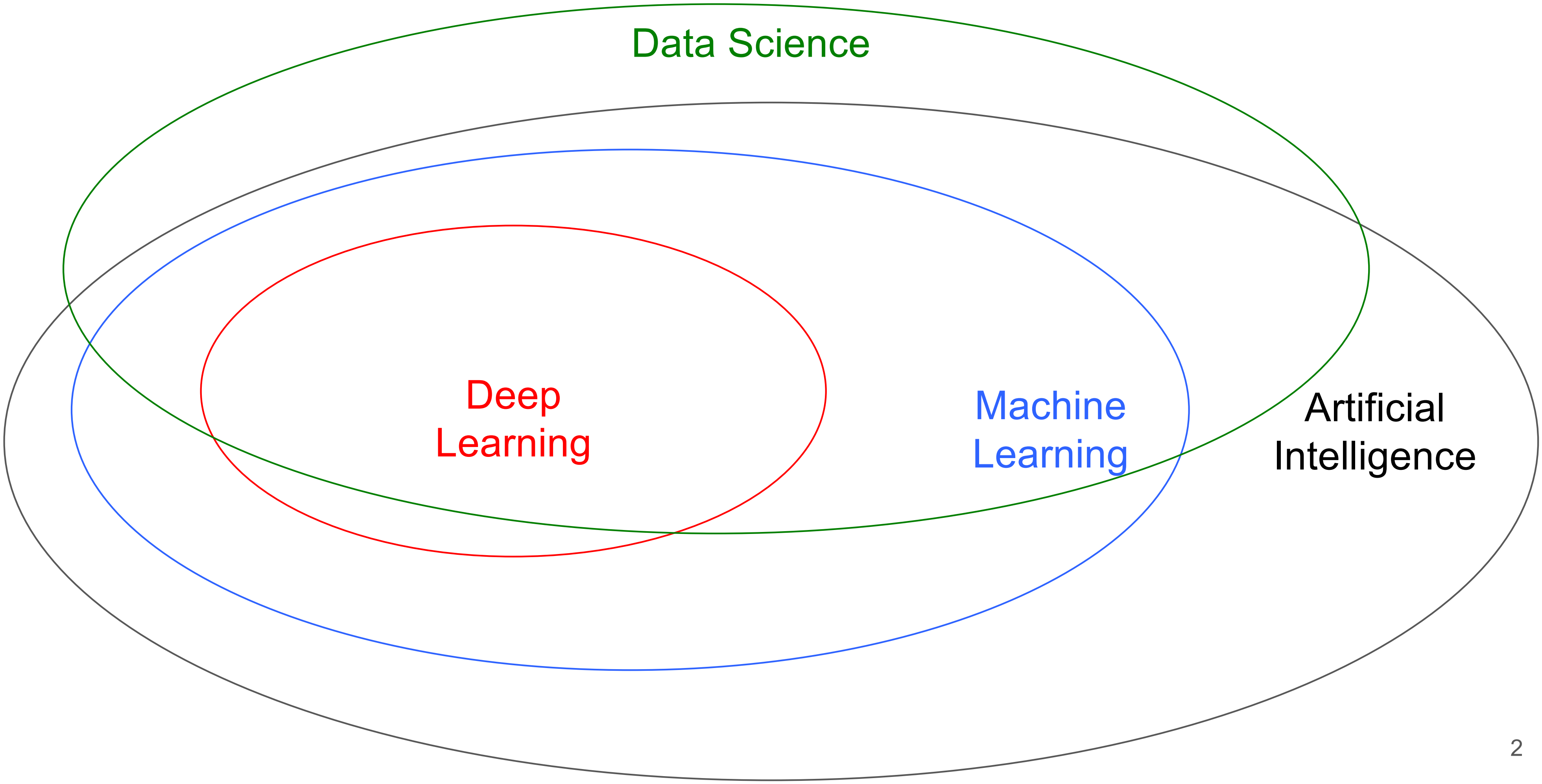
Biomedical Data Science: Mining and Modeling

Deep Learning: Part I

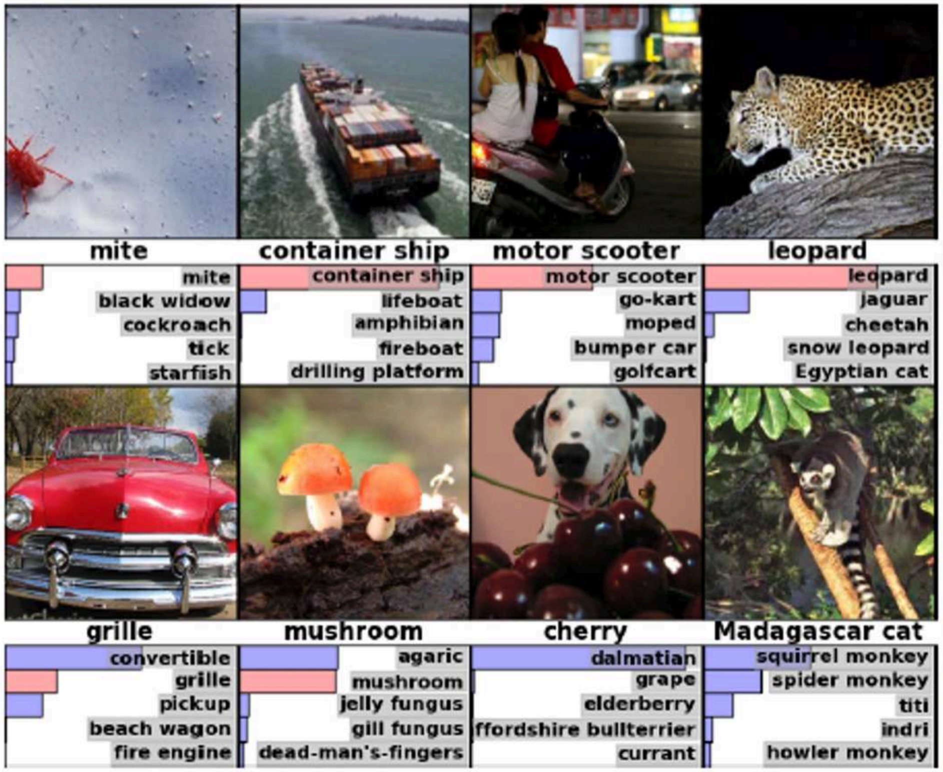
Dr. Martin Renqiang Min
Machine Learning Department
NEC Laboratories America

Mar 27, 2023

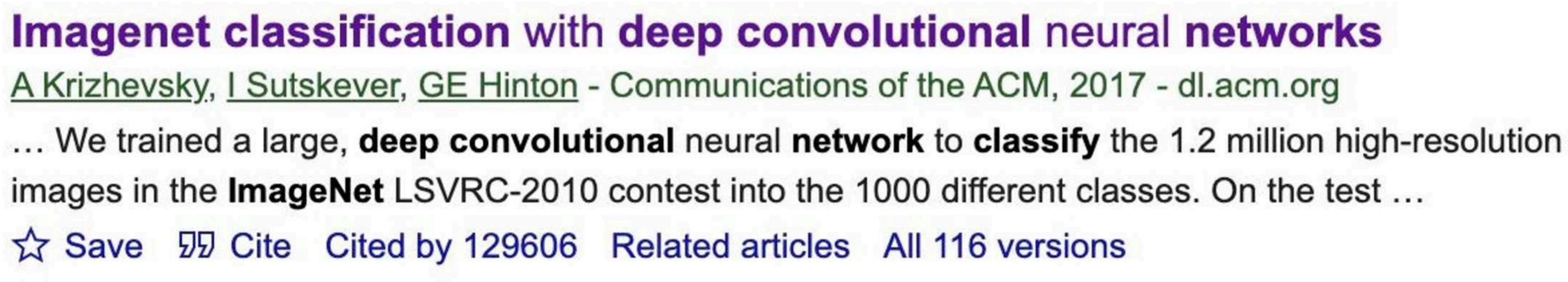
What is Deep Learning



The AI Revolution is Driven by Deep Learning: The ImageNet Challenge in Computer Vision



The AI Revolution is Driven by Deep Learning: The ImageNet Challenge in Computer Vision



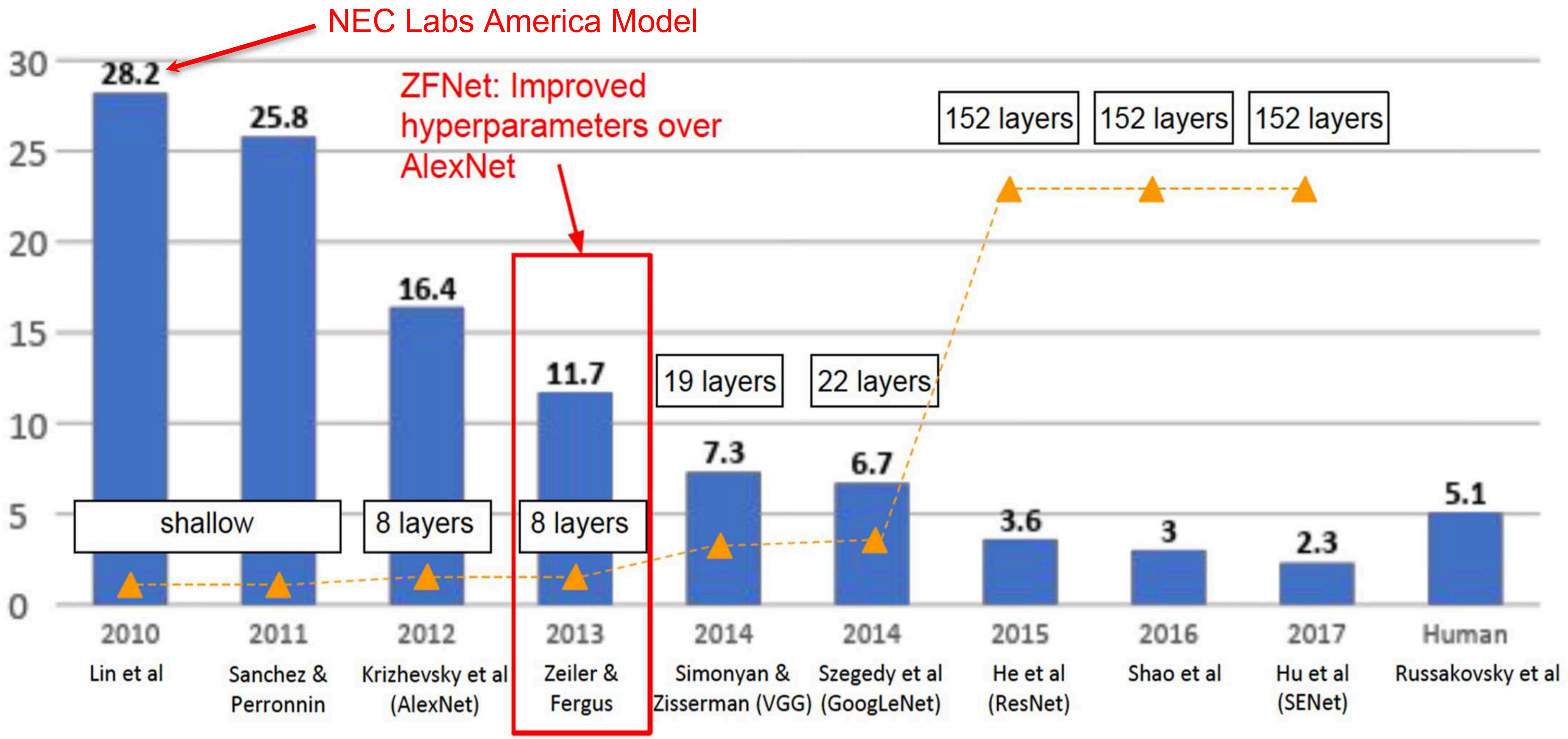
Imagenet classification with **deep convolutional** neural **networks**

A Krizhevsky, I Sutskever, GE Hinton - Communications of the ACM, 2017 - dl.acm.org

… We trained a large, **deep convolutional** neural **network** to **classify** the 1.2 million high-resolution images in the **ImageNet** LSVRC-2010 contest into the 1000 different classes. On the test …

☆ Save 99 Cite Cited by 129606 Related articles All 116 versions

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

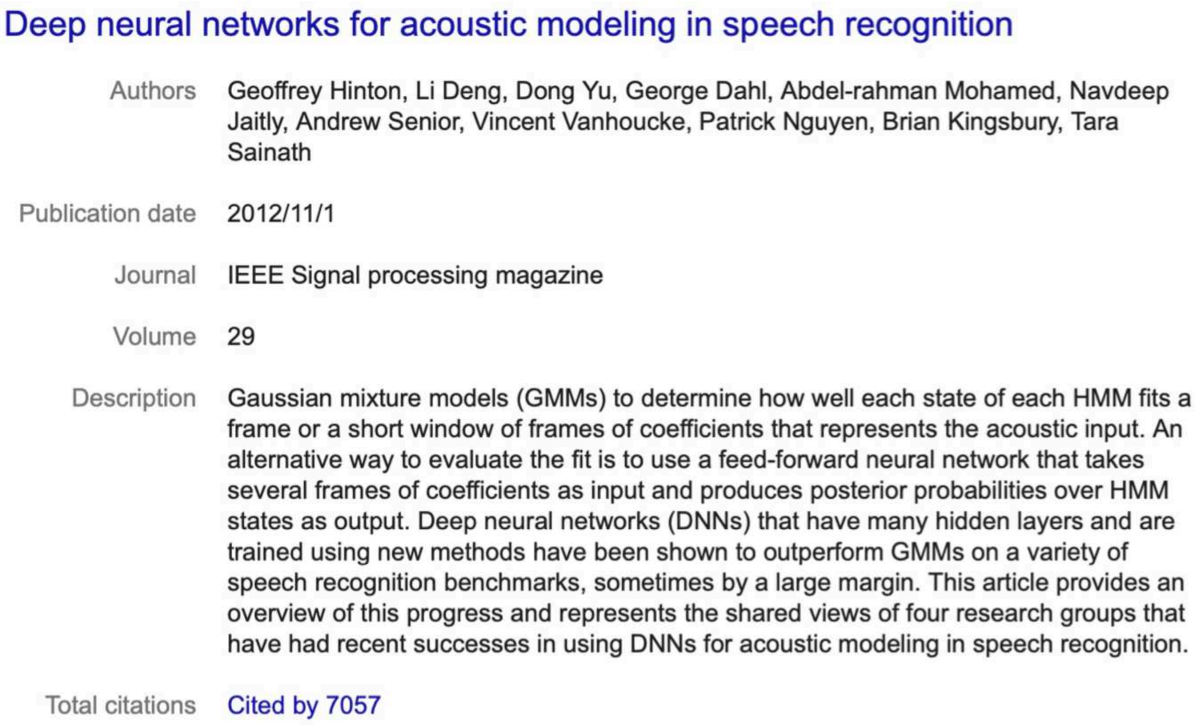


NEC Labs America Model

ZFNet: Improved hyperparameters over AlexNet

Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

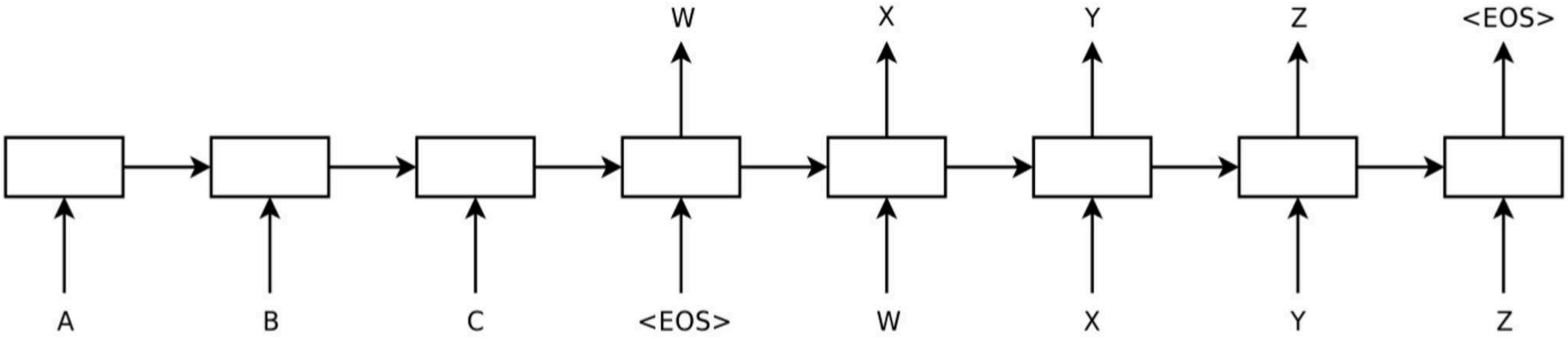
The AI Revolution is Driven by Deep Learning: Speech Recognition

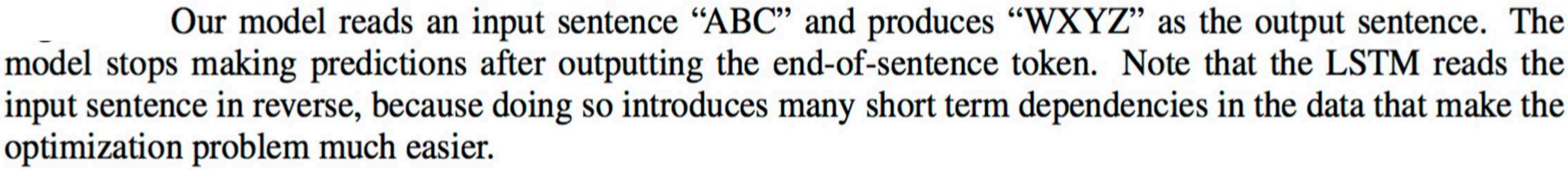


Deep neural networks for acoustic modeling in speech recognition

Authors	Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, Tara Sainath
Publication date	2012/11/1
Journal	IEEE Signal processing magazine
Volume	29
Description	Gaussian mixture models (GMMs) to determine how well each state of each HMM fits a frame or a short window of frames of coefficients that represents the acoustic input. An alternative way to evaluate the fit is to use a feed-forward neural network that takes several frames of coefficients as input and produces posterior probabilities over HMM states as output. Deep neural networks (DNNs) that have many hidden layers and are trained using new methods have been shown to outperform GMMs on a variety of speech recognition benchmarks, sometimes by a large margin. This article provides an overview of this progress and represents the shared views of four research groups that have had recent successes in using DNNs for acoustic modeling in speech recognition.
Total citations	Cited by 7057

The AI Revolution is Driven by Deep Learning: Machine Translation





Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Sutskever, Vinyals, and Le, NIPS 2014

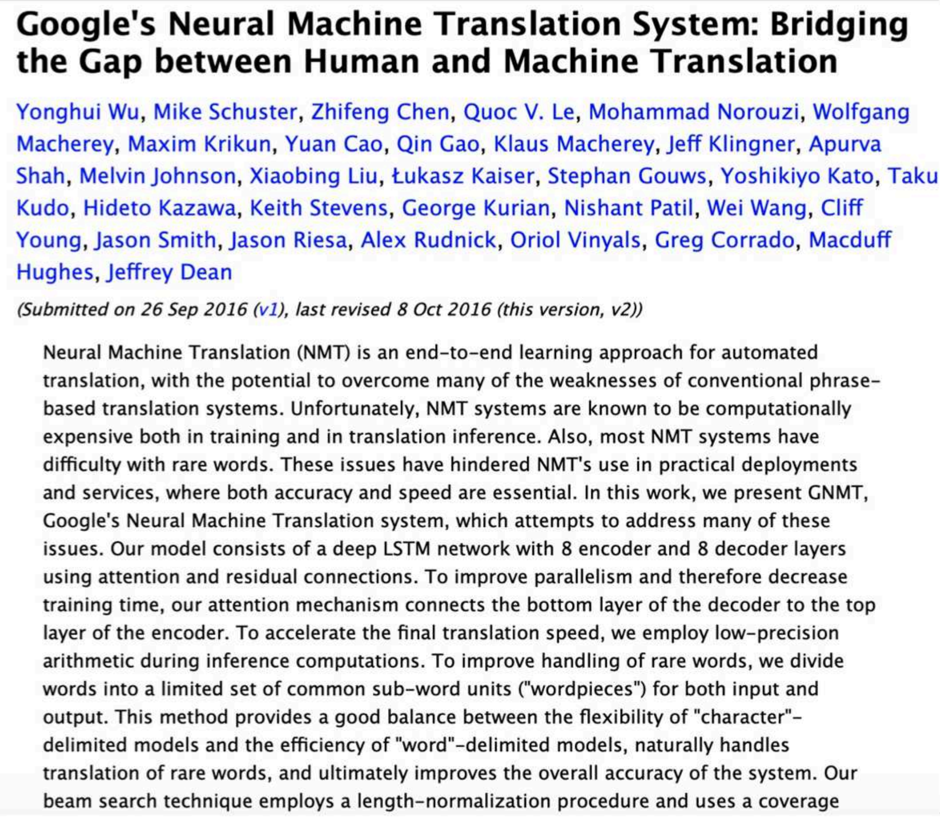
The AI Revolution is Driven by Deep Learning: Machine Translation

The New York Times Magazine

FEATURE

The Great A.I. Awakening

How Google used artificial intelligence to transform Google Translate, one of its more popular services — and how machine learning is poised to reinvent computing itself.



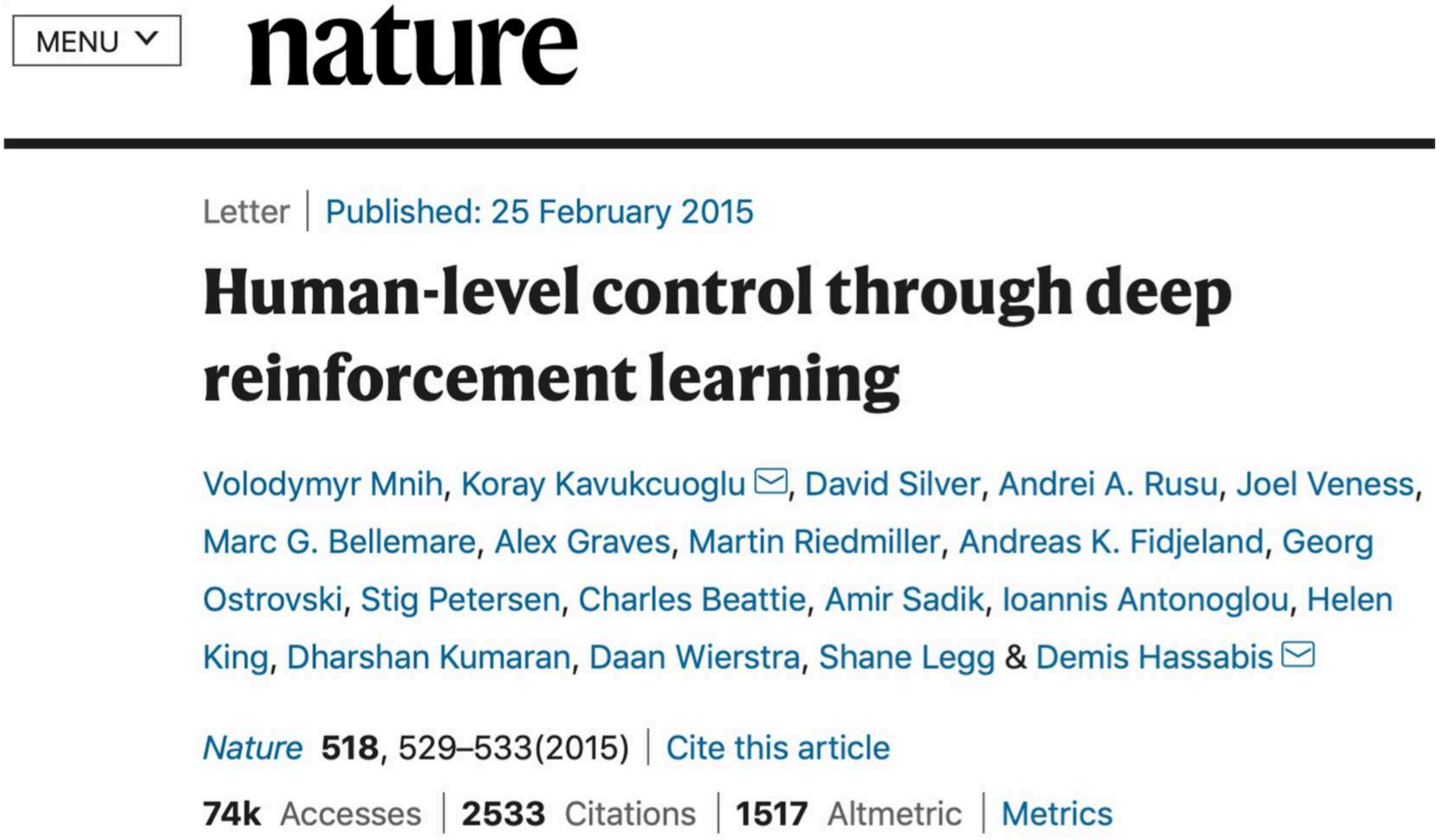
Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

(Submitted on 26 Sep 2016 (v1), last revised 8 Oct 2016 (this version, v2))

Neural Machine Translation (NMT) is an end-to-end learning approach for automated translation, with the potential to overcome many of the weaknesses of conventional phrase-based translation systems. Unfortunately, NMT systems are known to be computationally expensive both in training and in translation inference. Also, most NMT systems have difficulty with rare words. These issues have hindered NMT's use in practical deployments and services, where both accuracy and speed are essential. In this work, we present GNMT, Google's Neural Machine Translation system, which attempts to address many of these issues. Our model consists of a deep LSTM network with 8 encoder and 8 decoder layers using attention and residual connections. To improve parallelism and therefore decrease training time, our attention mechanism connects the bottom layer of the decoder to the top layer of the encoder. To accelerate the final translation speed, we employ low-precision arithmetic during inference computations. To improve handling of rare words, we divide words into a limited set of common sub-word units ("wordpieces") for both input and output. This method provides a good balance between the flexibility of "character"-delimited models and the efficiency of "word"-delimited models, naturally handles translation of rare words, and ultimately improves the overall accuracy of the system. Our beam search technique employs a length-normalization procedure and uses a coverage

The AI Revolution is Driven by Deep Learning: Video Game



Letter | Published: 25 February 2015

Human-level control through deep reinforcement learning

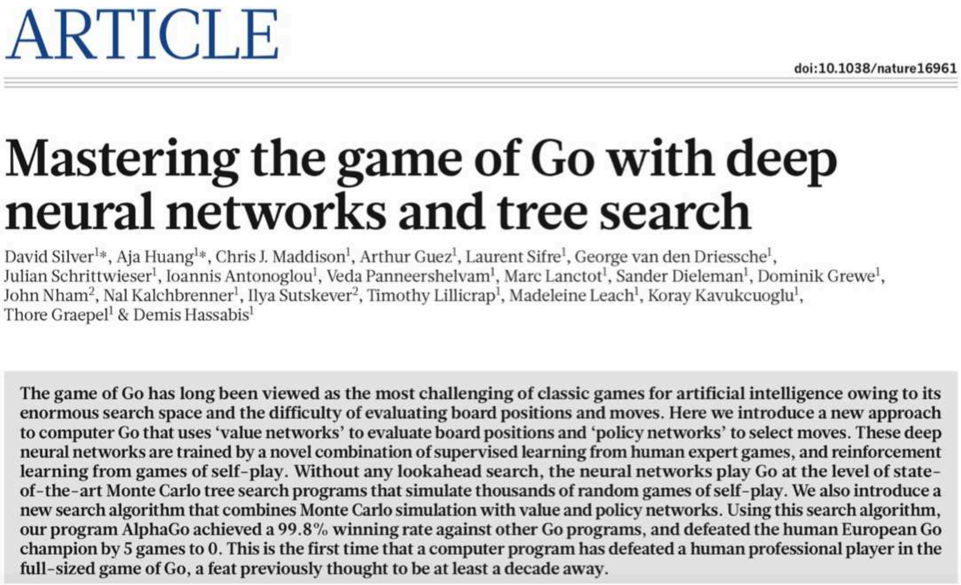
Volodymyr Mnih, Koray Kavukcuoglu ✉, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis ✉

Nature **518**, 529–533(2015) | Cite this article

74k Accesses | **2533** Citations | **1517** Altmetric | Metrics

The AI Revolution is Driven by Deep Learning: Breakout Video Game

The AI Revolution is Driven by Deep Learning: Go Game



ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

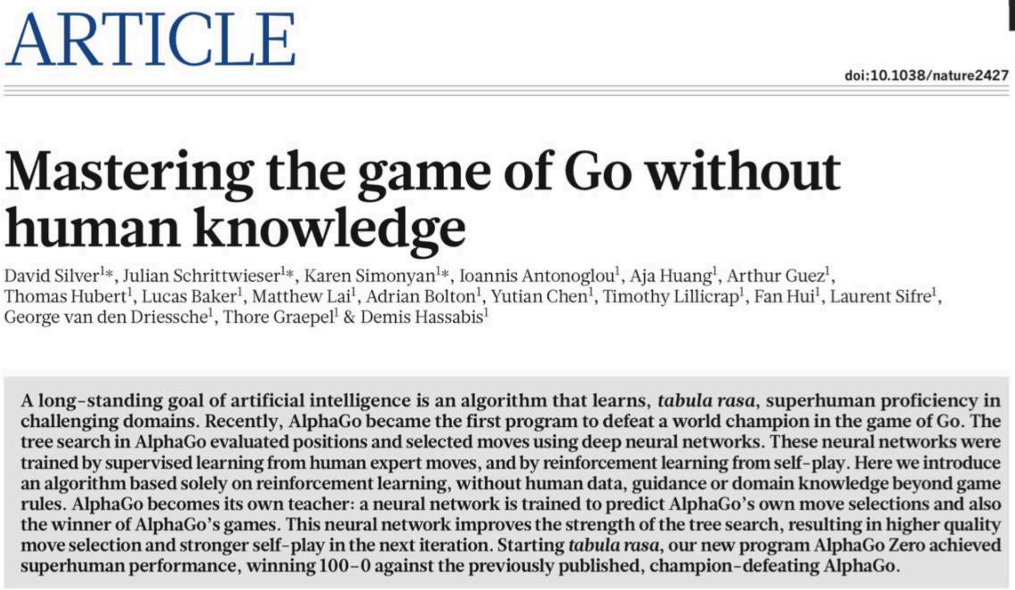
theguardian.com
Mar 2016

AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge

▲ The world's top Go player, Lee Sedol, lost the final game of the Google DeepMind challenge match. Photograph: Yonhap/Reuters

Learn How to Play Chess without Human Knowledge



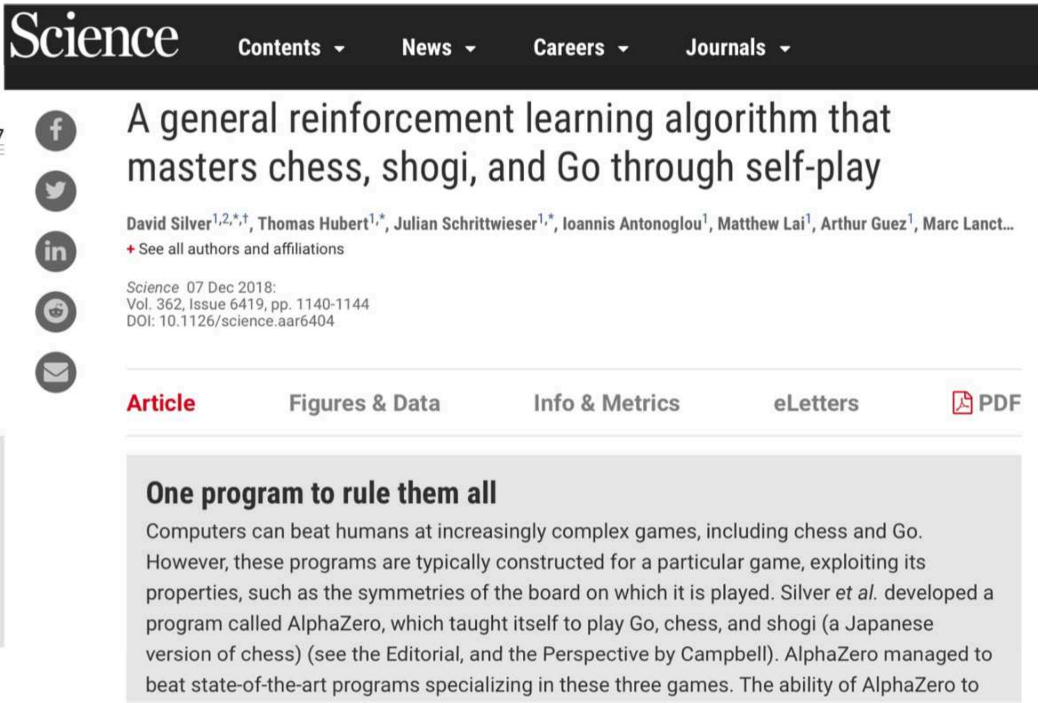
ARTICLE

doi:10.1038/nature2427

Mastering the game of Go without human knowledge

David Silver[1*], Julian Schrittwieser[1*], Karen Simonyan[1*], Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.



A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

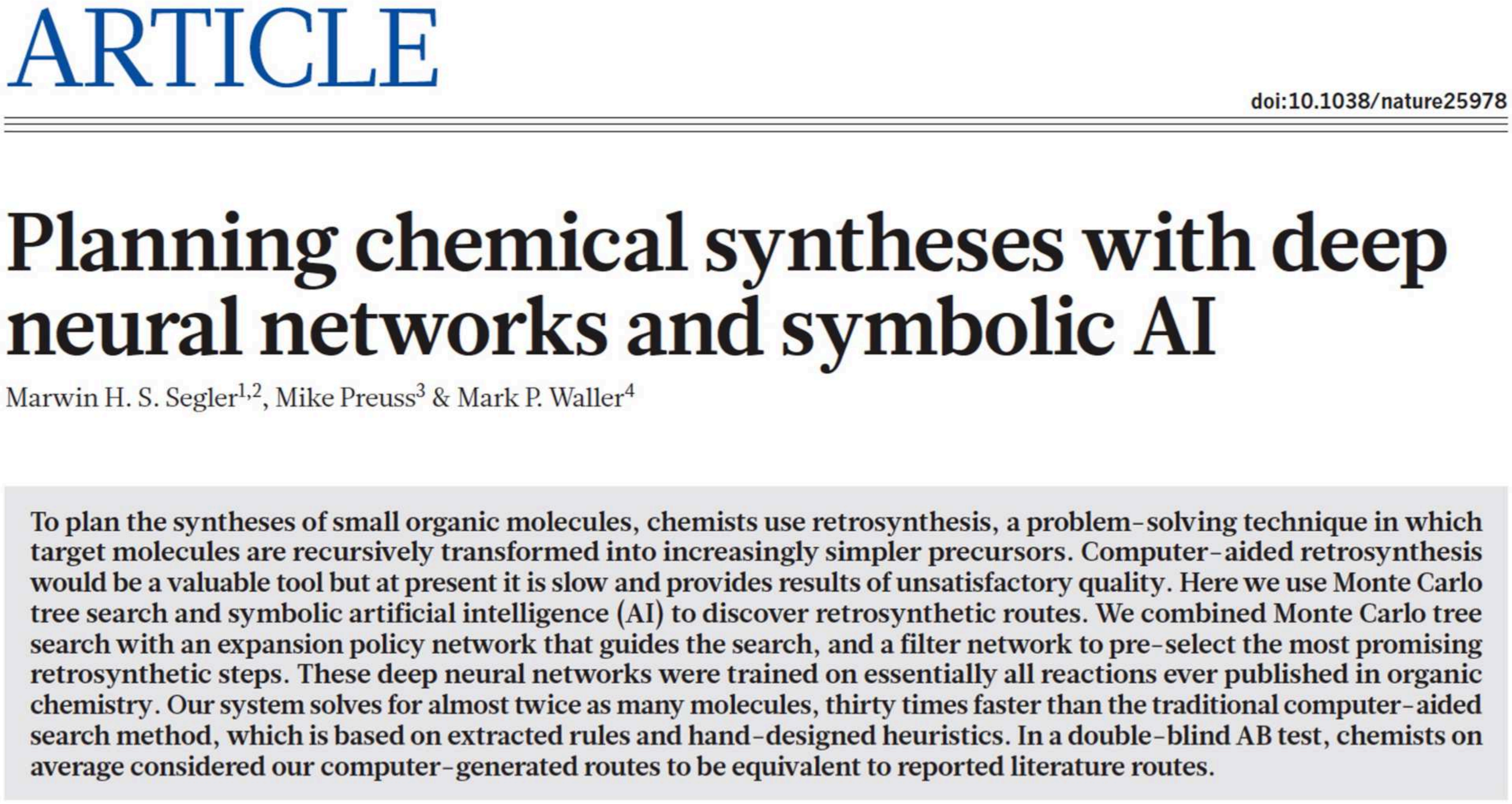
David Silver[1,2,*,†], Thomas Hubert[1,*], Julian Schrittwieser[1,*], Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanct...
+ See all authors and affiliations

Science 07 Dec 2018:
Vol. 362, Issue 6419, pp. 1140-1144
DOI: 10.1126/science.aar6404

One program to rule them all

Computers can beat humans at increasingly complex games, including chess and Go. However, these programs are typically constructed for a particular game, exploiting its properties, such as the symmetries of the board on which it is played. Silver *et al.* developed a program called AlphaZero, which taught itself to play Go, chess, and shogi (a Japanese version of chess) (see the Editorial, and the Perspective by Campbell). AlphaZero managed to beat state-of-the-art programs specializing in these three games. The ability of AlphaZero to

The AI Revolution is Driven by Deep Learning: Molecule (Drug) Synthesis



ARTICLE

doi:10.1038/nature25978

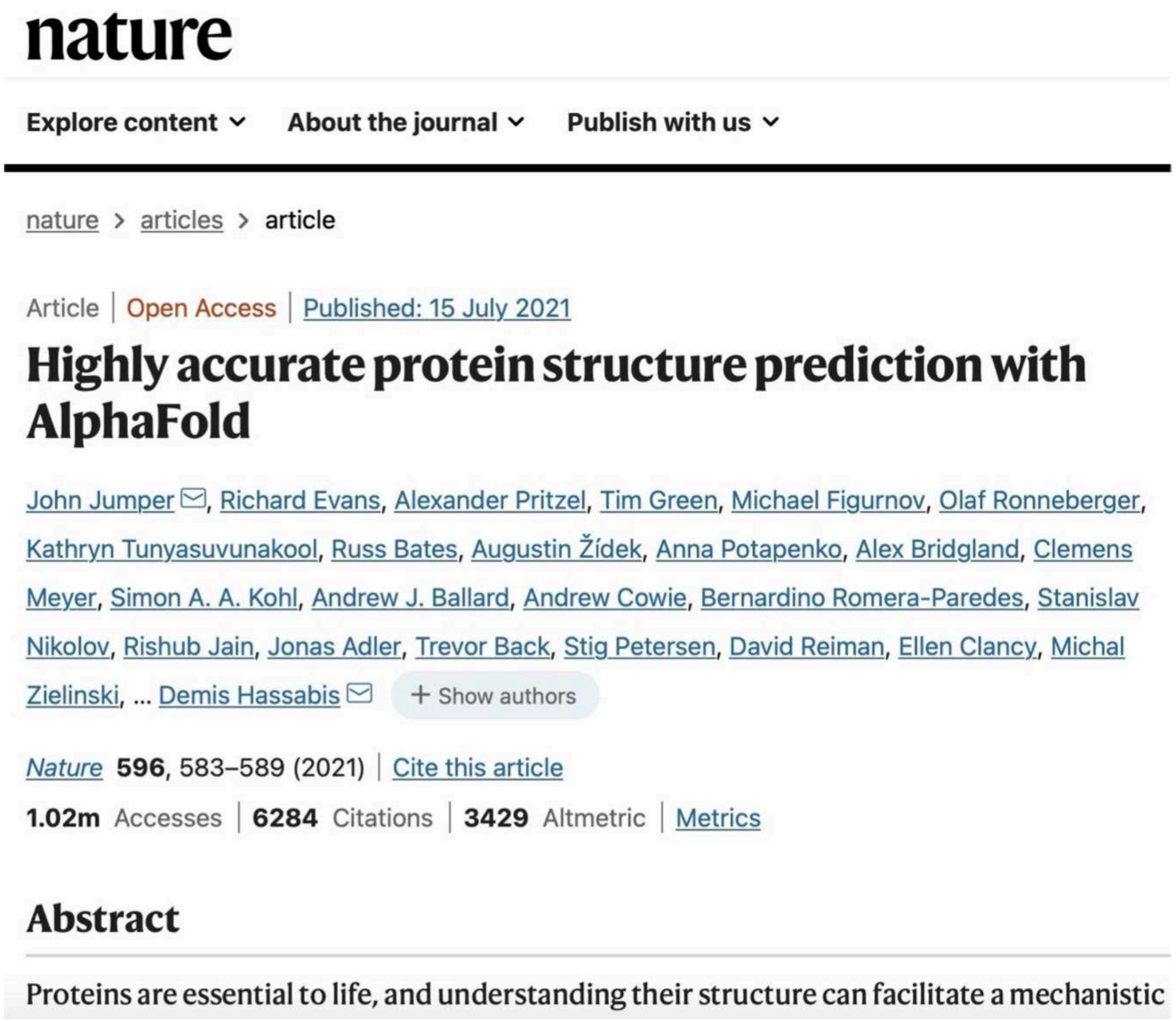
Planning chemical syntheses with deep neural networks and symbolic AI

Marwin H. S. Segler[1,2], Mike Preuss[3] & Mark P. Waller[4]

To plan the syntheses of small organic molecules, chemists use retrosynthesis, a problem-solving technique in which target molecules are recursively transformed into increasingly simpler precursors. Computer-aided retrosynthesis would be a valuable tool but at present it is slow and provides results of unsatisfactory quality. Here we use Monte Carlo tree search and symbolic artificial intelligence (AI) to discover retrosynthetic routes. We combined Monte Carlo tree search with an expansion policy network that guides the search, and a filter network to pre-select the most promising retrosynthetic steps. These deep neural networks were trained on essentially all reactions ever published in organic chemistry. Our system solves for almost twice as many molecules, thirty times faster than the traditional computer-aided search method, which is based on extracted rules and hand-designed heuristics. In a double-blind AB test, chemists on average considered our computer-generated routes to be equivalent to reported literature routes.

Nature, Mar 2018

The AI Revolution is Driven by Deep Learning: Protein Structure Prediction



Article | Open Access | Published: 15 July 2021

Highly accurate protein structure prediction with AlphaFold

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, … Demis Hassabis

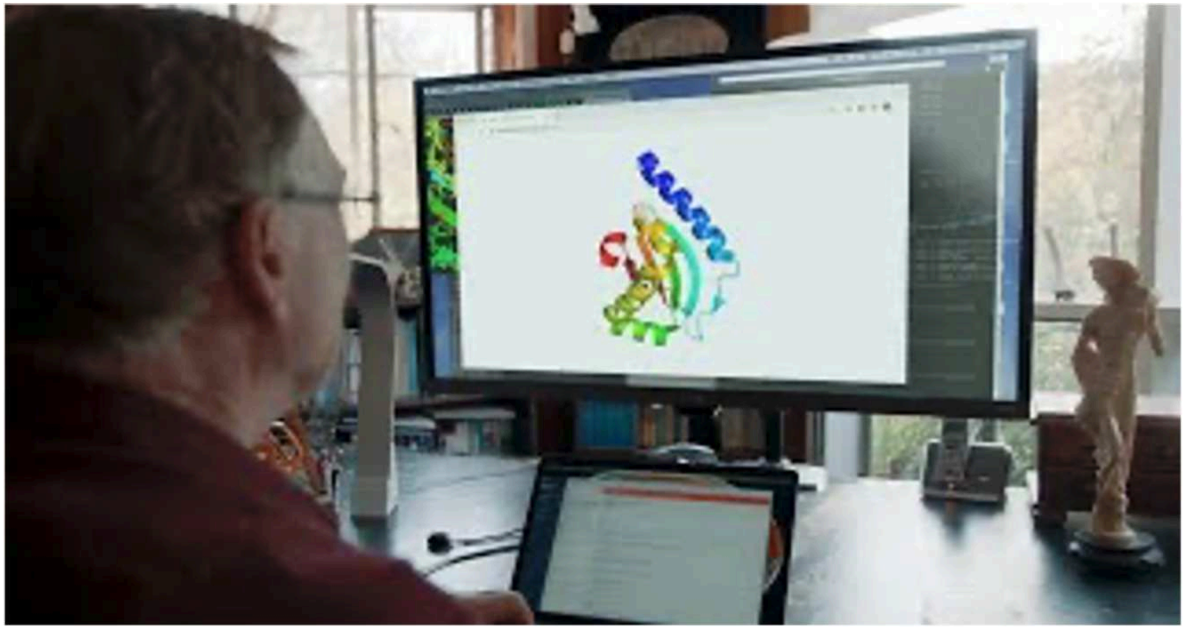
Nature **596**, 583–589 (2021) | Cite this article

1.02m Accesses | **6284** Citations | **3429** Altmetric | Metrics

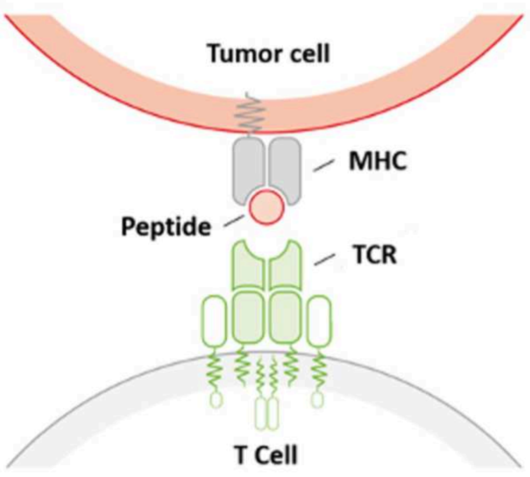
Abstract

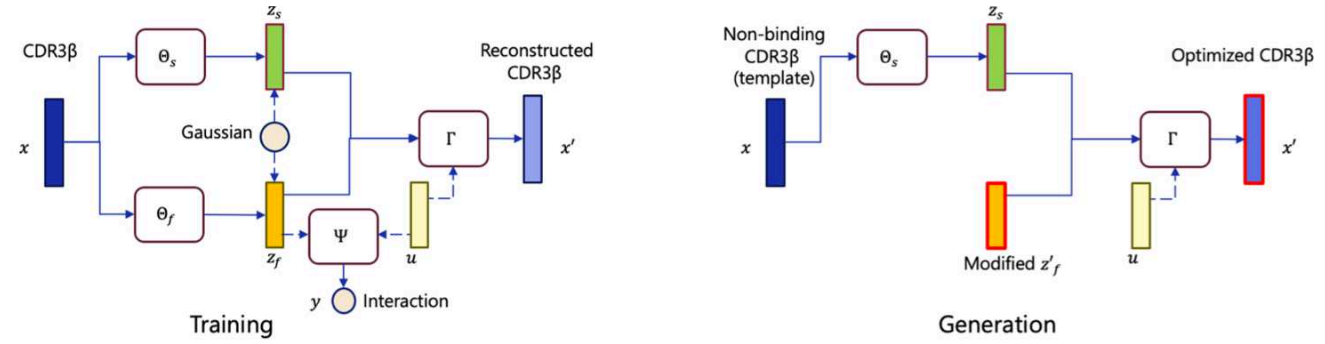
Proteins are essential to life, and understanding their structure can facilitate a mechanistic

The AI Revolution is Driven by Deep Learning: Protein Structure Prediction with AlphaFold2



Deep Learning for Personalized Medicine (joint work with Gerstein Lab)





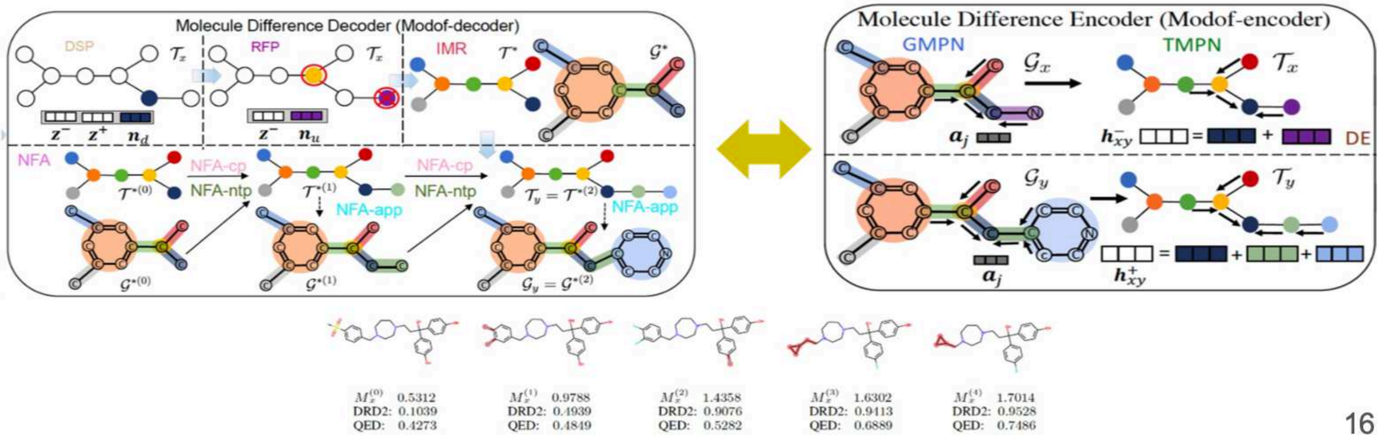
◆ The loss:

- Reconstruction loss $\mathcal{L}_{recon}(x, x')$
- Classification loss $\mathcal{L}_{cls}(\Psi(u, z_f), y)$
- Wasserstein loss $L_{wass}((z_f, z_s), N(0, I))$

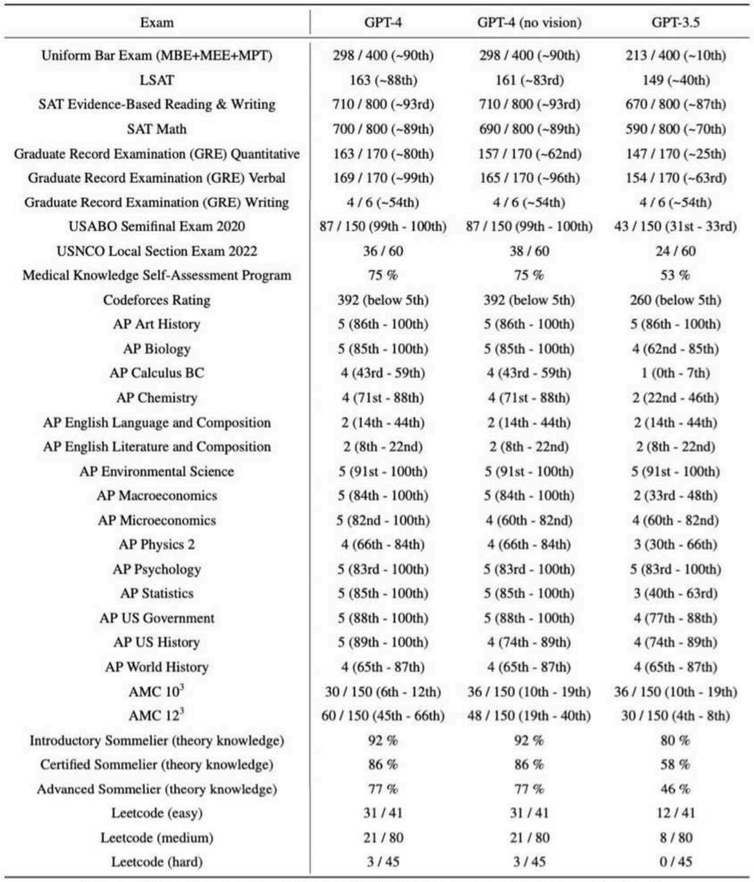
Disentanglement

Learning Disentangled Representations for T Cell Receptor Design. Martin Renqiang Min*#, Tianxiao Li*, Hongyu Guo, Filippo Grazioli, and Mark Gerstein. https://arxiv.org/abs/2210.08171.

Deep Generative Model for Molecule Optimization via One Fragment Modification. Z. Chen, M.R. Min, S. Parthasarathy, X. Ning. Nature Machine Intelligence 2021.

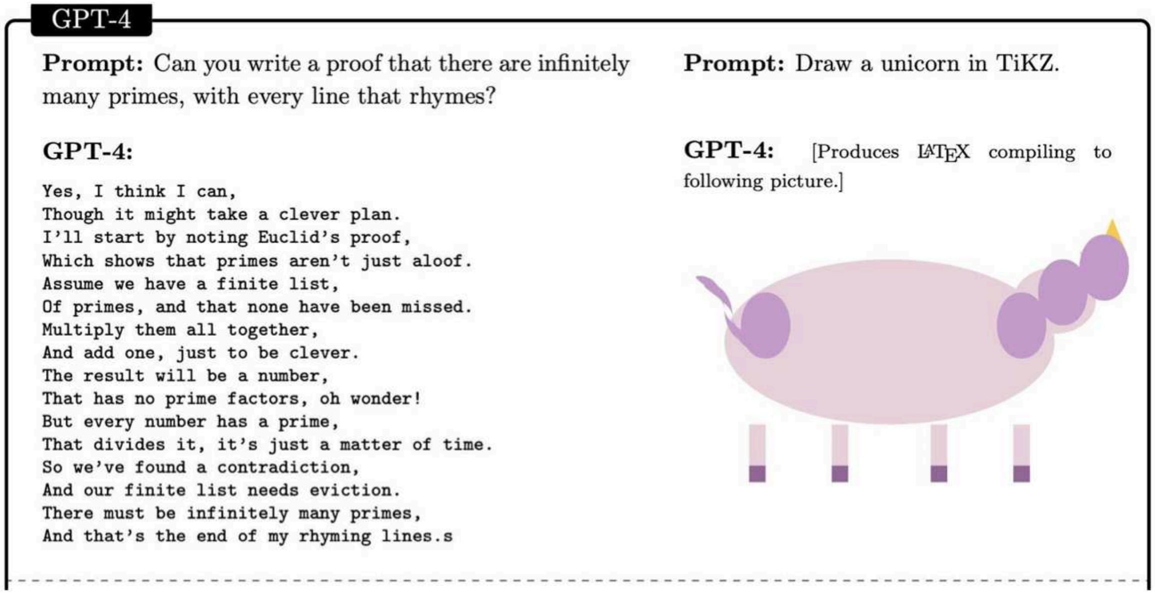

Molecule Difference Decoder (Modof-decoder)
DSP
RFP
IMR
$\mathcal{T}_x$
$\mathcal{T}^*$
$\mathcal{G}^*$
z^-
z^+
n_d
n_u
NFA
NFA-cp
NFA-ntp
NFA-app
$\mathcal{T}^{*(0)}$
$\mathcal{T}^{*(1)}$
$\mathcal{T}_y = \mathcal{T}^{*(2)}$
$\mathcal{G}^{*(0)}$
$\mathcal{G}^{*(1)}$
$\mathcal{G}_y = \mathcal{G}^{*(2)}$
Molecule Difference Encoder (Modof-encoder)
GMPN
TMPN
$\mathcal{G}_x$
$\mathcal{T}_x$
a_j
h^-_{xy}
DE
$\mathcal{G}_y$
$\mathcal{T}_y$
h^+_{xy}
$M_x^{(0)}$ 0.5312 DRD2: 0.1039 QED: 0.4273
$M_x^{(1)}$ 0.9788 DRD2: 0.4939 QED: 0.4849
$M_x^{(2)}$ 1.4358 DRD2: 0.9076 QED: 0.5282
$M_x^{(3)}$ 1.6302 DRD2: 0.9413 QED: 0.6889
$M_x^{(4)}$ 1.7014 DRD2: 0.9528 QED: 0.7486

The AI Revolution is Driven by Deep Learning: Large Language Models (ChatGPT and GPT-4)



Exam	GPT-4	GPT-4 (no vision)	GPT-3.5
Uniform Bar Exam (MBE+MEE+MPT)	298 / 400 (~90th)	298 / 400 (~90th)	213 / 400 (~10th)
LSAT	163 (~88th)	161 (~83rd)	149 (~40th)
SAT Evidence-Based Reading & Writing	710 / 800 (~93rd)	710 / 800 (~93rd)	670 / 800 (~87th)
SAT Math	700 / 800 (~89th)	690 / 800 (~89th)	590 / 800 (~70th)
Graduate Record Examination (GRE) Quantitative	163 / 170 (~80th)	157 / 170 (~62nd)	147 / 170 (~25th)
Graduate Record Examination (GRE) Verbal	169 / 170 (~99th)	165 / 170 (~96th)	154 / 170 (~63rd)
Graduate Record Examination (GRE) Writing	4 / 6 (~54th)	4 / 6 (~54th)	4 / 6 (~54th)
USABO Semifinal Exam 2020	87 / 150 (99th - 100th)	87 / 150 (99th - 100th)	43 / 150 (31st - 33rd)
USNCO Local Section Exam 2022	36 / 60	38 / 60	24 / 60
Medical Knowledge Self-Assessment Program	75 %	75 %	53 %
Codeforces Rating	392 (below 5th)	392 (below 5th)	260 (below 5th)
AP Art History	5 (86th - 100th)	5 (86th - 100th)	5 (86th - 100th)
AP Biology	5 (85th - 100th)	5 (85th - 100th)	4 (62nd - 85th)
AP Calculus BC	4 (43rd - 59th)	4 (43rd - 59th)	1 (0th - 7th)
AP Chemistry	4 (71st - 88th)	4 (71st - 88th)	2 (22nd - 46th)
AP English Language and Composition	2 (14th - 44th)	2 (14th - 44th)	2 (14th - 44th)
AP English Literature and Composition	2 (8th - 22nd)	2 (8th - 22nd)	2 (8th - 22nd)
AP Environmental Science	5 (91st - 100th)	5 (91st - 100th)	5 (91st - 100th)
AP Macroeconomics	5 (84th - 100th)	5 (84th - 100th)	2 (33rd - 48th)
AP Microeconomics	5 (82nd - 100th)	4 (60th - 82nd)	4 (60th - 82nd)
AP Physics 2	4 (66th - 84th)	4 (66th - 84th)	3 (30th - 66th)
AP Psychology	5 (83rd - 100th)	5 (83rd - 100th)	5 (83rd - 100th)
AP Statistics	5 (85th - 100th)	5 (85th - 100th)	3 (40th - 63rd)
AP US Government	5 (88th - 100th)	5 (88th - 100th)	4 (77th - 88th)
AP US History	5 (89th - 100th)	4 (74th - 89th)	4 (74th - 89th)
AP World History	4 (65th - 87th)	4 (65th - 87th)	4 (65th - 87th)
AMC 10[3]	30 / 150 (6th - 12th)	36 / 150 (10th - 19th)	36 / 150 (10th - 19th)
AMC 12[3]	60 / 150 (45th - 66th)	48 / 150 (19th - 40th)	30 / 150 (4th - 8th)
Introductory Sommelier (theory knowledge)	92 %	92 %	80 %
Certified Sommelier (theory knowledge)	86 %	86 %	58 %
Advanced Sommelier (theory knowledge)	77 %	77 %	46 %
Leetcode (easy)	31 / 41	31 / 41	12 / 41
Leetcode (medium)	21 / 80	21 / 80	8 / 80
Leetcode (hard)	3 / 45	3 / 45	0 / 45

Table 1. GPT performance on academic and professional exams. In each case, we simulate the conditions and scoring of the real exam. We report GPT-4's final score graded according to exam-specific rubrics, as well as the percentile of test-takers achieving GPT-4's score.



GPT-4

Prompt: Can you write a proof that there are infinitely many primes, with every line that rhymes?

GPT-4:

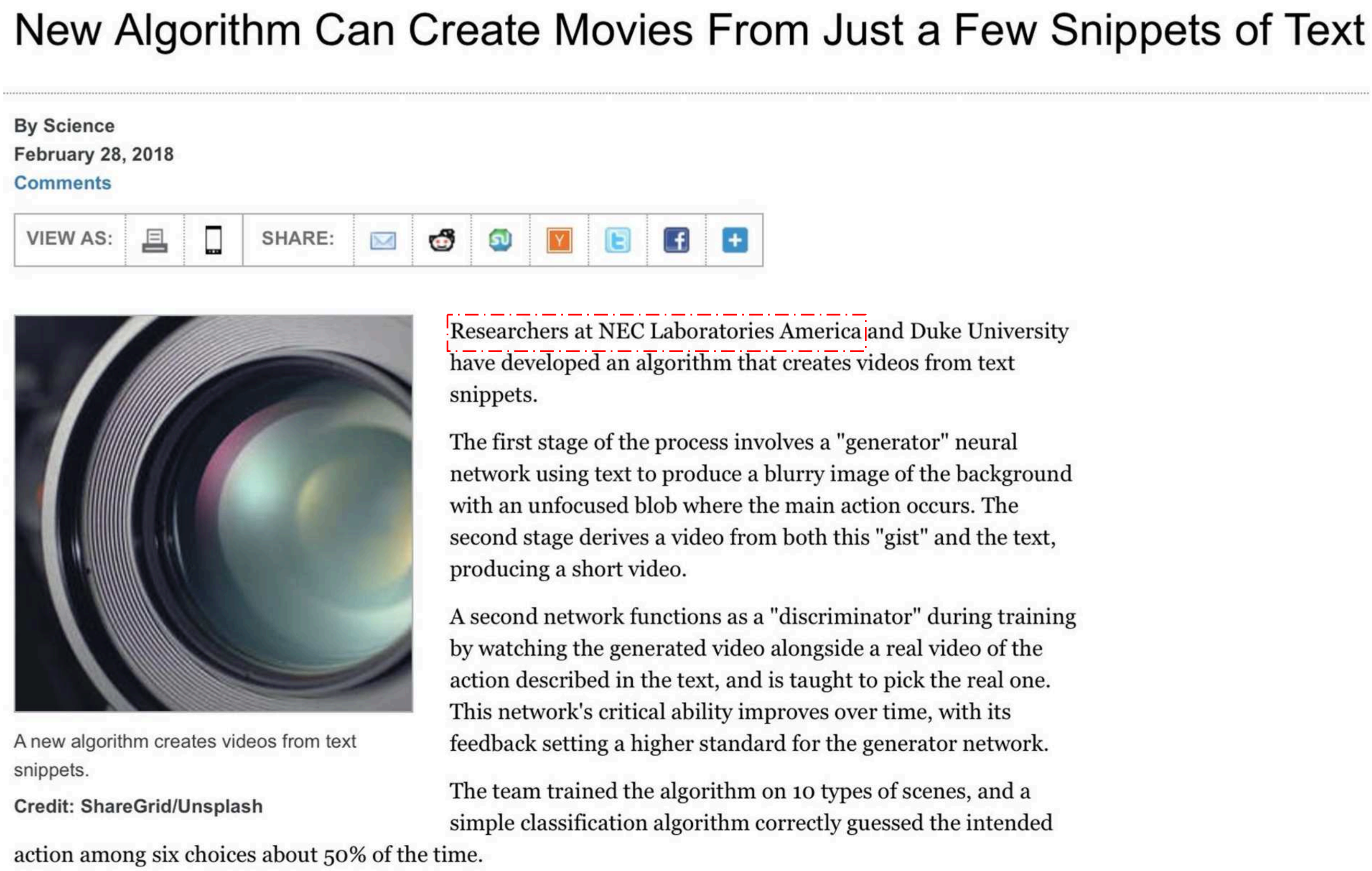
```
Yes, I think I can,
Though it might take a clever plan.
I'll start by noting Euclid's proof,
Which shows that primes aren't just aloof.
Assume we have a finite list,
Of primes, and that none have been missed.
Multiply them all together,
And add one, just to be clever.
The result will be a number,
That has no prime factors, oh wonder!
But every number has a prime,
That divides it, it's just a matter of time.
So we've found a contradiction,
And our finite list needs eviction.
There must be infinitely many primes,
And that's the end of my rhyming lines.s
```

Prompt: Draw a unicorn in TiKZ.

GPT-4: [Produces LaTeX compiling to following picture.]

https://cdn.openai.com/papers/gpt-4.pdf

https://arxiv.org/pdf/2303.12712.pdf



New Algorithm Can Create Movies From Just a Few Snippets of Text

By Science
February 28, 2018
Comments

Researchers at NEC Laboratories America and Duke University have developed an algorithm that creates videos from text snippets.

The first stage of the process involves a "generator" neural network using text to produce a blurry image of the background with an unfocused blob where the main action occurs. The second stage derives a video from both this "gist" and the text, producing a short video.

A second network functions as a "discriminator" during training by watching the generated video alongside a real video of the action described in the text, and is taught to pick the real one. This network's critical ability improves over time, with its feedback setting a higher standard for the generator network.

The team trained the algorithm on 10 types of scenes, and a simple classification algorithm correctly guessed the intended action among six choices about 50% of the time.

A new algorithm creates videos from text snippets.

Credit: ShareGrid/Unsplash

A sample video generated by Meta's new AI text-to-video model, Make-A-Video. The text prompt used to create the video was "a teddy bear painting a portrait."

Image: Meta

Deep Learning Methods Can Be Easily Fooled

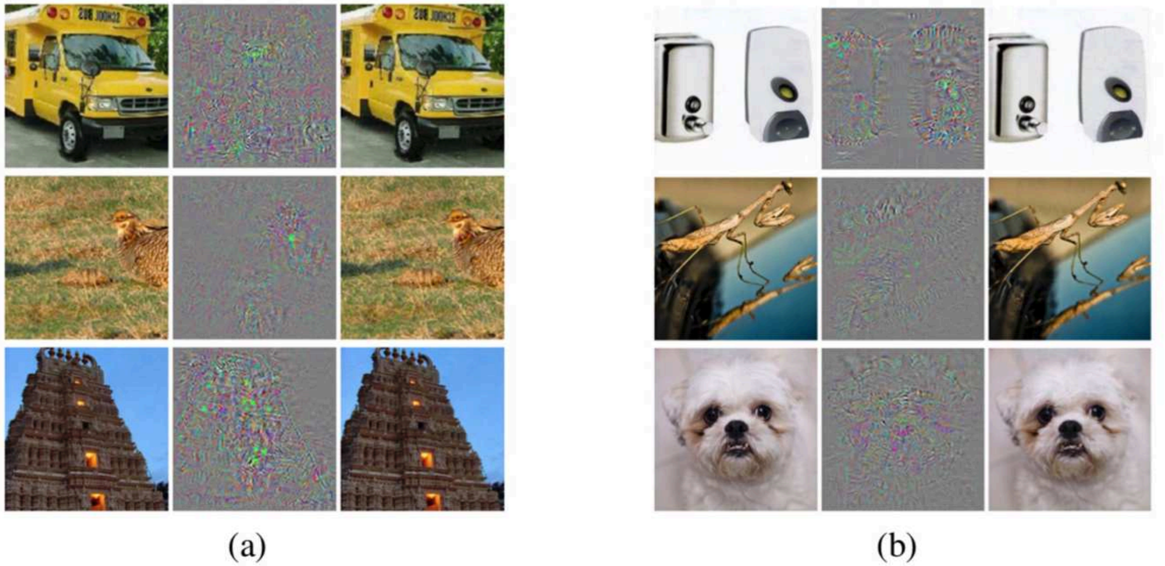
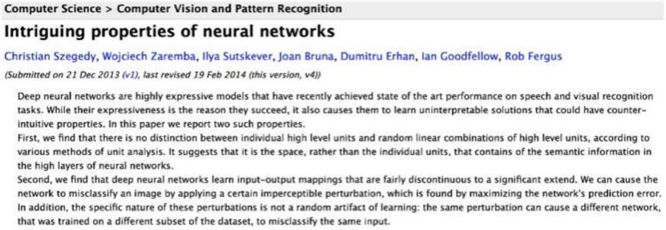

(a) (b)

Figure 5: Adversarial examples generated for AlexNet [9].(Left) is a correctly predicted sample, (center) difference between correct image, and image predicted incorrectly magnified by 10x (values shifted by 128 and clamped), (right) adversarial example. All images in the right column are predicted to be an *"ostrich, Struthio camelus"*. Average distortion based on 64 examples is 0.006508. Plase refer to `http://goo.gl/huaGPb` for full resolution images. The examples are strictly randomly chosen. There is not any postselection involved.


Computer Science > Computer Vision and Pattern Recognition

Intriguing properties of neural networks

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus

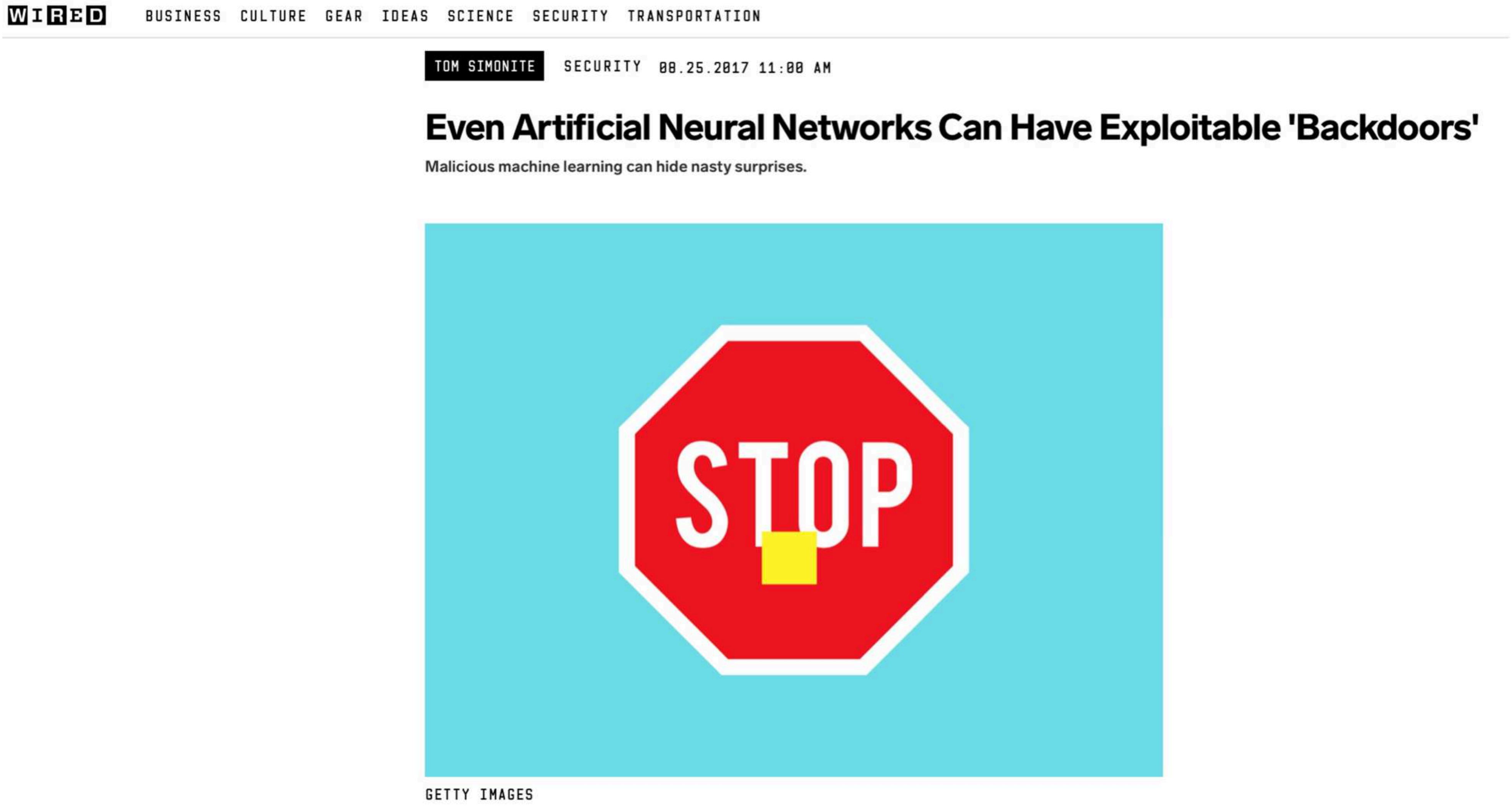
(Submitted on 21 Dec 2013 (v1), last revised 19 Feb 2014 (this version, v4))

Deep neural networks are highly expressive models that have recently achieved state of the art performance on speech and visual recognition tasks. While their expressiveness is the reason they succeed, it also causes them to learn uninterpretable solutions that could have counter-intuitive properties. In this paper we report two such properties.

First, we find that there is no distinction between individual high level units and random linear combinations of high level units, according to various methods of unit analysis. It suggests that it is the space, rather than the individual units, that contains of the semantic information in the high layers of neural networks.

Second, we find that deep neural networks learn input-output mappings that are fairly discontinuous to a significant extend. We can cause the network to misclassify an image by applying a certain imperceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input.

Deep Learning Methods Can Be Easily Fooled

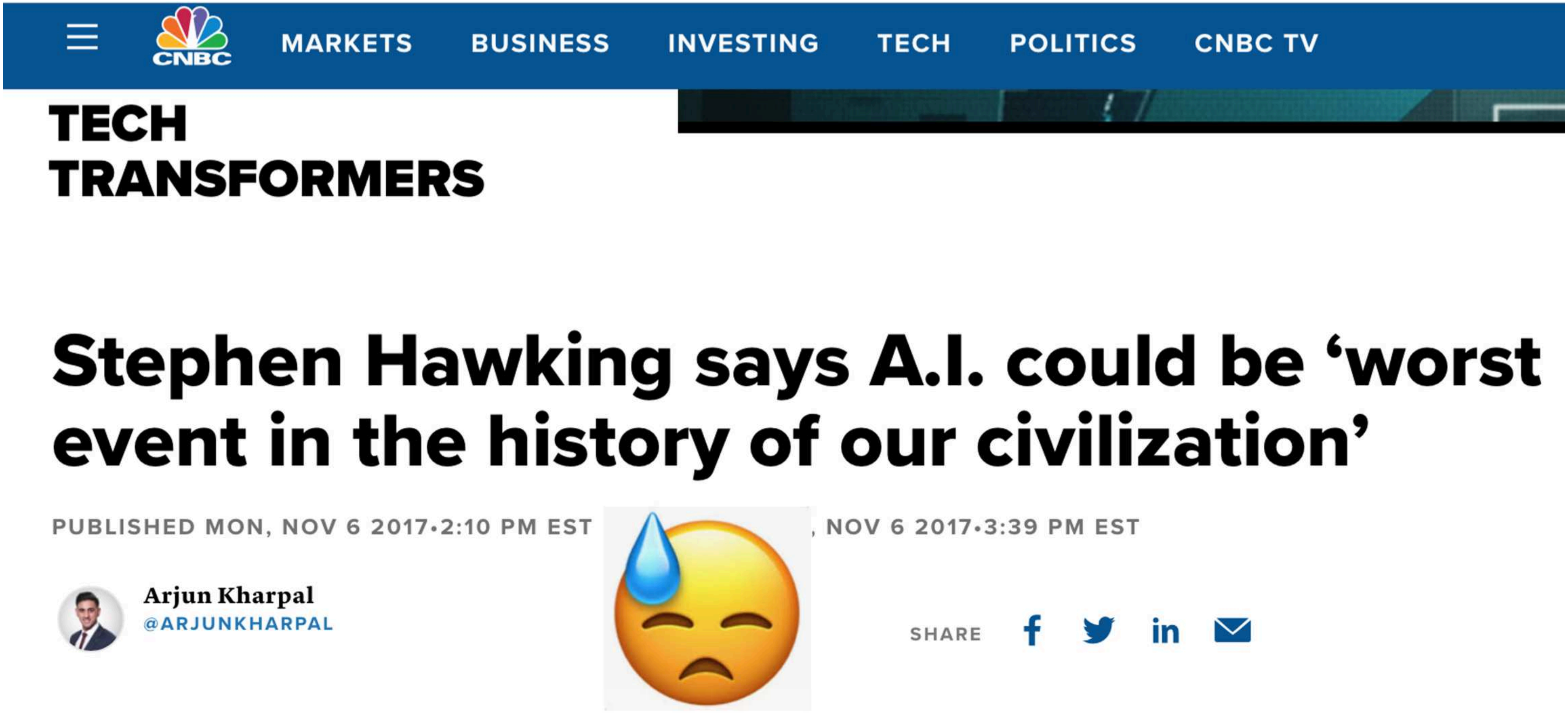


TOM SIMONITE SECURITY 08.25.2017 11:00 AM

Even Artificial Neural Networks Can Have Exploitable 'Backdoors'

Malicious machine learning can hide nasty surprises.

GETTY IMAGES



TECH
TRANSFORMERS

Stephen Hawking says A.I. could be ‘worst event in the history of our civilization’

PUBLISHED MON, NOV 6 2017•2:10 PM EST , NOV 6 2017•3:39 PM EST

Arjun Kharpal
@ARJUNKHARPAL

Face with Cold Sweat Emoji (U…
iemoji.com

Behind the Scene: Deep Learning

- Deep Learning:
  - The driving force of all these technological advancements
  - The root cause of all these controversial debates and potential dangers

Deep Learning is a Subfield of Machine Learning

- What is Machine Learning
    - Machine learning is about teaching computers to perform tasks by only showing them data/examples without explicitly programming instructions
    - Machine learning, data science, and computer science are more and more related to each other
    - Machine Learning: Instead of programming computers, let computers learn to program by themselves by showing them some examples
        - Feature Representation + Objective Function + Optimization (training data) □ Generalization (test data)

Three Types of Machine Learning Tasks

- Supervised Learning:
  - Given training examples of input feature vectors and corresponding target outputs, predict outputs on future inputs. For e.g., classification, regression, time series prediction.
- Unsupervised Learning:
  - Given only input feature vectors, automatically discover representations, structures, etc. For e.g., clustering, data compression, outlier detection
- Reinforcement Learning:
  - Given sequences of [inputs from an environment, actions from a fixed set, and occasional scalar rewards], learn to select action sequences in a way that maximizes the expected sum of (discounted) future reward

Machine Learning Models

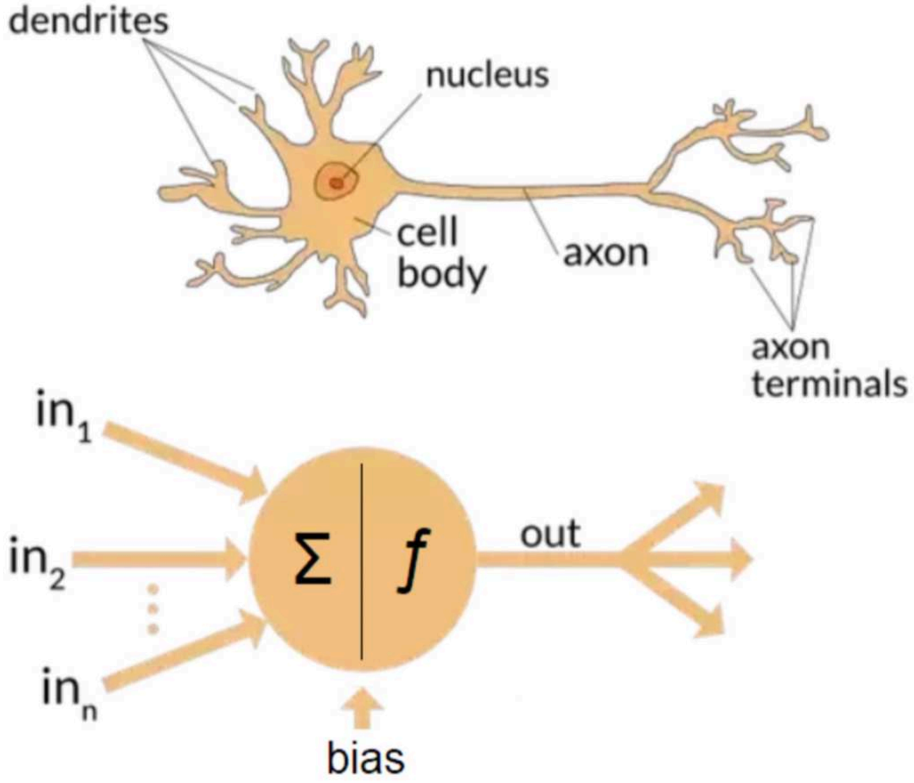
- Supervised Learning:
    - Linear Regression, Logistic Regression, Naïve Bayes, Linear Discriminant Analysis, k Nearest Neighbor, (kernel) SVM, Decision Tree, Random Forest, Multilayer Perceptron (Neural Network for classification/regression), Conditional Random Field
- Unsupervised Learning:
    - K-means, Gaussian Mixture Model, Hierarchical Clustering Methods, Principal Component Analysis, Independent Component analysis, Vector Quantification (Data Compression), Latent Dirichlet Allocation, (Denoising) Autoencoder, (Restricted) Boltzmann Machines
- Reinforcement Learning:
    - Q Learning, Policy Gradient Methods, REINFORCE

What is Deep Learning

- Deep learning
    - One of many research areas in Machine Learning
    - Deep Learning focuses on learning data representations (adaptive features) using deep neural networks
    - Deep neural networks are standard neural networks with many hidden layers
    - Deep learning can be applied to any type of learning task of machine learning

Since deep learning is a modern name of deep neural network invented decades ago, I will review the history of (deep) neural network and introduce the backpropagation algorithm for training deep learning models.

Modeling a Biological Neuron



A biological and an artificial neuron (via https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network)

https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

A Linear Neuron

bias

i th input

$$y = b + \sum_i x_i w_i$$

weight on

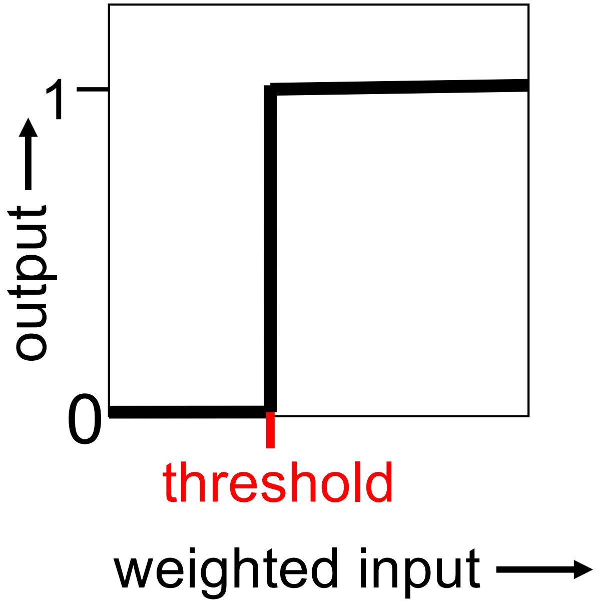
ith input

output

index over
input connections

Binary Threshold Neuron

- McCulloch-Pitts (1943): influenced Von Neumann.
  - First compute a weighted sum of the inputs.
  - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.



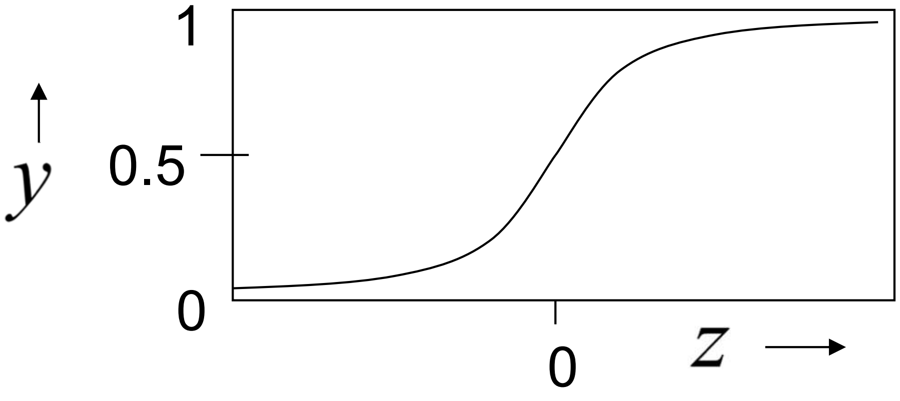
$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

A Sigmoid Neuron

- It gives a real-valued output that is a smooth and bounded function of its total input.
    - It has a nice probabilistic interpretation ([0, 1]).
    - It has nice derivatives that make learning easy (discuss it later).

$$z = b + \sum_i x_i w_i \qquad y = \frac{1}{1+e^{-z}}$$



Rosenblatt’s Perceptron (1958)

- A linear neuron

Learning algorithm for binary classification with a binary output neuron (1 or 0):

1. Initialize the weights and threshold to 0 or small random values
2. For each example i in the training set, perform the following steps:
    - Calculate the model output for input i using the current model based on the linear neuron
    - If the output is correct, do nothing
    - If the model incorrectly outputs 0, add the feature vector of i to the weight vector
    - If the model incorrectly outputs 1, minus the feature vector of i to the weight vector

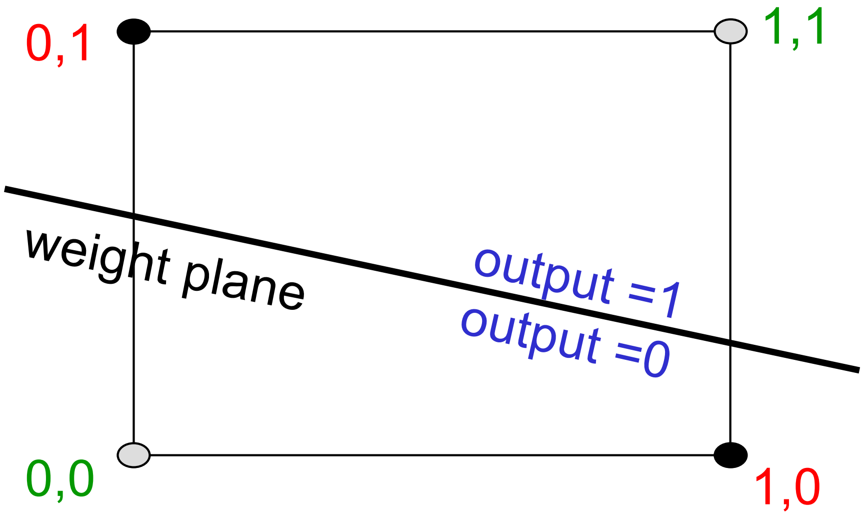
    [(ground-truth label – predicted label) * input feature vector x]
3. Repeat step 2 until some convergence criteria is met

Perceptron can’t even solve XOR!

Input		Output
x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0

A geometric view of what binary threshold neurons cannot do

- Imagine “data-space” in which the axes correspond to components of an input vector.
    - Each input vector is a point in this space.
    - A weight vector defines a plane in data-space.
    - The weight plane is perpendicular to the weight vector and misses the origin by a distance equal to the threshold.



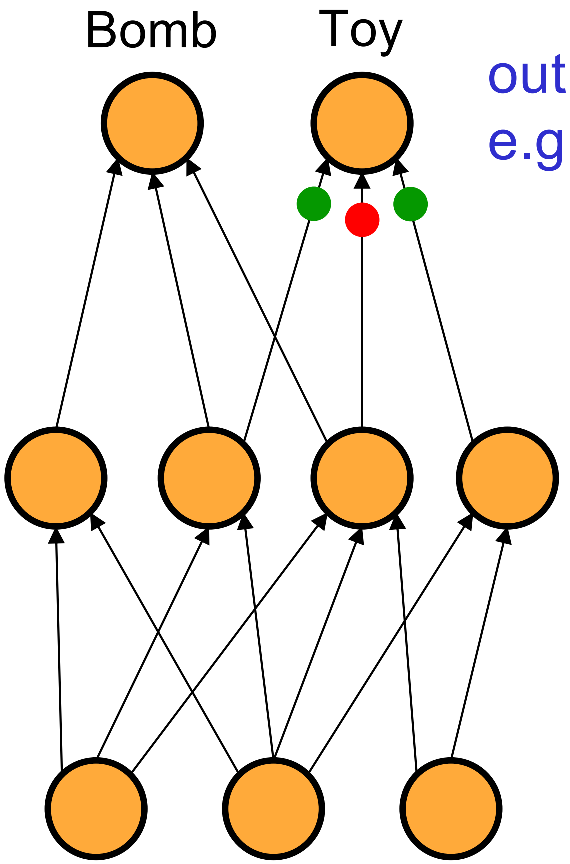
The positive and negative cases cannot be separated by a plane

What should we do? Handcraft Features (~1960s).

$$\phi_1(\mathbf{x}) = x_1$$
$$\phi_2(\mathbf{x}) = x_2$$
$$\phi_3(\mathbf{x}) = x_1 x_2$$

In this representation, our training set becomes

$\phi_1(\mathbf{x})$	$\phi_2(\mathbf{x})$	$\phi_3(\mathbf{x})$	t
0	*0*	*0*	*0*
0	*1*	*0*	*1*
1	*0*	*0*	*1*
1	*1*	*1*	*0*



output units
e.g. class labels

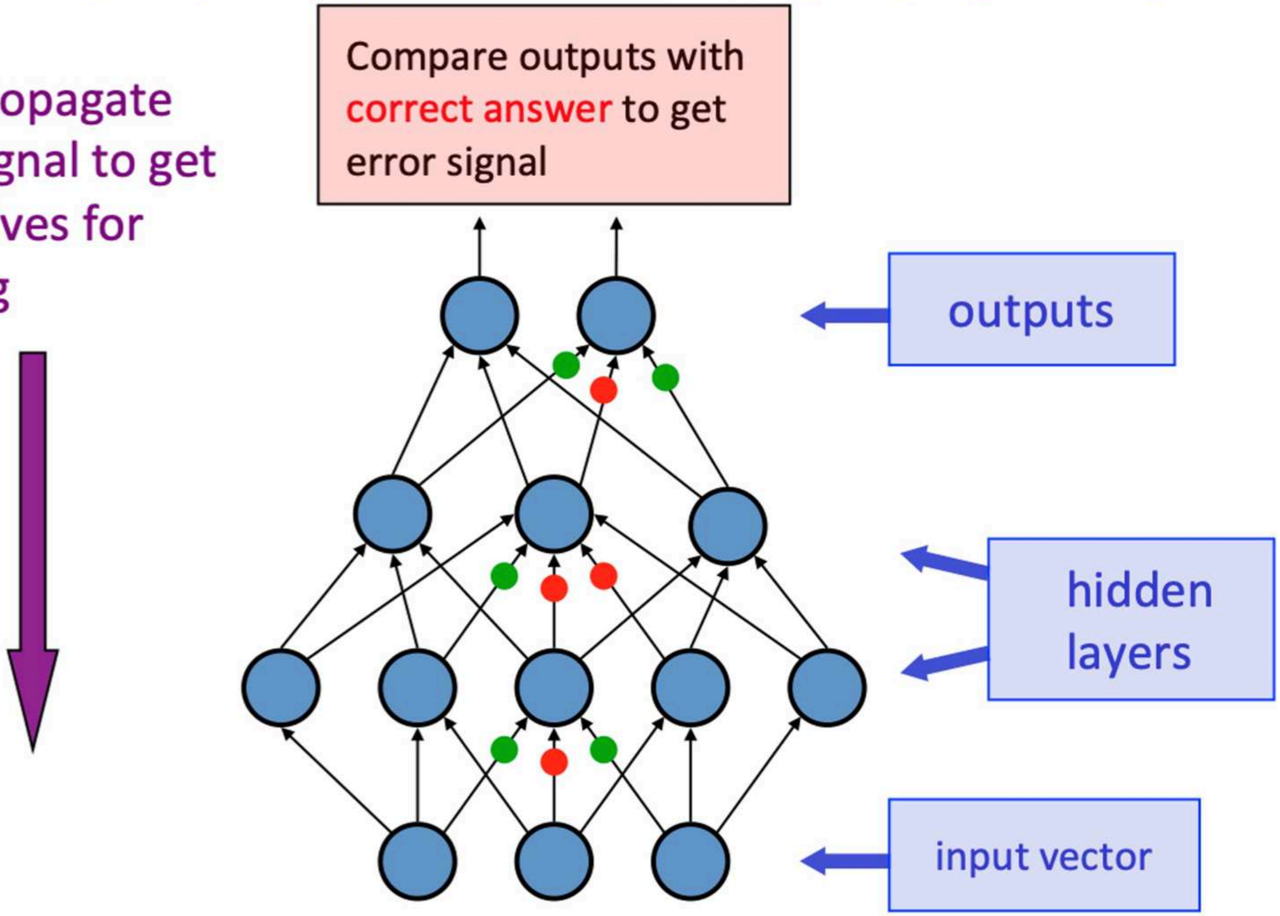
non-adaptive
hand-coded
features

input units
e.g. pixels

Learning with Hidden Units (Hinton)

- Networks without hidden units are very limited in the input-output mappings they can learn to model.
    - More layers of linear units do not help. Its still linear.
    - Fixed output non-linearities are not enough.
- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?
    - We need an efficient way of adapting all the weights, not just the last layer. This is hard.
    - Learning the weights going into hidden units is equivalent to learning features.
    - This is difficult because nobody is telling us directly what the hidden units should do.

Second generation neural networks (~1985, multilayer perceptron with backpropagation)



Back-propagate error signal to get derivatives for learning

Compare outputs with correct answer to get error signal

outputs

hidden layers

input vector

Rumelhart, Hinton, and Williams, Nature 1986

Benefits of MLP with Backpropagation

- Backpropagation allows neural networks to design their own features and multiple layers of features adaptively

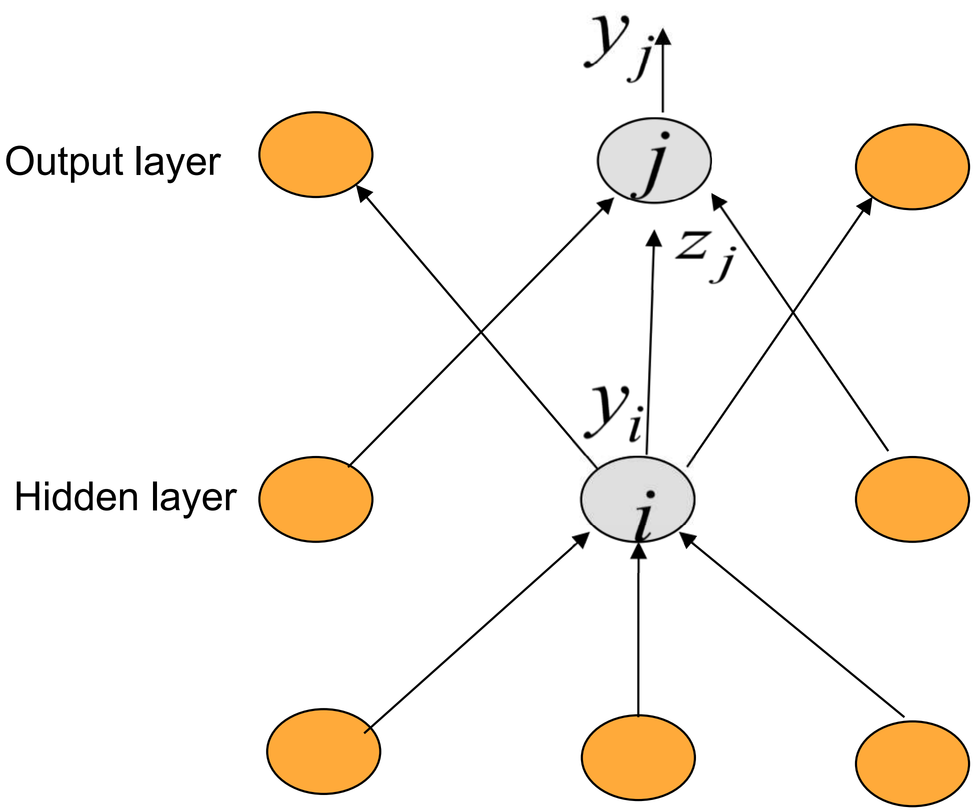
- No domain knowledge is needed. You just need to show the computer some training examples.

- Backpropagation is an efficient algorithm for computing how weight vectors should be updated to effect the output error
    - Instead of naively updating one weight at a time, BP computes the gradient of the weights in parallel and updates all weights through an efficient forward pass and backward pass

The idea behind backpropagation (Hinton)

- We don’t know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
- We can compute error derivatives for all the hidden units efficiently at the same time.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

Backpropagating dE/dy (Hinton)



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j\,(1-y_j)\,\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

A Linear Neuron in Matrix-Vector Form

weight
vector

$$y = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

neuron’s
estimate of the
desired output

input
vector

The Delta Rule For a Linear Neuron

- Define the error as the squared residuals summed over all training cases:

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$

- Now differentiate to get error derivatives for weights

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n}$$

$$= -\sum_n x_i^n \, (t^n - y^n)$$

- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon \, x_i^n (t^n - y^n)$$

The derivatives of a logistic neuron

- The derivatives of the logit, z, with respect to the inputs and the weights are very simple:

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

$$\frac{dy}{dz} = y\,(1 - y)$$

The derivatives of a logistic neuron

$$y = \frac{1}{1+e^{-z}} = (1+e^{-z})^{-1}$$

$$\frac{dy}{dz} = \frac{-1(-e^{-z})}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y)$$

because $\frac{e^{-z}}{1+e^{-z}} = \frac{(1+e^{-z})-1}{1+e^{-z}} = \frac{(1+e^{-z})}{1+e^{-z}} \frac{-1}{1+e^{-z}} = 1-y$

Using the chain rule to get the derivatives needed for learning the weights of a logistic unit

- To learn the weights we need the derivative of the output with respect to each weight:

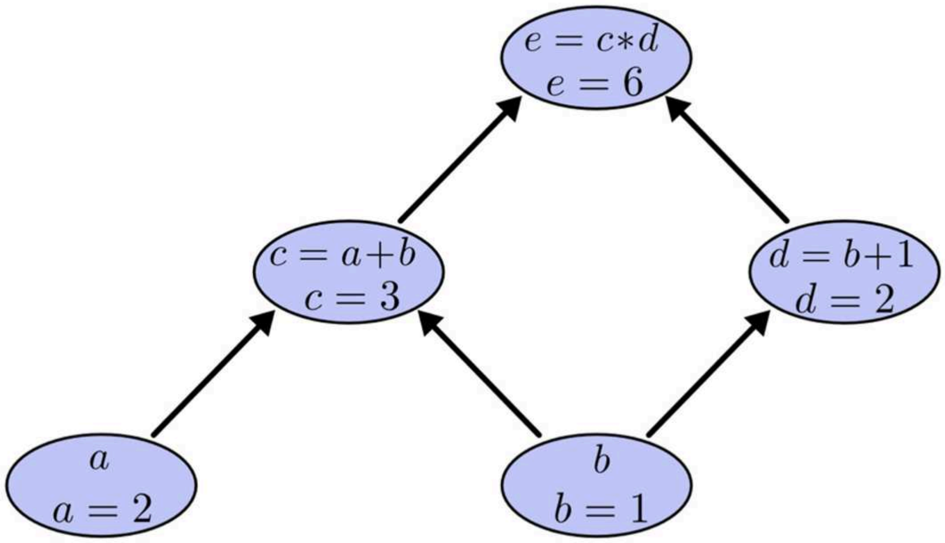
$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i\, y\,(1-y)$$

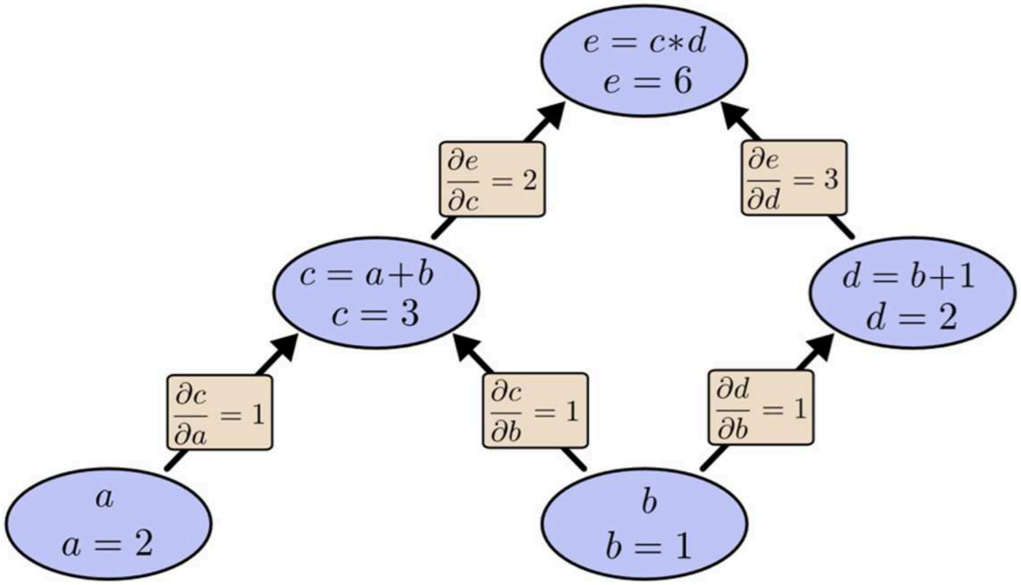
delta-rule

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i}\frac{\partial E}{\partial y^n} = -\sum_n \boxed{x_i^n}\ \boxed{y^n\,(1-y^n)}\ \boxed{(t^n - y^n)}$$

extra term = slope of logistic

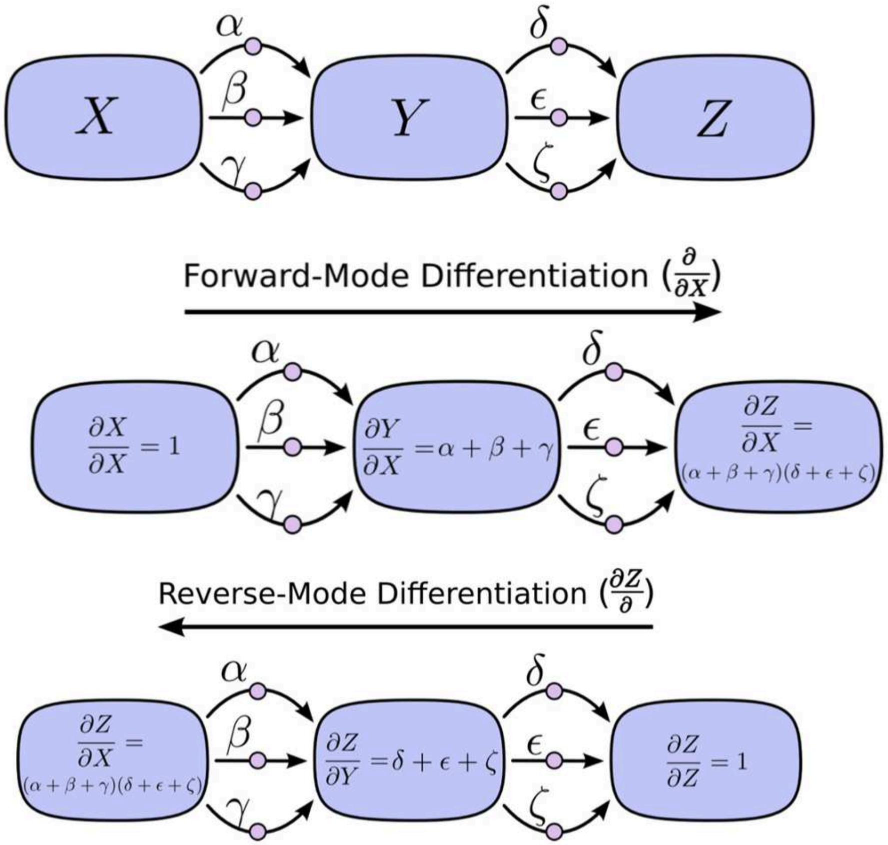
BP in a Computational Graph




$e = c*d$
$e = 6$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$c = a+b$
$c = 3$
$d = b+1$
$d = 2$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
a
$a = 2$
b
$b = 1$

http://colah.github.io/posts/2015-08-Backprop/

BP in a Computational Graph


α
β
γ
δ
ϵ
ζ
X
Y
Z
Forward-Mode Differentiation ($\frac{\partial}{\partial X}$)
$\frac{\partial X}{\partial X} = 1$
$\frac{\partial Y}{\partial X} = \alpha + \beta + \gamma$
$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$
Reverse-Mode Differentiation ($\frac{\partial Z}{\partial}$)
$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$
$\frac{\partial Z}{\partial Y} = \delta + \epsilon + \zeta$
$\frac{\partial Z}{\partial Z} = 1$

http://colah.github.io/posts/2015-08-Backprop/

Regularized Deep Autoencoder in 2005

Chapter 2

Regularized Autoencoder

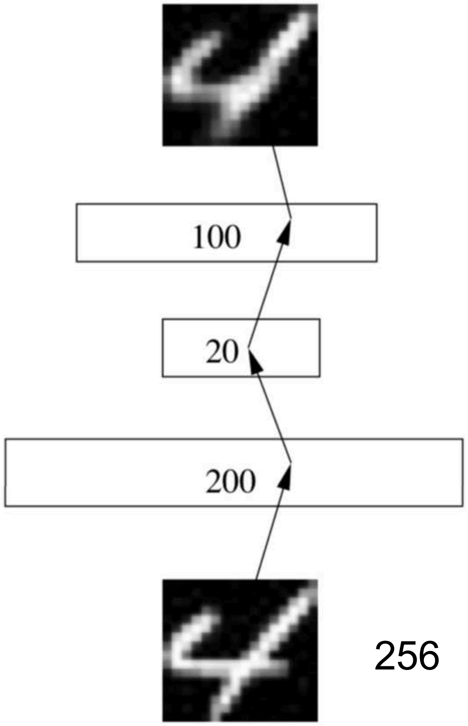


Figure 4.3: The configuration of the autoencoder and RAN for digit data.

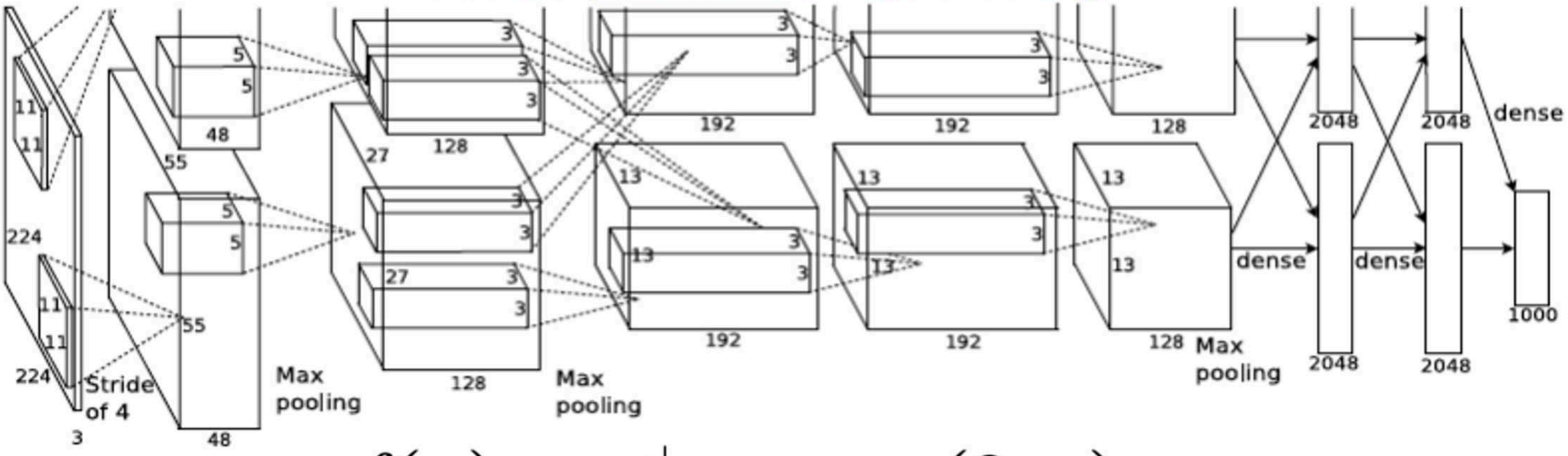
~83K parameters

~1 month training on
9K USPS digits with
a Desktop machine
from 2003 to 2005

Modern Deep Convolutional Neural Networks (2012-now, big data + **deep CNN** + **rectified linear** + **data augmentation** + dropout + GPUs)

Deep learning on GPU (Krizhevsky et al. NIPS12)

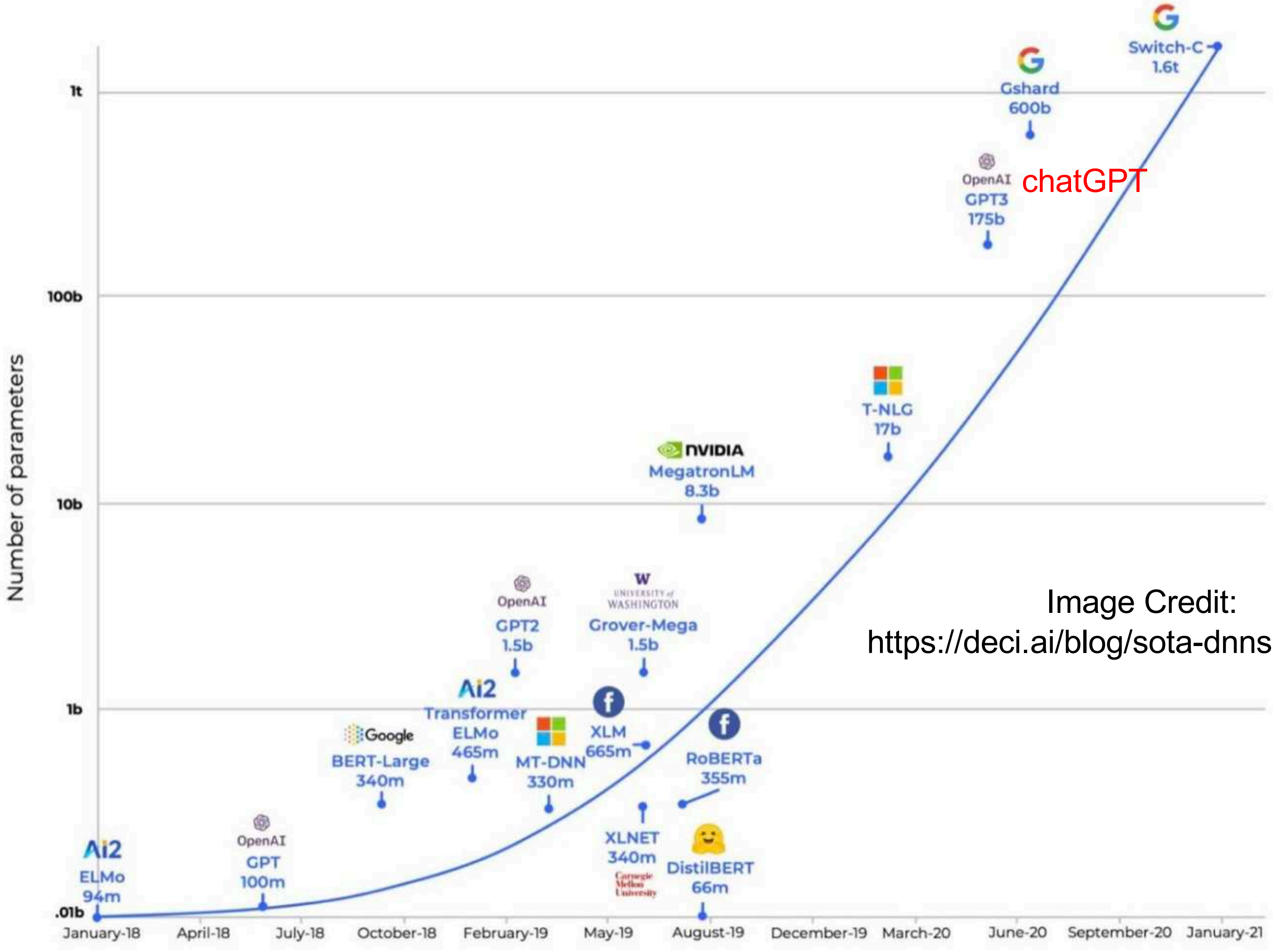
Revolutionary success on the ImageNet Challenge



$$f(x) = x^+ = \max(0, x)$$

60 million parameters and 650,000 neurons

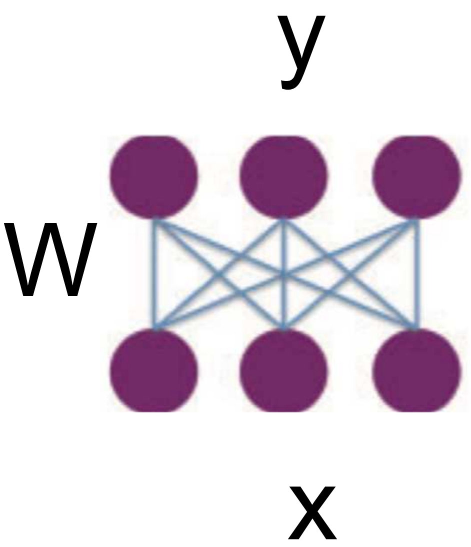
Parameter growth of large language models based on deep learning



chatGPT

Image Credit:
https://deci.ai/blog/sota-dnns-overview/

Fully Connected Layer

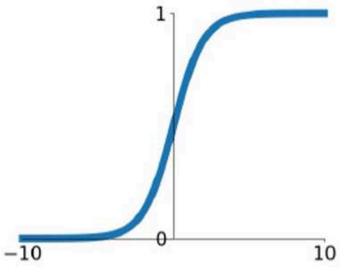


$$y = W x$$

Activation Functions

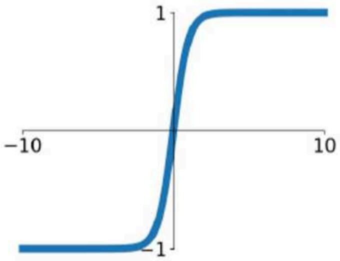
Sigmoid

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



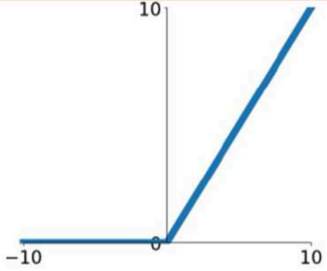
tanh

$\tanh(x)$



ReLU

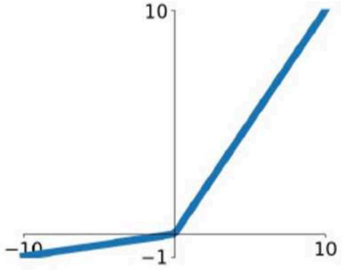
$\max(0, x)$



Good default choice

Leaky ReLU

$\max(0.1x, x)$

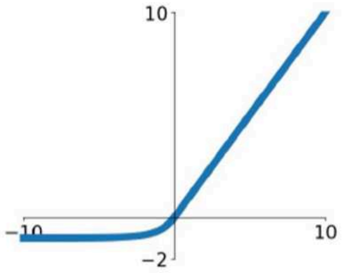


Maxout

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

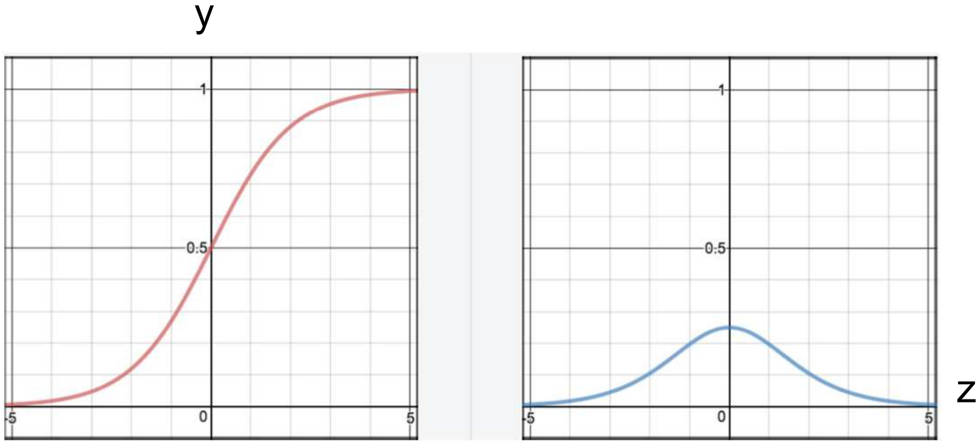
ELU

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



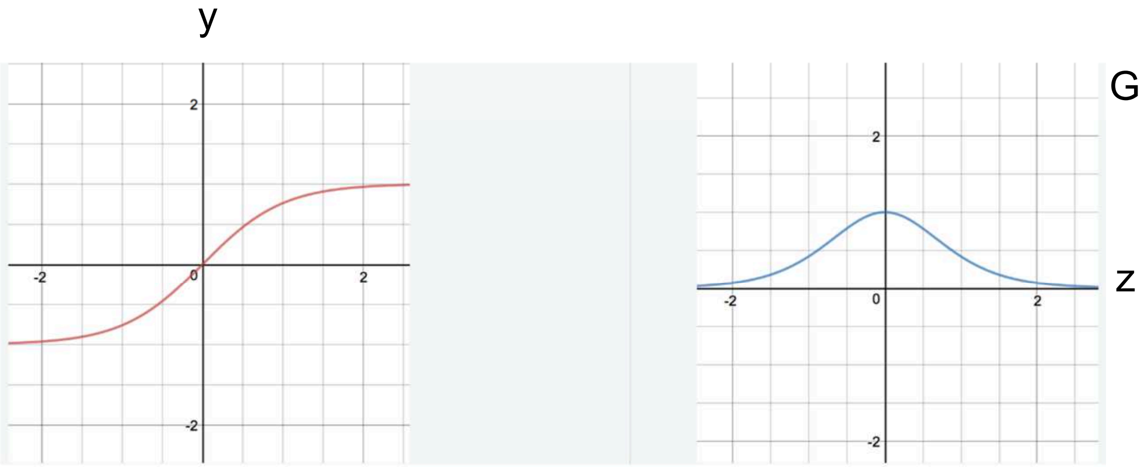
DNN with sigmoid and tanh activation functions has serious vanishing gradient and saturation issue

$$y = \frac{1}{1+e^{-z}}$$



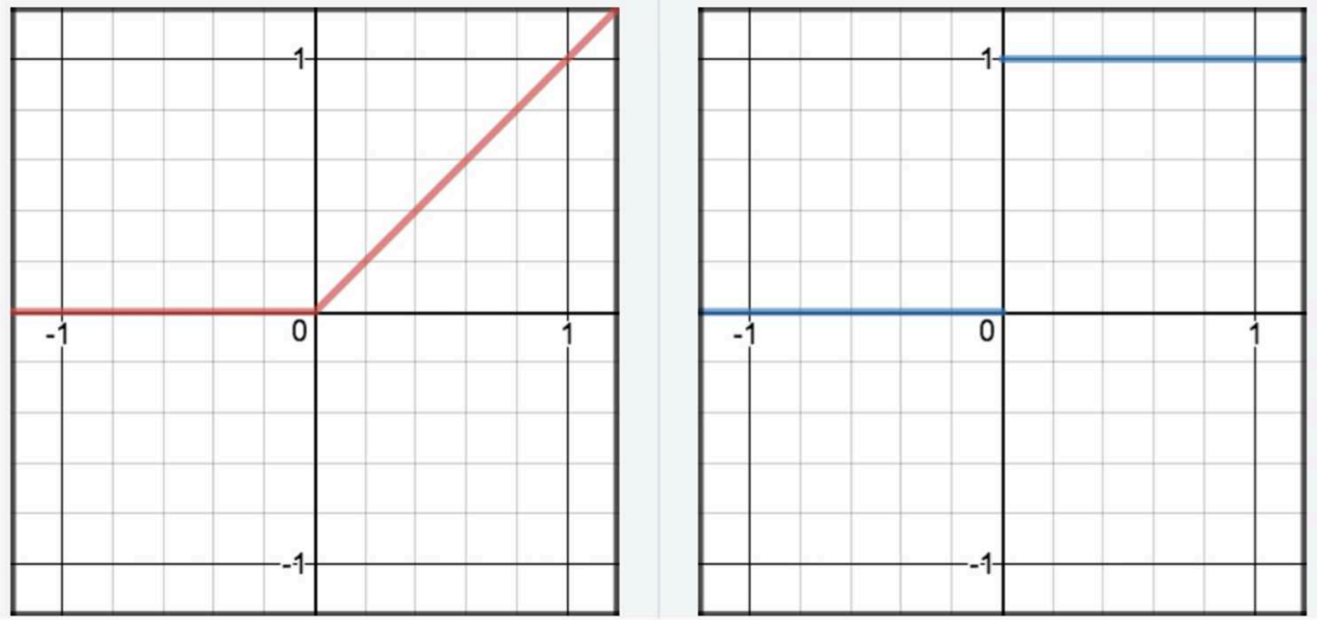
Grad = y(1-y)

$$y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Grad = $1-y^2$

ReLU Activation Function

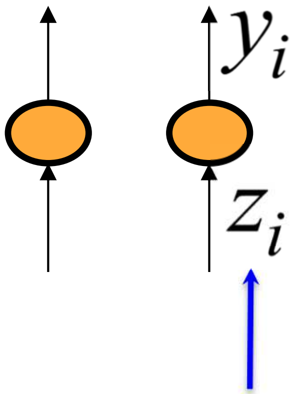


Avoid vanishing gradient and less computationally expensive than sigmoid and tanh

But it might cause dead neuron and the activity is not bounded above

Softmax Activation Function

The output units in a softmax group use a non-local non-linearity:



softmax group

this is called the “logit”

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i \left(1 - y_i\right)$$

Often used on top of a fully connected layer, which transforms an activity vector **z** into probabilities of classifying **x** into K classes

Loss Function: Cross-Entropy Loss

The right cost function is the negative log probability of the target class.

C has a very big gradient when the target value is 1 and the output is almost zero.

A value of 0.001 is much better than 0.0000001

The steepness of dC/dy exactly balances the flatness of dy/dz

$$C = -\sum_j t_j \log y_j$$

Target Class

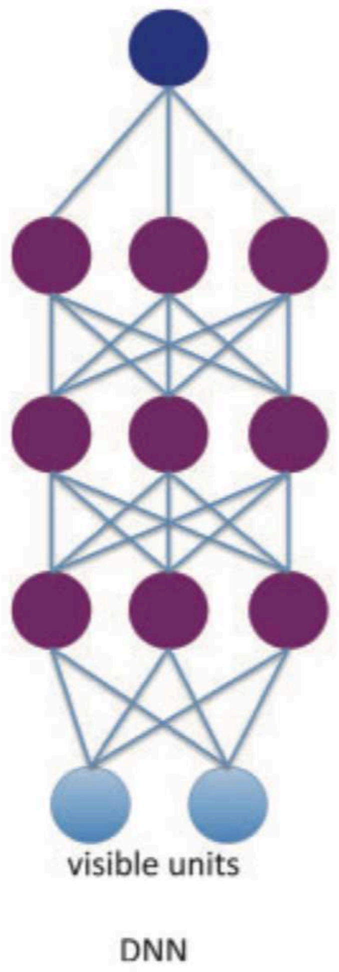
$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Loss Function: Mean Squared Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE is a very bad cost function for softmax output units.
Why?

Deep Feedforward Neural Network with Sigmoid Hidden Units



y

h3
w3

h2

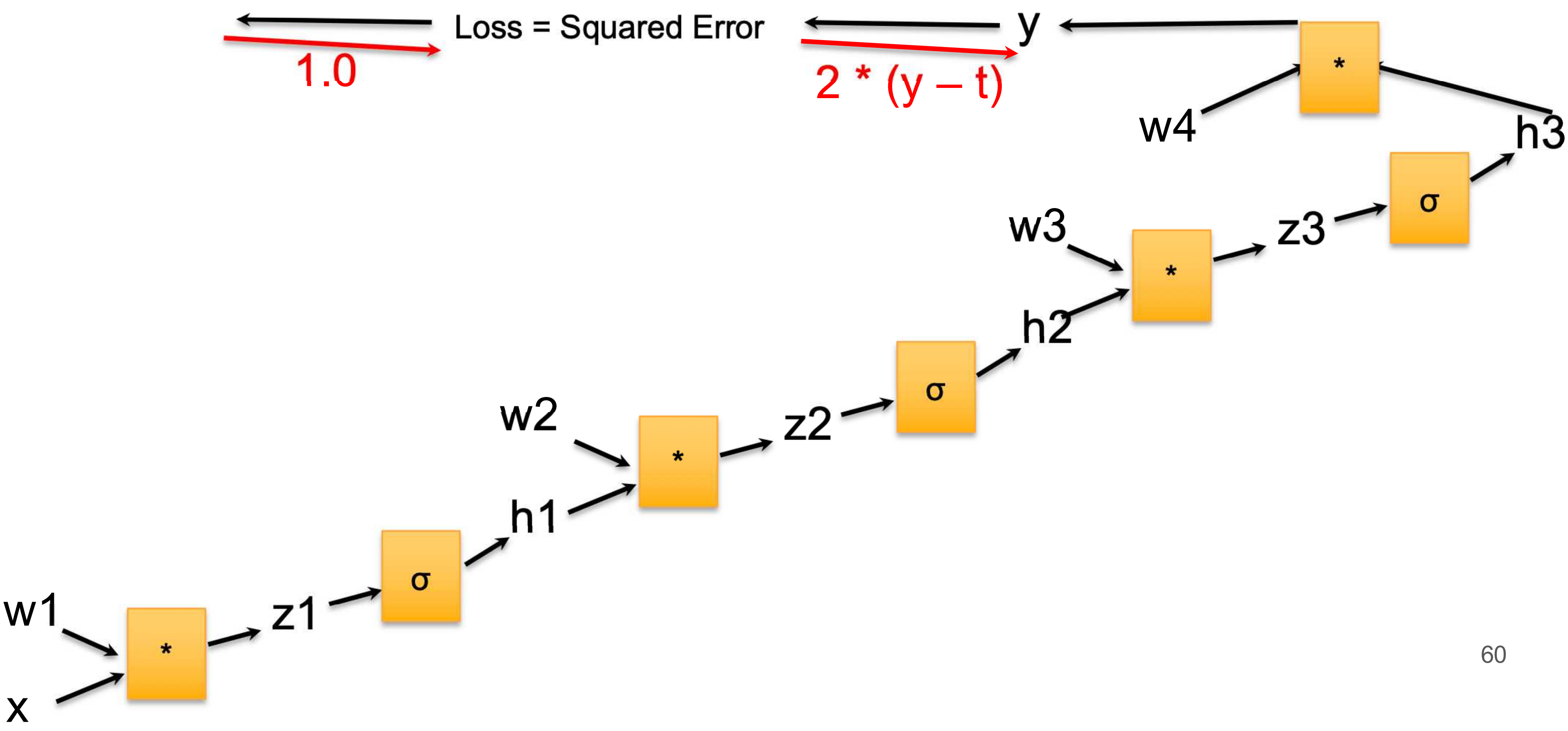
w2

h1

w1

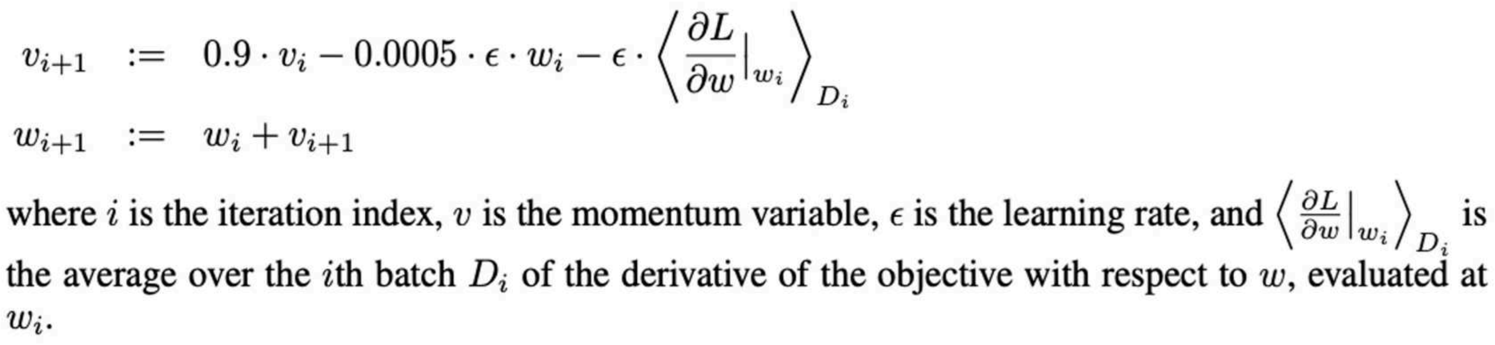
x

Backpropagation with a Computational Graph



Train a Deep Neural Network with SGD

```
Split our training dataset into N mini-batches with batch size b
For Iteration = 1, …, Num_Max_Iterations
    randomly choose a mini-batch $D_i$
```

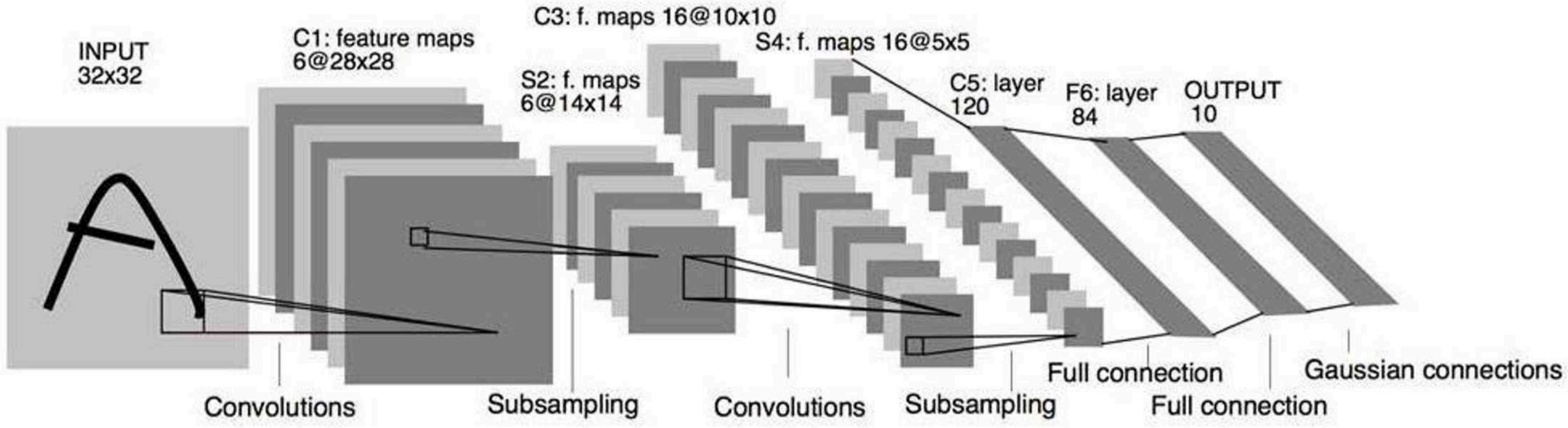


$$
\begin{aligned}
v_{i+1} &:= 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i} \\
w_{i+1} &:= w_i + v_{i+1}
\end{aligned}
$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\left\langle \frac{\partial L}{\partial w}\big|_{w_i} \right\rangle_{D_i}$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i.

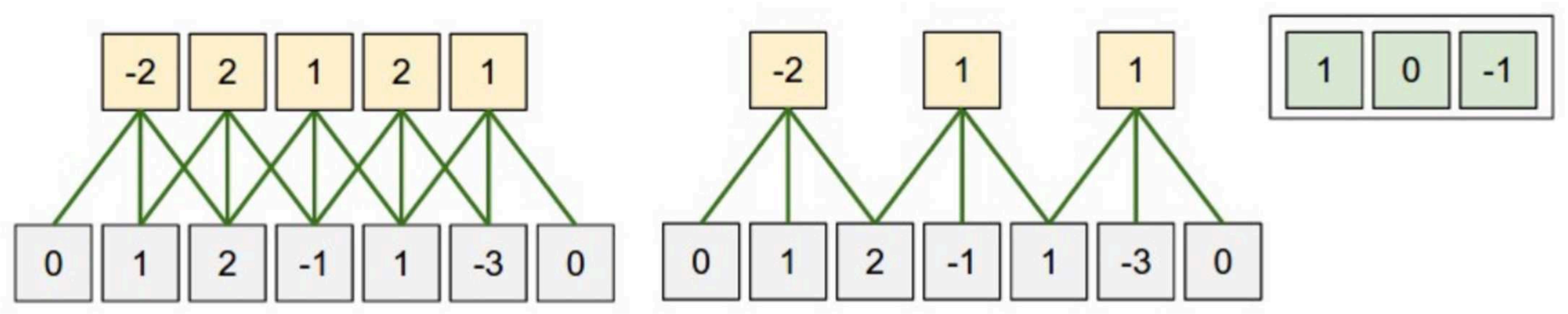
(you can also have two loops: outer loop over epochs, inner loop over mini-batches)

Convolutional Neural Network: LeNet (1998)



LeCun et al., 1998

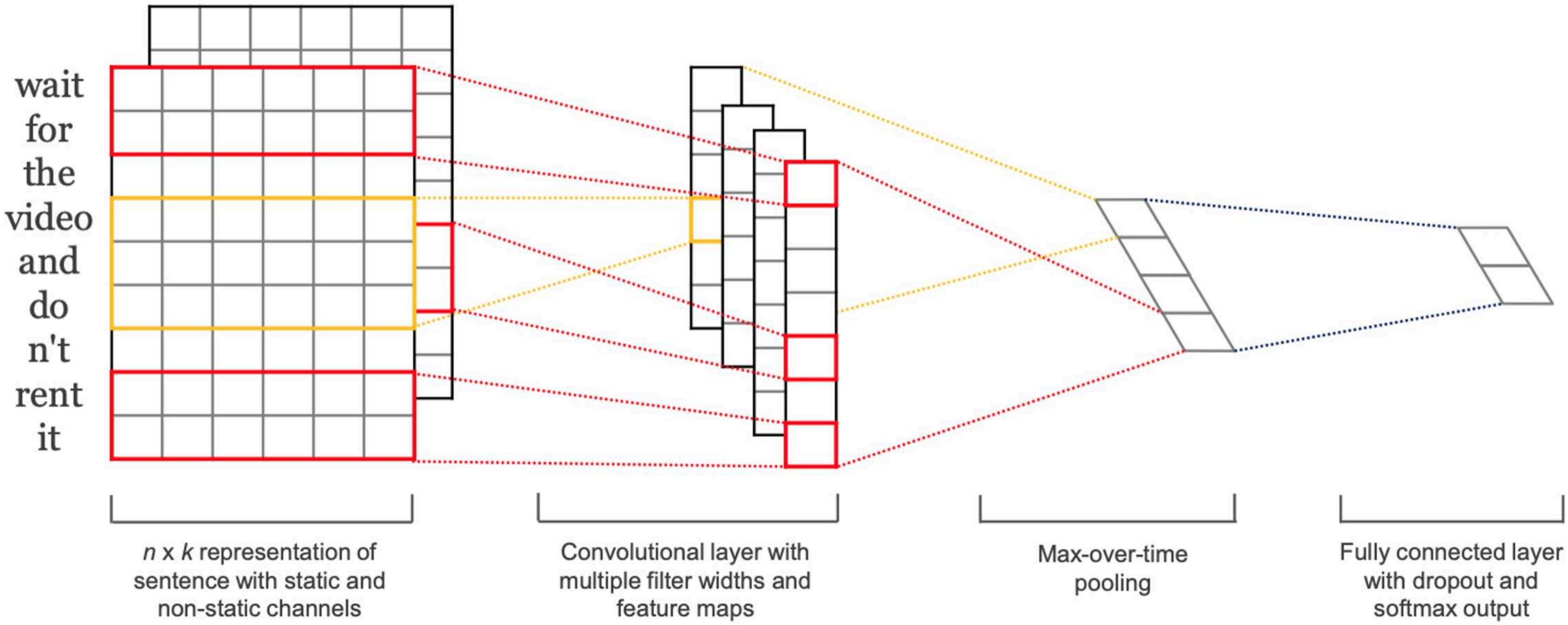
1D Convolution with W =5, F = 3, Stride = 2, Padding = 1



Output Size = $(W - F + 2P)/S + 1$

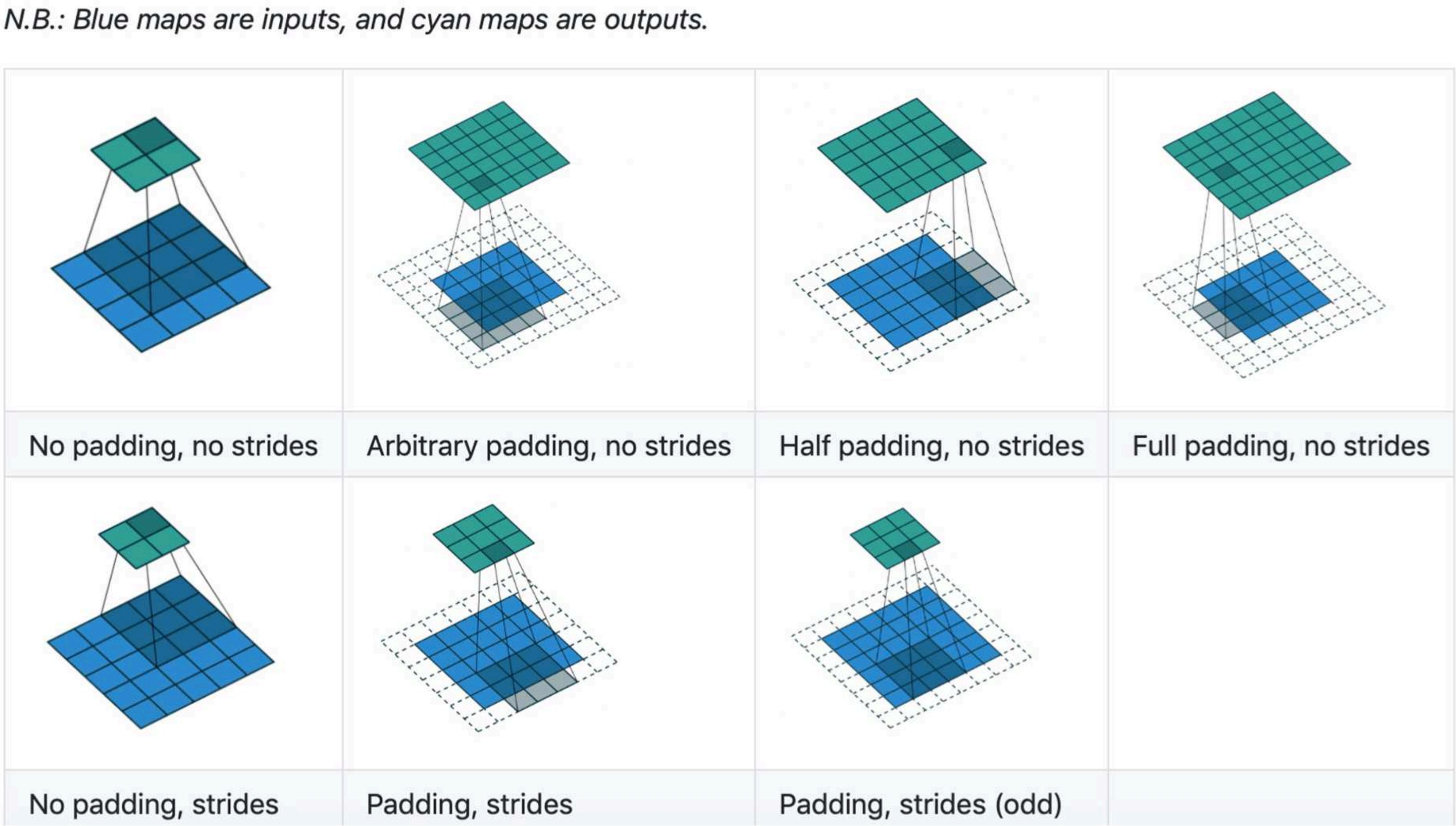
http://cs231n.github.io/convolutional-networks/

1D Convolution over Sentences


wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Yoon Kim, Convolutional Neural Networks for Sentence Classification. EMNLP 2014

2D Convolutions



N.B.: Blue maps are inputs, and cyan maps are outputs.

No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
No padding, strides	Padding, strides	Padding, strides (odd)	

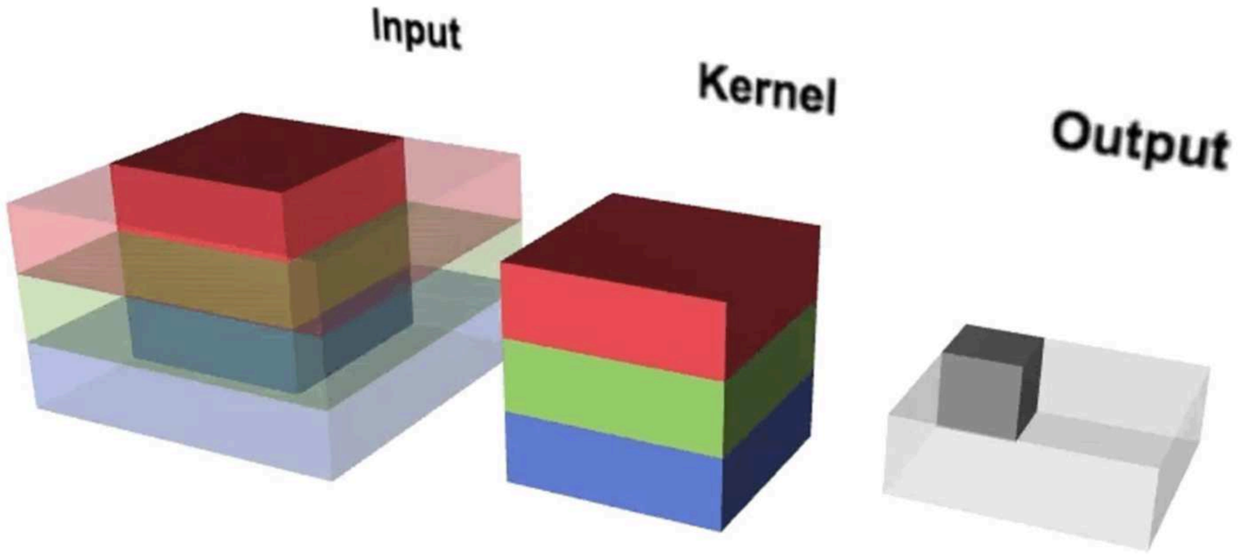
https://github.com/vdumoulin/conv_arithmetic

2D Convolution Animations

See the animation at

https://github.com/vdumoulin/conv_arithmetic

2D 3x3 Convolution Applied to RGB Input of Size 5x5

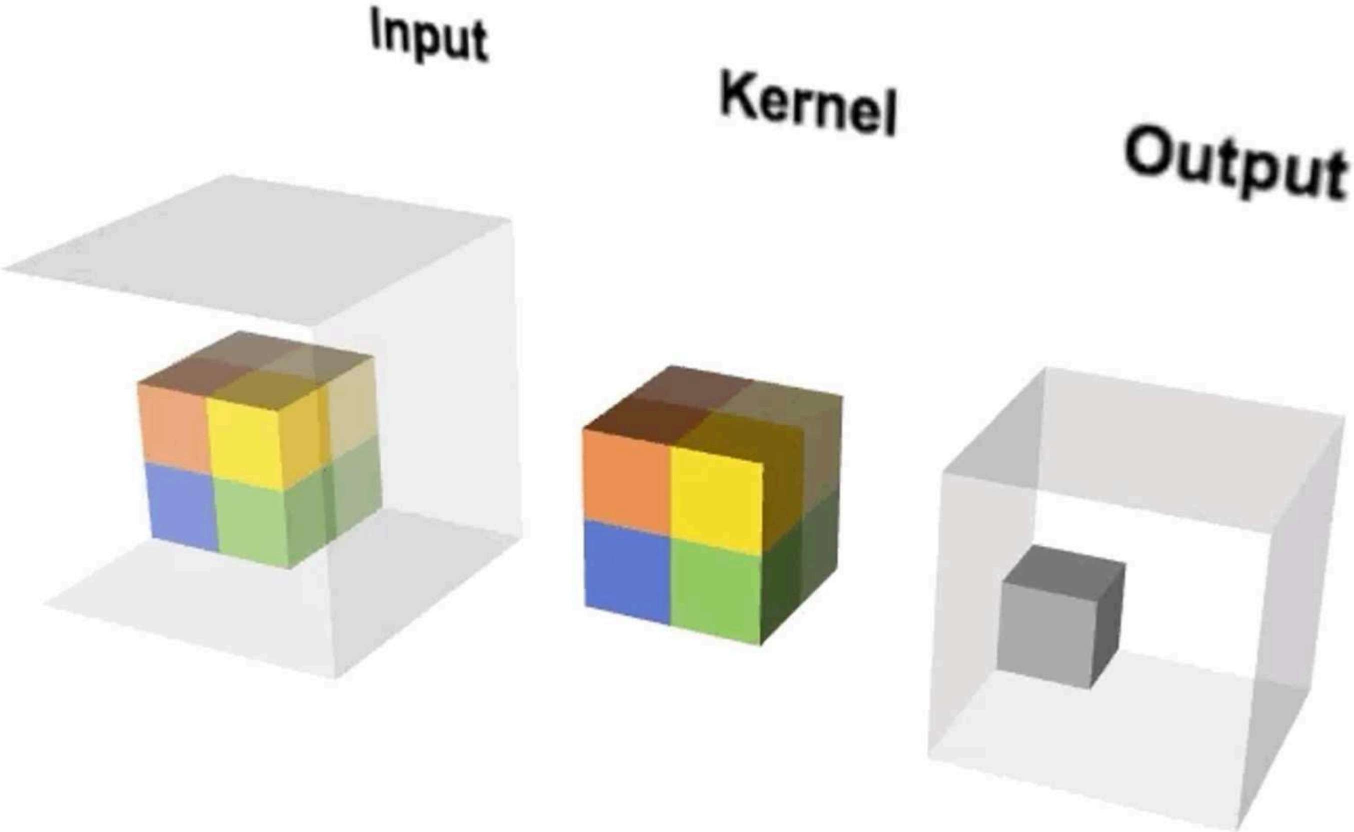


Picture credit: https://thomelane.github.io/convolutions/2DConvRGB.html

2D Convolutions in Numbers

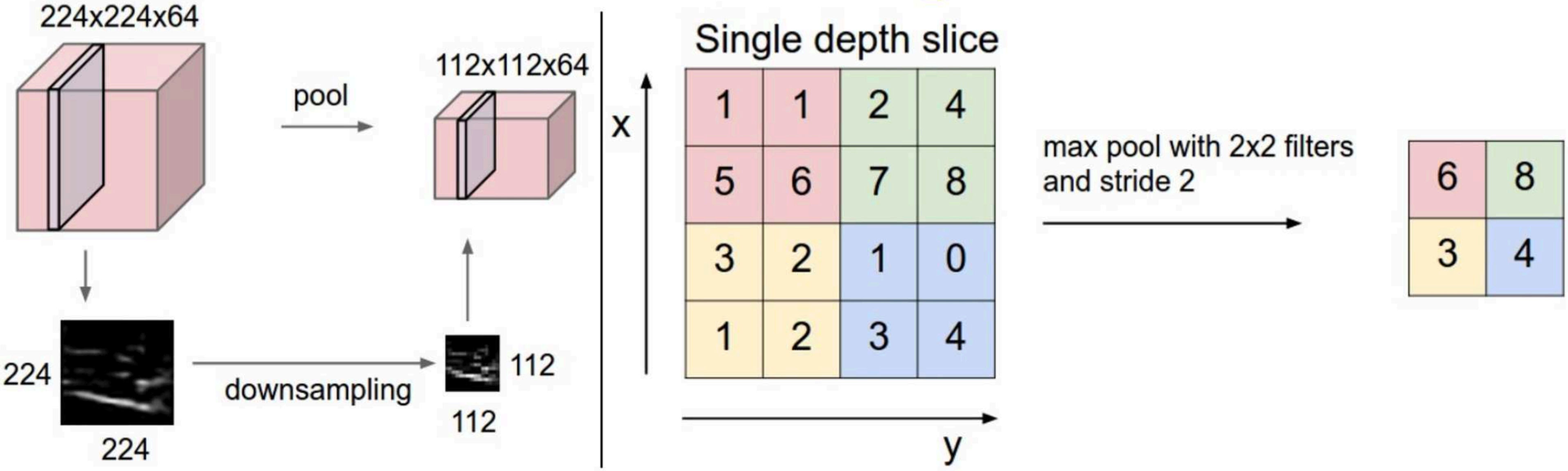
http://cs231n.github.io/convolutional-networks/

3D Convolution



Picture credit: https://thomelane.github.io/convolutions/3DConv.html

Max Pooling



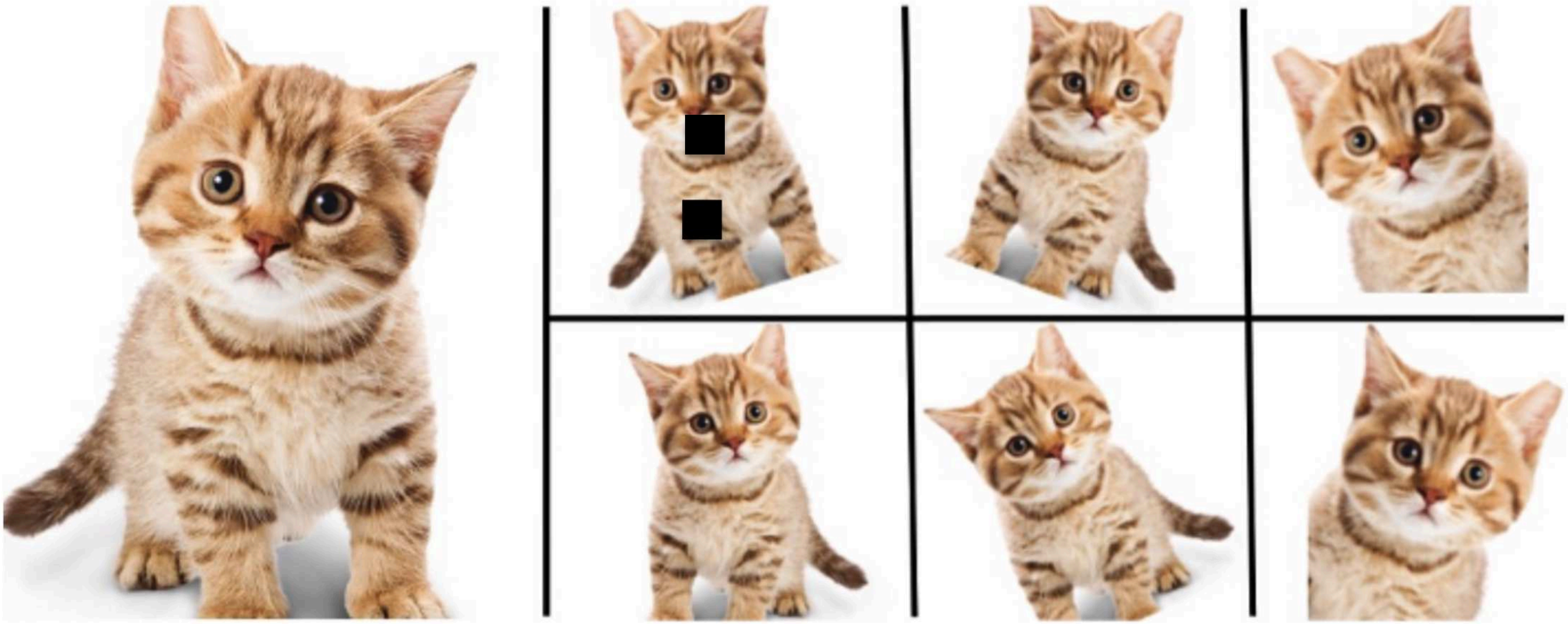
Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

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

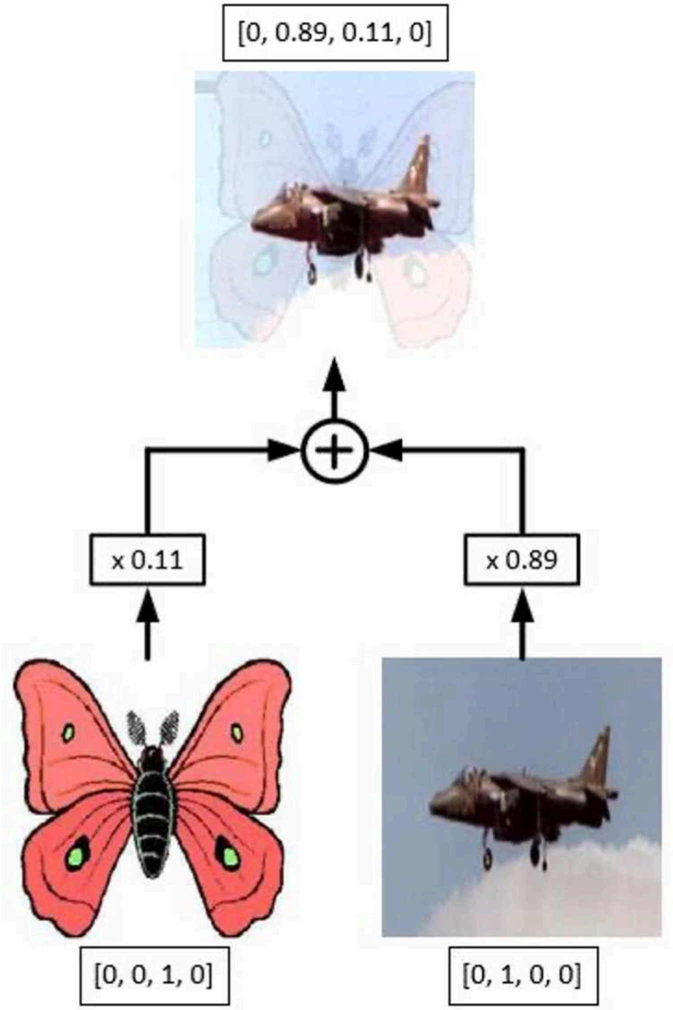
Average Pooling is also widely used, especially in NLP

Data Augmentation

Random erasing, horizontal flipping, rotation, scaling (with cropping), cropping, contrast, color



Picture credit: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/



Mixup

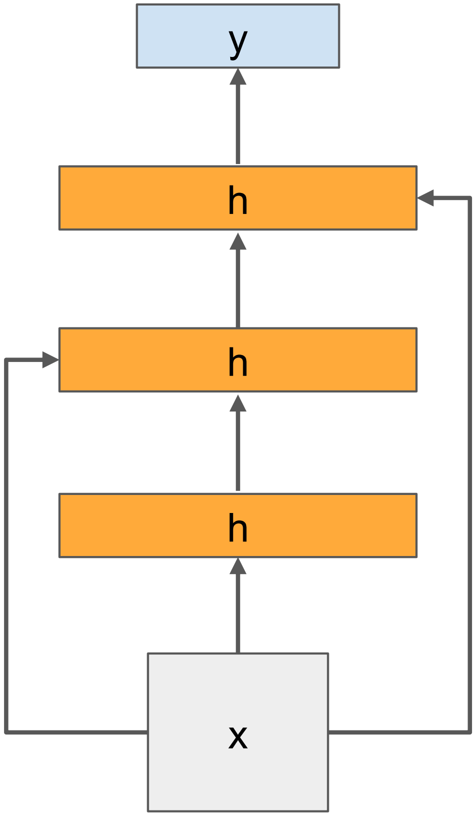
$$\tilde{x} = \lambda x_i + (1-\lambda) x_j, \qquad \text{where } x_i, x_j \text{ are raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1-\lambda) y_j, \qquad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

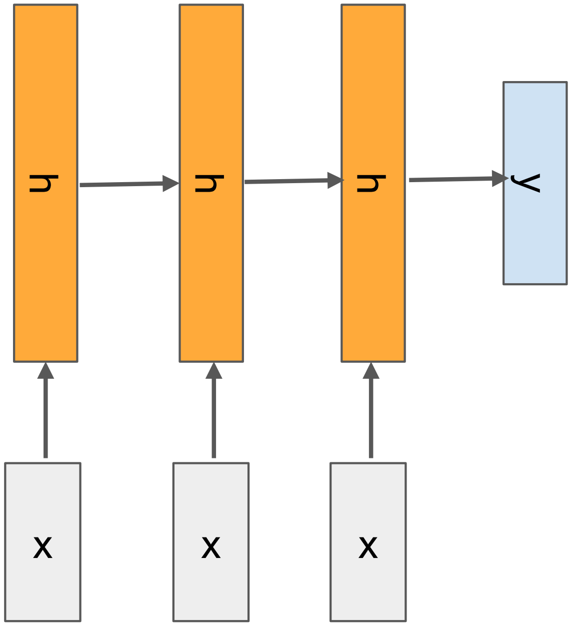
Zhang *et al*., Mixup: beyond empirical risk minimization.
ICLR 2018.

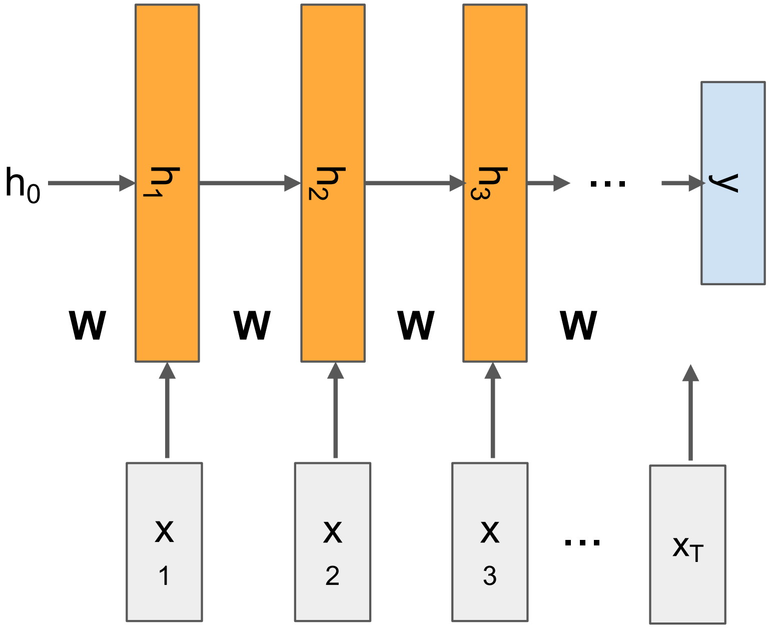
Picture credit: https://www.dlology.com/blog/how-to-do-mixup-training-from-image-files-in-keras/

From Standard Neural Networks to Recurrent Neural Networks

Let’s make the model easily extendable to model sequences with arbitrary lengths by weight sharing

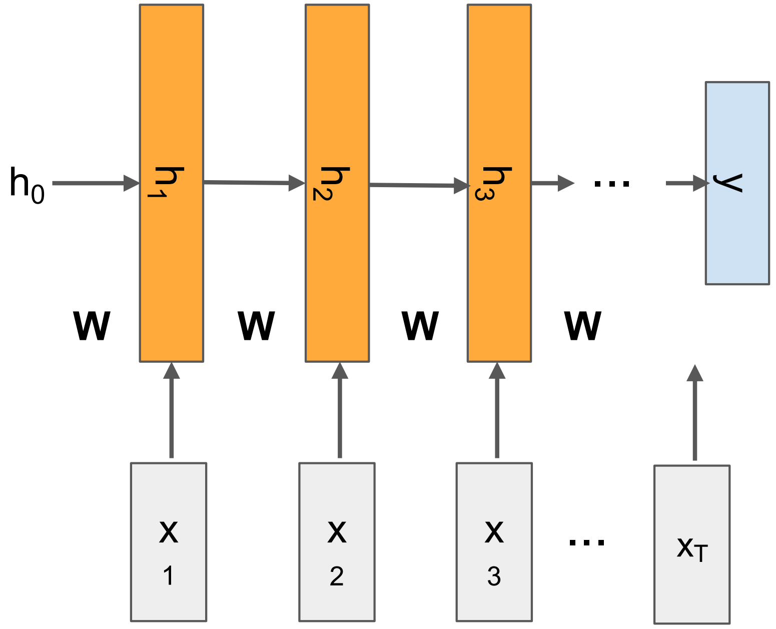






Recurrent Neural Networks (RNN)

At time step t, the hidden units accumulate past information about the input sequence. Hidden activity vector h_t only depend on current input x_t and previous hidden activity vector h_{t-1}



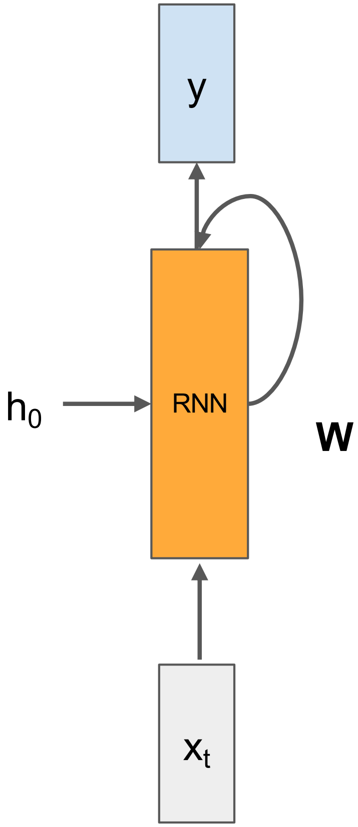
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

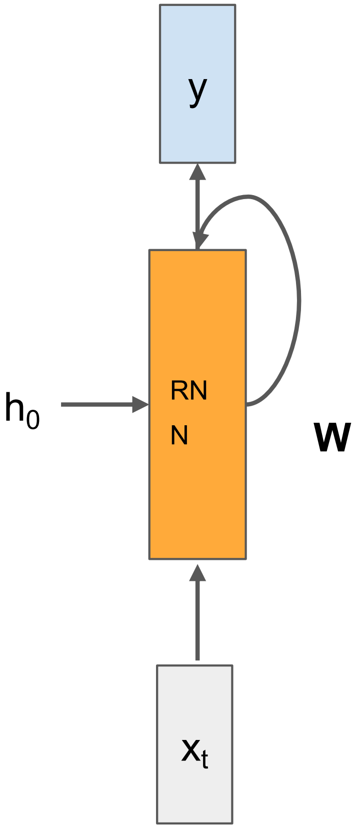
some function with parameters W

old state

input vector at some time step



Vanilla Recurrent Neural Networks



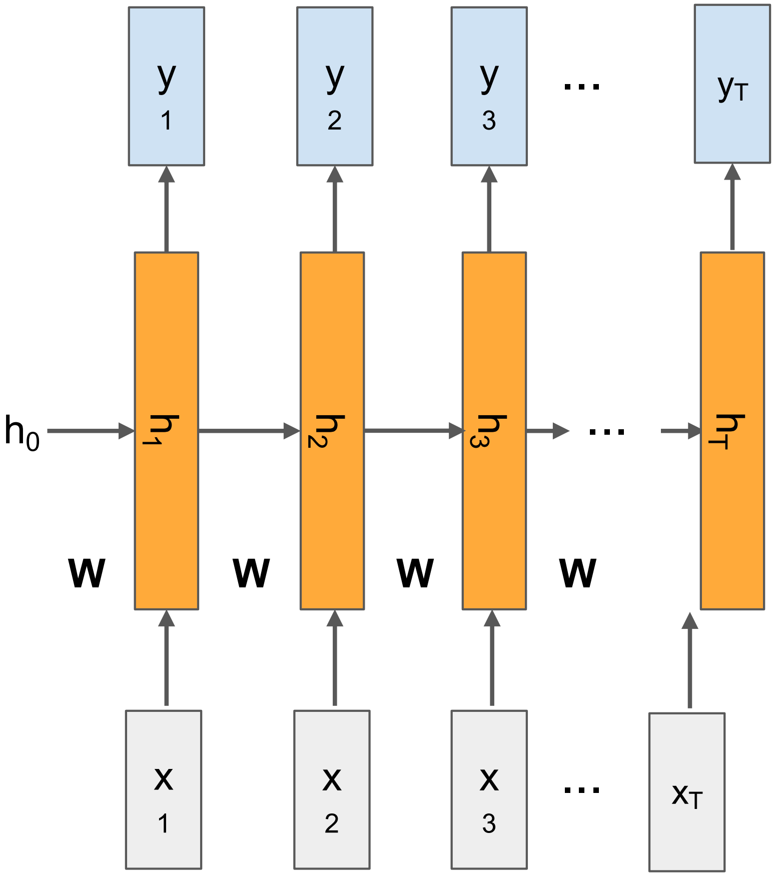
$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

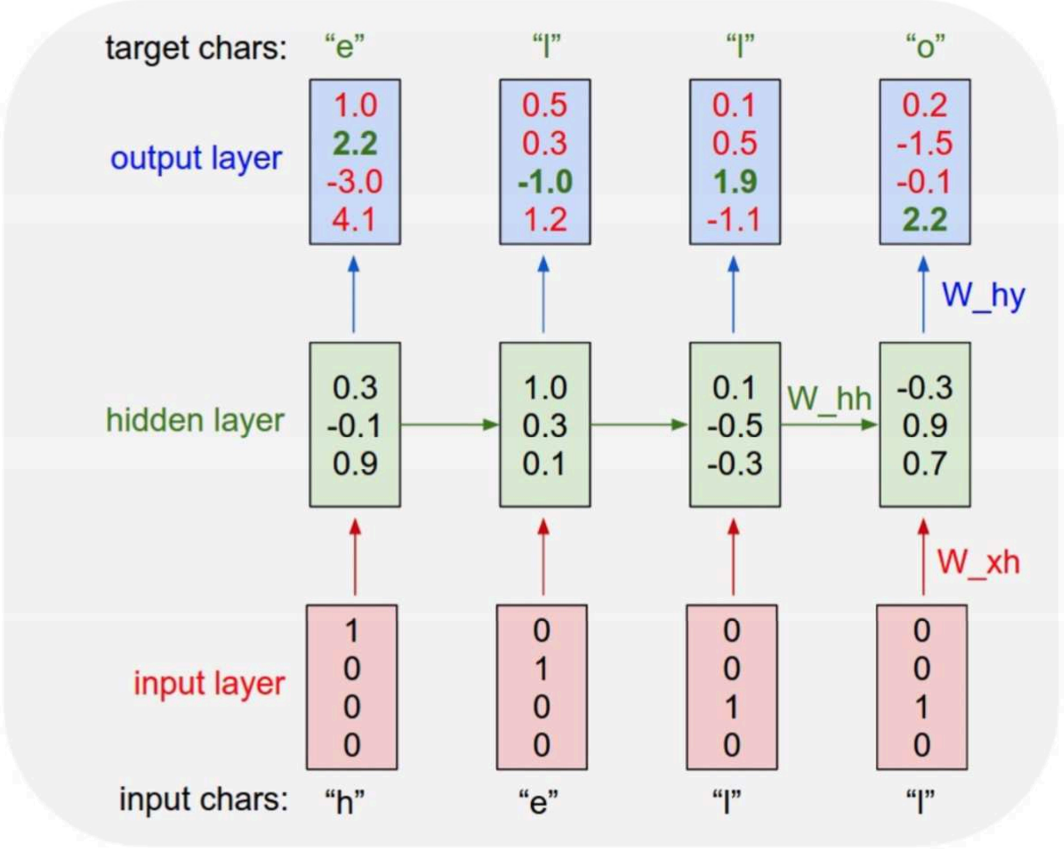
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Many-to-Many Vanilla RNN



Training of Char-RNN (Teacher Forcing)



An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low.

$$\Pr(\mathbf{x}) = \prod_{t=1}^{T} \Pr(x_{t+1}|y_t)$$

$$\hat{y}_t = b_y + \sum_{n=1}^{N} W_{h^n y} h_t^n$$

$$y_t = \mathcal{Y}(\hat{y}_t)$$

$$\Pr(x_{t+1} = k|y_t) = y_t^k = \frac{\exp\left(\hat{y}_t^k\right)}{\sum_{k'=1}^{K} \exp\left(\hat{y}_t^{k'}\right)}$$

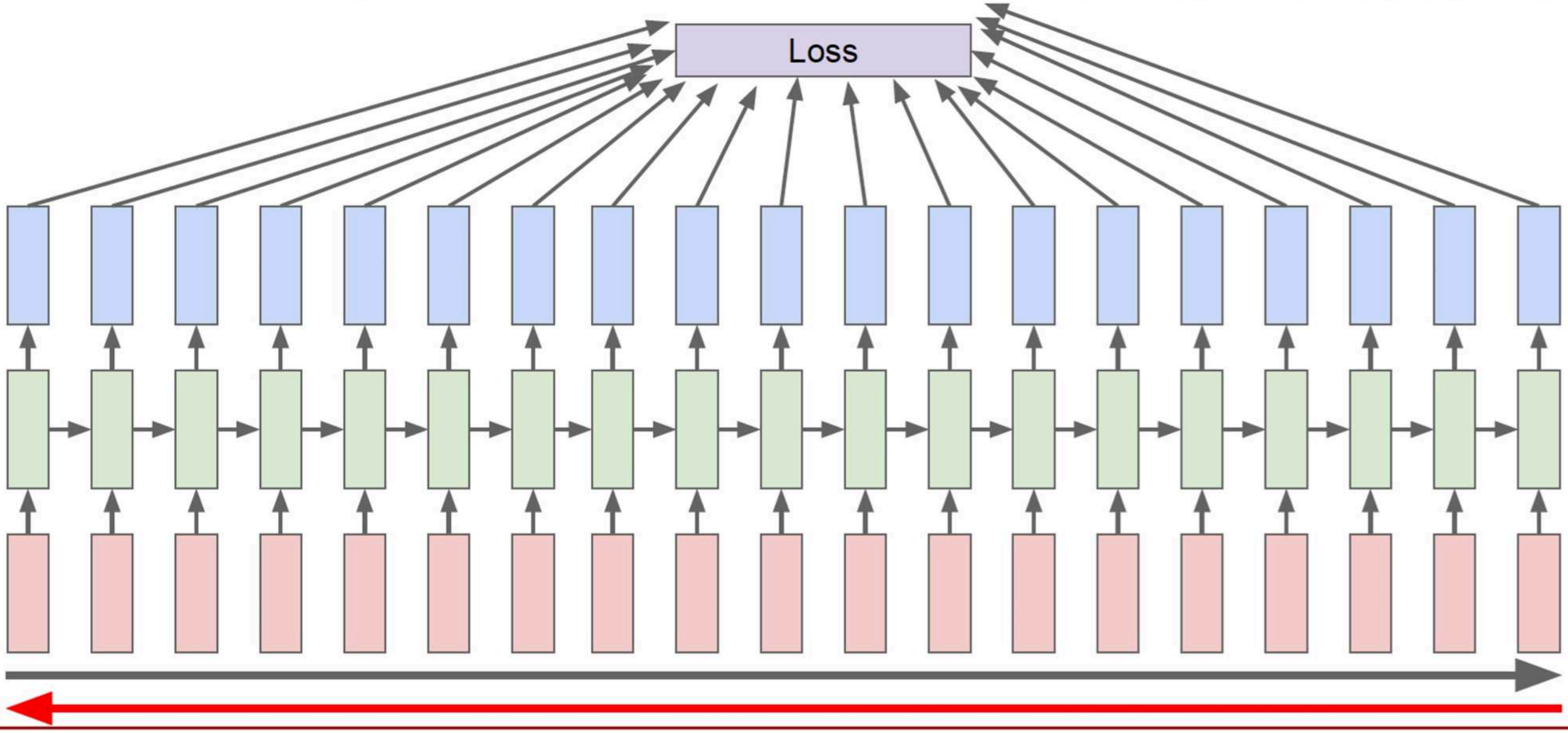
$$\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} \log y_t^{x_{t+1}}$$

$$\implies \frac{\partial \mathcal{L}(\mathbf{x})}{\partial \hat{y}_t^k} = y_t^k - \delta_{k,x_{t+1}}$$

Picture Credit: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

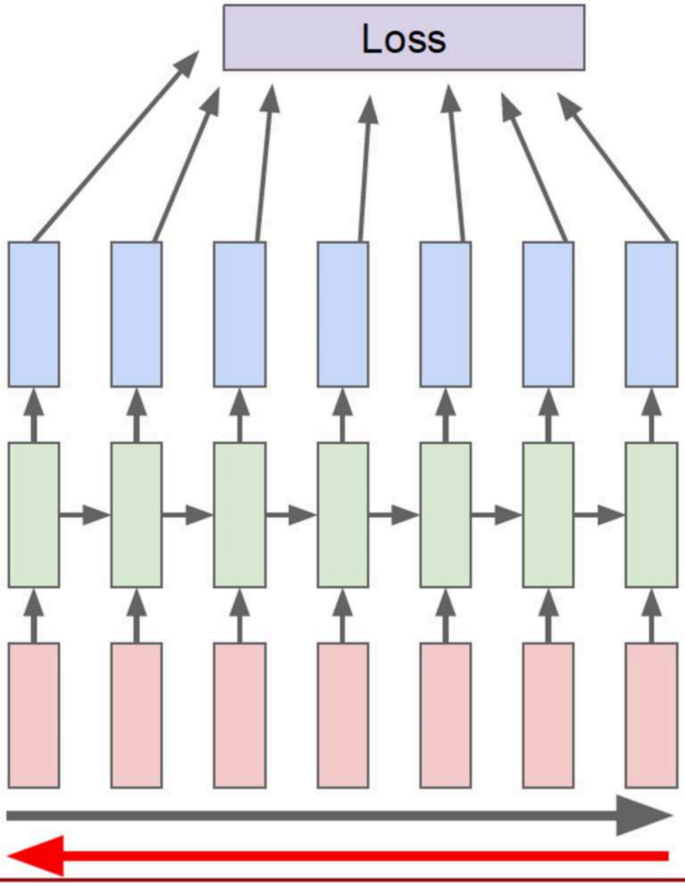
Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

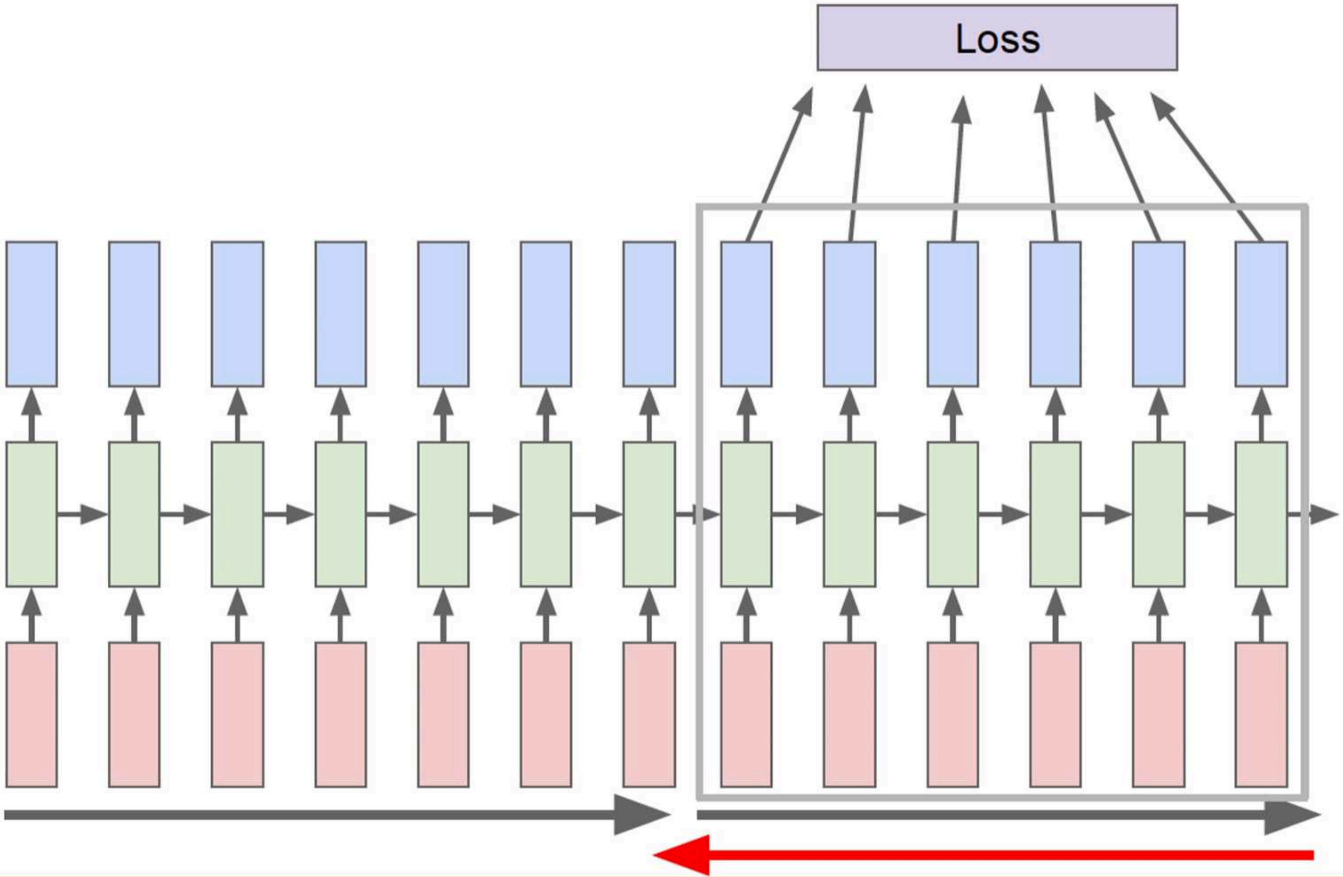
Truncated Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

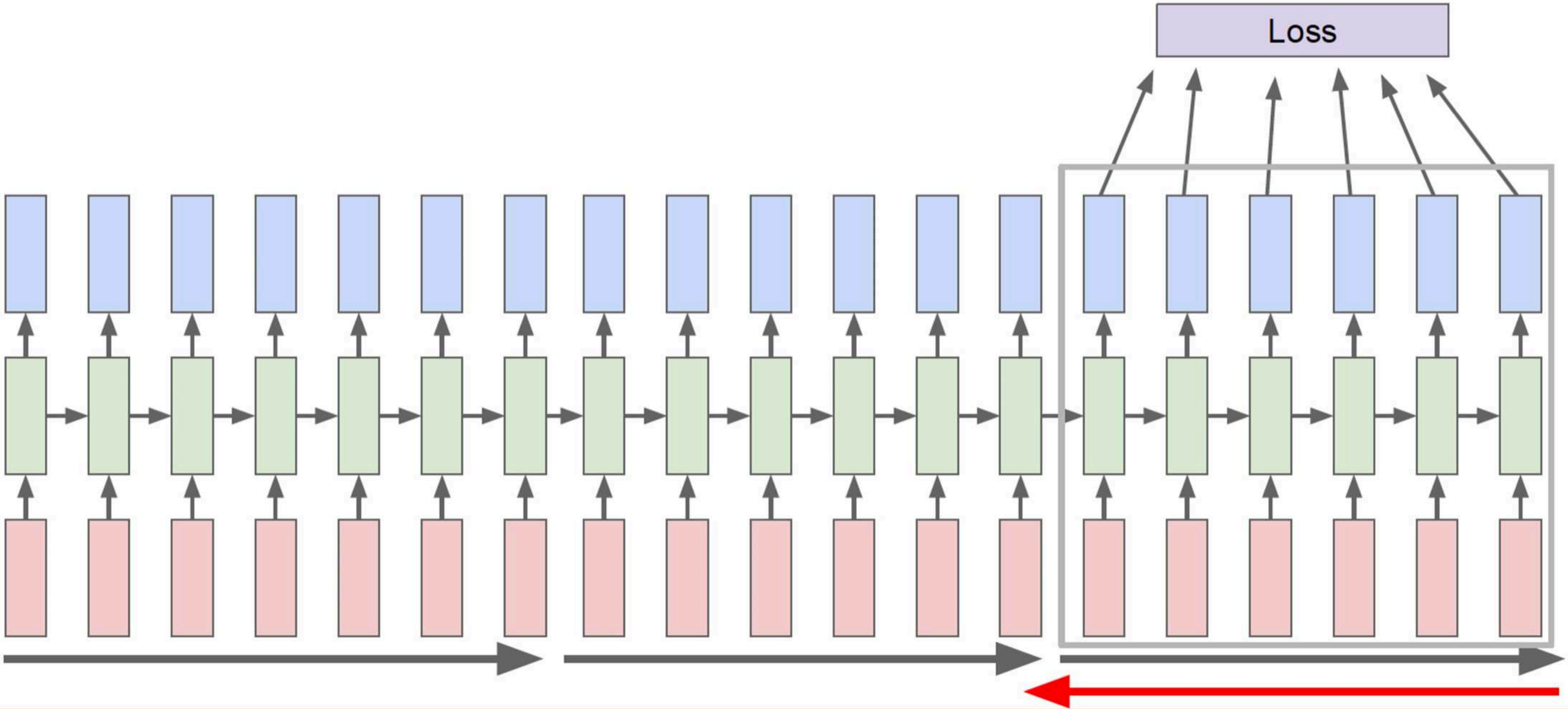
Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

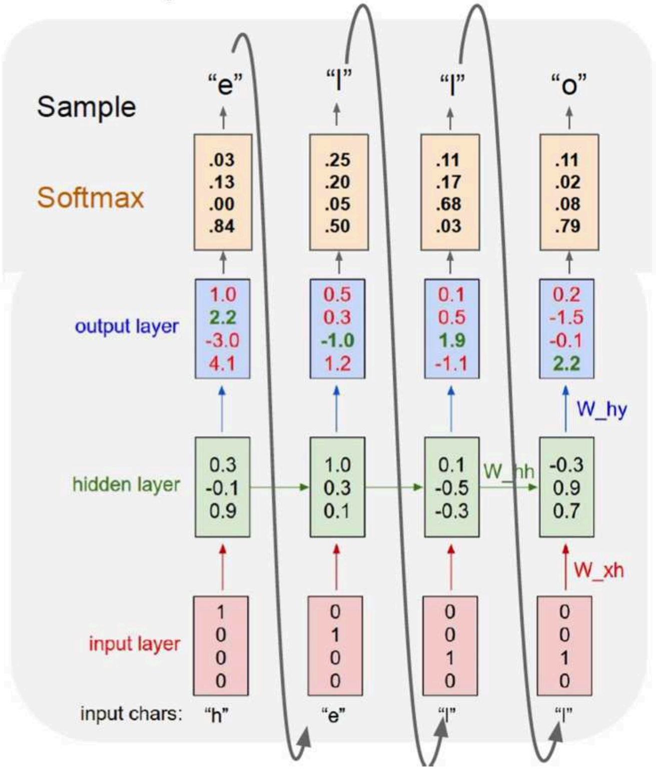
Truncated Backpropagation through time



Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

Inference of Char-RNN (Exposure Bias)

At test time, sample a character from the current model at each step, feed the current sampled character as input to the next time step



Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

Why Vanishing and Exploding Gradient of Vanilla RNN Happens

$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Suppose we are using a many-to-many RNN for sequence labeling

$$\mathcal{E}_t = \mathcal{L}(\mathbf{h}_t) \quad \mathcal{E} = \textstyle\sum_t \mathcal{E}_t$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \textstyle\sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k^+}{\partial \theta} \right)$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \Pi_{t \ge i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \Pi_{t \ge i > k} W_{hh}^T diag(tanh'(W_{hh}h_{i-1} + W_{xh}x_i))$$

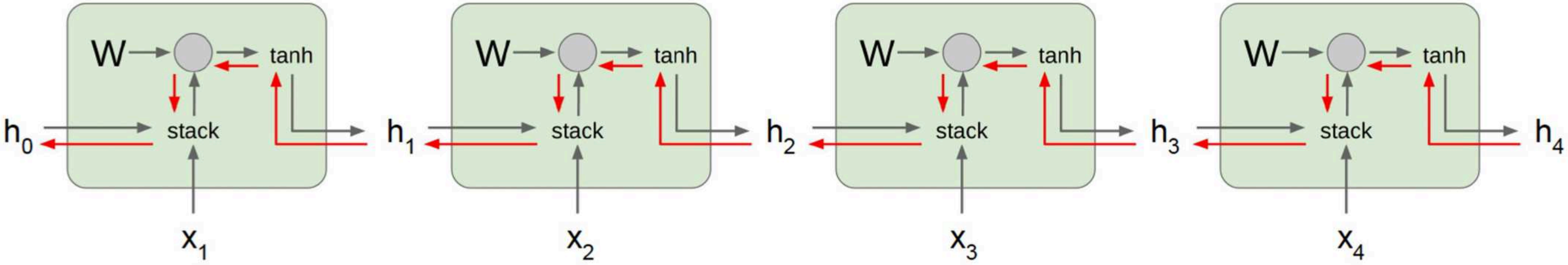
$$\frac{\partial \mathbf{h}_k^+}{\partial \theta}$$

is the immediate partial derivative of hidden activity vector with respect to network weights

Pascanu et al., On the difficulty of training recurrent neural networks. ICML 2013

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Gradient norm Clipping

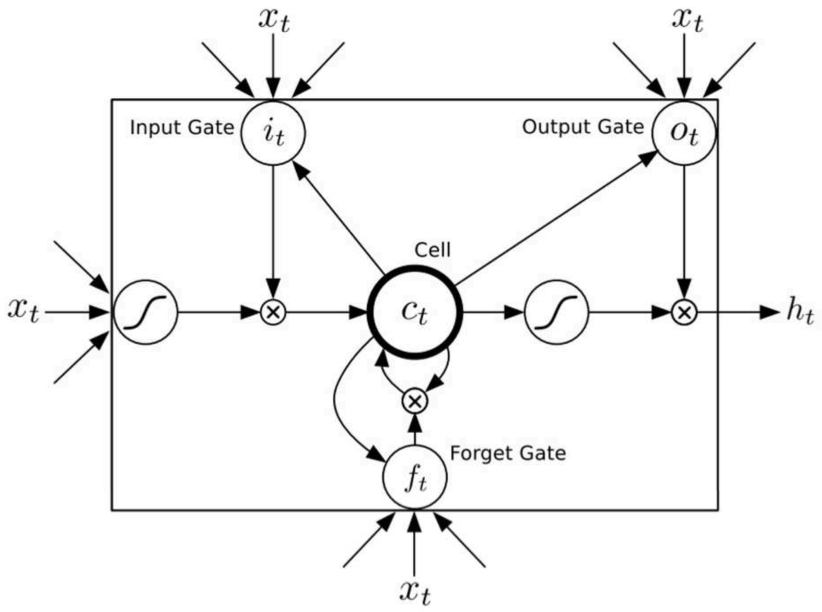
Algorithm 1 Pseudo-code for norm clipping

```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if
```

Design a better architecture

Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

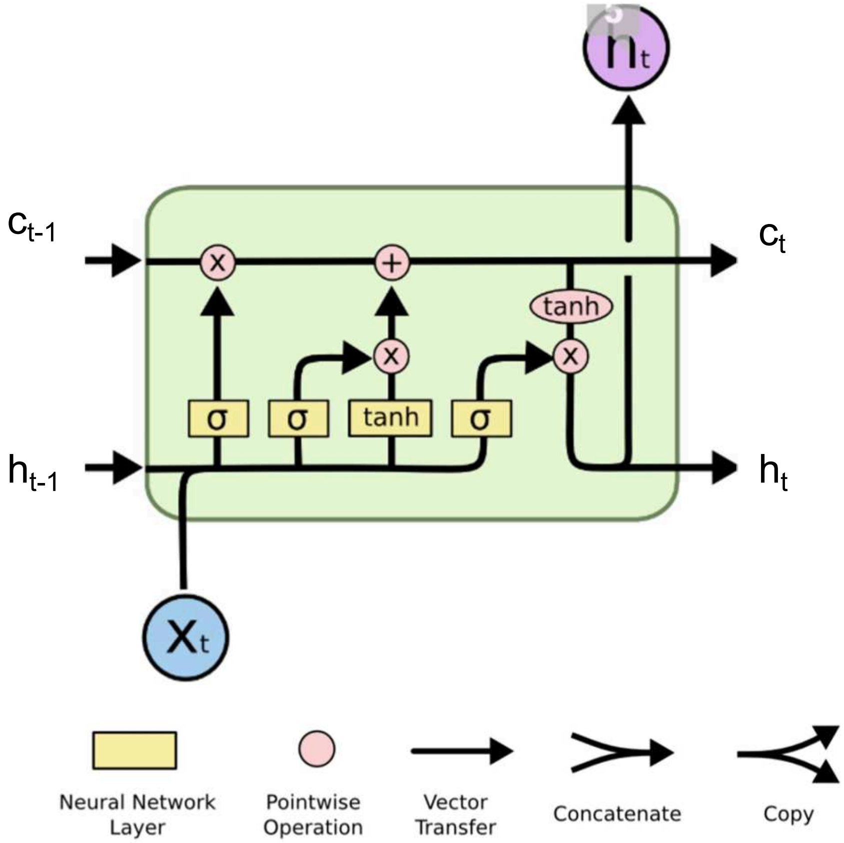
Long Short-Term Memory



$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$

Picture Credit: https://www.cs.toronto.edu/~graves/asru_2013.pdf

Long Short-Term Memory



$\mathcal{H}$

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$

$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$

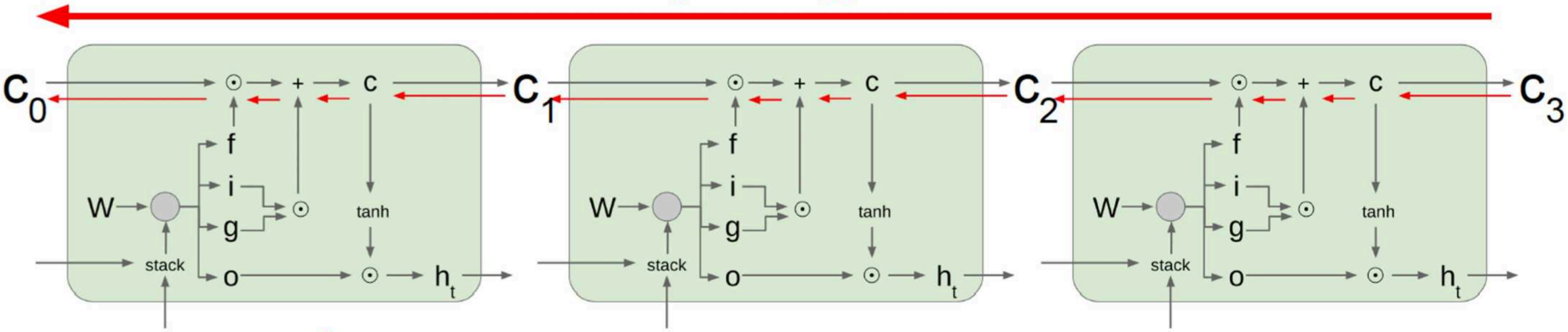
$$h_t = o_t \tanh(c_t)$$

Picture Credit: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

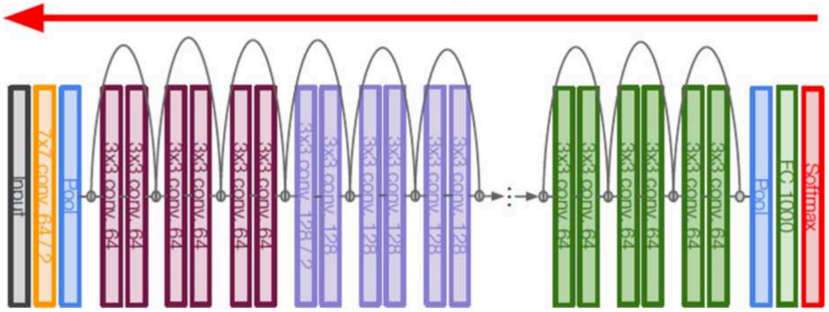
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

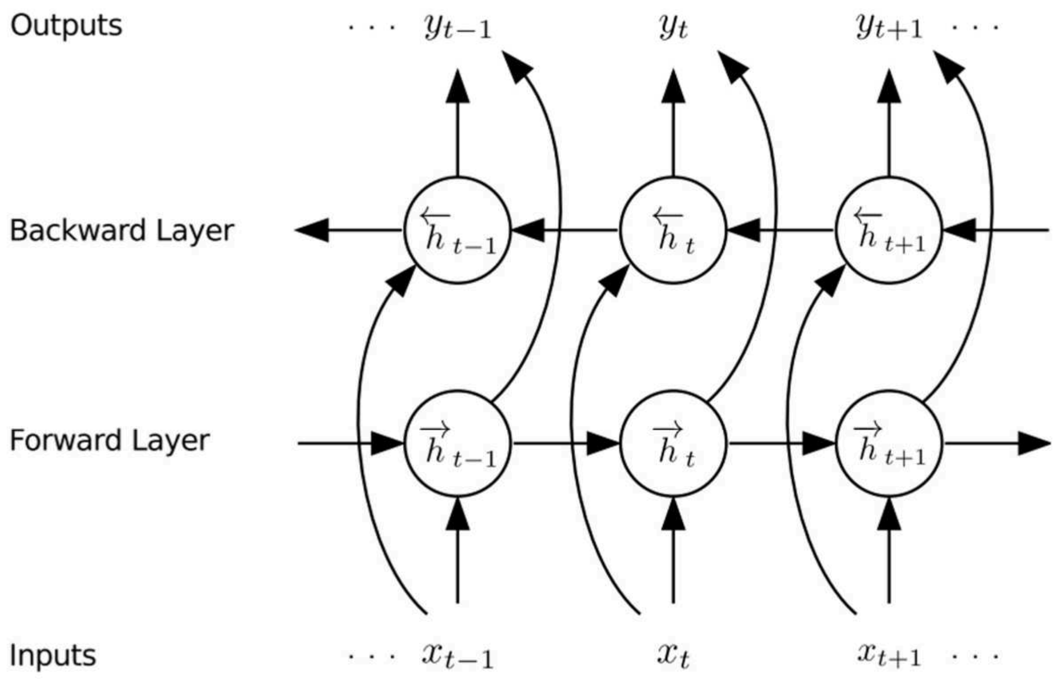
$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, “Highway Networks”,
ICML DL Workshop 2015

Slide Credit: Fei-Fei, Johnson, and Yeung, Stanford cs231n, 2019

Bidirectional LSTM



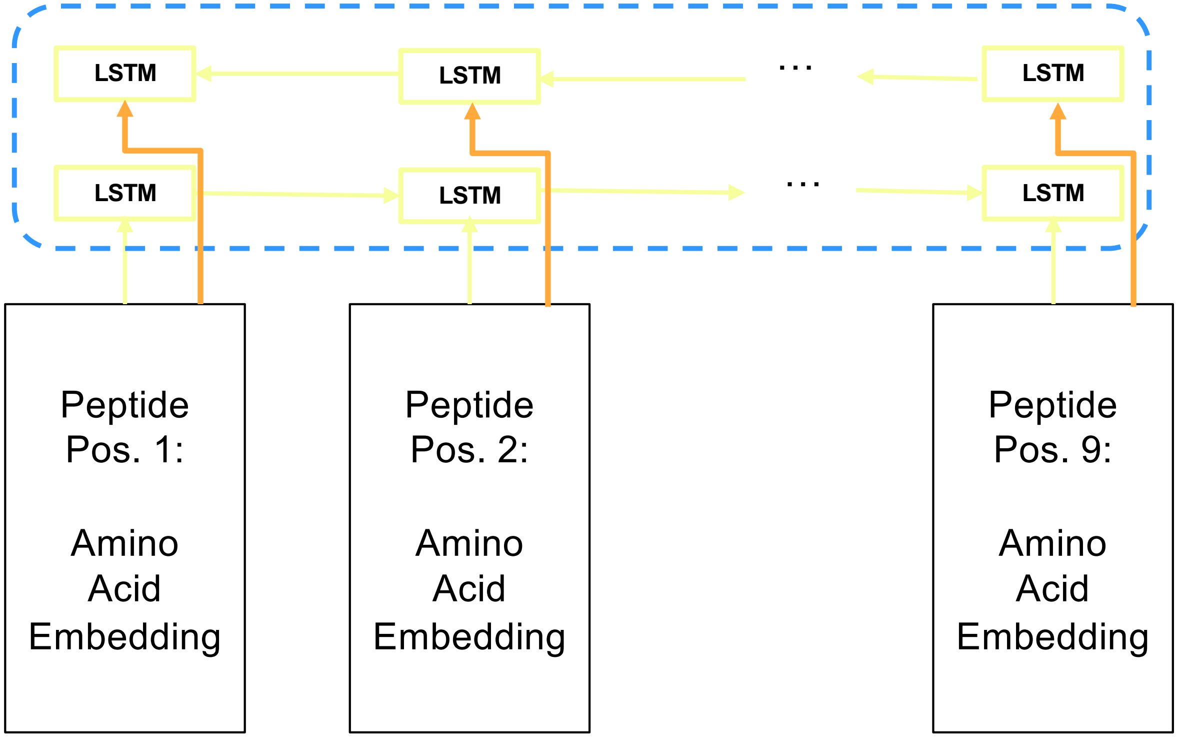
$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$

Picture Credit: https://www.cs.toronto.edu/~graves/asru_2013.pdf

Bidirectional LSTM



LSTM

LSTM

…

LSTM

LSTM

LSTM

…

LSTM

Peptide
Pos. 1:

Amino
Acid
Embedding

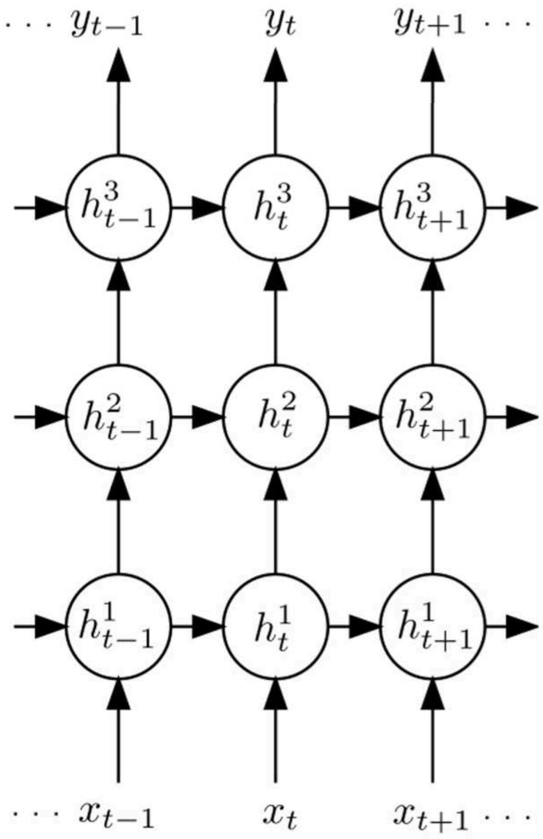
Peptide
Pos. 2:

Amino
Acid
Embedding

Peptide
Pos. 9:

Amino
Acid
Embedding

Deep LSTM



$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right.$$

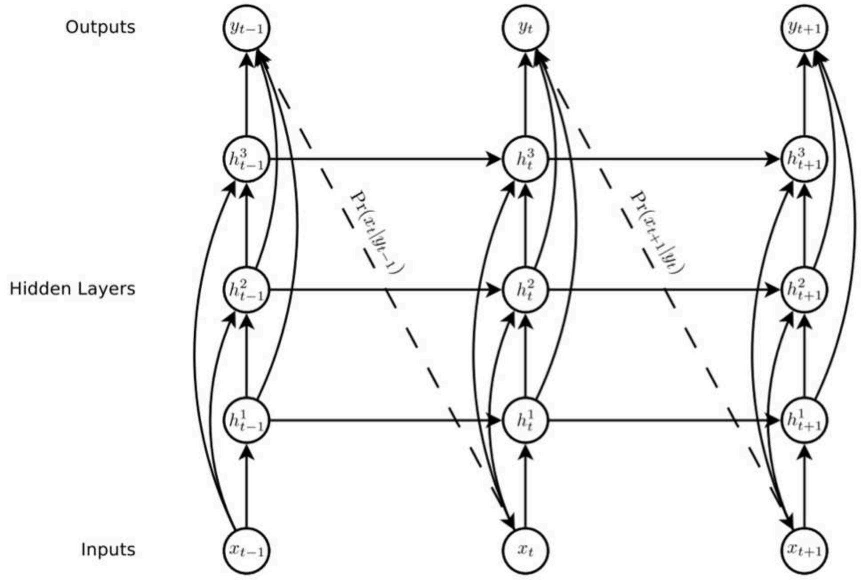
$$y_t = W_{h^N y} h_t^N + b_y$$

Picture Credit: https://www.cs.toronto.edu/~graves/asru_2013.pdf

Deep LSTM for Generating Complex Sequences

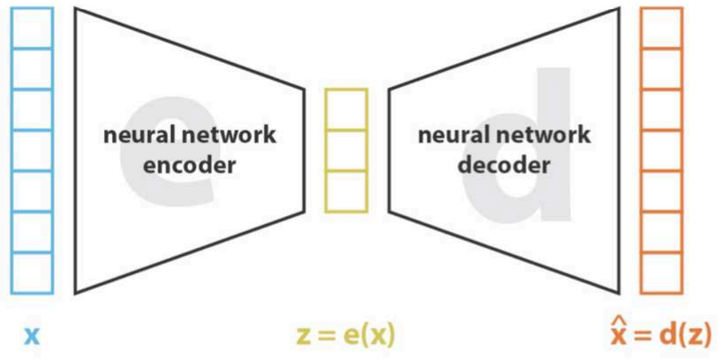
Generating text with characters or words as symbols

Generating handwriting with sequences of pen coordinates (x, y) and pen on/off whiteboard as input



Alex Graves, Generating Sequences With Recurrent Neural Networks. 2015
https://arxiv.org/pdf/1308.0850.pdf

Deep Encoder-Decoder Networks: Sequence-to-Sequence (Seq2Seq) Models



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

Illustration of an autoencoder with its loss function.

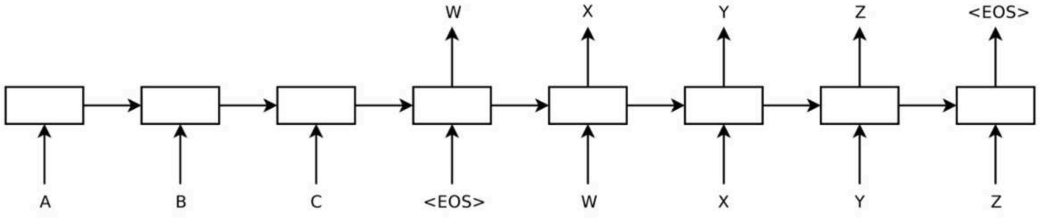


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Data Augmentation in Sequence-to-Sequence (Seq2Seq) Models for Machine Translation

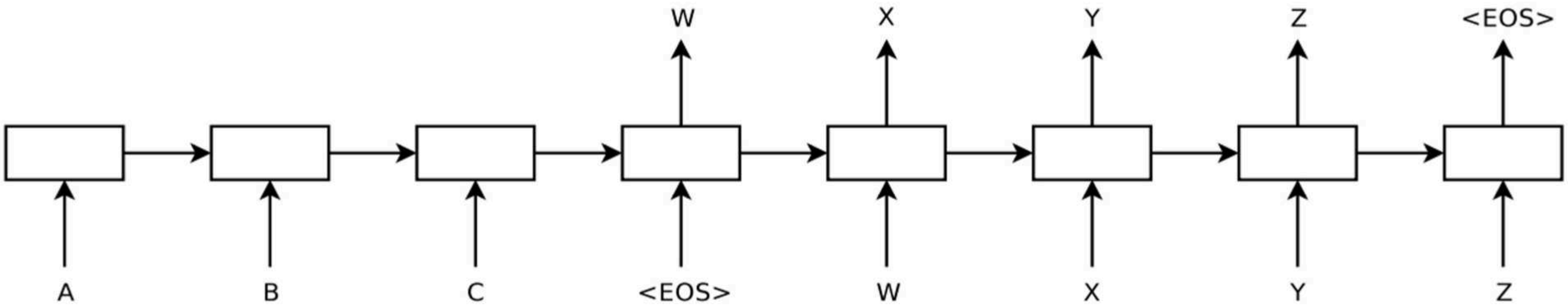


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

$$p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

representation v of the input sequence $(x_1, \ldots, x_T)$

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)\\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)\\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)\\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)\\
h_t &= o_t \tanh(c_t)
\end{aligned}$$

Learning Residual Feature Maps like in LSTM

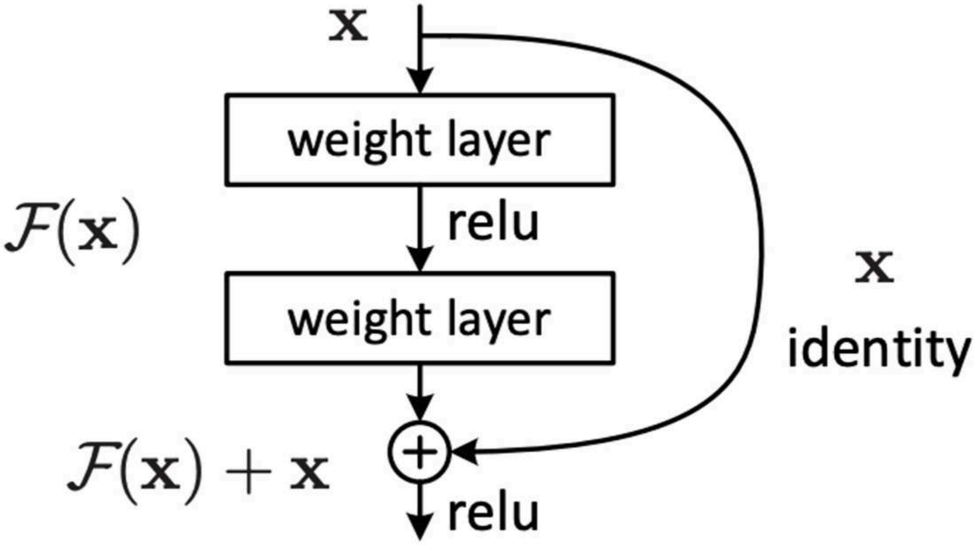
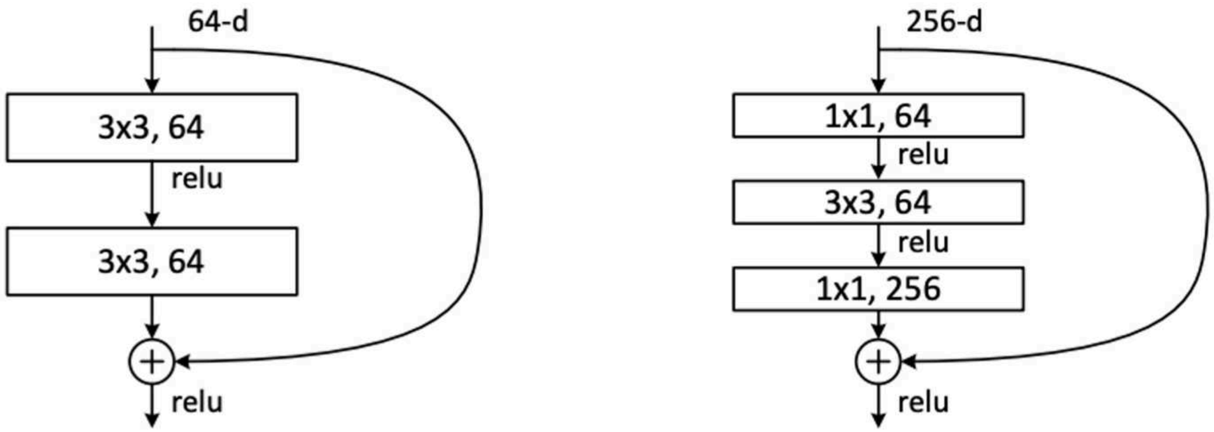


Figure 2. Residual learning: a building block.

He et al., Deep Residual Learning for Image Recognition. CVPR 2015

Learning Residual is Easier



He et al., Deep Residual Learning for Image Recognition. CVPR 2015

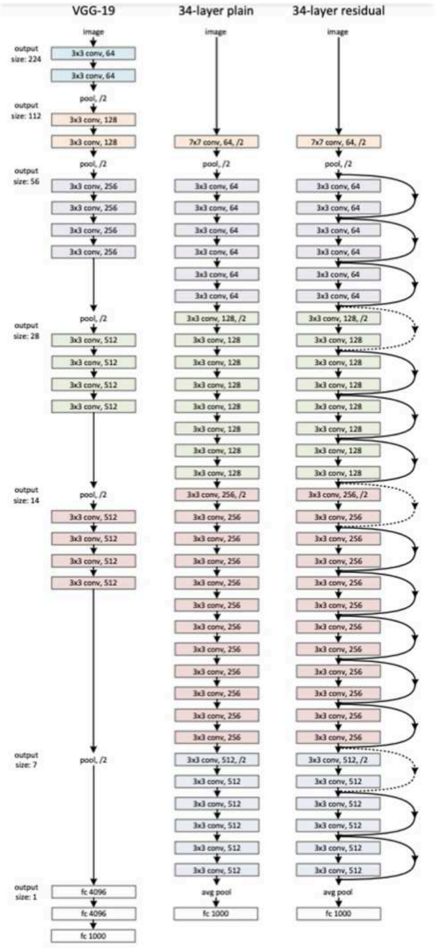
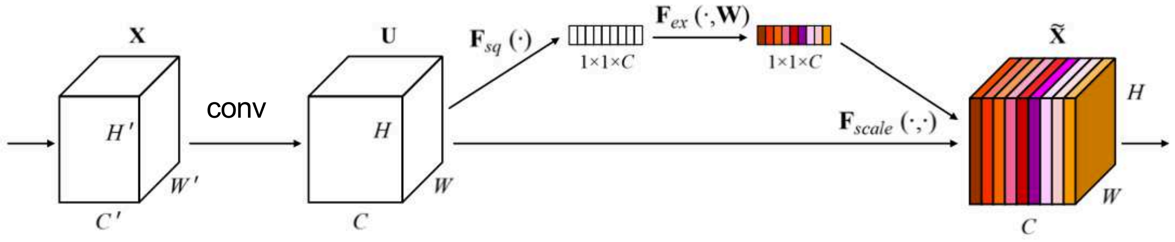


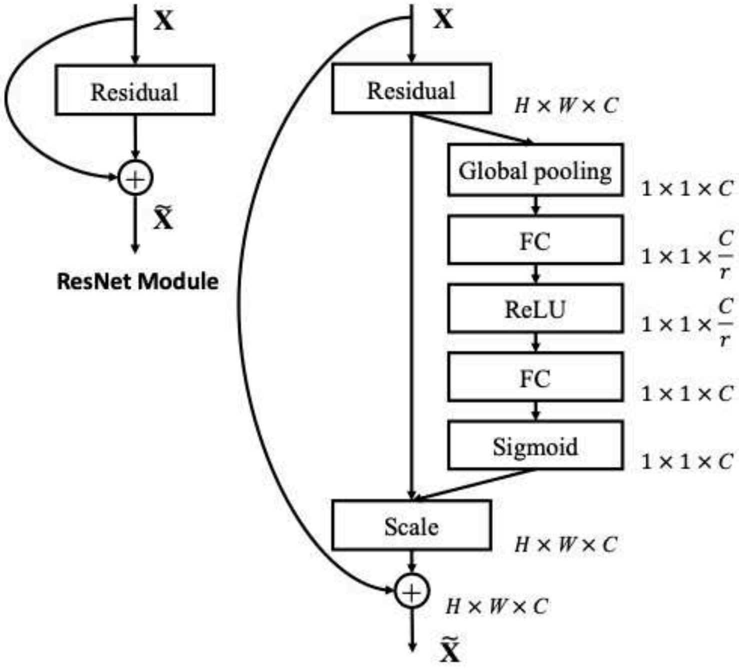
Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

VGG
vs.
ResNet

He et al., CVPR 2015

Squeeze-and-Excitation Blocks





$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s.$$

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W}\sum_{i=1}^{H}\sum_{j=1}^{W} u_c(i,j).$$

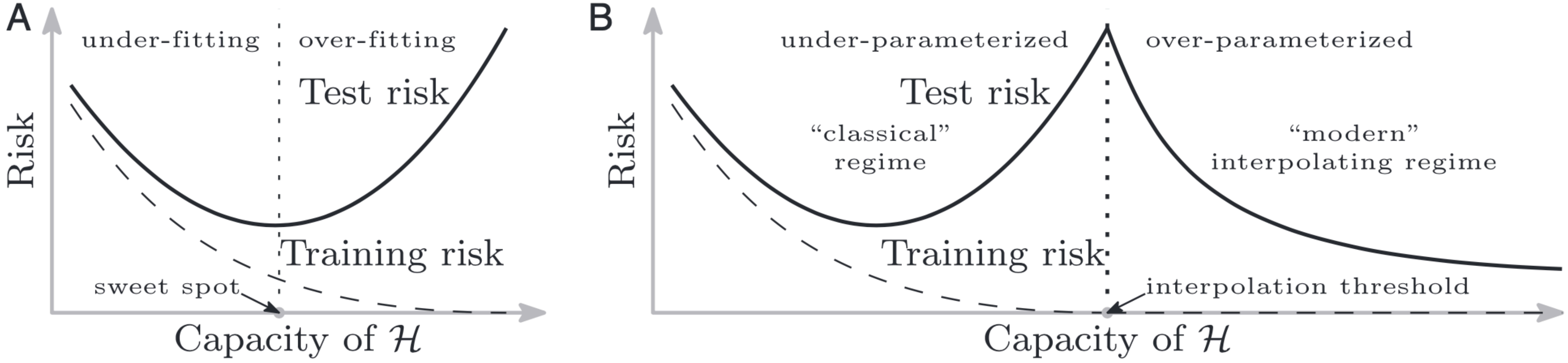
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z},\mathbf{W}) = \sigma(g(\mathbf{z},\mathbf{W})) = \sigma(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z}))$$

$$\widetilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c\,\mathbf{u}_c,$$

$$\widetilde{\mathbf{X}} = [\widetilde{\mathbf{x}}_1, \widetilde{\mathbf{x}}_2, \dots, \widetilde{\mathbf{x}}_C]$$

Hu *et al*., CVPR 2018.

Double Descent Phenomena



Curves for training risk (dashed line) and test risk (solid line). (*A*) The classical U-shaped risk curve arising from the bias–variance trade-off. (*B*) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high-capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning and the bias-variance tradeoff. PNAS. 2019.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, Ilya Sutskever. Deep Double Descent: Where Bigger Models and More Data Hurt. *ICLR 2020.*

Final Remarks: What makes deep learning revive?

- Deep learning is a modern name for deep neural networks
- Why deep neural networks revive (AlexNet, ResNet, Transformer, GPT-3, ChatGPT, GPT-4, ...)
    - Good model: has the intrinsic advantage of learning hierarchical distributed feature representations (CNN, RNN, Transformer)
    - Powerful computers and big data enable us to build models with huge capacity
    - Data augmentation (using heuristics to create more data to fill the input data distribution space)
    - Parameter learning without vanishing gradient or saturated activation functions (rectified linear hidden units) and regularizations (dropout and batch normalization)
    - Good open-source tools: cuda-convnet2, Torch, Caffe, TensorFlow, and PyTorch

Summary

- The ongoing AI revolution is driven by deep learning. Hidden units, backpropagation, convolutions, and attention mechanisms are great inventions in neural networks.

- A deep model + big data + powerful computers -> Success

- A quote about machine learning by Prof. Pedro Domingos from University of Washington, “Learning is more like farming, which lets nature do most of the work. Farmers combine seeds with nutrients to grow crops. Learners combine knowledge with data to grow programs.” This is also true for deep learning.

- Deep learners should combine their knowledge with large-scale data to grow programs, encode essential knowledge into network structures, and let backpropagation and stochastic gradient descent do the heavy lifting.

Thank You

Next lecture:

Deep Learning: Part II

Deep Generative Models - VAE, GAN, LLMs

Mar 29, 2023