Putting It All Together

John Urbanic

Parallel Computing Scientist
Pittsburgh Supercomputing Center

Distinguished Service Professor
Carnegie Mellon University

Building Blocks

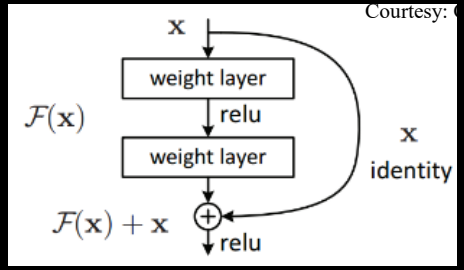
So far, we have used Fully Connected and Convolutional layers. These are ubiquitous, but there are many others:

- Fully Connected (FC)
- Convolutional (CNN)
- Residual (ResNet) [Feed forward]
- Recurrent (RNN), [Feedback, but has vanishing gradients so...]
- Long Short Term Memory (LSTM)
- Bidirectional LSTM
- Transformer (Attention based)
- Restricted Boltzmann Machine
-
-

Several of these are particularly common...

Residual Neural Nets

We've mentioned that disappearing gradients can be an issue, and we know that deeper networks are more powerful. How do we reconcile these two phenomena? One, very successful, method is to use some feedforward.



- Helps preserve reasonable gradients for very deep networks
- Very effective at imagery
- Used by AlphaGo Zero (40 residual CNN layers) in place of previous complex dual network
- 100s of layers common, Pushing 1000

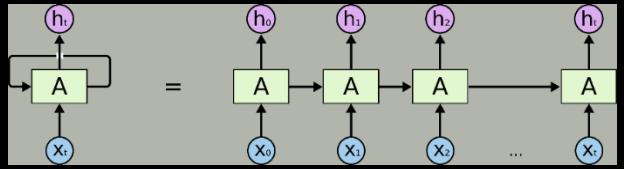
```
#Example: input 3-channel 256x256 image
x = Input(shape=(256, 256, 3))
y = Conv2D(3, (3, 3))(x)
z = keras.layers.add([x, y])
```

Haven't all of our Keras networks been built as strict layers in a *sequential* method? Indeed, but Keras supports a *functional* API that provides the ability to define network that branch in other ways (multiple inputs or multiple outputs, or layers with multiple inputs or multiple outputs, or any non-linear topology such as here). It is easy and here (https://www.tensorflow.org/guide/keras/functional) is an MNIST example with a 3 dense layers.

More to our current point, here (https://www.kaggle.com/yadavsarthak/residual-networks-and-mnist) is a neat experiment that uses *15*(!) residual layers to do MNIST. Not the most effective approach, but it works and illustrates the concept beautifully.

Recurrent Networks (RNNs)

If feedforward is useful, is there a place for feedback? Indeed, it is currently at the center of the many of the most effective techniques in deep learning.



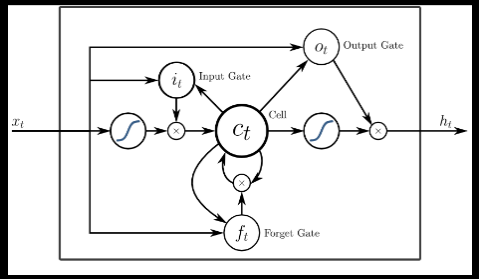
Many problems occur in some context. Our MNIST characters are just pulled from a hat. However most character recognition has some context that can greatly aid the interpretation, as suggested by the following - not quite true - text.

"Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteers be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe."

To pick a less confounding example. The following smudged character is pretty obvious by its context. If our network can "look back" to the previous words, it has a good chance at guessing the, otherwise unreadable, "a".

LSTMs

This RNN idea seems an awful lot like "memory", and suggests that we might actually incorporate a memory into networks. While the Long Short Term Memory (LSTM) idea was first formally proposed in 1997 by Hochreiter and Schmidhuber, it has taken on many variants since. This is often not explained and can be confusing if you aren't aware. I recommend "LSTM: A Search Space Odyssey" (Greff, et. al.) to help.



The basic design involves a memory cell, and some method of triggering a forget. *tf.keras.layers.LSTM* takes care of the details for us (but has a *lot* of options).

The Keras folks even provide us with an MNIST version (https://keras.io/examples/mnist_hierarchical_rnn/), although I think it is confusing as we are now killing a fly with a bazooka.

I recommend https://keras.io/examples/conv_lstm/, which uses network is used to predict the next frame of an artificially generated movie which contains moving squares. A much more natural fit.

Bi-directional LSTMs

Often, and especially in language processing, it is helpful to see both forward and backward. Take this example:

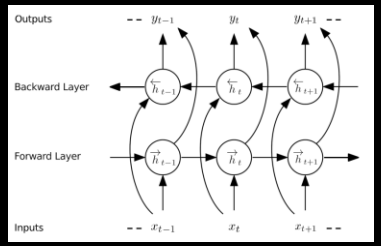
The dog chased the ca

Is the dog chasing a cat, or a car? If we read the rest of the sentence, it is obvious:

The dog chased the ca up the tree.

Adding even this very sophisticated type of network is easy in TF. Here is the network definition from the Keras *IMDB movie review sentiment analysis* example (https://www.tensorflow.org/tutorials/text/text_classification_rnn).

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
```



The first, embedding, layer introduces the concept of word embeddings - of central importance to any of you interested in natural language processing, and related to our running theme of dimensionality reduction. To oversimplify, here we are asking TF to reduce our vocabulary of vocab_size, so that every word's meaning is represented by a 64 dimensional vector.

Transformers

We have strayed solidly into the realm of Natural Language Processing (NLP). The current state of the art here, which has largely subsumed these earlier techniques, are Transformer, or self-attention based networks. These form the basis of ChatGPT and similar applications.

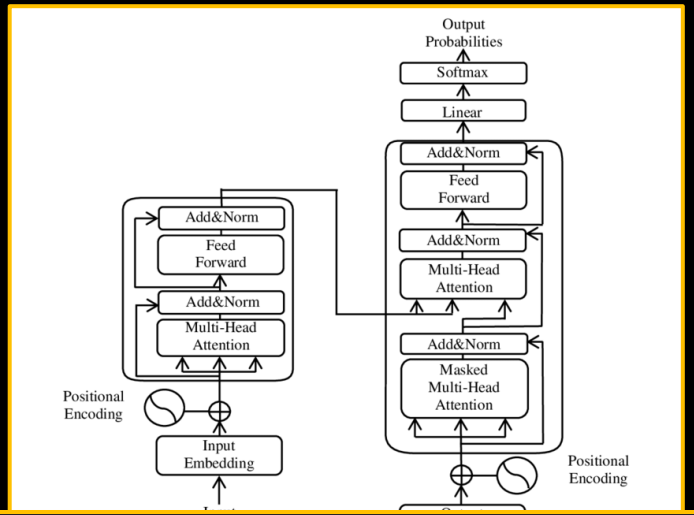
The seminal implementation goes all the way back to 2017 with *"Attention Is All You Need," Vaswani et al*.

While we use the basic building blocks you have learned, the overall architectures have a lot of parts.

The idea is to process the kind of sequential data we have been discussing, but with the ability to learn the relative important of different, perhaps distant, tokens. In other words, pay more attention to some relationships than others.

While these designs have proven ***surprisingly*** powerful in NLP (*emergent! emergent!*), they have yet to find a central use in scientific problems.

Given the incredible effort and funding inv
that the scientific community will find son



Once again, 3Blue1Brown
https://www.youtube.com/watch?v=wjZofJX0v4M

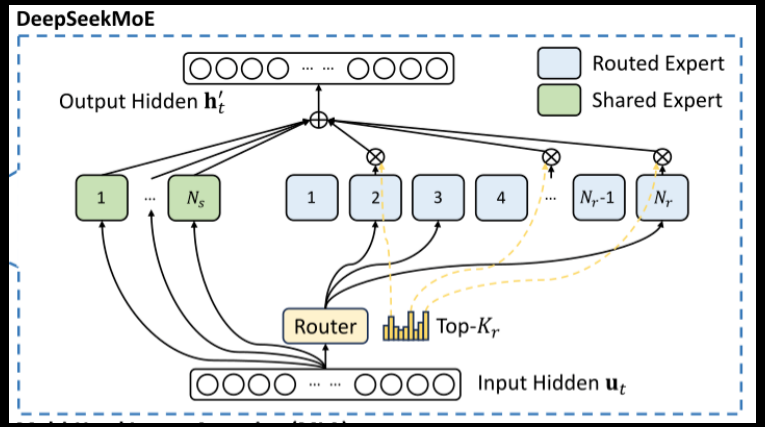
Mixture of Experts

And now we have reached the point where our 100+ billion parameter models are both expensive to train and to operate. As transformers in particular grow larger, their feedforward portions come to dominate, with 90+% or the parameters in the feedforward layers.

A technique that dates back to 1991 (""“Adaptive Mixture of Local Experts”) has become a very effective way to mitigate this issue.

MoE work by designating a number of experts, each its own sub-network within a larger neural network, and training a gating network to activate only the specific experts best suited to a given input.

This overall parameter count is commonly referenced as the *sparse parameter count* and can generally be understood as a measure of model capacity as well as how much memory your model will actually require. The number of parameters that will actually be used to process an individual token (as it transits through some expert blocks and bypasses others) is called the *active parameter count*, and can be understood as a measure of the model’s inference cost.



Mixture of Experts

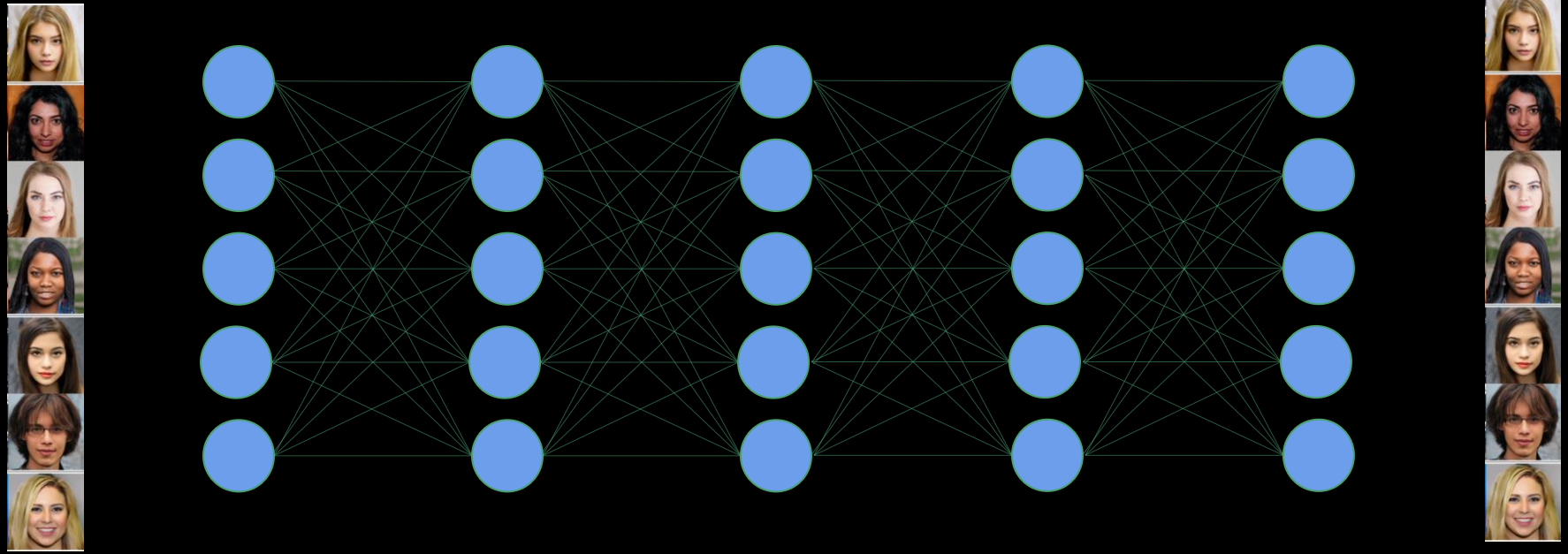
As we have already mentioned numerous times, GPU training hours have become the limiting resource. Training a fully dense model has become prohibitively expensive in terms of both time and cost. The Llama 2 set of models (fully dense) trained by Meta reportedly spent 3.3 million NVIDIA A100 GPU hours in training. This is roughly 3.3 million GPU hours on 1,024 GPUs at full capacity. With no downtime this would take would take over 3 months. This does not account for any hyperparameter investigation at all. So even though they introduce complexity the ability for MoEs to greatly reduce training time has made them essential in this space.

For example, using the most obvious top-expert routing strategy introduces the potential for the gating network to converge to activating just a few experts. This is a self-reinforcing problem: if a handful of experts are disproportionately selected early on, those experts will be trained more quickly, and then continue to be selected more as they now output more reliable predictions than the other, less-trained experts.

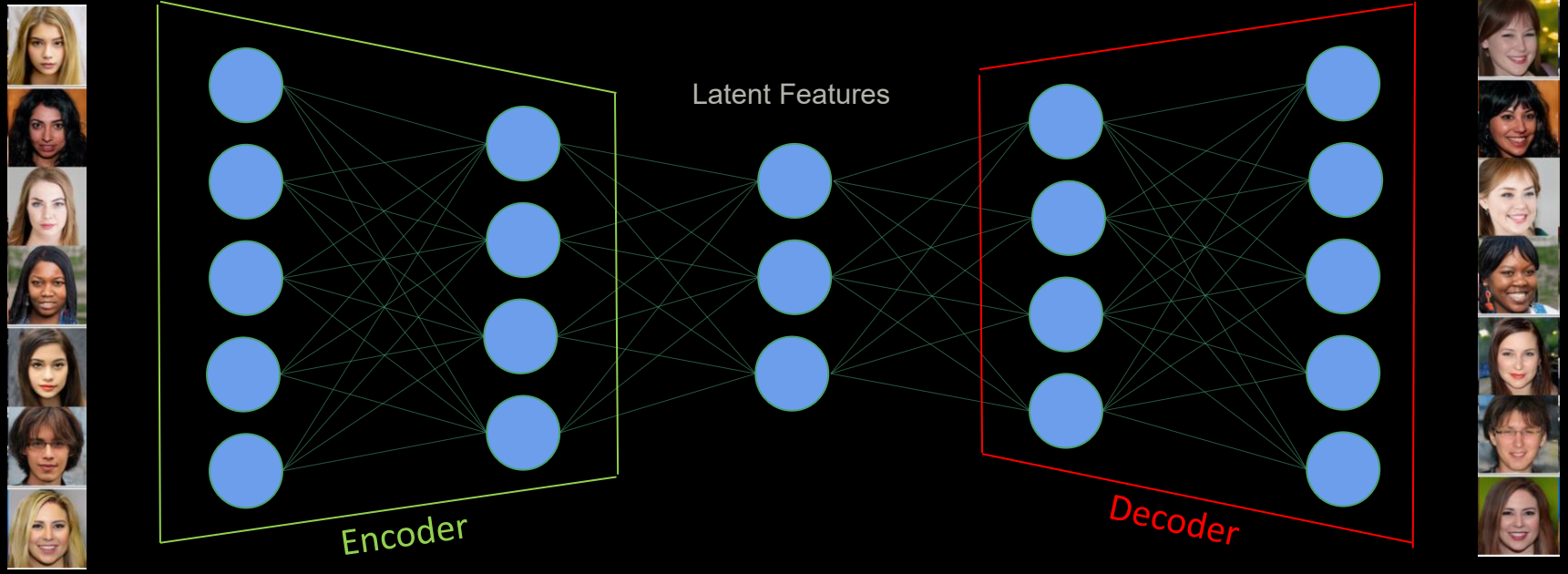
MoE are also very effective at inference time, which is becoming an increasing resource sink. Not only do they decrease the overall computation required, they also reduce latency. Latency decreases are particularly important in use cases like retrieval-augmented generation (RAG) and autonomous agents, which may require many calls to the model, compounding single-call latency.

In summary, the primary attraction of the MoE approach is that by enforcing sparsity, rather than activating the entire neural network for each input token, model capacity can be increased while essentially keeping computational costs constant.

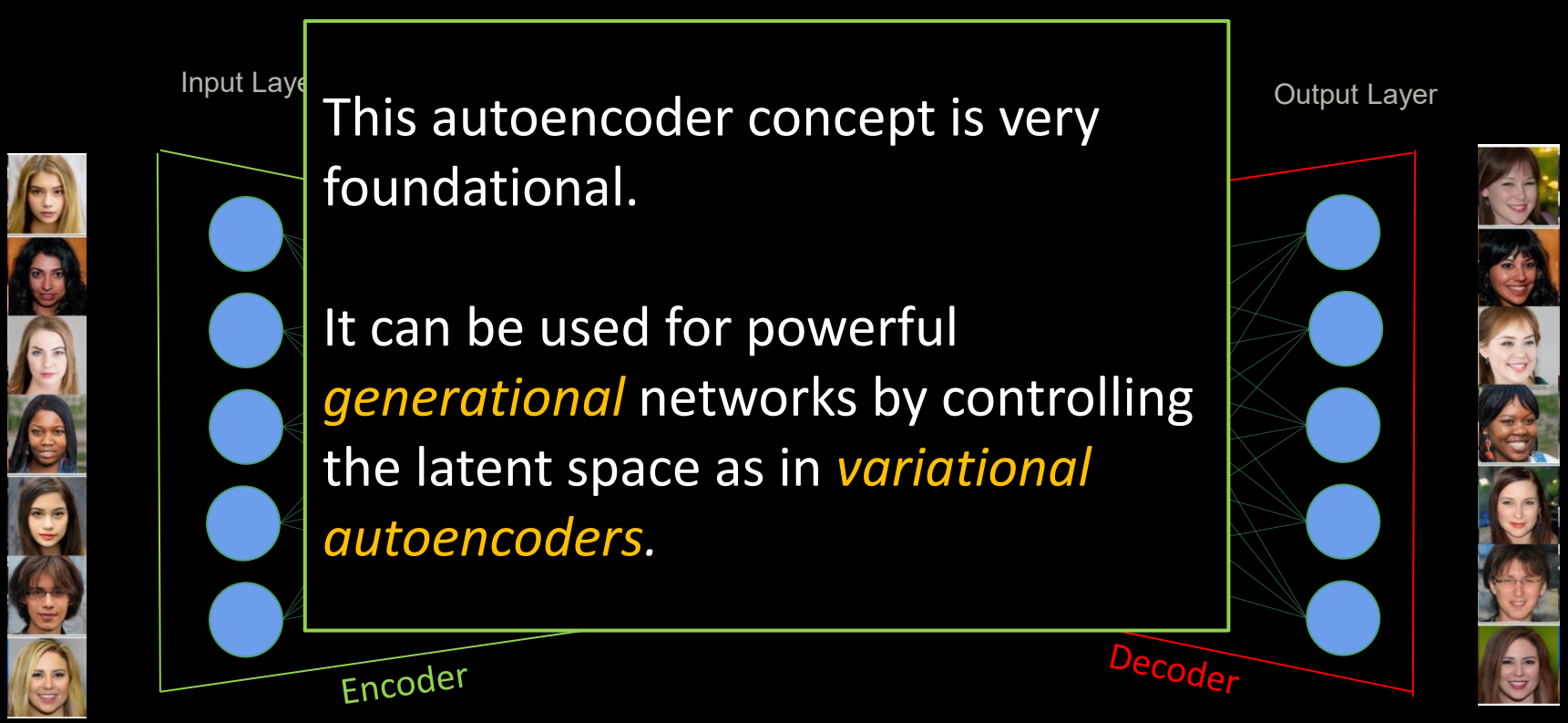
Autoencoder



Autoencoder



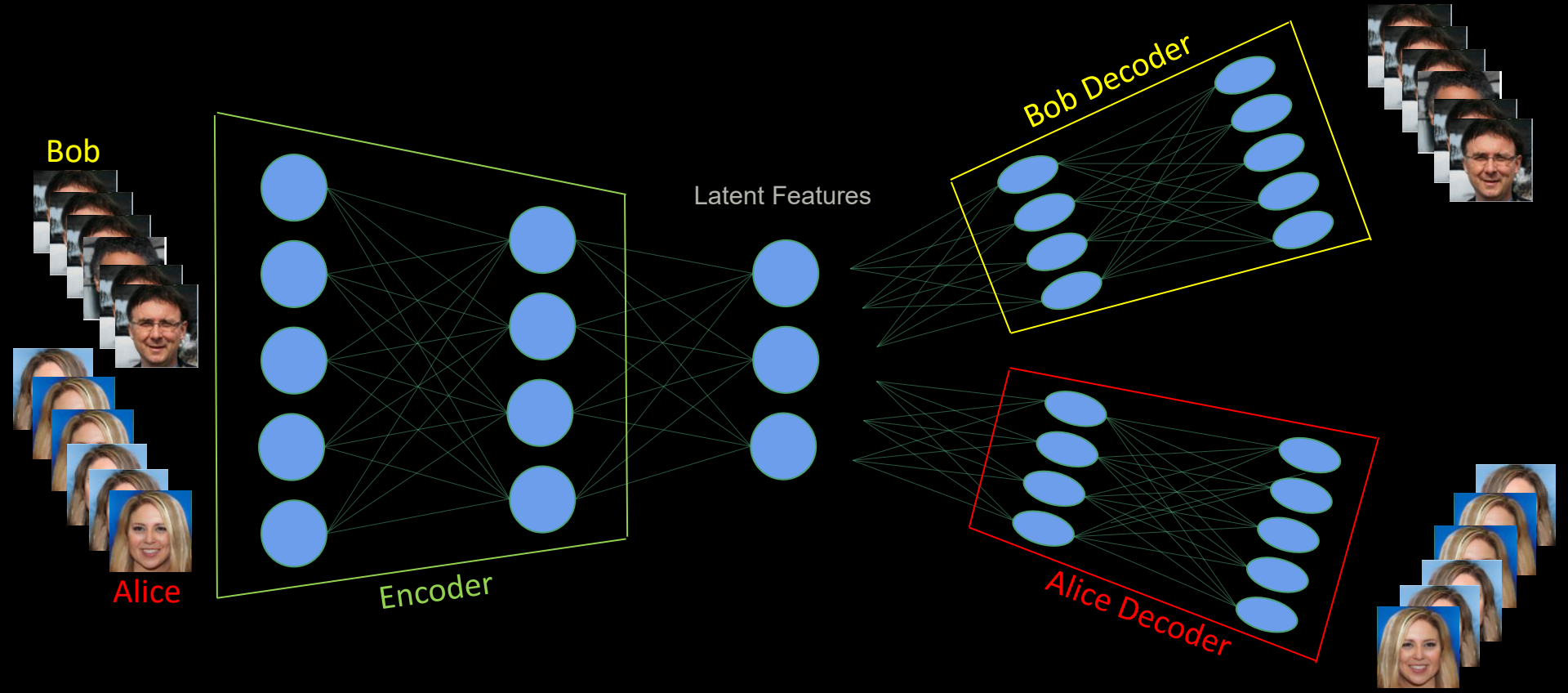
Autoencoder



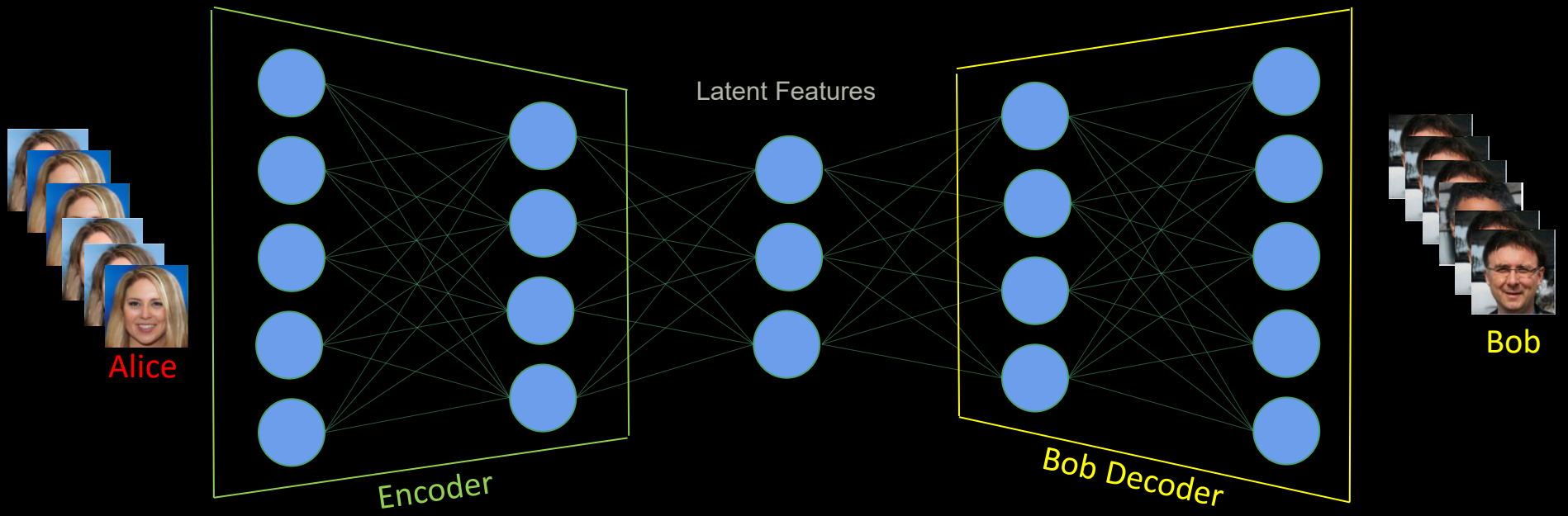
This autoencoder concept is very foundational.

It can be used for powerful *generational* networks by controlling the latent space as in *variational autoencoders*.

Deepfake Training



Deepfake At Work



Rando on Reddit

https://github.com/hacksider/Deep-Live-Cam

As he says, "me with a laptop in my bedroom."

Super Resolution

VideoGigaGAN: Towards Detail-rich Video Super-Resolution

Yiran Xu[1,2], Taesung Park[1], Richard Zhang[1], Yang Zhou[1], Eli Shecht
Feng Liu[1], Jia-Bin Huang[2], and Difan Liu[1]

[1] Adobe Research
[2] University of Maryland, College Park
http://videogigagan.github.io

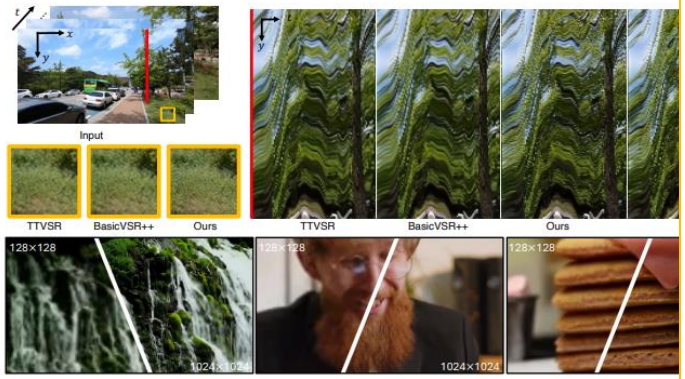


Fig. 1: We present **VideoGigaGAN**, a generative video super-resolution m
that can upsample videos with high-frequency details while maintaining temp
consistency. *Top:* we show the comparison of our approach with TTVSR [33]
BasicVSR++ [7]. Our method produces temporally consistent videos with m
fine-grained detailed than previous methods. *Bottom:* our model can produc
high-quality videos with 8× super-resolution. Please see the video results on
project page.

3.6 Loss functions

We use stardard, non-saturating GAN loss [14], R1 regularization [34], LPIPS [62] and Charbonnier loss [9] during the training.

$$\mathcal{L}(\mathbf{X}_t, \mathbf{x}_t) = \mu_{GAN}\mathcal{L}_{GAN}(\mathcal{G}(\mathbf{x}_t), \mathcal{D}(\mathcal{G}(\mathbf{x}_t))) + \mu_{R1}\mathcal{L}_{R1}(\mathcal{D}(\mathbf{X}_t)) + \mu_{LPIPS}\mathcal{L}_{LPIPS}(\mathbf{X}_t, \mathbf{x}_t) + \mu_{Char}\mathcal{L}_{Char}(\mathbf{X}_t, \mathbf{x}_t)\,, \tag{2}$$

where Charbonnier loss is a smoothed version of pixelwise ℓ_1 loss, μ_{GAN}, μ_{R1}, μ_{LPIPS}, μ_{Char} are the scales of different loss functions. $\mathbf{x}_t$ is one of the LR input frames, $\mathbf{X}_t$ is the corresponding ground-truth HR frame. We average the loss over all the frames in a video clip during the training.

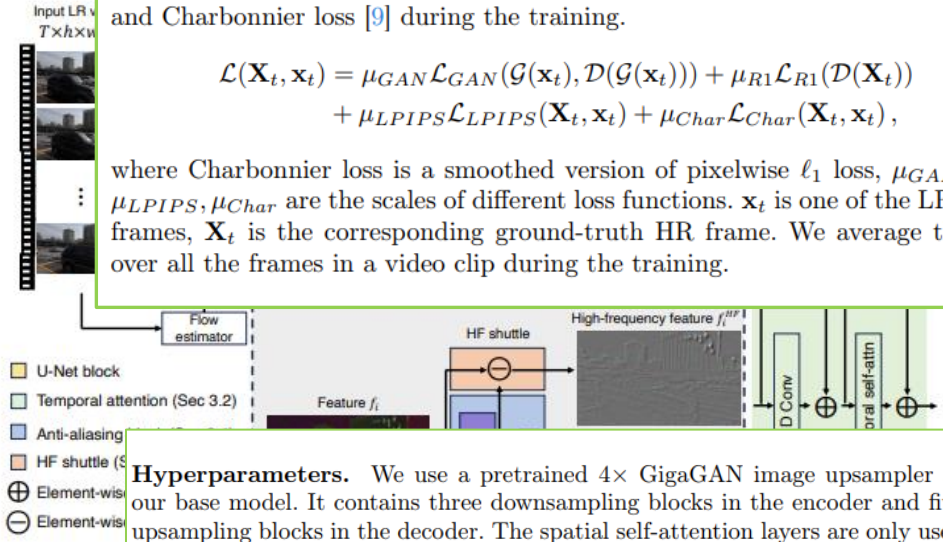
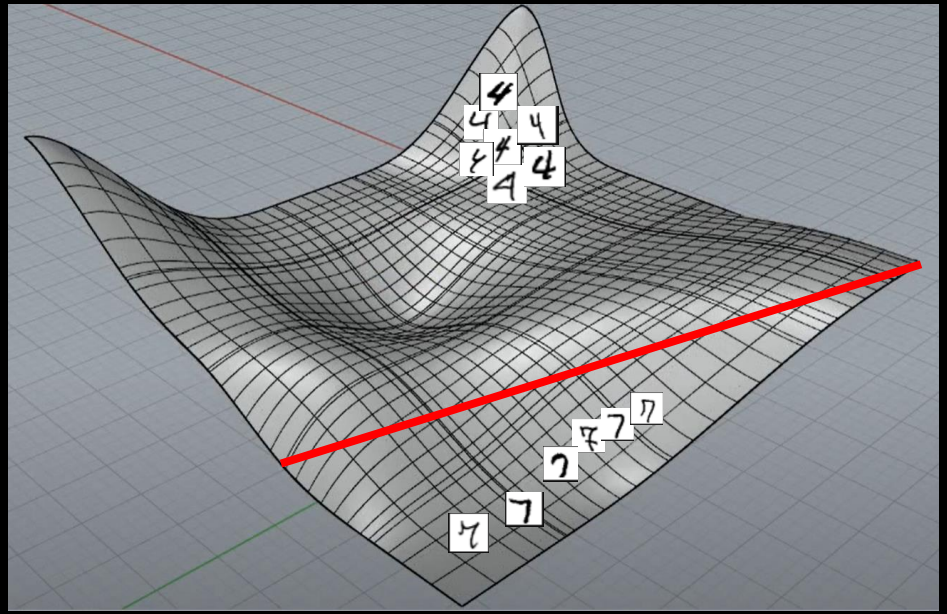


Fig. 3: Overvi
(VSR) model is
GigaGAN upsa
upsampler into
decoder blocks.
flow-guided pro
Anti-aliasing bl
shuttle the high frequency features via skip connection to the decoder layers to compensate for the loss of details in the BlurPool process.

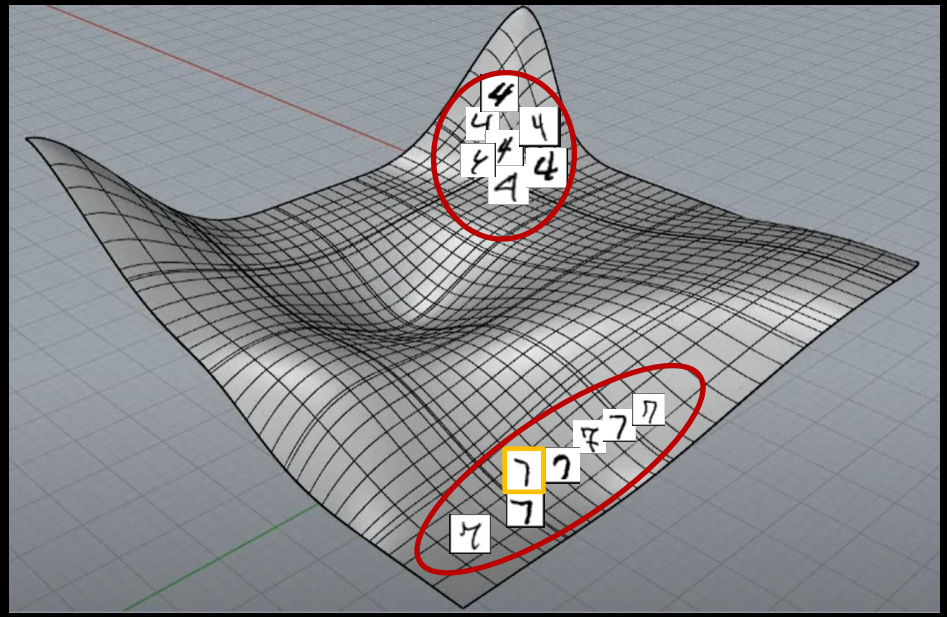
Hyperparameters. We use a pretrained 4× GigaGAN image upsampler as our base model. It contains three downsampling blocks in the encoder and five upsampling blocks in the decoder. The spatial self-attention layers are only used in the first block of the decoder for memory efficiency. For the flow network, we use a lightweight SpyNet [38]. For the low-pass filters, we use a kernel of $\frac{1}{16}[1, 4, 6, 4, 1]$ before the downsampling. We set $\mu_{GAN} = 0.05$, $\mu_{R1} = 0.2048$, $\mu_{LPIPS} = 5$, $\mu_{Char} = 10$ in Eqn. 2. During training, we randomly crop a 64×64 patch from each LR input frame at the same location. We use 10 frames of each video and a batch size of 32 for training. The batch is distributed into 32 NVIDIA A100 GPUs. We use a fixed learning rate of 5×10^{-5} for both generator and discriminator. The total number of training iterations is 100,000.

Discriminative vs. Generative

Discriminative models classify things, and need only know which side of the hyper-plane the instance lies on. Generative models need to understand the distribution to generate new instances.



Discriminative



Generative

Discriminative models need only capture the conditional probability of digit Y, given image X: P(Y|X). Generative models must understand the joint probability P(X,Y).

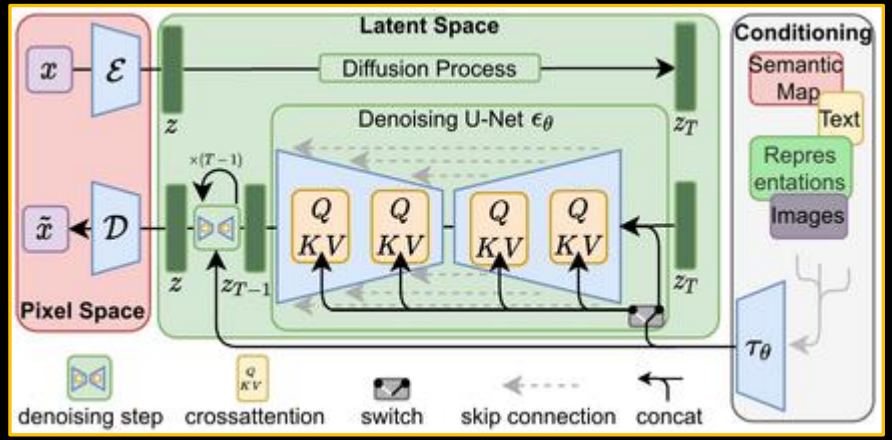
Generative in Action

Stable Diffusion, *DALL-E*, *Midjourney* and other such applications are built upon this idea.

For example, Stable Diffusion was trained on pairs of images and captions from Common Crawl data scraped from the web, where 5 billion image-text pairs were classified.

In a clever mashup of ideas we have discussed, this network attempts to de-noise images in conjunction with text prompts, resulting in some amazing "comprehension".

Stable Diffusion's code and model weights have been released, and it runs on consumer GPUs with 8 GB of VRAM!



Generative Networks

The New York Times

Meet DALL-E, the A.I. That Draws Anything at

New technology that
graphic artists — and

Give this article

Image Generated by AI

DALL-E generated these images by following a command for "a teapot in the shape of an avocado."
OpenAI

Image Generated by AI

DALL-E generated this image from a
playing chess." OpenAI

Hierarchical Text-Conditional Image Generation with CLIP Latents

Aditya Ramesh* OpenAI aramesh@openai.com

Prafulla Dhariwal* OpenAI prafulla@openai.com

Alex Nichol* OpenAI alex@openai.com

Casey Chu* OpenAI casey@openai.com

Mark Chen OpenAI mark@openai.com

Abstract

Contrastive models like CLIP have been shown to learn robust representations of images that capture both semantics and style. To leverage these representations for image generation, we propose a two-stage model: a prior that generates a CLIP image embedding given a text caption, and a decoder that generates an image conditioned on the image embedding. We show that explicitly generating image representations improves image diversity with minimal loss in photorealism and caption similarity. Our decoders conditioned on image representations can also produce variations of an image that preserve both its semantics and style, while varying the non-essential details absent from the image representation. Moreover, the joint embedding space of CLIP enables language-guided image manipulations in a zero-shot fashion. We use diffusion models for the decoder and experiment with both autoregressive and diffusion models for the prior, finding that the latter are computationally more efficient and produce higher-quality samples.

1 Introduction

Recent progress in computer vision has been driven by scaling models on large datasets of captioned images collected from the internet [10, 44, 60, 39, 31, 16]. Within this framework, CLIP [39] has emerged as a successful representation learner for images. CLIP embeddings have a number of desirable properties: they are robust to image distribution shift, have impressive zero-shot capabilities, and have been fine-tuned to achieve state-of-the-art results on a wide variety of vision and language tasks [45]. Concurrently, diffusion models [46, 48, 25] have emerged as a promising generative modeling framework, pushing the state-of-the-art on image and video generation tasks [11, 26, 24]. To achieve best results, diffusion models leverage a guidance technique [11, 24] which improves sample fidelity (for images, photorealism) at the cost of sample diversity.

In this work, we combine these two approaches for the problem of text-conditional image generation. We first train a diffusion *decoder* to invert the CLIP image *encoder*. Our inverter is non-deterministic, and can produce multiple images corresponding to a given image embedding. The presence of an encoder and its approximate inverse (the decoder) allows for capabilities beyond text-to-image translation. As in GAN inversion [62, 55], encoding and decoding an input image produces semantically similar output images (Figure 3). We can also interpolate between input images by inverting interpolations of their image embeddings (Figure 4). However, one notable advantage of using the CLIP latent space is the ability to semantically modify images by moving in the direction of any encoded text vector (Figure 5), whereas discovering these directions in GAN latent space involves

*Equal contribution

vibrant portrait painting

an espresso machine that makes coffee from human souls, artstation

a close up of a handpalm with leaves growing from it

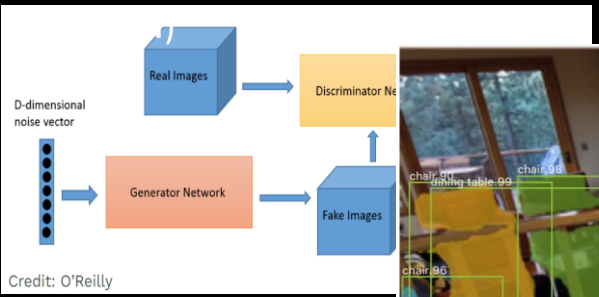
panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

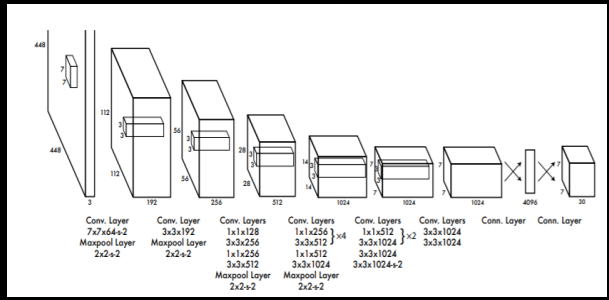
Architectures

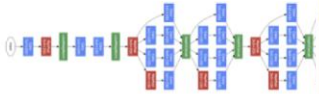
With the layers we have discussed, we can build countless different networks (and use TensorFlow to define them). Indeed, you may get the feel that the current "building block" is actually a functional network.

Generative Adversarial Network



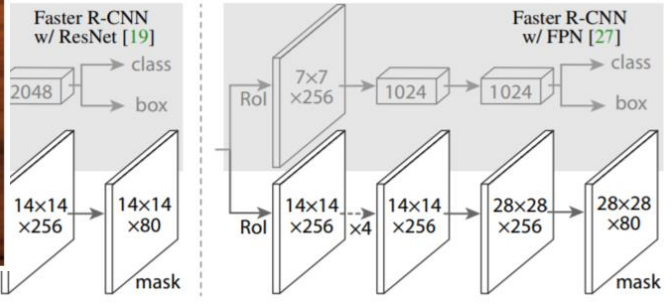
YOLO (You Only Look Once)





GoogLeNet / Inception

Convolution
Pooling
Softmax
Other



Mask R-CNN

Images from original papers

Some Taxonomies

So far we have focused on images, and their classification. You know that deep learning has had success across a wide, and rapidly expanding, number of domains. Even our digit recognition task could be more sophisticated:

- Classification (What we did)
- Localization (Where is the digit?)
- Detection (Are there digits? How many?)
- Segmentation (Which pixels are the digits?)

These tasks would call for different network designs. This is where our Day 3 would begin, and we would use some other building blocks.

As you learn more about machine learning, you will see various ways to categorize the algorithms or tasks or general approaches to doing something useful. Don't believe any of them are either comprehensive or canonical. They are just useful ways to keep track of the explosion of options in this space.

Tasks

Classification	What we've been doing.
Regression	Return a value. *Stock price.*
Transcription	Convert between representations. *OCR, speech recognition.*
Synthesis	Create new input examples. *Speech synthesizer. Lots of science these days!*
Translation	Like the word says. *Google Translate.*
Segmentation	Return a relabeled input vector. *Tumor detection*.
Denoising	Return uncorrupted example. *Video game ray tracing*.

Again, neither comprehensive nor definitive. The definitions vary from one author to the next, and the list grows all the time.

Learning Approaches

Supervised Learning

- How you learned colors.
- What we have been doing just now.
- Used for: image recognition, tumor identification, segmentation.
- Requires labeled data. Lots of it. Augmenting helps.
- Essence: Learning to map one vector to another, given enough examples of the mapping.

Fuzzy Line

Unsupervised Learning

- (Maybe) how you learned to see.
- What we did earlier with clustering and our recommender, and Deepfake.
- Find patterns in data, compress data into model, find reducible representation of data.
- Used for: Learning from unlabeled data.
- Might be a great way to bootstrap Supervised Learning (train an autoencoder and build from those weights).

Reinforcement Learning

- How you learned to walk.
- Requires goals (maybe long term, i.e. arbitrary delays between action and reward).
- Used for: Go (AlphaGo Zero), robot motion, video games.
- Don't just read data, but *interact* with it!

All of these have been done with and without deep learning. DL has moved to the forefront of all of these.

AI Based Simulation?

A wise man once (not that long ago) told me "John, I don't need a neural net to rediscover conservation of energy."

Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach

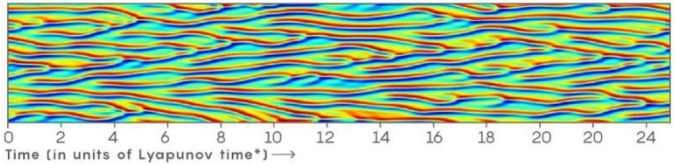
Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott
Phys. Rev. Lett. 120, 024102 – Published 12 January 2018

Training Computers to Tame Chaos

A machine-learning algorithm has been shown to accurately predict a chaotic system far further into the future than previously possible.

A Chaos Model

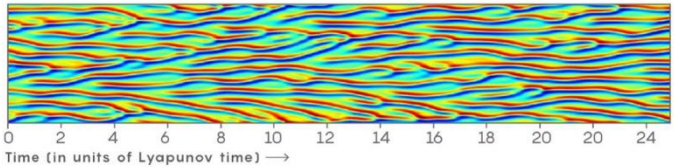
Researchers started with the evolving solution to the Kuramoto-Sivashinsky equation, which models propagating flames:



* Lyapunov time = Length of time before a small difference in the system's initial state begins to diverge exponentially. It typically sets the horizon of predictability, which varies from system to system.

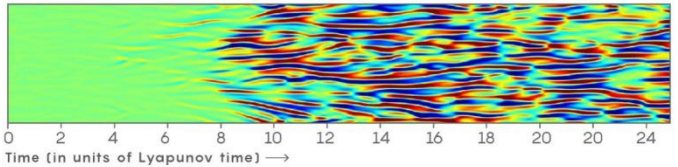
B Machine Learning

After training itself on data from the past evolution of the Kuramoto-Sivashinsky system, the "reservoir computing" algorithm predicts its future evolution:

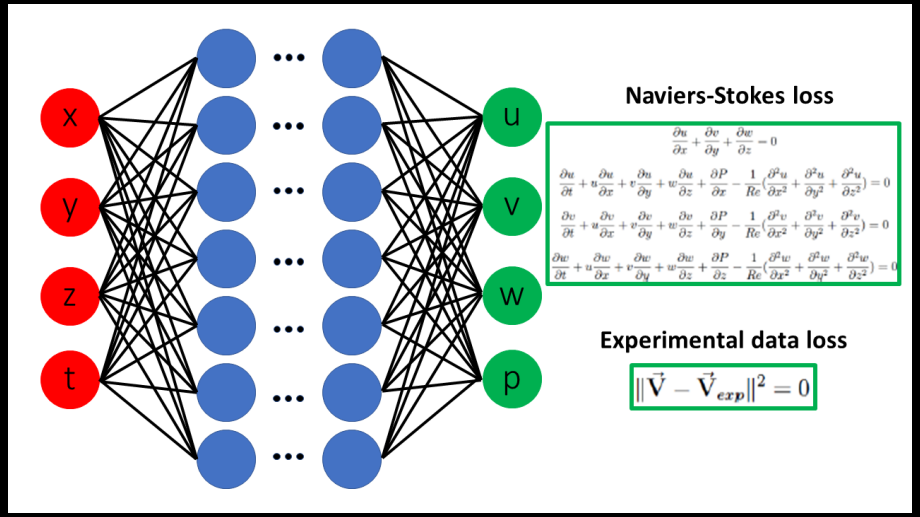


A–B Do They Match?

Subtracting B from A shows that the algorithm accurately predicts the model out to an impressive 8 Lyapunov times, before chaos ultimately prevails:



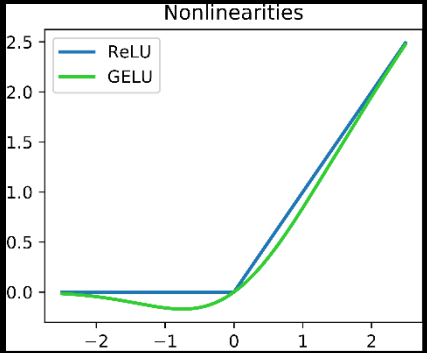
Physics Informed Neural Networks



Wikipedia

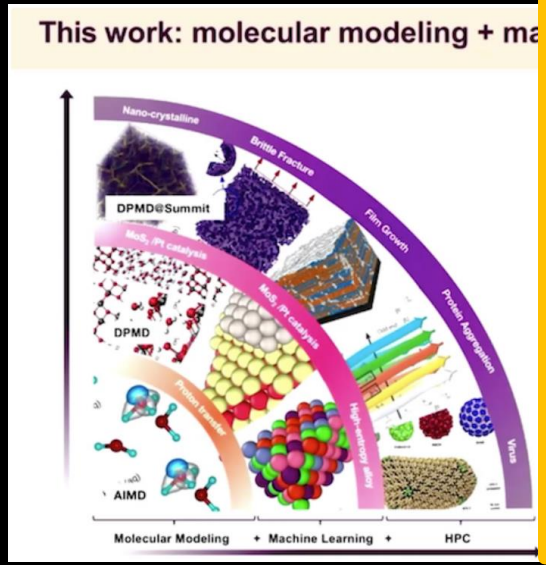
But maybe we can include our *a priori* knowledge. These types of networks (PINNs) are rapidly gaining interest in the world of physical modeling.

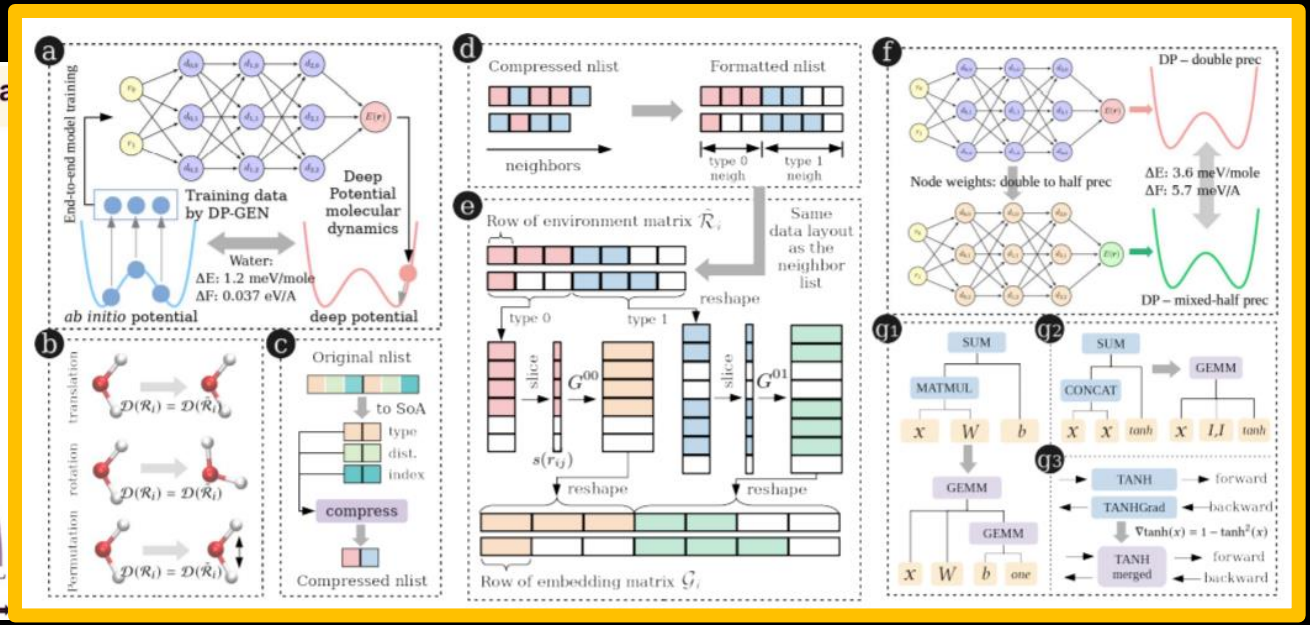
They are also no magic bullet. We now have two competing loss functions, for the data and the physics. And if the gradients in our network now have physical significance, we have to be more rigorous in our treatment of them. No ReLU activation functions, but instead something like the Gaussian Error Linear Unit (GELU).



AI Based Simulation Is Here To Stay

"We report that a machine learning-based simulation protocol (Deep Potential Molecular Dynamics), while retaining ab initio accuracy, can simulate more than 1 nanosecond-long trajectory of over 100 million atoms per day, using a highly optimized code (GPU DeePMD-kit) on the Summit supercomputer. Our code can efficiently scale up to the entire Summit supercomputer, attaining 91 PFLOPS in double precision (45.5% of the peak) and 162/275 PFLOPS in mixed-single/half precision.





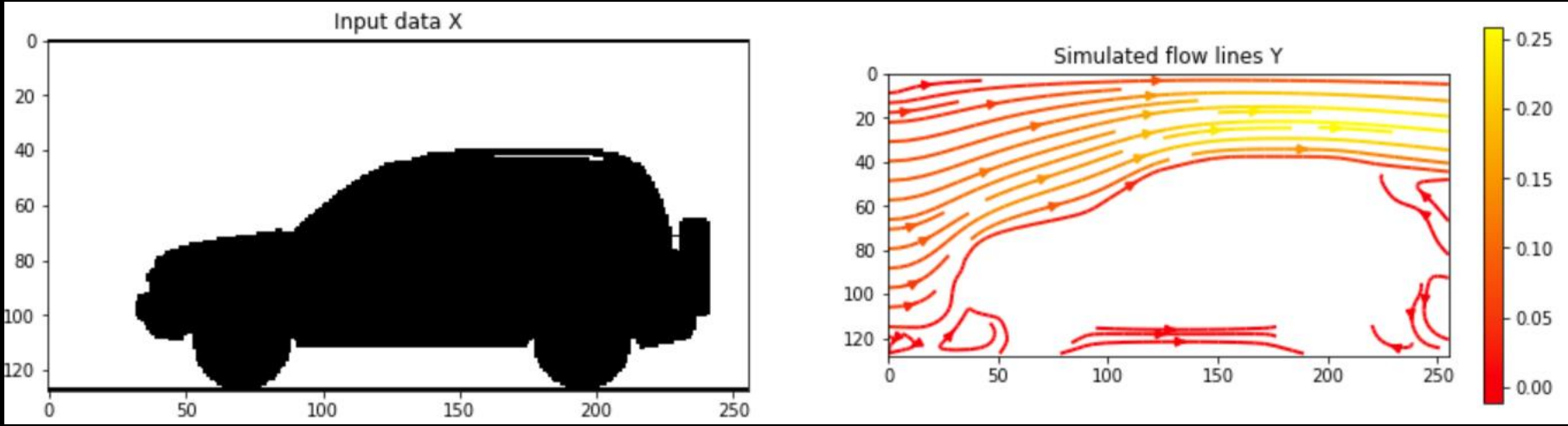
Pushing the Limit of Molecular Dynamics with Ab Initio Accuracy to 100 Million Atoms with Machine Learning

Weile Jia, Han Wang, Mohan Chen, Denghui Lu, Lin Lin, Roberto Car, Weinan E, Linfeng Zhang

2020 ACM Gordon Bell Prize Winner

Try It Yourself

NVIDIA's GPU Bootcamp materials contain a great example of this type of work. The premise is to learn a mapping from boundary conditions to steady state fluid flow. The tutorial works through several different models, starting with a Fully Connected Network, then using a CNN and finally introducing a more advance Residual Network approach. You should be able to jump right in with what we have learned here.



https://github.com/gpuhackathons-org/gpubootcamp/blob/78e9fee3432b60348489682a978fa63f29f7e839/hpc_ai/ai_science_cfd/English/python/jupyter_notebook/CFD/Start_Here.ipynb

From recent research paper to CMU physics undergraduate problem.

Newton vs. the machine

MNRAS **000**, 1–6 (2019) Preprint 17 October 2019 Compiled using MNRAS LATEX style file v3.0

Newton vs the machine: solving the chaotic three-body problem using deep neural networks

Philip G. Breen[1]*†, Christopher N. Foley[2] *‡, Tjarda Boekholt[3] and Simon Portegies Zwart[4]

[1]*School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, Kings Buildings, Edinburgh, EH9 3JZ*
[2]*MRC Biostatistics Unit, University of Cambridge, Cambridge, CB2 0SR, UK.*
[3]*Instituto de Telecomunicações, Campus Universitário de Santiago, 3810-193, Aveiro, Portugal*
[4]*Leiden Observatory, Leiden University, PO Box 9513, 2300 RA, Leiden, The Netherlands.*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

Since its formulation by Sir Isaac Newton, the problem of solving the equations of motion for three bodies under their own gravitational force has remained practically unsolved. Currently, the solution for a given initialization can only be found by performing laborious iterative calculations that have unpredictable and potentially infinite computational cost, due to the system's chaotic nature. We show that an ensemble of solutions obtained using an arbitrarily precise numerical integrator can be used to train a deep artificial neural network (ANN) that, over a bounded time interval, provides accurate solutions at fixed computational cost and up to 100 million times faster than a state-of-the-art solver. Our results provide evidence that, for computationally challenging regions of phase-space, a trained ANN can replace existing numerical solvers, enabling fast and scalable simulations of many-body systems to shed light on outstanding phenomena such as the formation of black-hole binary systems or the origin of the core collapse in dense star clusters.

Key words: stars: kinematics and dynamics, methods: numerical, statistical

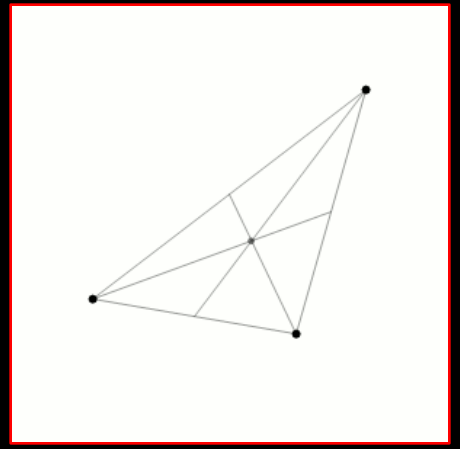
1 INTRODUCTION

Newton's equations of motion describe the evolution of many bodies in space under the influence of their own gravitational force (Newton 1687). The equations have a central role in many classical problems in Physics. For example, the equations explain the dynamical evolution of globular star clusters and galactic nuclei, which are thought to be the production sites of tight black-hole binaries that ultimately merge to produce gravitational waves (Portegies Zwart & McMillan 2000). The fate of these systems depends crucially on the three-body interactions between black-hole binaries and single black-holes (e.g. see Breen & Heggie 2013A,B; Samsing & D'Orazio 2018), often referred to as close encounters. These events typically occur over a fixed time interval and, owing to the tight interactions between the three nearby bodies, the background influence of the other bodies can be ignored, i.e. the trajectories of three bodies can be generally computed in isolation (Portegies Zwart & McMillan 2018).

The focus of the present study is therefore the timely computation of accurate solutions to the three-body problem.

Despite its age and interest from numerous distinguished scientists (de Lagrange 1772; Heggie 1975; Hut & Bahcall 1983; Montgomery 1998; Stone & Leigh 2019), the problem of solving the equations of motion for three-bodies remains impenetrable due to the system's chaotic nature (Valtonen et al 2016) which typically renders identification of solutions feasible only through laborious numerical integration. Analytic solutions exist for several special cases (de Lagrange 1772) and a solution to the problem for all time has been proposed (Valtonen et al 2016), but this is based on an infinite series expansion and has limited use in practice. Computation of a numerical solution, however, can require holding an exponentially growing number of decimal places in memory and using a time-step that approaches zero (Boekholt et al 2019). Integrators which do not allow for this often fail spectacularly, meaning that a single numerical solution is unreliable whereas the average of an ensemble of numerical solutions appear valid in a statistical sense, a concept referred to as nagh Hoch (Portegies Zwart & Boekholt 2018). To overcome these issues, the Brutus integrator was developed (Boekholt & Portegies Zwart 2015),

* Authors contributed equally
† Contact e-mail: phil.breen@ed.ac.uk
‡ Contact e-mail: christopher.foley@mrc-bsu.cam.ac.uk



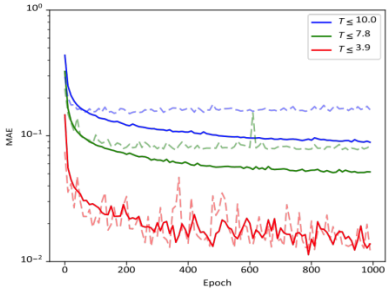


Figure 3. Mean Absolute Error (MAE) vs epoch. The ANN has the same training structure in each time interval. Solids lines are the loss on the training set and dashed are the loss on the validation set. $T \leq 3.9$ corresponds to 1000 labels per simulation, similarly $T \leq 7.8$ to 2000 labels and $T \leq 10.0$ to 2561 labels/time-points (the entire dataset).The results illustrate a typical occurrence in ANN training, there is an initial phase of rapid learning, e.g. âĹš100 epochs, followed by a stage of much slower learning in which relative prediction gains are smaller with each epoch.

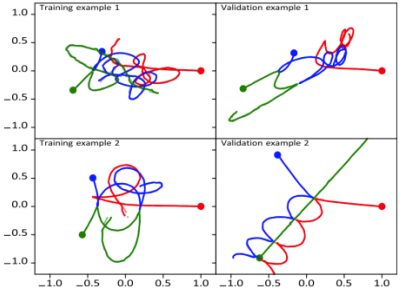


Figure 4. Validation of the trained ANN. Presented are two examples from the training set (left) and two from the validation set (right). All examples were randomly chosen from their datasets. The bullets indicate the initial conditions. The curves represent the orbits of the three bodies (red, blue and green, the latter obtained from symmetry). The solution from the trained network (solid curves) is hardly distinguishable from the converged solutions (dashes, acquired using Brutus (Boekholt & Portegies Zwart 2015)). The two scenarios presented to the right were not included in the training dataset.

Has Deep Learning left any room for other approaches?

Character Recognition
Capchas
Chess
Go

DL

AI

“Theoretician’s Nightmare” and Other Perspectives

The above is paraphrasing Yann LeCun, the godfather of Deep Learning.

If it feels like this is an oddly empirical branch of computer science, you are spot on.

Many of these techniques were developed through experimentation, and many of them are not amenable to classical analysis. A theoretician would suggest that non-convex loss functions are at the heart of the matter, and that situation isn’t getting better, as many of the latest techniques have made this much worse.

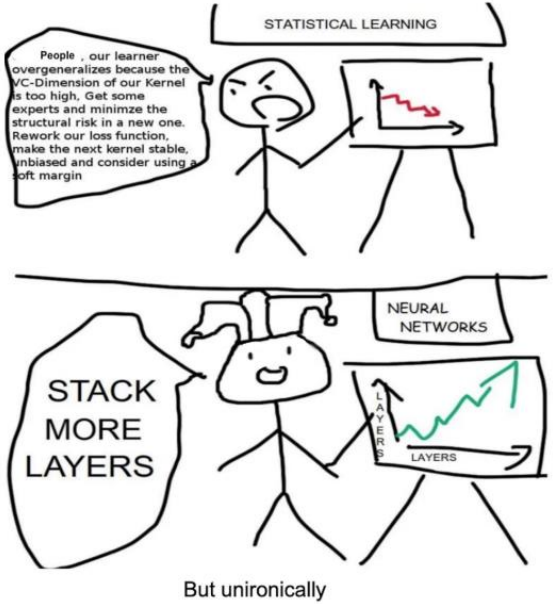
You may also have noticed that many of the techniques we have used today have very recent provenance. This is true throughout the field. Rarely is the undergraduate researcher so reliant upon groundbreaking papers of a few years ago.

The previously mentioned Christopher Olah has this rather useful summation: *"People sometimes complain: 'Neural networks are so hard to understand! Why can’t we use understandable models, like SVMs?' Well, you understand SVMs, and you don’t understand visual pattern recognition. If SVMs could solve visual pattern recognition, you would understand it. Therefore, SVMs are not capable of this, nor is any other model you can really understand."*

My own humble observation: Deep Learning looks a lot like late 19th century chemistry. There is a weak theoretical basis, but significant experimental breakthroughs of great utility. The lesson from that era was "expect a lot more perspiration than inspiration."

Most meta.
With some advice for you!

User Can you explain why this is funny. Think about it step-by-step.



But unironically

GPT-4 The comic is satirizing the difference in approaches to improving model performance between statistical learning and neural networks.

In statistical learning, the character is shown to be concerned with overfitting and suggests a series of complex and technical solutions, such as minimizing structural risk, reworking the loss function, and using a soft margin.

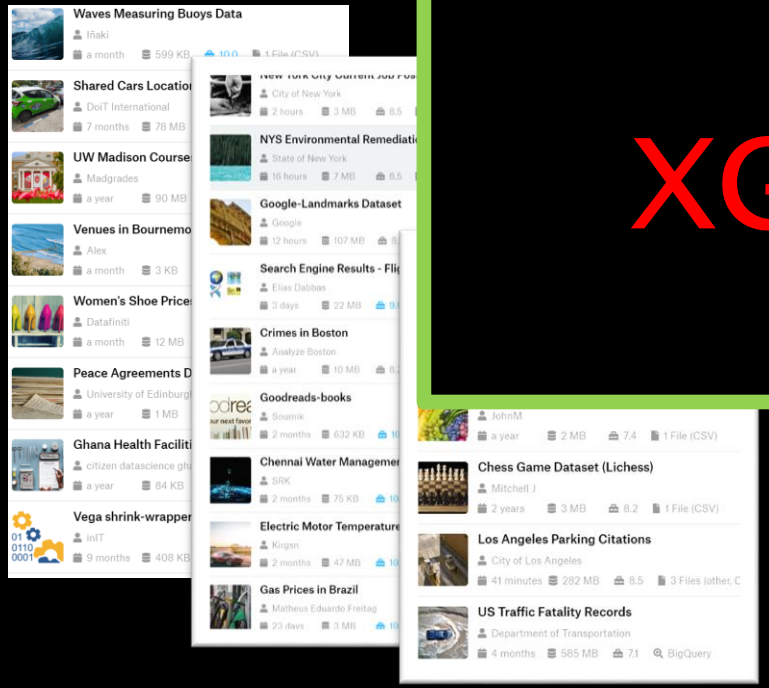
In contrast, the neural networks character simply suggests adding more layers to the model. This is often seen as a common solution to improving performance in neural networks, but it's also considered a simplistic and brute-force approach.

The humor comes from the contrast between the complexity and specificity of the statistical learning approach and the simplicity and generality of the neural network approach. The "But unironically" comment adds to the humor by implying that, despite being simplistic, the "stack more layers" approach is often effective in practice.

Lazy Scientist's Survey of the Field

Kaggle Challenge

The benchmark driven nature of deep learning research, and its competitive consequences, have found a nexus at Kaggle.com. There you can find over 20,000 datasets:



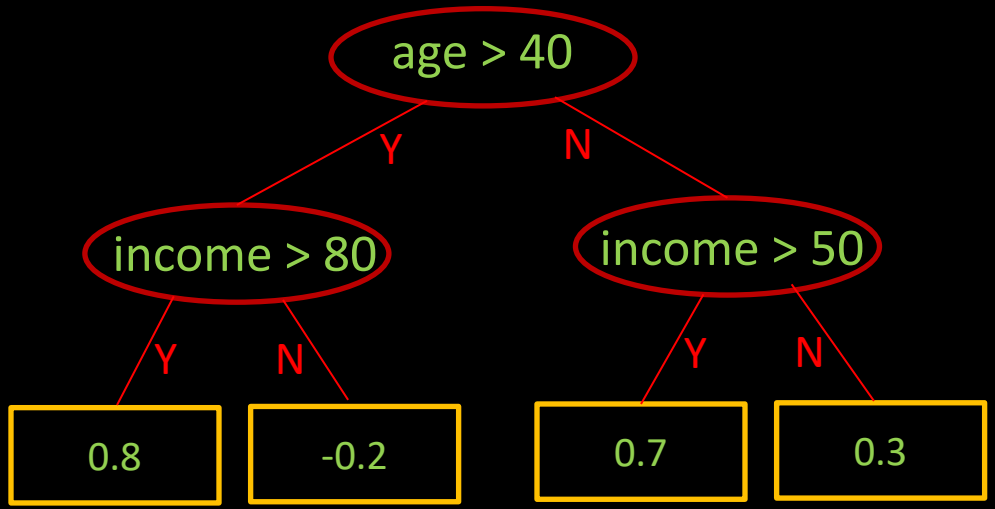
XGBoost?

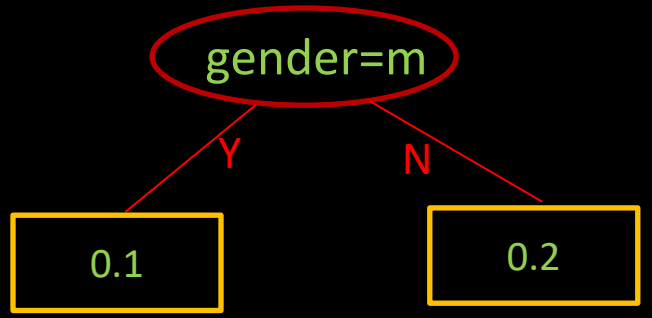
and competitions:

Trees

(How much of our earlier learning can we apply here?)

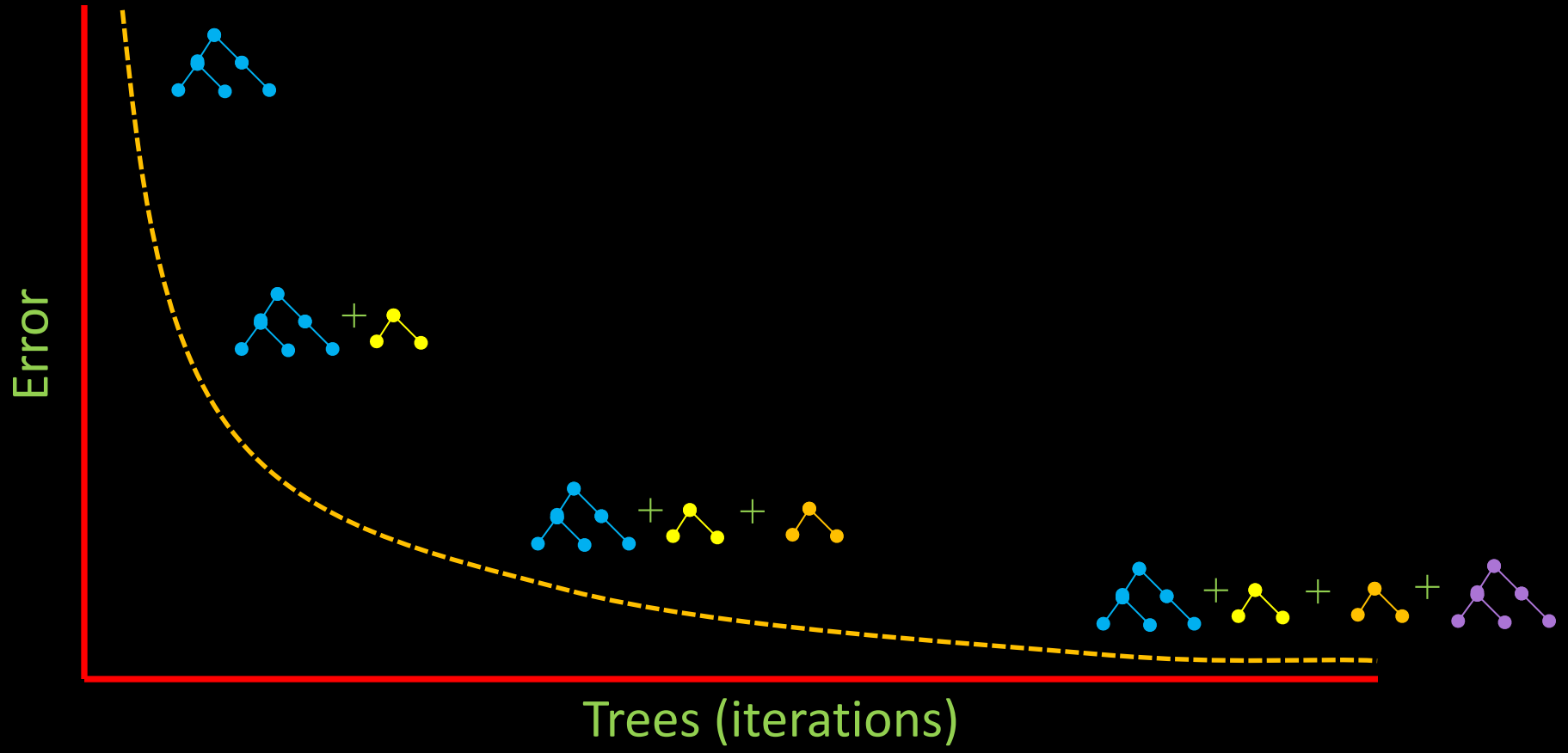
XGBoost is the latest, and most popular, evolution of the Decision Tree approach. Let's say we want to predict is some given person is likely to be a buyer of a certain car model:



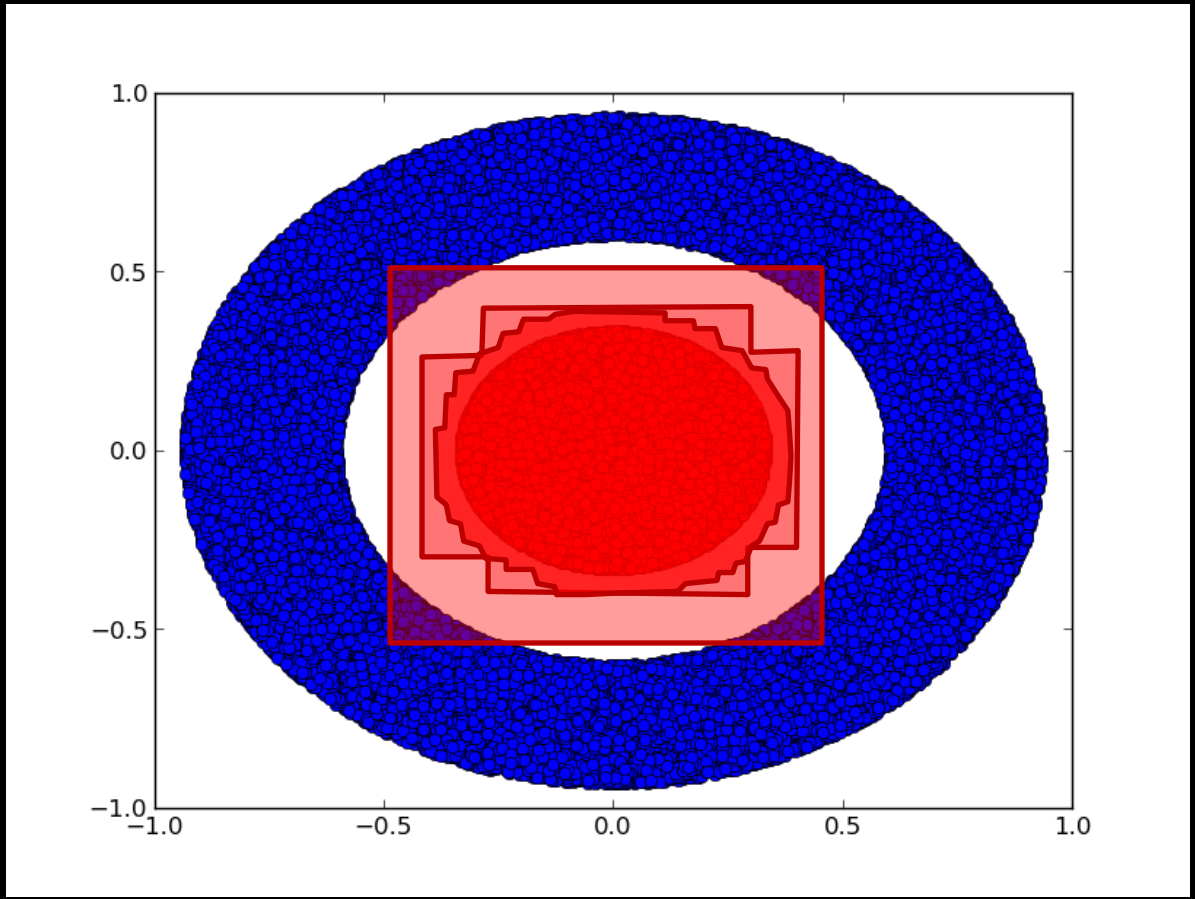


Trees are desirable in that they are non-linear, but still analytically tractable, and can do both regression and classification.

Gradient Boosted Trees



Remember This?



XGBoost

A very cool interactive application to explore these concepts, and try various hyperparameters, was done by Alex Rogozhnikov and can be found at:

http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html

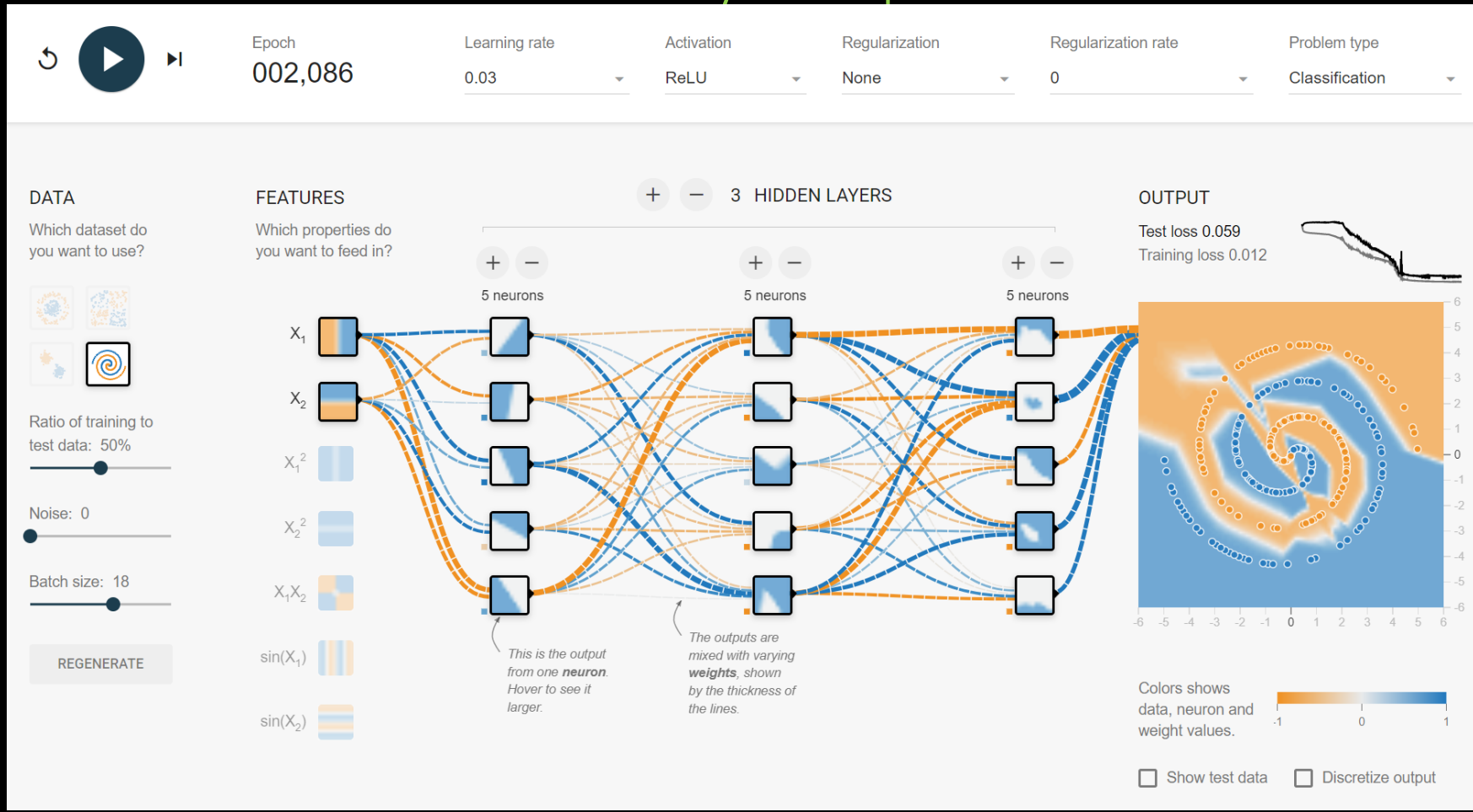
If you want to understand XGBoost in detail, you can find the original paper at:

https://arxiv.org/pdf/1603.02754.pdf

An in-depth, but still beginner-friendly, video from StatsQuest can be found at:

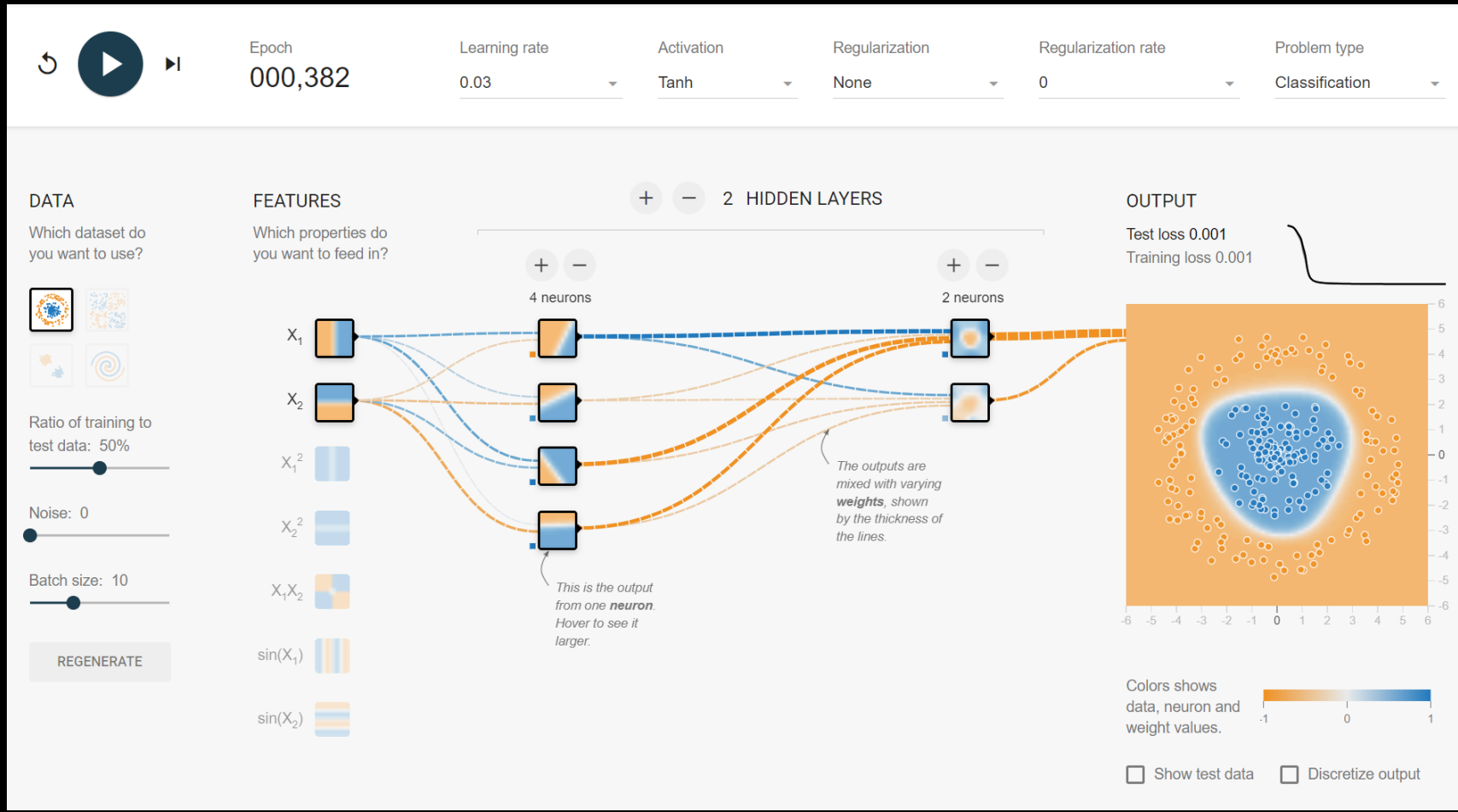
https://www.youtube.com/watch?v=GrJP9FLV3FE

Great Way To Compare



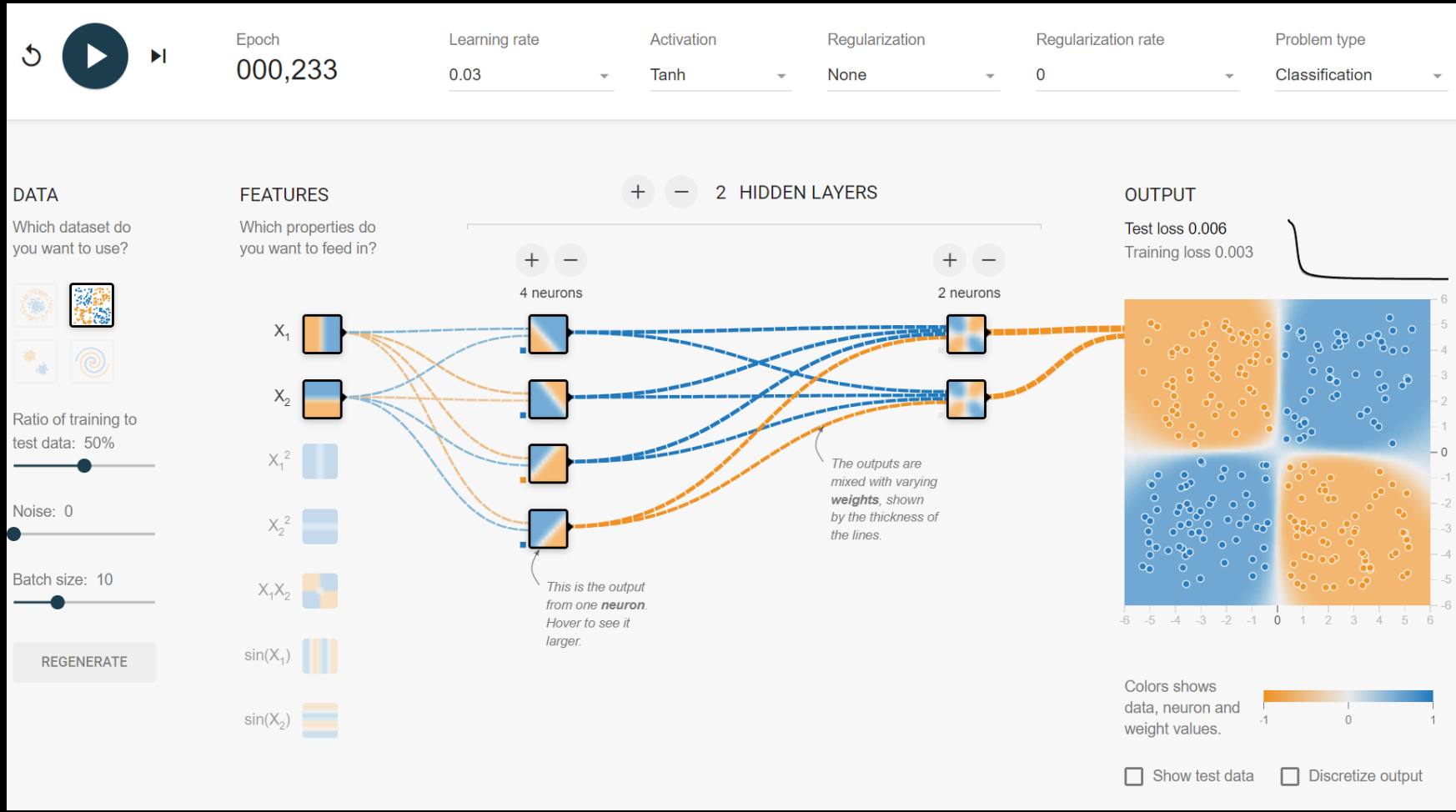
https://playground.tensorflow.org/

Great Way To Compare



https://playground.tensorflow.org/

The Classic Conundrum



https://playground.tensorflow.org/

XGBoost in Particular

There are various implementations of gradient boosted trees. XGBoost combines several important innovations:

- Parallelizes well both across cores and nodes
- Clever cache optimization
- Works well with missing data

The end result is an efficient algorithm that works well enough with non-optimal hyperparameters the beginners can often make quick progress.

The scikit-learn version is probably the most popular, but there is a Spark version (https://xgboost.readthedocs.io/en/latest/jvm/xgboost4j_spark_tutorial.html), and if you want a deeper dive, NDVIDIA has this pretty nice taxi fare regression model that uses GPUs with Spark and does a hyperparameter search. Note that I have not tried these myself:

https://developer.nvidia.com/blog/accelerating-spark-3-0-and-xgboost-end-to-end-training-and-hyperparameter-tuning/

TensorFlow has a boosted tree API along with a nice walkthrough example in the docs:

https://www.tensorflow.org/tutorials/estimator/boosted_trees

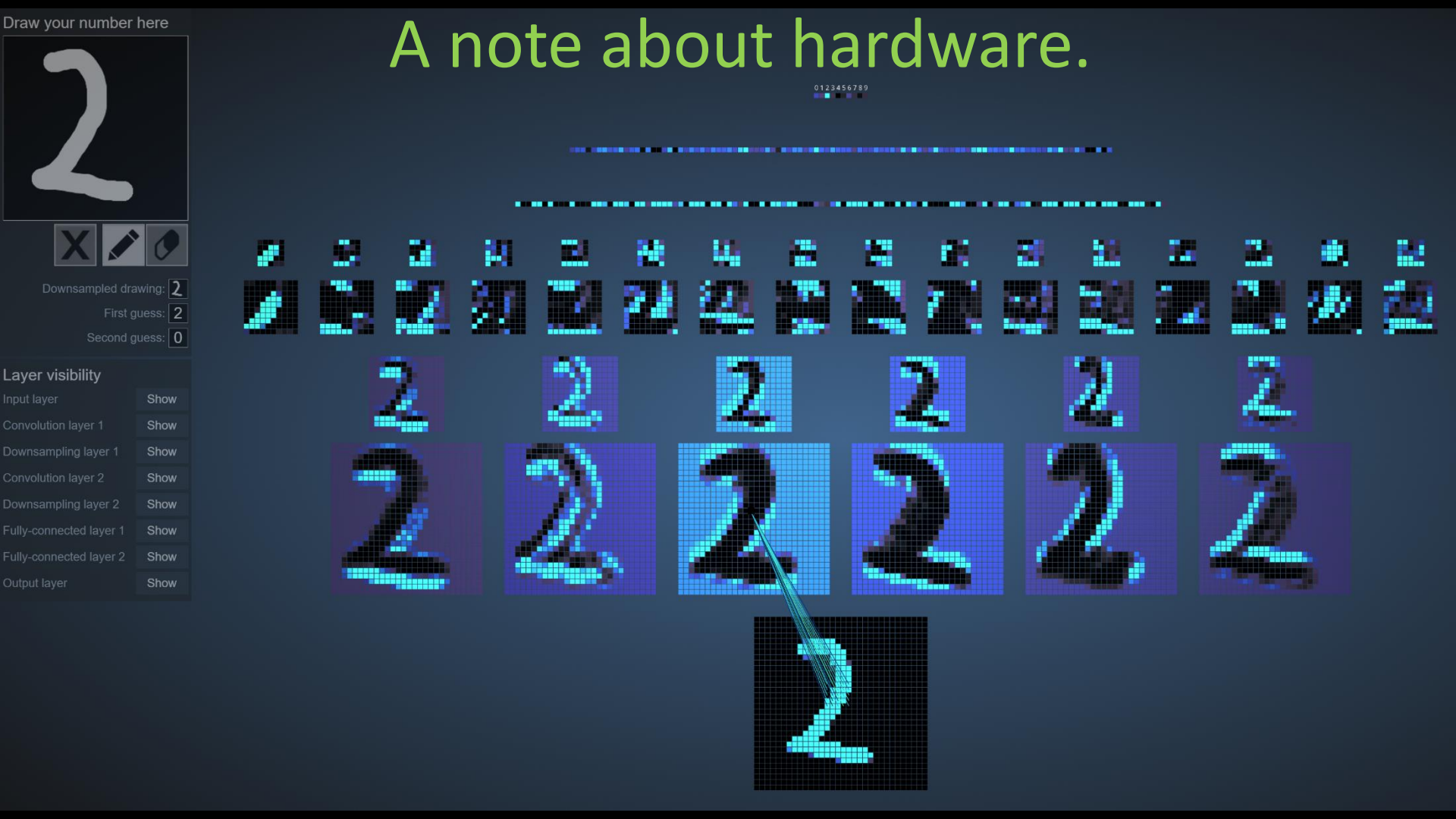
However, note that this is not the XGBoost version (yet).

Other Toolboxes

You have a plethora of alternatives available as well. You are now in a position to appreciate some comparisons.

Package	Applications	Language	Strengths
TensorFlow	Neural Nets	Python, C++	Very popular.
PyTorch	Neural Nets	Python (Lua)	Also very popular. Used to be very different with it's dynamic graphs and eager execution, but lacked simple layers. Now fairly similar in approach.
Spark MLLIB	Classification, Regression, Clustering, etc.	Python, Scala, Java, R	Very scalable. Widely used in serious applications. Lots of plugins to DL frameworks: TensorFrames, TF on Spark, CaffeOnSpark, Keras Elephas.
Scikit-Learn	Classification, Regression, Clustering	Python	Integrates well with TF to create powerful workflows.
Keras	Neural Nets	Python (on top of TF, Theano)	Now completely absorbed into TF.
Jax	Neural Nets	Python	Latest DeepMind (part of Google) framework. Missing pieces, but getting there. Similar to TF & PT.

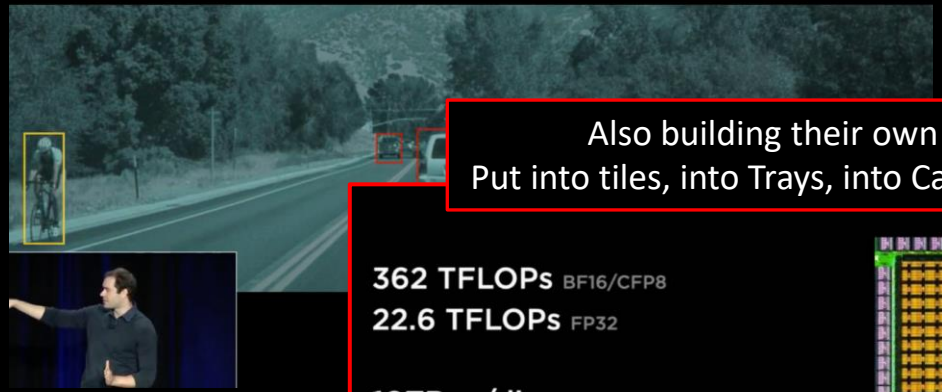
A note about hardware.



Inference Is Fast

Perceptual Labs

iPhone Demo

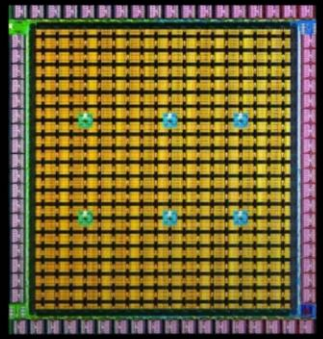


Also building their own training chips.
Put into tiles, into Trays, into Cabinets to create Dojo.

362 TFLOPs BF16/CFP8
22.6 TFLOPs FP32

10TBps/dir. On-Chip Bandwidth
4TBps/edge. Off-Chip Bandwidth

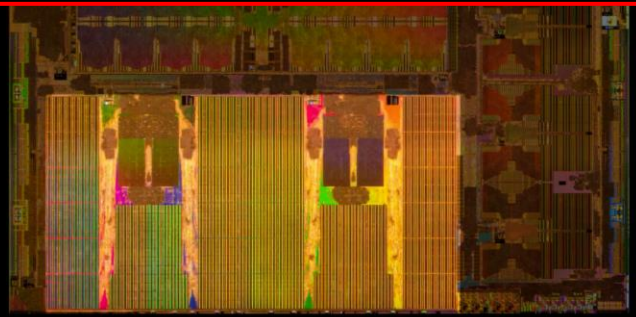
400W TDP



645mm²
7nm Technology

50 Billion
Transistors

11+ Miles
Of Wires



32MB SRAM

96x96 Mul/Add array

ReLU hardware

Pooling hardware

36 TOPS @ 2 GHz

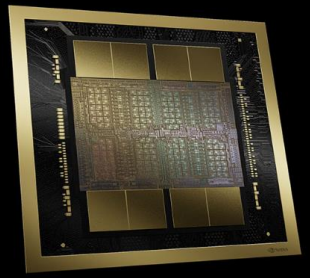
2 per chip, 72 TOPS total

TESLA LIVE

Everyone Doing Specialized Hardware

NVIDIA

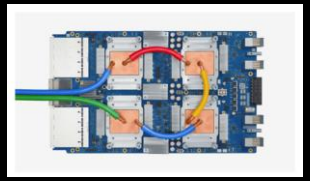
Blackwell



Highlighting 4 bit precision!

Google

TPU



Cloud TPU v6
1.8 exaflops (int 8)
purported...
Due 2024Q4

Intel

Loihi 2

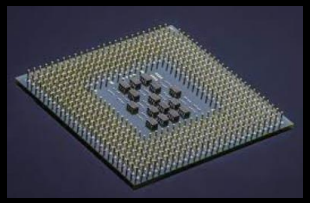
1,048,576 artificial neurons, and 120 million synapses

Kapoho Point a 4-inch form factor board featuring eight Loihi 2 chips

Also new AVX512_VNNI (Vector Neural Network) instructions like an FMA instruction for 8-bit multiplies with 32-bit accumulates on new processors.

Amazon

Inferentia2
Trainium



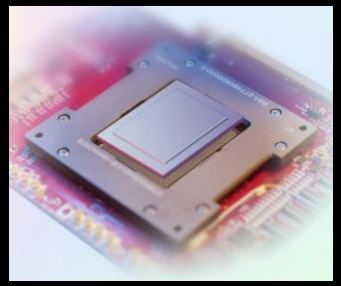
Inferentia for inference and *Trainium* for training. Use standard TensorFlow and Torch in their EC2 Cloud.

This is where Alexa runs.

Everyone Doing Specialized Hardware

Meta

MTIA 2



708 TFLOPS/s (INT8)

Microsoft

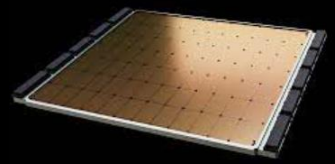
Athena

"Launched" in late 2023, but info still sparse....

Cerebras

CS-2

PSC



At PSC!

- 850,000 Sparse Linear Algebra Compute Cores
- 2.6 trillion transistors
- 20 PB/s aggregate memory bandwidth
- 220 Pb/s interconnect bandwidth

Neuromorphic

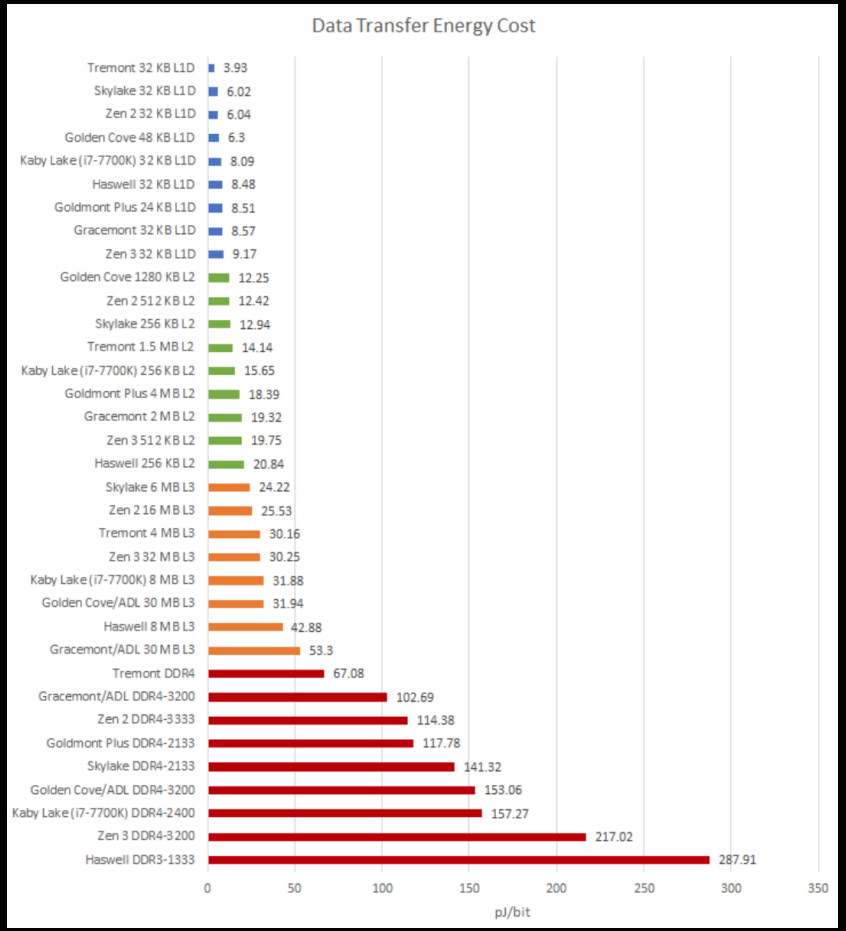
IBM, ...

Brain only uses 20W.

Analog, pruning, spiking, lots of new directions.

We are also continuously learning how little we know about how biological mechanisms work.

More Inspiration



- Laughlin was the first to provide explicit quantities for the energetic cost of processing sensory information. Their findings in blowflies revealed that for visual sensory data, the cost of transmitting one bit of information is around 50 fJ (5×10^{-14} Joules), or equivalently 104 ATP molecules.
- The units on this graph are pJ, 1000X larger. Thus, neural processing efficiency is still far from Landauer's limit of kTln(2) J, but still considerably more efficient than a modern computer's near memory. For far (MPI network, or further) accesses it is a huge difference.

Applied Inspiration

Scalable MatMul-free Language Modeling

Rui-Jie Zhu[1], Yu Zhang[2], Ethan Sifferman[1], Tyler Sheaves[3], Yiqiao Wang[4], Dustin Richmond[1], Peng Zhou[1,4], Jason K. Eshraghian[1*]

[1]University of California, Santa Cruz [2]Soochow University
[3]University of California, Davis [4]LuxiTech

Abstract

Matrix multiplication (MatMul) typically dominates the overall computational cost of large language models (LLMs). This cost only grows as LLMs scale to larger embedding dimensions and context lengths. In this work, we show that MatMul operations can be completely eliminated from LLMs while maintaining strong performance at billion-parameter scales. Our experiments show that our proposed MatMul-free models achieve performance on-par with state-of-the-art Transformers that require far more memory during inference at a scale up to at least 2.7B parameters. We investigate the scaling laws and find that the performance gap between our MatMul-free models and full precision Transformers narrows as the model size increases. We also provide a GPU-efficient implementation of this model which reduces memory usage by up to 61% over an unoptimized baseline during training. By utilizing an optimized kernel during inference, our model's memory consumption can be reduced by more than 10× compared to unoptimized models. To properly quantify the efficiency of our architecture, we build a custom hardware solution on an FPGA which exploits lightweight operations beyond what GPUs are capable of. We processed billion-parameter scale models at 13W beyond human readable throughput, moving LLMs closer to brain-like efficiency. This work not only shows how far LLMs can be stripped back while still performing effectively, but also points at the types of operations future accelerators should be optimized for in processing the next generation of lightweight LLMs. Our code implementation is available at https://github.com/ridgerchu/matmulfreellm.

1 Introduction

Matrix Multiplication (MatMul) is the dominant operation in most neural networks, where dense layers involve vector-matrix multiplication (VMM), convolutions can be implemented as block-sparse VMMs with shared weights, and self-attention relies on matrix-matrix multiplication (MMM). The prevalence of MatMul is primarily due to Graphics Processing Units (GPUs) being optimized for MatMul operations. By leveraging Compute Unified Device Architecture (CUDA) and highly optimized linear algebra libraries such as cuBLAS, the MatMul operation can be efficiently parallelized and accelerated. This optimization was a key factor in the victory of AlexNet in the ILSVRC2012 competition and a historic marker for the rise of deep learning [1]. AlexNet notably utilized GPUs to boost training speed beyond CPU capabilities, and as such, deep learning won the 'hardware lottery' [2]. It also helped that both training and inference rely on MatMul.

Despite its prevalence in deep learning, MatMul operations account for the dominant portion of computational expense, often consuming the majority of the execution time and memory access during

*Corresponding author, email to: pzhou10@ucsc.edu, jsn@ucsc.edu

The secret sauce is ternery logic, with values of +1, 0 and -1.

The matrix multiplies degenerate into some simpler operations.

This is the may be the finish line of the race to the bottom of precision.

Binary doesn't seem to capture the behaviors we hope for (although it is early days), but ternary does.

This is a bit awkward for our currently binary electronics.

Read, read, read!

One of our major goals is to leave you with the ability to understand many of the latest publications in *applied, scientific* AI.

Of course, 2 days is not enough for you to become an expert, but you might be surprised how much of the literature you can understand. You should be well-positioned to fill in the gaps.

If you want to test your knowledge, this "state of the field" lecture by the foremost pioneers of deep learning is an excellent summation of the current leading edge. It is targeted at practitioners of the art, so don't feel intimidated by any unknown references. But if you do get the gist of it, congratulations, you are holding your own with current researchers.

https://dl.acm.org/doi/pdf/10.1145/3448250

turing lecture

DOI:10.1145/3448250

How can neural networks learn the rich internal representations required for difficult tasks such as recognizing objects or understanding language?

BY YOSHUA BENGIO, YANN LECUN, AND GEOFFREY HINTON

Deep Learning for AI

TURING LECTURE

Yoshua Bengio, Yann LeCun, and Geoffrey Hinton are recipients of the 2018 ACM A.M. Turing Award for breakthroughs that have made deep neural networks a critical component of computing.

RESEARCH ON ARTIFICIAL neural networks was motivated by the observation that human intelligence emerges from highly parallel networks of relatively simple, non-linear neurons that learn by adjusting the strengths of their connections. This observation leads to a central computational question: How is it possible for networks of this general kind to learn the complicated internal representations that are required for difficult tasks such as recognizing objects or understanding language? Deep learning seeks to answer this question by using many layers of activity vectors as representations and learning the connection strengths that give rise to these vectors by following the stochastic gradient of an objective function that measures how well the network is performing. It is very surprising that such a conceptually simple approach has proved to be so effective when applied to large training sets using huge amounts of computation and it appears that a key ingredient is depth: shallow networks simply do not work as well.

We reviewed the basic concepts and some of the breakthrough achievements of deep learning several years ago.[63] Here we briefly describe the origins of deep learning, describe a few of the more recent advances, and discuss some of the future challenges. These challenges include learning with little or no external supervision, coping with test examples that come from a different distribution than the training examples, and using the deep learning approach for tasks that humans solve by using a deliberate sequence of steps which we attend to consciously—tasks that Kahneman[56] calls *system 2* tasks as opposed to *system 1* tasks like object recognition or immediate natural language understanding, which generally feel effortless.

From Hand-Coded Symbolic Expressions to Learned Distributed Representations

There are two quite different paradigms for AI. Put simply, the logic-inspired paradigm views sequential reasoning as the essence of intelligence and aims to implement reasoning in computers using hand-designed rules of inference that operate on hand-designed symbolic expressions that formalize knowledge. The brain-inspired paradigm views learning representations from data as the essence of intelligence and aims to implement learning by hand-designing or evolving rules for modifying the connec-

Demos

Ray-traced videogames! Recurrent CNN.

http://research.nvidia.com/sites/default/files/publications/dnn_denoise_author.pdf


A little more about
GPT-4

A little GPT-4

- 99 pages!
- "Given both the competitive landscape and the safety implications of large-scale models like GPT-4, this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar."

GPT-4 Technical Report

OpenAI*

Abstract

We report the development of GPT-4, a large-scale, multimodal model which can accept image and text inputs and produce text outputs. While less capable than humans in many real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-training alignment process results in improved performance on measures of factuality and adherence to desired behavior. A core component of this project was developing infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to accurately predict some aspects of GPT-4's performance based on models trained with no more than 1/1,000th the compute of GPT-4.

1 Introduction

This technical report presents GPT-4, a large multimodal model capable of processing image and text inputs and producing text outputs. Such models are an important area of study as they have the potential to be used in a wide range of applications, such as dialogue systems, text summarization, and machine translation. As such, they have been the subject of substantial interest and progress in recent years [1–34].

One of the main goals of developing such models is to improve their ability to understand and generate natural language text, particularly in more complex and nuanced scenarios. To test its capabilities in such scenarios, GPT-4 was evaluated on a variety of exams originally designed for humans. In these evaluations it performs quite well and often outscores the vast majority of human test takers. For example, on a simulated bar exam, GPT-4 achieves a score that falls in the top 10% of test takers. This contrasts with GPT-3.5, which scores in the bottom 10%.

On a suite of traditional NLP benchmarks, GPT-4 outperforms both previous large language models and most state-of-the-art systems (which often have benchmark-specific training or hand-engineering). On the MMLU benchmark [35, 36], an English-language suite of multiple-choice questions covering 57 subjects, GPT-4 not only outperforms existing models by a considerable margin in English, but also demonstrates strong performance in other languages. On translated variants of MMLU, GPT-4 surpasses the English-language state-of-the-art in 24 of 26 languages considered. We discuss these model capability results, as well as model safety improvements and results, in more detail in later sections.

This report also discusses a key challenge of the project, developing deep learning infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to make predictions about the expected performance of GPT-4 (based on small runs trained in similar ways) that were tested against the final run to increase confidence in our training.

Despite its capabilities, GPT-4 has similar limitations to earlier GPT models [1, 37, 38]: it is not fully reliable (e.g. can suffer from "hallucinations"), has a limited context window, and does not learn

*Please cite this work as "OpenAI (2023)". Full authorship contribution statements appear at the end of the document.

GPT-4 isn't cheap. This is the trend.

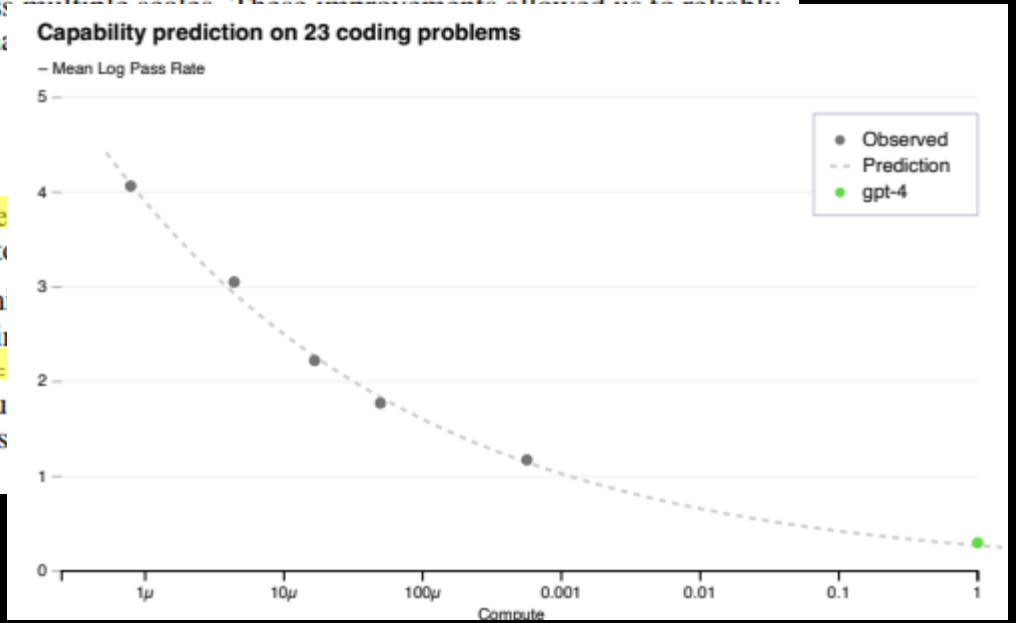
3 Predictable Scaling

A large focus of the GPT-4 project was building a deep learning stack that scales predictably. The primary reason is that for very large training runs like GPT-4, it is not feasible to do extensive model-specific tuning. To address this, we developed infrastructure and optimization methods that have very predictable behavior across multiple scales. These improvements allowed us to reliably predict some aspects of the performance of GPT-4 from smaller models trained using $10,000\times$ less compute.

3.1 Loss Prediction

The final loss of properly-trained large language models is thought to follow smooth power laws in the amount of compute used to train them.

To verify the scalability of our optimization infrastructure, we predicted GPT-4's final loss on our internal codebase (not part of the training set) by fitting a scaling law with an irreducible loss term (as in Henighan et al. [15]): $L(C) = aC^b + c$, from models trained using the same methodology but using at most 10,000x less compute than GPT-4. This prediction was made shortly after the run started, without use of any partial results. The fitted scaling law predicted GPT-4's final loss with high accuracy (Figure 1).



What can't it do?

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.

Now with vision and spooky reasoning!

User What is funny about this image? Describe it panel by panel.

Source: https://www.reddit.com/r/hmmm/comments/ubab5v/hmmm/

GPT-4 The image shows a package for a "Lightning Cable" adapter with three panels.

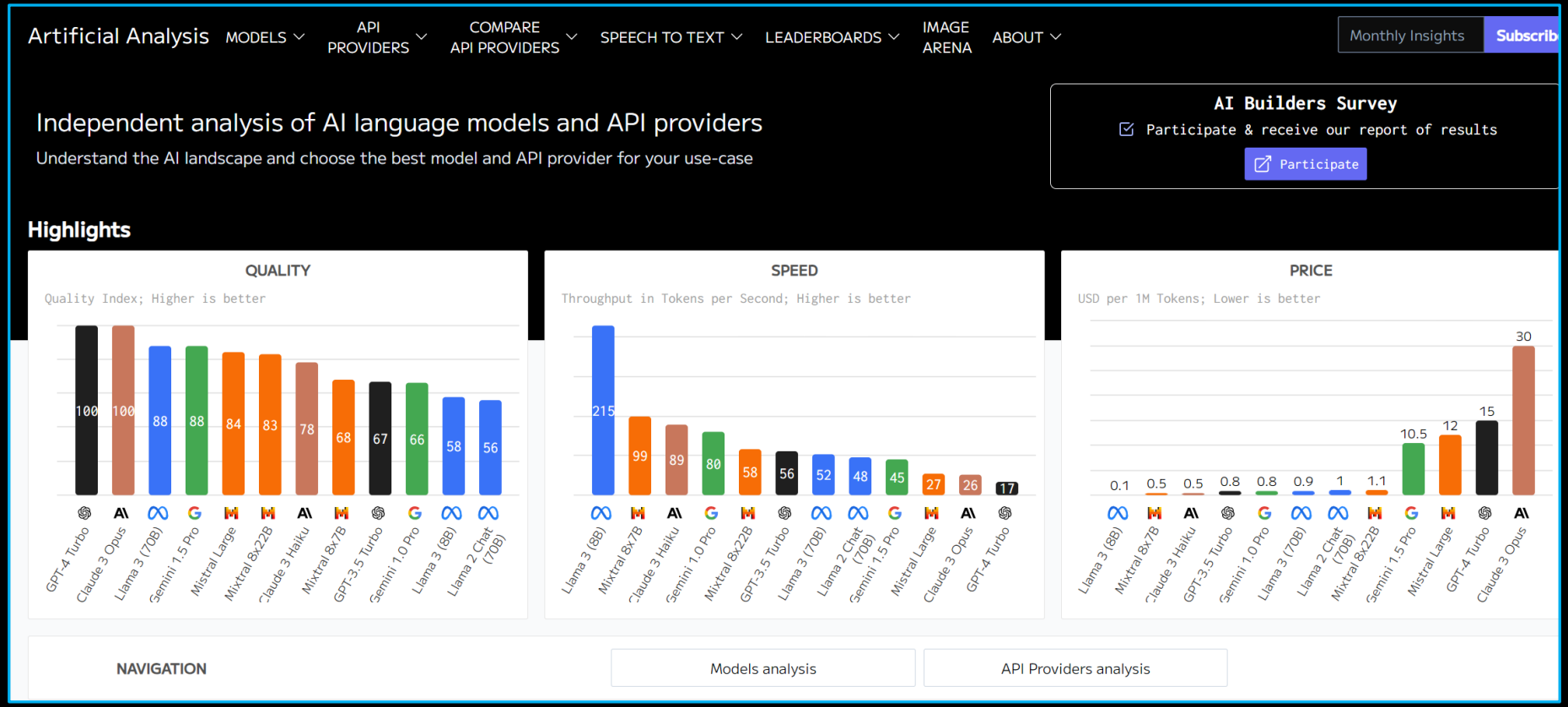
Panel 1: A smartphone with a VGA connector (a large, blue, 15-pin connector typically used for computer monitors) plugged into its charging port.

Panel 2: The package for the "Lightning Cable" adapter with a picture of a VGA connector on it.

Panel 3: A close-up of the VGA connector with a small Lightning connector (used for charging iPhones and other Apple devices) at the end.

The humor in this image comes from the absurdity of plugging a large, outdated VGA connector into a small, modern smartphone charging port.

Plenty of competition.



From artificialanalysis.ai


Why you really care...
Can this code?

From this different paper.

Sparks of Artificial General Intelligence: Early experiments with GPT-4

Sébastien Bubeck Varun Chandrasekaran Ronen Eldan Johannes Gehrke
Eric Horvitz Ece Kamar Peter Lee Yin Tat Lee Yuanzhi Li Scott Lundberg
Harsha Nori Hamid Palangi Marco Tulio Ribeiro Yi Zhang

Microsoft Research

Abstract

Artificial intelligence (AI) researchers have been developing and refining large language models (LLMs) that exhibit remarkable capabilities across a variety of domains and tasks, challenging our understanding of learning and cognition. The latest model developed by OpenAI, GPT-4 [Ope23], was trained using an unprecedented scale of compute and data. In this paper, we report on our investigation of an early version of GPT-4, when it was still in active development by OpenAI. We contend that (this early version of) GPT-4 is part of a new cohort of LLMs (along with ChatGPT and Google's PaLM for example) that exhibit more general intelligence than previous AI models. We discuss the rising capabilities and implications of these models. We demonstrate that, beyond its mastery of language, GPT-4 can solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, psychology and more, without needing any special prompting. Moreover, in all of these tasks, GPT-4's performance is strikingly close to human-level performance, and often vastly surpasses prior models such as ChatGPT. Given the breadth and depth of GPT-4's capabilities, we believe that it could reasonably be viewed as an early (yet still incomplete) version of an artificial general intelligence (AGI) system. In our exploration of GPT-4, we put special emphasis on discovering its limitations, and we discuss the challenges ahead for advancing towards deeper and more comprehensive versions of AGI, including the possible need for pursuing a new paradigm that moves beyond next-word prediction. We conclude with reflections on societal influences of the recent technological leap and future research directions.

Contents

Impressive results on standard exam benchmarks.

Note how they *test* against unseen examples.

3.1.1 Coding challenges

A common way to measure coding skill is to pose coding challenges that require implementing a specific functionality or algorithm. We first benchmark GPT-4 on HumanEval [CTJ+21], a docstring-to-code dataset consisting of 164 coding problems that test various aspects of programming logic and proficiency. As shown in Table 1, GPT-4 outperforms other LLMs, including `text-davinci-003` (the base model of ChatGPT) and other models trained specifically on code, `code-davinci-002`, and CODEGEN-16B [NPH+22].

Model	**GPT-4**	`text-davinci-003`	Codex(`code-davinci-002`)	CODEGEN-16B
Accuracy	**82%**	65%	39%	30%

Table 1: Zero-shot pass@1 accuracy comparison of different models on HumanEval

Although GPT-4's accuracy shows a big jump compared to previous models, it could be that GPT-4 has seen and memorized some (or all) of HumanEval during pre-training. To account for this possibility, we also evaluate it on LeetCode (`https://leetcode.com`), a popular platform for software engineering interviews, where new problems are constantly posted and updated. We used LeetCode in Figure 1.5 in the introduction, where GPT-4 passes all stages of mock interviews for major tech companies. Here, to test on *fresh* questions, we construct a benchmark of 100 LeetCode problems posted after October 8th, 2022, which is after GPT-4's pretraining period. As seen in the example in Figure 3.1, we paste the problem instructions into a prompt, ask GPT-4 to write a python function, and use the official LeetCode online judge to check for correctness. We present the results in Table 2, where we compare GPT-4 to other models and to human performance based on LeetCode contest results (users who fail all questions are not included, and thus this is a strong sample of humans). We report both pass@1 and pass@5 accuracies, which measure whether the model produces a correct solution in the first or in the first five attempts, respectively. GPT-4 significantly outperforms the other models, and is comparable to human performance (which we measure in Appendix C.1).

	Easy		Median		Hard		Overall	
pass@k	$k=1$	$k=5$	$k=1$	$k=5$	$k=1$	$k=5$	$k=1$	$k=5$
GPT-4	**68.2**	**86.4**	**40.0**	**60.0**	**10.7**	**14.3**	**38.0**	**53.0**
`text-davinci-003`	50.0	81.8	16.0	34.0	0.0	3.6	19.0	36.0
Codex (`code-davinci-002`)	27.3	50.0	12.0	22.0	3.6	3.6	13.0	23.0
Human (LeetCode users)	72.2		37.7		7.0		38.2	

Table 2: Zero-shot pass@1 and pass@5 accuracies (%) on LeetCode.

It can take instructions at a very high level and emit code in many different languages.

It also responds to feedback in the form of error messages (not shown here) to fix or refine results.

Data Visualization In Figure 3.2, we ask both GPT-4 and ChatGPT to extract data from the LaTeX code for Table 2 and produce a plot in Python based on a conversation with the user. Afterwards, we ask both models to perform various operations on the produced plots. While both models extract the data correctly (not a trivial task, since one must infer from the multicolumn that the Human row has the same value for $k = 1$ and $k = 5$), ChatGPT never produces the desired plot. In contrast, GPT-4 responds appropriately to all user requests, manipulating the data into the right format and adapting the visualization. In Appendix C.2, we include another example where GPT-4 visualizes the IMDb dataset.

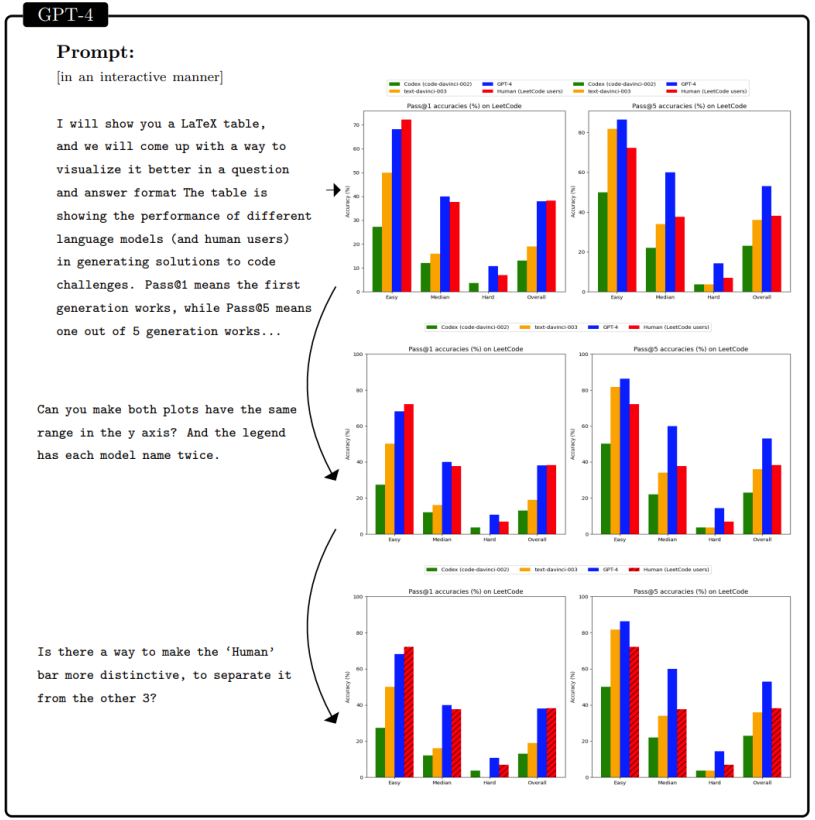


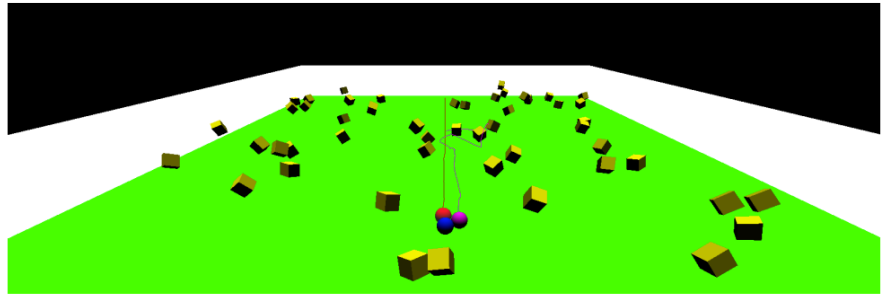
Figure 3.2: GPT-4 visualizes data from a LaTeX table (i.e., Table 2). We point out that GPT-4 also generates the format for this figure. We asked the model how to plot arrows connecting figures in LaTeX and GPT-4 produced a working Tikz snippet with the layout and arrows that we adopt here.

And we aren't just talking code snippets. Here is a complete game.

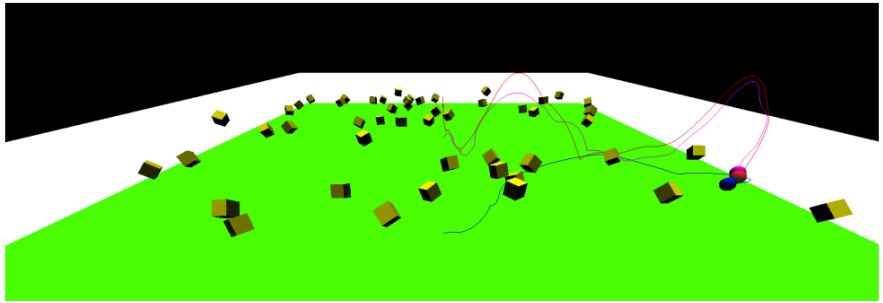
GPT-4

Prompt:

```
Can you write a 3D game in HTML with Javascript, I want:
-There are three avatars, each is a sphere.
-The player controls its avatar using arrow keys to move.
-The enemy avatar is trying to catch the player.
-The defender avatar is trying to block the enemy.
-There are also random obstacles as cubes spawned randomly at the beginning and moving randomly. The avatars cannot cross those cubes.
-The player moves on a 2D plane surrounded by walls that he cannot cross. The wall should cover the boundary of the entire plane.
-Add physics to the environment using cannon.
-If the enemy catches the player, the game is over.
-Plot the trajectories of all the three avatars.
```



Episode 1: the player avatar (blue) stands still. The enemy (red) rushes straight towards the player, while the defender (magenta) attempts to 'body block' the enemy (see its curved trajectory in magenta).



Episode 2: the player (blue) wanders around. The enemy (red) attempts to close the gap but crashes into a cube and bounces into the air. Trying to obstruct, the defender (magenta) mimics its move to follow up into the air, but ends up failing. The enemy (red) catches the player (blue) in the end.

Figure 3.3: GPT-4 develops a complex 3D HTML game in JavaScript.

Prompt:

```
I want to write a customized optimizer in pytorch, the optimizer should do:
1. the parameters for the optimizer is learning rate, momentum ,weigth decay, d_dim, k and factor alpha
2. the optimizer goes through each parameter in the network, and then
-- Get the gradient of this parameter, reshape it to 2D by looping through the dimensions, and keeping the dimension whose actual
dimension (shape) is equal to d_dim, do an SVD decomposition of the parameter so W = U\Sigma V.
-- Keep the top k singular vectors of W, so we get a new matrix W_k = U \Sigma_k V.
-- Keep the top 2k singular vectors of W, so we get a new matrix W_{2k} = U \Sigma_{2k} V.
-- Normalize W_k according to the momentum of the F-norm of W_{2k}.
-- Truncate each coordinate of W_k to the top alpha  percentile of all the coordinates, in absolute value.
-- Apply momentum on W_k, and then update the network parameters using this momentum.
-- Apply weight decay.
```

The truest meta: GPT-4 writing DL code. Note that this is not a pre-existing code that it has learned.

GPT-4:

```
def step(self):
    for group in self.param_groups:
        for p in group['params']:
            if p.grad is None:
                continue
            grad = p.grad.data
            # reshape grad to 2D
            grad_2d = grad.reshape(-1, group['d_dim'])
            # SVD decomposition
            U, S, V = torch.svd(grad_2d)
            # keep top k singular vectors
            G_k = U[:, :group['k']] @ torch.diag(S[:group['k']]) @
            V[:, :group['k']].T
            # keep top 2k singular vectors
            G_2k = U[:, :2*group['k']] @ torch.diag(S[:2*group['k']])
            @ V[:, :2*group['k']].T
            # normalize G_k
            G_k /= torch.norm(G_2k, p=2)
            # truncate G_k
            threshold = torch.quantile(torch.abs(G_k),
            group['alpha'])
            G_k[torch.abs(G_k) < threshold] = 0
            # apply momentum
            if 'momentum_buffer' not in group:
                group['momentum_buffer'] = torch.zeros_like(G_k)
            group['momentum_buffer'] = group['momentum'] *
            group['momentum_buffer'] + G_k
            # update parameters
            p.data.add_(-group['lr'] * group['momentum_buffer'])
```

Deep Learning. Writing code for deep learning requires knowledge of mathematics, statistics, and familiarity with frameworks and libraries such as PyTorch, TensorFlow, Keras, etc. In Figure 3.4 we ask both GPT-4 and ChatGPT to write a custom optimizer module, a task that can be challenging and error-prone even for human deep learning experts. We give these models a natural language description, which includes a sequence of non-trivial operations such as applying SVD, truncating a matrix spectrally at the top-k and top-2k eigenvalues, normalizing the top-k truncated matrix using the F-norm of the top-2k truncated matrix, applying momentum and weight decay. These instructions are not spelled out in complete detail, e.g., "apply momentum on G_k" requires "deep learning common sense". It is important to note that this particular optimizer does not exist in the literature or on the internet, and thus the models cannot have it memorized, and must instead compose the concepts correctly in order to produce the code.

Reasoning about code execution. In the example in Figure 3.6, we ask GPT-4 and ChatGPT to predict and explain the output of a C program that prints the size of two structures. GPT-4 correctly explains that the output may vary depending on the alignment rule used by the compiler, and gives an example of a possible output with 4-byte alignment. ChatGPT ignores the alignment issue and gives a wrong output, and also makes a false statement about the order of the members not affecting the size of the structure.

This fairly direct analysis does better than many programmers I come across.

Prompt:

```
struct mystruct_A {  char a; int b; char c; } x;
struct mystruct_B {  int b;  char a; char c; } y;

#include <stdio.h>
int main(void) {
    printf("Size of x: %zu bytes\n", sizeof(x));
    printf("Size of y: %zu bytes\n", sizeof(y));
    return 0;
}

Possible Output:
```

GPT-4:

There are different ways to implement and align structures in memory, so the exact output may vary depending on the compiler, platform, and options used. However, one possible output is:

Size of x: 12 bytes
Size of y: 8 bytes

This output assumes that the compiler uses 4-byte alignment for structures, meaning that each structure member starts at an offset that is a multiple of 4 bytes from the beginning of the structure. This may result in some padding bytes being inserted between or after the members to maintain alignment. For example, in x, there may be 3 padding bytes after a and 3 padding bytes after c, while in y, there may be no padding at all. The size of a structure is the sum of the sizes of its members plus any padding bytes.