Landmark Papers of the 2010s

Brief Primer to Modern Deep Learning

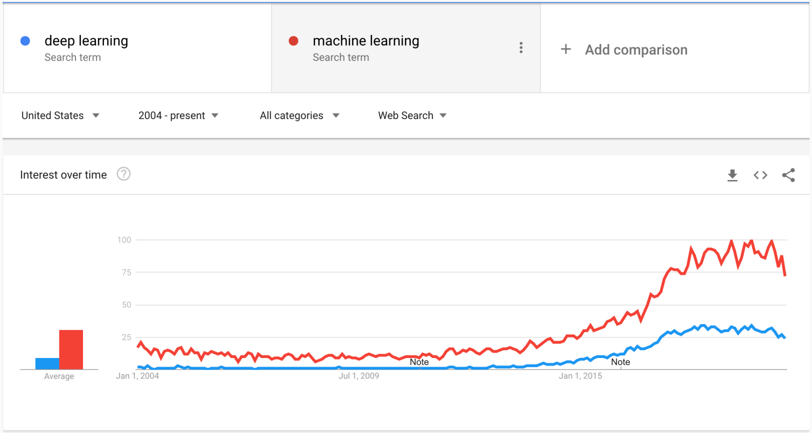
Ben Zhang

February 5th, 2020

UWDSC Reading Group

Introduction

A trend



● deep learning
Search term

● machine learning
Search term

Add comparison

United States | 2004 - present | All categories | Web Search

Interest over time

A timeline of DL innovation

- 1812: Bayes’ Theorem
- 1913: Markov Chains
- 1957: Perceptrons (early 1 layer neural network)
- 1982: Recurrent Neural Networks (Hopfield)
- 1986: Backpropogation
- 1989: Reinforcement Learning
- 1995: Convolutional Neural Networks
- 1997: LSTMs
- 2004: Deep Belief Networks
- 2006: Autoencoders

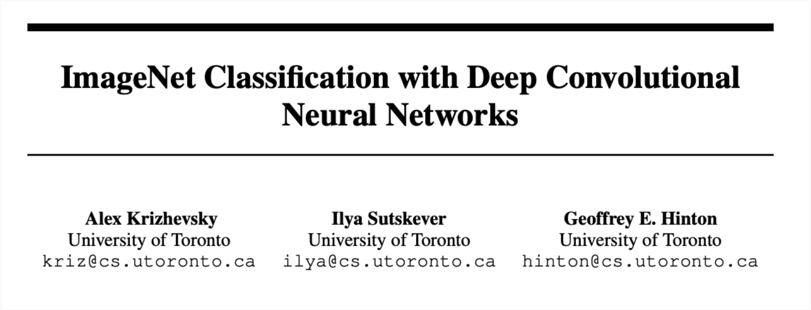
Around 2005 marks the end of the AI winter – into the golden age!

Papers We Will Look At Today

1. *ImageNet Classification with Deep Convolutional Neural Networks*,
   Krizhevsky, et. Al (NIPS 2012)
2. *Generative Adversarial Networks*,
   Goodfellow et. Al (NIPS 2014)
3. *Adam: A Method for Stochastic Optimization*,
   Kingma et. Al (ICLR 2015)
4. *Mastering the game of Go with Deep Neural Networks & Tree Search*,
   Deepmind (Nature 2016)
5. *Attention Is All You Need*,
   Vaswani et. Al (2017)

AlexNet

AlexNet (2012)



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

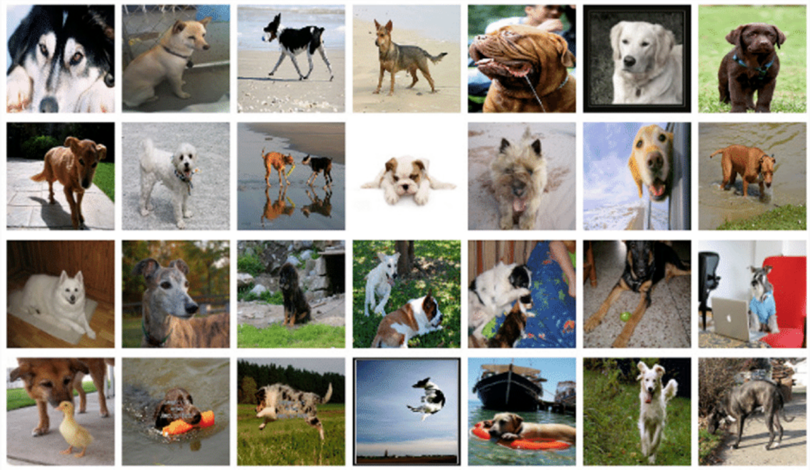
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

The dawn of a new era for computer vision.

ImageNet

A 10+ million image dataset released in 2009, organized by the Stanford Vision Lab. Pictures of objects tagged with a WordMap – contains hundreds of pictures per word class.



ILSVRC

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a challenge hosted by ImageNet each year, which usually features some image classification challenge. It is the Olympics of Computer Vision.

- 1000 classes
- 1.2 million training images (256 by 256 pixels cropped from ImageNet).
- 50K validation set, 150K test set.
- Labelled with Mechanical Turk.

AlexNet (2012)

A deep convolutional neural network model trained on GPUs! While not the first deep CNN, see: LeNet-5 (LeCun, 1998), this result was such a breakthrough that it propelled CNNs' popularity into every single computer vision task.

They were the winning submission to the ILSVRC 2012 competition! Shattered the previous state of the art image classification benchmarks.

- ILSVRC 2012 Top-5 error rate of 15%, compared to runner-up 26%.

AlexNet Classification

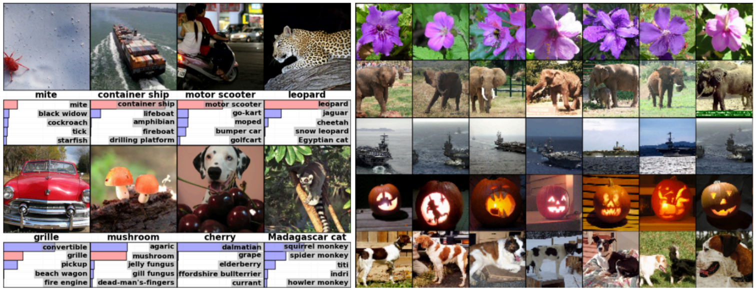


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

AlexNet Innovations

- Deeeeeep Learning.
- GPU for training.
- ReLU:
  - Faster convergence compared to saturating non-linearity.
- Dropout:
  - Less overfitting
- Overlap pooling
- Stochastic batch gradient descent – fixed weight decay and momentum for optimizer.
- Trained on two GTX580 GPUs for about a week!

Architecture

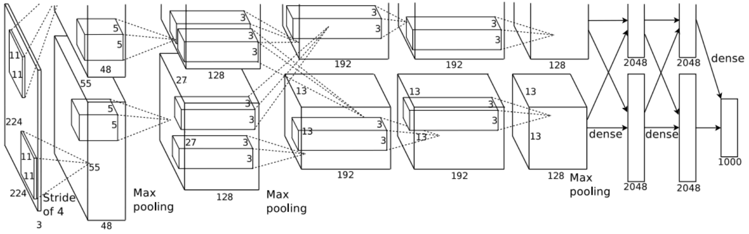


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

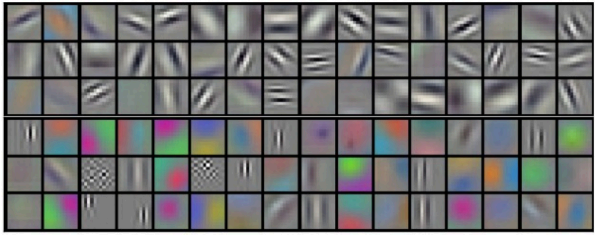
The split in this diagram denotes the split between the two GPUs.

Architecture

- Input: (224, 224, 3) dimensional vector.
- 8 layers: 5 convolutional layers + 3 fully connected layers.
- Number of kernels per layer: 96, 256, 384, 384, 256.
- Dimensions of kernels per layer:
  (11, 11, 3), (5, 5, 48), (3, 3, 256), (3, 3, 192), (3, 3, 192)
- Fully connected layer units: 4096, 4096, 1000.
- In total, 660K units, 61M parameters, and over 600M connections.

Today, we can reduce the number of parameters in AlexNet by 10x and reach same accuracy.

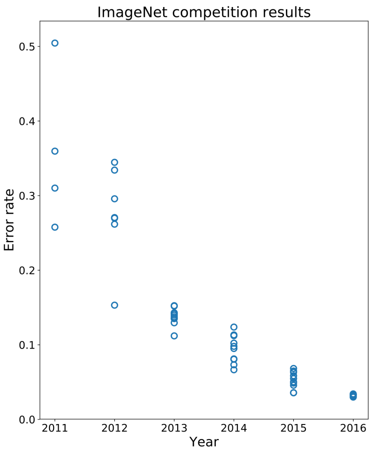
Convolutional Kernels


Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

Successors to AlexNet

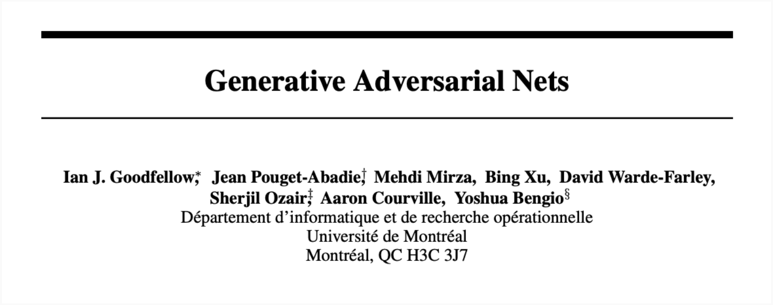
- VGG-16 and VGG-19
- GoogLeNet
- ResNet
    - Winner of ILSVRC 2015
    - Skip layer allows for super deep models without overfitting.
    - 152 layer version submitted for competition.
    - 3.7% top-5 error rate for ImageNet Classification.

Progress


ImageNet competition results
Error rate
Year
0.0
0.1
0.2
0.3
0.4
0.5
2011
2012
2013
2014
2015
2016

Generative Adversarial Networks

Generative Adversarial Nets (2014)



Generative Adversarial Nets

Ian J. Goodfellow[*], Jean Pouget-Abadie[†], Mehdi Mirza, Bing Xu, David Warde-Farley,
Sherjil Ozair[‡], Aaron Courville, Yoshua Bengio[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

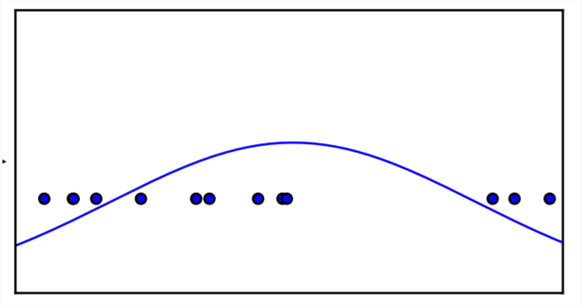
”The most exciting idea in Deep Learning in decades.” – Yann LeCun

Generative Models

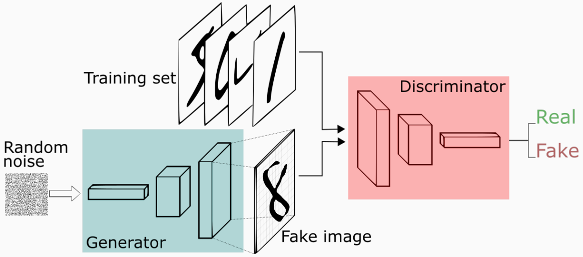
The Problem: Instead of performing classification or regression, we want to **generate** data from a distribution.

- Density Estimation: Learn the underlying density function of the sample data.
- Sample Generation: Sample points from the underlying density function.

Generative Models



Overview


Training set
Random noise
Generator
Fake image
Discriminator
Real
Fake

Generative Adversarial Networks

Two players (Neural Networks) that are adversarial to each other.

- The Discriminator, D: A classifier which takes a data point, and determines whether it is generated by G, or drawn from the true distribution.
- The Generator, G: generates adversarial examples from the sample distribution against the discriminator.

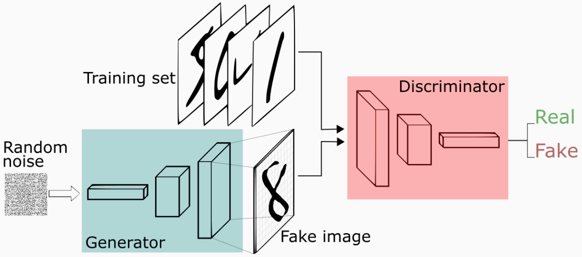
Example: Counterfeit Money vs Police

Consider this analogy in the real world.

- Counterfeiters create fake coins that try to look as real as possible.
- Police try to determine whether coins are real or fake.
- The two entities learn from each other, producing better coins, and better detectors.

What does this model converge to?

Overview


Training set
Discriminator
Real
Fake
Random noise
Generator
Fake image

Discriminator

$D : X \to \{0,1\}$, where is a neural network which takes in a data point from the domain X and determines whether it is real or not.

D is trained such that it maximizes the probability of labelling the inputs correctly. Its objective is to maximize:

$$\mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Equivalently, we can set the loss function to be:

$$J^{(D)} = -\mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] - \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

This is the *Jenson-Shannon divergence* of the true distribution and $G(z)$.

Generator

$G : Z \to X$, where Z is the latent domain, for example $Z := [0,1]^d$.

G is the generator network, which takes some random noise $z \in Z$ and produced $G(z) \in X$, a ”fake data point”. If X is the domain of pictures then $G(z)$ would be a generated picture.

During training, we want G to minimize D's accuracy when it comes to generated samples. Its objective function is:

$$J^{(G)} = \log(1 - D(G(z)))$$

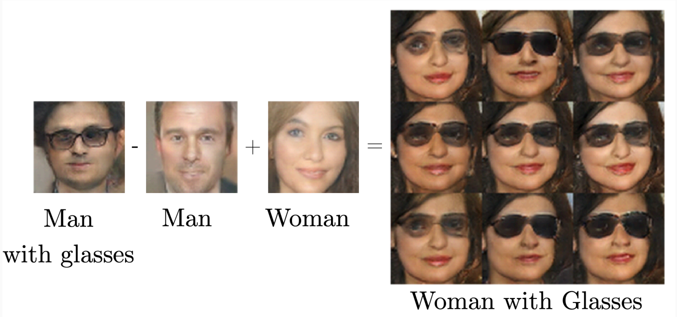
There are a lot of variations that produce similar optimal values, with stronger gradients, eg.

$$J^{(G)} = -J^{(D)}$$

Z-space

While many examples of GANs treat the z vector that is inputted into G as a random vector, we can consider it as a latent space.

Specifically, we can perform arithmetic on the latent space vectors. Thus, we can see Z as a low dimensional latent representation of X where important features are preserved on linear relations.



(Radford et al, 2015)

Minimax

Then, training the two models together, we have the following minimax value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

Note that the order matters!

$$\min_G \max_D V(D,G) \neq \max_D \min_G V(D,G)$$

The latter suffers from mode collapse: all mass converges to the most likely point.

A Diagram

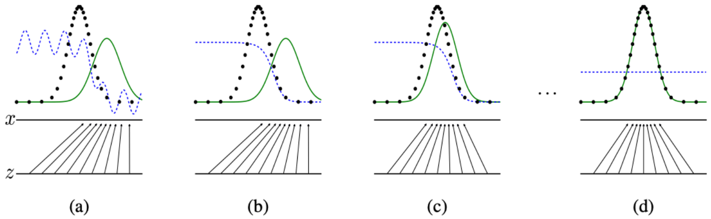
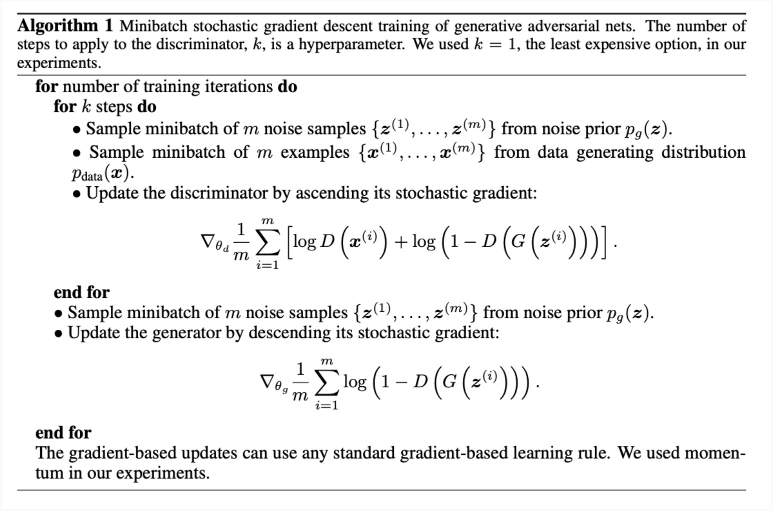


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

The Original Algorithm



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

The Optimal Discriminator

For a fixed G, the optimal discriminator is:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}$$

An intuitive result, but proven in the paper.

Then, the optimal point for our GAN is achieved when minimizing $C(G) =$

$$\mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)} \right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \frac{p_{model}(x)}{p_{data}(x) + p_{model}(x)} \right]$$

Equivalence of $C(G)$ with Jenson-Shannon divergence

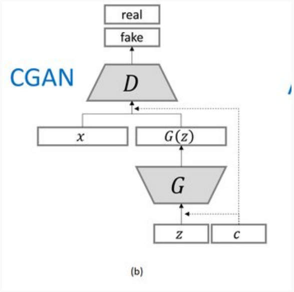
With a bit of algebra, we can show that $C(G)$

$$
\begin{aligned}
&= \mathbb{E}_{x\sim p_{data}(x)}\left[\log \frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}\right] \\
&\quad + \mathbb{E}_{z\sim p_z(z)}\left[\log \frac{p_{model}(x)}{p_{data}(x)+p_{model}(x)}\right] \\
&= -\log(4) + KL\left(p_{data} \,||\, \frac{p_{model}+p_{data}}{2}\right) + KL\left(p_{model} \,||\, \frac{p_{model}+p_{data}}{2}\right) \\
&= -\log(4) + 2 * JS(p_{data} \,||\, p_{model})
\end{aligned}
$$

Since the Jenson-Shannon divergence is always non-negative, and only zero of $p_{data} = p_{model}$, it follows that the optimal value exists only when $p_{data} = p_{model}$.

This is an elegant proof of this fact: at the optimal value, assuming perfect training, the GAN will converge to a perfect generator that is indistinguishable from the true distribution.

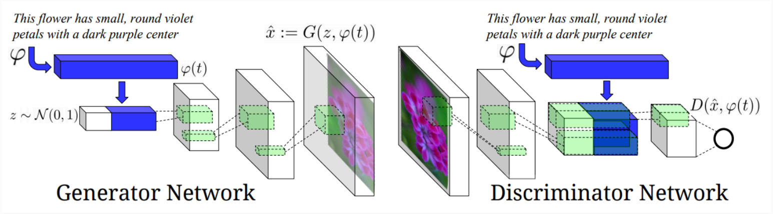
Conditional GANs



- Instead of only *z*, we also pass in an auxiliary information into *G* and *D*.
- For example, we can pass in the class label.

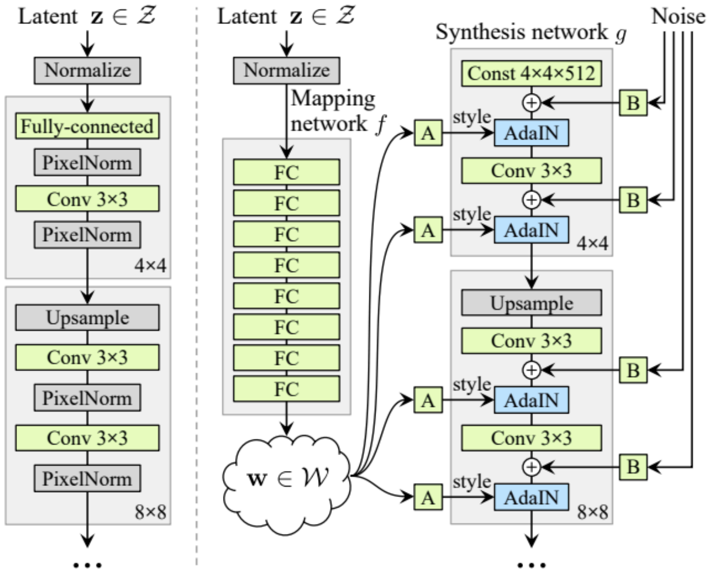
Text to Image Generation with Conditional GANs

From *Generative Adversarial Text to Image Synthesis*, Reed et. Al 2016.



This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
Generator Network
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Discriminator Network

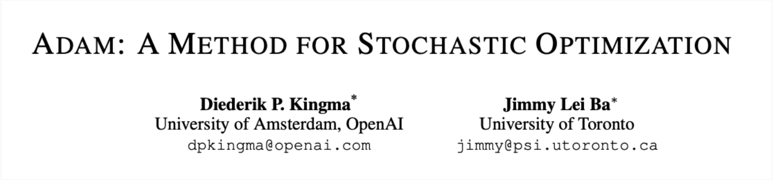
Generating Faces with DC-GANs, NVIDIA 2018



DC-GANs Faces

Adam: A Method for Stochastic Optimization

Adam (2014)


ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

The most popular optimization solver today.

What are Optimization Algorithms?

- Machine Learning is very concerned with a specific problem: given a function $f : S \to \mathbb{R}$, find $\arg_{\theta \in S} \min f(\theta)$.
- The f that Machine Learning practitioners want to optimize is the loss function.
- Often, we can make assumptions about f: eg. f is continuous, f is convex, f is analytic. There are entire CO courses dedicated to optimization under assumptions.
- Finally, sometimes, there is an analytic solution. Example:

$$\theta = (X^T X)^{-1} X^T y$$

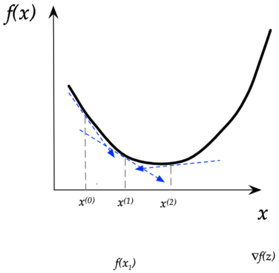
What are Optimization Algorithms?

- In Deep Learning, there is almost never ever an analytic solution. Functions defined by neural networks are far too complex, and the manifolds are wacky.
- Instead, we try to find a solution using **numerical** methods – algorithms that try to approximate the minimum through a systematic convergence test.
- The most famous class of numerical methods is **Gradient Descent**.

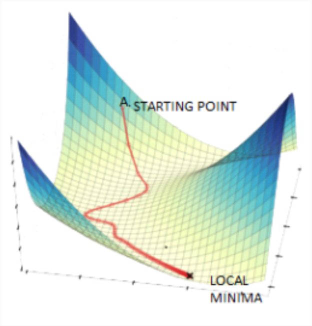
Gradient Descent

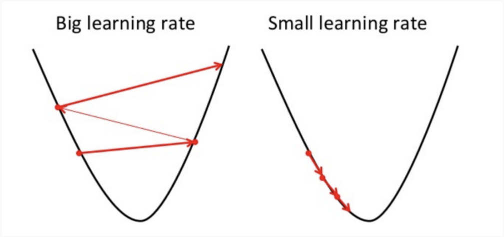
Algorithm 1 Gradient Descent

```
1: Guess $\mathbf{x}^{(0)}$, set $k \leftarrow 0$
2: while $||\nabla f(\mathbf{x}^{(k)})|| \geq \epsilon$ do
3:     $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$
4:     $k \leftarrow k + 1$
5: end while
6: return $\mathbf{x}^{(k)}$
```



Gradient Descent





The learning rate α matters a lot on whether gradient descent converges.

What is Adam?

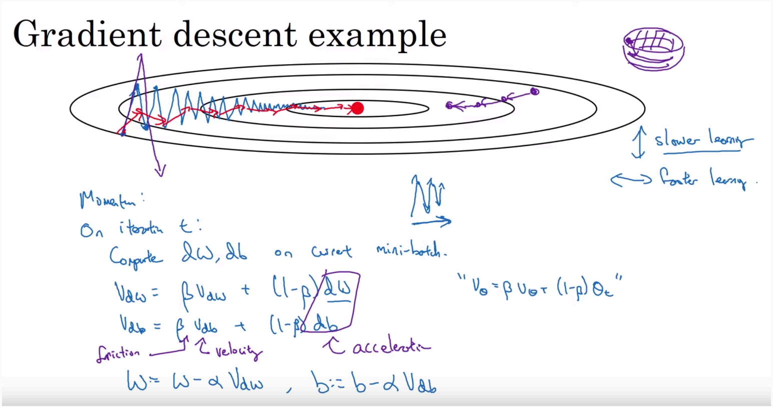
Adam combines two existing concepts:

- Momentum
- RMSProp

Momentum
Instead of using just the gradient, instead, use an exponentially weighted moving average (EWMA) of your last few gradients.

RMSProp
Keep weights for components of direction, and re-weight gradient component-wise based on square root of second derivative.

Momentum



Gradient descent example

Momentum:

On iteration t:

Compute dW, db on current mini-batch.

$$V_{dW} = \beta V_{dW} + (1-\beta)\, dW$$

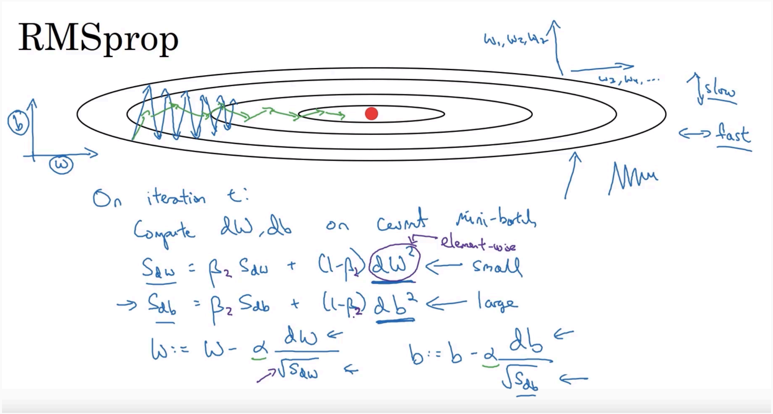
$$V_{db} = \beta V_{db} + (1-\beta)\, db$$

friction → β ← velocity ← acceleration

"$V_\theta = \beta V_\theta + (1-\beta)\,\theta_t$"

$W := W - \alpha V_{dW}$, $b := b - \alpha V_{db}$

Root Mean Squared Prop



RMSprop

On iteration t:

Compute dW, db on current mini-batch

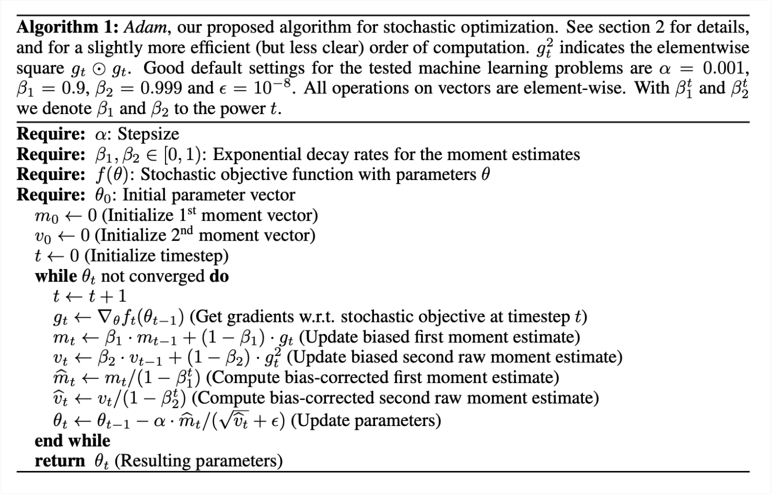
element-wise

$$\underline{S_{dW}} = \beta_2 S_{dW} + (1-\beta_2)\, \underline{dW^2} \longleftarrow \text{small}$$

$$\rightarrow \underline{S_{db}} = \beta_2 S_{db} + (1-\beta_2)\, \underline{db^2} \longleftarrow \text{large}$$

$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

Adam Algorithm



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Results

Faster convergence for loss functions during training, saving time and money.

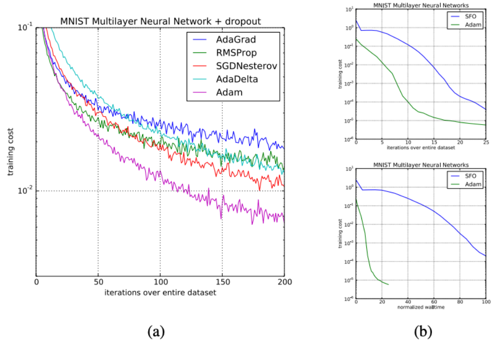


Figure 2: Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function. We compare with the sum-of-functions (SFO) optimizer (Sohl-Dickstein et al., 2014)

More results

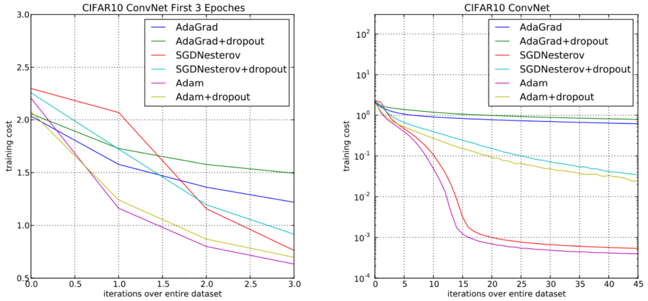
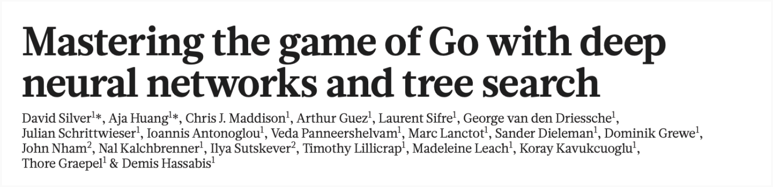


Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

AlphaGo

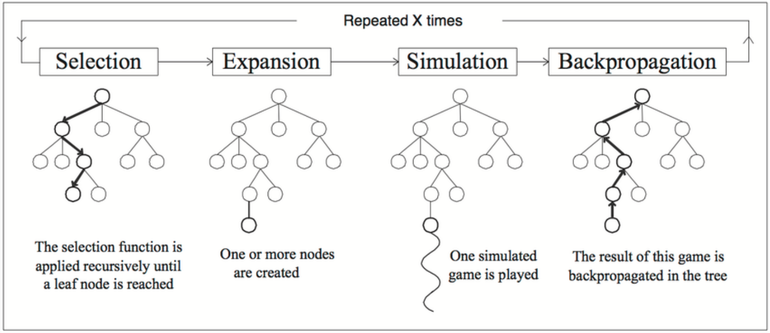
AlphaGo



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]

Traditional Algorithm: Monte Carlo Tree Search


Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Policy Network and Value Network

Policy Network
$P : S \to Q$. Given the state of the game, produces a distribution of actions to take. Uses convolutional neural networks to learn representations of the game.

Value Network
$V : S \to \mathbb{R}$. Given the state of the game, produces a single scalar value: the value of the state, or the chances of winning in that state. Uses convolutional neural networks to learn representations of the game.

These are obviously great to have, but how do we train them?

Policy Gradient

Answer: Pretraining + **Reinforcement Learning**

AlphaGo's policy network is first trained on a dataset of 30 million human games, to predict what human professionals would play in each position. This is a supervised learning task.

This isn't sufficient: we need to use reinforcement learning, through self play. We play against a randomly selected previous iteration of the policy network.

The reward of each terminal state is +1 or -1, and 0 for all nonterminal states. Update policy with stochastic gradient ascent on the policy gradient.

Training Value Network

After training the policy network through self play, we use reinforcement learning to train the value network, by having it predict the results of games where the policy gradient plays itself.

Problem: Cannot use multiple states from the same game to train, as it leads to overfitting. Successive positions are highly correlated, and share the same terminal position, thus games would be memorized instead of learned.

Solution: Only use 1 move from a single game. Play 30 million distinct games just to create a training set for the value network. Use millions of dollars in compute.

AlphaGo’s MTCS

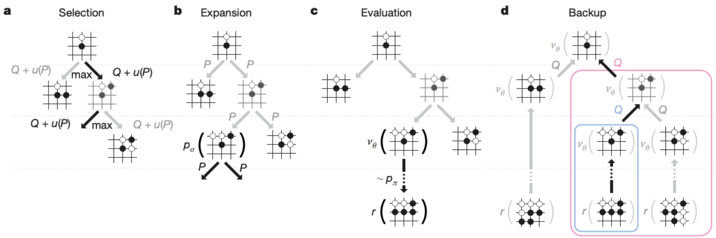
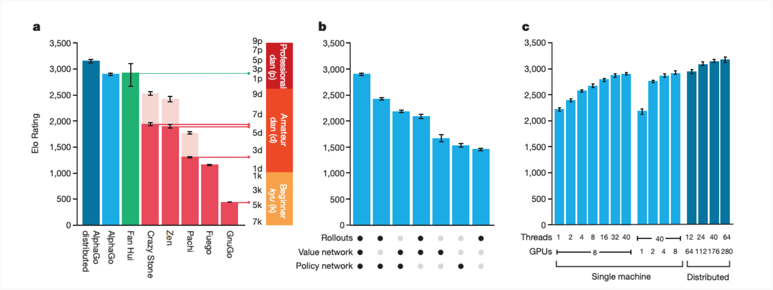


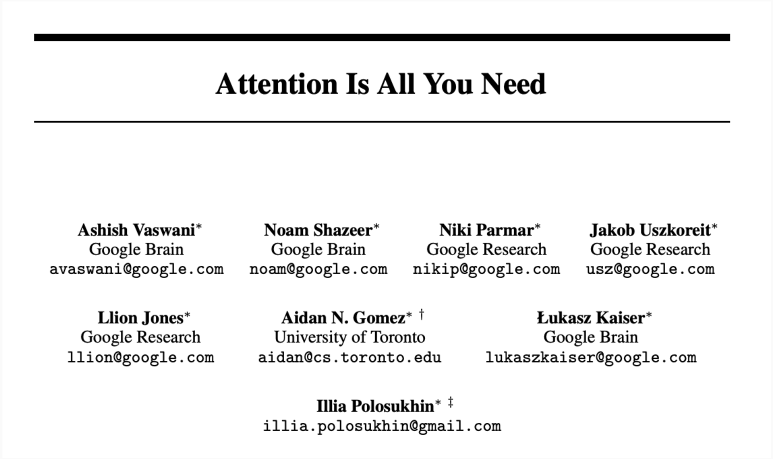
Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b,** The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c,** At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d,** Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

Evaluating Performance



Attention Is All You Need

Attention Is All You Need



Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

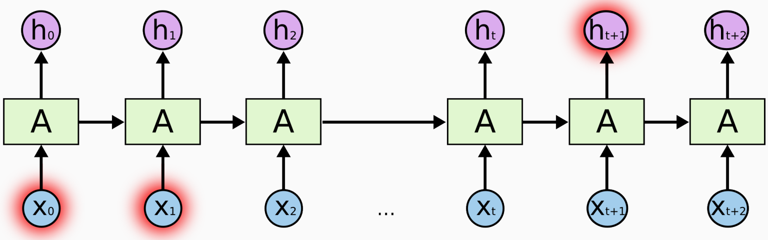
Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Ushered in revolutionary NLP technology in last 2 years.

NLP before Transformers

- Seq2Seq encoder-decoder networks.
- Sequential data is modelled with RNNs and LSTMs.



- But what about long term dependence?

Long Term Dependence

BLEU scores

BLEU = bilingual evaluation understudy

An algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Score between 0 to 1.

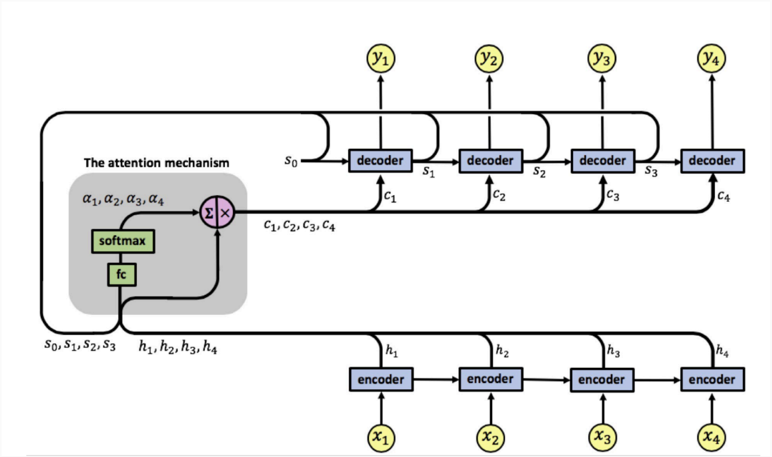
BLEU scores for long sentences (> 30 words) using RNN / LSTM models are much lower, because they still struggle to maintain long term dependence in a sentence (words far apart are less connected in RNN/LSTM model).

Example: When I visited France, I had a lot of fun with my friends, my parents, my professors, and the locals, even though I am cannot fluently speak <>.

Attention

Intuition: Humans don’t read the entire sentence before translating. They translate blocks at a time based on what part of the sentence they’re paying attention to. The words that they pay attention to can come from much earlier or much later.

Attention

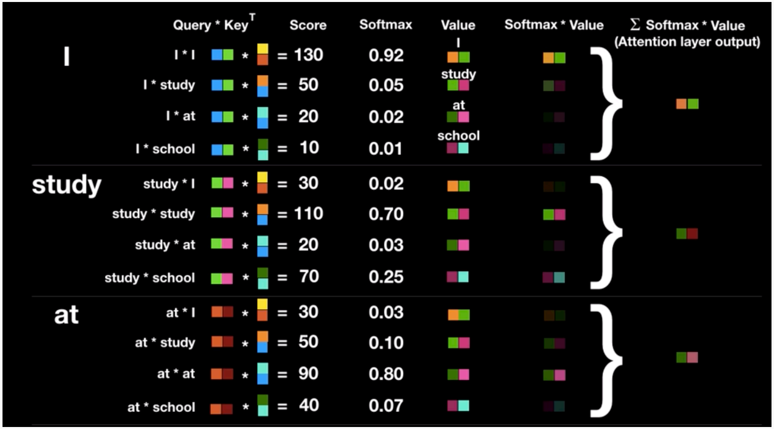


y_1
y_2
y_3
y_4
The attention mechanism
s_0
decoder
s_1
decoder
s_2
decoder
s_3
decoder
$\alpha_1, \alpha_2, \alpha_3, \alpha_4$
Σ ×
c_1
c_2
c_3
c_4
c_1, c_2, c_3, c_4
softmax
fc
s_0, s_1, s_2, s_3
h_1, h_2, h_3, h_4
h_1
h_2
h_3
h_4
encoder
encoder
encoder
encoder
x_1
x_2
x_3
x_4

Attention

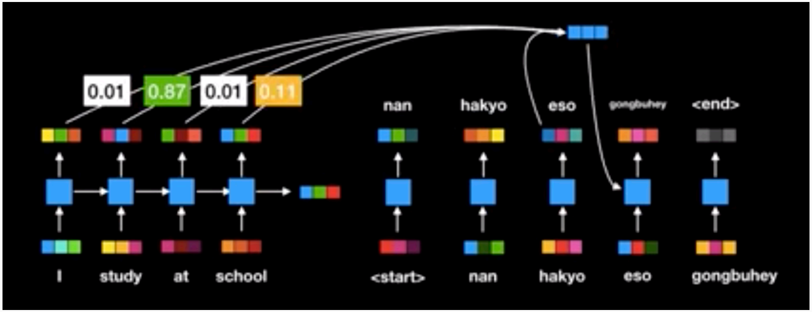
Attention function: Maps a query and a set of key-value pairs to an output.

Output: weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.



	Query * Key^T	Score	Softmax	Value	Softmax * Value	Σ Softmax * Value (Attention layer output)
I	I * I	= 130	0.92	I		
	I * study	= 50	0.05	study		
	I * at	= 20	0.02	at		
	I * school	= 10	0.01	school		
study	study * I	= 30	0.02			
	study * study	= 110	0.70			
	study * at	= 20	0.03			
	study * school	= 70	0.25			
at	at * I	= 30	0.03			
	at * study	= 50	0.10			
	at * at	= 90	0.80			
	at * school	= 40	0.07			

RNN + Attention



The big question.

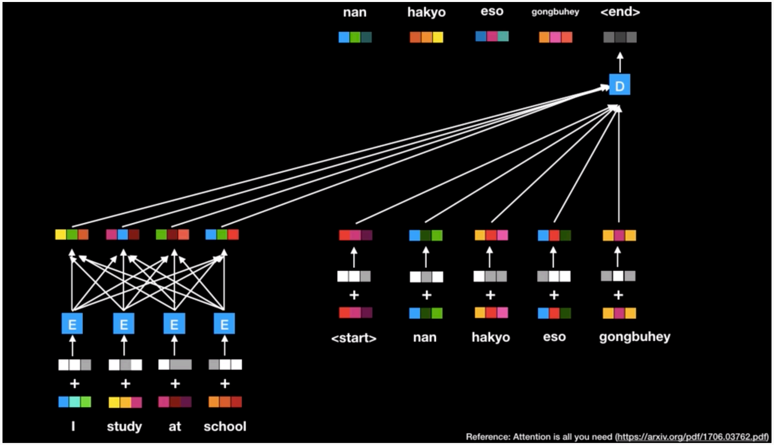
RNNs have a long compute chain – difficult to parallelize for training.
GPUs are only efficient at performing computations in parallel.

Do you really need to use the RNN?

The answer: No.

Attention is all you need.

Attention is all you need.



Reference: Attention is all you need (https://arxiv.org/pdf/1706.03762.pdf)

Transformers

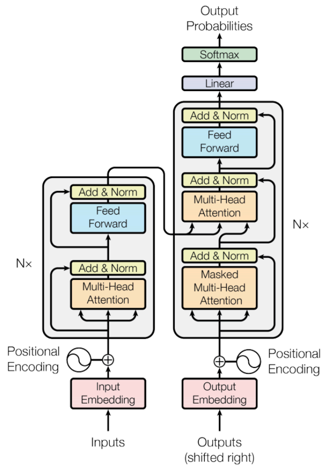
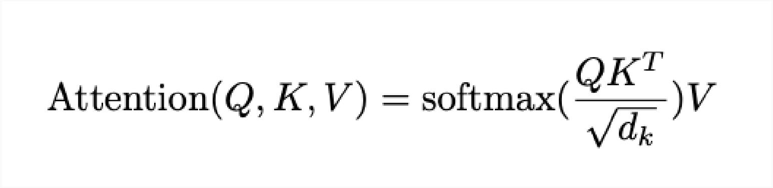


Figure 1: The Transformer - model architecture.

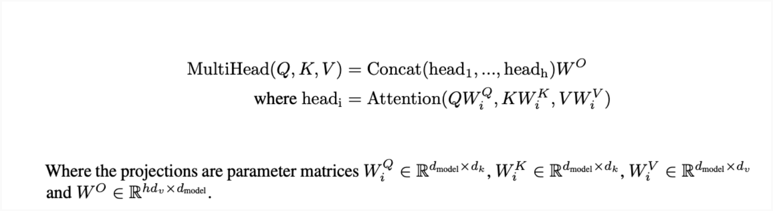
Positional Encoding

- The model still needs to know the position of each word – otherwise, how will it decide word order?
- Solution: Positional encoding. Apply a ”position vector” for each word to the input – which depends on the order of the word.

Multi-Head Attention



$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

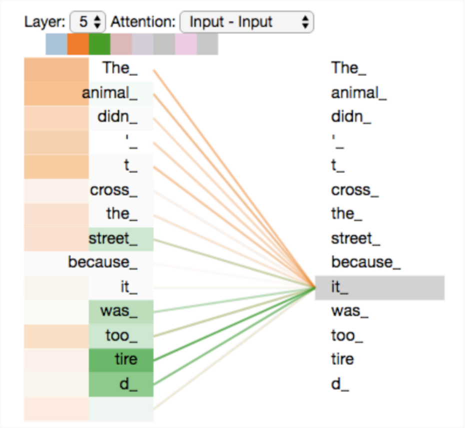


$$\begin{aligned}\mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O \\ \text{where}\ \mathrm{head_i} &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

- Instead of only 1 attention function, instead store several (8).

Self Attention



Trained on WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs with 8 NVIDIA P100 GPUs for 12 hours (base model) or 3 days (large model), with the Adam optimizer + dropout.

Outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4!

All while using significantly less training cost compared to other models.