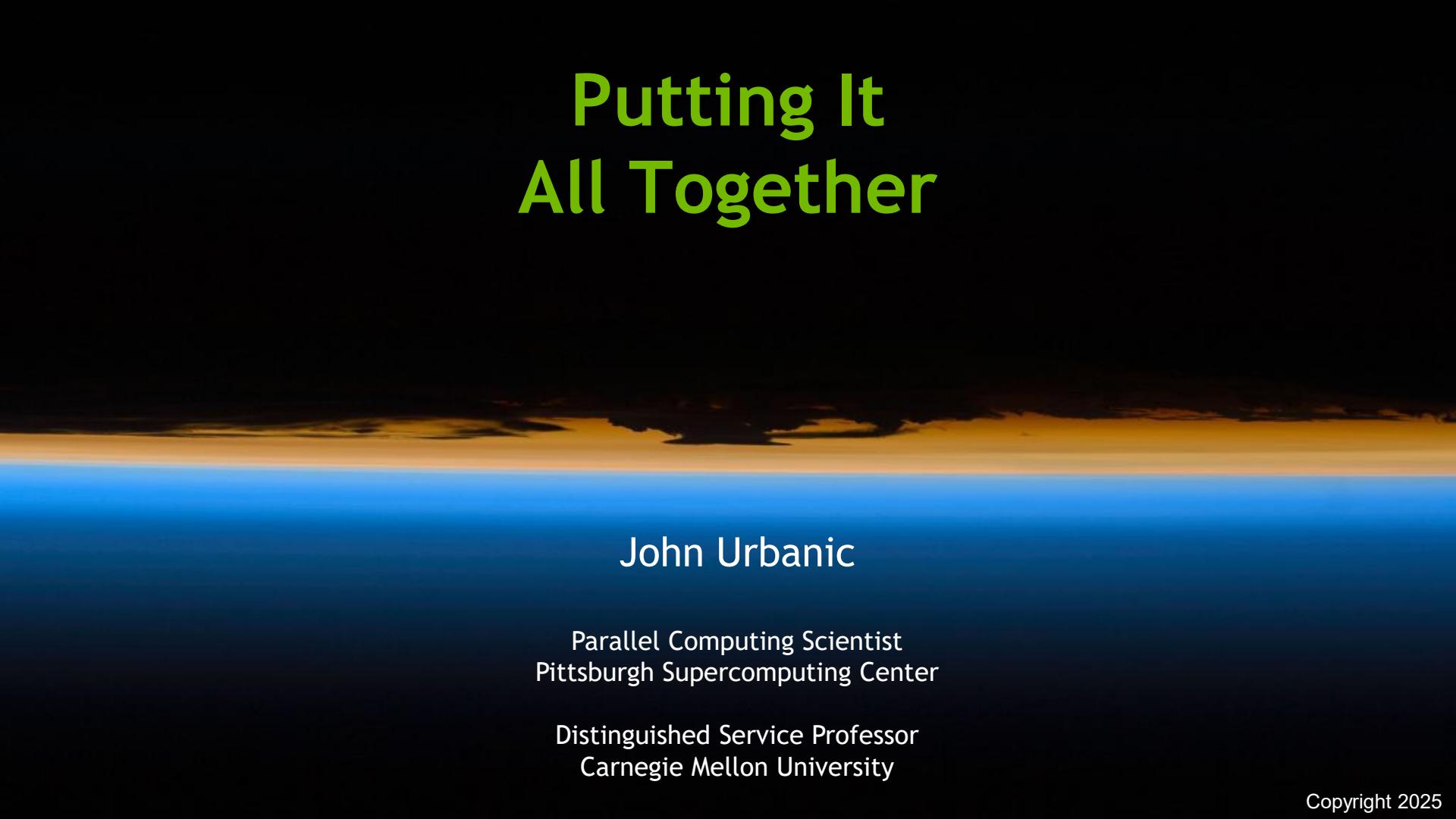


# Putting It All Together



John Urbanic

Parallel Computing Scientist  
Pittsburgh Supercomputing Center

Distinguished Service Professor  
Carnegie Mellon University

# Building Blocks

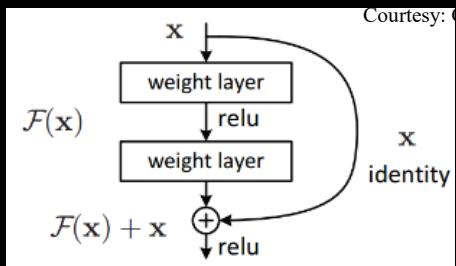
So far, we have used Fully Connected and Convolutional layers. These are ubiquitous, but there are many others:

- 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

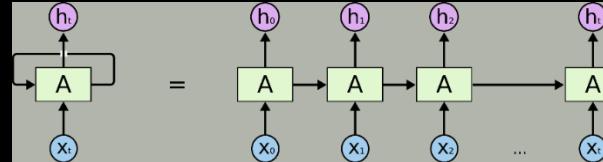
```
#Example: input 3-channel 256x256 image
x = Input(shape=(256, 256, 3))
y = Conv2D(3, (3, 3))(x)
z = keras.layers.add([x, y])
```

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/yadavdarshak/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.

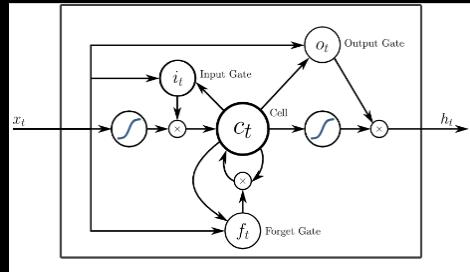
*"Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteers be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos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

The dog chased the cat up the tree.

# 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/](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/](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:

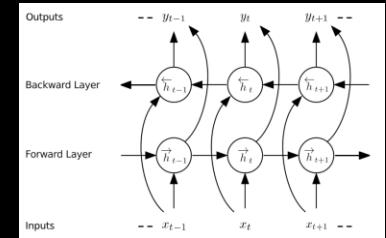
The dog chased the cat

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

The dog chased the cat 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](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.

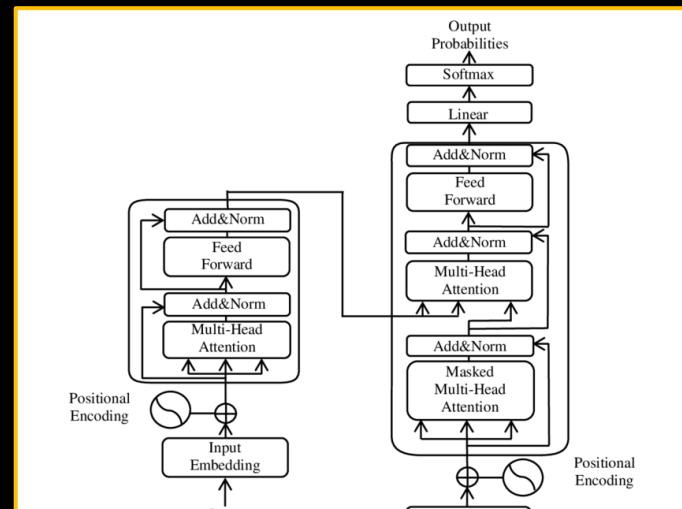
The seminal implementation goes all the way back to 2017 with *“Attention Is All You Need,” Vaswani et al.*

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 importance 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 involved, I expect that the scientific community will find some use for them.



Once again, 3Blue1Brown

<https://www.youtube.com/watch?v=wjZofJX0v4M>

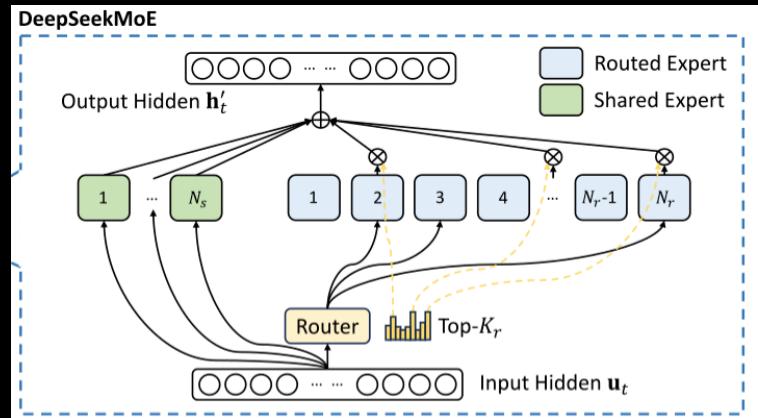
# Mixture of Experts

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%+ of 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's 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 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 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.

As an example of this complexity: 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's 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.

# Foundation Models

The most pragmatic solution to these great training costs is to build on top of existing models. Foundation Models are trained on broad data (and are often multi-model) and only require fine-tuning for a specialized task. Most of the well-known LLMs are foundation models.

Fine tuning consists of additional training to target a specific application. Keeping some of the parameters fixed or frozen is an effective way to decrease the fine-tuning training cost of these very large models. This can also help to preserve the generalization of the original model.

Two related techniques are often employed in helping to make these very large foundation models less resource intensive. Quantization is where the parameter precision of the original model is reduced, and distillation is where a smaller model is trained to mimic the larger model.

Foundation models aren't just the giant LLMs. There are many more modest ones. For example, the *YOLO* object detection network is very popular as a starting point for many image recognition applications. There are many different versions which can be selected as a foundation for fine tuning to a specific application.

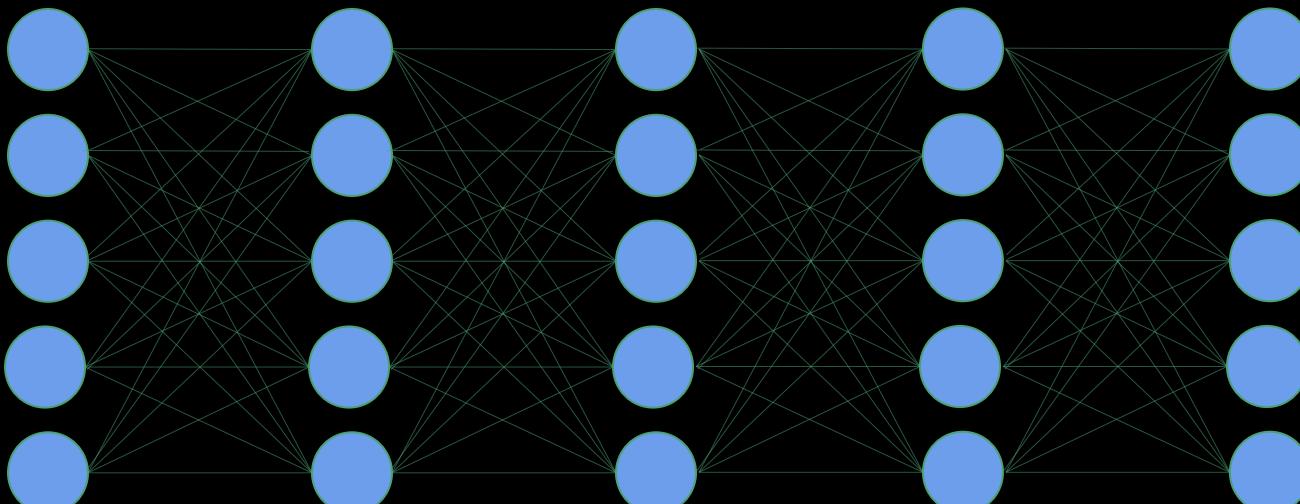
# Autoencoder



Input Layer



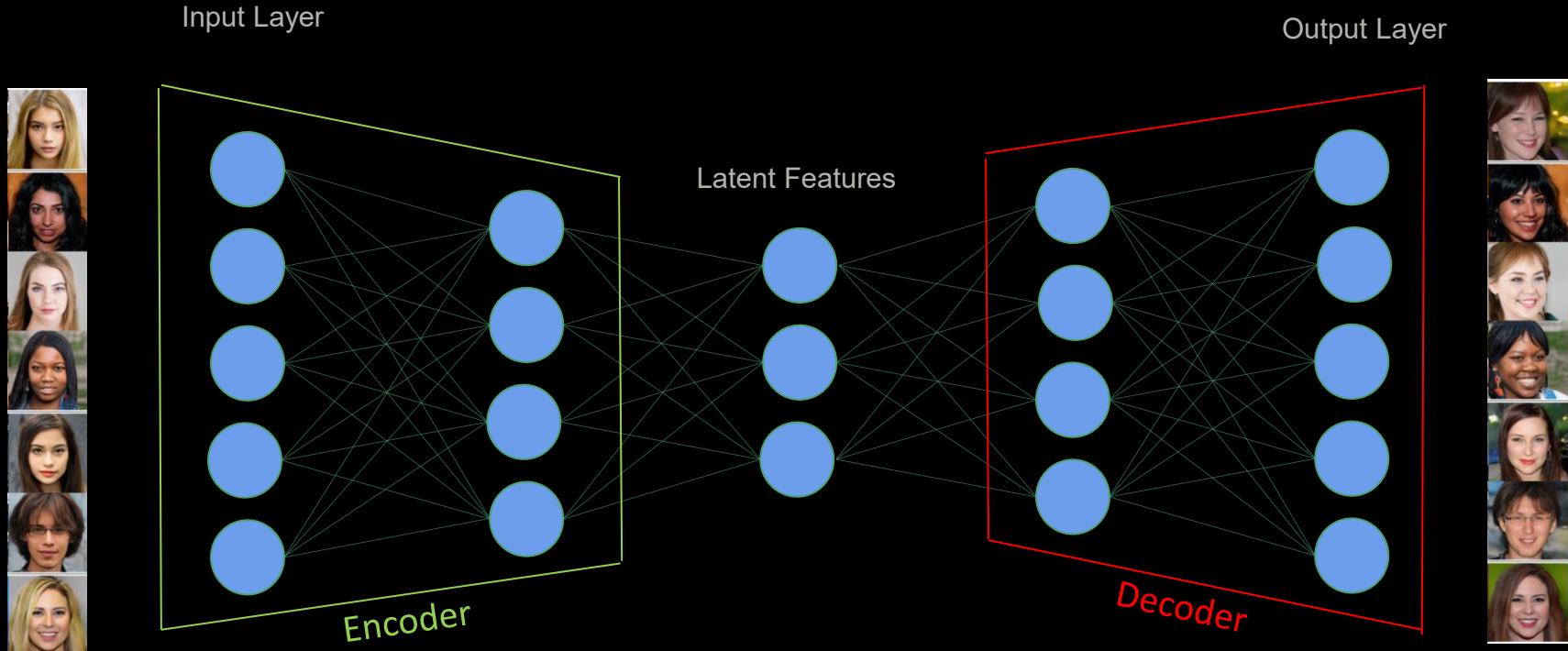
Hidden Layers



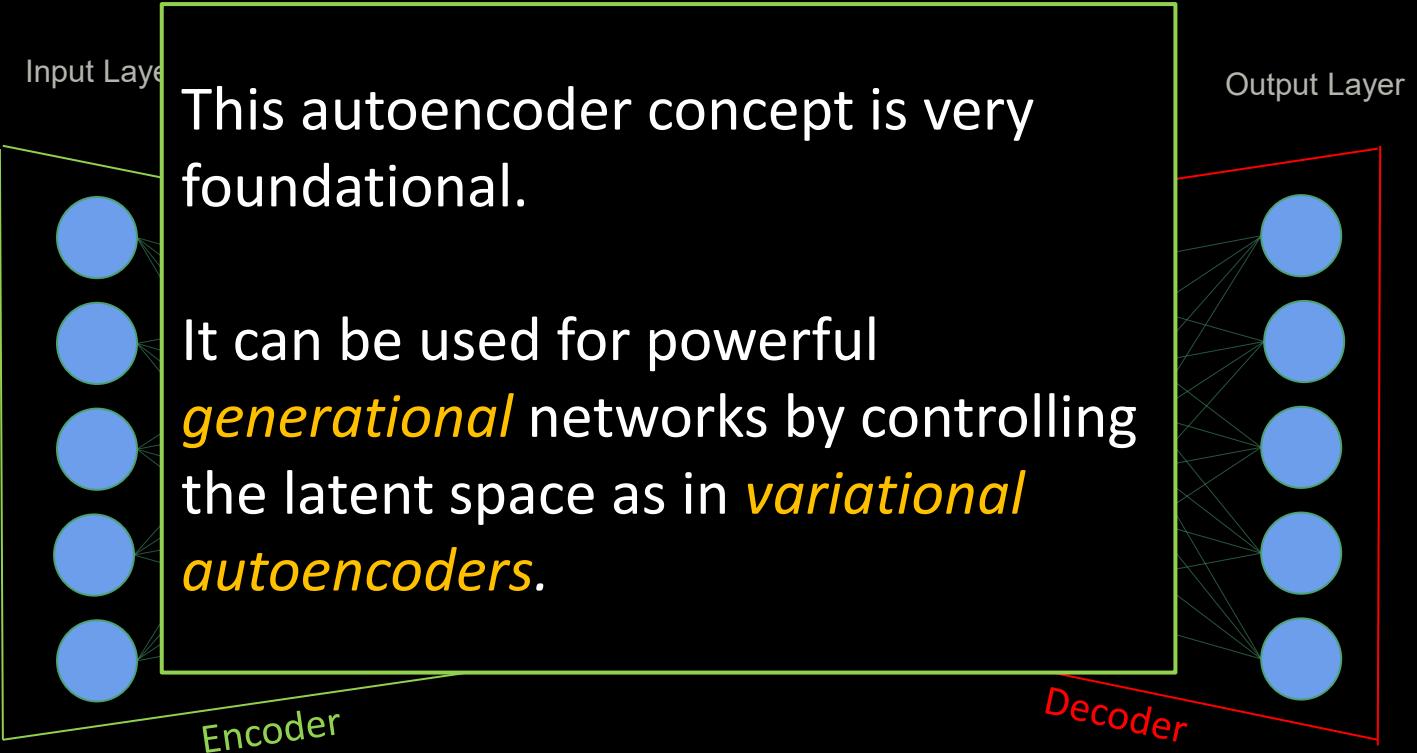
Output Layer



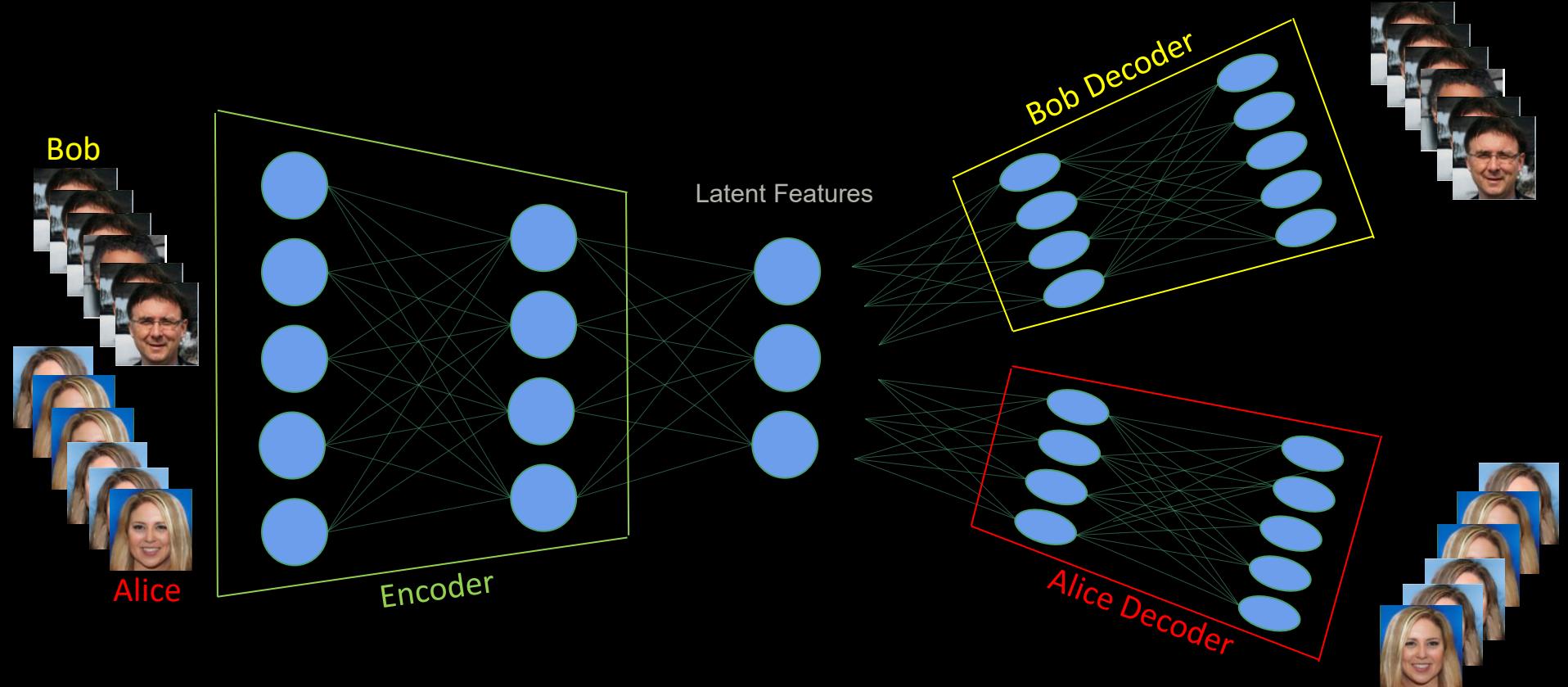
# Autoencoder



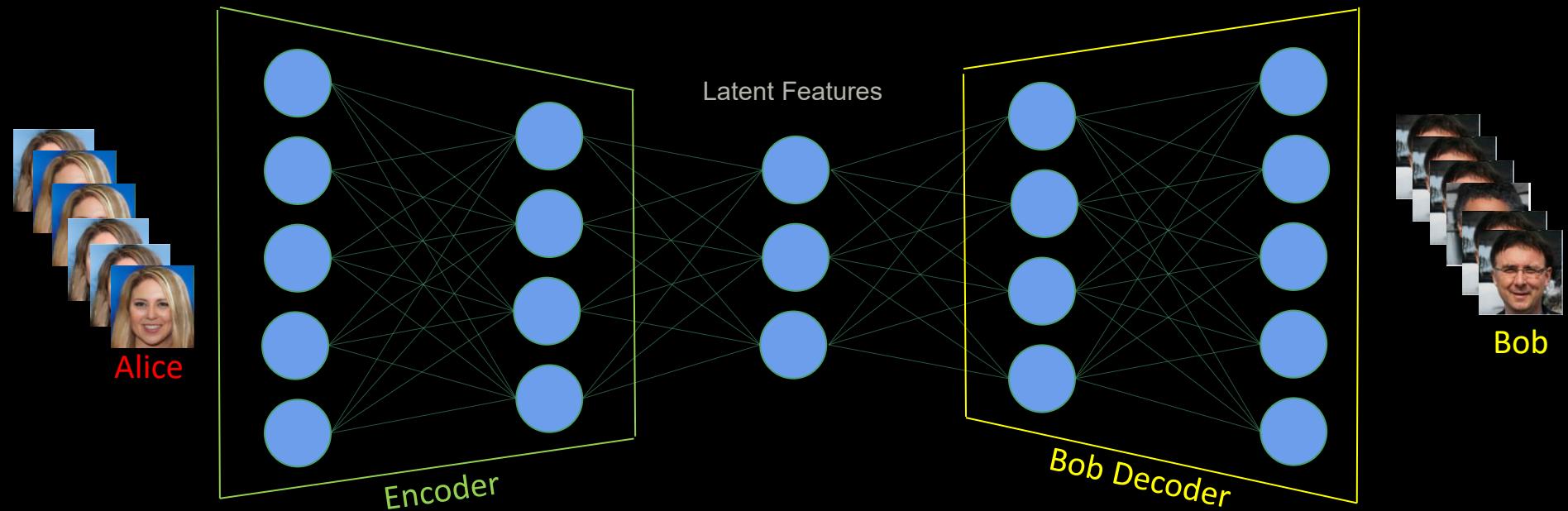
# Autoencoder



# 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."

Deep Fake I'm working on. Running on python 3.10 in real time. I struggled to find the correct drivers but ended up getting a good result.



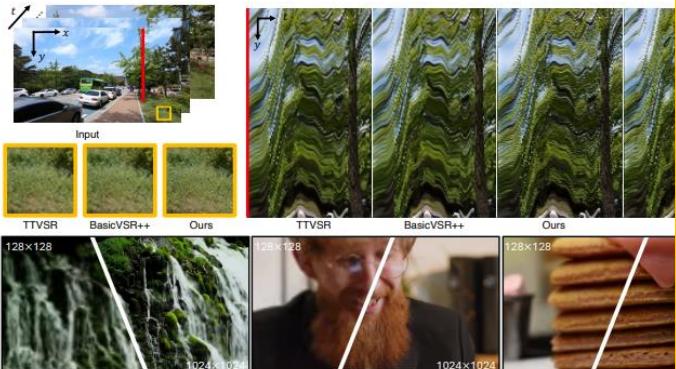
# Super Resolution

## VideoGigaGAN: Towards Detail-rich Video Super-Resolution

Yiran Xu<sup>1,2</sup>, Taesung Park<sup>1</sup>, Richard Zhang<sup>1</sup>, Yang Zhou<sup>1</sup>, Eli Shecht  
Feng Liu<sup>1</sup>, Jia-Bin Huang<sup>2</sup>, and Difan Liu<sup>1</sup>

<sup>1</sup> Adobe Research

<sup>2</sup> University of Maryland, College Park  
<http://videogigagan.github.io>



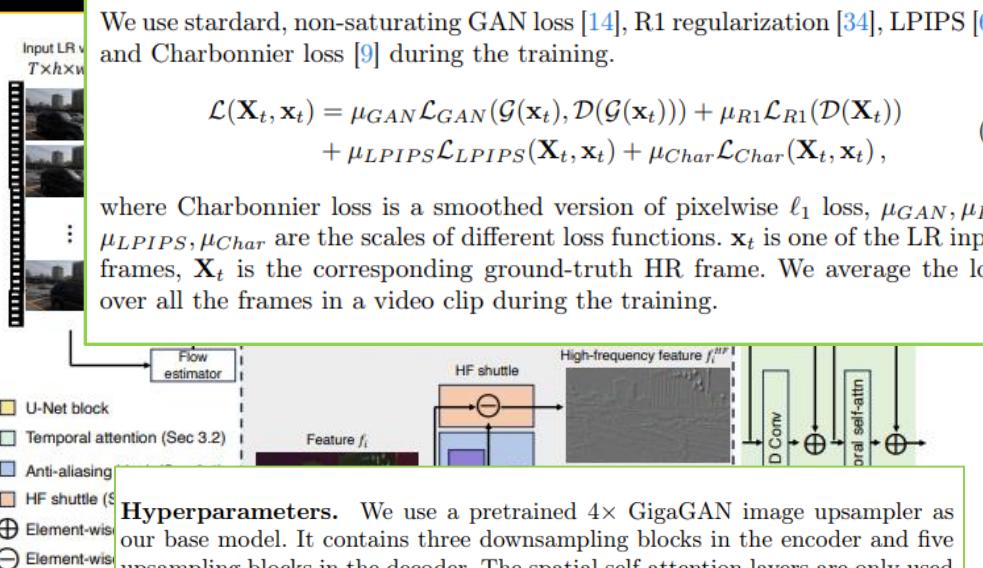
**Fig. 1:** We present **VideoGigaGAN**, a generative video super-resolution model that can upsample videos with high-frequency details while maintaining temporal consistency. *Top*: we show the comparison of our approach with TTVSR [33], BasicVSR++ [7]. Our method produces temporally consistent videos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

## 3.6 Loss functions

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

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

where Charbonnier loss is a smoothed version of pixelwise  $\ell_1$  loss,  $\mu_{GAN}$ ,  $\mu_{R1}$ ,  $\mu_{LPIPS}$ ,  $\mu_{Char}$  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.



**Hyperparameters.** We use a pretrained 4 $\times$  GigaGAN image upsample 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 $\times$ 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  $5 \times 10^{-5}$  for both generator and discriminator. The total number of training iterations is 100,000.

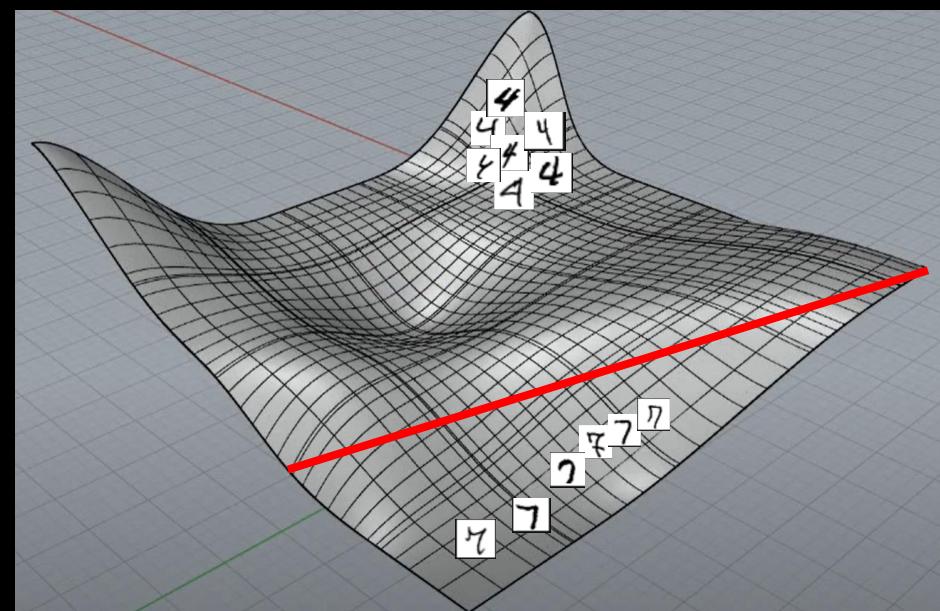
**Fig. 3: Overview**  
The VideoGigaGAN (VSR) model is based on the GigaGAN upsample into four decoder blocks.

**Flow-guided pro-**  
**Anti-aliasing bl-**

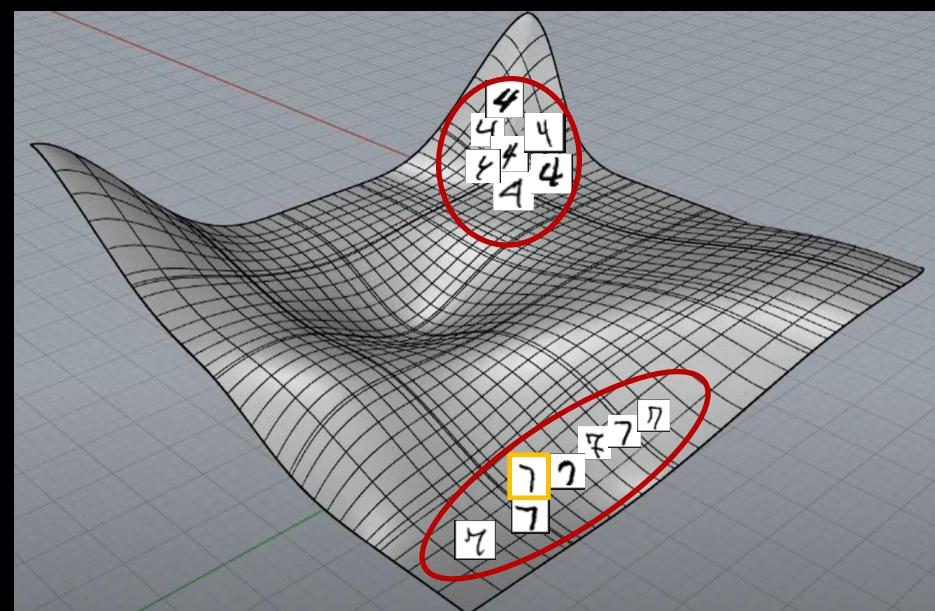
shuttle the high frequency features via skip connection to the decoder layers to compensate for the loss of details in the BlurPool process.

# 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  $P(X,Y)$ .

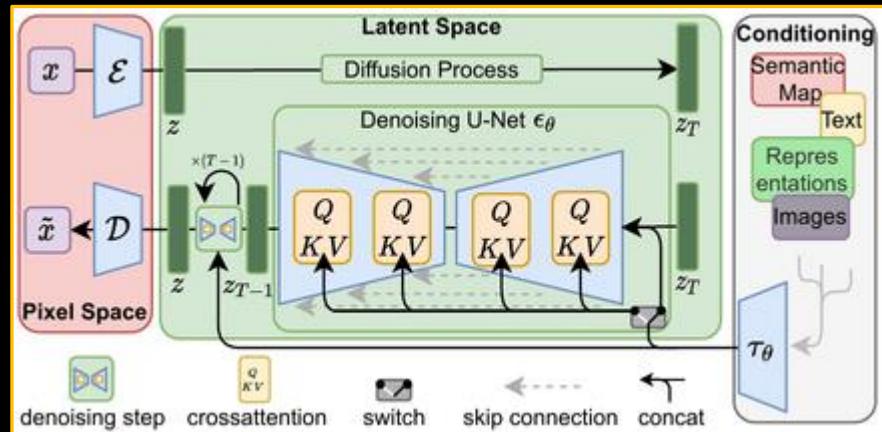
# Generative in Action

*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

## Meet DALL-E, the AI That Draws Anything at

New technology that graphic artists — and

Give this article



**Image Generated by AI**

DALL-E generated this image from a  
“Teapot playing chess.” OpenAI

DALL-E generated these images by following a few simple text prompts. See more examples at [openai.com/dalle](https://openai.com/dalle).



an espresso machine that makes coffee from human souls, artstation

## Hierarchical Text-Conditional Image Generation with CLIP Latents

Aditya Ramesh<sup>\*</sup>  
OpenAI  
aramesh@openai.com

Prafulla Dhariwal<sup>\*</sup>  
OpenAI  
prafulla@openai.com

Alex Nichol<sup>\*</sup>  
OpenAI  
alex@openai.com

Casey Chu<sup>\*</sup>  
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 in 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



a close up of a hand palm with leaves growing from it



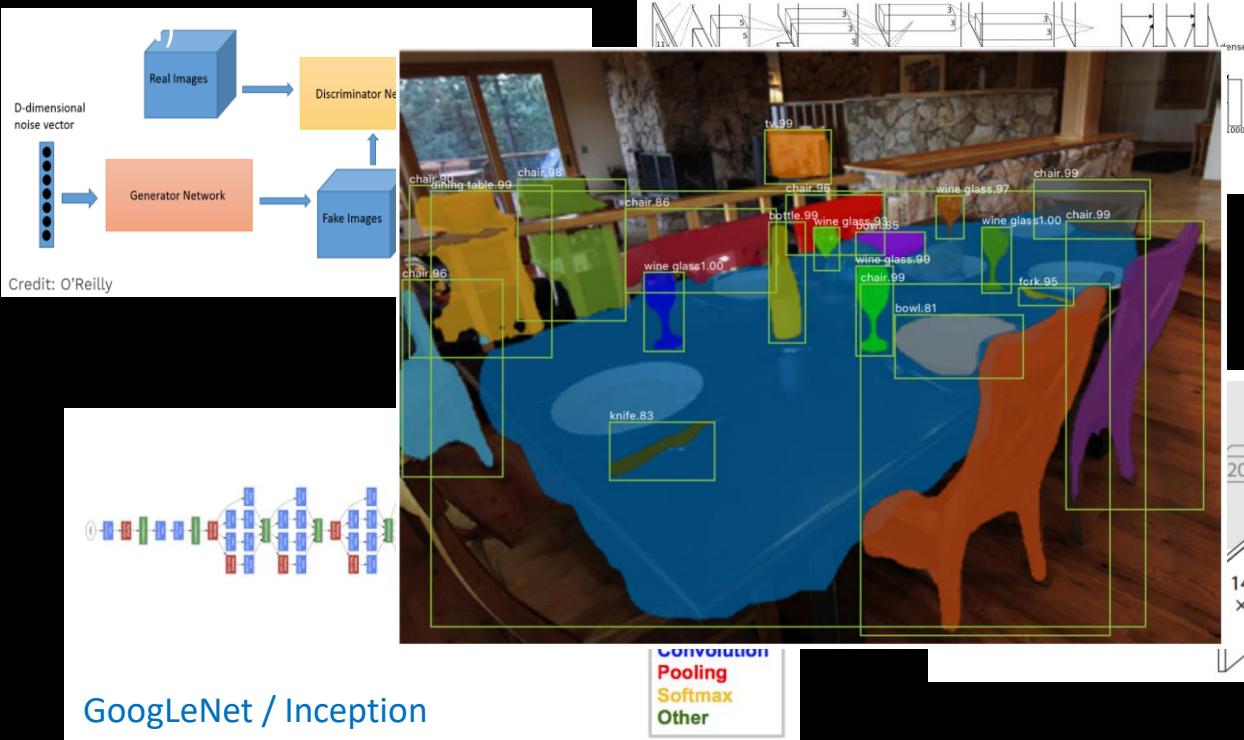
a corgi's head depicted as an explosion of a nebula

panda mad scientist mixing sparkling chemicals, artstation

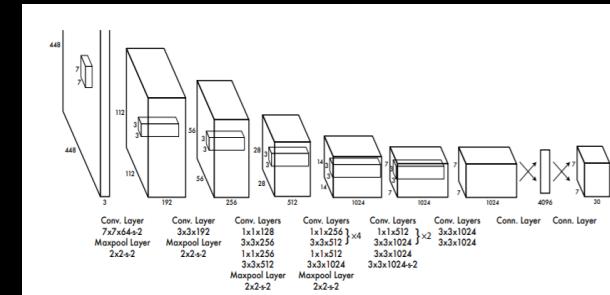
# Architectures

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.

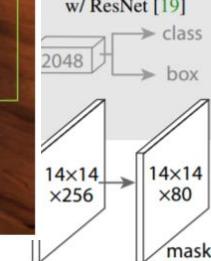
## Generative Adversarial Network



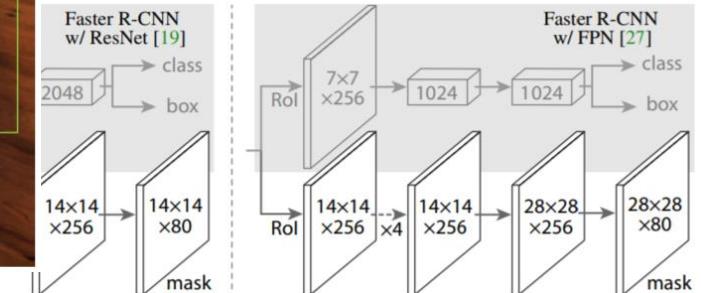
## YOLO (You Only Look Once)



## Faster R-CNN w/ ResNet [19]



## Mask R-CNN



# 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. <i>Stock price.</i>
Transcription	Convert between representations. <i>OCR, speech recognition.</i>
Synthesis	Create new input examples. <i>Speech synthesizer. Lots of science these days!</i>
Translation	Like the word says. <i>Google Translate.</i>
Segmentation	Return a relabeled input vector. <i>Tumor detection.</i>
Denoising	Return uncorrupted example. <i>Video game ray tracing.</i>

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.



## Unsupervised Learning

*Fuzzy Line*

(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

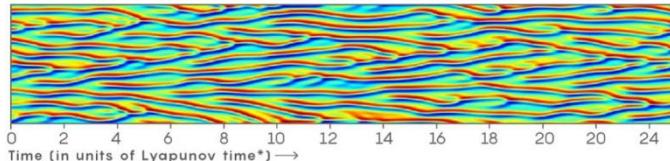
Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott  
Phys. Rev. Lett. 120, 024102 – Published 12 January 2018

## 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

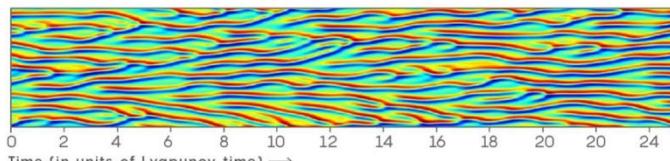
Researchers started with the evolving solution to the Kuramoto-Sivashinsky equation, which models propagating flames:



\* 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.

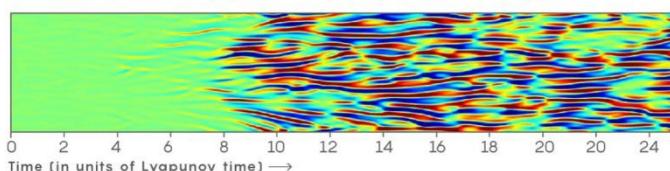
### B Machine Learning

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



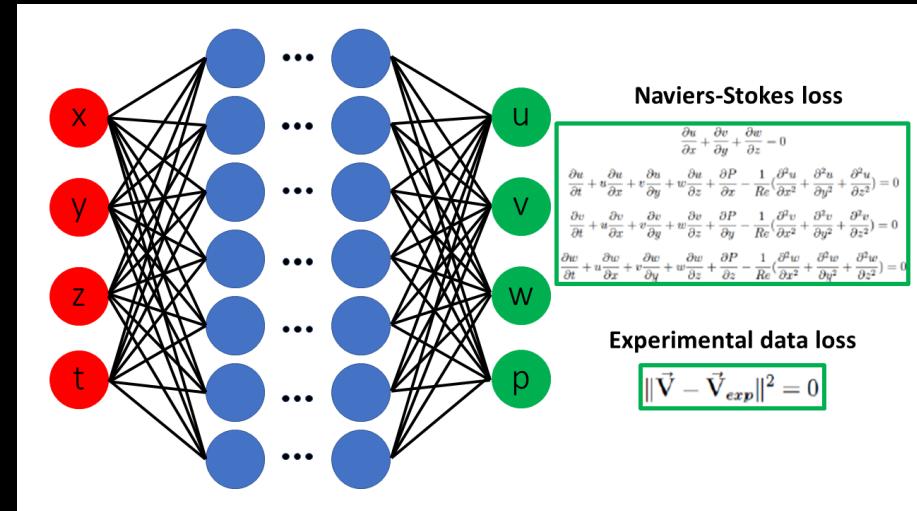
### A – B Do They Match?

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



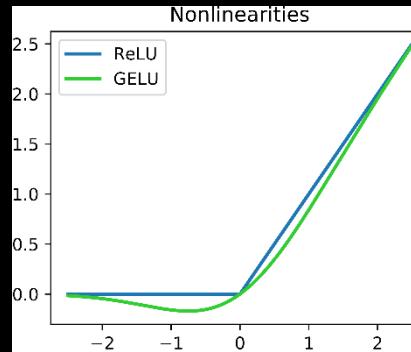
# Physics Informed Neural Networks

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



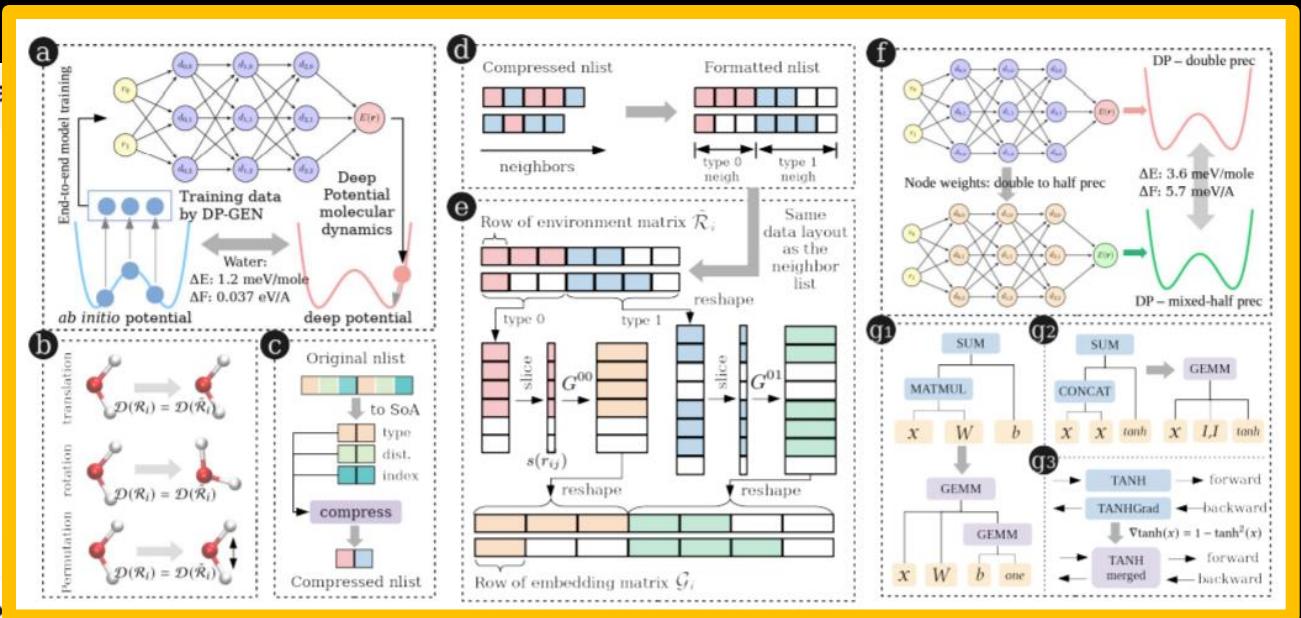
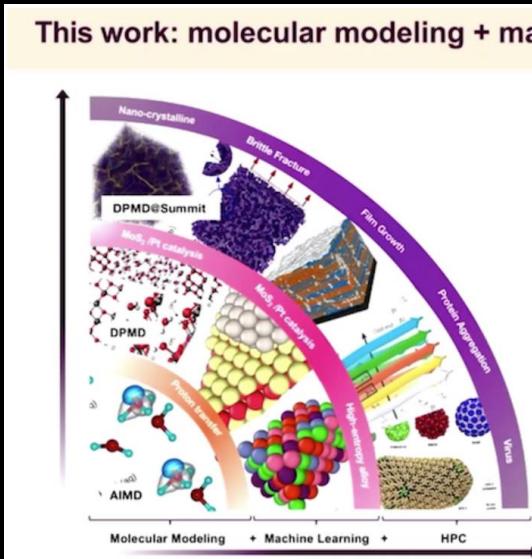
*Wikipedia*

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 DeePMDD-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.

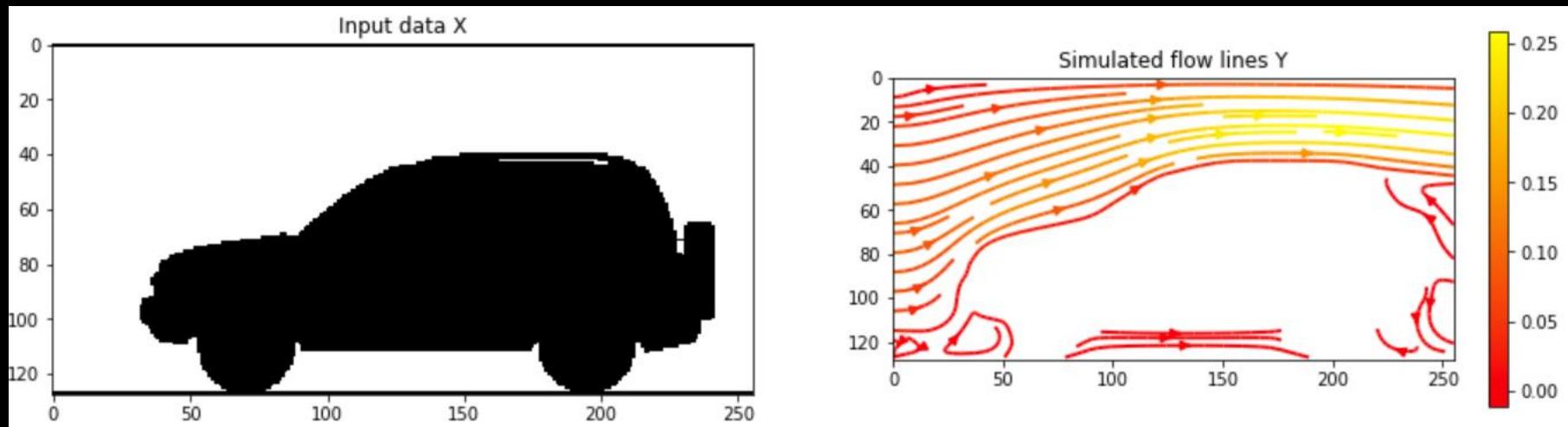


Pushing the Limit of Molecular Dynamics with Ab Initio Accuracy to 100 Million Atoms with Machine Learning

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/gpuboocamp/blob/78e9fee3432b60348489682a978fa63f29f7e839/hpc\\_ai/ai\\_science\\_cfd/English/python/jupyter\\_notebook/CFD/Start\\_Here.ipynb](https://github.com/gpuhackathons-org/gpuboocamp/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 L<sup>A</sup>T<sub>E</sub>X style file v3.0

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

Philip G. Breen<sup>1\*</sup>, Christopher N. Foley<sup>2</sup> †, Tjarda Boekholt<sup>3</sup> and Simon Portegies Zwart<sup>4</sup>

<sup>1</sup>School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, Kings Buildings, Edinburgh, EH9 3JZ

<sup>2</sup>MRC Biostatistics Unit, University of Cambridge, Cambridge, CB2 0SR, UK

<sup>3</sup>Instituto de Telecomunicações, Campus Universitário de Santiago, 9810-193, Aveiro, Portugal

<sup>4</sup>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 initialisation 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.

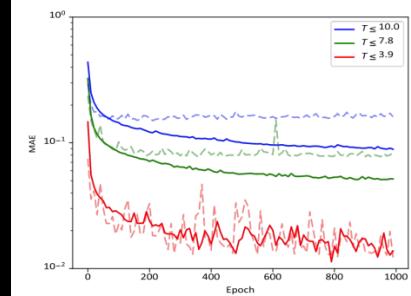
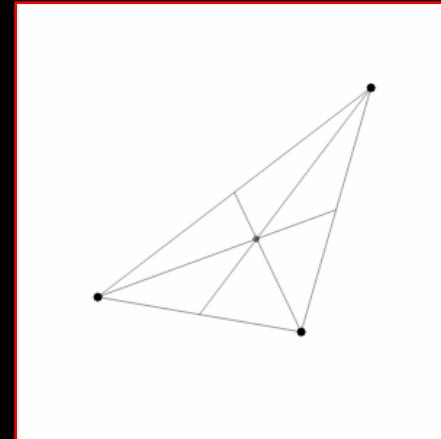
**Key words:** stars: kinematics and dynamics, methods: numerical, statistical

### 1 INTRODUCTION

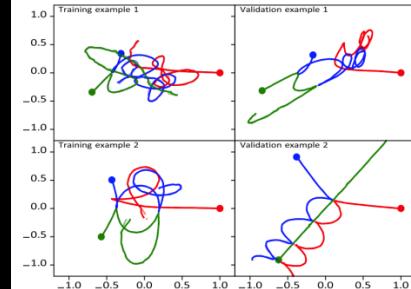
Newton's equations of motion describe the evolution of many bodies in space under the influence of their own gravitational force (Newton 1687). The equations have a central role in many classical problems in Physics. For example, the equations explain the dynamical evolution of globular star clusters 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 (Valenzen 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 (Valenzen 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),



**Figure 3.** Mean Absolute Error (MAE) vs epoch. The ANN has the same training structure in each time interval. Solid 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.  $\approx$ 100 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 with Brutus (Boekholt & Portegies Zwart 2015)). The two scenarios presented to the right were not included in the training dataset.

\* Authors contributed equally

† Contact e-mail: phil.breen@ed.ac.uk

‡ Contact e-mail: christopher.foley@mrc-bsu.cam.ac.uk

Has Deep Learning left any room for other approaches?

Big  
Data



DL  
Deep Neural Nets

DL

ML

AI

Character Recognition  
Capchas  
Chess  
Go

Go

Character Recognition  
Capchas  
Chess  
Go

# “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.

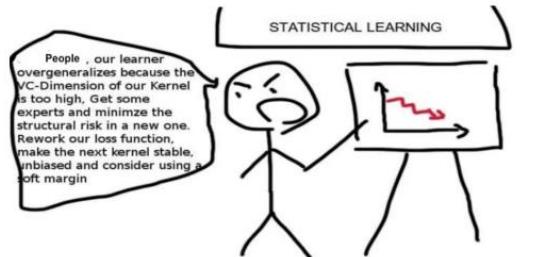
The previously mentioned Christopher Olah has this rather useful summation: *“People sometimes complain: ‘Neural networks are so hard to understand! Why can’t we use understandable models, like SVMs?’ Well, you understand SVMs, and you don’t understand visual pattern recognition. If SVMs could solve 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 19<sup>th</sup> 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.



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:

and competitions:



### Severstal: Steel Defect Detection

Can you detect and classify defects in steel?

Featured · Kernels Competition · 3 months to go · manufacturing, image data

\$120,000  
299 teams

\$100,000  
2,927 teams

\$50,000  
2,106 teams

\$30,000  
1,281 teams

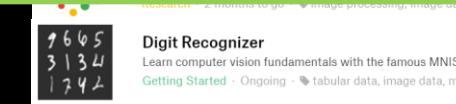
\$30,000  
2,300 teams

\$25,000  
384 teams

Knowledge  
3,008 teams

# XGBoost?

A screenshot of the Kaggle website showing a grid of dataset cards. The cards include: Waves Measuring Buoys Data (by Iñaki, 599 KB, 10.0 files, 1 CSV), Shared Cars Location (by DoiT International, 7 months, 78 MB), UW Madison Course (by Madgares, a year, 90 MB), Venues in Bournemouth (by Alex, a month, 3 KB), Women's Shoe Prices (by Datasfiniti, a month, 12 MB), Crimes in Boston (by Analyze Boston, a year, 10 MB), Peace Agreements Data (by University of Edinburgh, a year, 1 MB), Goodreads-books (by Soumik, 2 months, 602 KB, 10.0 files, 1 CSV), Ghana Health Facilities (by citizen datascience gh, a year, 84 KB), Vega shrink-wrapper (by mIT, 9 months, 408 KB), Electric Motor Temperature (by Kirjan, 2 months, 47 MB, 10.0 files, 1 CSV), Gas Prices in Brazil (by Matheus Eduardo Freitag, 23 days, 3 MB, 10.0 files, 1 CSV), and US Traffic Fatality Records (by Department of Transportation, 4 months, 585 MB, 7.1 files, 1 BigQuery).



### Digit Recognizer

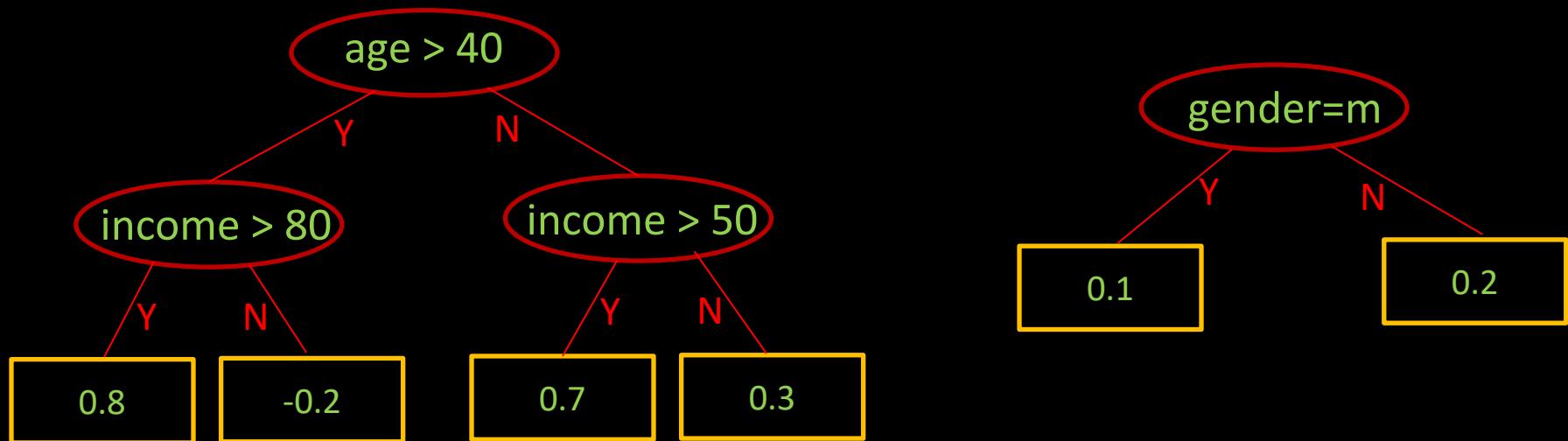
Learn computer vision fundamentals with the famous MNIST data

Getting Started · Ongoing · tabular data, image data, multiclass classification, object identification

# Trees

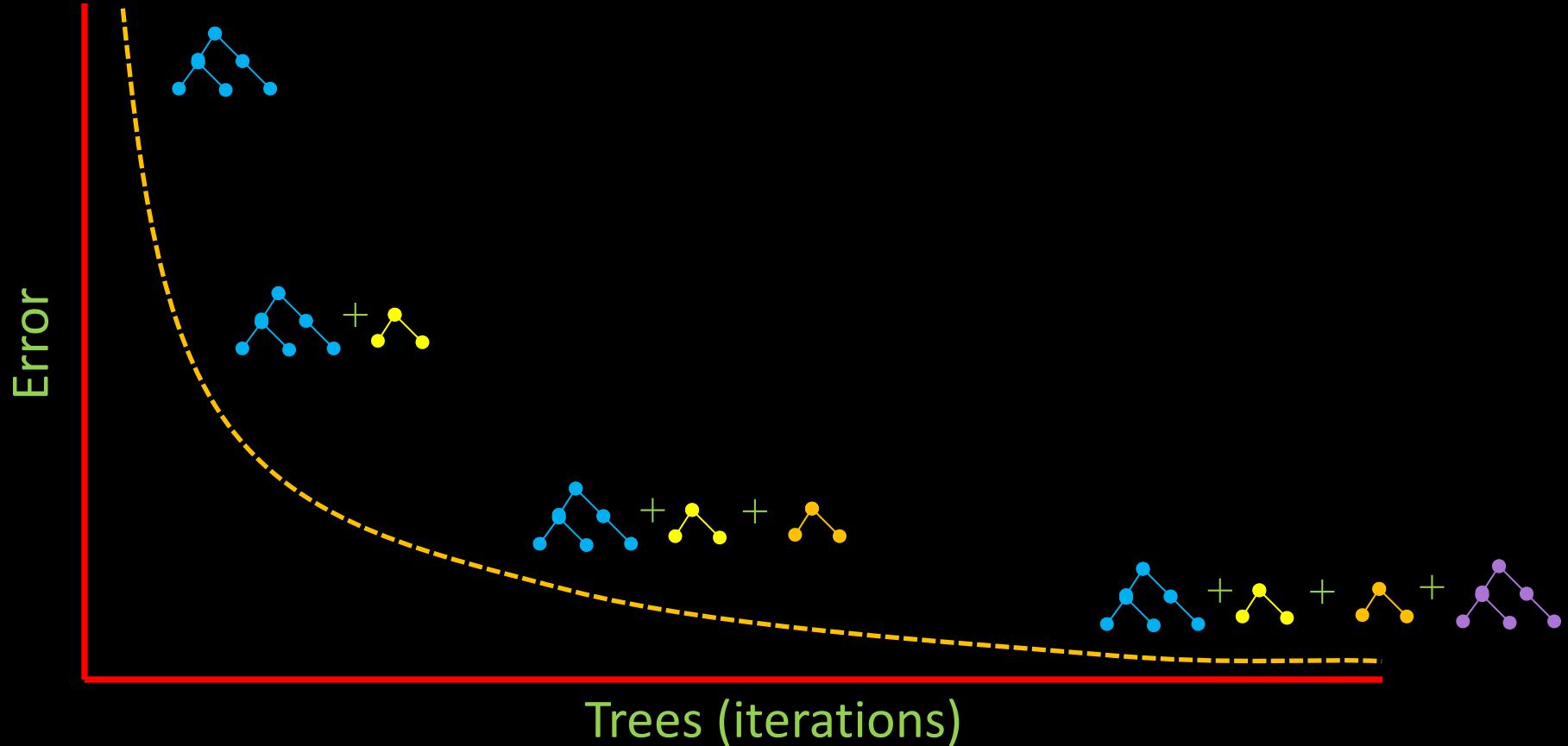
(How much of our earlier learning can we apply here?)

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

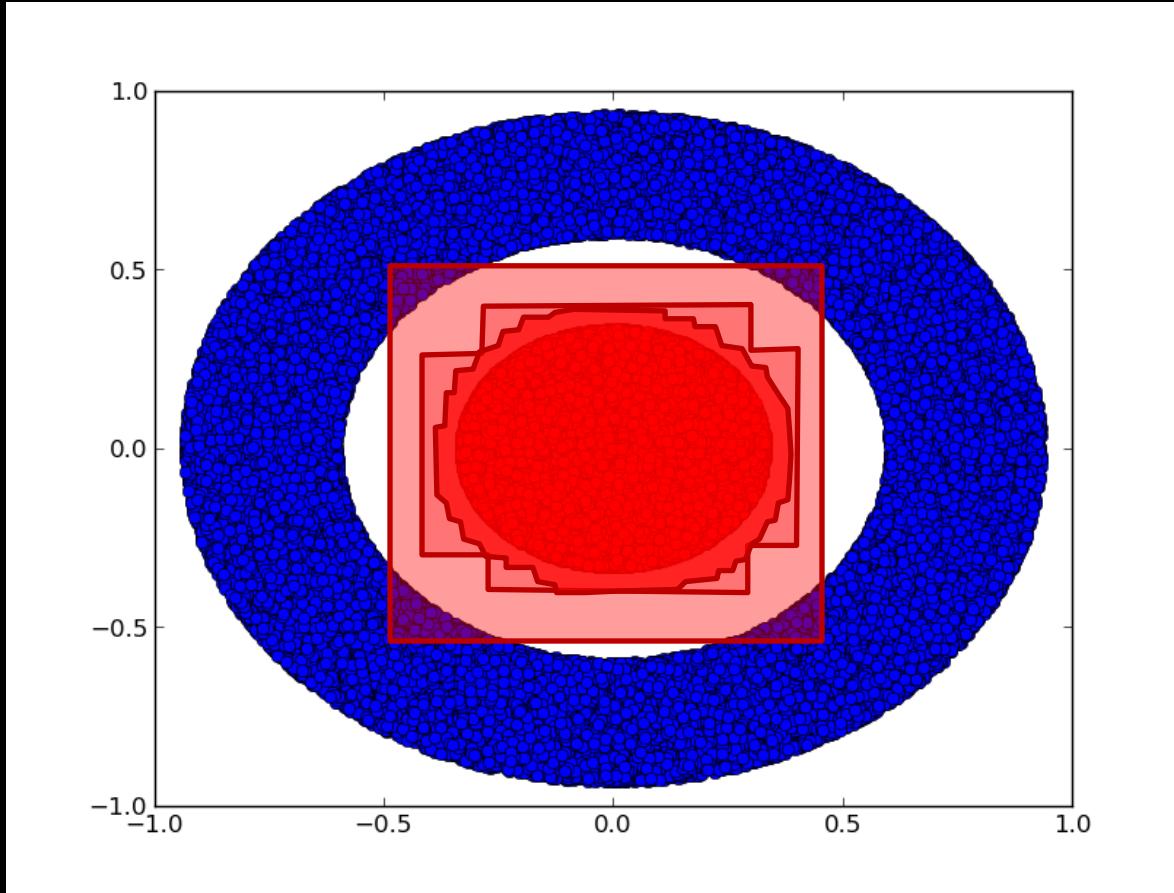


Trees are desirable in that they are non-linear, but still analytically tractable, and can do both regression and classification.

# Gradient Boosted Trees

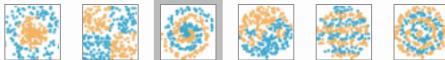


# Remember This?

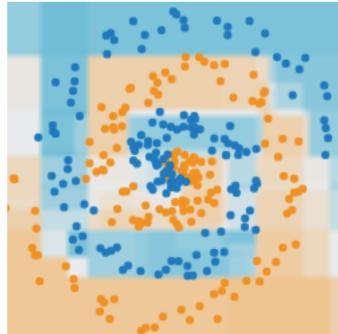


# XGBoost

Dataset to classify:



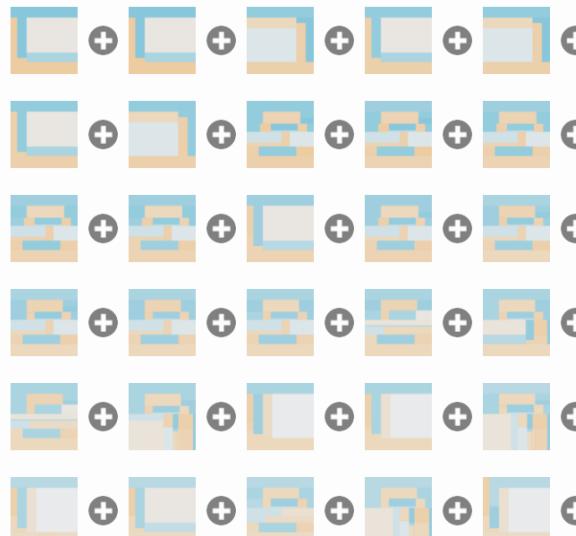
Prediction:



predictions of GB (all 50 trees)

train loss: 0.381    test loss: 0.430

Decision functions of first 30 trees



tree depth: 5

learning rate: 0.1

subsample: 100%

rotate dataset:

# trees: 50

- rotate trees
- show gradients on hover
- use Newton-Raphson update

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](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



Epoch  
002,086

Learning rate  
0.03

Activation  
ReLU

Regularization  
None

Regularization rate  
0

Problem type  
Classification

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%



Noise: 0



Batch size: 18



**REGENERATE**

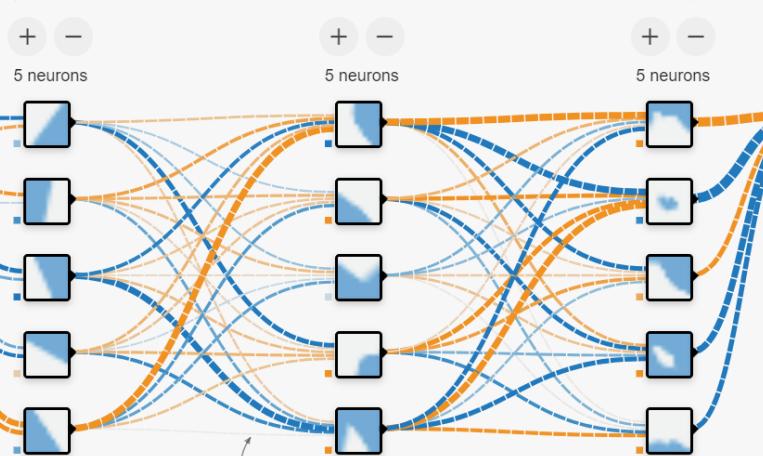
## FEATURES

Which properties do you want to feed in?

$x_1$     $x_2$     $x_1^2$     $x_2^2$     $x_1x_2$

$\sin(x_1)$     $\sin(x_2)$

+ - 3 HIDDEN LAYERS



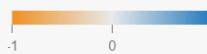
The outputs are mixed with varying weights, shown by the thickness of the lines.

## OUTPUT

Test loss 0.059  
Training loss 0.012



Colors shows data, neuron and weight values.



Show test data

Discretize output

# Great Way To Compare

Epoch 000,382 Learning rate 0.03 Activation Tanh Regularization None Regularization rate 0 Problem type Classification

**DATA**  
Which dataset do you want to use?  
   
 

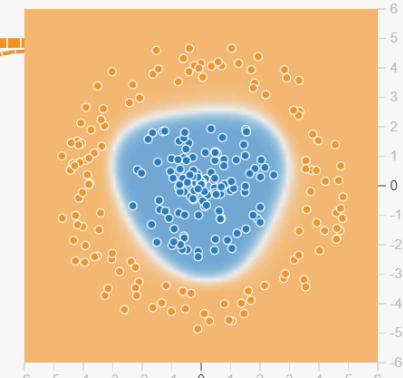
Ratio of training to test data: 50%

Noise: 0

Batch size: 10

**FEATURES**  
Which properties do you want to feed in?  
 $X_1$   $X_2$   $X_1^2$   $X_2^2$   $X_1 X_2$   $\sin(X_1)$   $\sin(X_2)$

**2 HIDDEN LAYERS**  
+ -  
4 neurons      2 neurons  
The outputs are mixed with varying weights, shown by the thickness of the lines.  
This is the output from one neuron. Hover to see it larger.

**OUTPUT**  
Test loss 0.001  
Training loss 0.001  
  
Colors shows data, neuron and weight values.  
 Show test data  Discretize output

# The Classic Conundrum

Epoch  
000,233Learning rate  
0.03Activation  
TanhRegularization  
NoneRegularization rate  
0Problem type  
Classification

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

**REGENERATE**

## FEATURES

Which properties do you want to feed in?



+ - 2 HIDDEN LAYERS



+

-

4 neurons

+

-

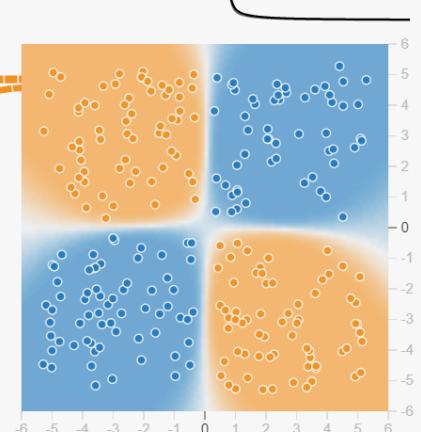
2 neurons

This is the output from one neuron.  
Hover to see it larger.

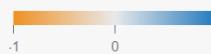
The outputs are mixed with varying weights, shown by the thickness of the lines.

## OUTPUT

Test loss 0.006  
Training loss 0.003



Colors shows data, neuron and weight values.



Show test data

Discretize output

# 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](https://xgboost.readthedocs.io/en/latest/jvm/xgboost4j_spark_tutorial.html)), and if you want a deeper dive, NVIDIA 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](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 its 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.

Draw your number here



Downsampled drawing: 2

First guess: 2

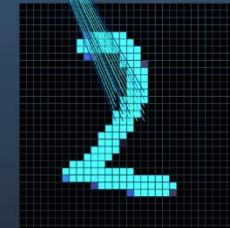
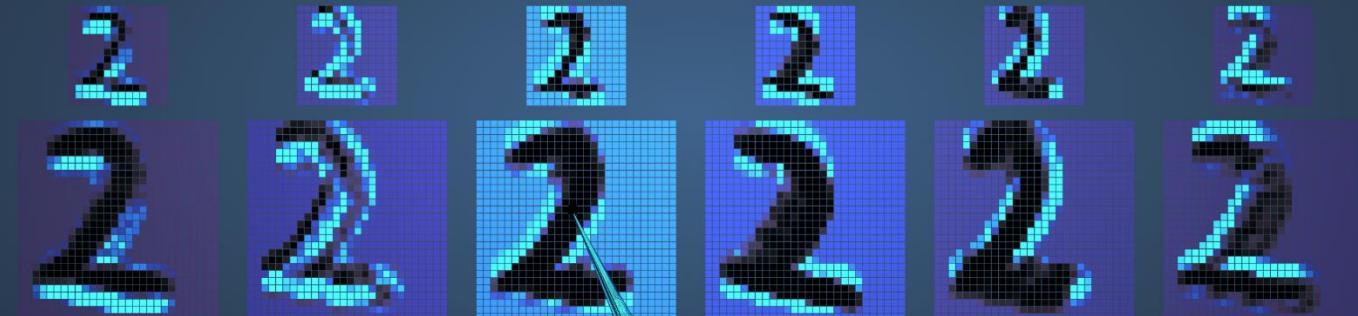
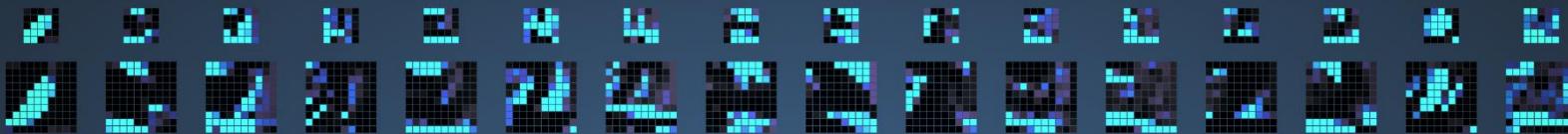
Second guess: 0

Layer visibility

Input layer	Show
Convolution layer 1	Show
Downsampling layer 1	Show
Convolution layer 2	Show
Downsampling layer 2	Show
Fully-connected layer 1	Show
Fully-connected layer 2	Show
Output layer	Show

# A note about hardware.

0 1 2 3 4 5 6 7 8 9

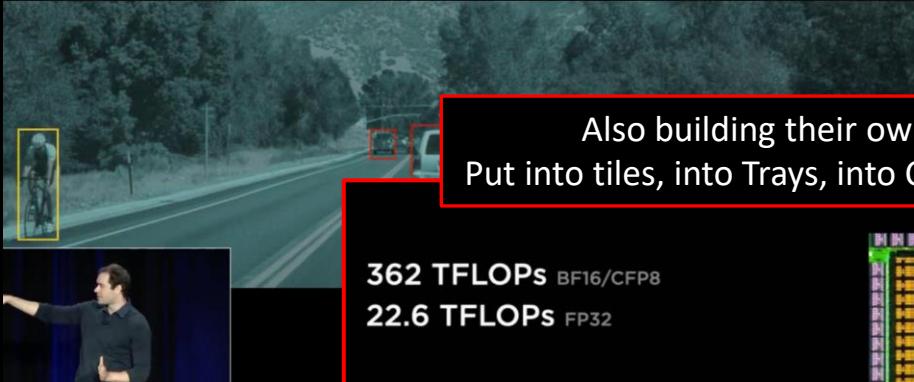


# Inference Is Fast

## Perceptual Labs



iPhone Demo



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

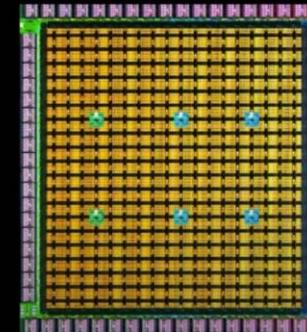
**362 TFLOPs** BF16/CFP8

**22.6 TFLOPs** FP32

**10TBps/dir.** On-Chip Bandwidth

**4TBps/edge.** Off-Chip Bandwidth

**400W TDP**



**645mm<sup>2</sup>**  
7nm Technology

**50 Billion**  
Transistors

**11+ Miles**  
Of Wires

**32MB SRAM**

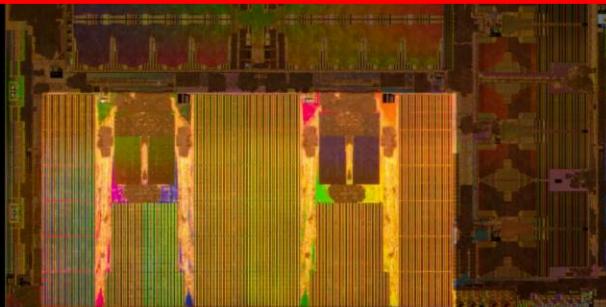
**96x96 Mul/Add array**

**ReLU hardware**

**Pooling hardware**

**36 TOPS @ 2 GHz**

**2 per chip, 72 TOPS total**

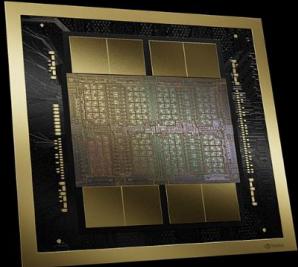


TESLA **LIVE**

# Everyone Doing Specialized Hardware

NVIDIA

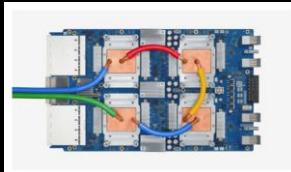
Blackwell



Highlighting 4 bit precision!

Google

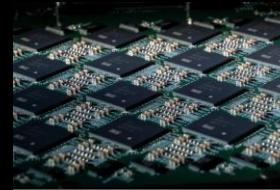
TPU



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

Intel

Loihi 2



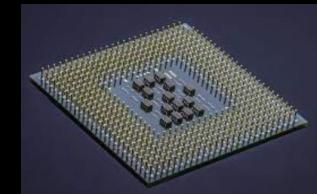
1,048,576 artificial neurons,  
and 120 million synapses

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



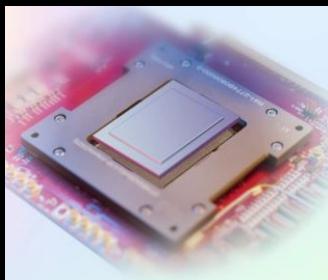
*Inferentia* for inference and  
*Trainium* for training. Use  
standard TensorFlow and  
Torch in their EC2 Cloud.

This is where Alexa runs.

# Everyone Doing Specialized Hardware

Meta

MTIA 2



708 TFLOPS/s (INT8)

Microsoft

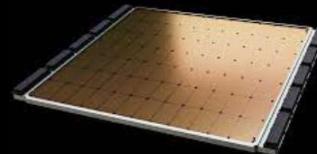
Athena



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

Cerebras

CS-2



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, ...



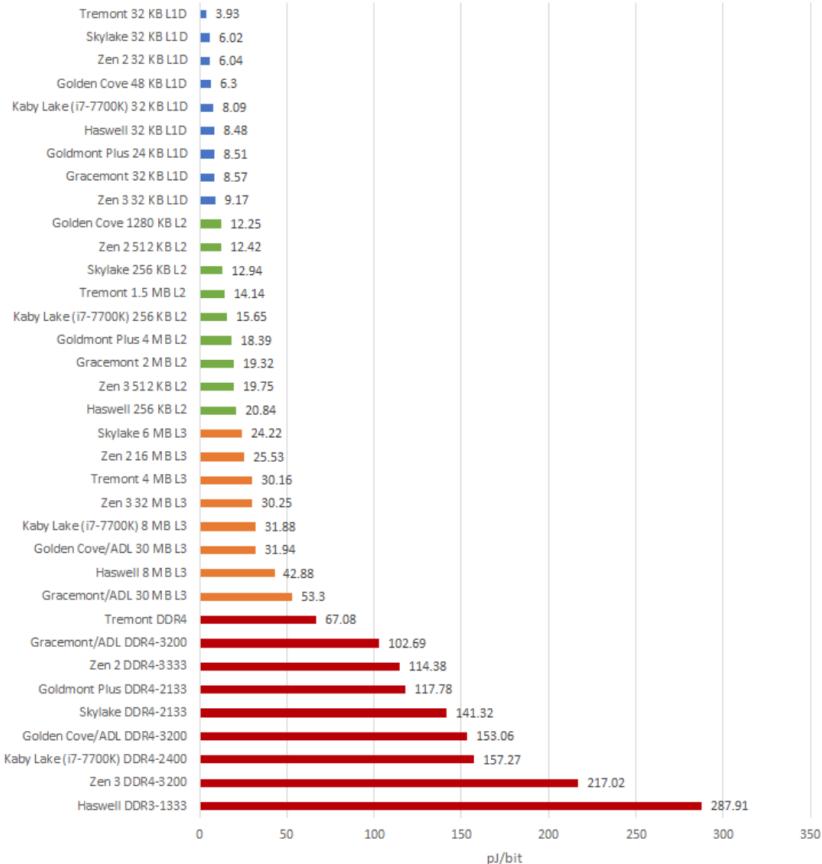
Brain only uses 20W.

Analog, pruning, spiking, lots of new directions.

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

# More Inspiration

Data Transfer Energy Cost



- 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 ( $5 \times 10^{-14}$  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  $kT\ln(2)$  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<sup>1</sup>, Yu Zhang<sup>2</sup>, Ethan Siferman<sup>1</sup>, Tyler Sheaves<sup>3</sup>, Yiqiao Wang<sup>4</sup>,  
Dustin Richmond<sup>1</sup>, Peng Zhou<sup>1,4</sup>, Jason K. Eshraghian<sup>1,\*</sup>

<sup>1</sup>University of California, Santa Cruz <sup>2</sup>Soochow University  
<sup>3</sup>University of California, Davis <sup>4</sup>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 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 ternary 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.

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.<sup>4</sup> 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 call to consciously—tasks that Kahneman<sup>5</sup> 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.