# Lecture 3

# Lecture 3

1 Deep learning introduction:

2. Neural networks error backpropagation,

3. activation function,

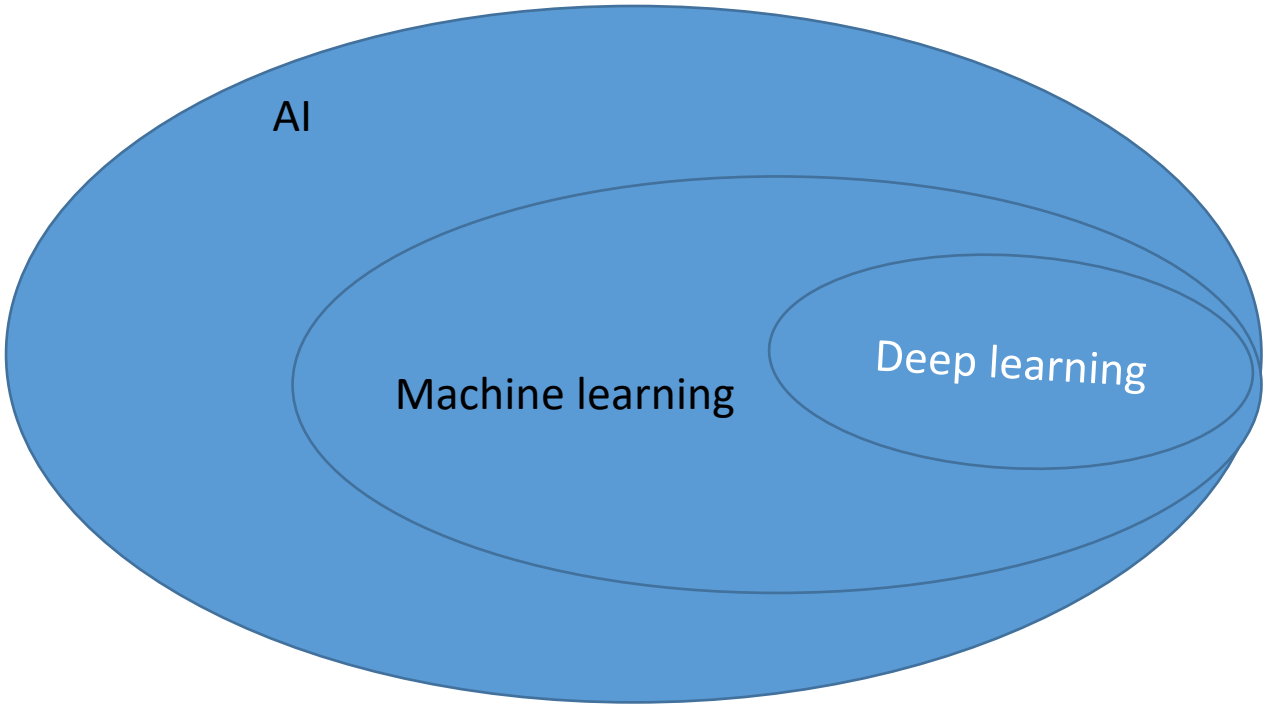
4. stochastic gradient descent,

5. softmax regression,

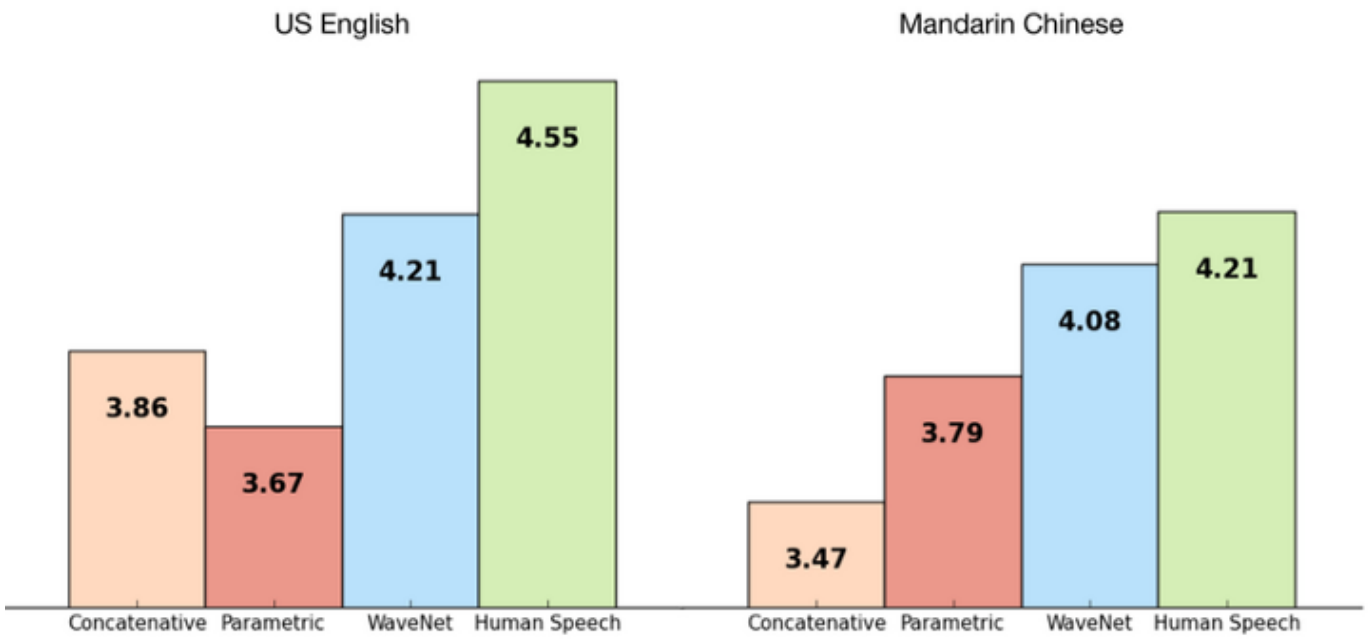
6. regularization,  sparse penalty,  dropout, loss function

7 Optimization,

# Deep learning & AI



# Deep learning achievements



- Computer GO games
- Great developments in text, voice, and computer vision technologies

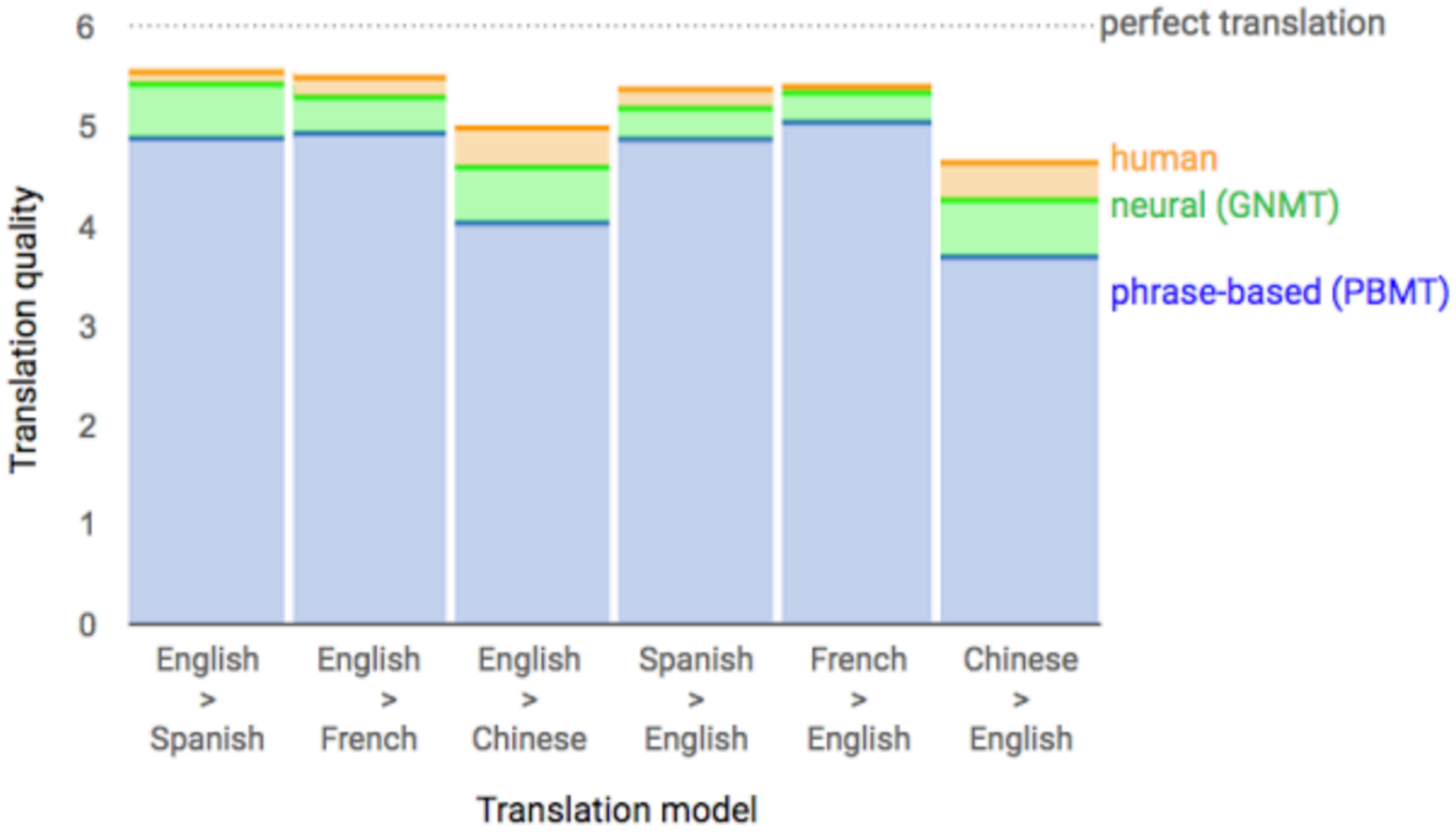
# Image understanding

- self-driving cars, which identify and respond to what they “see,” enabling an entirely new industry.

- Deep learning models have used detailed image analysis in health care to greatly improve disease diagnoses, including diabetic retinopathy and some cancers.

- **Google Neural Machine Translation**

# Machine translation achievement




Google Sequence-to-Sequence based model performance. Source

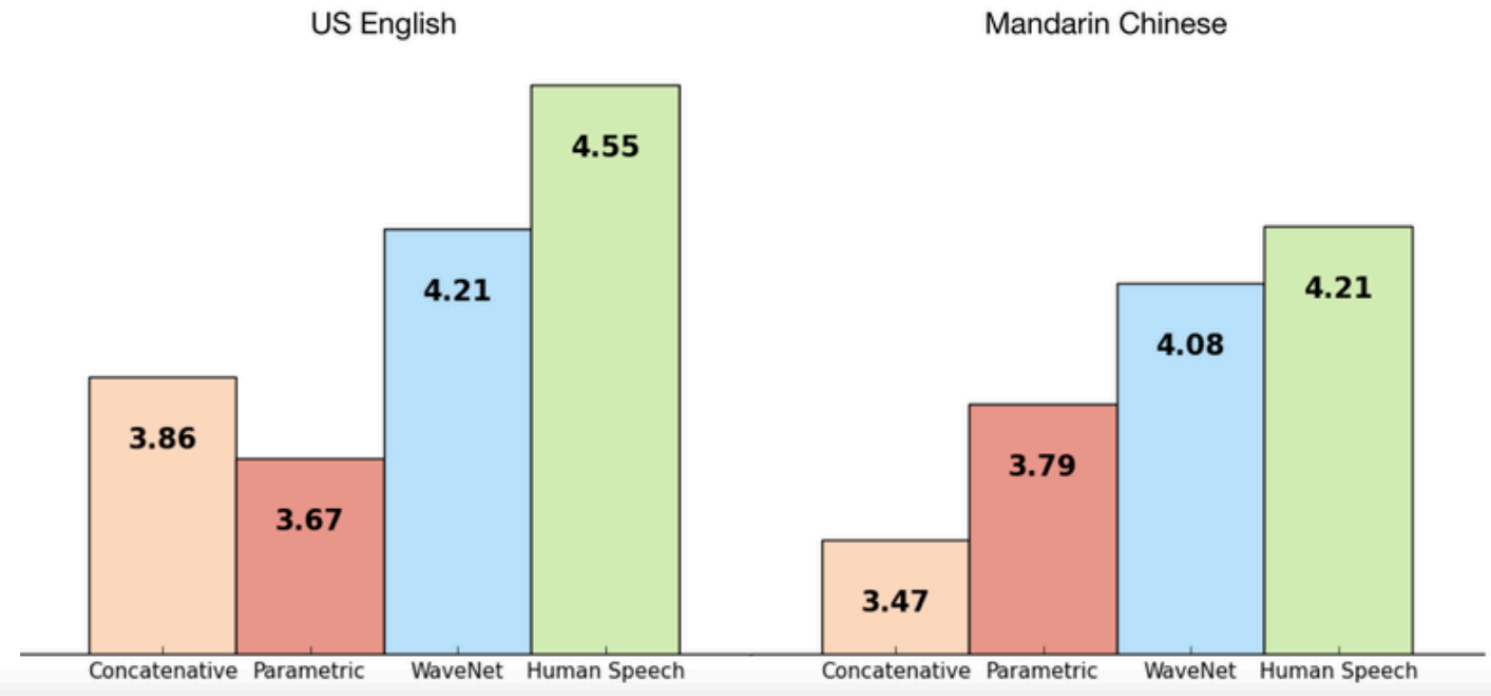
# Speech interaction

- Amazon’s Alexa, Apple’s Siri, Microsoft’s Cortana, or the many voice-responsive features of Google. Chinese search giant Baidu working on speech interface.

- Google use Chatbot to negotiate with a deal.

# Text-2-speech generation

- end-to-end training on waveNet for Text-to-Speech



# Lip reading

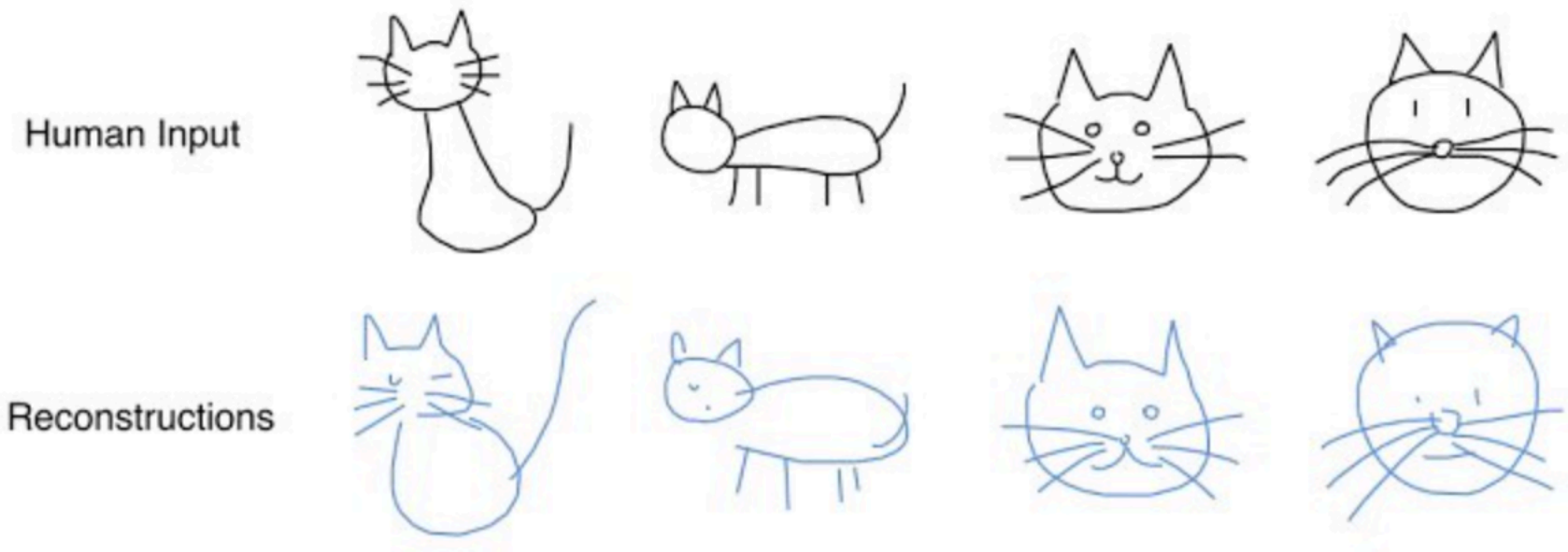
- Google Deepmind, in collaboration with Oxford University

- trained on a television dataset, was able to surpass the professional lip reader from the BBC channel.

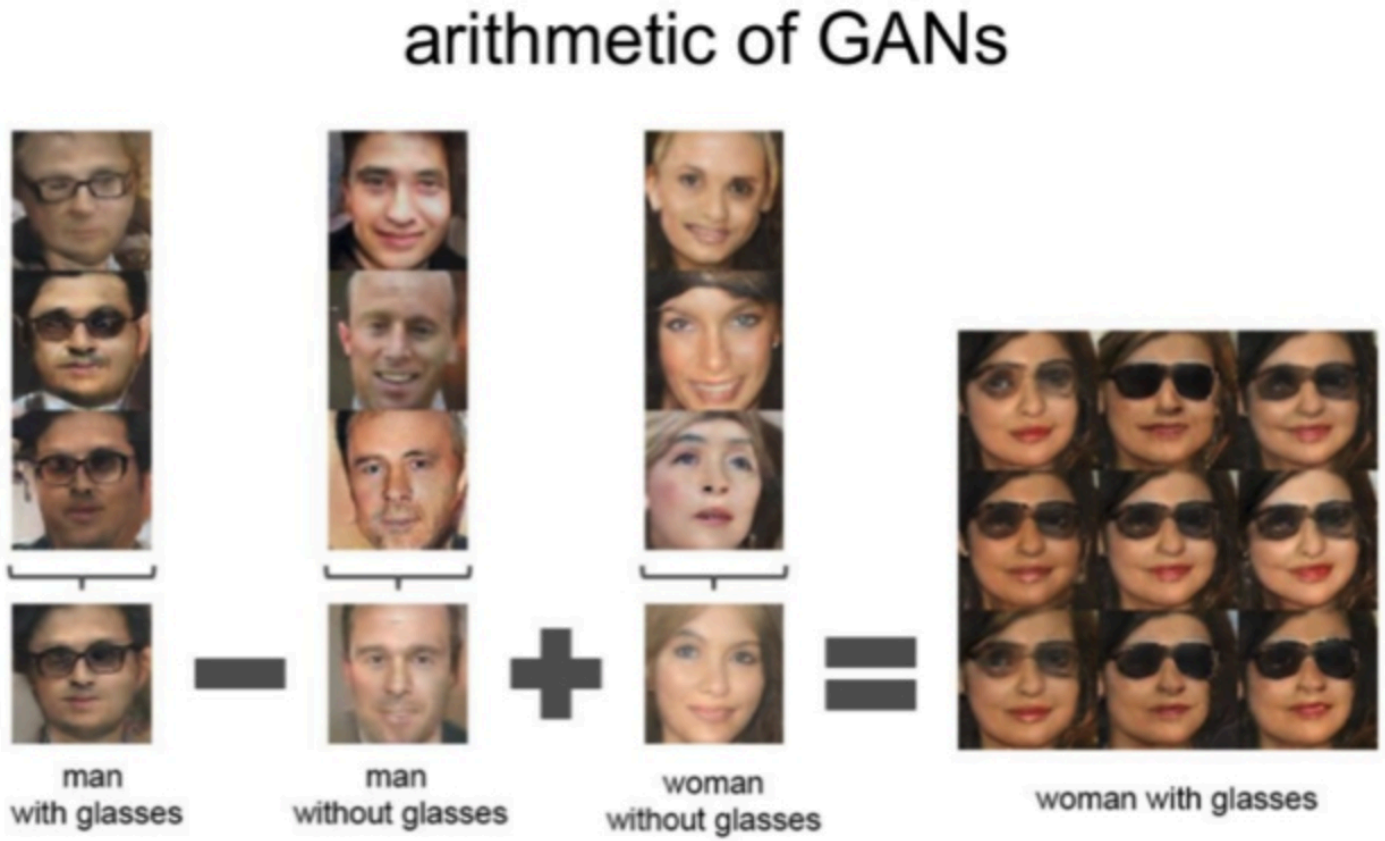
# OCR: Google Maps and Street View

- Google Brain Team reported on how they introduced a new OCR (Optical Character Recognition) engine into its Maps, through which street signs and store signs are recognized.

# SketchRNN: teaching a machine to driving



# Arithmetic logic on images based on GAN



# FOUR TECH GIANTS GET SERIOUS ABOUT DEEP LEARNING

- GOOGLE
- Google launched the deep-learning-focused Google Brain project in 2011, introduced neural nets into its speech-recognition products in mid-2012, and retained neural nets pioneer Geoffrey Hinton in March 2013. It now has more than 1,000 deep-learning projects underway, it says, extending across search, Android, Gmail, photo, maps, translate, YouTube, and self-driving cars. In 2014 it bought DeepMind, whose deep reinforcement learning project, AlphaGo, defeated the world's go champion, Lee Sedol, in March, achieving an artificial intelligence landmark.
- MICROSOFT
- Microsoft introduced deep learning into its commercial speech-recognition products, including Bing voice search and X-Box voice commands, during the first half of 2011. The company now uses neural nets for its search rankings, photo search, translation systems, and more. "It's hard to convey the pervasive impact this has had," says Lee. Last year it won the key image-recognition contest, and in September it scored a record low error rate on a speech-recognition benchmark: 6.3%.
- FACEBOOK
- In December 2013, Facebook hired French neural nets innovator Yann LeCun to direct its new AI research lab. Facebook uses neural nets to translate about 2 billion user posts per day in more than 40 languages, and says its translations are seen by 800 million users a day. (About half its community does not speak English.) Facebook also uses neural nets for photo search and photo organization, and it's working on a feature that would generate spoken captions for untagged photos that could be used by the visually impaired.
- BAIDU
- In May 2014, Baidu hired Andrew Ng, who had earlier helped launch and lead the Google Brain project, to lead its research lab. China's leading search and web services site, Baidu uses neural nets for speech recognition, translation, photo search, and a self-driving car project, among others. Speech recognition is key in China, a mobile-first society whose main language, Mandarin, is difficult to type into a device. The number of customers interfacing by speech has tripled in the past 18 months, Baidu says.

# Deep learning in medicine

- Startup **Enlitic** uses deep learning to analyze radiographs and CT and MRI scans. CEO Igor Barani, formerly a professor of radiation oncology at the University of California in San Francisco, says Enlitic's algorithms outperformed four radiologists in detecting and classifying lung nodules as benign or malignant. (The work has not been peer reviewed, and the technology has not yet obtained FDA approval.)
- **Merck** is trying to use deep learning to accelerate drug discovery, as is a San Francisco startup called **Atomwise**. Neural networks examine 3D images—thousands of molecules that might serve as drug candidates—and predict their suitability for blocking the mechanism of a pathogen. Such companies are using neural nets to try to improve what humans already do; others are trying to do things humans can't do at all. Gabriel Otte, 27, who has a Ph.D. in computational biology, started **Freenome**, which aims to diagnose cancer from blood samples. It examines DNA fragments in the bloodstream that are spewed out by cells as they die. Using deep learning, he asks computers to find correlations between cell-free DNA and some cancers. "We're seeing novel signatures that haven't even been characterized by cancer biologists yet," says Otte.
- When Andreessen Horowitz was mulling an investment in Freenome, AH's Pande sent Otte five blind samples—two normal and three cancerous. Otte got all five right, says Pande, whose firm decided to invest.
-

# What is deep Learning?

- A merge of traditional machine learning, neural networks, and statistical learning.
- Boosted by big data analysis and GPU hardware of Nvidia.
- Initiated around late 2000s, and by “learning for *deep belief nets” paper of Hinton, Osindero and Teh in 2006.*
- Pro and con:
- Pro: Show many successful and break thru in speech recognition, image recognition, natural language processing, drug discovery, etc.
- Con: Lack of accountability. Theory, ad hoc. No explanation on logical inference and causality. No guarantee converge, speed, and approximation accuracy.

# Reading Resources of deep learning

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, An MIT Press book

- http://ufldl.stanford.edu/tutorial

- https://en.wikipedia.org/wiki/Deep_learning

-

# Open source tools for deep learning

- **TensorFlow** by Google.
- **Keras** by François Chollet.
- **CNTK** by Microsoft.
- **MXNET** by Distributed (Deep) Machine Learning Community. Adapted by Amazon.
- **Theano** by Université de Montréal.
- **Torch** by Ronan Collobert, Koray Kavukcuoglu, Clement Farabet. Widely used by Facebook.

# Deep Learning libraries

- Most popular C++ and Python based

| | CORE | TEST BATCH (MS) | Multi-GPU | DISTRIBUTED | advantage |
|---|---|---|---|---|---|
| CAFFE | C++ | 651.6 | O | X | PRE-TRAINED MODELS |
| NEON | python | 386.8 | O | X | SPEED |
| TENSOR FLOW | C++ | 962 | O | O | HEREOGENEOUS DISTRIBUTED |
| THEANO | python | 733.5 | X | X | EASE TO USE WITH WRAPPER: KERAS, LASAGNE, BLOCK |
| TORCH | lu | 506.6 | 0 | X | FUNCTIONAL EXTENSION |

**Table 3.** Comparison of Learning-based action representation approaches.

| Method | Feature Type | Performance (%) |
|---|---|---|
| KTH [35] | | |
| Wang et al. 2012 [125] | Dictionary Learning | 94.17 |
| Liu et al. 2016 [131] | Genetic Programming | 95.0 |
| Le et al. 2011 [138] | Subspace analysis | 93.9 |
| Ballan et al. 2012 [141] | Codebook | 92.66 |
| Hasan and Chowdhury 2014 [140] | DBNs (Deep Belief Networks) | 96.6 |
| Ji et al 2013 [149] | 3D CNN (Convolutional Neural Networks) | 90.2 |
| Zhang and Tao 2012 [151] | Slow Feature Analysis (SFA) | 93.50 |
| Sun et al. 2014 [152] | Deeply-Learned Slow Feature Analysis (D-SFA) | 93.1 |
| Alfaro et al. 2016 [163] | Sparse coding | 97.5% |
| HDMB-51 [47] | | |
| Liu et al. 2016 [131] | Genetic Programming | 48.4 |
| Simonyan and Zisserman 2014 [148] | CNN | 59.4 |
| Luo et al. 2015 [164] | Actionness | 56.38 |
| Wang et al. 2015 [165] | convolutional descriptor | 65.9 |
| Lan et al. 2015 [166] | Multi-skip Feature Stacking | 65.1 |
| Sun et al. 2015 [154] | Spatio-Temporal CNN | 59.1 |
| Park et al. 2016 [156] | Deep CNN | 54.9 |
| Yu et al. 2016 [157] | SP(stratified pooling)-CNN | 74.7 |
| Bilen et al. 2016 [167] | Multiple Dynamic Images (MDI), trajectory | 65.2 |
| Mahasseni and Todorovic 2016 [168] | Lon Short term Memory- Convolutional Neural Network (LSTM-CNN) | 55.3 |
| Fernando et al. 2016 [169] | Rank pooling + CNN | 65.8 |
| Zhu et al. 2016 [170] | Key volume mining | 63.3 |
| Hollywood2 [56] | | |
| Liu et al. 2016 [131] | Genetic Programming | 46.8 |
| Le et al. 2011 [138] | Subspace analysis | 53.3 |
| Ballan et al. 2012 [141] | Codebook | 45.0 |
| Sun et al. 2014 [152] | DL-SFA | 48.1 |
| Fernando et al. 2016 [169] | Rank pooling + CNN | 75.2 |
| MSR Action3D [61] | | |
| Du et al. 2015 [153] | RNN (Recurrent Neural Network) | 94.49 |
| Wang et al. 2016 [171] | 3D Key-Pose-Motifs | 99.36 |
| Veeriah et al. 2015 [172] | Differential RNN | 92.03 |
| University of Central Florida (UCF-101) [173] | | |
| Simonyan and Zisserman 2014 [148] | Two-stream CNN | 88.0 |
| Ng et al. 2015 [174] | CNN | 88.6 |
| Wang et al. 2015 [165] | convolutional descriptor | 91.5 |
| Lan et al. 2015 [166] | Multi-skip Feature Stacking | 89.1 |
| Sun et al. 2015 [154] | Spatio-Temporal CNN | 88.1 |
| Tran et al. 2015 [155] | 3D CNN | 90.4 |
| Park et al. 2016 [156] | Deep CNN | 89.1 |
| Yu et al. 2016 [157] | SP-CNN | 91.6 |
| Bilen et al. 2016 [167] | MDI and trajectory | 89.1 |
| Mahasseni and Todorovic 2016 [168] | LSTM-CNN | 86.9 |
| Zhu et al. 2016 [170] | Key volume mining | 93.1 |
| UCF Sports [43,44] | | |
| Sun et al. 2014 [152] | DL-SFA | 86.6 |
| Weinzaepfel et al. 2015 [175] | Spatio-temporal | 91.9% |
| ActivityNet Dataset [176] | | |
| Heilbron et al. 2015 [176] | Deep Features, Motion Features, and Static Features | 42.2 (Untrimmed) |
| Heilbron et al. 2015 [176] | Deep Features, Motion Features, and Static Features | 50.2 (Trimmed) |

- Million Song dataset: http://labrosa.ee.columbia.edu/millionsong/
- Piano-midi.de: classical piano pieces (http://www.piano-midi.de/)
- Nottingham : over 1000 folk tunes (http://abc.sourceforge.net/NMD/)
- MuseData: electronic library of classical music scores (http://musedata.stanford.edu/)
- JSB Chorales: set of four-part harmonized chorales (http://www.jsbchorales.net/index.shtml)
- FMA: A Dataset For Music Analysis (https://github.com/mdeff/fma)

# Natural Images

- MNIST: handwritten digits (http://yann.lecun.com/exdb/mnist/)
- NIST: similar to MNIST, but larger
- Perturbed NIST: a dataset developed in Yoshua's class (NIST with tons of deformations)
- CIFAR10 / CIFAR100: 32×32 natural image dataset with 10/100 categories ( http://www.cs.utoronto.ca/~kriz/cifar.html)
- Caltech 101: pictures of objects belonging to 101 categories (http://www.vision.caltech.edu/Image_Datasets/Caltech101/)
- Caltech 256: pictures of objects belonging to 256 categories (http://www.vision.caltech.edu/Image_Datasets/Caltech256/)
- Caltech Silhouettes: 28×28 binary images contains silhouettes of the Caltech 101 dataset
- STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the CIFAR-10 dataset but with some modifications. http://www.stanford.edu/~acoates//stl10/
- The Street View House Numbers (SVHN) Dataset – http://ufldl.stanford.edu/housenumbers/
- NORB: binocular images of toy figurines under various illumination and pose (http://www.cs.nyu.edu/~ylclab/data/norb-v1.0/)
- Imagenet: image database organized according to the WordNethierarchy (http://www.image-net.org/)
- Pascal VOC: various object recognition challenges (http://pascallin.ecs.soton.ac.uk/challenges/VOC/)
- Labelme: A large dataset of annotated images, http://labelme.csail.mit.edu/Release3.0/browserTools/php/dataset.php
- COIL 20: different objects imaged at every angle in a 360 rotation(http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php)
- COIL100: different objects imaged at every angle in a 360 rotation (http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php)

# Faces

- Labelled Faces in the Wild: 13,000 images of faces collected from the web, labelled with the name of the person pictured (**http://vis-www.cs.umass.edu/lfw/**)
- Toronto Face Dataset
- Olivetti: a few images of several different people (**http://www.cs.nyu.edu/~roweis/data.html**)
- Multi-Pie: The CMU Multi-PIE Face Database (**http://www.multipie.org/**)
- Face-in-Action (**http://www.flintbox.com/public/project/5486/**)
- JACFEE: Japanese and Caucasian Facial Expressions of Emotion (**http://www.humintell.com/jacfee/**)
- FERET: The Facial Recognition Technology Database (**http://www.itl.nist.gov/iad/humanid/feret/feret_master.html**)
- mmifacedb: MMI Facial Expression Database (**http://www.mmifacedb.com/**)
- IndianFaceDatabase: **http://vis-www.cs.umass.edu/~vidit/IndianFaceDatabase/**)
- (e.g. The Yale Face Database (**http://vision.ucsd.edu/content/yale-face-database**) and The Yale Face Database B (**http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html**)).

# Artificial Datasets

- **Arcade Universe** – An artificial dataset generator with images containing arcade games sprites such as tetris pentomino/tetromino objects. This generator is based on the O. Breleux's **bugland** dataset generator.
- A collection of datasets inspired by the ideas from **BabyAISchool**:
  - **BabyAIShapesDatasets** : distinguishing between 3 simple shapes
  - **BabyAIImageAndQuestionDatasets** : a question-image-answer dataset
- Datasets generated for the purpose of an empirical evaluation of deep architectures (**DeepVsShallowComparisonICML2007**):
  - **MnistVariations** : introducing controlled variations in MNIST
  - **RectanglesData** : discriminating between wide and tall rectangles
  - **ConvexNonConvex** : discriminating between convex and nonconvex shapes
  - **BackgroundCorrelation** : controlling the degree of correlation in noisy MNIST backgrounds

# Text

- 20 newsgroups: classification task, mapping word occurences to newsgroup ID (http://qwone.com/~jason/20Newsgroups/)
- Reuters (RCV*) Corpuses: text/topic prediction (http://about.reuters.com/researchandstandards/corpus/)
- Penn Treebank : used for next word prediction or next character prediction (http://www.cis.upenn.edu/~treebank/)
- Broadcast News: large text dataset, classically used for next word prediction (http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC97S44)
- Wikipedia Dataset
- Multidomain sentiment analysis dataset: http://www.cs.jhu.edu/~mdredze/datasets/sentiment/

# Speech

- TIMIT Speech Corpus: phoneme classification (**http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1**)
- Aurora : Timit with noise and additional information

# Recommendation Systems

- MovieLens: Two datasets available from **http://www.grouplens.org**. The first dataset has 100,000 ratings for 1682 movies by 943 users, subdivided into five disjoint subsets. The second dataset has about 1 million ratings for 3900 movies by 6040 users.
- Jester: This **dataset** contains 4.1 million continuous ratings (-10.00 to +10.00) of 100 jokes from 73,421 users.
- Netflix Prize: Netflix released an anonymised version of their movie rating **dataset**; it consists of 100 million ratings, done by 480,000 users who have rated between 1 and all of the 17,770 movies.
- Book-Crossing dataset: This **dataset** is from the Book-Crossing community, and contains 278,858 users providing 1,149,780 ratings about 271,379 books.

# Misc

- “Musk” dataset
- CMU Motion Capture Database: (**http://mocap.cs.cmu.edu/**)
- Brodatz dataset: texture modeling (**http://www.ux.uis.no/~tranden/brodatz.html**)
- Merck Molecular Activity Challenge – **http://www.kaggle.com/c/MerckActivity/data**

- Python：An interpreter language easy to handle and visualize data processing
- Numpy：NumPy是Python語言的一個擴充程式庫。支援高階大量的維度陣列與矩陣運算，此外也針對陣列運算提供大量的數學函式函式庫。
- Keras: A python deep learning library. Before installing Keras, please install one of its backend engines:
- TensorFlow, Theano, or CNTK. We recommend the TensorFlow backend.

- Github:（open source library) :GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

- https://github.com/scikit-learn/scikit-learn

**Algorithm 8.9** Conjugate gradient meth

**Require:** Initial parameters $\boldsymbol{\theta}_0$

**Require:** Training set of $m$ examples

Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$

Initialize $g_0 = 0$

Initialize $t = 1$

**while** stopping criterion not met **do**

Initialize the gradient $\boldsymbol{g}_t = \mathbf{0}$

Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)

(Nonlinear conjugate gradient: optionally reset $\beta_t$ to zero, for example if $t$ is a multiple of some constant $k$, such as $k = 5$)

Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$

Perform line search to find: $\epsilon^* = \text{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$

(On a truly quadratic cost function, analytically solve for $\epsilon^*$ rather than explicitly searching for it)

Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$

$t \leftarrow t + 1$

**end while**

**Table 4.** Well-known public datasets for human activity recognition.

| Dataset | Year | No. of Actions | Method | Highest Accuracy |
|---|---|---|---|---|
| KTH | 2004 | 6 | [36] | 98.2% |
| Weizmann | 2005 | 9 | [87] | 100% |
| IXMAS | 2006 | 13 | [120] | 100% |
| UCF Sports | 2008 | 10 | [36] | 95.0% |
| Hollywood2 | 2009 | 12 | [169] | 75.2% |
| YouTube | 2009 | 11 | [52] | 93.38% |
| HDMB-51 | 2011 | 51 | [157] | 74.7% |
| UCF-101 | 2012 | 101 | [157] | 91.6 |
| ActivityNet (Untrimmed) | 2015 | 200 | [176] | 42.2 (baseline) |
| ActivityNet (Trimmed) | 2015 | 200 | [176] | 50.2 (baseline) |

# Deep Learning benchmark for images https://deeplearning4j.org/opendata

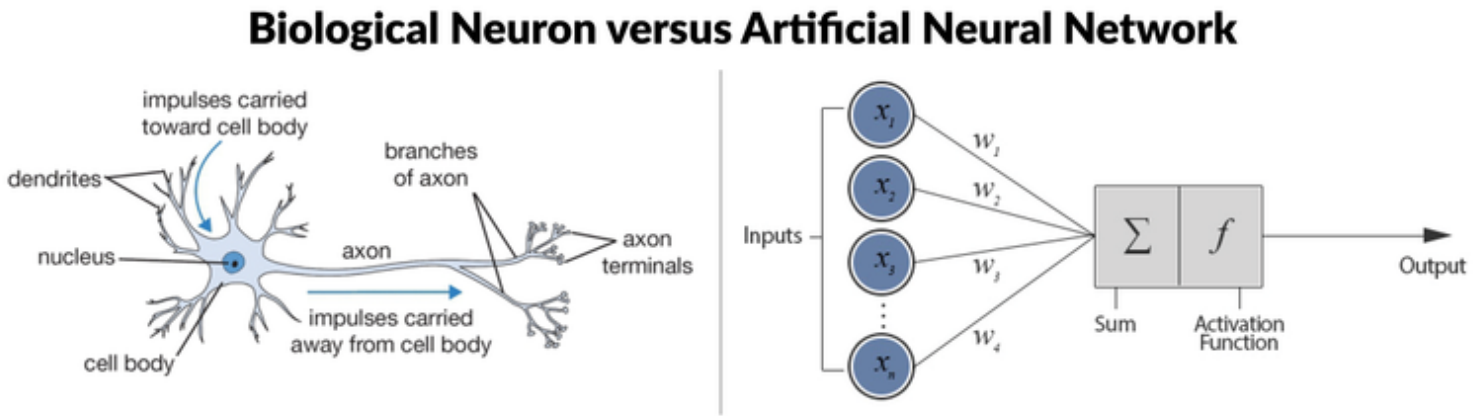
- MNIST: handwritten digits: The most commonly used sanity check. Dataset of 25x25, centered, B&W handwritten digits. It is an easy task — just because something works on MNIST, doesn't mean it works.
- CIFAR10 / CIFAR100: 32x32 color images with 10 / 100 categories. Not commonly used anymore, though once again, can be an interesting sanity check.
- Caltech 101: Pictures of objects belonging to 101 categories.
- Caltech 256: Pictures of objects belonging to 256 categories.
- STL-10 dataset: is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. Like CIFAR-10 with some modifications.
- The Street View House Numbers (SVHN): House numbers from Google Street View. Think of this as recurrent MNIST in the wild.
- NORB: Binocular images of toy figurines under various illumination and pose.
- Pascal VOC: Generic image Segmentation / classification — not terribly useful for building real-world image annotation, but great for baselines
- Labelme: A large dataset of annotated images.
- ImageNet: The de-facto image dataset for new algorithms. Many image API companies have labels from their REST interfaces that are suspiciously close to the 1000 category; WordNet; hierarchy from ImageNet.
- LSUN: Scene understanding with many ancillary tasks (room layout estimation, saliency prediction, etc.) and an associated competition.
- MS COCO: Generic image understanding / captioning, with an associated competition.
- COIL 20: Different objects imaged at every angle in a 360 rotation.
- COIL100 : Different objects imaged at every angle in a 360 rotation.
- Google's Open Images: A collection of 9 million URLs to images "that have been annotated with labels spanning over 6,000 categories" under Creative Commons.
-

# Human activity recognition (video)

**Table 4.** Well-known public datasets for human activity recognition.

| Dataset | Year | No. of Actions | Method | Highest Accuracy |
|---|---|---|---|---|
| KTH | 2004 | 6 | [36] | 98.2% |
| Weizmann | 2005 | 9 | [87] | 100% |
| IXMAS | 2006 | 13 | [120] | 100% |
| UCF Sports | 2008 | 10 | [36] | 95.0% |
| Hollywood2 | 2009 | 12 | [169] | 75.2% |
| YouTube | 2009 | 11 | [52] | 93.38% |
| HDMB-51 | 2011 | 51 | [157] | 74.7% |
| UCF-101 | 2012 | 101 | [157] | 91.6 |
| ActivityNet (Untrimmed) | 2015 | 200 | [176] | 42.2 (baseline) |
| ActivityNet (Trimmed) | 2015 | 200 | [176] | 50.2 (baseline) |

# Inspired by brain neurons



,

# Activation function of a neuron

- Sigmodal： $h = \frac{1}{1+e^{-W^T x + b}}$

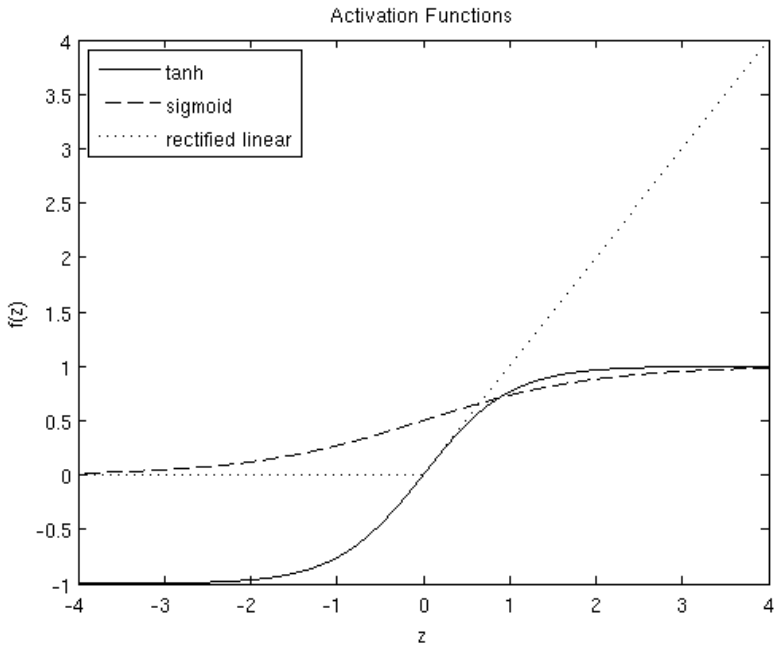
- Hypertangent:： $h = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

- Note:
- Activation function: Must be a differentiable function

# Logistic Sigmoid and Hyperbolic Tangent

- g(z) = σ(z)  logistic function ( sigmoidal function)
- σ(x) = 1 /1 + exp(−x)

- g(z) = tanh(z) = $\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
-
- tanh(z) = 2 σ(2z) − 1




- When a sigmoidal activation function must be used, the hyperbolic tangent activation function typically performs better than the logistic sigmoid.
- It resembles the identity function more closely, in the sense that
- tanh (0) = 0 while σ (0) = 1/2 .

# Rectified Linear Units (ReLU)

- ReLU: linear
- The activation function $g(z) = \max\{0, z\}$.

- Maxout units (Goodfellow et al., 2013a) generalize rectified linear units further.
- Instead of applying an element-wise function g(z),
- Maxout units divide z into groups of k values.

# Leaky Relu

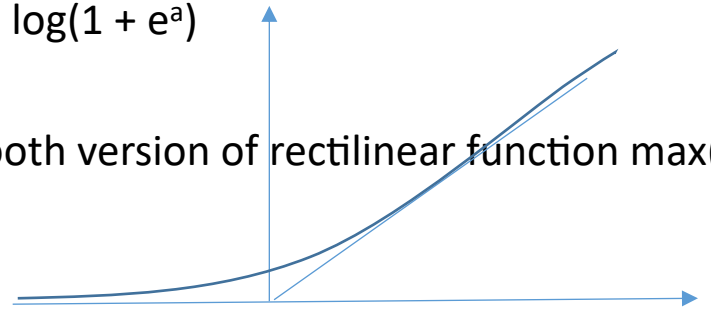
- LeakyReLU:  allow a small non-zero gradient when $x < 0$.

- $f(x) = \max(x, \alpha x)$ with $\alpha \in (0,1)$

- LeakyReLU
- $f1(x) = \max(0,x) - \alpha \max(0,-x)$
- $f2(x) = \max(0,x) - \alpha \max(0,-x)$

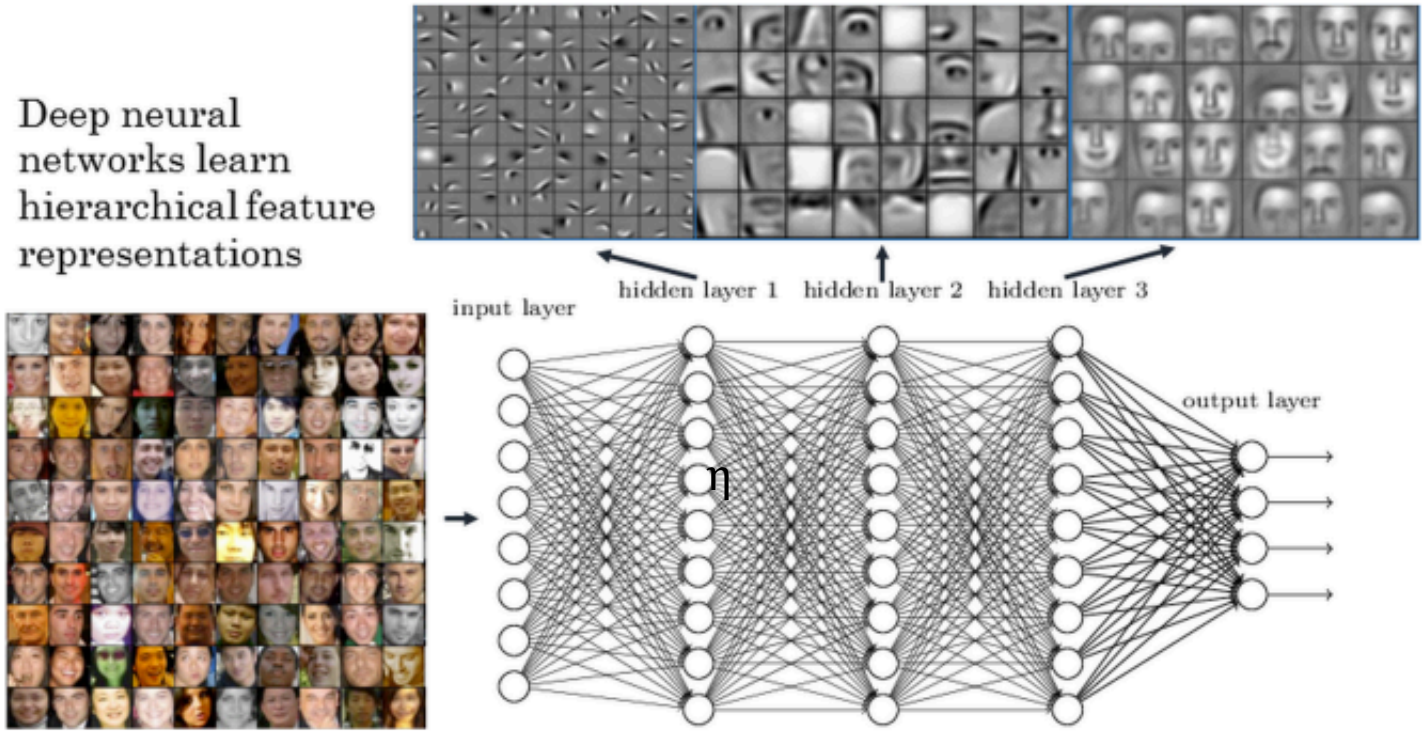
# Other hidden unit types

- Softplus function: $g(a) = \zeta(a) = \log(1 + e^a)$
- Softplus can be viewed as smooth version of rectilinear function max(0,x) ◦



- Radial basis function or RBF unit: $h_i = \exp -1/2\boldsymbol{\sigma_i} \, ||W_{:,i} - x||^2$

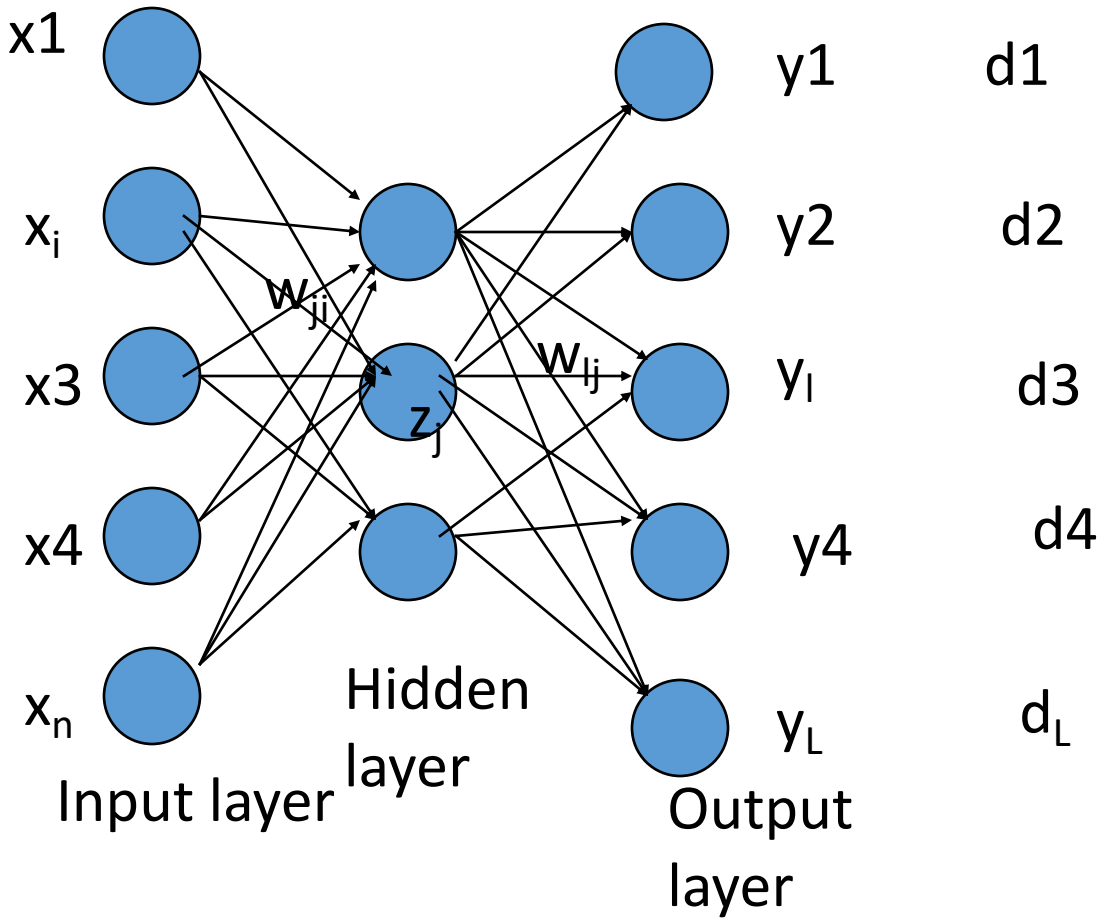
# Deep neural networks



# Layers architecture

- $h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)})$
- $h^{(2)} = g^{(2)}(W^{(2)T}h^{(1)} + b^{(2)})$
- …

# Adaptive multilayer feedforward neural networks



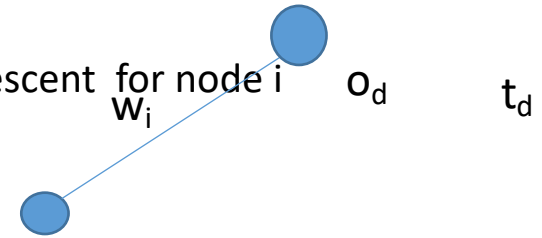
# Objective as error minimization

- o(x) = w x % output
- E(w) = 1/2 $\sum_{d\in \in D}$(td−od)2$\Sigma$ % sum square error

# Gradient Descent and the Delta Rule as weight update

- $w_{i,new} = w_{i,old} + \Delta w_i$ % update weight

  where $\Delta w_i = -\eta\eta\nabla E(w_i)$ % gradient descent for node i




$\nabla E(w) = \equiv [\partial E/\partial w_0, \partial E/\partial w_1, \ldots, \partial E/\partial w_n]$

$\partial\, \partial E/\partial w_i = \sum_{d_\epsilon \in D} (td - od)(-xid)\, \Sigma\, \Delta$

$\Delta w_i = \eta \sum_{d_\epsilon \in D} (td - od)(xid)$

# Error backpropagation

- Error function: $E(w) = 1/2\sum_{l=1,L} (dl-yl)2\ \Sigma$
- $y_l = f_o(net_l) = f_o(\sum_{l=1,L} w_{lj}z_j\Sigma)$ %activation function

  $z_j = f_h(net_j) = f_h(\sum_{i=1,n} w_{ji}x_i\Sigma)$ %activation function
- delta learning rule:

  $\Delta w_{lj} = -\eta\partial E/\partial W_{lj} = \eta\ (d_l-y_l)f'_o(net_l)z_j$

  $\Delta\ w_{ji} = -\eta\partial E/\partial W_{ji}$ (by chain rule)

# Chain rule of differentiation

$\Delta\Delta w_{ji} = -\eta\, \partial E / \partial W_{ji}$

by chain rule of differentiation:

$\partial E/ \partial W_{ji} = (\partial E/ \partial z_j)\, (\partial z_j/ \partial net_j)\, (\partial net_j/ \partial W_{ji})$

$\partial\, \partial E/ \partial z_j = -\sum_{l=1,L} (dl - yl) f'_o(netl) wlj\ \Sigma$

$\partial\, \partial z_j/ \partial net_j = f_h'(net_j)$

$\partial\, \partial\ net_j/ \partial W_{ji} = x_i$

$\Delta\Delta w_{ji} = -\eta\, [\sum_{l=1,L} \Sigma (dl - yl) f'_o(netl) wlj]\ f_h'(net_j)\ x_i$

$= -\eta\, f_h'(net_j)(d_j - z_j) x_i$

# Error backpropagation learning algorithm (incremental version)

1) Initialize all weights $w_{lj}$, $w_{ji}$
2) For each example x calculate the output y by feedforward thru the network
3) Calculate the weight change of hidden-to-output: $\Delta w_{lj}$
4) Calculate the weight change of input-to-hidden: $\Delta w_{ji}$

# Cont’d (incremental learning)

5) update the weight change

$w^{new} = w' + \Delta w$ for all i, l, j

6) Test convergence otherwise go step2

# Error backpropagation (batch version)

$E(w) = 1/2 \sum_{k=1,m} \sum_{l=1,L} (dl - yl)2 \; \Sigma\Sigma$

m is total number of input x's

Incremental is more desired for two reasons:

1) less storage required

2) makes search stochastic (each input is randomly selected)

Finnoff (1993,1994) showed that for very small learning rates, incremental backpropagation approaches batch backpropagation and produces essential the same result

# Mini-batch and stochastic gradient

- (A deep learning approach)
- Sampling a subset from the set of training examples several times
- Compute the gradient based on the sampling results…

# universal approximation theorem (Hornik et al., 1989; Cybenko, 1989)

- A feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units .

- However, we are not guaranteed that the training algorithm will be able to learn that function.

# Why deep learning? (multiple layers?)

- Single layer NN is theoretically enough in universally
- approximately learning any function…(universal approximation theorem)
- Why multi-layers?
  - Easier to impose additional constraints such as sparse coding.
  - Reducing computational cost of representing  some function.
  - Easy to differential functions at different layers and share the features at different level of abstraction

# Basic concepts

- Maximal likelihood of $p(y|x,\Theta)$ is equivalent to the minimum of square errors of estimation of y.

- Concept of alternate feature extraction (detection) with pooling to make the deep learning more robust in the sense of translational invariant.

# maximizing the log-likelihood = minimizing the mean squared error

- $MSE_{train} = 1/m \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2$
- $\hat{y}^{(i)}$ is the output of the linear regression on the i-th input $x^{(i)}$
- $\theta_{ML} = \text{argmax}_\theta P(Y|X; \theta)$.
- Let $p(y \mid x) = N(y ; \hat{y}(x ; w), \sigma^2)$ .

- And N is Gaussian function, $\hat{y}(x ; w)$ is predicted mean of the Gaussian,
- $\sum_{i=1}^{m} \boldsymbol{log}\, \boldsymbol{P}(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = -m \log \sigma - m/2 \log(2\boldsymbol{\pi}) - \sum_{i=1}^{m} \frac{||\hat{y}^{(i)} - y^{(i)}||^2}{\boldsymbol{2\sigma^2}}$
- 
- ∴ Maximize log likelihood is equivalent to minimize mean squared error.
- Note: $N(y ; \hat{y}(x ; w), \sigma^2) = \sqrt{\frac{\boldsymbol{1}}{\boldsymbol{2\pi\sigma^2}}} \exp(- -\frac{\boldsymbol{1}}{\boldsymbol{2\sigma^2}}(\hat{y}(x ; w) - y)^2$

-

- $f^* = \text{argmin}_f E_{x,y \sim pdata} ||y - f(x)||^2$
- yields
- $f^*(x) = E_{y \sim pdata(y|x)} [y]$
- This means
- if we could train on infinitely many samples from the true data-generating distribution,
- minimizing the mean squared error cost function gives
- a function that predicts the mean of y for each value of x.

# Different cost functions

- A cost function as mean absolute error
- $f^* = \text{argmin}_f \; E_{x,y \sim p\,data} \, ||y - f(x)||_1$
- Yields a function that predicts the median value of y for each x
-

# Linear regression

- We must find a function h(x) where $y^{(i)} \sim h(x^{(i)})$.
- Assume h(x) is linear function

$$h_\theta(x) = \Sigma_j \theta_j x_j = \theta^T x$$

- We seek for θ such that it minimizes:

$$J(\theta) = 1/2\ \Sigma_i (h_\theta(x^{(i)}) - y^{(i)})^2 = 1/2\ \Sigma_i (\theta^T x^{(i)} - y^{(i)})^2$$

- Gradient of with J(θ) respect to θ is:

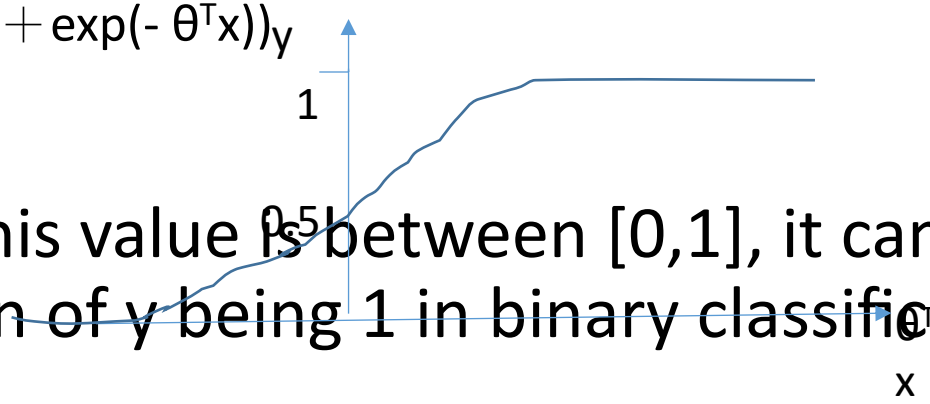
$$\nabla_\theta J(\theta) = \begin{bmatrix} \partial J(\theta)/\partial \theta_1 \\ \partial J(\theta)/\partial \theta_2 \\ \vdots \\ \partial J(\theta)/\partial \theta n \end{bmatrix}$$

$$\partial J(\theta)/\partial \theta_j = \sum_i x^{(i)}_j (h_\theta(x^{(i)}) - y^{(i)})$$

# Logistic regression

- Suppose the $h_\theta(x)$ is not linear function but a logistic function (sigmoidal)

$$y = 1/(1 + \exp(-\theta^T x))$$



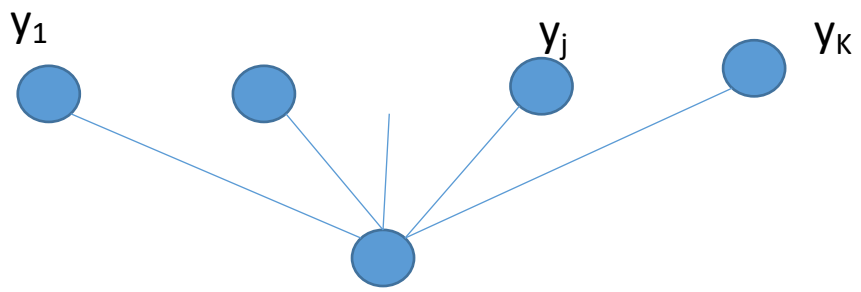

- Since this value is between [0,1], it cam be interpreted as probability function of y being 1 in binary classification.

$$d/dx\ \sigma(x) = \sigma(x)(1 - \sigma(x))$$

# Softmax regression

- Extend binary logistic regression to multi-classes
- Softmax regression has an unusual property that it has a “redundant” set of parameters.
- i.e too many sets of parameters can fit the data to the same hypothesis function h

$$P(y^{(i)} = k \mid x^{(i)}; \vartheta) = \exp(\vartheta(k)^{\top} x^{(i)}) \,/\, \sum_{j=1}^{K} \exp(\vartheta^{(j)\top} x^{(i)})$$




$$\text{softmax}(x)_i = \exp(x_i)/\Sigma_j \exp(xj) \ .$$

# Expected Loss function in terms of averaging over per-example loss function

- $J(\theta) = E_{x,y \sim \hat{p}data} L(x, y, \theta) = 1/m \sum_{i=1}^{m} \boldsymbol{L}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{\theta})$
-
- L is per example loss:
- $L(x, y, \theta) = -\log p(y \mid x; \theta)$.

- $\boldsymbol{\nabla_{\theta} J(\theta)} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{\nabla_{\theta} L}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}, \boldsymbol{\theta})$
- The cost is $O(m)$.

- Note: 1. minimize 'negative' log likelihood = maximize log likelihood = minimize expected error
- 2. Gradient of loss function is expected value of all gradients from each training sample

# Stochastic Gradient Descent (SGD)

- *Update the parameter Θ*

  $\Theta = \vartheta - \alpha \nabla_{\vartheta} E\,[J(\vartheta)]$

  where $\alpha$ *is the learning rate, E is expectation*
- Compute by sampling from training examples:
- $\theta = \theta - \alpha\ 1/n \sum_{i=1}^{n} \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)})$

# Overfitting is hard to detect

- When we minimize the training error, it is easy to overfit the training data and obtain a result that leads to poor generalization error.
- That mean when we try to minimizing we don’t know when to stop is proper.

# Why regularization?

- To avoid overfitting in machine learning training, we impose a constraint to prefer smooth the hypothesis output . ( A kind of learning bias (preference))
- This is usually expressed in terms of a regularization penalty in optimization.

# Regularization strategies

- Regularization strategy is to avoid overfitting.
- Add penalty term to objective loss function so that the weight parameters tend to converge to a smaller absolute value. (e.g. we prefer small weights)
- Dropout: randomly removes hidden units during training . (e.g. we don’t want all hidden units to involved in each decision)
- Maxout; rnnDrop;
  - Batch normalization
  - Early stop

# Early stopping in deep Training

- A common method used in regularization
- Training set; validation set; test set
- When the error in the validation set starts to worsen several times, then terminate the training. [in avoid of overfitting]
- How to decide when to stop learning early ?
- Use validation set to ensure the training results will not overtrain. (e.g. If current training results cause the prediction error on validation set to increase rather than decrease then stop learning.)

# Regularization in deep leaning

1. Parameter norm penalty
   $J(\theta; X, y) = J(\theta; X, y) + \alpha \Omega(\theta)$
   - L2 norm: $\Omega(\theta) = 1/2||w||^2_2$
   - L1 norm: $\Omega(\theta) = ||w||_1 = \sum_{i=1} |w_i|$, Sparse representation: place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse.
2. Dataset augmentation
   - create fake data and add it to the training set (rotate, translate, etc.)
   - Injecting noise in the input to a neural network
   - Injecting noise to the hidden units
   - Injecting noise to weights
   - Injecting noise to output targets
3. Early stopping
4. Dropout
5. Parameter tying and parameter sharing:
   Express our prior knowledge about suitable values of the model parameters
6. Bagging and ensemble methods
7. Adversarial training: generate adversarial example x' close to x that human cannot tell
8. Tangent prop and manifold tangent classifier

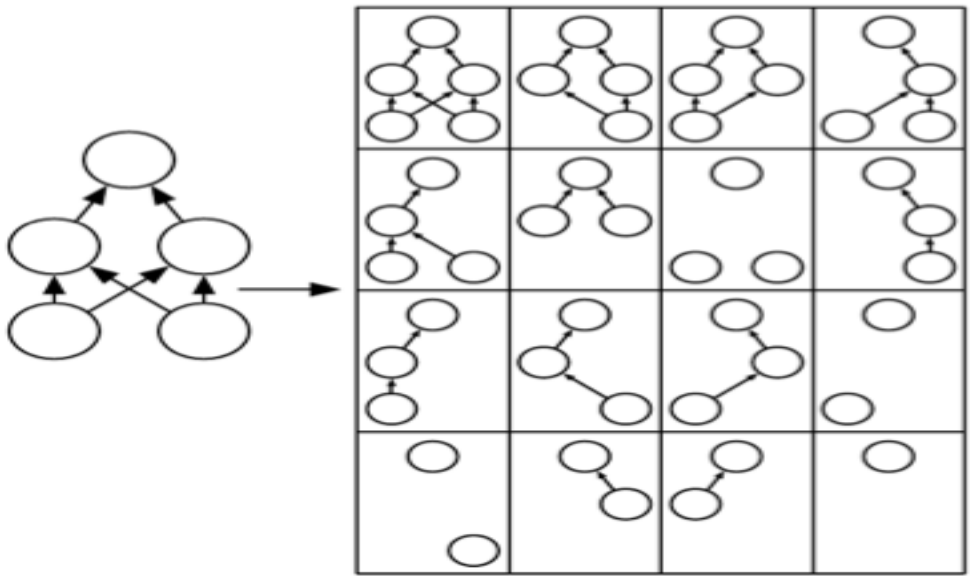
# Early stopping

- Let n be the number of steps between evaluations.
  Let p be the "patience," the number of times to observe worsening validation set v error before giving up.
  Let θo be the initial parameters.
  $\theta \leftarrow \theta o$ ; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $\theta* \leftarrow \theta$; $i* \leftarrow i$
- while j < p do
- Update θ by running the training algorithm for n steps. $i \leftarrow i + n$
  $v' \leftarrow$ ValidationSetError(θ)
  if v' < v then
- $j \leftarrow 0$ ; $\theta* \leftarrow \theta$; $i* \leftarrow i$ ; $v \leftarrow v'$
- else $j \leftarrow j+1$
- end if
- end while

# dropout

- Dropout is a kind of regularization technique  for reducing overfitting in neural networks by preventing complex co-adaptations on training data.
- The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.
- Since we do not use all units in NN training, it becomes more “relaxed” for fitting training data.

# Dropout



# Clipping gradient

- When gradient becomes too large: ||g||> v
- Where v is norm threshold
- Then  g = g v/||g||

- How about Gradient vanishing problem?

# momentum update

- $v = r\, v + \alpha\, \nabla_{\vartheta} J(\vartheta; x^{(i)}, y^{(i)})$
- 
- $\vartheta = \vartheta - v$
- *Where v is the current velocity*
- *r is the momentum constant*

# Hyper parameters

- Learning rate
- Number of hidden units
- Convolution kernel width (for CNN)
- Implicit zero padding (for CNN)
- Weight decay coefficient
- Dropout rate

# Whitening (reduce the burden of learning)

- The goal of whitening is to make the input less redundant;
- more formally, to make learning algorithms see a training input
  - (i) the features are less correlated (independent) with each other, and
  - (ii) the features all have the same variance.

  - PCA is a technique of whitening

# Optimization for training deep learning models

- Finding the parameters θ of a neural network that significantly reduce a cost function J(θ), which typically includes a performance measure evaluated on the entire training set as well as additional regularization terms.
- The cost function can be written as an average over the training set, such as
- $J(\theta) = E_{(x,y) \sim \hat{p}_{data}} L(f(x; \theta), y)$,
- $\hat{p}_{data}$ is the empirical distribution.
- Minimize the corresponding objective function where the expectation is taken across the data generating distribution $p_{data}$ rather than just over the finite training set:
- $J^*(\theta) = E_{(x,y) \sim p_{data}} L(f(x; \theta), y)$.

- Batch gradient descent
- $\Theta = \theta - \eta \cdot \nabla_\theta J(\theta)$
- Stochastic gradient descent
- $\Theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$
- Mini-batch gradient descent
- $\Theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

# Nesterov accelerated gradient

- Parameter Update with momentum
- $V_t\theta = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$
- $\theta = \theta - v_t$

- Momemtum coefficient is the fraction γ of the update vector of the past time step to the current update vector:
- Nesterov:
- $V_t = \gamma\, v_{t\ -1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$

# Adagrad

- Adagrad modifies the general learning rate η at each time step t for every parameter θi based on the past gradients that have been computed for θi:
- $\theta_{t+1,I} = \theta_{t,I} - \eta / \sqrt{G_{t,ii}+\epsilon} \cdot g_{t,I}$
- $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element ii,is the sum of the squares of the gradients w.r.t. $\theta_i$ up to time step t.

- We could write it as:
- $\theta_{t+1} = \theta_t - \eta / \sqrt{G_t+\epsilon} \odot g_t$

# Adadelta

- Adadelta [6] is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

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

- default values of 0.9 for $\beta_1$, 0.999 for $\beta_2$, and $10^{-8}$ for $\epsilon$.

# RMSprop

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g^2_t$$

$$\theta_{t+1} = \theta_t - \eta \, g_t / \sqrt{E[g^2]_t + \epsilon}$$

Hinton suggests $\gamma$ to be set to 0.9, while a good default value for the learning rate $\eta$ is 0.001.

# Adam

Adaptive learning rates for each parameter.
In addition to storing an exponentially decaying average of past squared gradients vtvt like Adadelta and RMSprop

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

- $m_t$ and $v_t$ are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively,

$\hat{m}_t = m_t / (1-\beta_1^t)$
$\hat{v}_t = v_t / (1-\beta_2^t)$

- $\theta_{t+1} = \theta_t - \eta\, \hat{m}_t / \sqrt{\hat{v}_t + \epsilon}$.

- propose default values of 0.9 for $\beta_1$, 0.999 for $\beta_2$, and $10^{-8}$ for $\epsilon$

# AdaMax

$V_t = \beta_2 v_{t-1} + (1-\beta_2)|g_t|^2$ generalized with respect to norm p

$V_t = \beta_2^p v_{t-1} + (1-\beta_2^p)|g_t|^p$

as p approaches infinity

$V_t = \max(\beta_2 v_{t-1}, |g_t|)$

- Change notation $v_t$ to $u_t$

$$\theta_{t+1} = \theta_t - \eta\, \hat{m}_t / u_t$$

- $\hat{m}_t$ is same as ADAM

- $\eta$=0.002, $\beta$1=0.9, and $\beta$2=0.999.

# Nadam

- Nadam (Nesterov-accelerated Adaptive Moment Estimation) thus combines Adam and NAG. In order to incorporate NAG into Adam,
- $g_t = \nabla\theta_t J(\theta_t)$
- $m_t = \gamma m_{t-1} + \eta g_t$
- $\theta_{t+1} = \theta_t - m_t$

# AMSGrad

# Additional strategies to optimize SGD

- Suffling and curriculum learning: avoid pre-fixed order of feeding training examples but prefer feeding from easier to difficult ones.
- Early stopping: a free-lunch to avoid overfit
- Batch normalization: avoid shifting parameters values during training

# Which optimizer to use?

- So, which optimizer should you now use? If your input data is sparse, then you likely achieve the best results using one of the adaptive learning-rate methods. An additional benefit is that you won't need to tune the learning rate but likely achieve the best results with the default value.
- In summary, RMSprop is an extension of Adagrad that deals with its radically diminishing learning rates. It is identical to Adadelta, except that Adadelta uses the RMS of parameter updates in the numinator update rule. Adam, finally, adds bias-correction and momentum to RMSprop. Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. Kingma et al. [15] show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice.
- Interestingly, many recent papers use vanilla SGD without momentum and a simple learning rate annealing schedule. As has been shown, SGD usually achieves to find a minimum, but it might take significantly longer than with some of the optimizers, is much more reliant on a robust initialization and annealing schedule, and may get stuck in saddle points rather than local minima. Consequently, if you care about fast convergence and train a deep or complex neural network, you should choose one of the adaptive learning rate methods.

# Stochastic gradient descent (SGD) update at training iteration k

- Learning rate $\epsilon_k$
- Initial parameter $\theta$
- while stopping criterion not met do
  Sample a mini-batch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ corresponding targets $y^{(i)}$.
  Compute gradient estimate:
- $\hat{g} \leftarrow +1/m \nabla_\theta \sum_i L(f(x^{(i)};\theta), y)$
- Apply update: $\theta \leftarrow \theta - \epsilon_k \hat{g}$
- end while

# Mini-batch estimation of Stochastic gradient optimization

- Learning rate $\epsilon$, momentum parameter $\alpha$
- Initial parameter $\theta$, initial velocity v.
- while stopping criterion not met do
  Sample a minibatch of m examples from the training set
- $\{x^{(1)}, \ldots, x^{(m)}\}$ corresponding targets $y^{(i)}$.
- Compute gradient estimate: $g \leftarrow 1/m \nabla_\theta \sum_i L(f(x ; \theta), y)$
- Compute velocity update: $v \leftarrow \alpha v - \epsilon g$
  Apply update: $\theta \leftarrow \theta + v$
- end while

# Adaptive learning rate

**Algorithm 8.4** The AdaGrad algorithm

**Require:** Global learning rate $\epsilon$

**Require:** Initial parameter $\boldsymbol{\theta}$

**Require:** Small constant $\delta$, perhaps $10^{-7}$, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

**while** stopping criterion not met **do**

Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end while**

# Adaptive Learning Rates: ADAM

- Step size $\epsilon$ (default: 0.001)
- Exponential decay rates for moment estimates, ρ1 and ρ2 in [0, 1). (defaults: 0.9 and 0.999 respectively)
- Small constant δ used for numerical stabilization. (default: $10^{-8}$ )
- Initial parameters θ
  Initialize 1st and 2nd moment variables s = 0, r = 0
- Initialize time step t = 0
- while stopping criterion not met do
- Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$
- corresponding targets $y^{(i)}$.
  Compute gradient: $g \leftarrow 1/m \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
- t←t+1
- Update biased first moment estimate: $s \leftarrow \rho_1 s + (1 - \rho_1) g$
- Update biased second moment estimate: $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
- Correct bias in first moment: $\hat{s} \leftarrow s/(1-\rho_1^t)$
- Correct bias in second moment: $\hat{r} \leftarrow r/(1-\rho_2^t)$
- Compute update $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}+\delta}}$ (element-wise)
- Apply update: $\theta \leftarrow \theta + \Delta\theta$
- End while

- Note: Element-wise product or Hadamard product, and is denoted as A ⊙ B.

# RMSprop optimization

**Algorithm 8.5** The RMSProp algorithm

**Require:** Global learning rate $\epsilon$, decay rate $\rho$.
**Require:** Initial parameter $\boldsymbol{\theta}$
**Require:** Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
**while** stopping criterion not met **do**
  Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
  Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
**end while**

**Algorithm 8.8** Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$.

**Require:** Initial parameter $\boldsymbol{\theta}_0$

**Require:** Training set of $m$ examples

**while** stopping criterion not met **do**

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Compute Hessian inverse: $\boldsymbol{H}^{-1}$

Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$

Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end while**