The Bigger Picture

John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center
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)
- Transformer (Attention based)
- Bidirectional RNN
- Restricted Boltzmann Machine

Several of these are particularly common...
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

---

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](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](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 mtttaer 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 ervy 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 cut up the tree.
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).

```python
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.
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!), they have yet to find a central use in scientific problems.

Given the incredible effort and funding invested in these, it seems inevitable that the scientific community will find some way to leverage this capability.

Latest News.

Tesla has recently moved away from transformers to a diffusion based approach.
Autoencoder

Input Layer

Latent Features

Output Layer

Encoder

Decoder
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

Encoder

Latent Features

Bob Decoder

Alice Decoder
Deepfake At Work

Alice

Bob

Encoder

Latent Features

Bob Decoder
Zao Does DiCaprio

The Chinese app Zao did the below in 8 seconds from one photo.

twitter.com/AllanXia/status/1168049059413643265
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 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!
Hierarchical Text-Conditional Image Generation with CLIP Latents

Aditya Ramesh\textsuperscript{*}  
Pradulla Dhariwal  
Alex Nichol
OpenAI
arneen@openai.com  
pradulla@openai.com  
alax@openai.com

Casey Chu  
Mark Chen
OpenAI  
casey@openai.com  
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, 34, 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 shifts, 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, 26) 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 invert 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 accurate 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.
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

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.

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.
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
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.

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).
"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 DeepPMKit) 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.
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.

[GitHub link](https://github.com/gpuhackathons-org/gpubootcamp/blob/78e9fee3432b60348489682a978fa63f29f7e839/hpc_ai/ai_science_cfd/English/python/jupyter_notebook/CFD/Start_Here.ipynb)
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, King’s Buildings, Edinburgh, EH9 3JZ, UK.
2 Department of Mathematics, University of Cambridge, Cambridge, CB2 1QA, UK.
3 Department of Astronomy, University of Amsterdam, Amsterdam, 1098 XH, Amsterdam, The Netherlands.
4 Leiden Observatory, Leiden University, PO Box 9513, 2300 RA, Leiden, The Netherlands.

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 laboratory-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

Newtonian 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 global star systems and galactic models, which are thought to be the product of the collapse of giant black-hole binaries and single black holes (e.g., see [1], [2], [3], [4]).

This led to a recent trend in the study of complex systems that have been observed to deviate from the predictions of classical mechanics. These deviations are largely due to the chaotic nature of the underlying physical laws, which can lead to unpredictable outcomes even when the initial conditions are known with high precision. The complexity of these systems makes it difficult to obtain accurate solutions to the equations of motion, particularly when dealing with systems that are highly nonlinear or have multiple degrees of freedom.

The focus of the present study is to use the recent breakthroughs in the accuracy and efficiency of deep neural networks to address the long-standing challenge of solving the equations of motion for three-body systems using classical mechanics. This is achieved by training a deep artificial neural network (ANN) to accurately predict the trajectories of the three bodies over a given time interval, providing solutions that are orders of magnitude faster than existing numerical methods.

The main contributions of this work are:

- Development of a deep neural network architecture specifically designed for solving the equations of motion for three-body systems.
- Demonstration of the effectiveness of the proposed method through extensive numerical experiments.
- Comparison of the ANN solutions with existing numerical methods to validate the accuracy and efficiency of the approach.

The results illustrate a typical occurrence in ANN training, where there is an initial phase of rapid learning, followed by a period of slower learning in which the relative prediction gains are smaller with each epoch.
Has Deep Learning left any room for other approaches?

Big Data

AI

ML

DL

Neural Nets

Character Recognition

Capchas

Chess

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 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.

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.
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?
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.
Remember This?
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:


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

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


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.

<table>
<thead>
<tr>
<th>Package</th>
<th>Applications</th>
<th>Language</th>
<th>Strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>Neural Nets</td>
<td>Python, C++</td>
<td>Very popular.</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Neural Nets</td>
<td>Python (Lua)</td>
<td>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.</td>
</tr>
<tr>
<td>Spark MLLIB</td>
<td>Classification, Regression, Clustering, etc.</td>
<td>Python, Scala, Java, R</td>
<td>Very scalable. Widely used in serious applications. Lots of plugins to DL frameworks: TensorFrames, TF on Spark, CaffeOnSpark, Keras Elephas.</td>
</tr>
<tr>
<td>Scikit-Learn</td>
<td>Classification, Regression, Clustering</td>
<td>Python</td>
<td>Integrates well with TF to create powerful workflows.</td>
</tr>
<tr>
<td>Keras</td>
<td>Neural Nets</td>
<td>Python (on top of TF, Theano)</td>
<td>Now completely absorbed into TF.</td>
</tr>
<tr>
<td>Jax</td>
<td>Neural Nets</td>
<td>Python</td>
<td>Latest DeepMind (part of Google) framework. Missing pieces, but getting there. Similar to TF &amp; PT.</td>
</tr>
</tbody>
</table>
A note about hardware.
Inference Is Fast

Perceptual Labs

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

32MB SRAM
96x96 Mul/Add array
ReLU hardware
Pooling hardware
36 TOPS @ 2 GHz
2 per chip, 72 TOPS total
Everyone Doing Specialized Hardware

**NVIDIA**
- Turing/Hopper
- Tensor Cores

4x4 matrix mixed precision matrix multiply machines. 125 FP16 TFlops.

**Google**
- TPU

Cloud TPU v3
420 teraflops
128 GB HBM.

v4 is ~2X performance.

**Intel**
- Loihi
  (soon Loihi 2 at around 10X size and performance)

128-core, 130,000 artificial neurons, and 130 million synapses + 3 managing Lakemont cores.

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.
Everyone Doing Specialized Hardware

**Meta**
- MTIA
- Scrapped first generation inference chip and now working on a training capable chip.

**Microsoft**
- Athena
- Apparently undergoing testing now.

**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.
Demos

Ray-traced videogames! Recurrent CNN.

Style vs. Content

Deep Dream Generator

https://deepdreamgenerator.com/feed

Keras example at
https://keras.io/examples/deep_dream/
Loads a pre-trained ImageNet model!
This is a valuable capability.

Cool real-time Zelda demo:
https://www.youtube.com/watch?v=wou1dOlg2Fk
Demos

Style vs. Content: A little more subtle

Grab it at https://github.com/NVIDIA/FastPhotoStyle
A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras
NVIDIA
tkarras@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

Timo Aila
NVIDIA
tails@nvidia.com

Abstract

We propose an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair), and it enables intuitive, scale-specific control of the synthesis. The new generator improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties, and also better disentangles the latent factors of variation. To quantify interpolation quality and disentanglement, we propose two new, automated methods that are applicable to any generator architecture. Finally, we introduce a new, highly varied and high-quality dataset of human faces.

1. Introduction

The resolution and quality of images produced by generative methods—especially generative adversarial networks (GAN) [18]—have seen rapid improvement recently [26, 38, 5]. Yet the generators continue to operate as black boxes, and despite recent efforts [3], the understanding of various aspects of the image synthesis process, e.g., the origin of stochastic features, is still lacking. The properties of the latent space are also poorly understood, and the commonly demonstrated latent space interpolations [12, 45, 32] (e.g., pose, identity) from stochastic variation (e.g., freckles, hair) in the generated images, and enables intuitive scale-specific mixing and interpolation operations. We do not modify the discriminator or the loss function in any way, and our work is thus orthogonal to the ongoing discussion about GAN loss functions, regularization, and hyperparameters [20, 38, 5, 34, 37, 31].

Our generator embeds the input latent code into an intermediate latent space, which has a profound effect on how the factors of variation are represented in the network. The input latent space must follow the probability density of the training data, and we argue that this leads to some degree of unavoidable entanglement. Our intermediate latent space is free from that restriction and is therefore allowed to be disentangled. As previous methods for estimating the degree of latent space disentanglement are not directly applicable in our case, we propose two new automated metrics — perceptual path length and linear separability — for quantifying these aspects of the generator. Using these metrics, we show that compared to a traditional generator architecture, our generator admits a more linear, less entangled representation of different factors of variation.

Finally, we present a new dataset, Faces-HQ, FFHQ, that offers a considerably wider variety of resolution datasets (Appendix A) publicly available, along with pretrained networks. The accompanying video in http://stylegan.xyz/video

1. Style-based generator
What is reality?
C. Hyperparameters and training details

We build upon the official TensorFlow [1] implementation of Progressive GANs by Karras et al. [26], from which we inherit most of the training details. This original setup corresponds to configuration A in Table 1. In particular, we use the same resolution-dependent minibatch sizes, Adam [28] hyperparameters, and discriminator architecture. We enable mirror augmentation for both CelebA-HQ and FFHQ. Our training time is approximately one week on an NVIDIA DGX-1 with 8 Tesla V100 GPUs.

For our improved baseline (B in Table 1), we make several modifications to improve the overall result quality. We use the same 40 classifiers, one for each CelebA attribute, are used for measuring the separability metric for all generators. We will release the pre-trained classifier networks so that our measurements can be reproduced.

We do not use batch normalization [25], spectral normalization [38], attention mechanisms [55], dropout [51], or pixelwise feature vector normalization [26] in our networks.
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
A little more about

GPT-4
"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."

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
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: 10,000× less compute.

3.1 Loss Prediction

The final loss of properly-trained large models is independent of the amount of compute used to train them.

To verify the scalability of our optimization methods, we used a small, internal codebase (not part of the training, as in Henighan et al. [15]): $L(C) = I(C)$, but using at most 10,000× less compute started, without use of any partial resources, to achieve high accuracy (Figure 1).
<table>
<thead>
<tr>
<th>Exam</th>
<th>GPT-4</th>
<th>GPT-4 (no vision)</th>
<th>GPT-3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Bar Exam (MBE+MEE+MPT)</td>
<td>208 / 400 (~90th)</td>
<td>208 / 400 (~90th)</td>
<td>213 / 400 (~10th)</td>
</tr>
<tr>
<td>LSAT</td>
<td>163 (~88th)</td>
<td>161 (~83rd)</td>
<td>149 (~40th)</td>
</tr>
<tr>
<td>SAT Evidence-Based Reading &amp; Writing</td>
<td>710 / 800 (~93rd)</td>
<td>710 / 800 (~93rd)</td>
<td>670 / 800 (~87th)</td>
</tr>
<tr>
<td>SAT Math</td>
<td>700 / 800 (~89th)</td>
<td>600 / 800 (~989th)</td>
<td>590 / 800 (~70th)</td>
</tr>
<tr>
<td>Graduate Record Examination (GRE) Quantitative</td>
<td>163 / 170 (~80th)</td>
<td>157 / 170 (~62nd)</td>
<td>147 / 170 (~25th)</td>
</tr>
<tr>
<td>Graduate Record Examination (GRE) Verbal</td>
<td>169 / 170 (~99th)</td>
<td>165 / 170 (~95th)</td>
<td>154 / 170 (~62nd)</td>
</tr>
<tr>
<td>Graduate Record Examination (GRE) Writing</td>
<td>4 / 6 (~54th)</td>
<td>4 / 6 (~54th)</td>
<td>4 / 6 (~54th)</td>
</tr>
<tr>
<td>USAFO Semifinal Exam 2020</td>
<td>87 / 150 (99th - 100th)</td>
<td>87 / 150 (99th - 100th)</td>
<td>83 / 150 (71st - 100th)</td>
</tr>
<tr>
<td>USNCO Local Section Exam 2022</td>
<td>36 / 60</td>
<td>38 / 60</td>
<td>24 / 60</td>
</tr>
<tr>
<td>Medical Knowledge Self-Assessment Program</td>
<td>75 %</td>
<td>75 %</td>
<td>75 %</td>
</tr>
<tr>
<td>Codeforces Rating</td>
<td>392 (below 5th)</td>
<td>392 (below 5th)</td>
<td>260 (below 5th)</td>
</tr>
<tr>
<td>AP Art History</td>
<td>5 (86th - 100th)</td>
<td>5 (86th - 100th)</td>
<td>5 (86th - 100th)</td>
</tr>
<tr>
<td>AP Biology</td>
<td>5 (85th - 100th)</td>
<td>5 (85th - 100th)</td>
<td>5 (62nd - 85th)</td>
</tr>
<tr>
<td>AP Calculus BC</td>
<td>4 (43rd - 59th)</td>
<td>4 (43rd - 59th)</td>
<td>1 (6th - 7th)</td>
</tr>
<tr>
<td>AP Chemistry</td>
<td>4 (71st - 88th)</td>
<td>4 (71st - 88th)</td>
<td>2 (22nd - 48th)</td>
</tr>
<tr>
<td>AP English Language and Composition</td>
<td>2 (14th - 44th)</td>
<td>2 (14th - 44th)</td>
<td>2 (14th - 44th)</td>
</tr>
<tr>
<td>AP English Literature and Composition</td>
<td>2 (8th - 22nd)</td>
<td>2 (8th - 22nd)</td>
<td>2 (8th - 22nd)</td>
</tr>
<tr>
<td>AP Environmental Science</td>
<td>5 (91st - 100th)</td>
<td>5 (91st - 100th)</td>
<td>5 (91st - 100th)</td>
</tr>
<tr>
<td>AP Macroeconomics</td>
<td>5 (84th - 100th)</td>
<td>5 (84th - 100th)</td>
<td>2 (33rd - 48th)</td>
</tr>
<tr>
<td>AP Microeconomics</td>
<td>5 (82rd - 100th)</td>
<td>4 (60th - 82nd)</td>
<td>4 (60th - 82nd)</td>
</tr>
<tr>
<td>AP Physics 2</td>
<td>4 (66th - 84th)</td>
<td>4 (66th - 84th)</td>
<td>3 (30th - 66th)</td>
</tr>
<tr>
<td>AP Psychology</td>
<td>5 (83rd - 100th)</td>
<td>5 (83rd - 100th)</td>
<td>5 (83rd - 100th)</td>
</tr>
<tr>
<td>AP Statistics</td>
<td>5 (58th - 100th)</td>
<td>5 (85th - 100th)</td>
<td>3 (40th - 63rd)</td>
</tr>
<tr>
<td>AP US Government</td>
<td>5 (88th - 100th)</td>
<td>4 (77th - 88th)</td>
<td>4 (77th - 88th)</td>
</tr>
<tr>
<td>AP US History</td>
<td>5 (89th - 100th)</td>
<td>4 (74th - 89th)</td>
<td>4 (74th - 89th)</td>
</tr>
<tr>
<td>AP World History</td>
<td>4 (65th - 87th)</td>
<td>4 (65th - 87th)</td>
<td>4 (65th - 87th)</td>
</tr>
<tr>
<td>AMC 10</td>
<td>30 / 150 (6th - 12th)</td>
<td>36 / 150 (10th - 19th)</td>
<td>36 / 150 (10th - 19th)</td>
</tr>
<tr>
<td>AMC 12</td>
<td>60 / 150 (45th - 66th)</td>
<td>48 / 150 (19th - 40th)</td>
<td>30 / 150 (4th - 8th)</td>
</tr>
</tbody>
</table>

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.

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.
Why you really care...

Can this code?
From this different paper.
Impressive results on standard exam benchmarks.

Note how they **test** against unseen examples.
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.

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