DEEP LEARNING

Deep Learning

- CNNs – Convolutional Neural Networks
- Deep Supervised Networks with managed gradient approaches
  - Learning tricks: dropout, batch normalization, ResNets, etc.
- GANs
- Deep Reinforcement Learning
- Recurrent Networks - LSTM, GRU
- Transformers - GPT

Why Deep Learning

- Biological Plausibility – e.g. Visual Cortex
- Hastad proof - Problems which can be represented with a polynomial number of nodes with k layers, may require an exponential number of nodes with k-1 layers (e.g. parity)
- Highly varying functions can be efficiently represented with deep architectures
- Sub-features created in deep architecture can potentially be shared between multiple tasks
  - Type of transfer/multi-task learning

Early Work

- Fukushima (1980) – Neo-Cognitron
- LeCun (1989) – Convolutional Neural Networks (CNN)
  - Miminal interest at the time before other critical advances
- Many layered MLP with backpropagation
  - Tried early but without much success
    - Very slow
    - Vanishing gradient
  - More recent work demonstrated significant accuracy improvements by "patiently" training deeper MLPs with BP using fast machines (GPUs)
    - More general learning!
    - Much improved since 2012 with some important extensions to the original MLP/BP approach

More Efficient Deep Learning

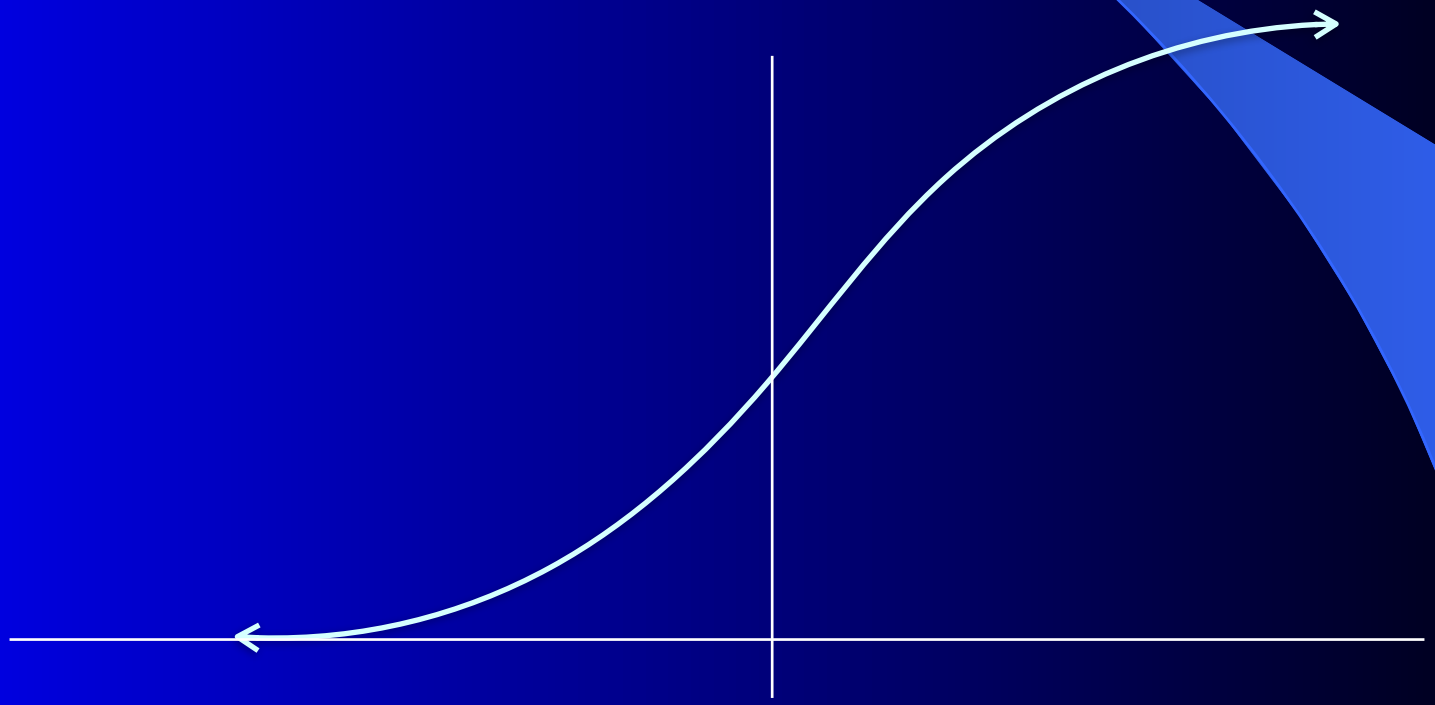
- Recent success in doing supervised deep learning with extensions that diminish the effect of previous difficulties (unstable gradient, etc.)
- Patience (now that we know it can be worth it), faster computers, and use of GPU/TPUs
- More efficient activation functions (e.g. ReLU) in terms of both computation and avoiding $f'(net)$ saturation
  - Also can be helpful to have 0 mean activations at each level, so sigmoid is frowned upon these days. If you want a saturating activation function, tanh is often preferred.
- Speed up and regularization approaches
- Improved Hyperparameters
- Batch Normalization – re-normalize activations at each layer
- Residual Nets

Vanishing/Unstable Gradient

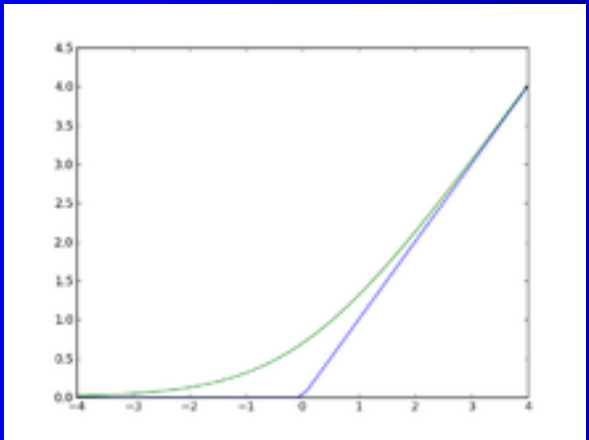
- Instability of gradient in deep networks: Vanishing *or* exploding gradient
  - Product of many terms, which unless "balanced" just right, is unstable
  - Either early or late layers stuck while "opposite" layers are learning
- Vanishing Gradient – error losses effect as it propagates to earlier layers – $t - z < 1$, $f'(net)$, scaled by small initial weights
- Leads to very slow training (especially at early layers when top layers saturate and have small $f'(net)$)
- Exacerbated since top couple layers can usually learn any task "pretty well" and thus the error to earlier layers drops quickly as the top layers "mostly" solve the task
- If lower layers never get the opportunity to use their capacity to improve results, they can just be stuck with their initial random weights

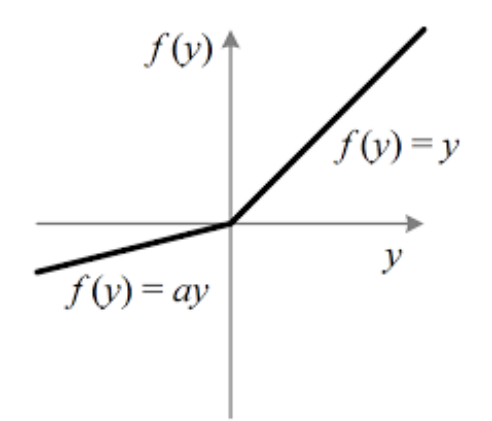
Vanishing/Exploding Gradient

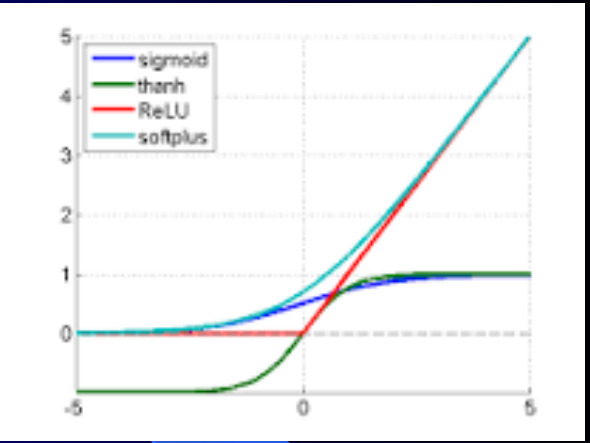
- Recent algorithmic improvements - Rectified Linear Units, better weight initialization, normalization between layers, residual deep learning, etc.



Rectified Linear Units







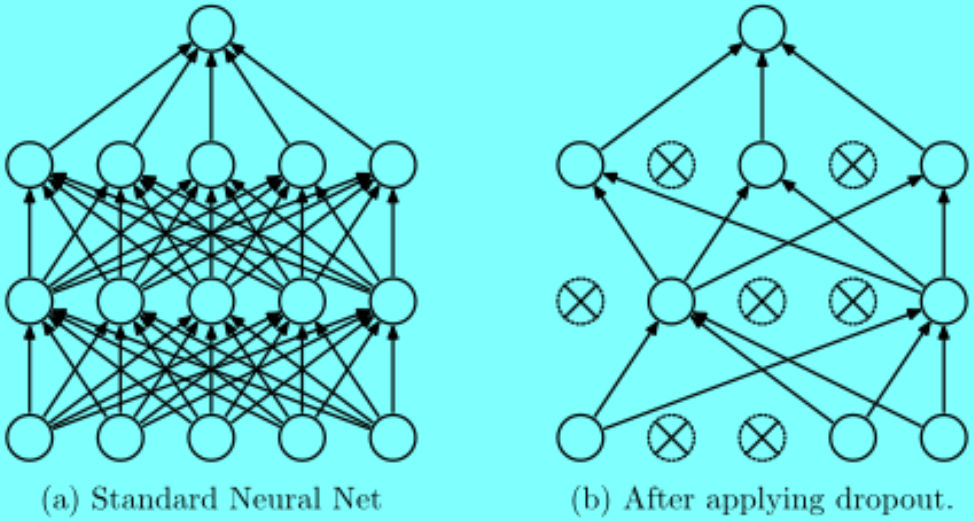
- $f(x)$ = Max(0,x) More efficient gradient propagation, derivative is 0 or constant, just fold into learning rate
  - Helps $f'(net)$ issue, but still left with other unstable gradient issues
- More efficient computation: Only comparison, addition and multiplication.
  - Leaky ReLU $f(x) = x$ if $x > 0$ else ax, where $0 \leq a <= 1$, so that derivate is not 0 and can do some learning for net < 0 (does not "die").
  - Lots of other variations
- Sparse activation: For example, in a randomly initialized network, only about 50% of hidden units are activated (having a non-zero output)
- Learning in linear range easier for most learning models

Speed up variations of SGD

- Use mini-batch rather than single instance for better gradient estimate – Helpful if using GD variation more sensitive to bad gradient, and *especially* for parallel implementations

- Momentum (i.e. Adaptive learning rate) approaches are important since anything to speed-up learning is helpful
  - Standard Momentum
    - Note the these approaches already do an averaging of grading also making mini-batch less critical
  - Nesterov Momentum – Calculate point you would go to if using normal momentum. Then, compute gradient at that point. Do normal update using *that* gradient and momentum.
  - R-prop – Resilient BP, if gradient sign inverts, decrease the node's individual LR, else increase it – common goal is faster in the flats, there are variants that backtrack a step, etc.
  - AdaGrad – Scale LRs inversely proportional to sqrt(sum(historical values)) – LRs with smaller derivatives are decreased less
  - RMS-prop – AdaGrad but uses exponentially weighted moving average, older updates basically forgotten
  - Adam (Adaptive moments) –Momentum terms on both gradient and squared gradient (1st and 2nd moments) – update based on both

Regularization – Dropout Common



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- For each instance drop a node (hidden or input) and its connections with probability *p* and train
- Final net just has all averaged weights (actually scaled by 1-*p* since that better matches the expected values at training time)
- As if ensembling 2^n different network substructures

Improved Initial Hyperparameter Settings

- Deep networks are more sensitive than shallow networks to hyperparameter settings
- More critical for deep learning in order to get more balanced learning across all layers
- Smaller LRs – patience
- To encourage sparsity sometimes initial biases set negative or more initial 0 weights are interspersed
- Initial weights – initialize a little larger in effort to find a balance which learns well across all layers. Common is to select initial weights from a uniform distribution between
  - -*c*/root(node fan-in), *c*/root(node fan-in) (*c* = 1 Xavier, *c* = 2 He)
  - -*c*/root(node fan-in + fan-out), *c*/root(node fan-in + fan-out)
    - Can do Gaussian distribution with above as variances
    - Lots of other variations and current work

Batch Normalization (2015)

- Rather than just guess initial parameters to maintain learning balance, renormalize activations at each layer to maintain balance
- Just like it is critical to normalize our initial features, we consider each layer to be a new feature set which can also be normalized
- We like 0-mean and unit variance inputs (standard 1st layer normalization), we can just re-normalize the net value (less commonly the activation value) for each input dimension *k* at each layer

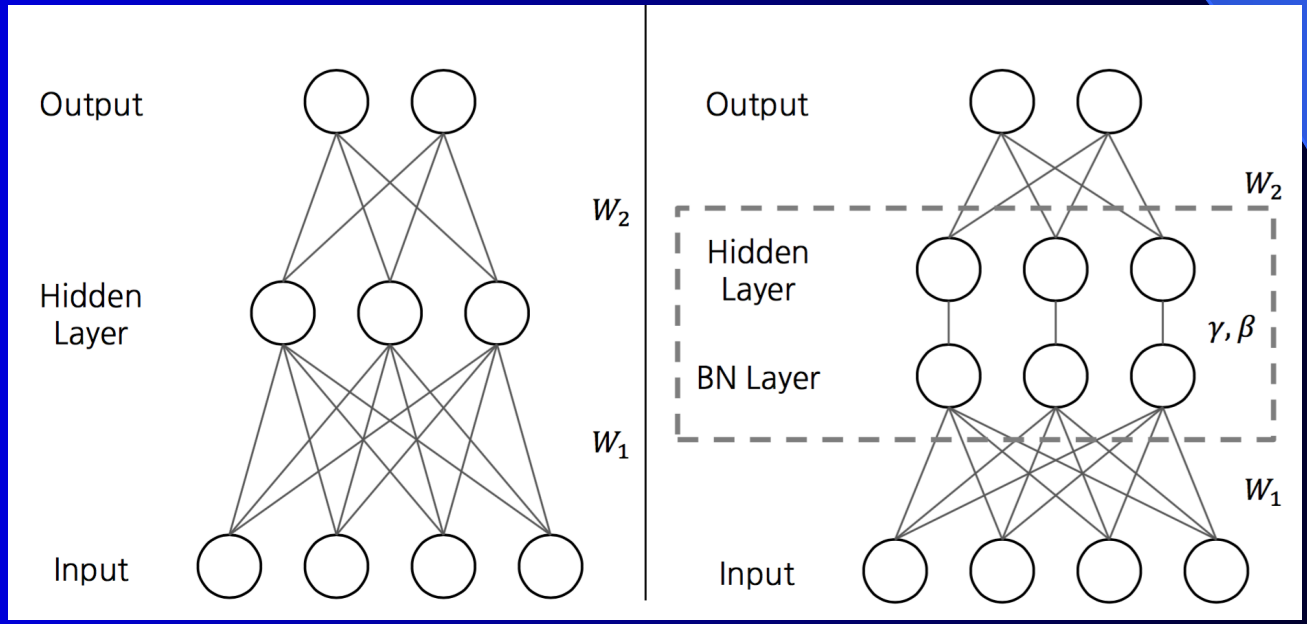
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Think of each net value as a new feature. Want mean and variance of that activation for the entire data set. Changing! Approximate the empirical mean and variance over a mini-batch of instances.
- Goes beyond standard normalization by allowing scaling and shifting of the normalized values with 2 learnable weights per input, γ and β, to attain the final batch normalization (allows recovery of initial or any needed function)

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Batch Normalization

- Typically normalize before the non-linearity (e.g. ReLU)
- Allows larger LR, improves gradient flow and reduces dependence on initialization
- Doubles the layers, giving more learnable parameters



Layer Normalization (2016)

- Shortly after batch normalization (Ba, Kiros, and Hinton)
- Rather than normalize a single feature across a batch, normalize the net value of each feature across the layer (the width of the feature vector)
- No need of a batch, do normalization for each instance
- Exact same at learning and execution time
- More common for Recurrent neural networks and transformers

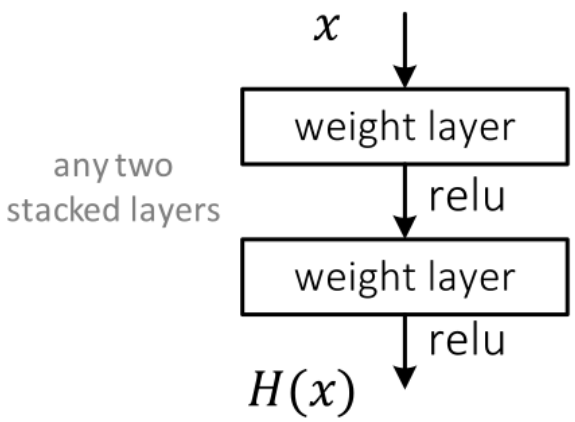
Deep Residual Learning

- Residual Nets – 100s of layers
- 2015 ILSVRC winner
  - CNN
  - Also used Batch Normalization
- Learns the residual mapping with respect to the identity – i.e. the *difference* (residual) between the current input and the goal mapping
- Simple concept which tends to make the function to be learned simpler across depth

Deep Residual Learning

Deep Residual Learning

- Plain net



$H(x)$ is any desired mapping,

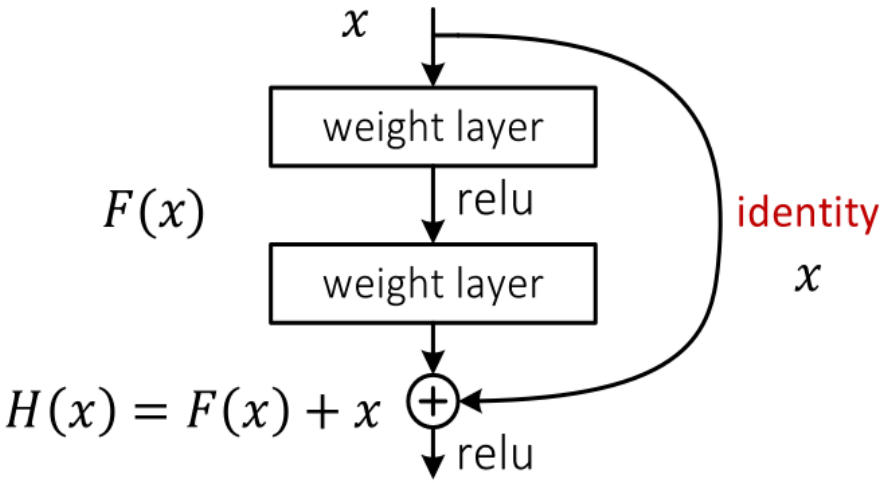
hope the 2 weight layers fit $H(x)$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Deep Residual Learning

Deep Residual Learning

- $F(x)$ is a residual mapping w.r.t. identity



- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

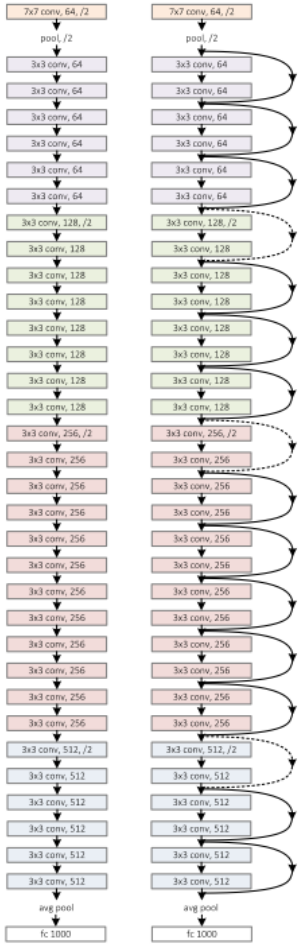
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Deep Residual Learning

Network "Design"

- Keep it simple
- Our basic design (VGG-style)
  - all 3x3 conv (almost)
  - spatial size /2 => # filters x2 (~same complexity per layer)
  - Simple design; just deep!
- Other remarks:
  - no hidden fc
  - no dropout

plain net



ResNet

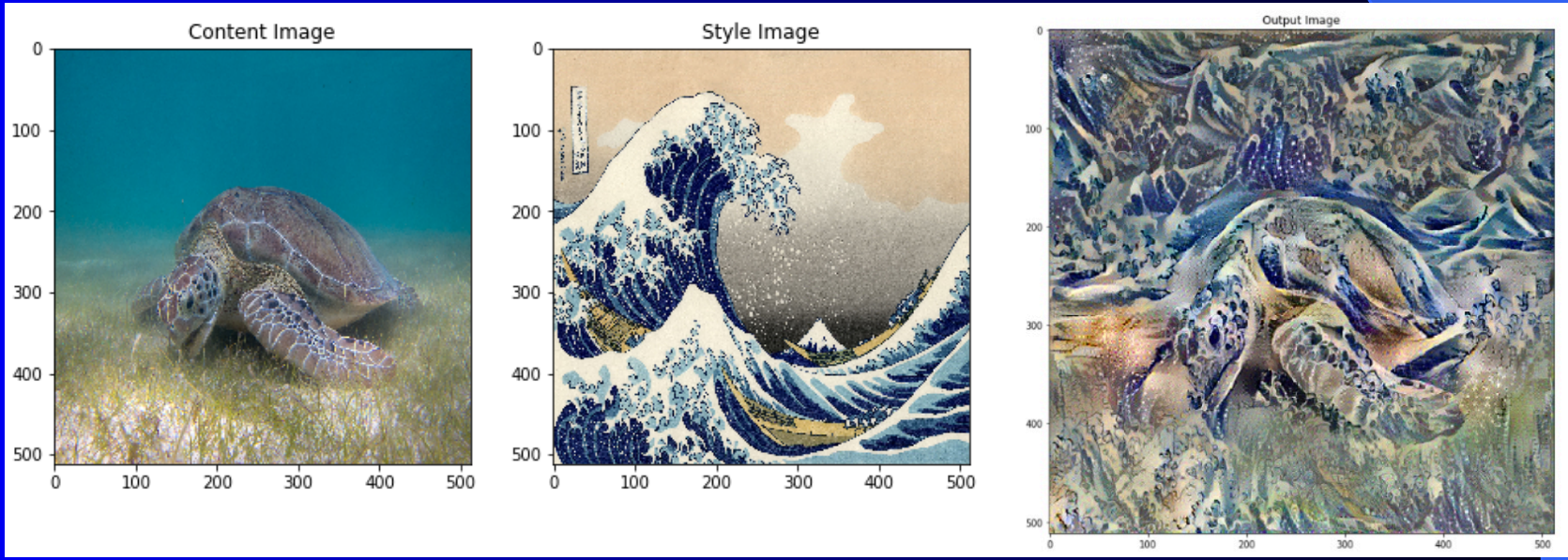
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Generative Neural Networks

DEEP LEARNING

Neural Style Transfer

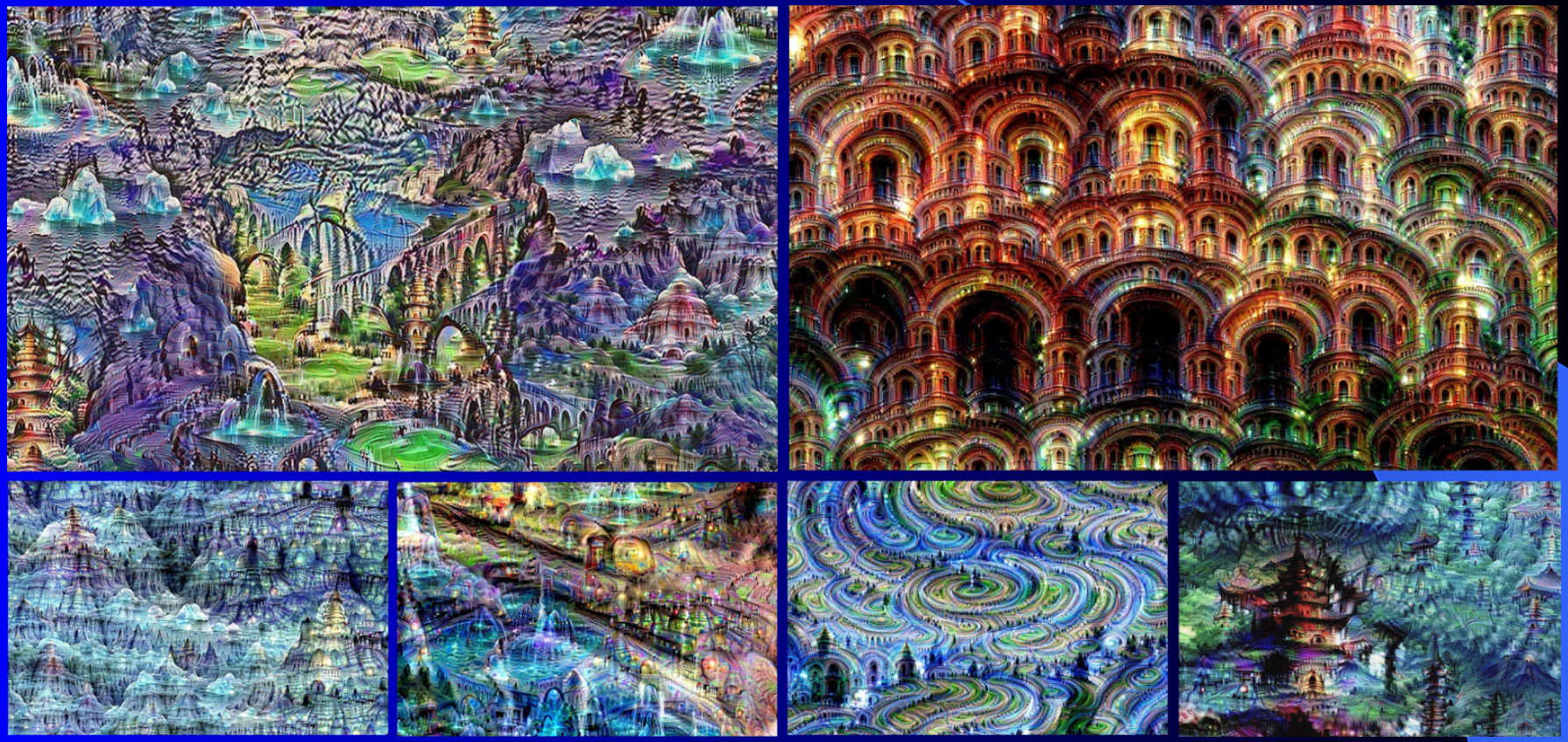
- Train on lots of images and styles
- CNN trained with two loss functions
  - content and style
- Then supply any content and style image
- Creates new image of content, but in the style of the style image



Neural Style Transfer

Neural Style Transfer

Deep Dreaming



Real vs Fake?

President Russel M Nelson:

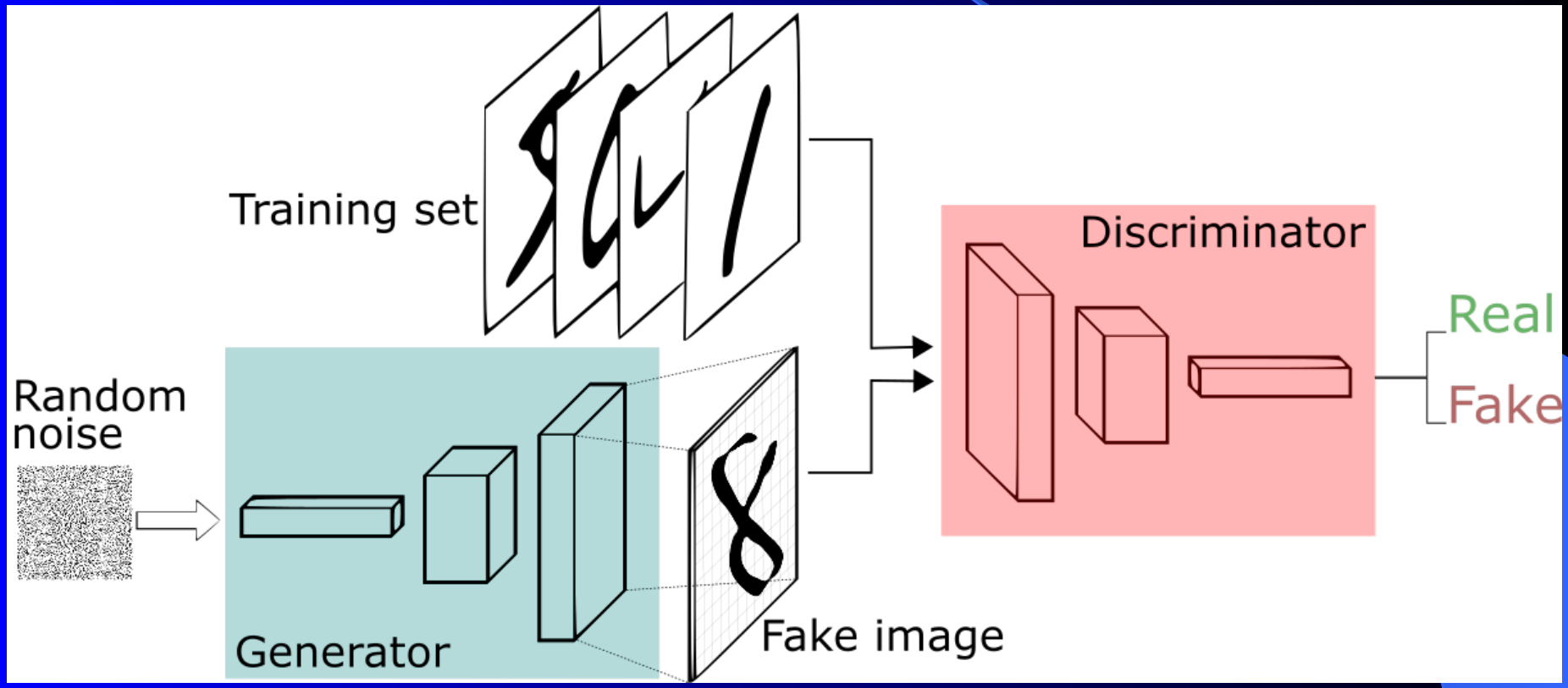
"If we are to have any hope of sifting through the myriad of voices and the philosophies of men that attack truth, we must learn to receive revelation.

In coming days, it will not be possible to survive spiritually without the guiding, directing, comforting, and constant influence of the Holy Ghost."

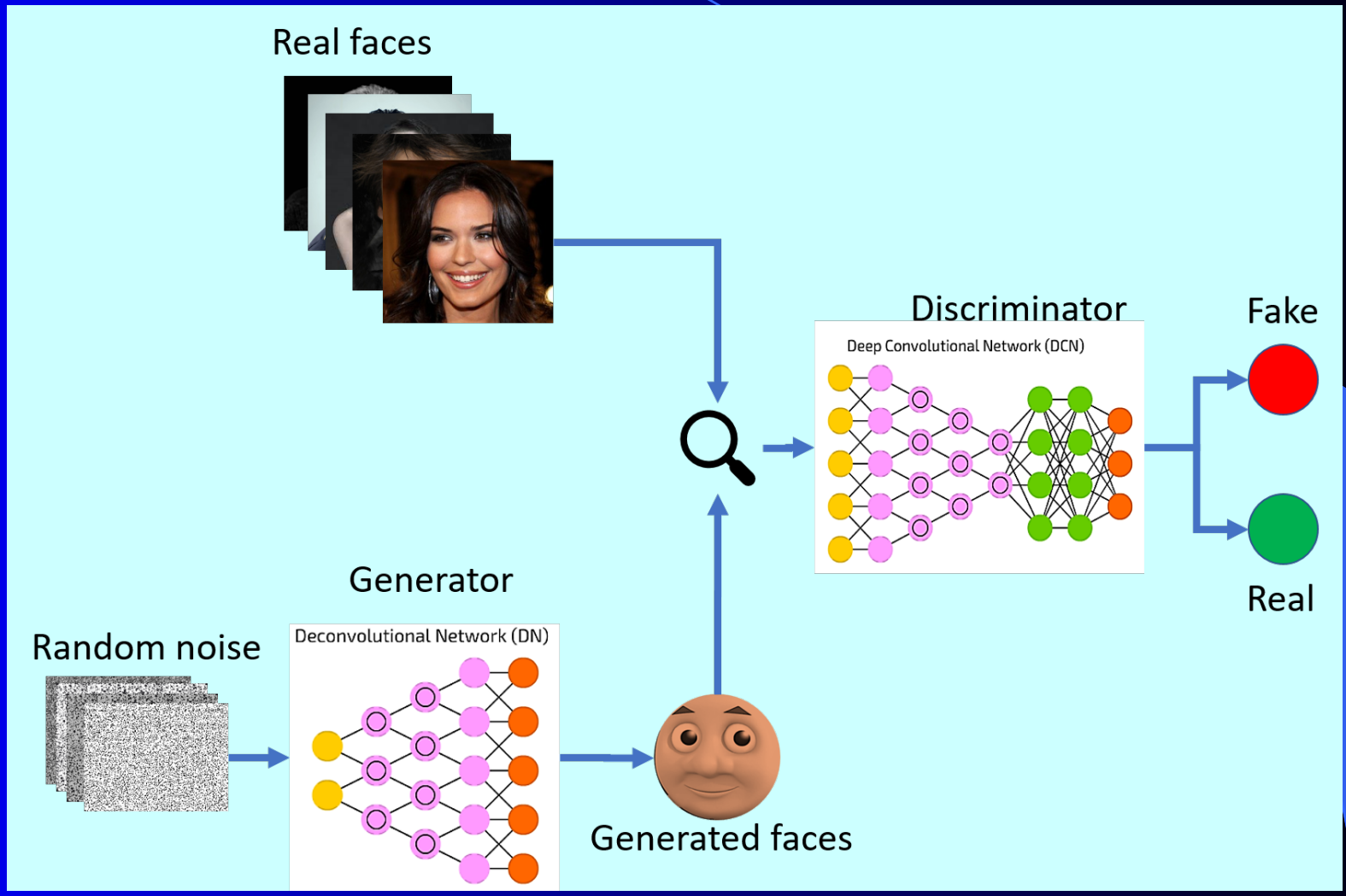
Generative Adversarial Networks GANs (2014 Ian Goodfellow)

- Unsupervised in that no labels needed
- Generative networks which generate novel samples similar to training samples (images, text, etc.)
- Discriminative net (adversary) must differentiate between samples from the generative net and the training set
- Use loss feedback on discriminator net to create gradient for both nets, until discriminator can no longer distinguish, then can discard discriminator net – increasingly difficult for humans to distinguish
- “Universal loss function" for lots of "difficult/creative" applications

GAN - Generative Adversarial Network



GAN - Generative Adversarial Network



GAN – Celebrity Data Set (2017)

GAN 2.0 NVIDIA (2018) - Improved Face Generator

Edmond de Belamy

- Created by a GAN and sold at auction in 2018 for $432,500
- Note the author inscription – it is the GAN loss function

GauGan2 ’21:

Create your own images by sketching or by just typing in a description of the image you want to create

An output from GauGAN2 for the phrase "coast ripples cliffs."

GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models

"a hedgehog using a calculator"

"a corgi wearing a red bowtie and a purple party hat"

"robots meditating in a vipassana retreat"

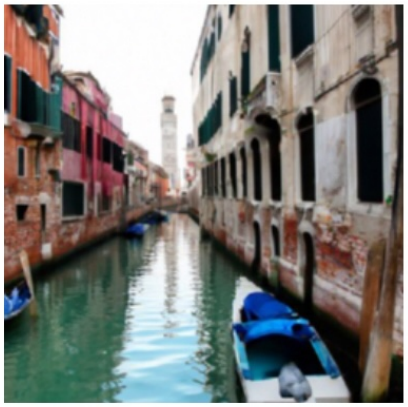
"a fall landscape with a small cottage next to a lake"

"a surrealist dream-like oil painting by salvador dalí of a cat playing checkers"

"a professional photo of a sunset behind the grand canyon"

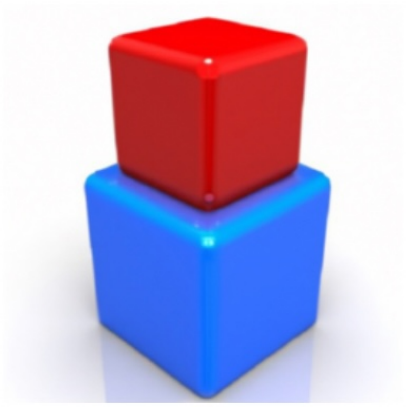
"a high-quality oil painting of a psychedelic hamster dragon"

"an illustration of albert einstein wearing a superhero costume"

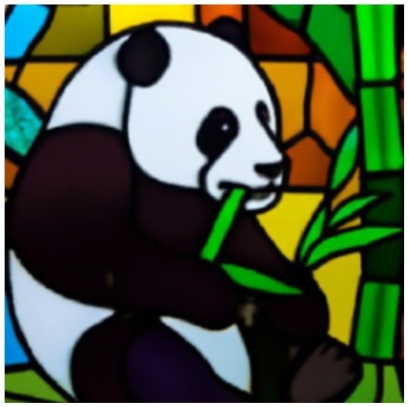


"a boat in the canals of venice"

"a painting of a fox in the style of starry night"



"a red cube on top of a blue cube"



"a stained glass window of a panda eating bamboo"

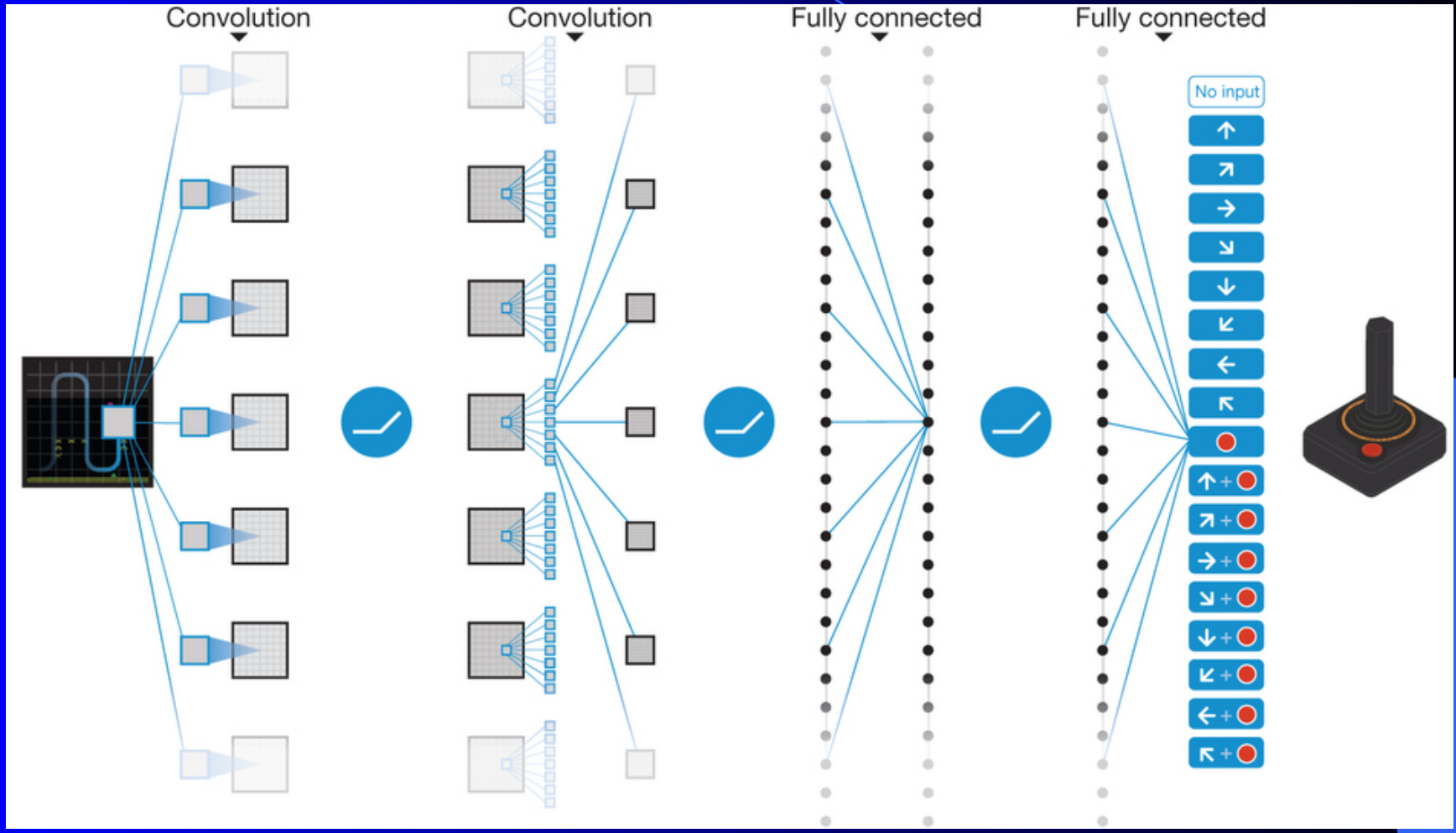
"a crayon drawing of a space elevator"

"a futuristic city in synthwave style"

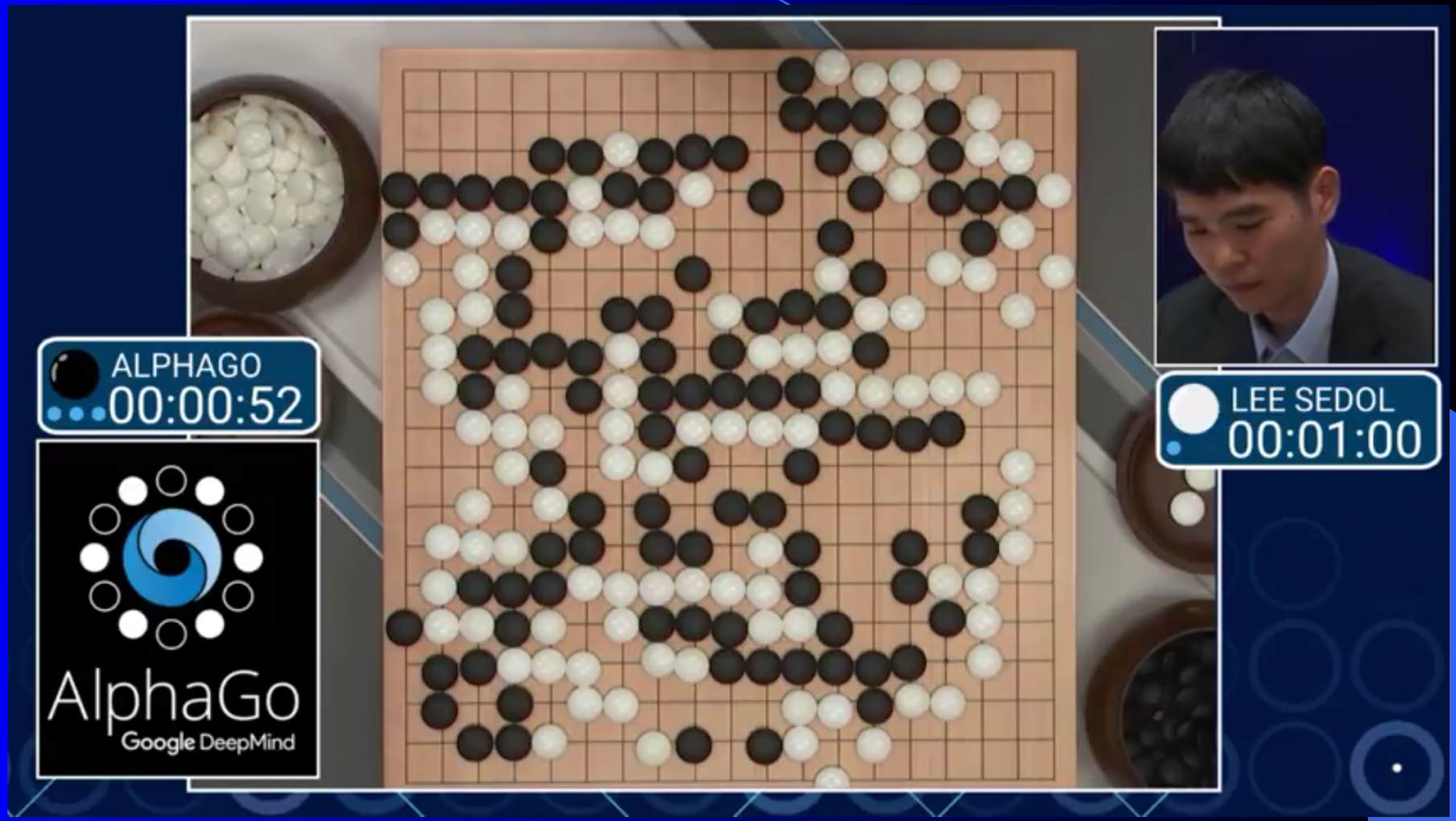
"a pixel art corgi pizza"

"a fog rolling into new york"

Deep Reinforcement Learning



AlphaGo - Google DeepMind



Alpha Go

- Reinforcement Learning with Deep Net learning the value and policy functions
- Challenges world Champion Lee Se-dol in March 2016
  - AlphaGo Movie – Netflix, check it out, fascinating man/machine interaction!
- AlphaGo Master (improved with more training) then beat top masters on-line 60-0 in Jan 2017
- 2017 – Alpha Go Zero
  - Alpha Go started by learning from 1000's of expert games before learning more on its own, and with lots of expert knowledge
  - Alpha Go Zero starts from zero (Tabula Rasa), just gets rules of Go and starts playing itself to learn how to play – not patterned after human play – More creative
  - Beat AlphaGo Master 100 games to 0 (after 3 days of playing itself)

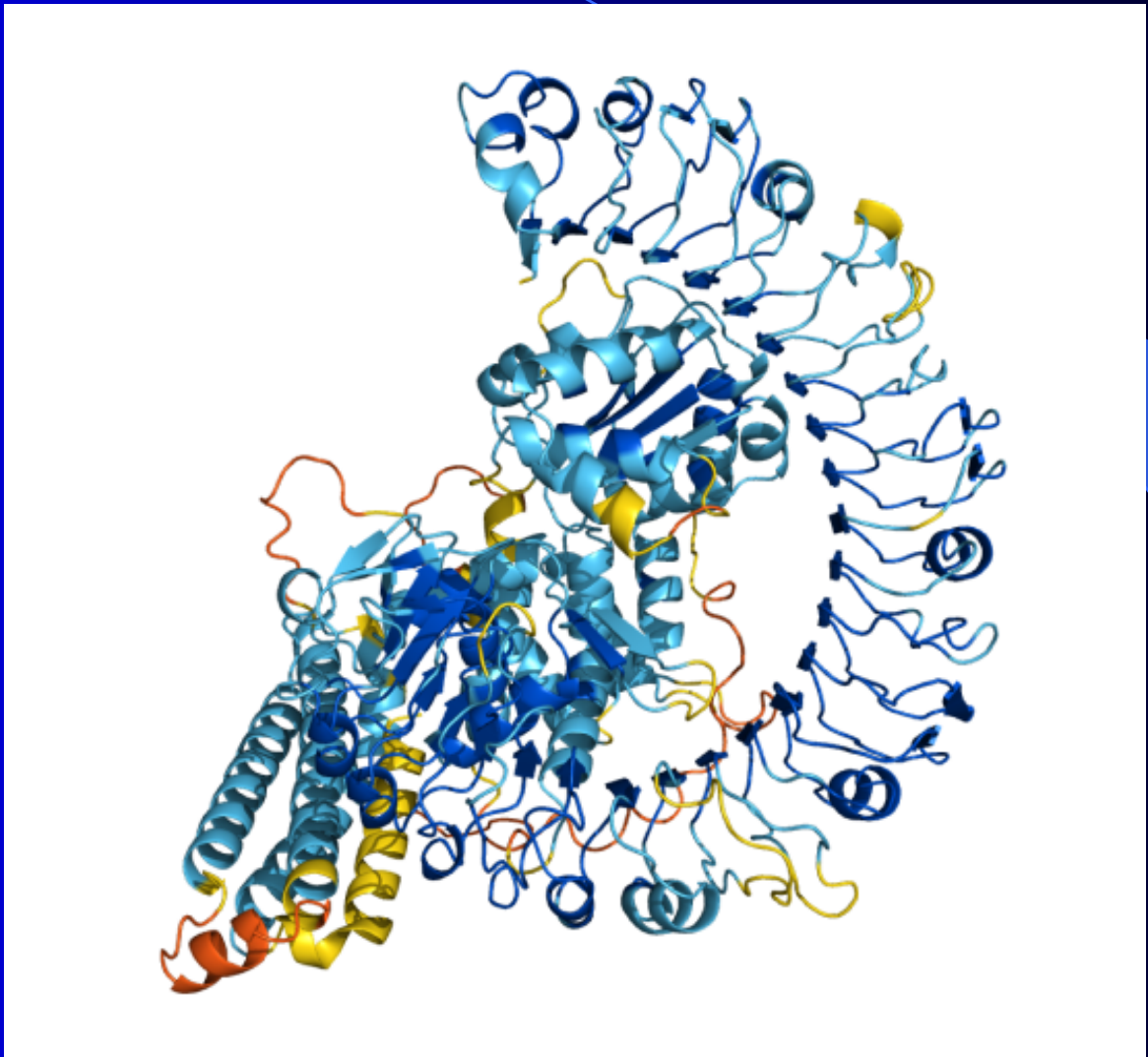
Alpha Zero

- Alpha Zero (late 2017)
- Generic architecture for any board game
  - Compared to AlphaGo (2016 - earlier world champion with extensive background knowledge) and AlphaGo Zero (2017)
- No input other than rules and self-play, and not set up for any specific game, except different board input
- With no domain knowledge and starting from random weights, beats worlds best players and computer programs (which were specifically tuned for their games over many years)
  - Go – after 8 hours training (44 million games) beats AlphaGo Zero (which had beat AlphaGo 100-0) – 1000's of TPU's for training
    - AlphaGo had taken many months of human directed training
  - Chess – after 4 hours training beats Stockfish8 28-0 (+72 draws)
    - Doesn't pattern itself after human play
  - Shogi (Japanese Chess) – after 2 hours training beats Elmo

AlphaStar

- DeepMind considered "perfect information" board games solved
- Next step was – Starcraft II - AlphaStar
  - Considered a next "Grand AI Challenge"
  - Complex, long-term strategy, stochastic, hidden info, real-time
  - Beats best Pros - AlphaStar limited to human speed in actions/clicks per minute – so just comparing strategy

AlphaFold – Moved to Transformers, World's most accurate protein folding



Recurrent Neural Networks

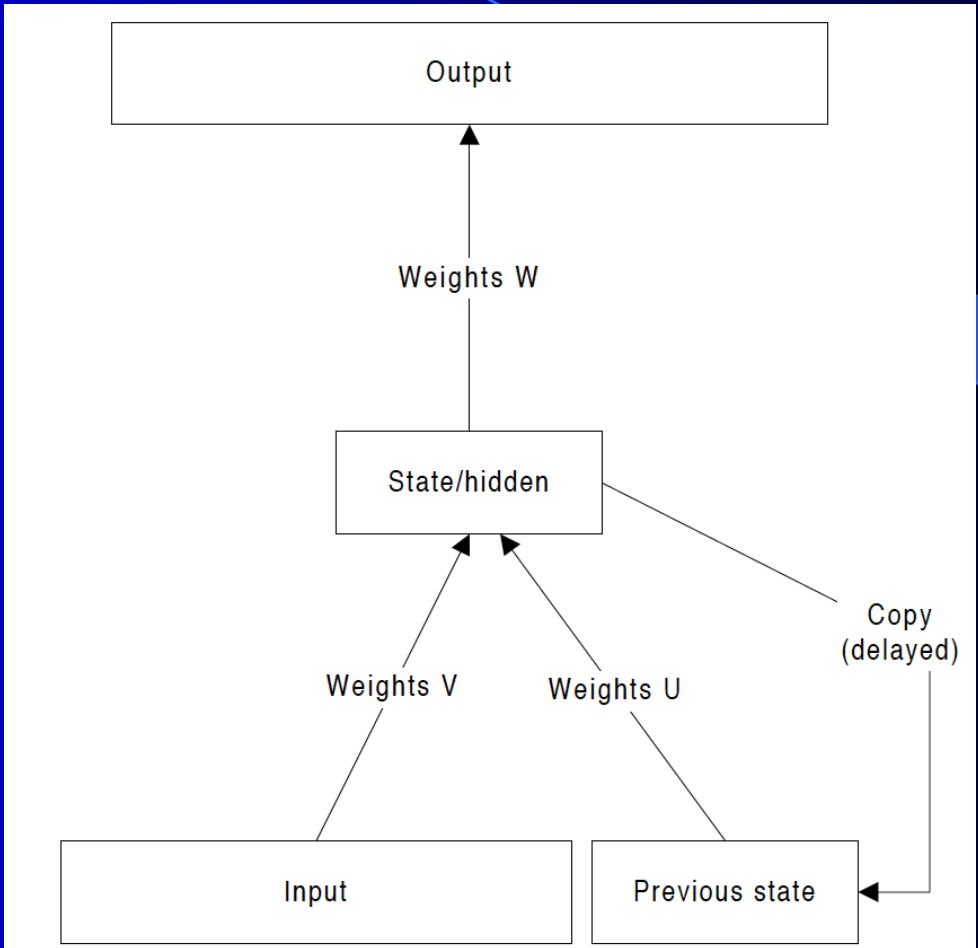
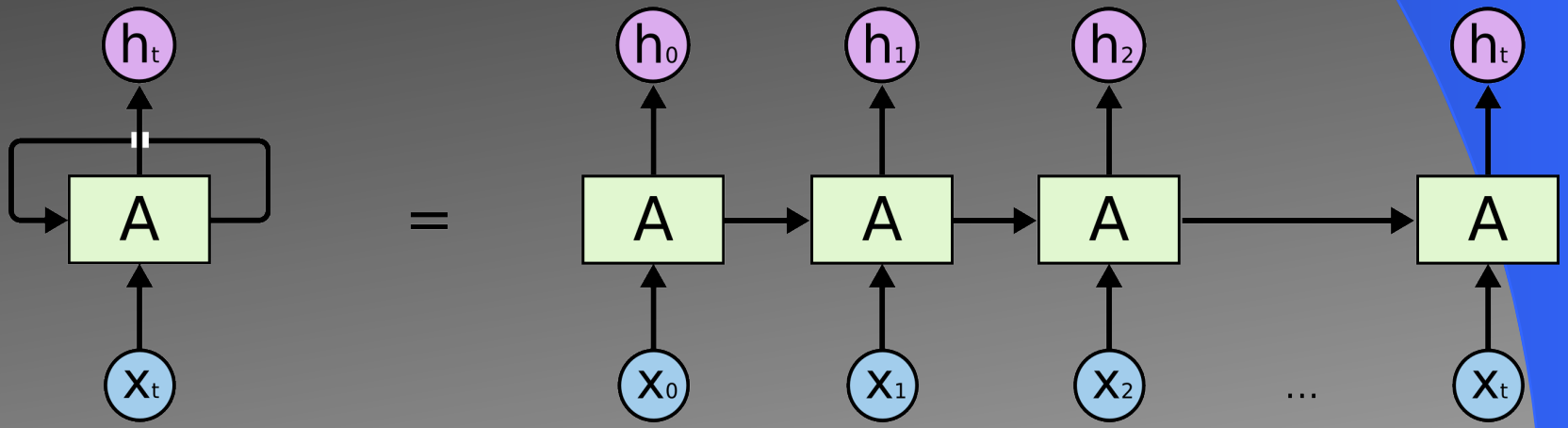


Figure 4: A simple recurrent network.

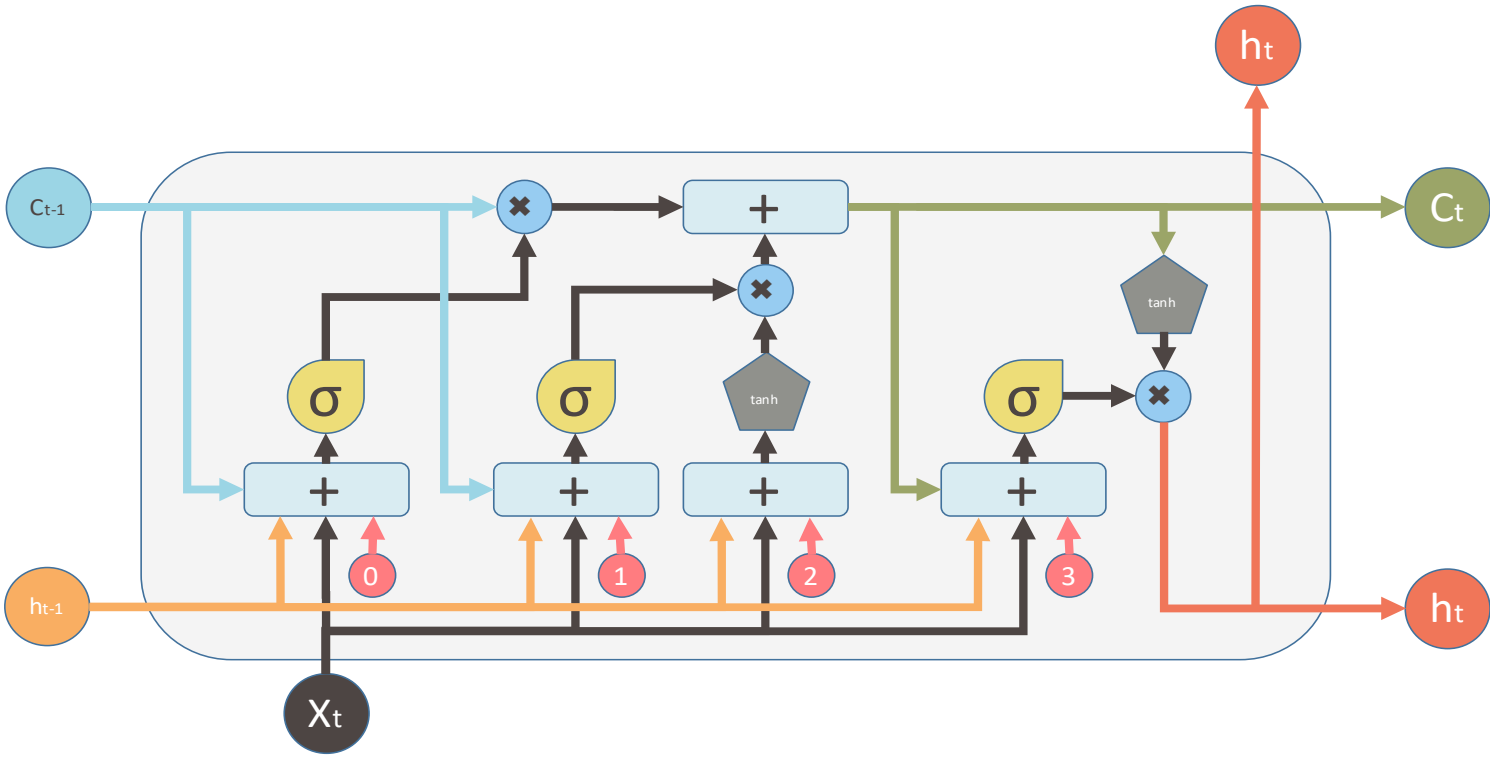
Unfolding Recurrent Net in Time

- Can consider an equivalent feedforward network unfolded *t* steps in time
- Then can train it as if it was a regular feedforward network – Backpropagation through Time (BPTT)
- Only difference is that the weights are tied (use average)
- Becomes a deep net in time – has vanishing gradient issues



LSTM – Long Short-Term Memory

- LSTM unit can just plug in for a standard node in an RNN
- Adds a state memory plus input, forget, and output gates
- Vanishing/exploding gradient avoidance
  - Cell value (state) has a self-feedback loop and can maintain its value indefinitely. Has derivate 1 with no vanishing gradient.
  - The cell value is multiplied by a forget gate output (0-1), which decides when and how much to forget, giving much more power.
- LSTM unit has lots more parameters but still trained with standard BP/SGD learning: typically BPTT
  - Forget gates, etc. don't know that is their job, but the capacity is there during training to learn that job as the overall network minimizes loss
  - Capacity plus training finds a way to solve the problem, capacity that wasn't there with simple network nodes



Inputs:

- X_t Input vector
- C_{t-1} Memory from previous block
- h_{t-1} Output of previous block

outputs:

- C_t Memory from current block
- h_t Output of current block

Nonlinearities:

- σ Sigmoid
- tanh Hyperbolic tangent

Vector operations:

- ✖ Element-wise multiplication
- \+ Element-wise Summation / Concatenation

Bias: 0

Transformers

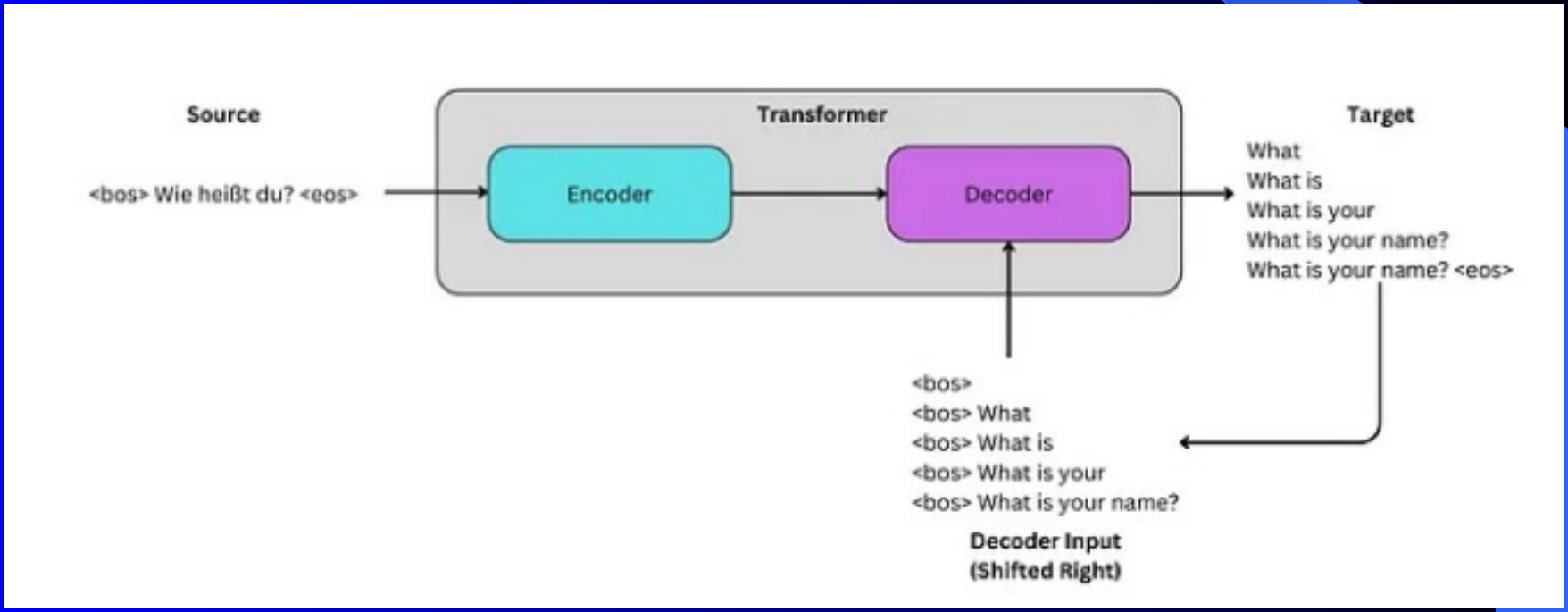
Transformers - GPT

- Replacing CNNs and RNNs in many cases
- Rather than use convolutional filters or recurrence to find and track relationships they use *attention*
- We need short and long-distance relationships between features (e.g. words, pixels, atoms, etc.). What does "it" in a following sentence refer to?
- Initial paper – Google 2017 - "Attention is all you need"
- Immediately followed by lots of variations such as BERT (Bidirectional Representations from Transformers)
- GPT 1-3 are basically the same 2017 Google model
- More parameters/weights make a huge difference
  - GPT 2: 1.5B parameters, GPT 3 same basic architecture as GPT 2: 175B parameters

Transformers: “Attention is all you need”

- Autoregressive – use the encoder embedding and the words inferred so far by the decoder as input to decide the next word

- For training, the next word is the current target



Transformer

- Full input sentence entered (Tokens)
- Learns embedding
- Adds position
- Skip connections and layer normalizations
- Weights how much certain words in the sequence are tied to others
- Allows more distant dependencies
- Self-Attention and encoder-decoder attention
- Masked on decoder so only considers past tokens
- Can also just use Encoder and Decoder models

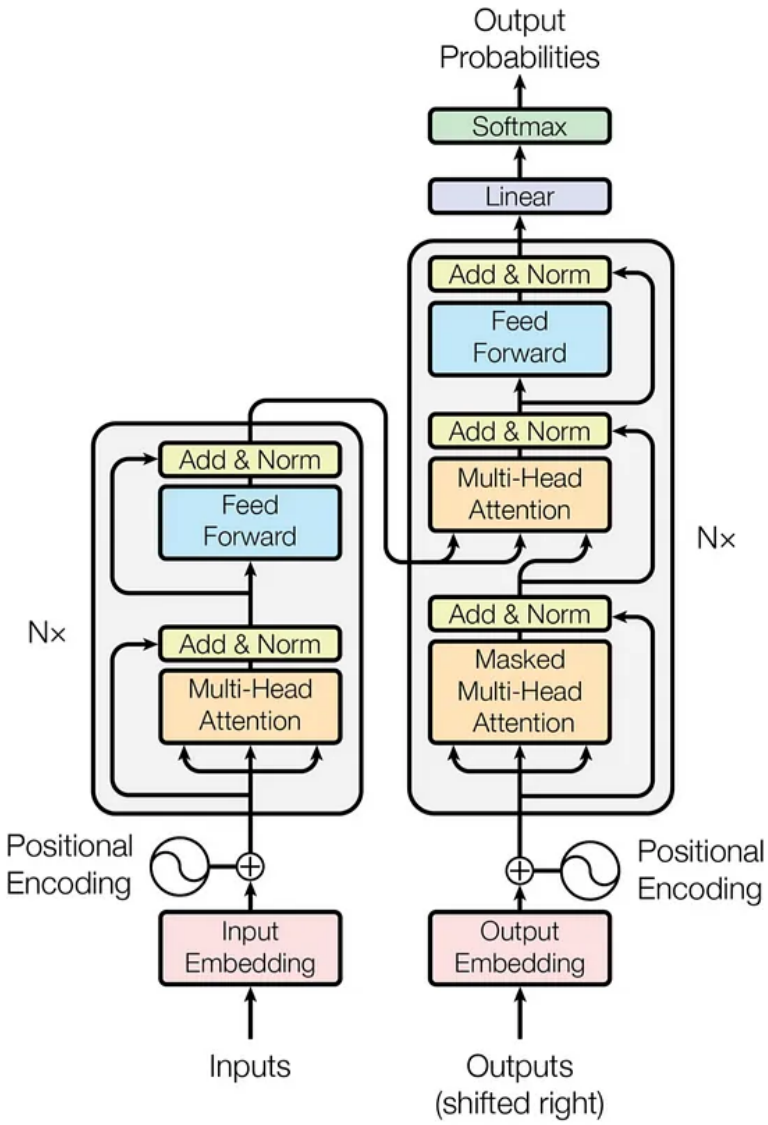
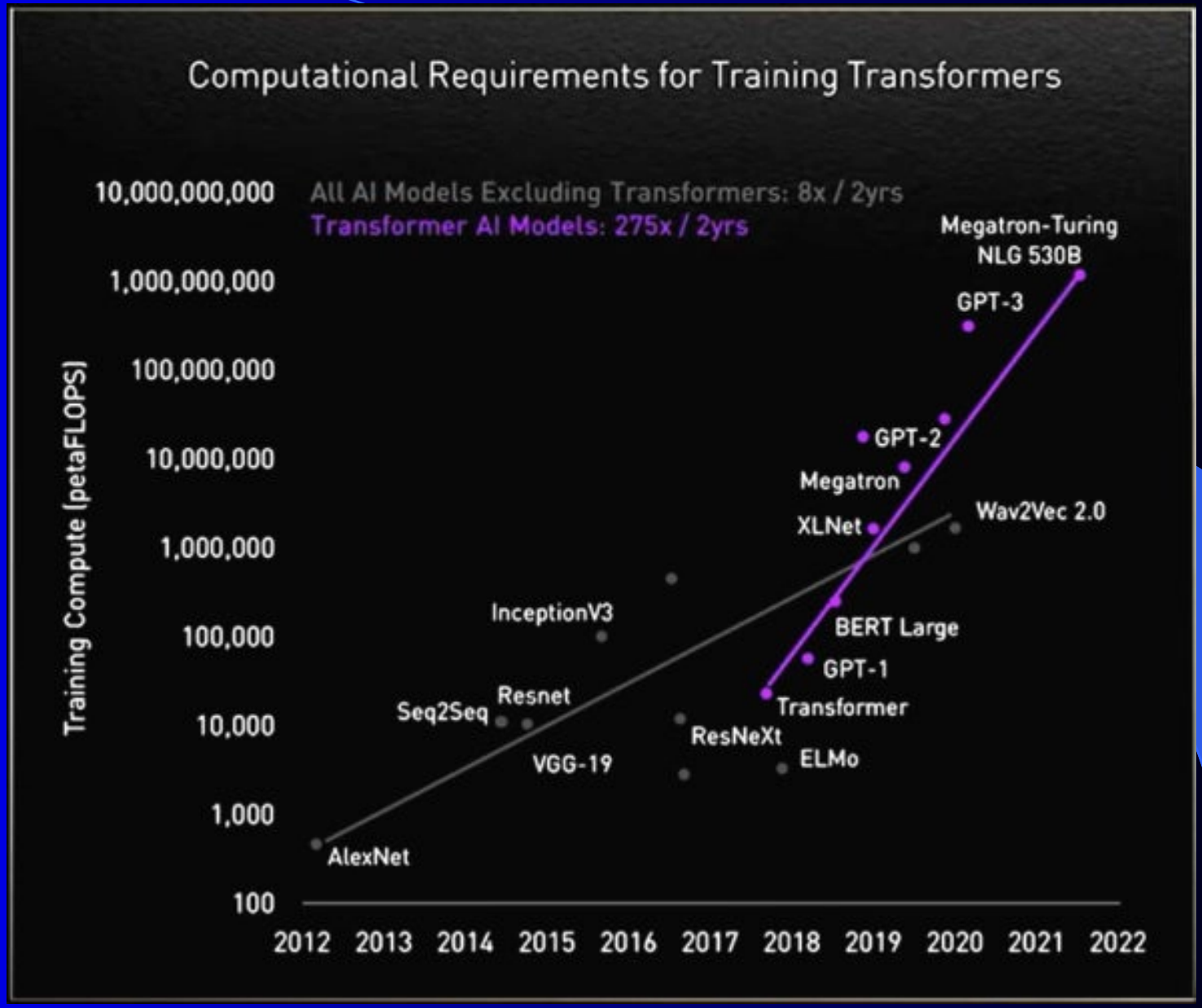


Figure 1: The Transformer - model architecture.

Transformers Intro

- Long training (though with some improved efficiencies over CNNs), high power hardware, and lots of training data – big companies right now, Google, NVIDIA, IBM, Microsoft, etc. coming out with their own versions
  - Now developing trillion parameter models
  - Latest work is increasing the ability for models to learn faster and better
- Many parameters make them challenging to run locally
- Still not doing real reasoning, just finds patterns from lots of data and mimics those
  - Which is still pretty cool
  - Don't overestimate current wisdom of content, but it will definitely "appear" like human style output
- A future goal is to learn more like humans with less data needed


Computational Requirements for Training Transformers
All AI Models Excluding Transformers: 8x / 2yrs
Transformer AI Models: 275x / 2yrs
Training Compute (petaFLOPS)
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
100
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
Megatron-Turing NLG 530B
GPT-3
GPT-2
Megatron
XLNet
Wav2Vec 2.0
InceptionV3
BERT Large
GPT-1
Transformer
Seq2Seq
Resnet
ResNeXt
ELMo
VGG-19
AlexNet

GPT (Generative Pre-trained Transformer)

- OpenAI– Some of the original Google authors, Elon Musk and others fund, Microsoft funding and Partnership
- Decoder only transformer (just right side of transformer model with the encoder-decoder attention head dropped)
- Autoregressive: Initialize with an initial sequence (the query), then repeatedly output the next word
- Does not need labeled data! Finds associations while training with large amounts of unlabeled data. Just uses next word in text as target during training.
- GPT-2 1.5B, GPT-3 similar to GPT-2 but with 175 B parameters, 800GB to store
- Pre-trained with a diverse corpus of unlabeled text datasets from web scraping (WebText)
- No fine tuning necessary (though you can for specific tasks)

GPT/Transformer Advances

- GPT 3.5
  - Adds RLHF (Reinforcement Learning from Human Feedback) - Turbo
  - 4K token span (vs 2K for GPT 3)
- GPT 4
  - Multi-modal: Images, voice as part of queries and responses
  - ~1.7 Trillion parameters
  - 32K token span
- Other large companies building competitive models (Google Bard, Microsoft Bing, etc.)
- Pretty amazing, but still not doing deep understanding, rather an amazingly complex statistical *n*-gram model – In language, coding, etc., it models what humans have demonstrated in training data. (i.e. don't expect it to give new wisdom)

Deep Learning Conclusion

- Much recent excitement, still much to be discovered
- Impressive results
- More work needed to understand how and why deep learning works so well – How deep should we go?
- Potential for significant improvements
- Works well in features spaces with local correlations in space and/or time – CNNs, RNNs, Transformers.
  - Important research question: To what extent can we use Deep Learning in arbitrary feature spaces?