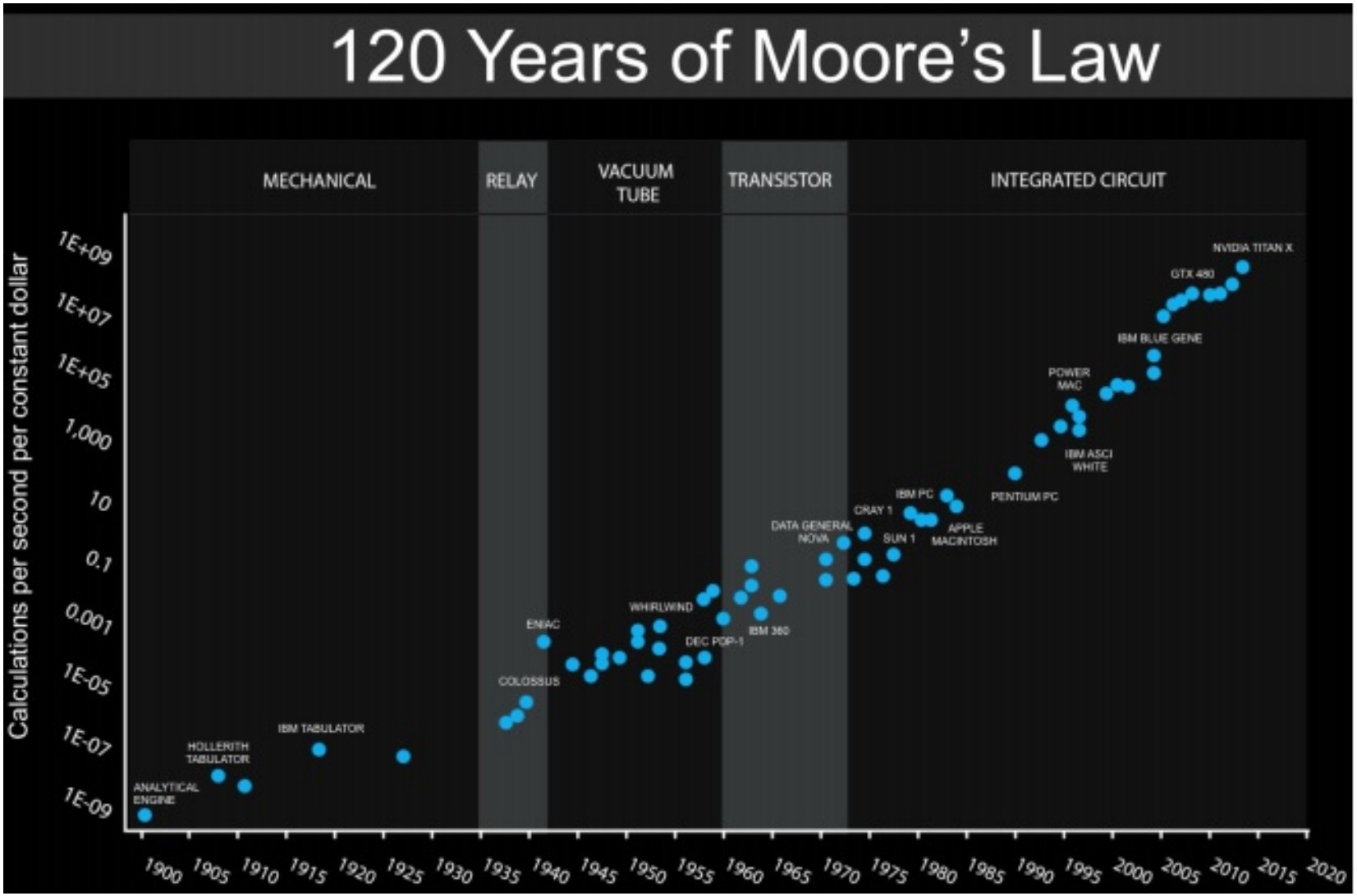



AlphaGo: value/policy networks

IBM Watson: Deep-learning cloud-based/cognitive apps/APIs

Open, geospatial data, infrastructure

Precision agriculture (automated data collection, analytics)

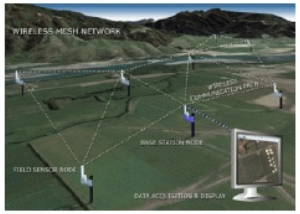



[https://en.wikipedia.org/wiki/Moore%27s_law]

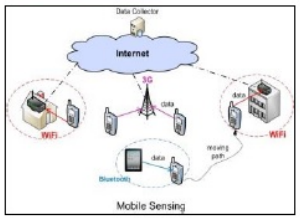
- Moore's law - number of transistors in a dense integrated circuit doubles about every two years

# Harnessing the predictive power of “Big data”

Passive/active hyperspectral satellite imagery




*In-situ* networks (fixed)




*In-situ* networks (mobile sensing)

Airborne imagery (drone scanning

*Ex-situ* networks (Seed/gene/tissue culture banks, breeding, botanical gardens, zoos)

- Open government
- Private-public
- Citizen science (participatory sensing, social computing)

# AI - statistical perspective

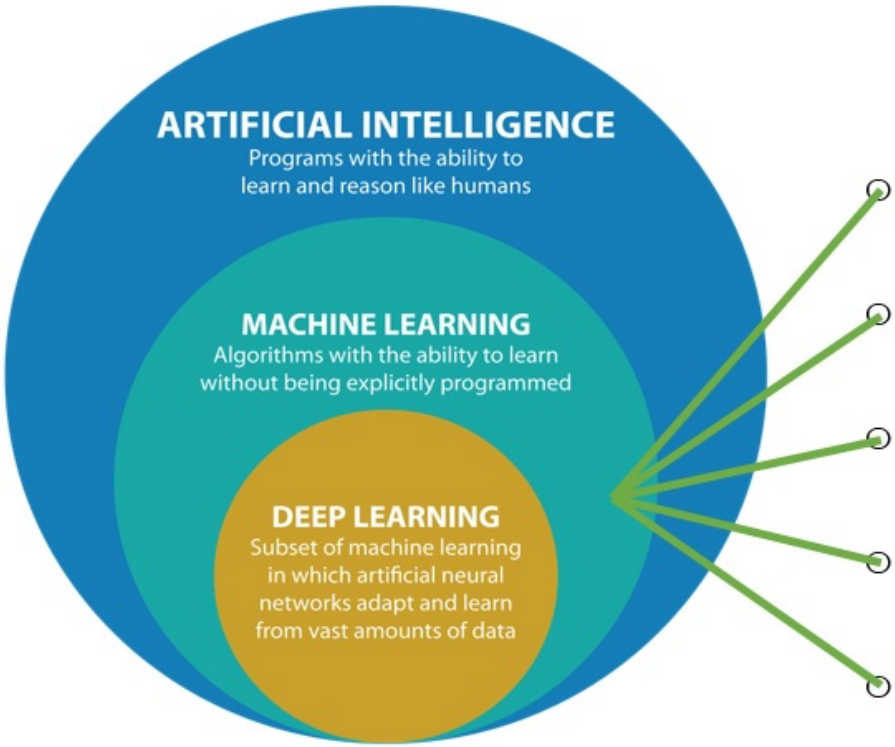
## A spectrum of machine learning tasks

### Typical Statistics----------Artificial Intelligence

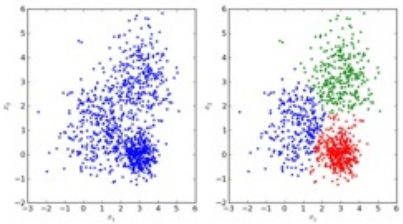
- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

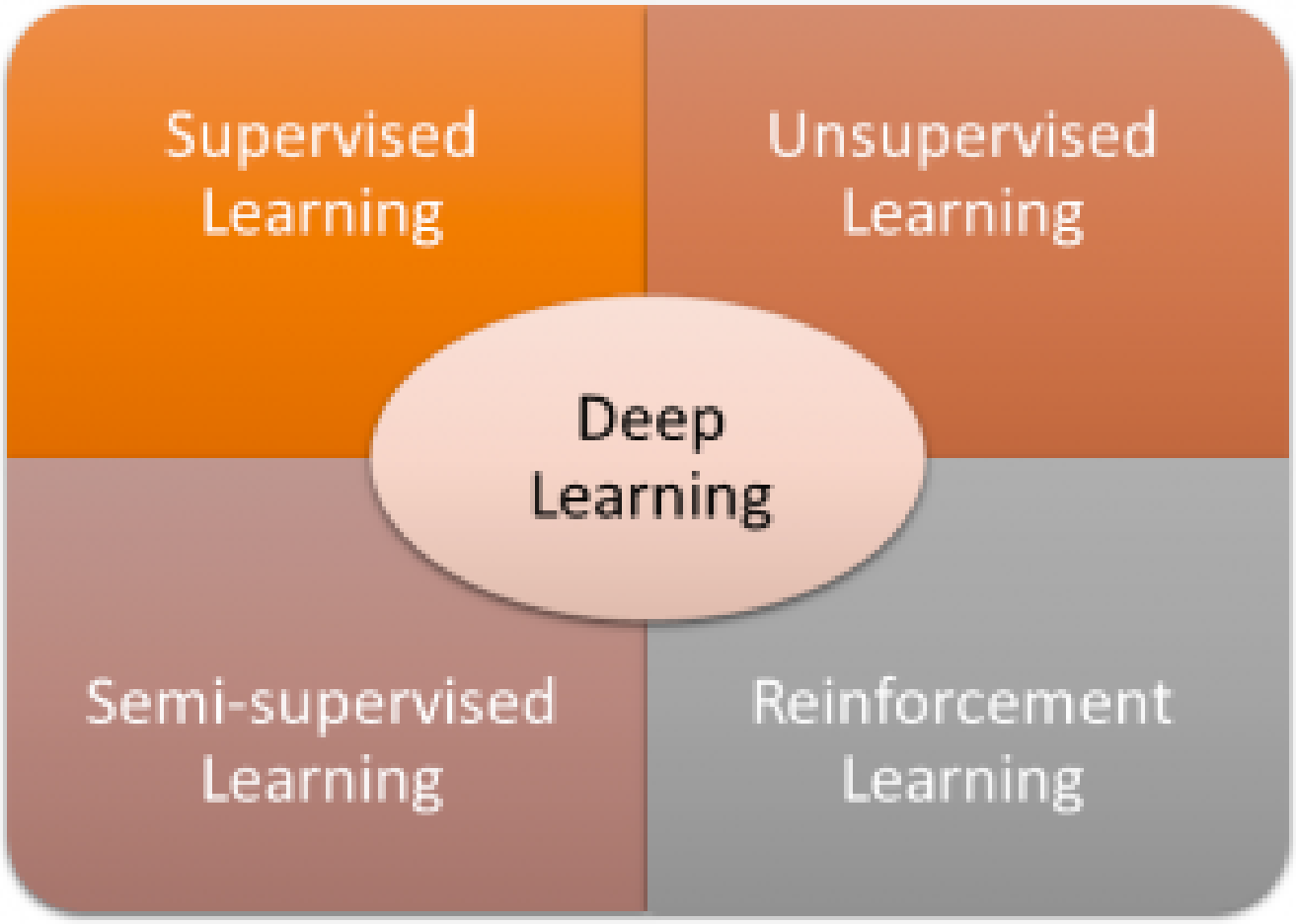
- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure so that it can be learned.

# Domains of AI




- Classification
- Regression
- Clustering
- Generative models
- Reinforcement learning

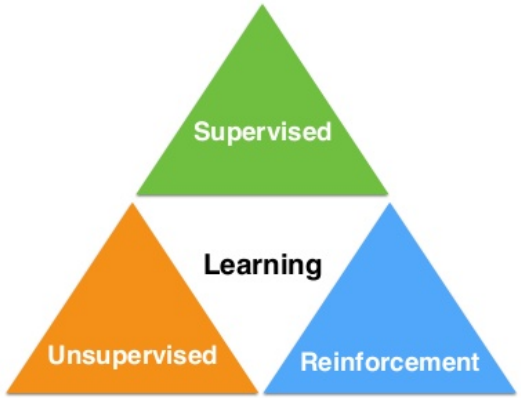



Supervised Learning
Unsupervised Learning
Deep Learning
Semi-supervised Learning
Reinforcement Learning

# Learning Algorithms

| Supervised | Unsupervised | Reinforcement |
| --- | --- | --- |
| Learning known patterns | Learning unknown patterns | Generating data<br>Learning patterns |

- Labeled data
- Direct feedback
- Predict outcome/future

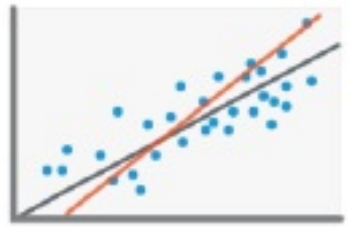



- No labels
- No feedback
- "Find hidden structure"

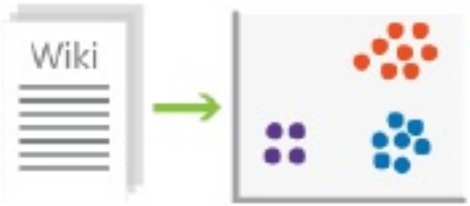
- Decision process
- Reward system
- Learn series of actions

# Descriptive vs. Predictive


Classification
(supervised – predictive)



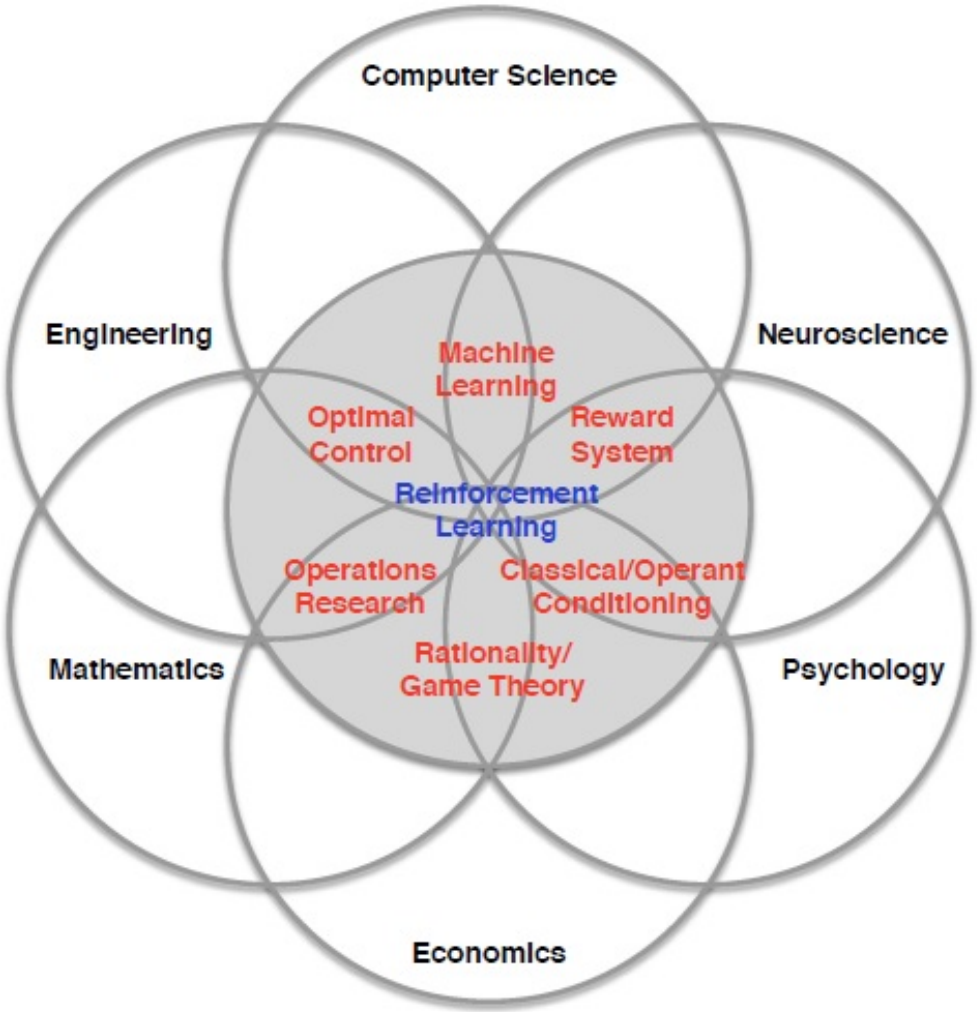
Regression
(supervised – predictive)




Clustering
(unsupervised – descriptive)


Anomaly Detection
(unsupervised– descriptive)

# Multidisciplinary - Reinforcement Learning



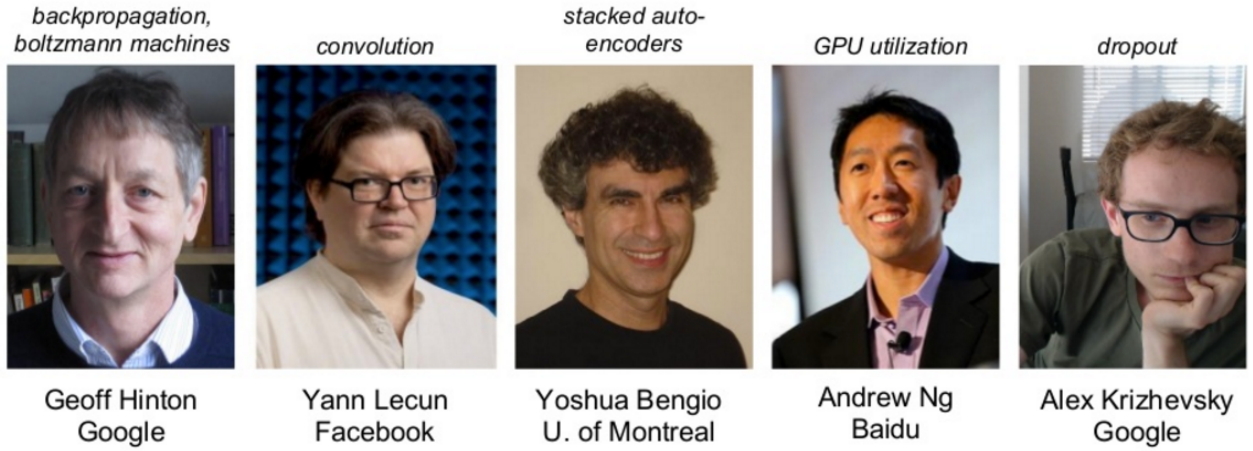
# Deep Learning: Benefits and Drawbacks

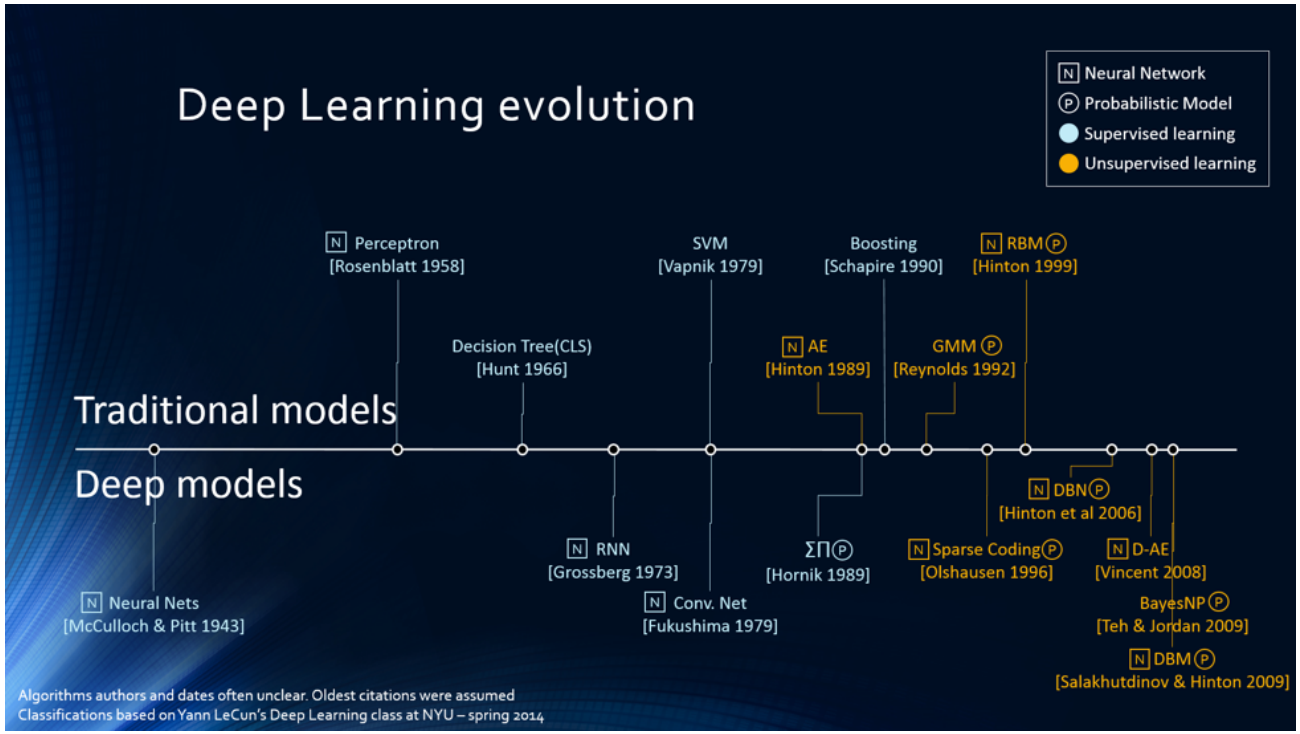

- hierarchy of multiple layers that mimic the brain
- learns, becomes more accurate with more data
- training data is key to success of deep learning
- supervised learning dominates in real world applications
- training process for deep learning can take weeks to run

# Deep learning (Use-cases)

unstructured data (images, sound, video, text)
structured data (sensor signal, physical, biological, multivariate)

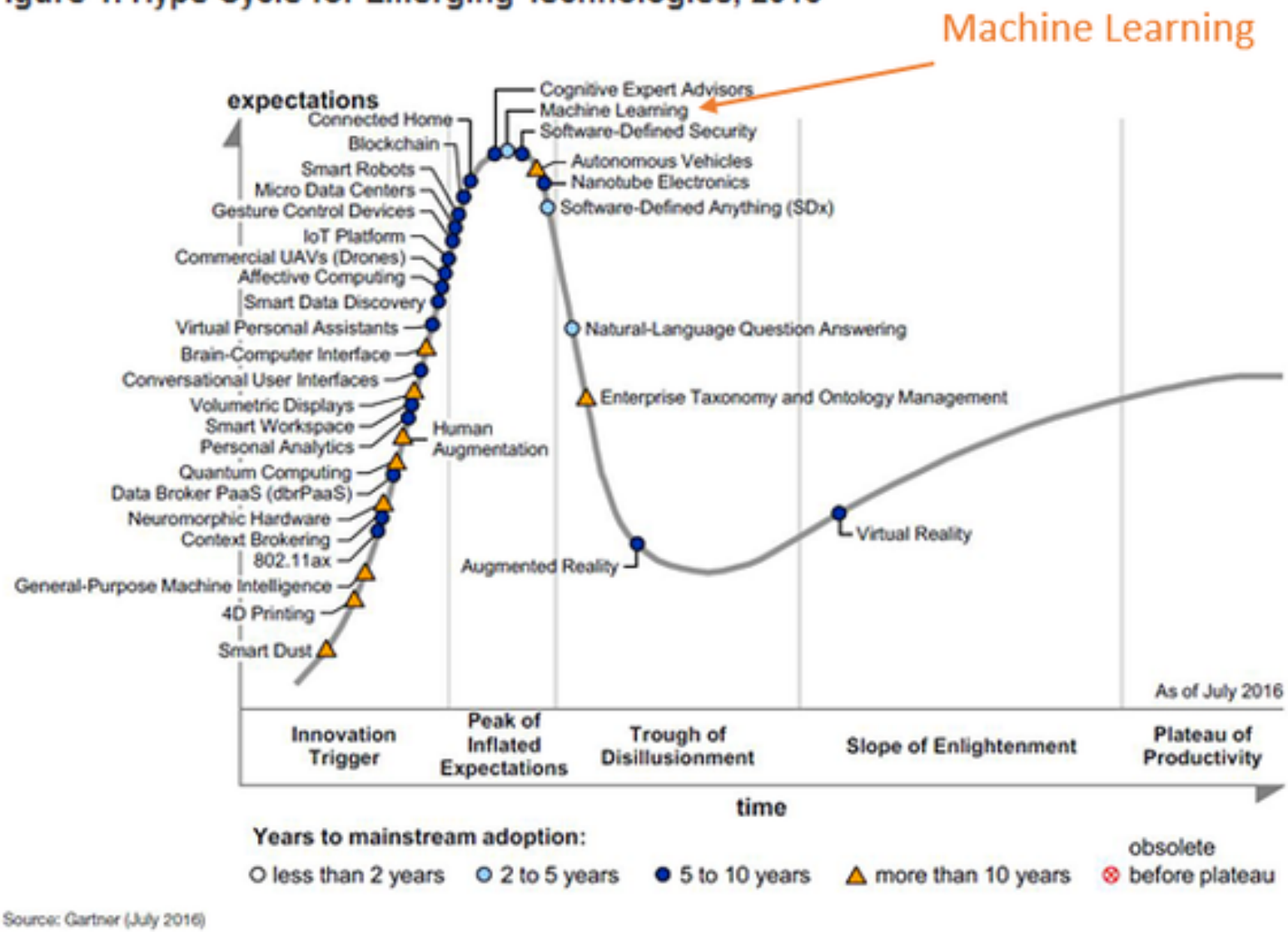
| General use case | Industry |
|---|---|
| **Sound** | |
| Voice recognition | UX/UI, Automotive, Security, IoT |
| Voice search | Handset maker, Telecoms |
| Sentiment analysis | CRM |
| Flaw detection (engine noise) | Automotive, Aviation |
| Fraud detection (latent audio artifacts) | Finance, Credit Cards |
| | |
| **Time Series** | |
| Log analysis/Risk detection | Data centers, Security, Finance |
| Enterprise resource planning | Manufacturing, Auto., Supply chain |
| Predictive analysis using sensor data | IoT, Smart home, Hardware manufact. |
| Business and Economic analytics | Finance, Accounting, Government |
| Recommendation engine | E-commerce, Media, Social Networks |
| | |
| **Text** | |
| Sentiment Analysis | CRM, Social media, Reputation mgt. |
| Augmented search, Theme detection | Finance |
| Threat detection | Social media, Govt. |
| Fraud detection | Insurance, Finance |
| | |
| **Image** | |
| Facial recognition | |
| Image search | Social media |
| Machine vision | Automotive, aviation |
| Photo clustering | Telecom, Handset makers |
| | |
| **Video** | |
| Motion detection | Gaming, UX, UI |
| Real-time threat detection | Security, Airports |




2055/06- Deep Boltzmann Machines (DBMs), Restricted Boltzmann Machine (RBM) (Hinton)
2012: AlexNet wins ImageNet, Google Brain Project (16K cores)

## Figure 1. Hype Cycle for Emerging Technologies, 2016




Source: Gartner (July 2016)

# Hype or Reality?

## Hype or Reality?

### Quotes

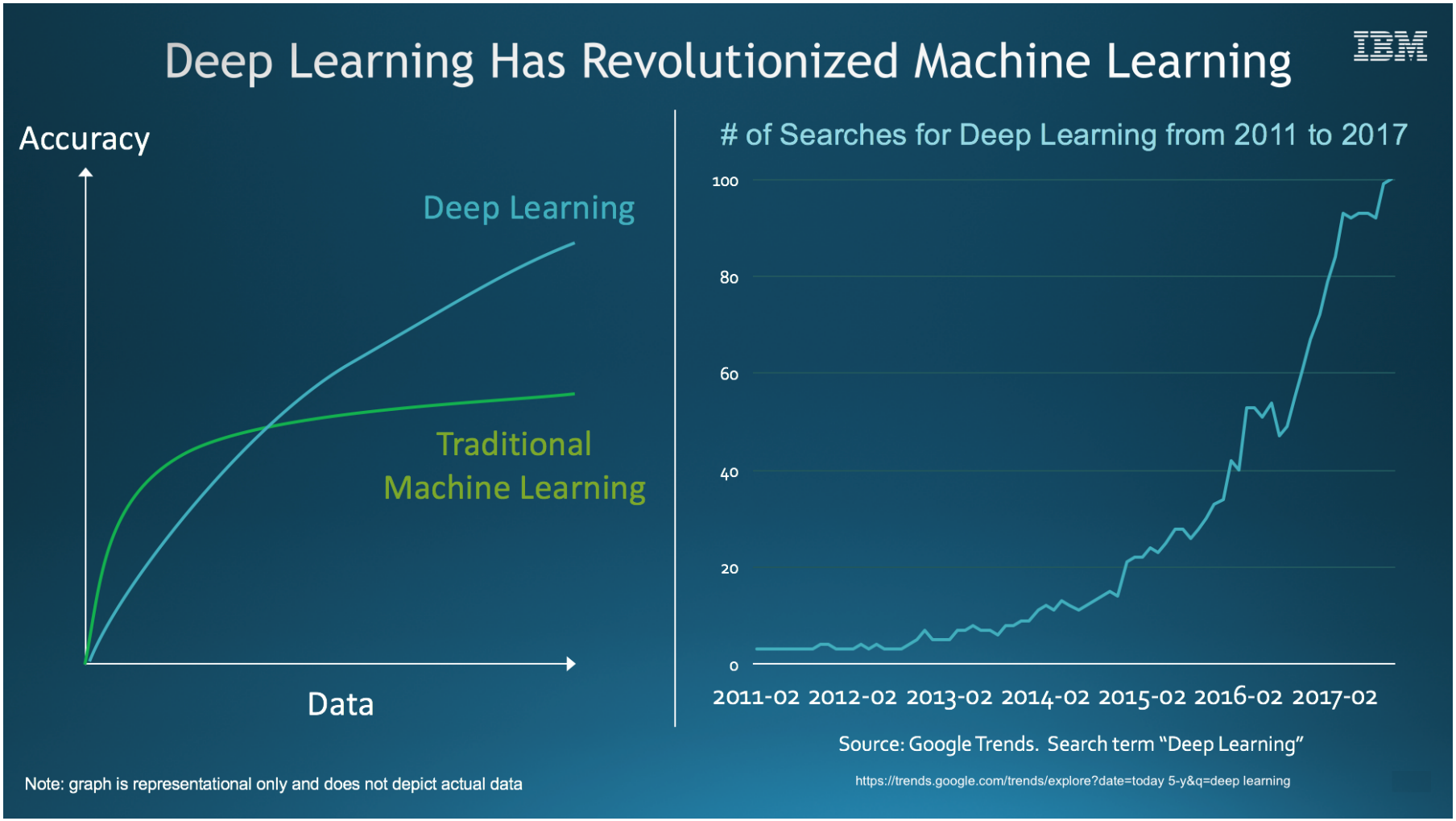
I have worked all my life in Machine Learning, and I've never seen one algorithm knock over benchmarks like Deep Learning
– *Andrew Ng (Stanford & Baidu)*

Deep Learning is an algorithm which has no theoretical limitations of what it can learn; the more data you give and the more computational time you provide, the better it is – *Geoffrey Hinton (Google)*

Human-level artificial intelligence has the potential to help humanity thrive more than any invention that has come before it – *Dileep George (Co-Founder Vicarious)*

For a very long time it will be a complementary tool that human scientists and human experts can use to help them with the things that humans are not naturally good – *Demis Hassabis (Co-Founder* DeepMind)

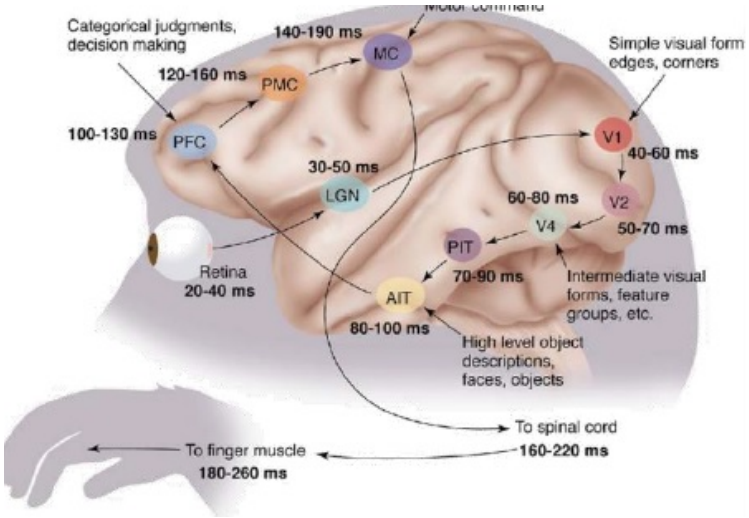
# Perspective



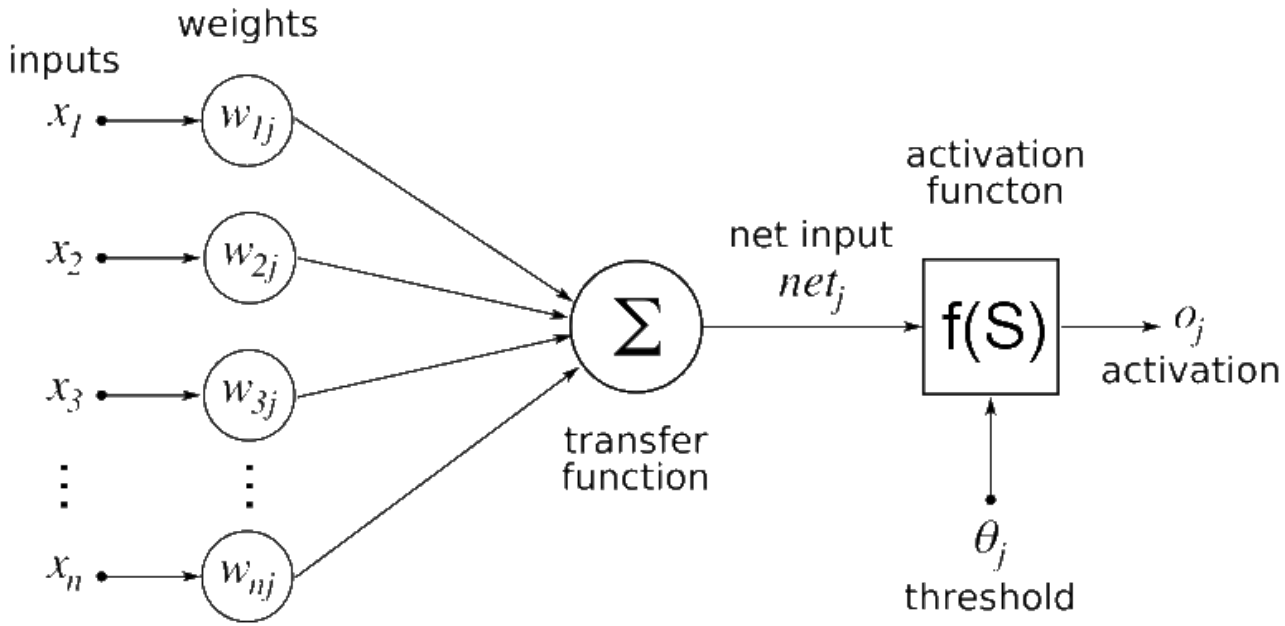
# Hello World Canada: The Rise of AI (Bloomberg)

https://www.bloomberg.com/news/videos/2018-05-23/hello-world-canada-the-rise-of-ai-video

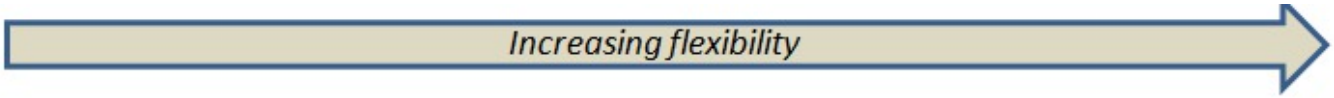
# Inspired by the Brain: Neural Networks




The first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.



# Neural networks - diverse structures




*Feedforward neural networks (FNNs)*

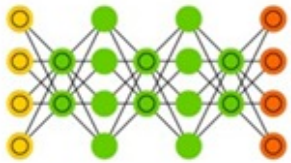
single hidden layer
no feedback, intra-layer links
directed links between layers
supervised/unsupervised learning

*Deep neural, feedforward networks (DNNs, DFFs)*

many hidden layers
undirected links
supervised learning

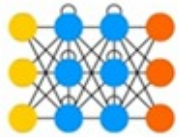
*Deep belief networks (DBNs)*

many hidden layers
undirected links
unsupervised learning



*Recurrent neural networks (RNNs)*

many hidden layers
feedback (memory)
supervised/unsupervised learning

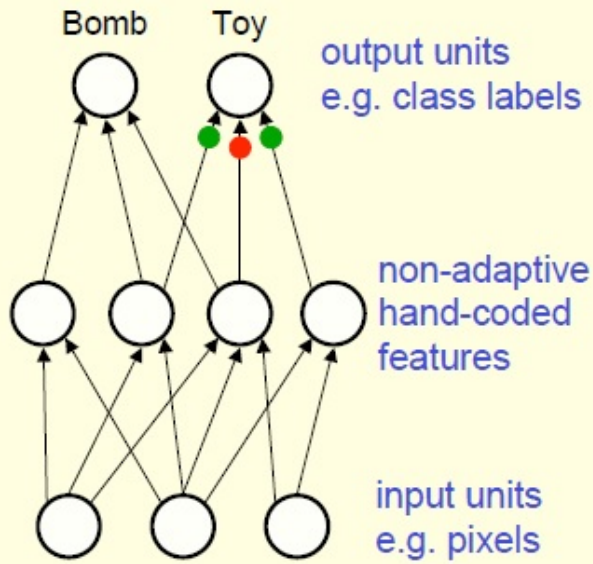


# Perceptron

- linear, binary classifier (eg, Heaviside step function)
- supervised learning (if an input can be represented by a vector of numbers, belongs to some specific class)
- Multi-layer perceptrons (MLPs): arbitrary, nonlinear activations
- MLPs can distinguish classes in data that are not linearly separable

## Historical background:
## First generation neural networks

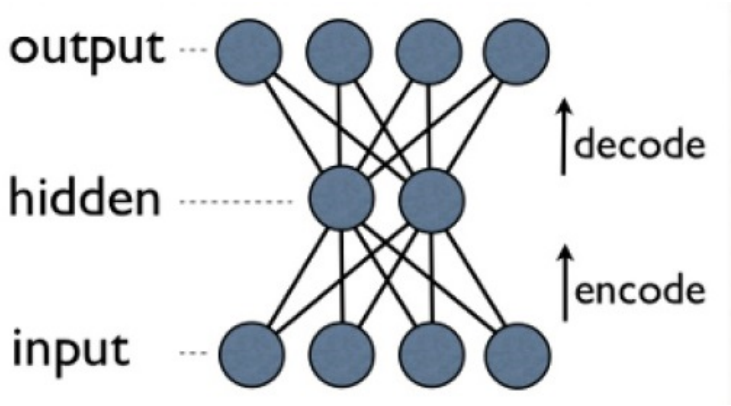
- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.



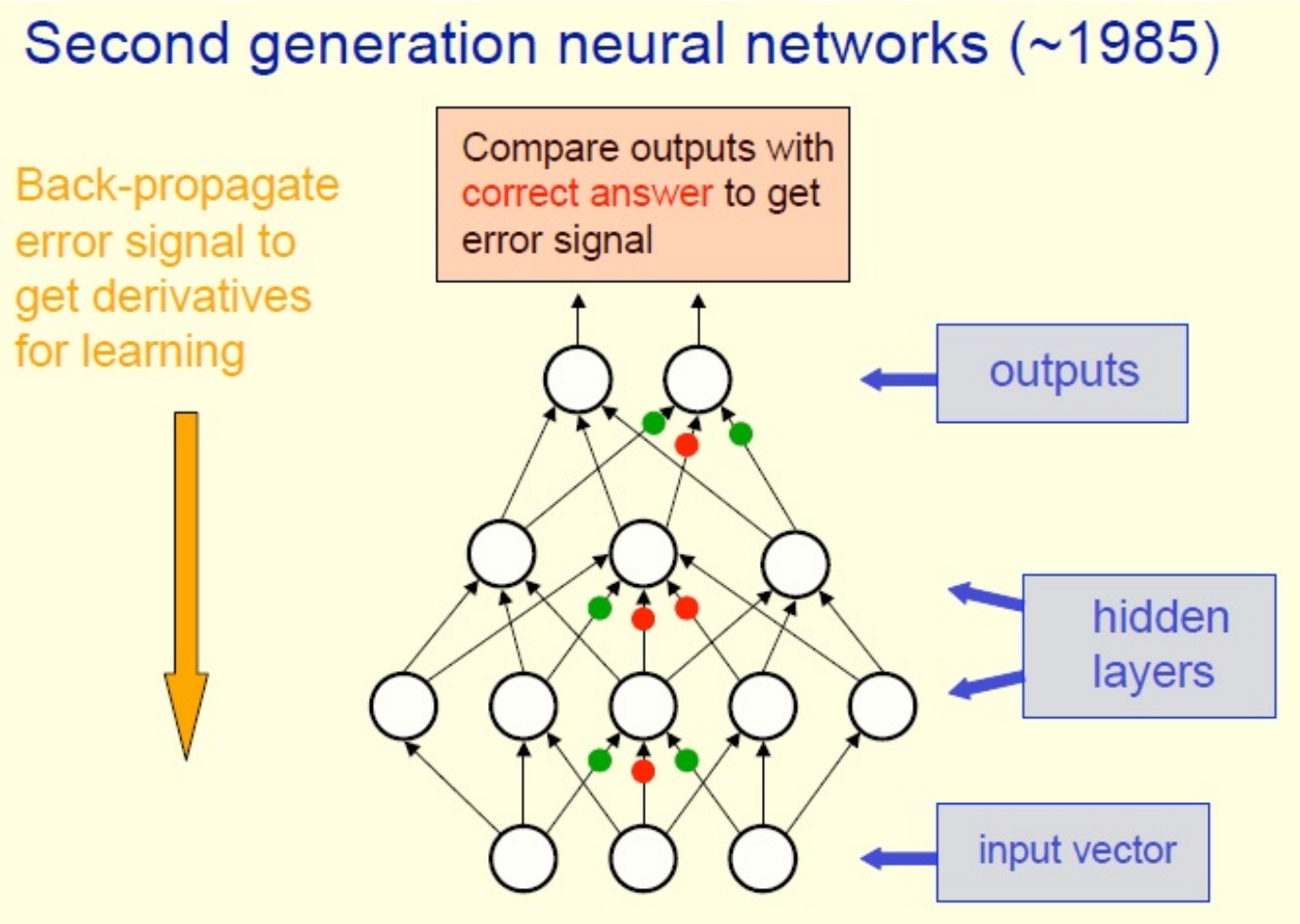

Sketch of a typical perceptron from the 1960's

# Autoencoder

- 3-layer network (input, hidden, output layers)
- learns encoding of data, dimensional reduction
- PCA: linear activations, sigmoidal hidden layer (weights $\neq$ components, nor orthogonal)
- backpropagation: with many hidden layers is slow, poor solutions without pretraining
- sparse autoencoder: hidden units > inputs, but sparsity of active hidden units
- sparsity: thresholds in loss function



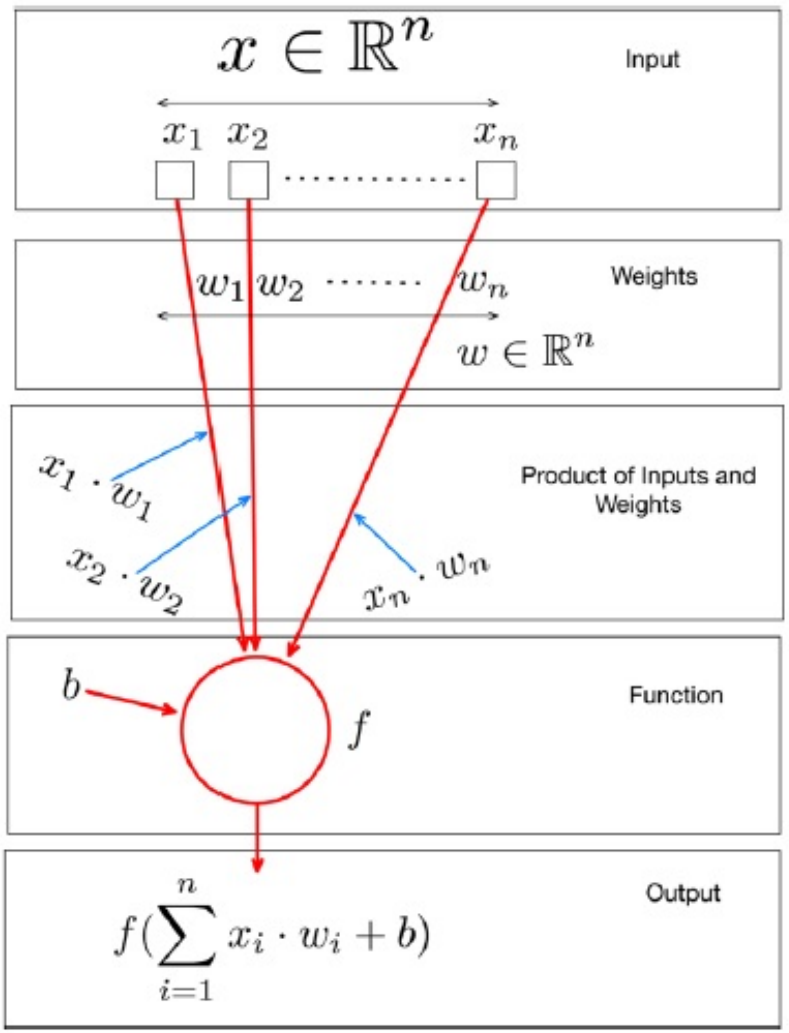
# Artificial Neural Network (ANN)



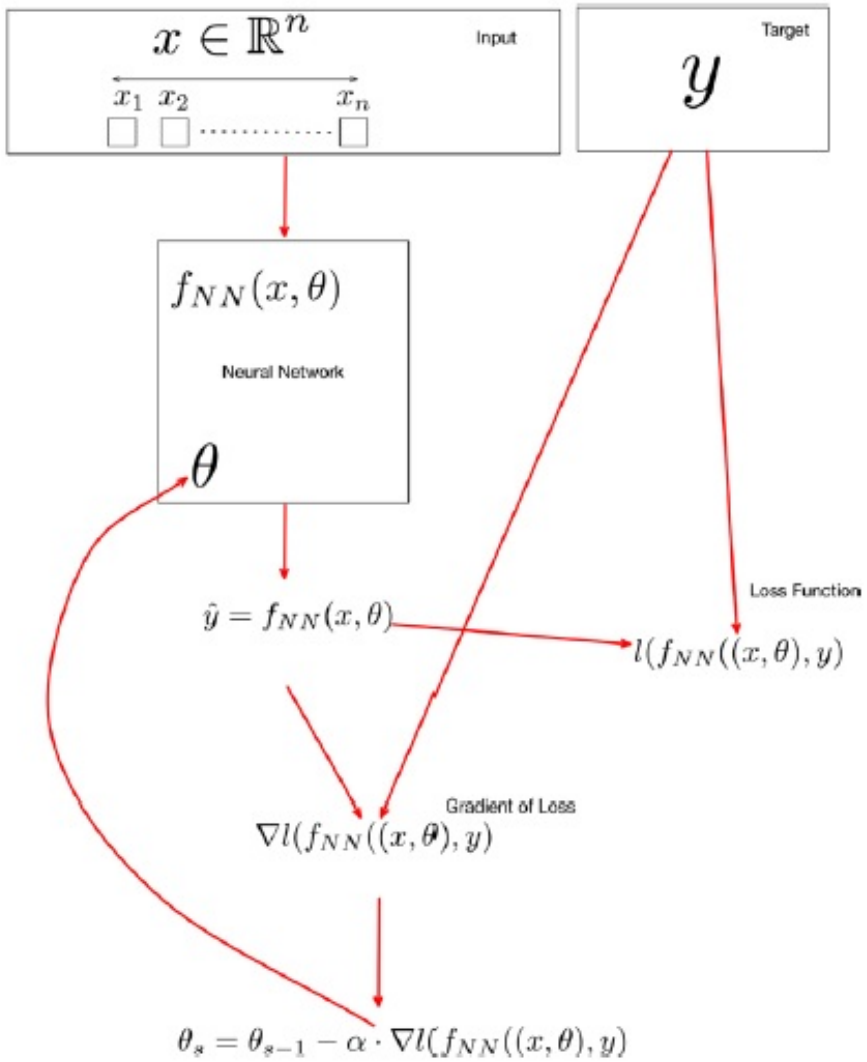
# Activation functions

- A two-layered NN can approximate a nonlinear function with sufficient hidden layer units
- function must be continuously differentiable to compute gradients for optimizing loss function
- functions with finite range (vs. infinite) provide more stable performance with gradient-opt
- smooth functions for convex error surfaces

# ANN - Adjusting weights



# ANN - Computing the loss function



# Backpropagation Principle

- generalization of the 'delta rule' or 'chain rule of calculus'
- Minimization of error E via gradient of loss/cost function: $E = \frac{1}{2}(t - y)^2$, $t$ target, $y$ actual
- learning rate $\eta < 0$, $\delta_j$ error signal
- input $net_j$ to a neuron $j$ is weighted sum of outputs, $o_k$ of previous neurons $k$
- $\phi$ activation, $net_j = \sum_{k=1}^{n} w_{kj} o_k$, $o_j = \phi(net_j) = \phi(\sum_{k=1}^{n} w_{kj} o_k)$

(single-layer network)

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

$$\frac{\partial E}{\partial w_{ij}} = \left(\frac{\partial E}{\partial o_j} \frac{\partial o}{\partial net_j}\right) \frac{\partial net_j}{\partial w_{ij}} = \delta_j o_j$$

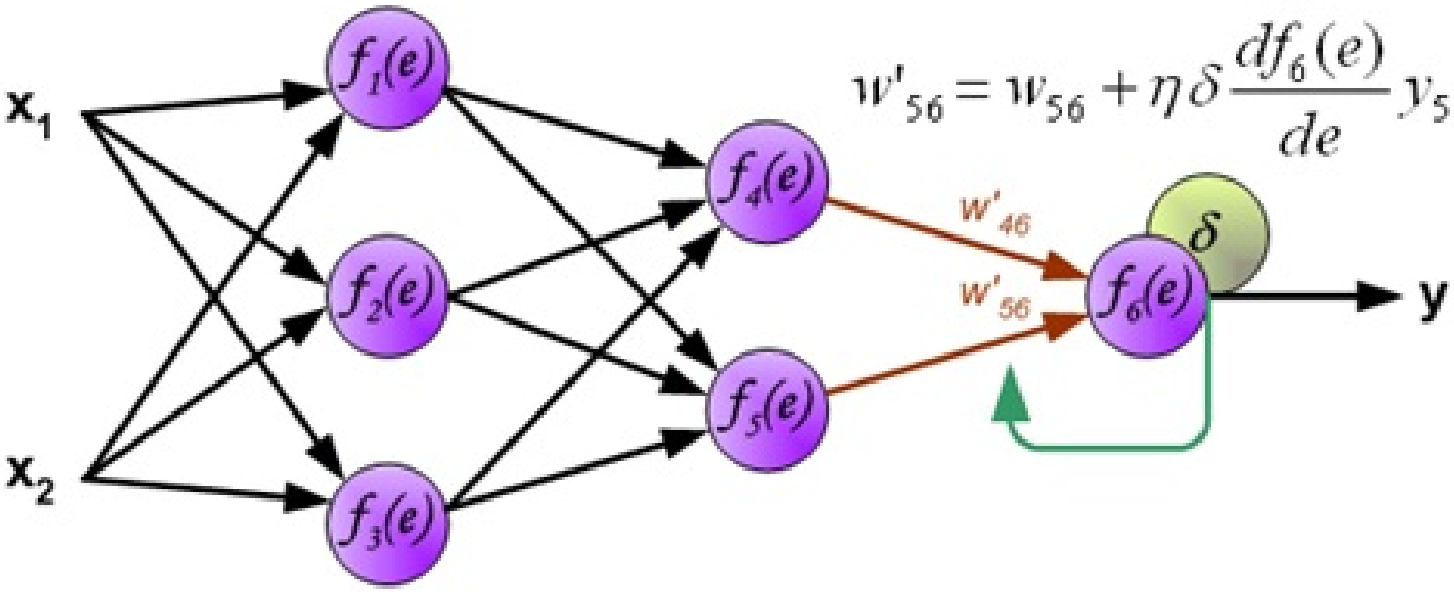
$$\delta_j = \left(\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j}\right) = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{if } output \\ \sum_{l \in L} w_{ij} \delta_l o_j (1 - o_j) & \text{if } inner \end{cases}$$

# Backpropagation

requires lots of labelled/structured data (80% is unstructured)
learning time doesn't scale well (slow with many hidden layers)
local optima vs. global optimum: solver can get trapped

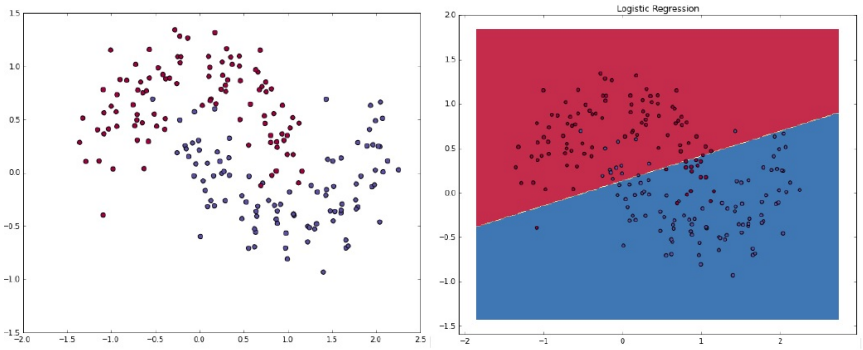
$$w'_{46} = w_{46} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_4$$

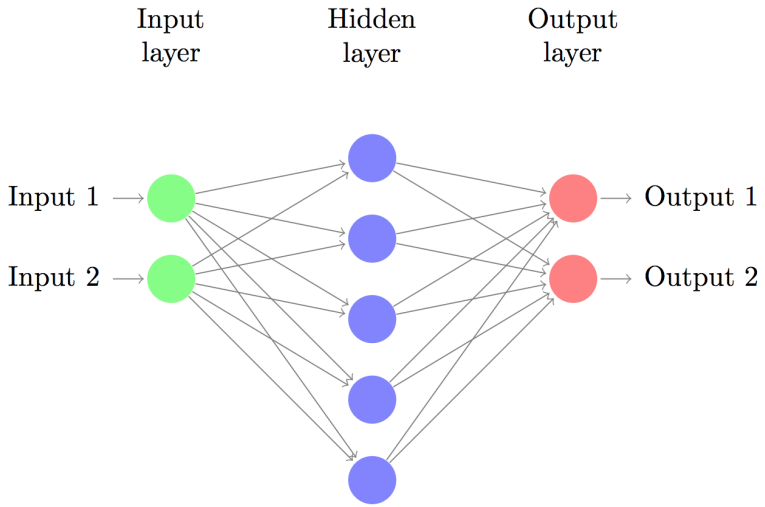
$$w'_{56} = w_{56} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_5$$



# 3-layer NN vs. logistic classifier

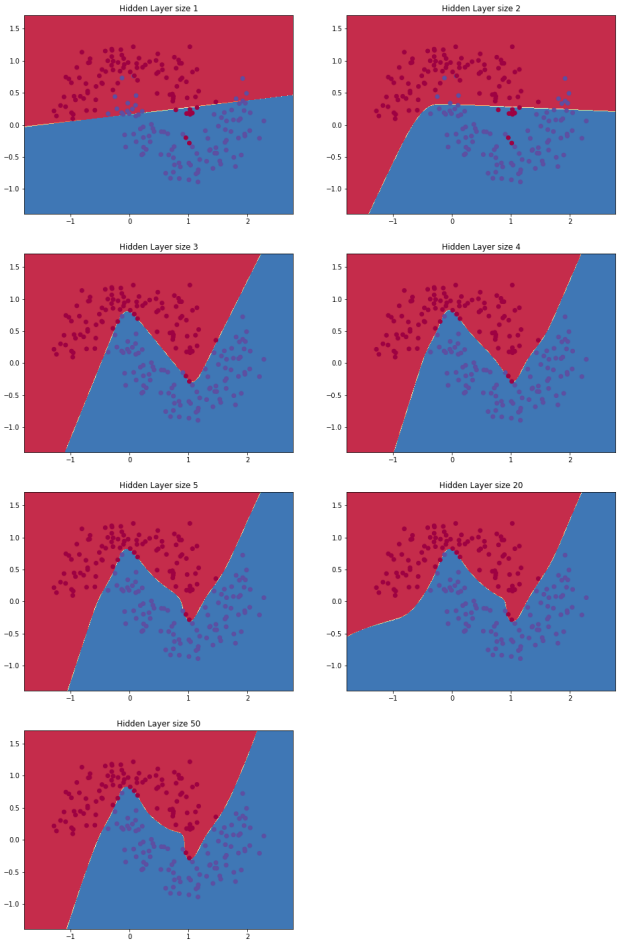
- Denny Britz (WildML, `www.wildml.com`)



# 3-layer NN: varying hidden layers

- tanh activation, softmax output, batch gradient decent (not stochastic)
- few hidden layers captures general trend, too many causes overfitting



# Optimization - high-dimensional datasets

- sgd - has trouble navigating ravines (surface curves are steeper in one dimension than another)
- momentum - accelerates sgd in relevant direction, dampens oscillations
- nag - looks-ahead by calculating approximate gradient of future parameters/position
- adagrad - automatically adapts learning rate to parameters, smaller updates, suited to sparse data
- adadelta - reduces aggressive, monotonically decreasing learning rate of Adagrad (restricting window of past gradients)
- rmsprop - divides the learning rate by exponentially decaying average of squared gradients
- adam - decaying averages of past and past squared gradients (momentum with friction)

# Optimization - high-dimensional datasets

stochastic gradient decent (sgd)

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

nesterov accelerated gradient (nag)

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

adagrad

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}).$$
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

adadelta

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2.$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

rmsprop

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

adaptive moment estimation (adam)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

# Gradient-decent optimization algorithms

- Logistic regression on noisy moons dataset (sklearn)
- shows smoothing effects of momentum-based techniques (results in over shooting and correction)
- error surface is visualized as an average over the whole dataset empirically
- trajectories show the dynamics of mini-batches on noisy data
- ANIMATION: `http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html`

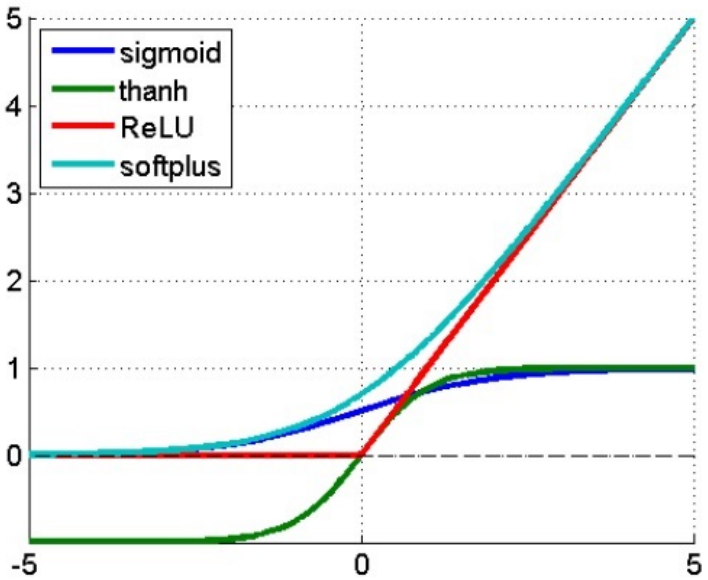
Alec Radford (Lead, Research, https://indico.io/)

# Vanishing gradient problem

- applied to: ANN training with gradient-based learning, backpropagation
- affects saturating neurons or units only
- weight update is proportional to partial derivative of error function with respect to current weight in each interaction
- $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$
- *adaptive* learning rate has inertia term, momentum $\alpha : [0, 1]$
  $\Delta w_{ij}(t+1) = (1-\alpha)\eta\delta_j o_j + \alpha \Delta w_{ij}(t)$
- the error signal passes backwards, starts approaching zero, especially through neurons near saturation.
- If network is deep enough, error signal from the output layer can be completely attenuated on it's way back towards the input layer.
- the gradient will be vanishingly small, effectively preventing the weight from changing its value

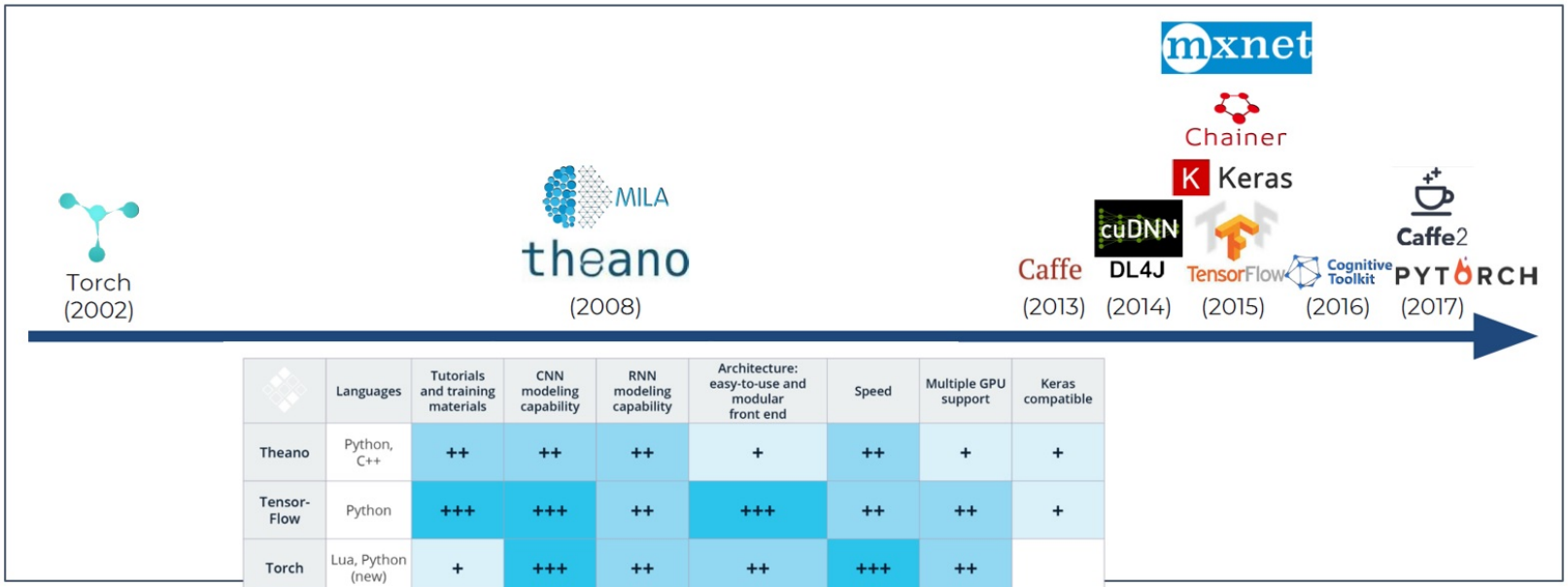
# Vanishing gradient problem

- Use ReLU: highly sparse neural nets are more efficient, more reliable performance
- Gradient of ReLU: 1 hence no attenuation of an error signal propagating backwards




Most deep networks use ReLU - max(0,x) - nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.

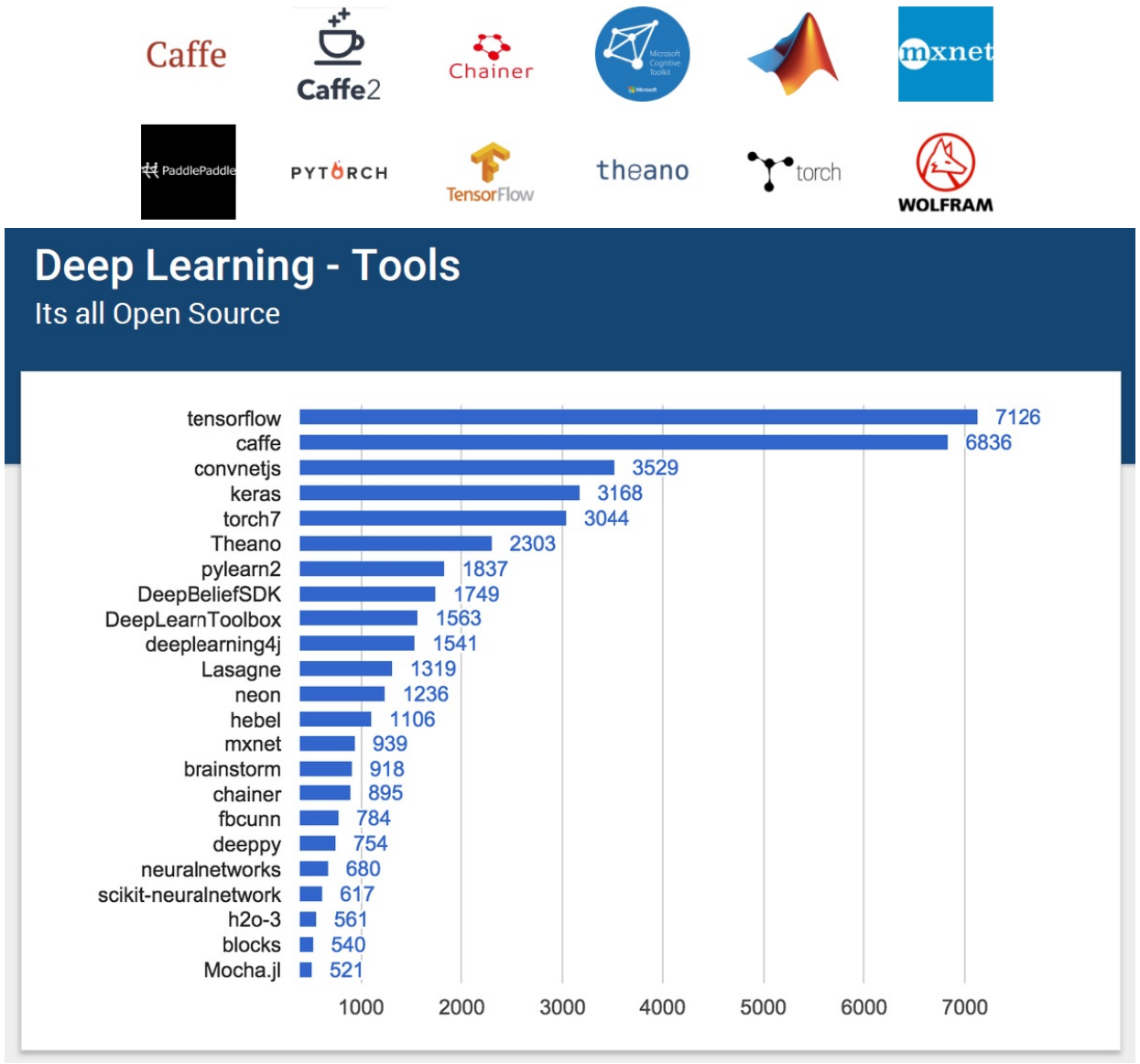
# DL Framework: Tools, libraries, databases




| | Languages | Tutorials and training materials | CNN modeling capability | RNN modeling capability | Architecture: easy-to-use and modular front end | Speed | Multiple GPU support | Keras compatible |
|---|---|---|---|---|---|---|---|---|
| Theano | Python, C++ | ++ | ++ | ++ | + | ++ | + | + |
| Tensor-Flow | Python | +++ | +++ | ++ | +++ | ++ | ++ | + |
| Torch | Lua, Python (new) | + | +++ | ++ | ++ | +++ | ++ | |
| Caffe | C++ | + | ++ | | + | + | + | |
| MXNet | R, Python, Julia, Scala | ++ | ++ | + | ++ | ++ | +++ | |
| Neon | Python | + | ++ | + | + | ++ | + | |
| CNTK | C++ | + | + | +++ | + | ++ | + | |

Gaétan Marceau Caron: École d'hiver MILA 2018
http://www.svds.com/getting-started-deep-learning/

# Popularity of Data Science/AI/ML tools




- Anaconda data science (Python, numpy, sci-kitlearn, Spyder IDE)
- ipython notebooks (Google Cloud): https://ipython.org/notebook.html
- coding language/widely used for ML/DL applications and research
- programs are often not as fast as equivalent C (or Fortran) ones, but offers faster implemention/coding
- numPy: building blocks for storing data in N-dimensional vectors and matrices, linear algebra
- sciPy: provides higher-level math operations (e.g., statistics, optimization, advanced linear algebra)
- scikit-learn a library that provides a wide range of machine learning algorithms
- BLAS Basic Linear Algebra Subprograms (Intel Math Kernel Library (MKL), OpenBLAS)
- (optimized math routines for science, engineering, and financial applications)

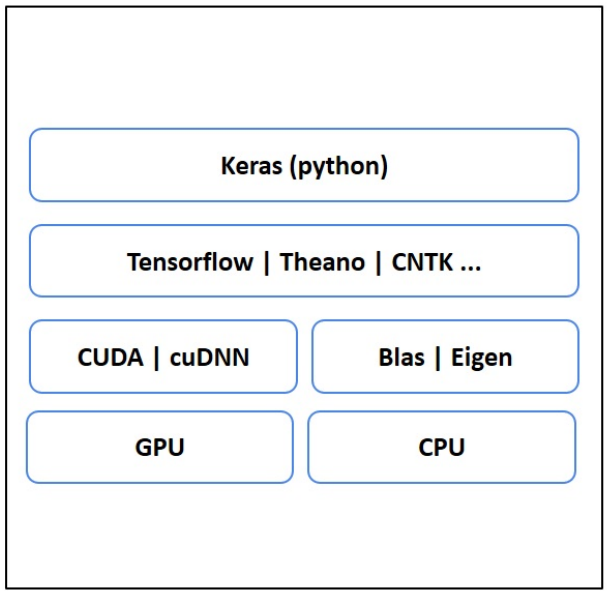

- Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently
- integration with NumPy – Use numpy.ndarray in Theano-compiled functions.
- use of a GPU – perform data-intensive computations much faster than on a CPU.
- efficient symbolic differentiation – derivatives for functions with one or many inputs.
- speed and stability optimizations – $log(1 + x)$ even when x is really tiny.
- dynamic C code - evaluate expressions faster
- extensive unit-testing and self-verification – detect/diagnose errors

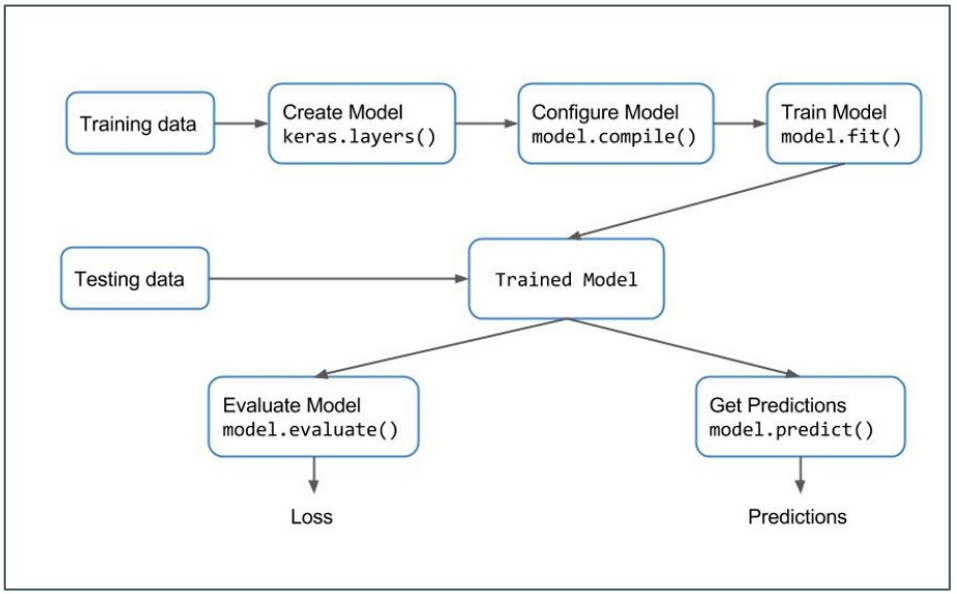

- a high-level neural networks/deep learning API (Python 2.7-3.6 compatible)
- runs on top of Theano, TensorFlow `https://keras.io/`
- Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit
- `https://docs.microsoft.com/en-us/cognitive-toolkit/`
- `https://github.com/Microsoft/CNTK`
- CNTK library in your Python, C#, or C++ programs

# Deep Learning with Keras


```
conda install -c anaconda numpy scikit-learn matplotlib pandas
conda install -c anaconda biopython
conda install -c conda-forge opencv
conda install -c conda-forge tensorflow keras
```

Important: if GPUs are present, use: `conda install -c anaconda tensorflow-gpu`

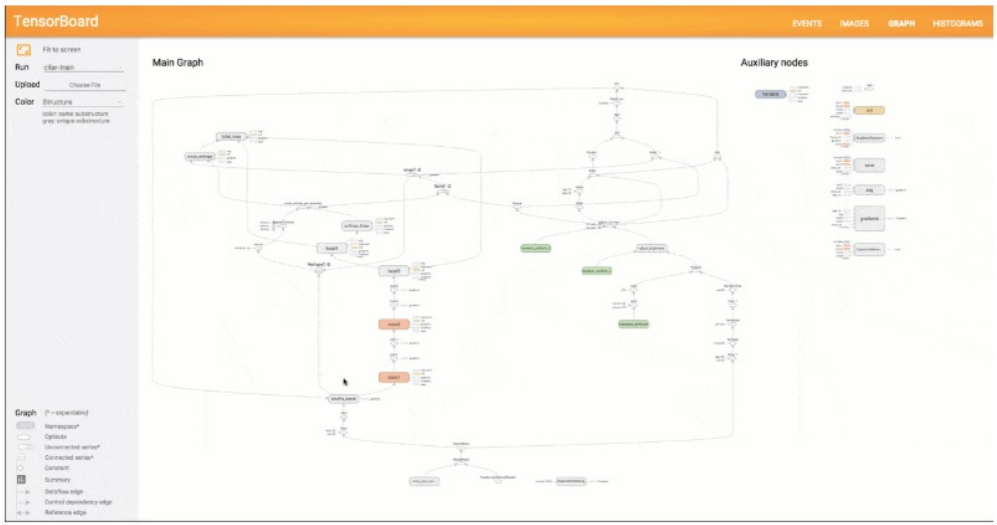


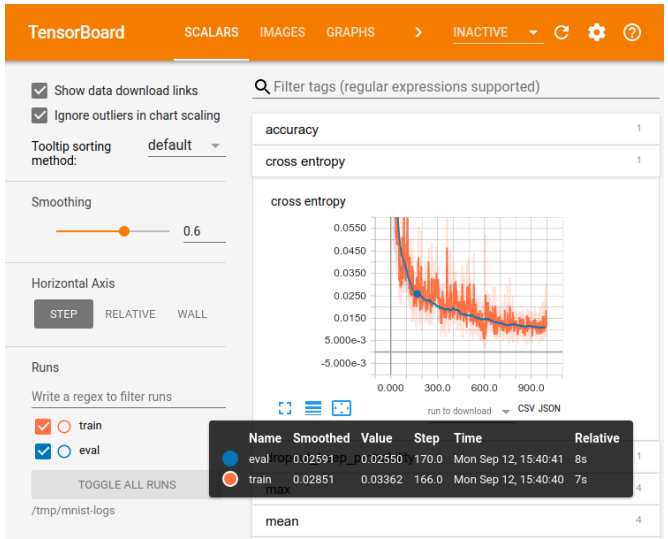




- Open source software library for numerical computation using data flow graphs
- Google Brain Team developed it for researchers and engineers
- Nodes in the graph represent mathematical operations
- Graph edges represent multidimensional data arrays (tensors)
- Tensors are high dimensional generalizations of matrices.
- Flexible architecture to deploy computation to one or more CPUs or GPUs

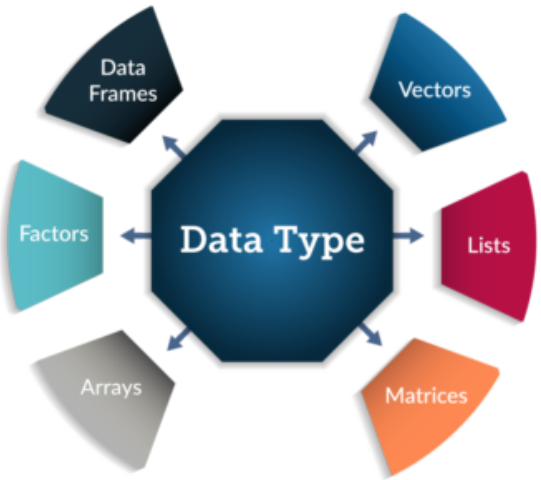
# TensorBoard

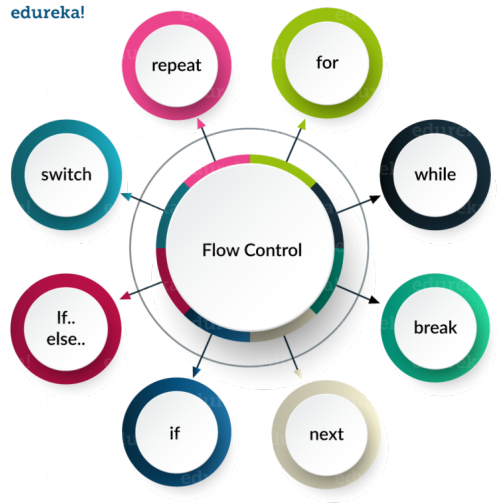
- A suite of visualization tools
- TensorFlow for R: `https://tensorflow.rstudio.com/`
- TensorFlow Playground: `http://playground.tensorflow.org/`

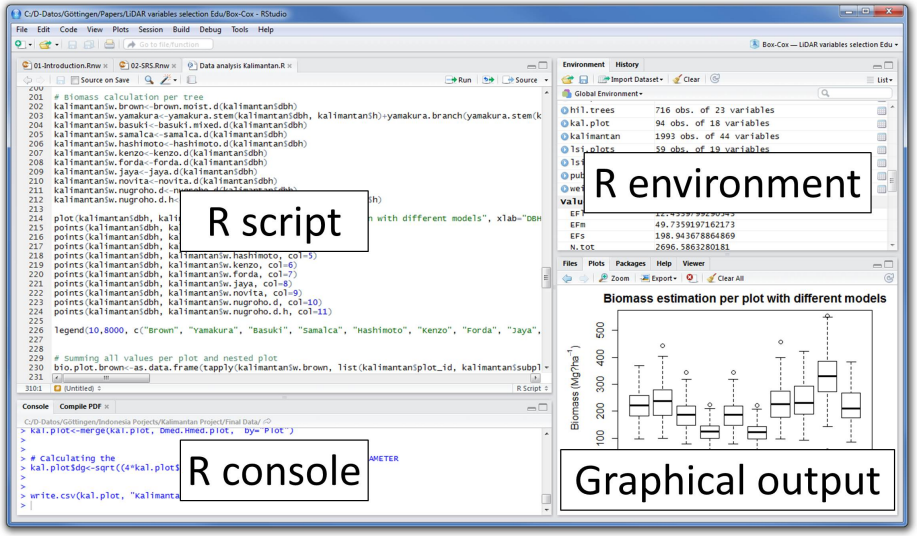


# R Statistical Software/R Studio



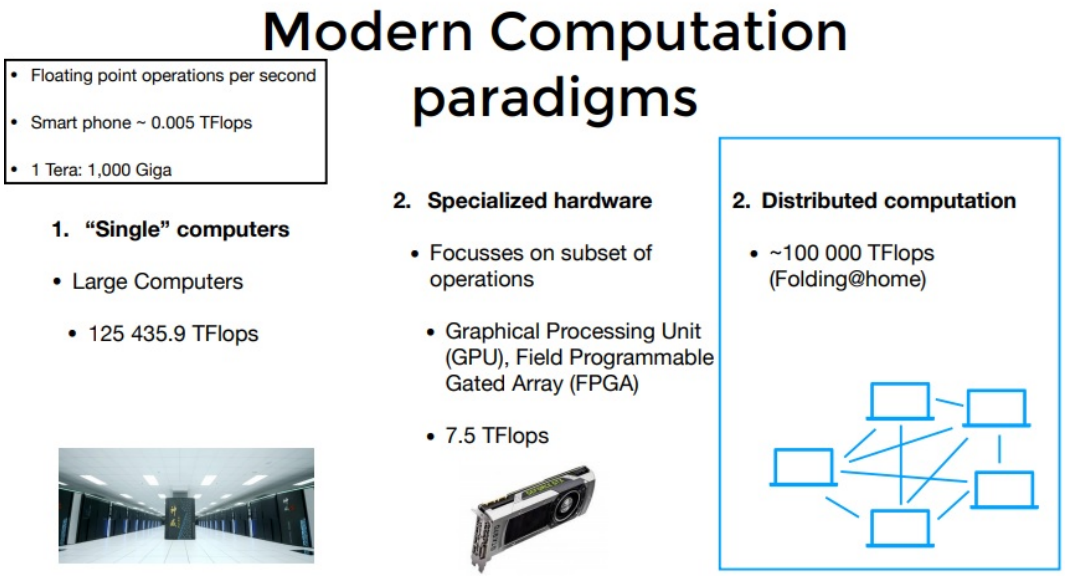



# R: *h2o* and *deepnet*

- R package *h2o*
- provides fast scalable open source tools for machine learning and deep learning, using in-memory compression techniques, Hadoop and Spark cluster-based computing, having a platform that interfaces with R, Python, Scala, Java, and more.
- GLMs, naive Bayes, PCA, time series analysis, *k*-means clustering, RF, GB, deep learning
- *h2o* follows the model of multi-layer, feed-forward neural networks (FNN, DNN)
- uses an improved stochastic gradient descent
- *deepnet* library for DBN

# Accelerated/GPU computing




- Graphics Processing Units (GPU) - key role in modern artificial intelligence
- CPUs have a few cores with lots of cache memory that can handle a few software threads
- GPUs have hundreds of cores that can handle thousands of threads simultaneously
- GPUs can accelerate some software by 100x over a CPU alone

# Cloud computing

- Google Cloud Platform: https://cloud.google.com/gpu/
- GPU Computing Platform: NVIDIA's CUDA Toolkit
- (https://developer.nvidia.com/cuda-zone) (http://www.r-tutor.com/gpu-computing)
- R Interface (Google CloudML)
- Paperspace cloud service (fully preconfigured Ubuntu 16.04 desktop environment equipped with a GPU)

# NVIDIA Deep Learning SDK

- powerful tools and libraries for designing and deploying GPU-accelerated DL applications
- (deep learning primitives, inference, video analytics, linear algebra, sparse matrices, multi-GPU communications)

## NVIDIA Deep Learning SDK

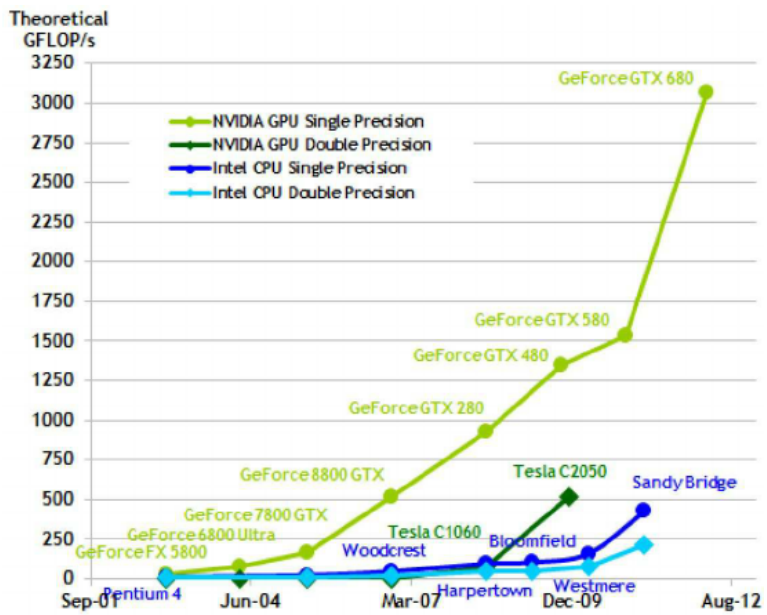
The NVIDIA Deep Learning SDK provides powerful tools and libraries for designing and deploying GPU-accelerated deep learning applications. It includes libraries for deep learning primitives, inference, video analytics, linear algebra, sparse matrices, and multi-GPU communications.


- **Deep Learning Primitives (cuDNN)**: High-performance building blocks for deep neural network applications including convolutions, activation functions, and tensor transformations
- **Deep Learning Inference Engine (TensorRT)**: High-performance deep learning inference runtime for production deployment
- **Deep Learning for Video Analytics (DeepStream SDK)**: High-level C++ API and runtime for GPU-accelerated transcoding and deep learning inference
- **Linear Algebra (cuBLAS)**: GPU-accelerated BLAS functionality that delivers 6x to 17x faster performance than CPU-only BLAS libraries
- **Sparse Matrix Operations (cuSPARSE)**: GPU-accelerated linear algebra subroutines for sparse matrices that deliver up to 8x faster performance than CPU BLAS (MKL), ideal for applications such as natural language processing
- **Multi-GPU Communication (NCCL)**: Collective communication routines, such as all-gather, reduce, and broadcast that accelerate multi-GPU deep learning training on up to eight GPUs

The Deep Learning SDK requires CUDA Toolkit, which offers a comprehensive development environment for building new GPU-accelerated deep learning algorithms, and dramatically increasing the performance of existing applications

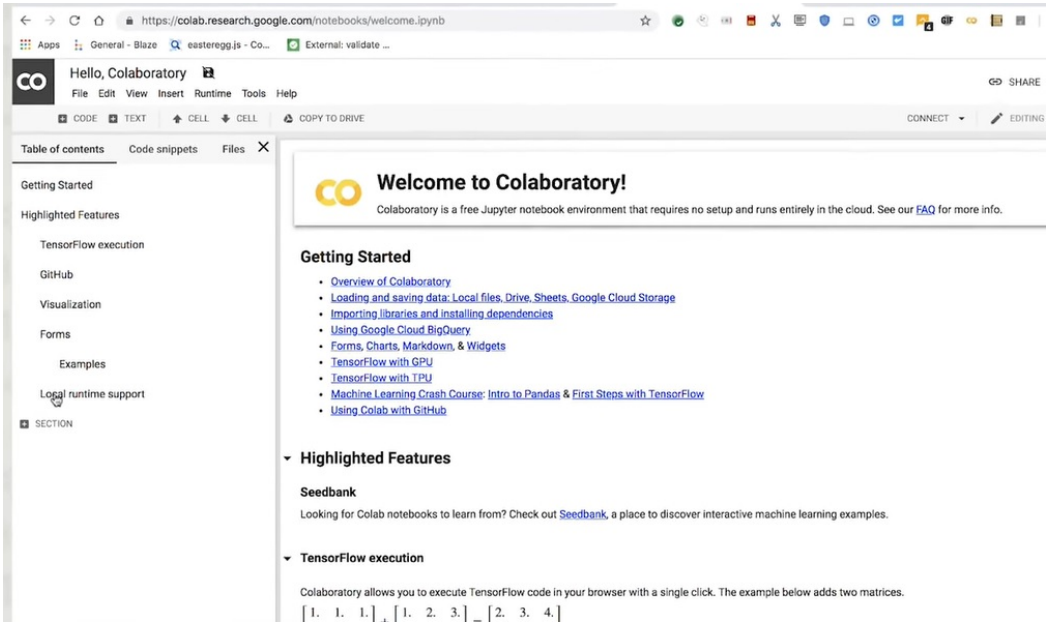
# Computational speed

- Performance each second: equivalent to the Earth's entire population doing one calculation a second for an entire year
- Summit HPC (Oak Ridge) Smashed the 100-petaflop barrier (NVIDIA Volta V100 Tesla)
- Quantum computing: quantum bits, or "qubits" instead of 1s and 0s to handle information

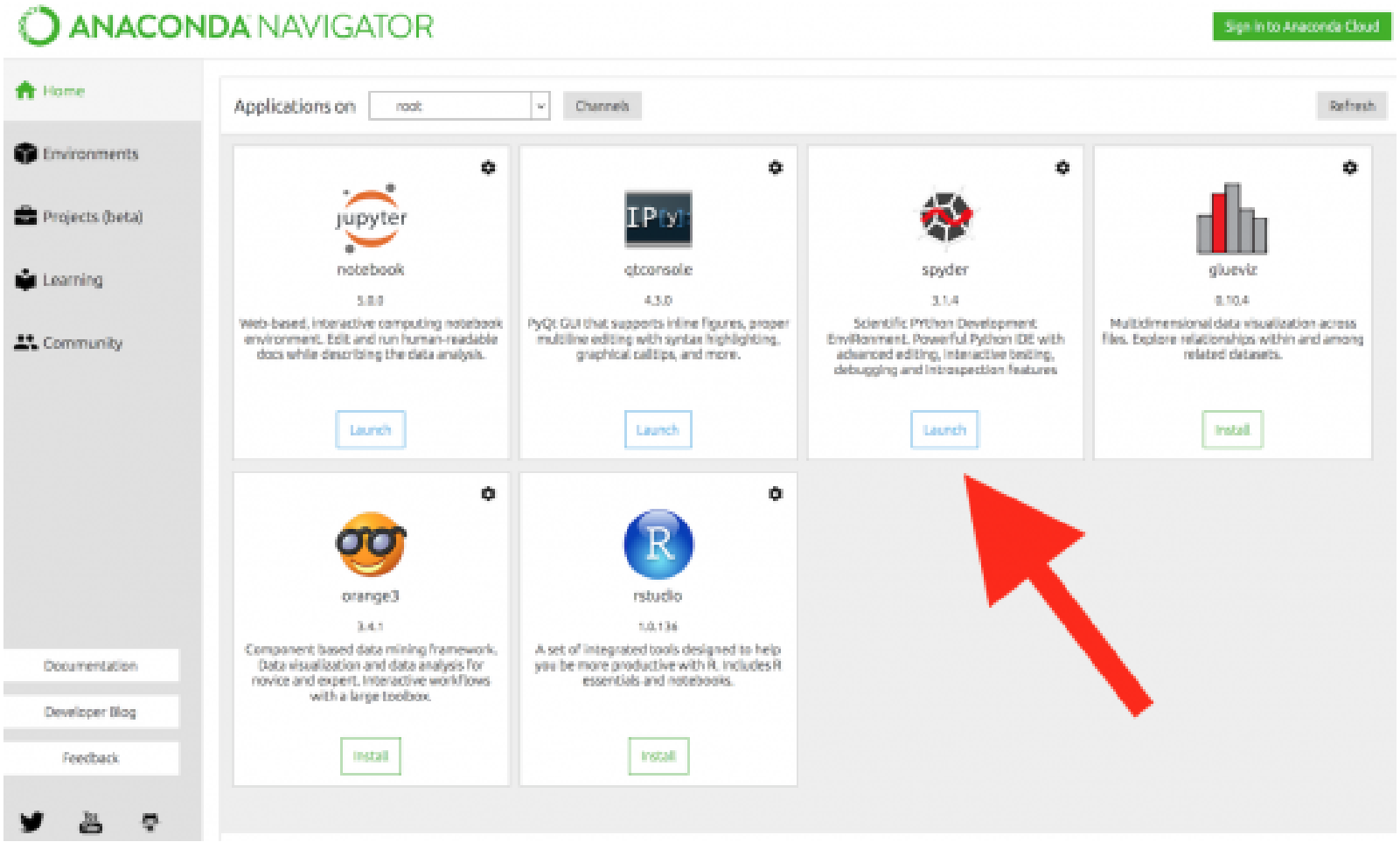



Credits:
Laurent Charlin (HEC Montreal/Toronto)-Machine Learning for Large-Scale Data Analysis and Decision Making
NVIDIA Corporation: NVIDIA CUDA C programming guide (2012) Version 4.2

# Colaboratory (Jupyter)
# (Google cloud environment)




- *colab.research.google.com*
- integrates text and python source code (Jupyter)
- shows result of code execution
- free access to powerful GPUs (Nvidia)


ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Projects (beta)
Learning
Community
Documentation
Developer Blog
Feedback
Applications on
root
Channels
Refresh
jupyter
notebook
5.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
IPy
qtconsole
4.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
spyder
3.1.4
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch
glueviz
0.10.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install
orange3
3.4.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install
R
rstudio
1.0.136
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

## Installing with Anaconda

**The Anaconda installation is community supported, not officially supported.**

Take the following steps to install TensorFlow in an Anaconda environment:

1. Follow the instructions on the Anaconda download site to download and install Anaconda.
2. Create a conda environment named `tensorflow` by invoking the following command:

```
C:> conda create -n tensorflow pip python=3.5
```

3. Activate the conda environment by issuing the following command:

```
C:> activate tensorflow
 (tensorflow)C:>  # Your prompt should change
```

4. Issue the appropriate command to install TensorFlow inside your conda environment. To install the CPU-only version of TensorFlow, enter the following command:

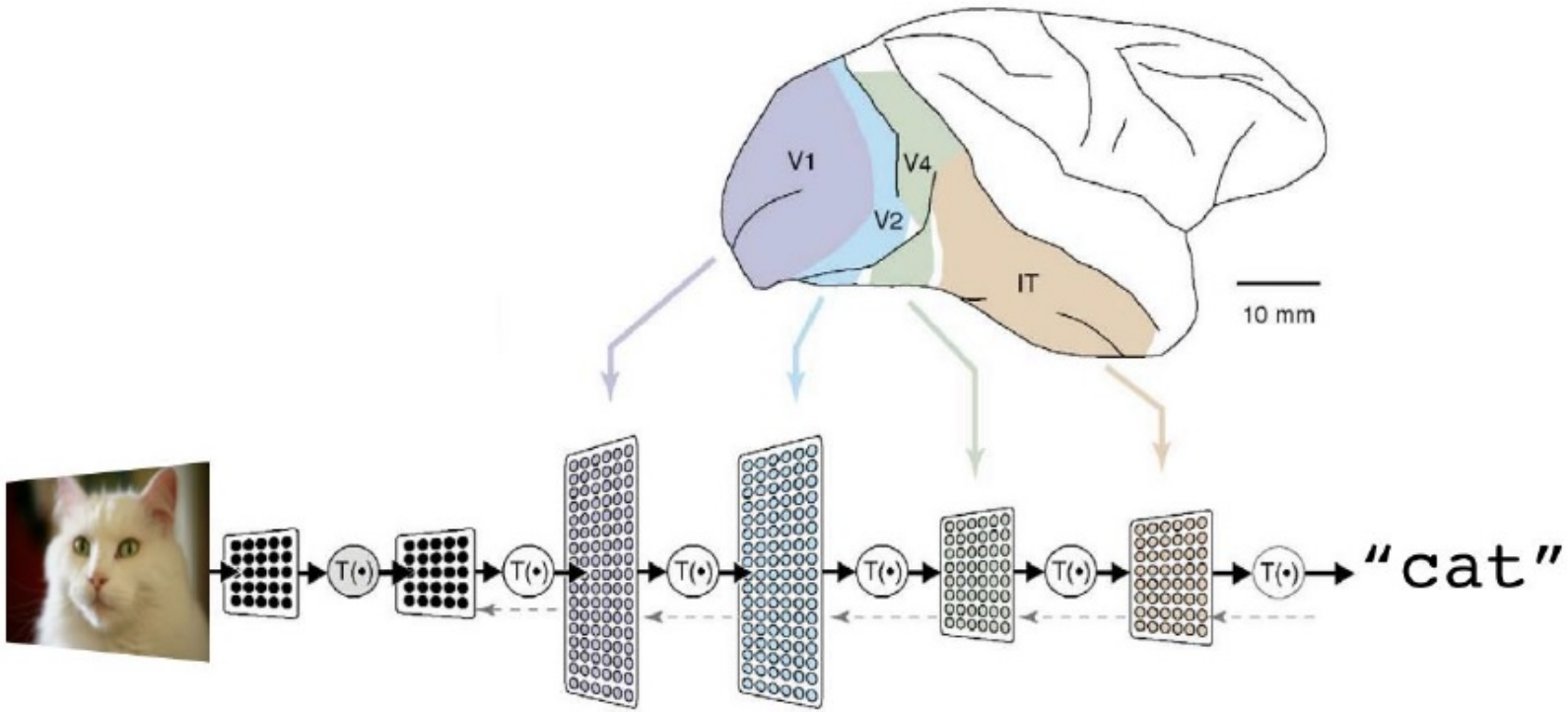
```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow
```

   To install the GPU version of TensorFlow, enter the following command (on a single line):

```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow-gpu
```

# Image Analysis: Convolutional Networks (CNNs)

- biologically-inspired variants of multilayer perceptrons (MLPs)
- mimic visual receptive fields, exploit spatial correlation

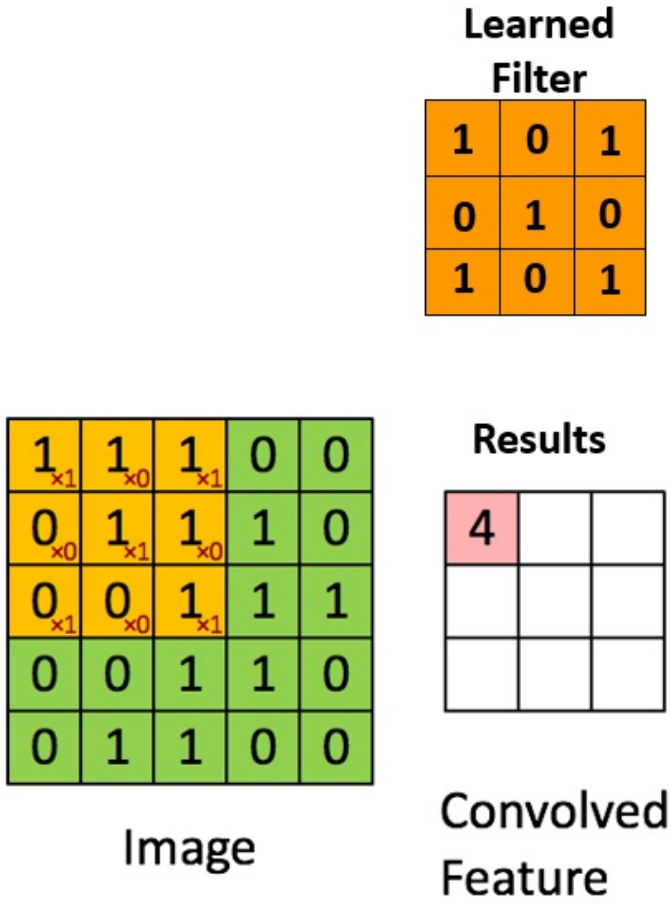



A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations (e.g. edge -> nose -> face). The output layer combines those features to make predictions.

# Convolution layer

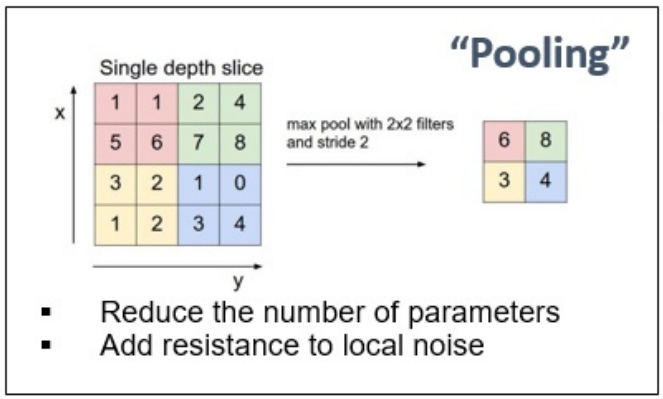
A layer in which **some learned filters** are applied on each input data. The inverse transformation is the deconvolution (deconv)

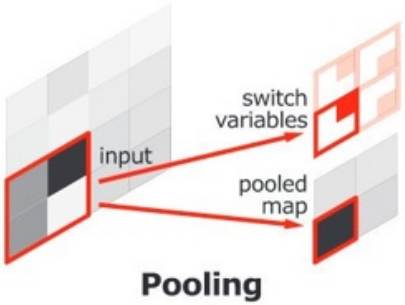
In these layers, the network try to learn characteristics of the data and/or remove noise.

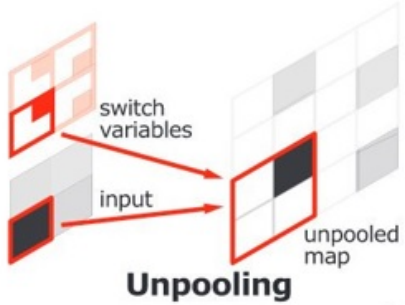


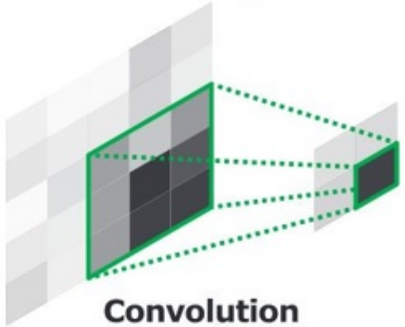
# Pooling layers

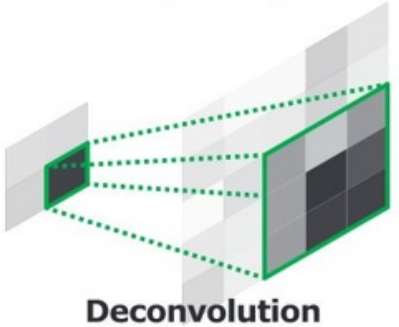
A layer **which reduces the dimensions** of the network by applying a **sum, maximum, minimum** or an **average** over the previous layer.



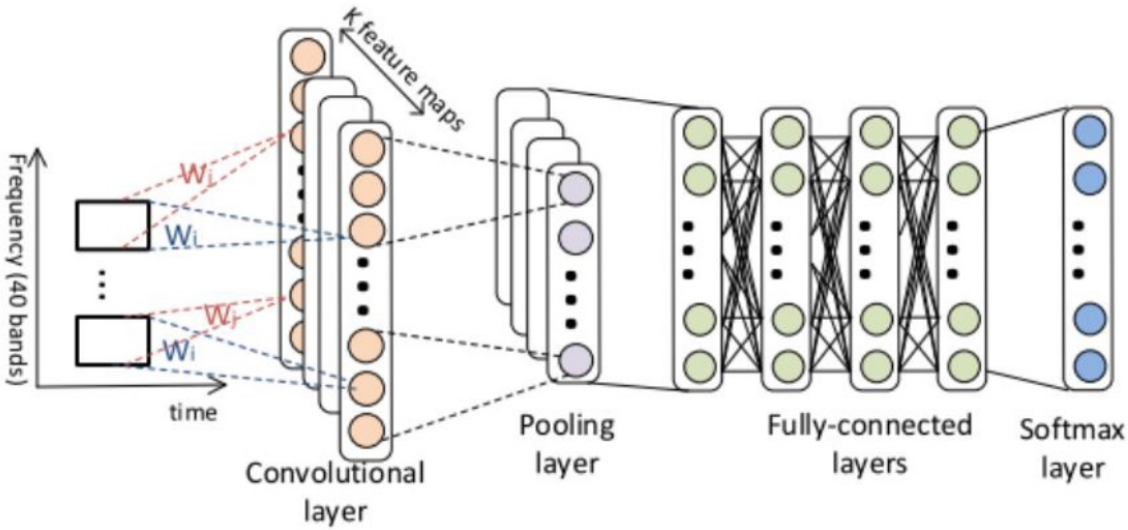



# Convolutional Neural Networks (CNNs)

- feature map: inputs of hidden units in layer $m$ form a subset of units in layer $m-1$
- convolution layer: feature detector that automatically learns to filter out information from an input (kernel)
- pooling layer: average/max of a feature over a region (subsampling/downsampling: helps detect objects, reduce memory size)
- softmax layer: softmax (multinomial logistic) regression, multi-classes



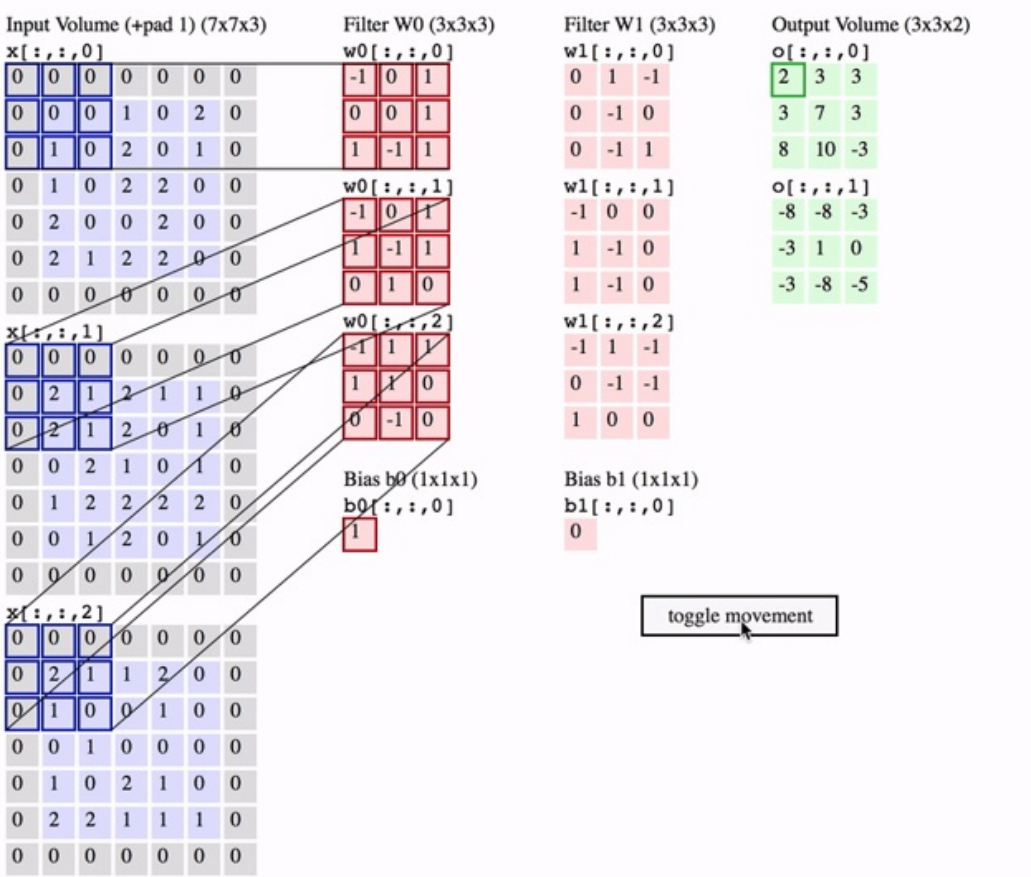
# Convolution in 3D

Convolution can be applied to more than one dimension and with more than one filter's set.

Red

Green

Blue



# Convolutional Neural Networks (CNNs)

- Convolution: adding each element of the image to its local neighbors, weighted by a kernel
- process described as "convolution" (really is "cross-correlation")

CNNs are increasingly popular due to three important factors:

1. Features are learned directly/automatically - no manual extraction
2. Produce state-of-the-art recognition results
3. Can be re-trained for new recognition tasks

$$\text{convoluted } x = f \otimes g = \int_{-\infty}^{\infty} f(x-u)g(u)\,du = \mathcal{F}^{-1}\left(\sqrt{2\pi}\mathcal{F}[f]\mathcal{F}[g]\right)$$

$$\text{cross-correlated } x = f \star g = \int_{-\infty}^{\infty} f(x+u)g(u)^*\,du = \mathcal{F}^{-1}\left(\sqrt{2\pi}\mathcal{F}[f](\mathcal{F}[g])^*\right)$$

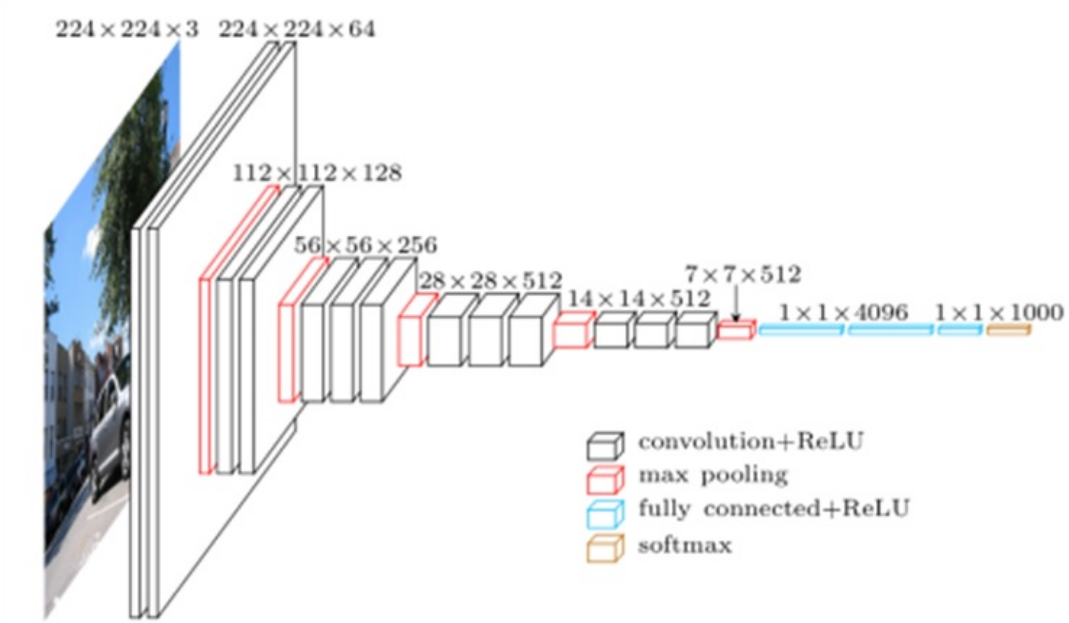
$$f(x) \star g(x) = f^*(-x) \otimes g(x)$$

# Pre-trained CNNs

- Pre-trained start with a pretrained network and use it to learn a new task
- Use an existing trained network on millions of images and retrain it for new object classification using only hundreds of images
- Re-trained convoluted networks (CNNs)
- GoogLeNet, AlexNet, VGG

# VGG-16 CNN pre-trained deep architectures

- VGG (Visual Graphics Group), Oxford University
- Pyramidal shape (19-layers): bottom layers are wide, top layers are deep



# DL on Crop Leaf Database (Plant Village)

PlantVillage has gathered a collection of more than 54'000 images over time. Pictures were shot by different people, using different cameras with different automatic adjustments, under varying lightning conditions. Some are already segmented. Here are a few examples:

- `https://plantvillage.psu.edu/`
- TensorFlow for cassava farmers: `https://plantvillage.psu.edu/blogposts/27-tensorflow-summit`

# CNN Processing...

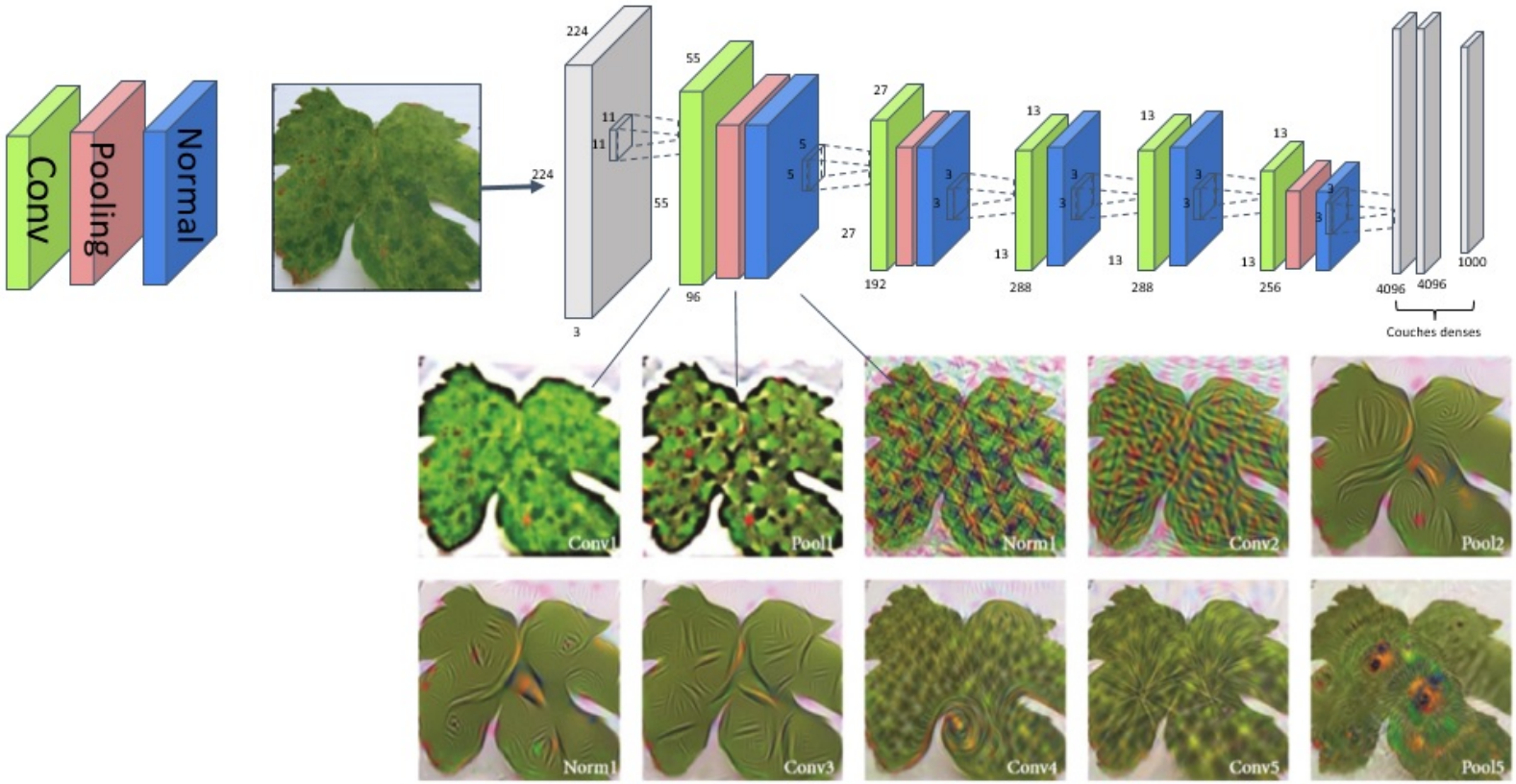
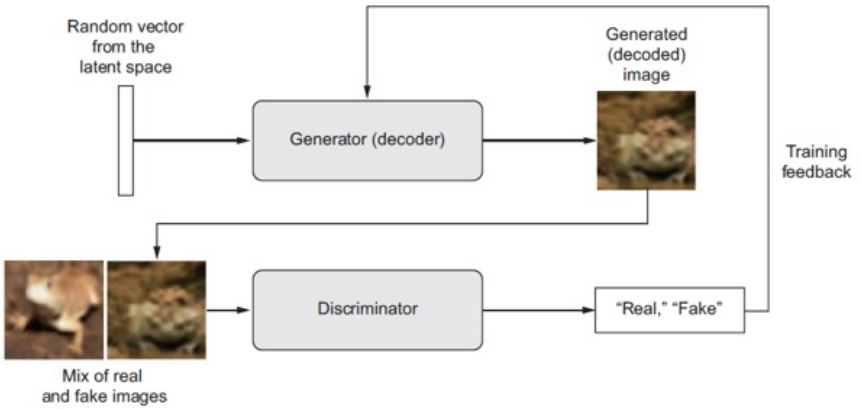
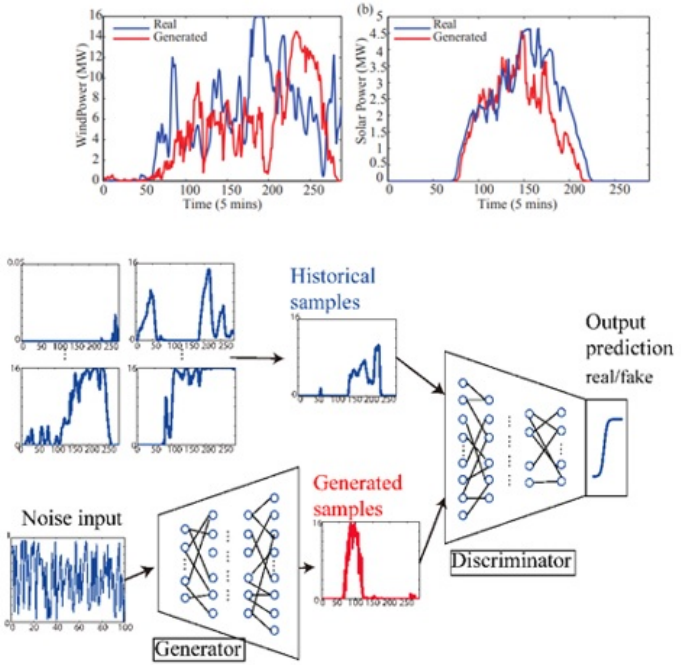



Fig. 2. Visualization of the output layers images after each processing step of the CaffeNet CNN (i.e. convolution, pooling, normalization) at a plant disease identification problem based on leaf images.
Source: Sladojevic et al. (2016).

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, *147*, 70-90

# Generative Adversarial Networks

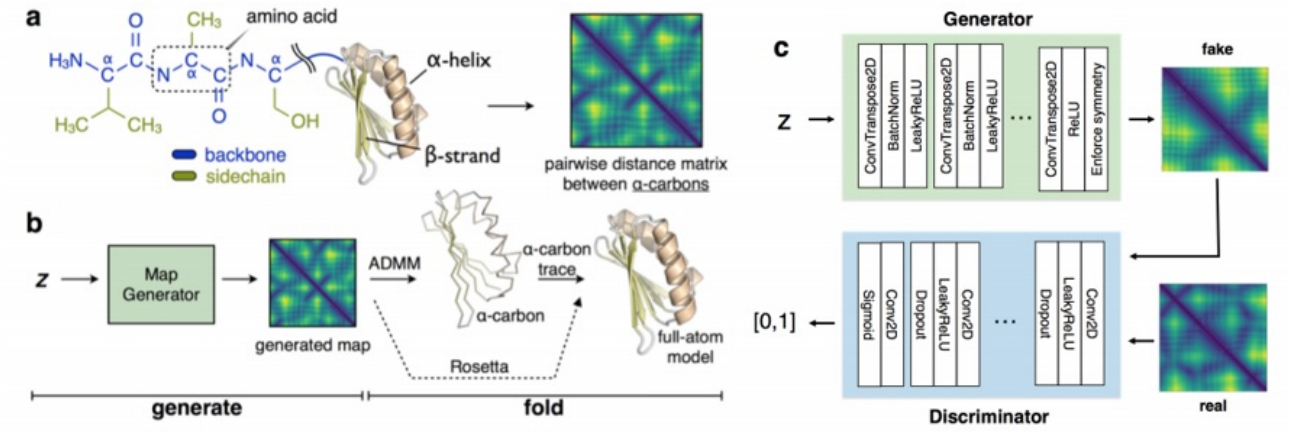



arxiv.org/pdf/1707.09676.pdf

https://github.com/eriklindernoren/Keras-GAN

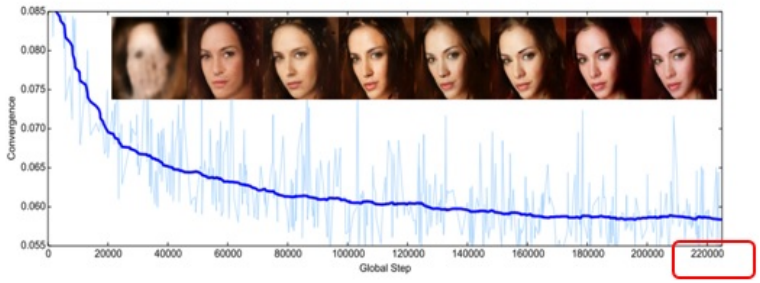
# Generative Adversarial Networks





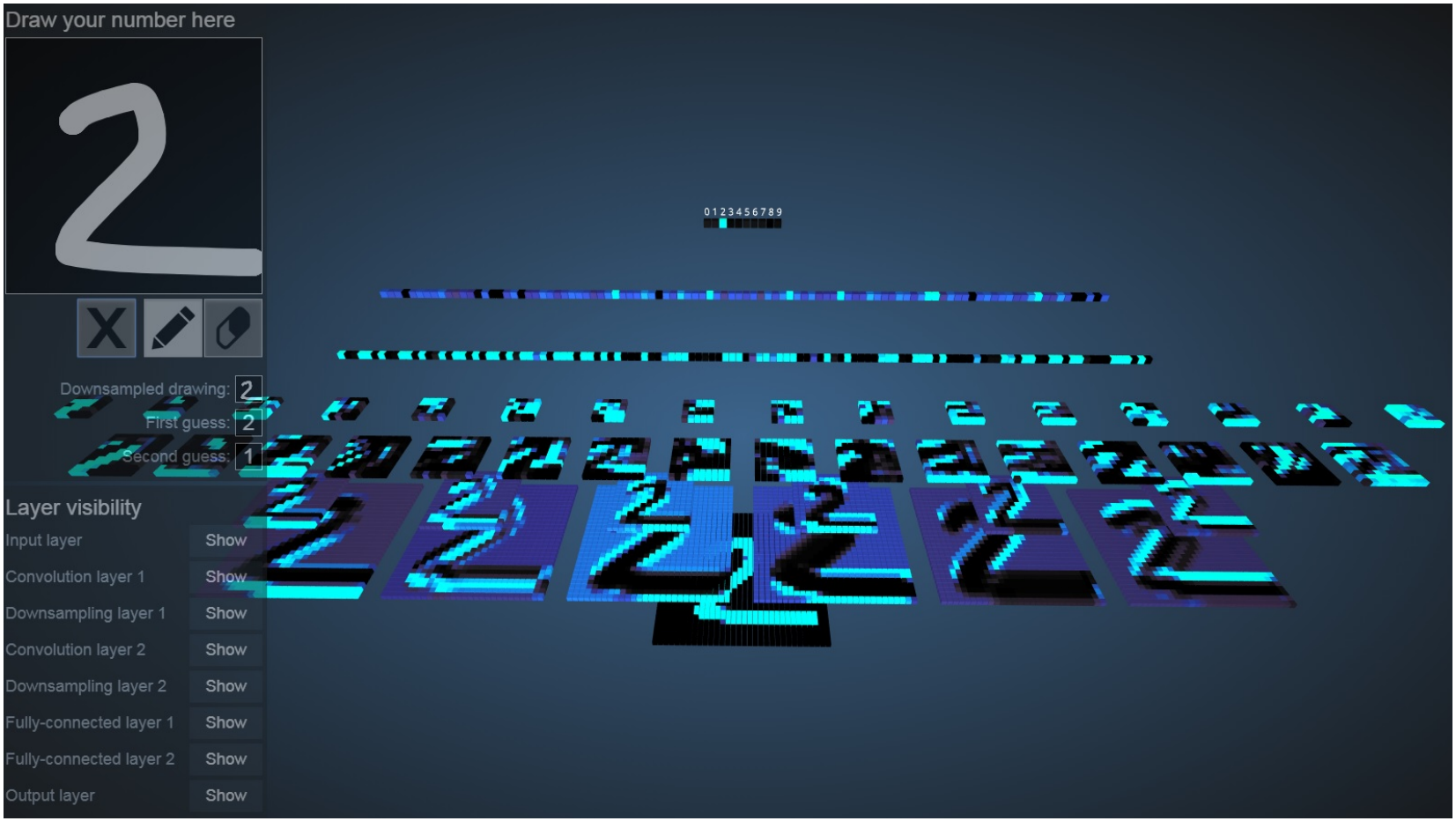
https://github.com/hanzhanggit/StackGAN

https://papers.nips.cc/paper/7978-generative-modeling-for-protein-structures.pdf



# 3D Visualization of CNN (ConvNet)

MNIST handwritten digit database

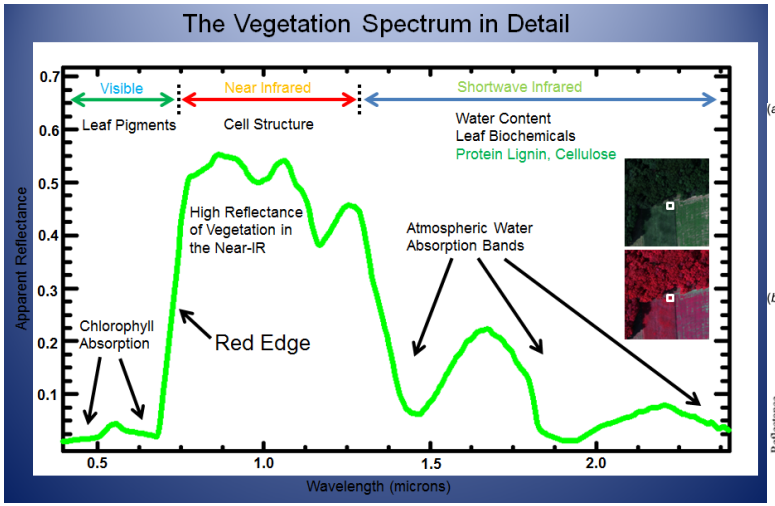


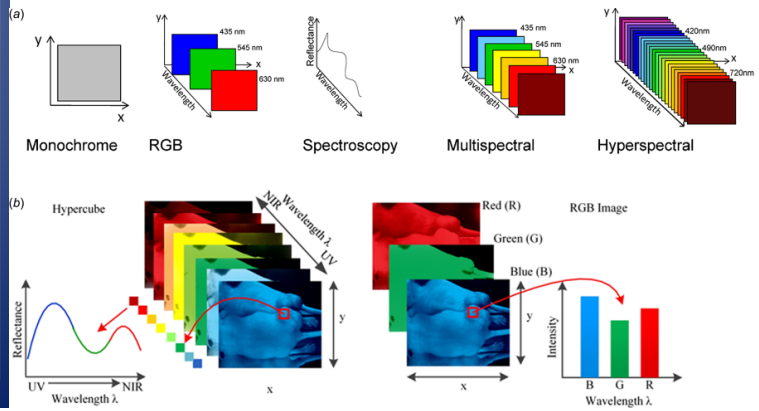

Adam Harley: `http://scs.ryerson.ca/~aharley/vis/conv/`

# Practical examples (field and satellite imagery)

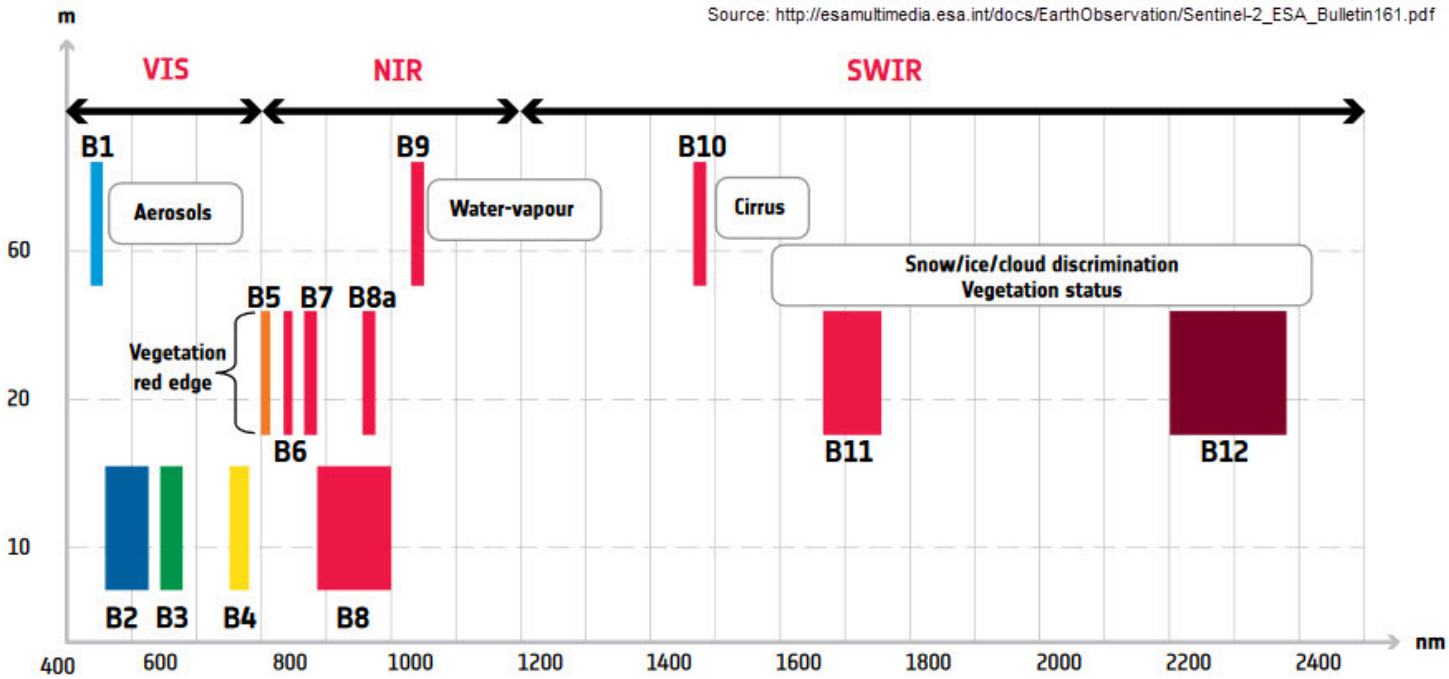
- Tutorial examples:
- Etienne Lord - Land-use using Deep CNN model and Sentinel data (Python)
- Extracting vegetation indices (R)
- Deep learning using Google Earth Engine (GEE) (Python)
- Fruit classification (R)

# Hyperspectral/multispectral satellite imagery



# ESA Sentinel-2 spectral bands: (10-60)m




↑ Spatial resolution versus wavelength: Sentinel-2's span of 13 spectral bands, from the visible and the near-infrared to the shortwave infrared at different spatial resolutions ranging from 10 to 60 m on the ground, takes land monitoring to an unprecedented level

# Classification of agricultural land use by ensemble of convolutional neural networks

Etienne Lord, Louis Longchamps

St-Jean-sur-Richelieu RDC

etienne.lord@canada.ca, researcher digital agronomy

Agriculture et Agroalimentaire Canada | Agriculture and Agri-Food Canada

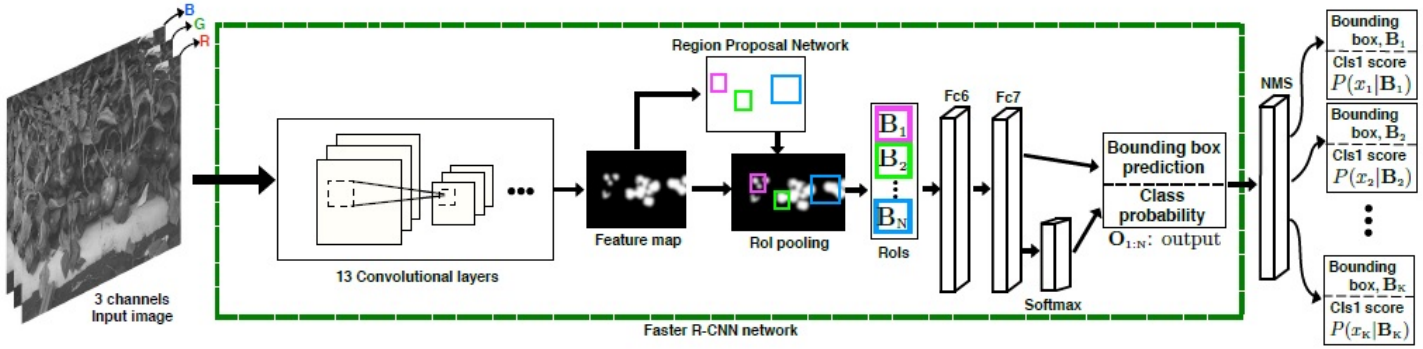
Canada

# DeepFruits (Fast-CNN)

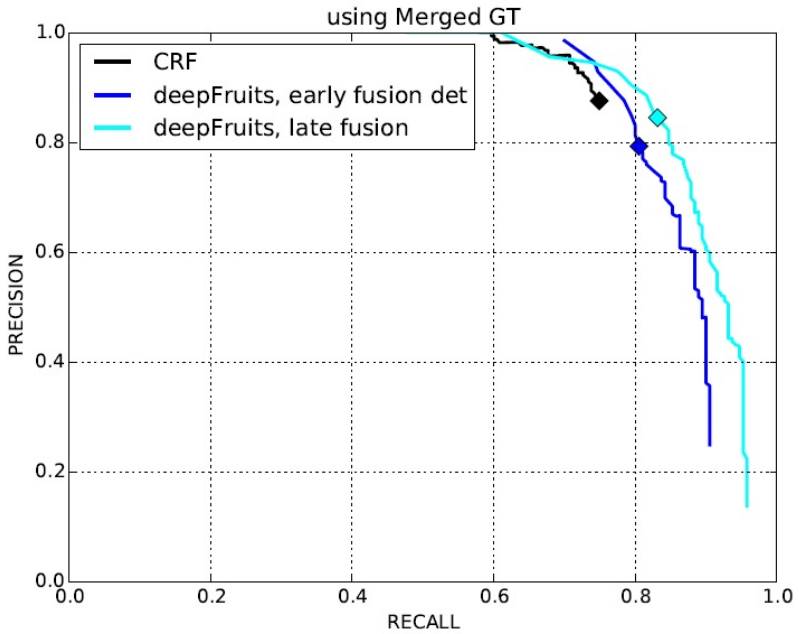
- accurate object detection still remains a challenging problem
- difficulty in pixel-level ground-truthing, huge processing time
- "DeepFruits" (Sa et al., 2016) using DNNs
- Faster R-CNN (using regional bounding boxes)
- VGG-16 pre-trained CNN model that has 13 convolutional layers
- early fusion: concatenates a 1-channel NIR image with a 3-channel RGB image
- late fusion: stacks outputs, N=300 bounding boxes
- (RGB and NIR networks)

# DeepFruits (Fast-CNN)


**Figure 4.** (a) The 3 × 3 (pixels) Conv164 filters of the RGB network from VGG, (b) The input data and (c) One of the feature activations from the conv5 layer. The cyan boxes in (b) are manually labelled in the data input layer to highlight the corresponding fruits of the feature map.



# DeepFruits (Fast-CNN)



# Fruit image recognition using DL (CNN)

- Fruits-360: A dataset of images containing fruits (Apple, Plum, Peach, Apricot etc..)
- A high-quality, dataset of 50590 images, 75 fruit types/classes
- Multi-fruits set size: 45 images (more than one fruit (or fruit class) per image)
- Image size: 100x100 pixels
- Horea Muresan, 2018: Acta Univ. Sapientiae, Informatica, 10(1):26-42 (`https://mihaioltean.github.io`)

# Fruit image recognition using DL (CNN)

- simple sequential CNN: Layers: 2 convolutional, 1 pooling layer, 1 dense
- Training set size: 37836 images (one fruit per image).
- Inference/Test set size: 12709 images (one fruit per image)

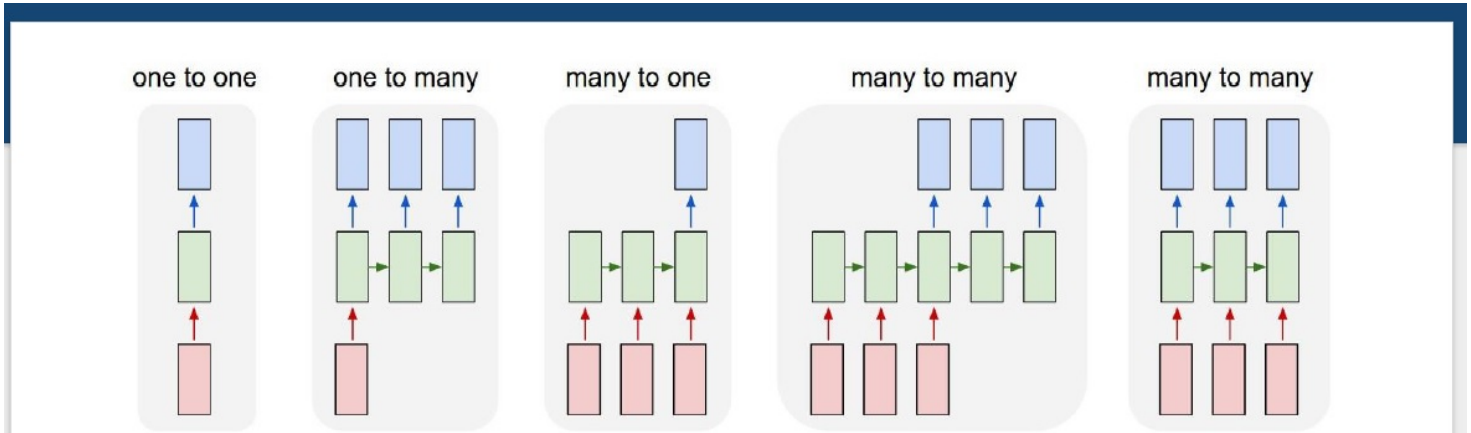
# Environmental multivariate datasets

- Wiki:`https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research`
- Python: http://earthpy.org/
- Physical (Earth science, Anomoly, Weather)
- Biological (Human/Animal/Plant/Microbe)
- Chemical (Drug discovery/assays)

| Dataset Name | Brief description | Preprocessing | Instances | Format | Default Task | Created (updated) | Reference | Creator |
|---|---|---|---|---|---|---|---|---|
| Cloud DataSet | Data about 1024 different clouds. | Image features extracted. | 1024 | Text | Classification, clustering | 1989 | [376] | P. Collard |
| El Nino Dataset | Oceanographic and surface meteorological readings taken from a series of buoys positioned throughout the equatorial Pacific. | 12 weather attributes are measured at each buoy. | 178080 | Text | Regression | 1999 | [377] | Pacific Marine Environmental Laboratory |
| Greenhouse Gas Observing Network Dataset | Time-series of greenhouse gas concentrations at 2921 grid cells in California created using simulations of the weather. | None. | 2921 | Text | Regression | 2015 | [378] | D. Lucas |
| Atmospheric CO2 from Continuous Air Samples at Mauna Loa Observatory | Continuous air samples in Hawaii, USA. 44 years of records. | None. | 44 years | Text | Regression | 2001 | [379] | Mauna Loa Observatory |
| Ionosphere Dataset | Radar data from the ionosphere. Task is to classify into good and bad radar returns. | Many radar features given. | 351 | Text | Classification | 1989 | [327][380] | Johns Hopkins University |
| Ozone Level Detection Dataset | Two ground ozone level datasets. | Many features given, including weather conditions at time of measurement. | 2536 | Text | Classification | 2008 | [381][382] | K. Zhang et al. |

# Recurrent (RNNs)

- feedback (output is next input)
- sequential inputs, outputs
- time-series: image captioning/motion detection, document classification

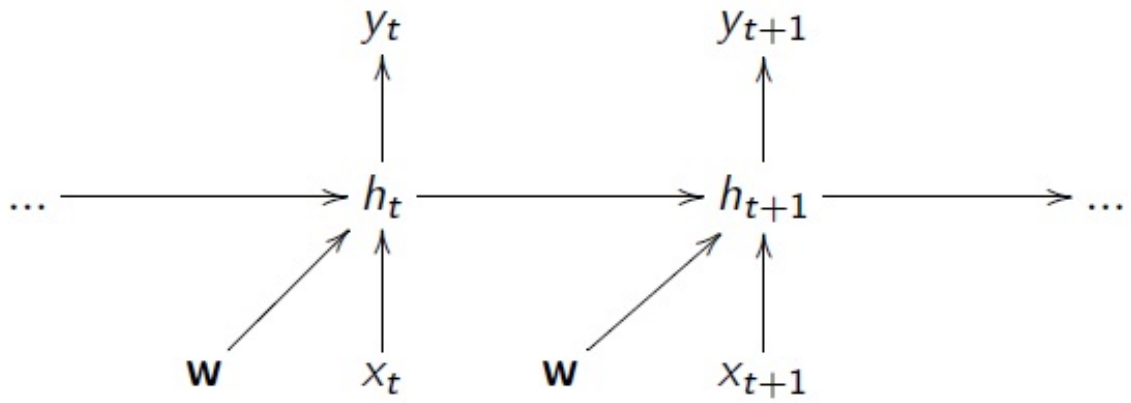



RNNs are general computers which can learn algorithms to map input sequences to output sequences (flexible-sized vectors). The output vector's contents are influenced by the entire history of inputs.

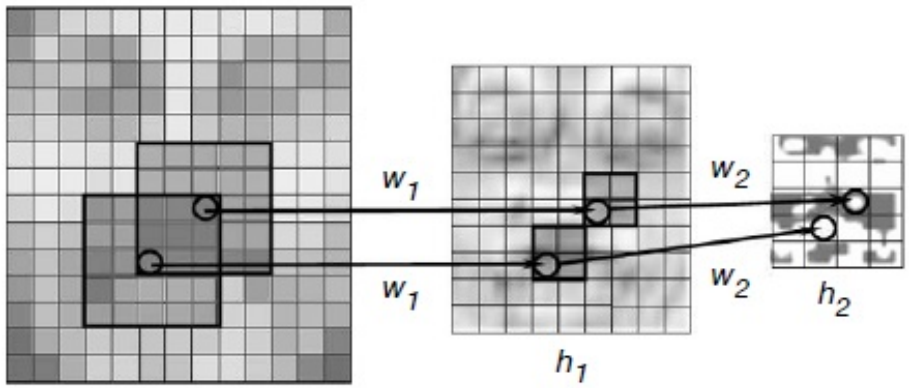
State-of-the-art results in time series prediction, adaptive robotics, handwriting recognition, image classification, speech recognition, stock market prediction, and other sequence learning problems. *Everything can be processed sequentially.*

# CNN and RNNs: Weight sharing

Recurrent neural network shares weights between time-steps

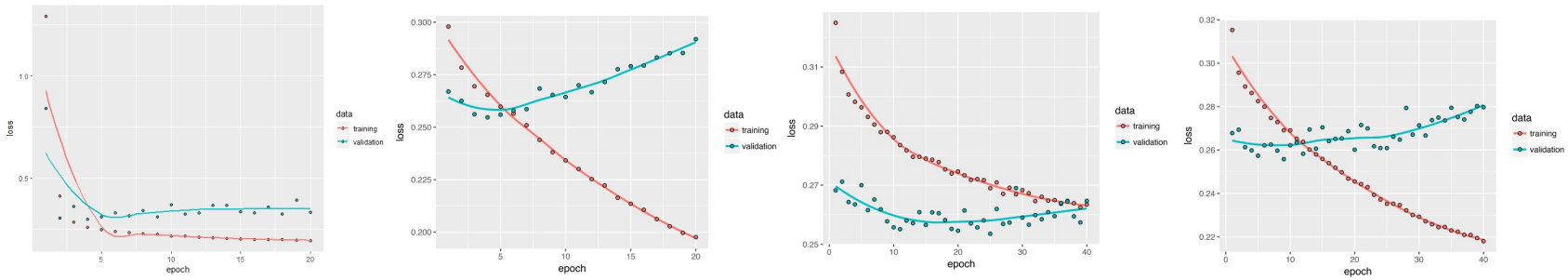



Convolutional neural network shares weights between local regions



# Weather forecasting using RNNs

- 2009-2016 weather station (Max Planck Institute, Jena, Germany)
- 14 variables/10 min interval: air T, atmospheric pressure, humidity, wind
- Baseline, Overfitting, Dropout, Increase Capacity




Credit: Chapter 5 of François Chollet's and J.J. Allaire's book, Deep Learning with R (Manning Publications)
Source: https://tensorflow.rstudio.com/blog/time-series-forecasting-with-recurrent-neural-networks.html

# Predicting wine quality using DL



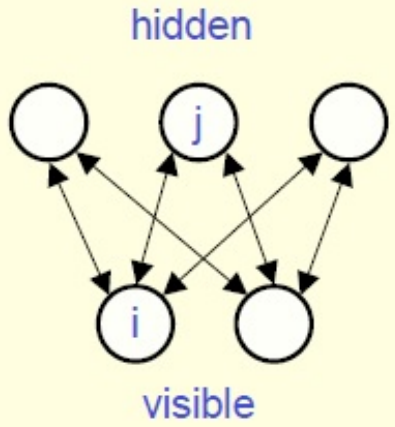
- UCIrvine ML Repository: Wine Quality Dataset (Portugal)
- Cortez et al. (2009), University of Minho, Porto, Portugal
- 4898 different wines, 12 attributes/physicochemical properties, rating (0-10 best)
- Variables: acidity, volatile acidity, citric acid, residual sugar, chlorides
- free/total sulfur dioxide, density, pH, sulfates, alcohol level
- alcohol content affects the BRIX value
- wine with residual sugar over $0.2 - 0.3$ %, high pH ($> 3.5$), low $S0_2$ susceptible to spoilage by lactic acid bacteria
- Model accuracy: 80 %
- More on wine and disease risk in my TIES talk!

# Restricted Boltzmann Machines (RBMs)

## Restricted Boltzmann Machines

(Smolensky ,1986, called them "harmoniums")

- We restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
    - We will deal with more layers later
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states.
  - So we can quickly get an unbiased sample from the posterior distribution when given a data-vector.
  - This is a big advantage over directed belief nets



# Restricted Boltzmann Machines (RBMs)

- 2-layer NNs (visible, hidden): stacked into a Deep Belief Network (DBN)
- is a log-linear Markov Random Field (MRF): linear energy function
- layers are conditionally independent
- stochastic nodes, symmetric: visible node connected to each hidden node
- weights randomly initialized, Gibbs sampling (MCMC)
- Gibbs: special case of Metropolis-Hastings: proposed moves are always accepted (acceptance probability is 1)
- generative learning (values of many varied points at once)

# RBM equations

$$p(\mathbf{x};\mathbf{W}) = \frac{1}{Z(\mathbf{W})} e^{-E(\mathbf{x};\mathbf{W})} \qquad \text{where } Z(\mathbf{W}) = \sum_{\mathbf{x}} e^{-E(\mathbf{x};\mathbf{W})}$$

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} + \eta \left. \frac{\partial L(\mathbf{W};\mathcal{X})}{\partial \mathbf{W}} \right|_{\mathbf{W}^{(\tau)}}$$

$$\begin{aligned} L(\mathbf{W};\mathcal{X}) &= \tfrac{1}{N} \textstyle\sum_{n=1}^{N} \log p(\mathbf{x}_n;\mathbf{W}) = \langle \log p(\mathbf{x};\mathbf{W}) \rangle_0 \\ &= -\langle E(\mathbf{x};\mathbf{W}) \rangle_0 - \log Z(\mathbf{W}) \end{aligned}$$

$$\frac{\partial L(\mathbf{W};\mathcal{X})}{\partial \mathbf{W}} = -\left\langle \frac{\partial E(\mathbf{x};\mathbf{W})}{\partial \mathbf{W}} \right\rangle_0 + \left\langle \frac{\partial E(\mathbf{x};\mathbf{W})}{\partial \mathbf{W}} \right\rangle_\infty$$

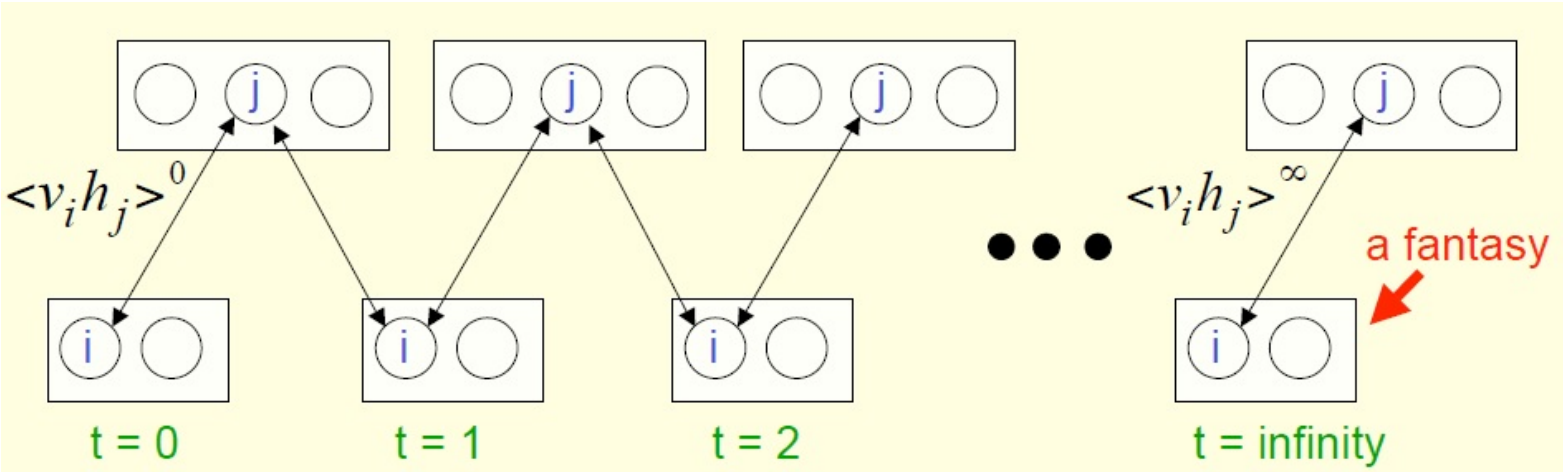
data model

$$\mathrm{KL}\left(p_0 \| p_\infty\right) = \sum_{\mathbf{x}} p_0(\mathbf{x}) \log \frac{p_0(\mathbf{x})}{p(\mathbf{x};\mathbf{W})}.$$

CD (Hinton) proposed to avoid difficulty in computing the gradient of the log-likelihood using a different function (divergence):

$$\mathrm{CD}_n = \mathrm{KL}\left(p_0 \| p_\infty\right) - \mathrm{KL}\left(p_n \| p_\infty\right).$$

# Restricted Boltzmann Machines (RBMs)

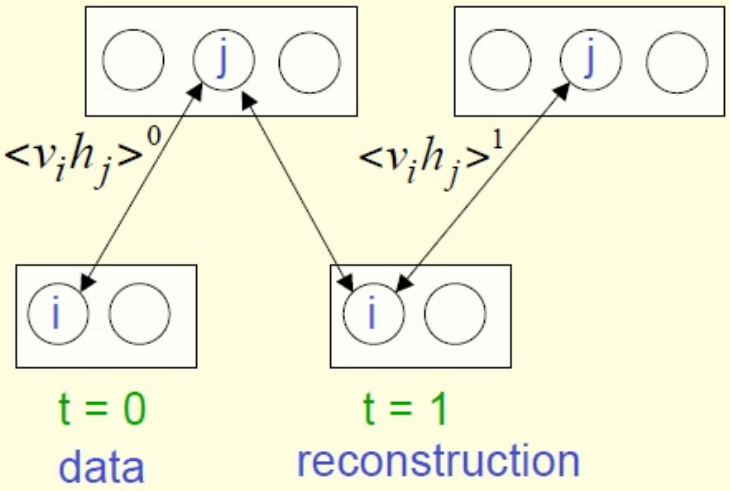



Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^\infty$$

# Restricted Boltzmann Machines (RBMs)




Start with a training vector on the visible units.

Update all the hidden units in parallel

Update the all the visible units in parallel to get a "reconstruction".

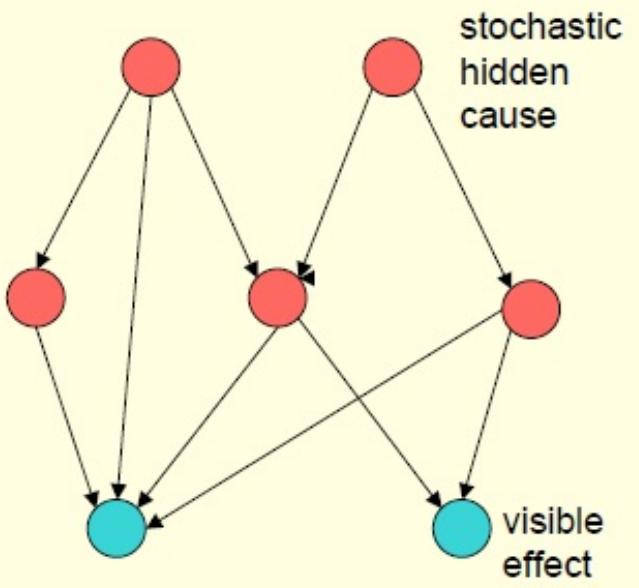
Update the hidden units again.

$$\Delta w_{ij} = \varepsilon \left( <v_i h_j>^0 - <v_i h_j>^1 \right)$$

# Deep architectures: Deep Belief Networks (DBNs)

## Belief Nets

- A belief net is a directed acyclic graph composed of stochastic variables.
- We get to observe some of the variables and we would like to solve two problems:
- The inference problem: Infer the states of the unobserved variables.
- The learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data.

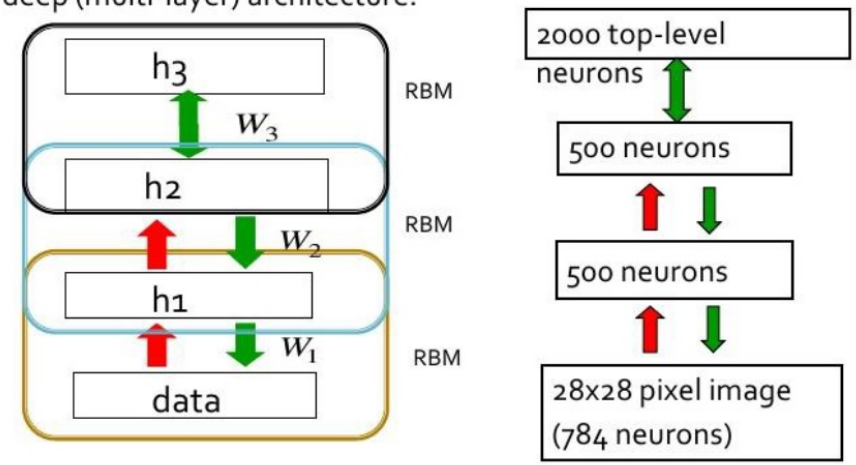



We will use nets composed of layers of stochastic binary variables with weighted connections. Later, we will generalize to other types of variable.

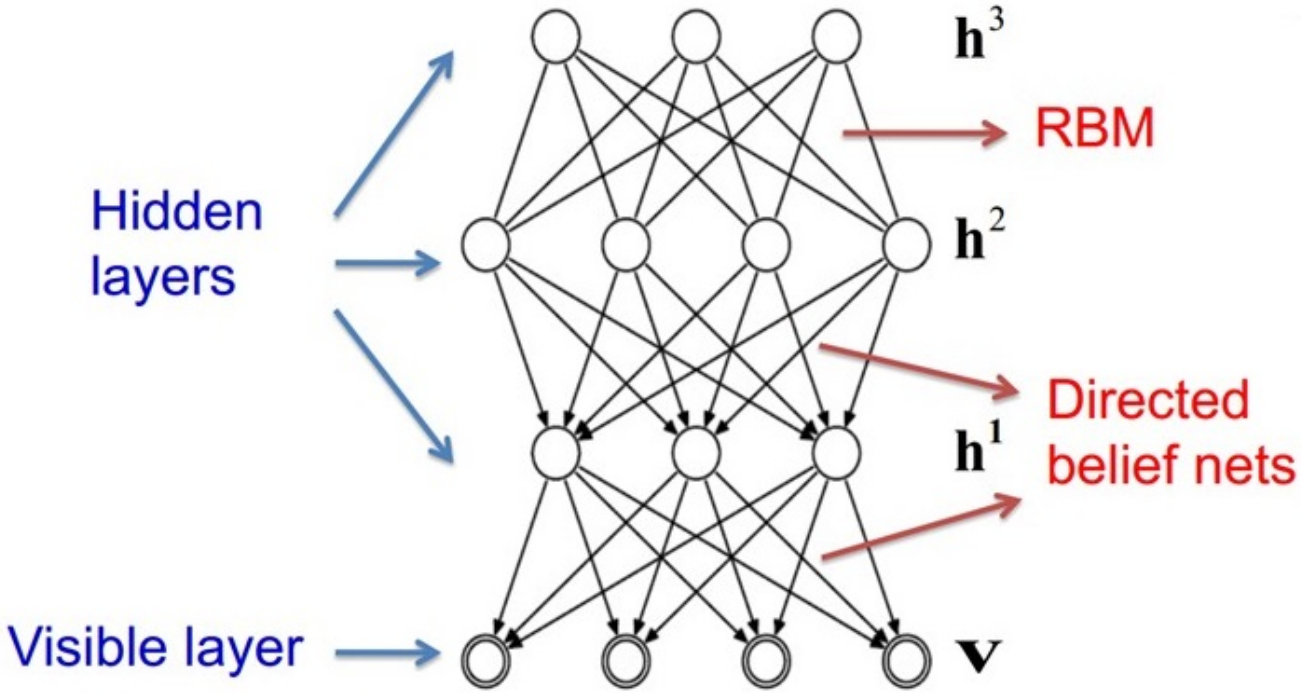
# Deep Belief Networks (DBNs)

- The Deep Belief Network, or DBN, was also conceived by Geoff Hinton.
- Used by Google for their work on the image recognition problem.
- DBN is trained two layers at a time, and these two layers are treated like an RBM.
- Throughout the net, the hidden layer of an RBM acts as the input layer of the adjacent one. So the first RBM is trained, and its outputs are then used as inputs to the next RBM. This procedure is repeated until the output layer is reached.

DBNs are stacks of restricted Boltzmann machines forming deep (multi-layer) architecture.



# Deep Belief Networks (DBNs)




$$P(\mathbf{v},\mathbf{h}^1,\mathbf{h}^2,...,\mathbf{h}^l) = P(\mathbf{v}\,|\,\mathbf{h}^1)P(\mathbf{h}^1\,|\,\mathbf{h}^2)...P(\mathbf{h}^{l-2}\,|\,\mathbf{h}^{l-1})P(\mathbf{h}^{l-1},\mathbf{h}^l)$$

# DBN learning on MNIST

- handwritten digits (LeCun et al., 1998), `http://yann.lecun.com/exdb/mnist/`
- training: 60,000, test: 10,000
- size-normalized, centered in a fixed-size image

# DBN learning on MNIST

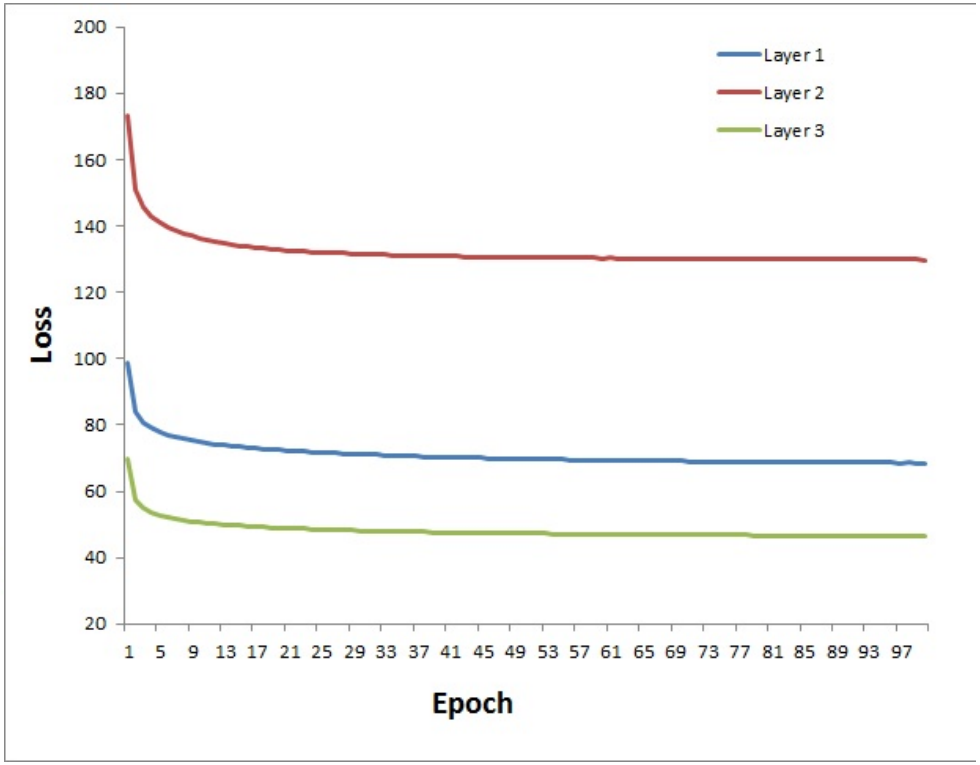
- Pre-training (DBN.py): 320.54 m ($\sim$ 5 hr)
- Fine-tuning: 106.95 m ($\sim$ 1 hr 48 m)
- Intel i7-4600U CPU 2.7 GHz, 8 GB RAM)

```
Reloaded modules: logistic_sgd, mlp, rbm, utils
loading data
building the model
getting the pretraining functions
pre-training the model

Pre-training layer 0, epoch 0, cost -98.53649282938044
Pre-training layer 0, epoch 1, cost -83.84455761672785
---
Pre-training layer 2, epoch 97, cost -46.511969084235034
Pre-training layer 2, epoch 98, cost -46.52219218396091
Pre-training layer 2, epoch 99, cost -46.511079640725626

getting the finetuning functions
The pretraining code for file DBN.py ran for 320.54m
finetuning the model
---
epoch 88, minibatch 5000/5000, validation error 1.310000 %
epoch 89, minibatch 5000/5000, validation error 1.310000 %

Optimization complete with best validation score of 1.300000 %,
obtained at iteration 225000, with test performance 1.400000 %
The fine tuning code for file DBN.py ran for 106.95m
Reloaded modules: logistic_sgd, mlp, rbm, utils,
loading data
Training epoch 0, cost is -90.17208078191256
Training epoch 1, cost is -79.79967622022373
Training epoch 2, cost is -74.41540295138371
```



# DBN learning on MNIST

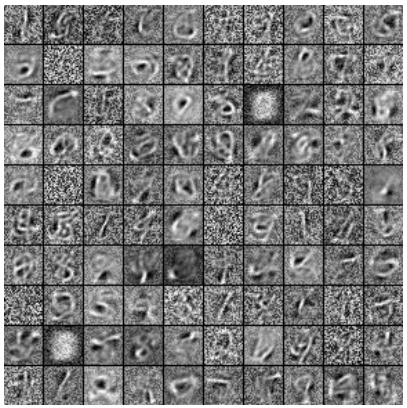


Figure: Epoch 0

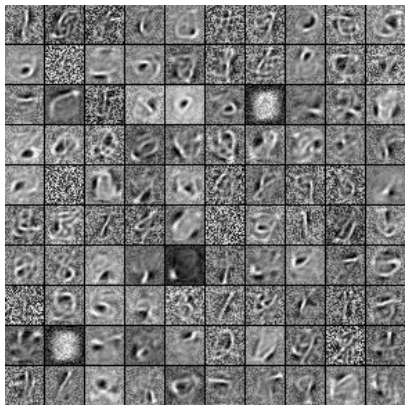


Figure: Epoch 1

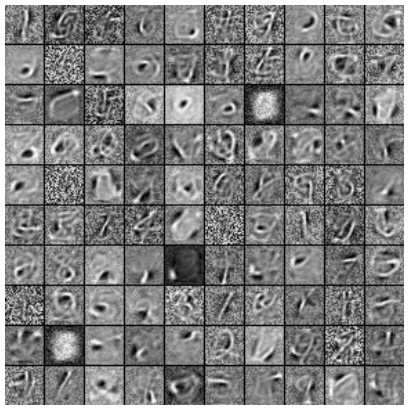


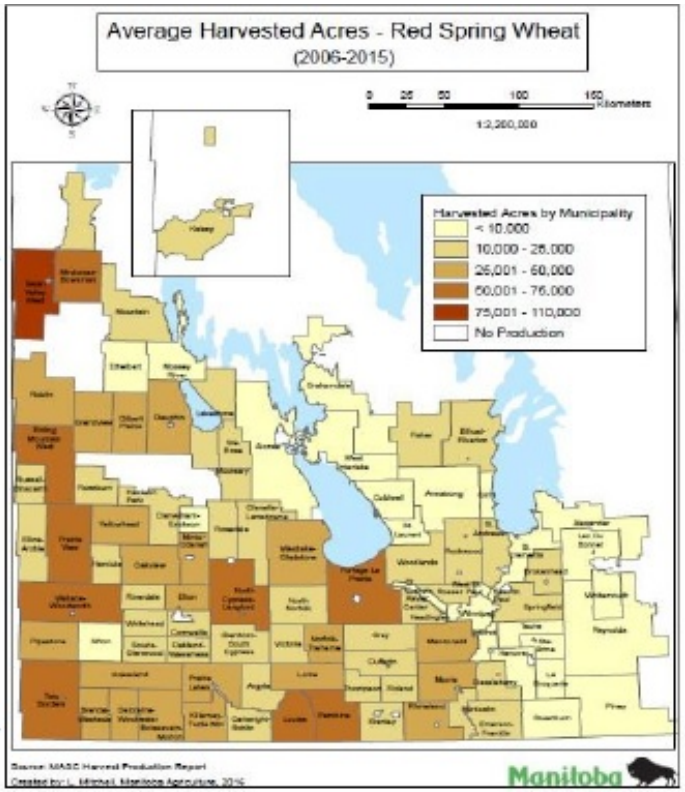
Figure: Epoch 2

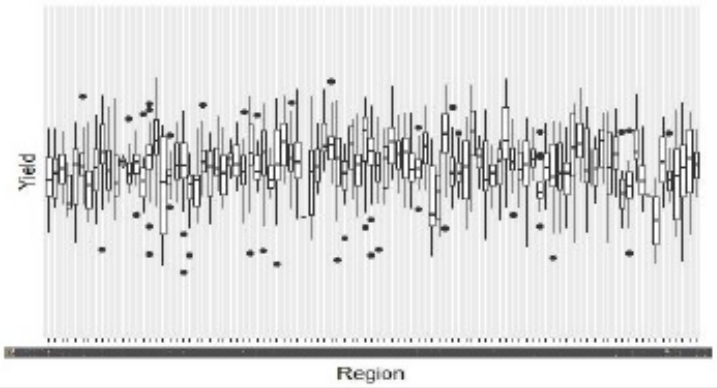
# Spatio-temporal prediction (yield, extreme indices)

## Study Region


- 99 rural municipalities (RM) (2 - 3,572) km²
- 19,238 farms
- Red spring wheat (21,605 acres)

  1.386 t/acre (50.9 bu/ac)
- 81 varieties



# Deep learning for improved insurance rate-making

- Extreme weather, such as heat and drought
- Weather index-based crop insurance
- Deep learning of agricultural insurance risk (basis risk)
- Re-stated (weighted average of yield for a given crop mix) (53 crops)
- ECCC Canada: Adjusted and Homogenized Canadian Climate Data (AHCCD) (n=38)
- European Centre for Medium-Range Weather Forecasts (ECMWF): gridded reanalysis (n=84)
- Variables: T, P, wind speed, cooling degree days (CDD), heating (HDD), growing (GDD)
- Extreme indices: average (denoted *avg*), events (i.e., minimum, maximum, denoted *min* and *max*)
- Extreme spells-waves (number of days during a defined period of extreme weather conditions, denoted *cot*)

# Benchmarking deep learning...

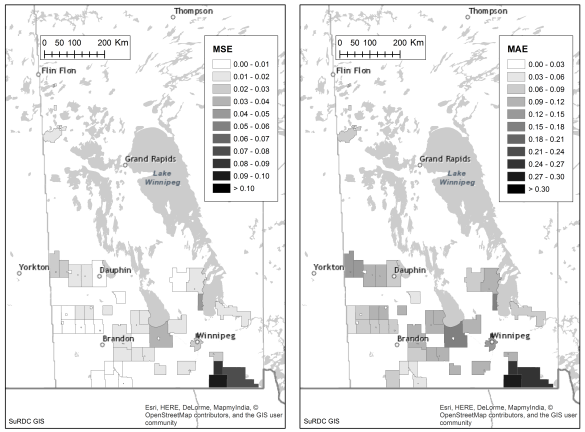
Table 1: Cross-Validated Errors Based on Different Inputs

| | Station-Based Weather Data | | Climate Reanalysis Data | |
|---|---|---|---|---|
| Method | MSE | MAE | MSE | MAE |
| GLM | 0.0539 | 0.1733 | 0.0772 | 0.2229 |
| SR | 0.0367 | 0.1506 | 0.0467 | 0.1712 |
| PCASR | 0.0277 | 0.1267 | 0.1297 | 0.1839 |
| GB | 0.0253 | 0.1228 | 0.0318 | 0.1378 |
| RF | 0.0274 | 0.1252 | 0.0308 | 0.1345 |
| DNN | 0.0230 | 0.1150 | 0.0296 | 0.1334 |
| DBN | 0.0250 | 0.1188 | 0.0290 | 0.1301 |

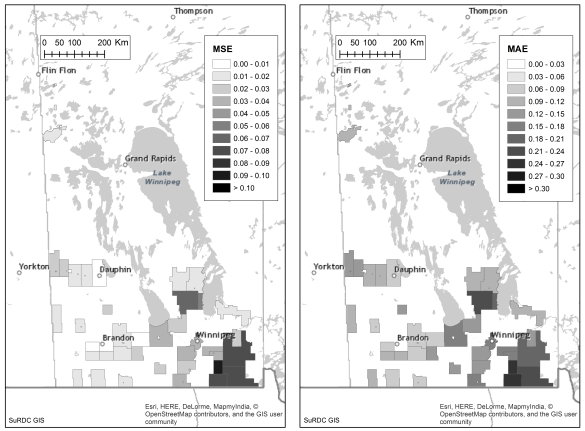
***Note:*** Methods are: generalized linear model (GLM), screening regression (SR), principal component analysis screening regression (PCASR), gradient boosting (GB), random forest (RF), deep neural network (DNN), and deep belief network (DBN). The mean squared and mean absolute errors (MSE and MAE) are calculated using formulas (7), $n = 34$; $T = 16$.

- DNN: 3 hidden layers of 5 units each,2 hidden layers with 10 units each, epochs (10,50), 10,000 validation, learning rate: 0.01, 0.05
- DBN: tanh-type, 2 hidden layers of 5 units, 5 epochs, 100 batch size, 0.5 learning rate
- DNN and DBN (stacking) out-performed all other models
- DNN: lowest prediction errors for the station data
- DBN: lowest error for the reanalysis data.
- Bagging (RF) better than boosting (GB) for reanalysis, not station data

# DNN cross-validation

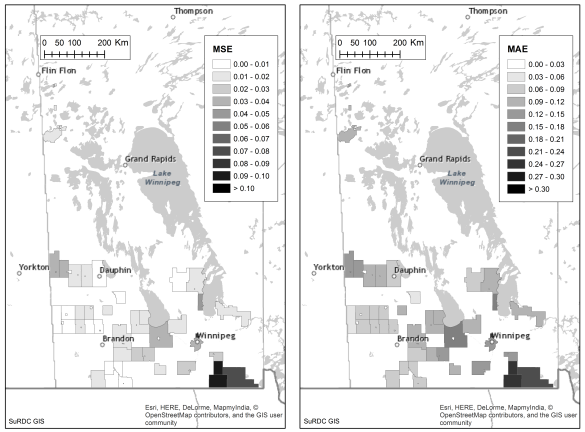



Weather station data (MSE, MAE)

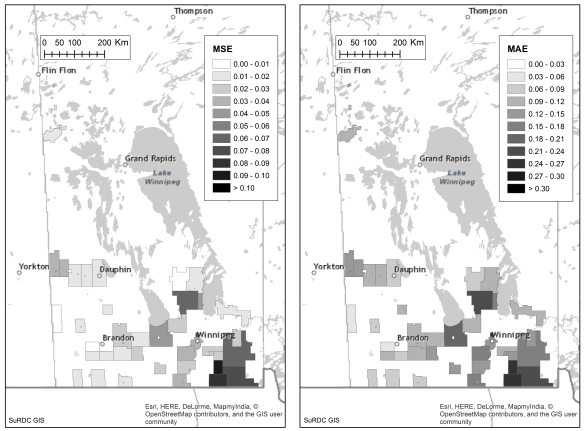



Climate reanalysis (ERA) data (MSE, MAE)

# DBN cross-validation




Weather station data (MSE, MAE)




Climate reanalysis (ERA) data (MSE, MAE)

# Reinforcement Learning (RL)

- At each step $t$ an agent executes action $a_t$, receives observation $o_t$, reward $r_t$
- Environment receives action $a_t$, emits observation $o_{t+1}$, emits reward $r_{t+1}$
- Experience is a sequence of observations, actions, rewards
- The state is a summary of experience, if fully observed: $s_t = f(o_t)$
- Policy: agent's behaviour (state-action mapping) (deterministic/stochastic)
- Value/Utility: benefit of a state and/or action
- Model: agent's learned representation of environment
- Bellmann's equation (dynamic optimization): reward/pay-off and remaining value of decision problem
- rewards are discounted by : immediate ($\lambda = 0$) vs. delayed future reward ($\lambda = 1$)

# RL approaches (Value, Policy, Model)

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Policy-based RL

- Search directly for the optimal policy $\pi^*$
- This is the policy achieving maximum future reward

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

# Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

- Once we have $Q^*$ we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[ r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Source: David Silver, Deep Reinforcement Learning, Google DeepMind

# Q-learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

- Converges to $Q^*$ using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Source: David Silver, Deep Reinforcement Learning, Google DeepMind

# Deep Q Networks (DQN)

To remove correlations, build data-set from agent's own experience

| | | | |
|---|---|---|---|
| | | $s_1, a_1, r_2, s_2$ | |
| | | $s_2, a_2, r_3, s_3$ | $\rightarrow$ $s, a, r, s'$ |
| | | $s_3, a_3, r_4, s_4$ | |
| | | ... | |
| $s_t, a_t, r_{t+1}, s_{t+1}$ | $\rightarrow$ | $s_t, a_t, r_{t+1}, s_{t+1}$ | |

Sample experiences from data-set and apply update

$$I = \left( r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

Source: David Silver, Deep Reinforcement Learning, Google DeepMind

# Deep Q Networks (DQN)

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left( r + \gamma Q(s', \underset{a'}{\text{argmax}}\, Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Duelling network: Split Q-network into two channels
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Source: David Silver, Deep Reinforcement Learning, Google DeepMind

# DQN Example

Source: https://keon.io/deep-q-learning/ Code: https://github.com/keon/deep-q-learning

# Atari (Cartpole Game) Deep Q Learning (DQL)


- `https://keon.io/deep-q-learning/`
- Hyperparameters:
- episodes - a number of games we want the agent to play
- gamma - aka decay or discount rate, to calculate the future discounted reward
- epsilon - aka exploration rate: rate in which an agent randomly decides its action rather than prediction
- epsilon decay - we want to decrease the number of explorations as it gets good at playing games
- epsilon min - we want the agent to explore at least this amount
- learning rate - Determines how much neural net learns in each iteration

# Evaluating ML methods: Performance metrics

- Learning curves (Training)
- Confusion Matrix
- Gain and Lift
- Kolmogorov-Smirnov (KS) statistic
- Precision-Recall Curve (PR)
- AOC of Receiver-Operator Characteristic (ROC)
- Gini Coefficient
- Concordant – Discordant Ratio
- Loss Functions (Cross-Validation RMSE, Binary Cross Entropy, Cross Entropy)
- Bias versus Variance Trade-off
- Drop-out (Information loss)

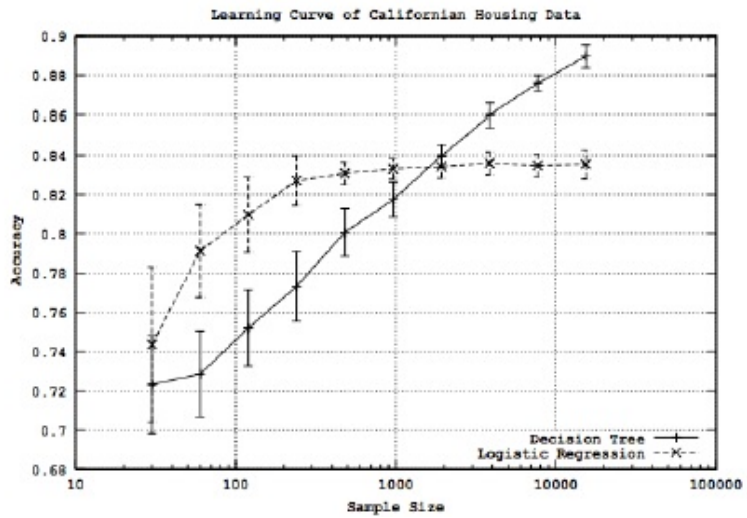
# Learning curves (Training)

- partition training data into separate training/validation sets
- larger training data (better learning process), larger validation data (more representative)
- don’t just use whatever data is available!
- must determine/acquire data for fast enough learning, real-world prediction
- "Contrastive Divergence" approximates gradient of log-likelihood for training RBMs

# Learning curves (Training)

given training/test set partition

- for each sample size $s$ on learning curve
  - (optionally) repeat $n$ times
    - randomly select $s$ instances from training set
    - learn model
    - evaluate model on test set to determine accuracy $a$
    - plot $(s, a)$ or ($s$, avg. accuracy and error bars)



# Confusion Matrix

- A confusion matrix is an *NxN* matrix, where *N* is the number of classes being predicted
- Accuracy : the proportion of the total number of predictions that were correct.
- Positive Predictive Value Precision: the proportion of positive cases that were correctly identified.
- Negative Predictive Value: the proportion of negative cases that were correctly identified.
- Sensitivity or Recall: the proportion of actual positive cases which are correctly identified.
- Specificity : the proportion of actual negative cases which are correctly identified

# Confusion Matrix

- accuracy may not be useful measure in cases where there is a large class skew
- differential misclassification costs: getting a positive wrong costs more than getting a negative wrong

## Confusion matrix for 2-class problems

| | | actual class: positive | actual class: negative |
|---|---|---|---|
| predicted class | positive | true positives (TP) | false positives (FP) |
| predicted class | negative | false negatives (FN) | true negatives (TN) |

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

# Gain and Lift

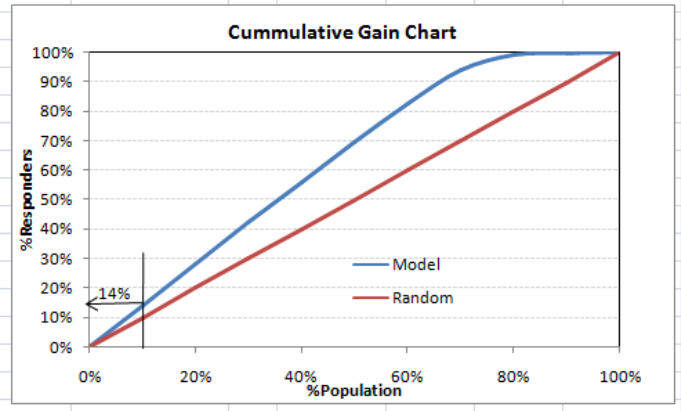
1. Calculate probability for each observation
2. Rank these probabilities in decreasing order
3. Build deciles
4. Calculate the response rate at each decile and total

- Gain and Lift chart are mainly concerned to check the rank ordering of the probabilities (quantiles)
- How well is your model segregating responders vs. non-responders
- Gain: Cumulative response per decile
- Lift: total lift and % population
- Lift / Gain charts are widely used in campaign targeting problems.
- The decile can we target customers for an specific campaign
- response expected from new target base

# Gain and Lift

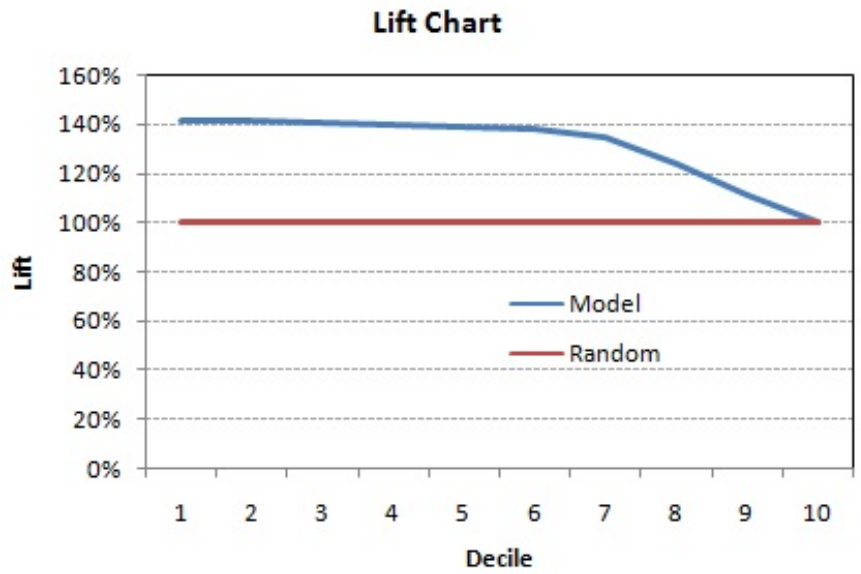
- The first decile (10% of the population) has 14% of responders, or a lift of 140% at the first decile.
- Maximum lift (at first decile): total responders of 3850 over 543 observations ($\sim$ 14.1%)

| Lift/Gain | Column Labels | | | %Rights | %Wrongs | %Population | Cum %Right | Cum %Pop | Lift @decile | Total Lift |
|---|---|---|---|---|---|---|---|---|---|---|
| Row Labels | 0 | 1 | Grand Total | 0% | 0% | 0% | 0% | 0% | | |
| 1 | | 543 | 543 | 14% | 0% | 10% | 14% | 10% | 141% | 141% |
| 2 | 2 | 542 | 544 | 14% | 0% | 10% | 28% | 20% | 141% | 141% |
| 3 | 7 | 537 | 544 | 14% | 0% | 10% | 42% | 30% | 139% | 141% |
| 4 | 15 | 529 | 544 | 14% | 1% | 10% | 56% | 40% | 137% | 140% |
| 5 | 20 | 524 | 544 | 14% | 1% | 10% | 69% | 50% | 136% | 139% |
| 6 | 42 | 502 | 544 | 13% | 3% | 10% | 83% | 60% | 130% | 138% |
| 7 | 104 | 440 | 544 | 11% | 7% | 10% | 94% | 70% | 114% | 134% |
| 8 | 345 | 199 | 544 | 5% | 22% | 10% | 99% | 80% | 52% | 124% |
| 9 | 515 | 29 | 544 | 1% | 32% | 10% | 100% | 90% | 8% | 111% |
| 10 | 540 | 5 | 545 | 0% | 34% | 10% | 100% | 100% | 1% | 100% |
| Grand Total | 1590 | 3850 | 5440 | | | | | | | |



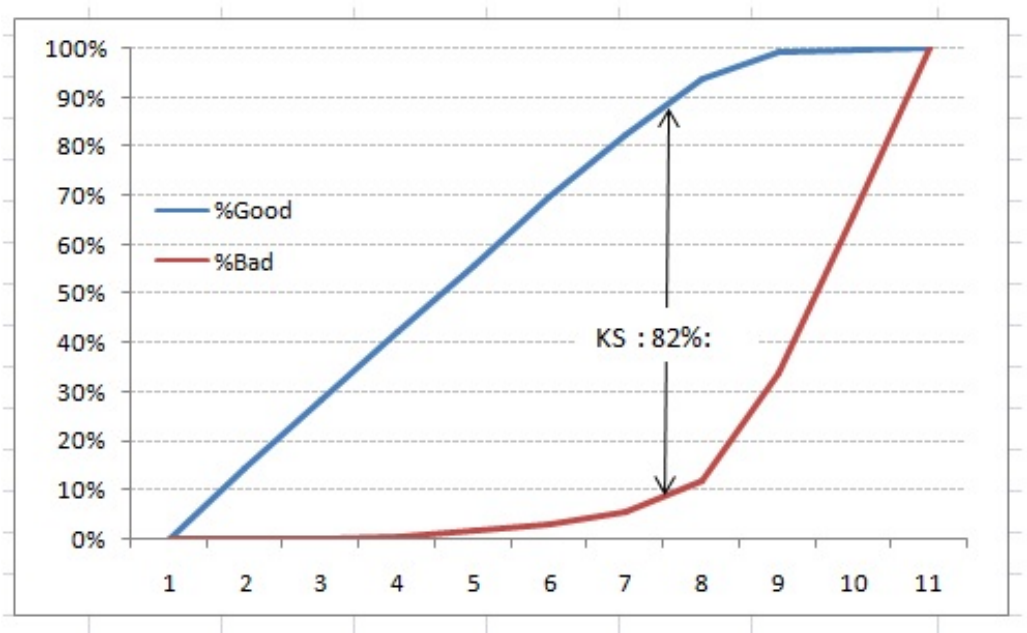
# Gain and Lift

- tells you how well is your model segregating responders from non-responders. The first decile (10% of the population) has 14% of responders or a lift of 140% at the first decile.
- Maximum lift (at first decile): total responders of 3850 over 543 observations ($\sim$ 14.1%)



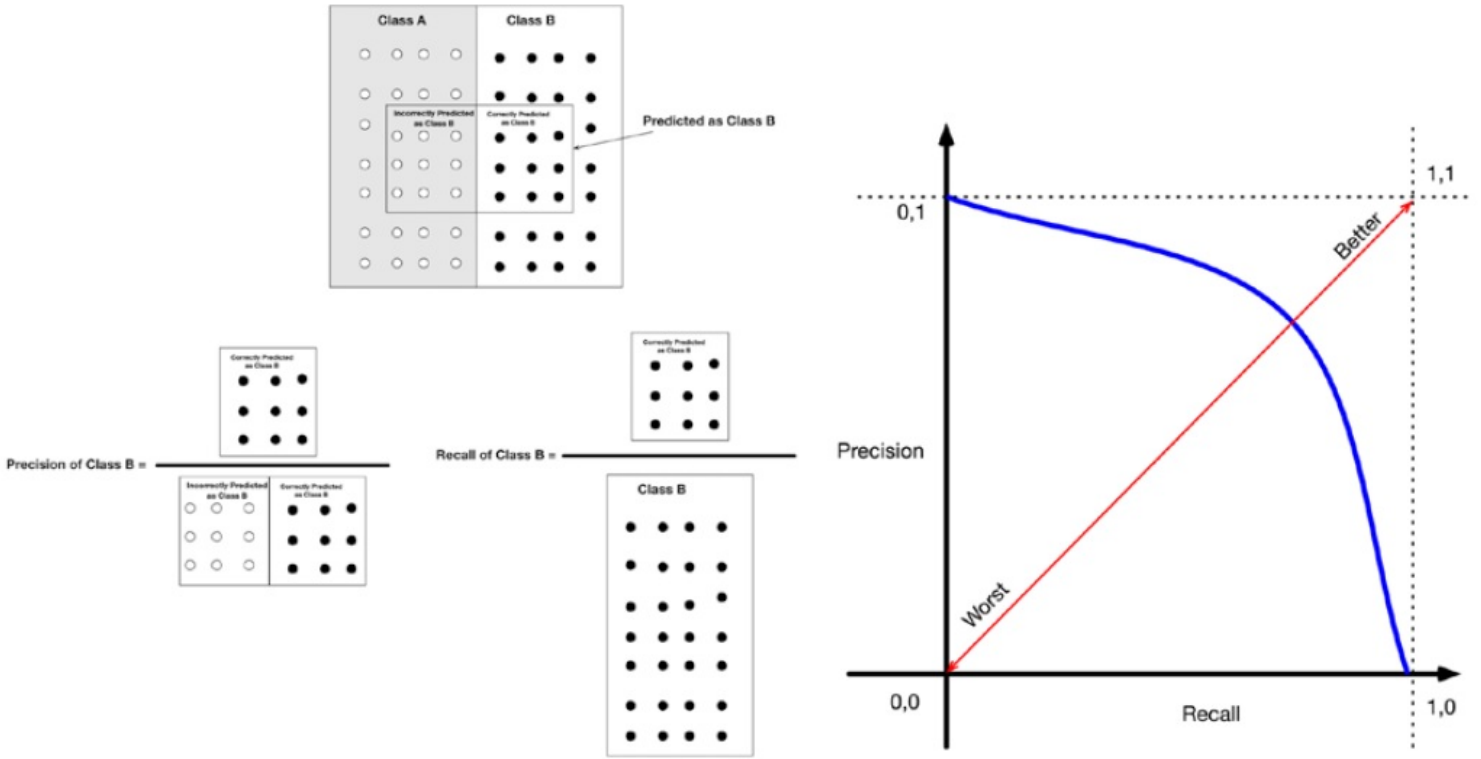
# Kolmogorov-Smirnov (KS)

- K-S measures performance of classification models (separation between positive/negative distributions)
- If a model selects cases randomly from a population, the K-S value is 0
- For most classification models, K-S [0,100], with higher values indicating a model separates cases better



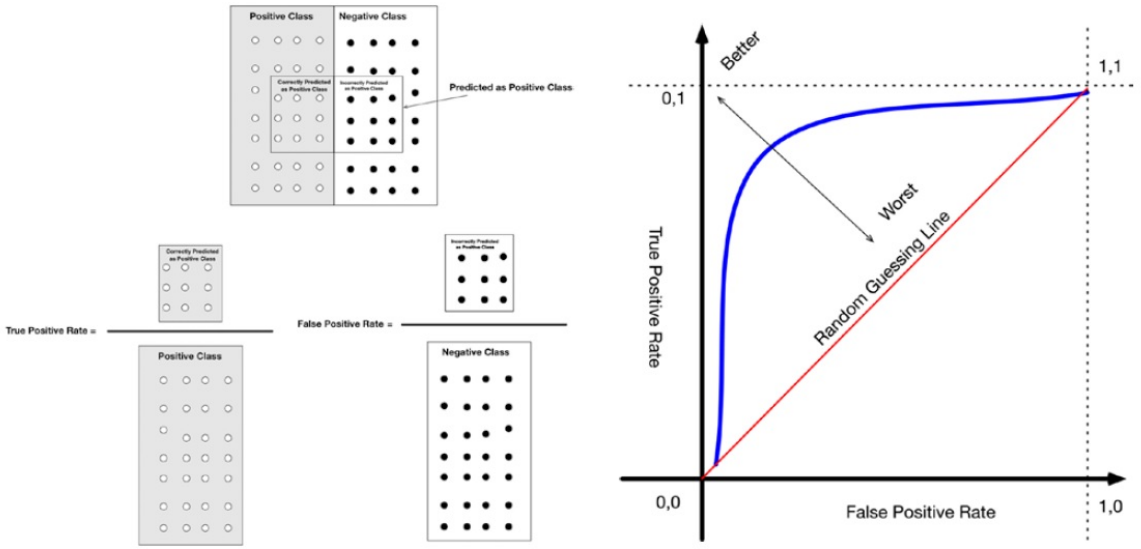
# Precision-Recall Curve (PR)

- Precision (TP/model predicted P), Recall (TP/actual P)
- Threshold on confidence when a positive instance is varied
- Identify prediction fraction of false positives (especially when lots of negative instances)
- default precision determined by the fraction of instances that are positive



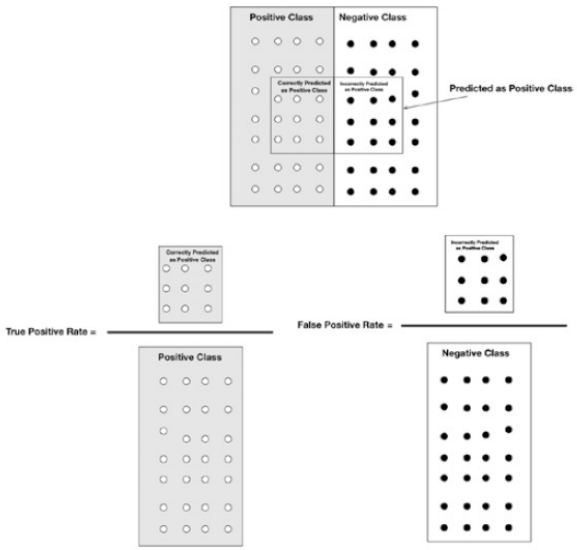
# AOC of Receiver-Operator Characteristic (ROC)

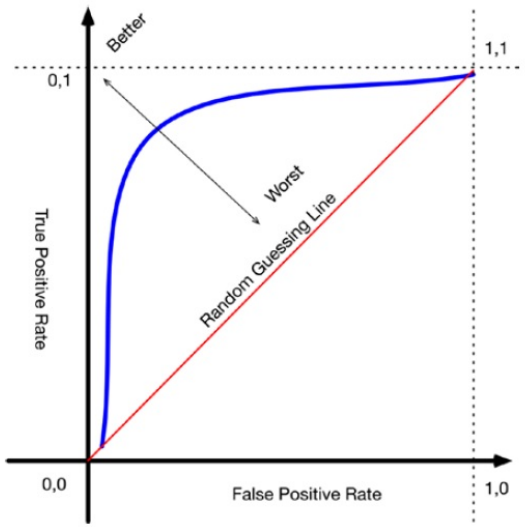
- A popular metric: sensitivity (true positive rate) versus (1- specificity) (false positive rate)
- ROC is independent of response rate (unlike Lift)
- ROC validation: in-sample versus out-of-sample model predictions
- Look at the entire curve for conclusive decision, might simply be due to over-fitting
- Different algorithms can work better in different parts of ROC space



# Gini Coefficient

- Gini coefficient: $Gini = 2AUC - 1$ (ratio of areas: ROC curve and the diagonal line vs. above triangle
- Gini above 60% is a good model



# Concordant – Discordant Ratio

- Metric for assessing model predictive power
- Concordant : pairs where the observation with the desired outcome (event) has higher predicted probability than without the outcome (non-event).
- Discordant pairs where the observation with the desired outcome (event) has lower predicted probability non-event.
- Tied : pairs where the observation with event has same predicted probability as non-event
- c statistics (AUC) : c-statistics or area-under-curve (AUC) (Concordance Percent and 0.5 times Tied Percent)
- Higher percentages of concordant pairs and lower percentages of discordant and tied pairs indicate a more desirable model
- (Concordant ratio of more than 60% is considered to be a good model)

# Loss Functions

- Typically assume error is unbiased and is normal-distributed
- RMSE is highly affected by outlier values

1. The Binary Cross entropy given by the expression

$$-\sum_{i=1}^{n} y_i \log f(x_i,\theta)+(1-y_i)\log\left(1-f(x_i,\theta)\right)$$

is the recommended loss function for binary classification. This loss function should typically be used when the Neural Network is designed to predict the probability of the outcome. In such cases, the output layer has a single unit with a suitable sigmoid as the activation function.

2. The Cross entropy function given by the expression

$$-\sum_{i=1}^{n} y_i \log f(x_i,\theta)$$

is the recommended loss function for multi-classification. This loss function should typically be used with the Neural Network and is designed to predict the probability of the outcomes of each of the classes. In such cases, the output layer has softmax units (one for each class).

3. The squared loss function given by $\sum_{i=1}^{n}\left(y-\hat{y}\right)^2$ should be used for regression problems. The output layer in this case will have a single unit.

# Bias-Variance Tradeoff

Binary partitioning: CART (Classification and Regression Trees)
Hyperplane partitioning: support vector machines (SVM)
Cluster partitioning: k-means, kNN

**Bias-Variance Trade-Off**

## Bagging

- Decrease variance: bootstrap aggregation (random forest)
- Suitable for high variance, low bias models

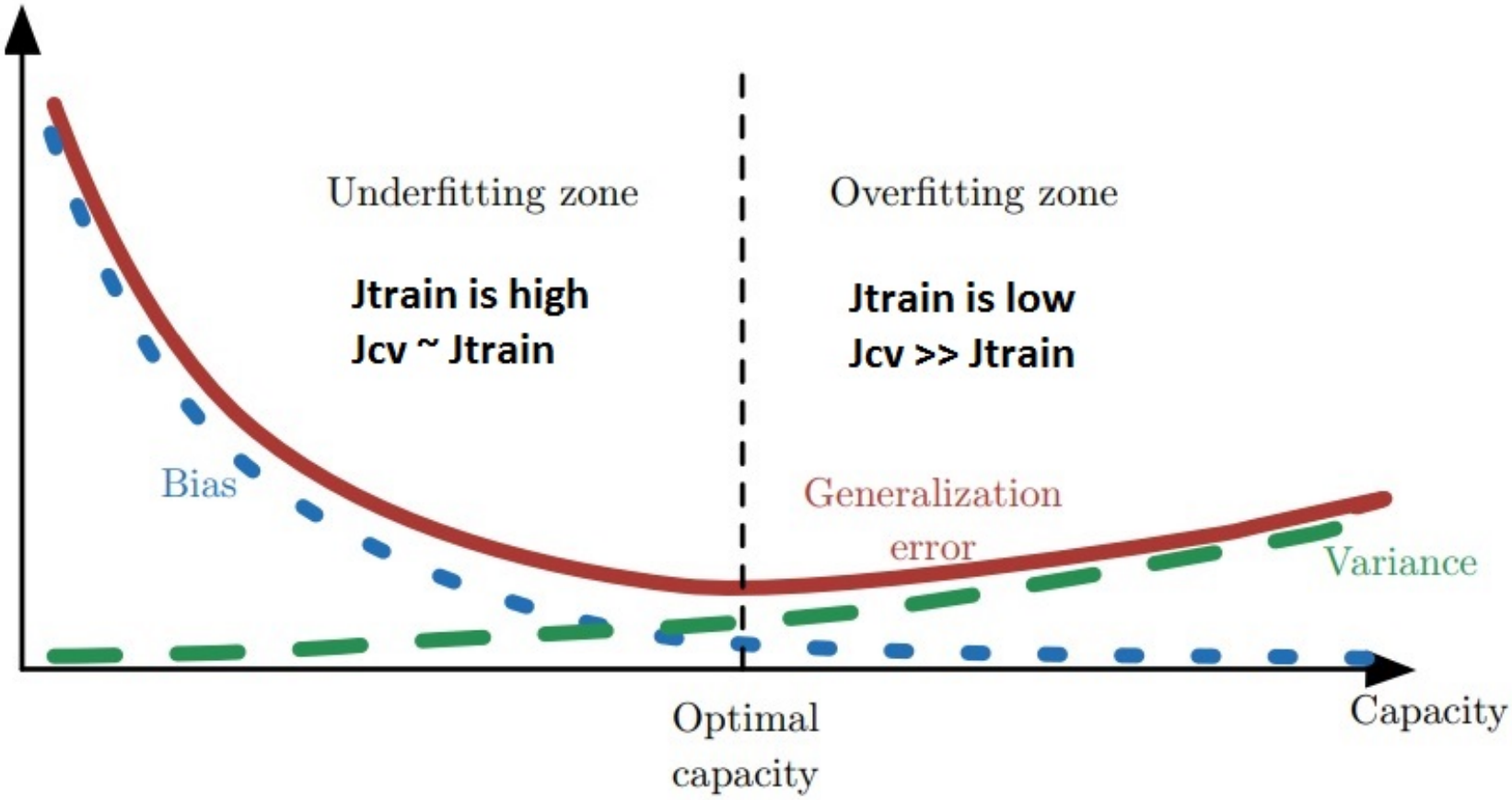
## Boosting

- Decrease bias and variance: ensemble averaging of different models/trees/rule-sets
- Focus on high error, iteratively add weak learners to build strong learners (*adaptive/Adaboost, See/C5.0 divide-conquer*)
- suitable for low variance high bias models

## Stacking

- hierarchical, deep learning with autoencoders (hidden layers, unsupervised) (*Naive Bayes, Bayesian belief networks, Extreme learning machines*)

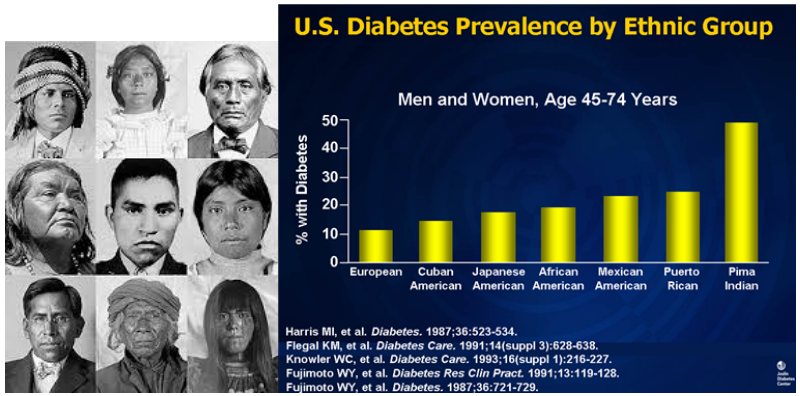
# Bias-variance trade-off



# Hyperparameters

- variables that determine network structure (hidden units) and how it is trained (learning rate)
- set before training (optimizing the weights and bias)
- tuning, optimization: random, grid-search (Cartesian)-all values, coordinate decent, tree-structured Parzen estimator
- Leading parameters:
- weight initialization (uniform distribution)
- activation function: sigmoidal (binary), softmax (multi-class), rectilinear (large networks)
- learning rate: how quicking parameters are updated, decaying rate preferred: $1 - 10^{-}06$, 0.01
- momentum: smooths gradient updates, preventing oscillations: 0.5-0.9
- number of epochs: number of time whole training set shown to network
- mini batch size: number of subsamples for parameter updates: 32-256
- training iterations: prevents overfitting, early-stopping
- weight decay: regularization (L2, L1 penalties) (negative log-prior on parameters)
- activation sparsity (hidden nodes): student-t, mean-squared, KL-divergence

# Hyperparameter Grid Search




- Pima Native Americans, Arizona (USA)
- sudden shift: traditional agricultural crops to processed foods
- highest prevalence of type 2 diabetes
- UCIrvine ML Repository dataset: 768 women with 8 characteristics:

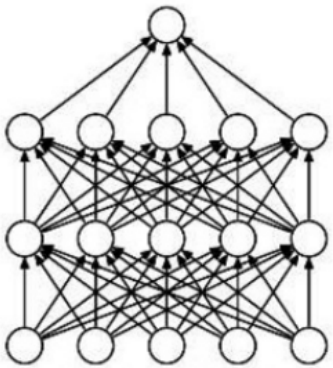
Number of times pregnant
Plasma glucose concentration a 2 hours in an oral glucose tolerance test
Diastolic blood pressure (mm Hg)
Triceps skin fold thickness (mm)
2-Hour serum insulin (mu U/ml)
Body mass index (weight in kg/(height in m)$^2$)
Diabetes pedigree function
Age (years)
Diabetes diagnosis (0,1)

# Hyperparameter Grid Search: Tuning...

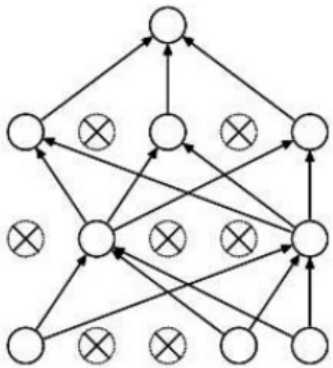
- 1: Batch size, epochs: (20, 100), 68% accuracy.
- 2: Training optimization algorithm: ADAM, 70% accuracy
- 3: Learning rate, momentum: SGD not good, learning rate 0.01, momentum of 0.0, 68%
- 4: Network weight initialization: uniform weight initialization, 72%
- 5: Neuron activation function: rectifier activation function, 72%
- 6: Dropout regularization: dropout rate of 0.2% , maxnorm weight constraint of 4, 72%
- 7: Number of hidden layer neurons: 5 neurons, 71%

# Drop-out (Information loss)

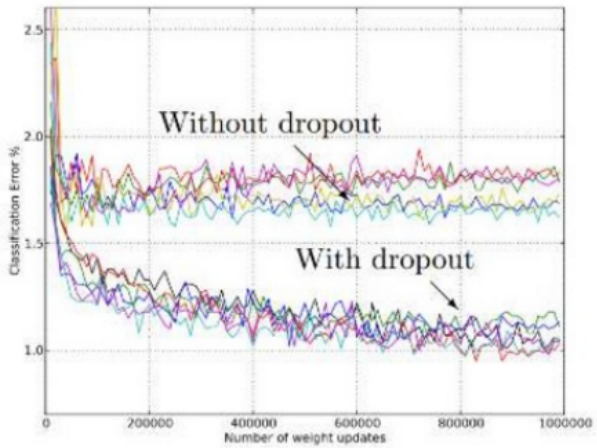
- random neurons are cancelled, use small dropout value: 20-50%, use on larger networks



(a) Standard Neural Net



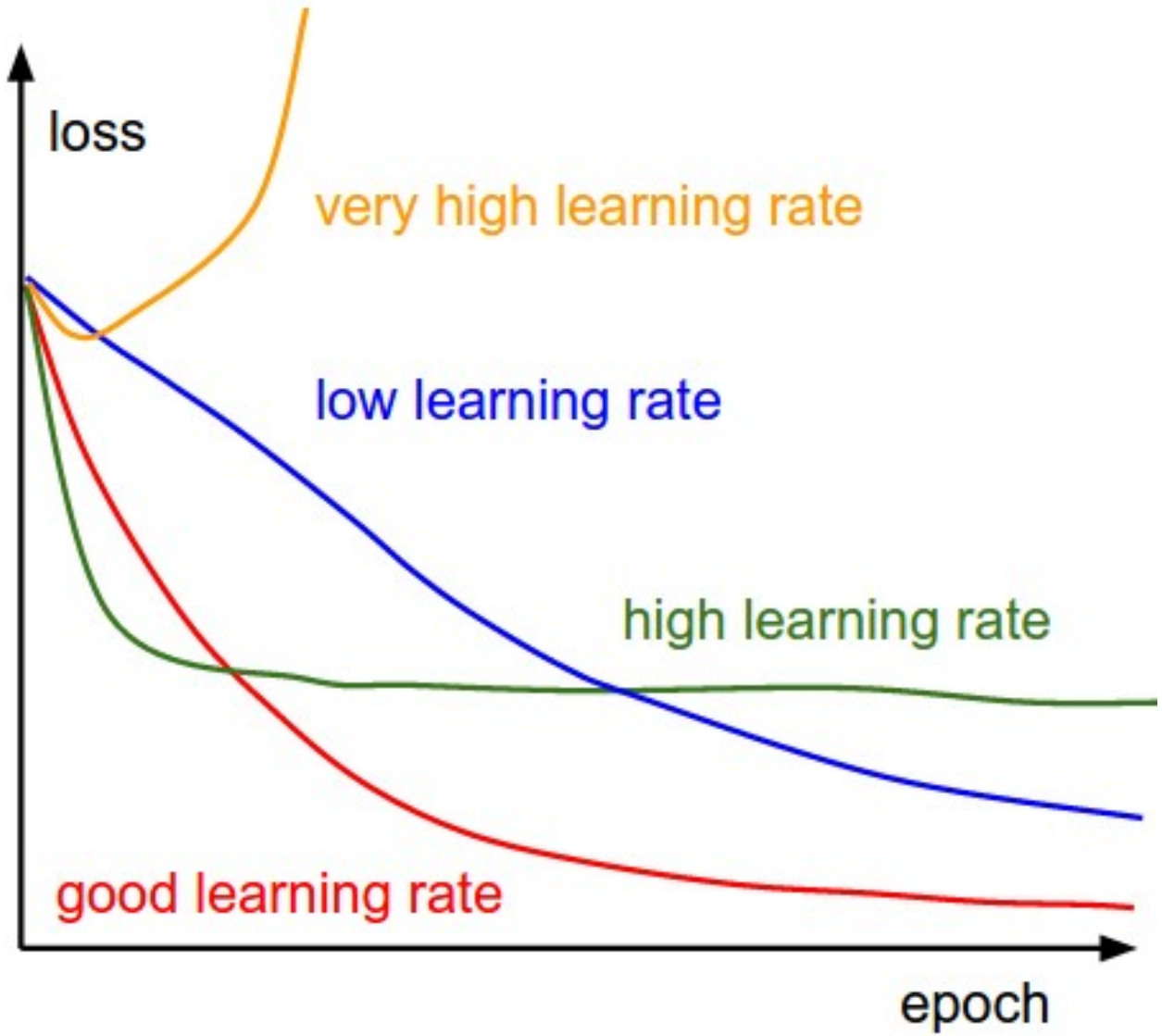
(b) After applying dropout.



# Learning rate

- Learning rate is a hyper-parameter that controls adjusting weights vs. loss gradient.
  - $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$
- train the model initially with a very low learning rate and increasing it linearly/exponentially
- loss vs. learning rate: loss stops decreasing and starts to increase
- traverse saddle plateaus: restart from last iteration, with triangular decrease, cosine cyclic function

# Learning rate



# Confronting Deep Uncertainties, Integrated Risks

- Most troubling risk challenges of our time are characterized by deep uncertainties
- (e.g., climate change, epidemics, financial systems, vulnerable ecosystems)
- Well-validated, trustworthy risk models of future consequences and alternatives not available
- Relevance of past data for predicting future outcomes is in doubt: "new normal"
- Experts disagree about the probable consequences of alternative policies
- Unwarranted consensus replaces acknowledgment of uncertainties and information gaps
- policy makers divided about what actions to take to reduce risks and increase benefits

# Confronting Deep Uncertainties, Integrated Risks

**Table I.** Methods for Decision Making with Unknown Models

| Method | Model Generation | Optimization/Adaptation | Combination |
|---|---|---|---|
| Expected utility/SEU theory | One model specified | Maximize expected utility (over all acts in the choice set, $A$) | None |
| Multiple priors, models, or scenarios; robust control, robust decisions | Identify multiple priors (or models or scenarios, etc.) e.g., all models close to a reference model (based on relative entropy) | Maximize the return from the worst-case model in the uncertainty set | Penalize alternative models based on their dissimilarity to a reference model |
| Robust optimization | Use decisionmaker's risk attitude, represented by a coherent risk measure, to define the uncertainty set | Optimize objective function while satisfying constraints, for all members of uncertainty set | None |
| Average models | Use multiple predictive (e.g., forecasting) models | None | Simple average or weighted majority |
| Resampling | Create many random subsets of original data and fit a model to each | Fit models using standard (e.g., least squares, maximum likelihood) statistical criteria | Create empirical distribution of estimates |
| Adaptive boosting (AdaBoost) | Iteratively update training data set and fit new model | Reweight past models based on predictive accuracy | Use weights to combine models |
| Bayesian model averaging (BMA) | Include all models that are consistent with data based on likelihood | Condition model probabilities on data | Weight models by their estimated probabilities |
| Low-regret online decisions | Set of experts, models, scenarios, etc. is given, $\{M_1, M_2, \ldots, M_n\}$ | Reduce weights of models that make mistakes | Weighted majority or selection probability |
| Reinforcement learning (RL) for MDPs: UCRL2 | Uncertainty set consists of confidence region around empirical values | Approximately solve Bellman equations for most optimistic model in uncertainty set to determine next policy | Update from episode to episode based on new data |
| Model-free reinforcement learning (RL) for MDPs: SARSA | No model used (model-free learning) | Approximately solve Bellman equations for unknown model | Update value estimates & policies based on new data |

Credit: L.A. Cox (2012) Confronting Deep Uncertainties in Risk Analysis, *Risk Analysis*, 32:10

# Advancing DL architectures...

- Stochastic weight averaging etc..
- Reinforcement Learning (Value, Policy, Model)-Based
- Bayesian deep networks (BDN) to open up black box
- Preprint on: “Tackling Climate Change with Machine Learning”
  https://arxiv.org/abs/1906.05433?utm_source=Machine+Learning+Yearning&utm_campaign=9bdd622a51-EMAIL_CAMPAIGN_2019_05_22_06_05_COPY_01&utm_medium=email&utm_term=0_bd0078304d-9bdd622a51-115774517

# Future Outlook

- Don't loose focus/sight of goal (not about fancy computer code tricks)
- Don't just collect Big/massive data (unobserved is equally/more important than observed)
- But do need shareable code libraries, pre-trained datasets (environmental science)
- Complex, adaptive, chaotic: higher complexity than set of fixed/mobile imagery
- Ecosystems exhibit high dimensional spatio-temporal chaos (time-dependent, external perturbations)
- Focus on deep AI learning architectures to map known/unknown model/data uncertainties
- Bayesian deep networks *inter*-variable is broader than *intra*-variable complexity)
- Broader Algorithms: Foresight, Predictability, Context Self-Awareness
- Integrated risk (multiple, synchronous/asynchronous interacting deep belief nets)

# Online Community

- Frameworks, Tools `https://www.nodesagency.com/65-frameworks-tools-for-machine-learning/`
- Industry `www.nvidia.com/en-us/deep-learning-ai/`
- Google `https://plus.google.com/communities/112866381580457264725`

# Online Community

- TensorFlow Blog: `https://developers.googleblog.com/search/label/TensorFlow`
- Keras Blog: `https://blog.keras.io/`
- OpenAI Blog: `https://blog.openai.com/`
- R Bloggers: `https://www.r-bloggers.com/`

# Fundamental Theory, Practical Applications


- online discussion forums, blogs, courses, data libraries...

# Selected - Online Courses

- AI Expert, Andrew Ng
  `www.coursera.org/specializations/deep-learning`
- AI book `www.mlyearning.org`
- Stanford `https://stats385.github.io/lecture_slides`
- Datasets, tutorials `http://deeplearning.net`
- DL Course (Google) `https://www.udacity.com/course/deep-learning--ud730`
- Practical DL for Coders (fast.ai) `http://course.fast.ai/`
- Towards Data Science
  `https://towardsdatascience.com/`
- NVIDIA Deep Learning GPU Training System (DIGITS)
  `https://developer.nvidia.com/digits`

# Acknowledgement/s


- Mix of own content and other DL expert shared slides, open source code/data
- Lucas Masuch (Lead Software Engineer - Machine Intelligence - SAP) (DL - a visual introduction)
- David Silver (Google Deepmind) - Reinforcement learning
- Geoffrey Hinton (Canadian Institute for Advanced Research)
- (Computer Science University of Toronto)