Introduction to Scientific Machine Learning and Deep Learning

or

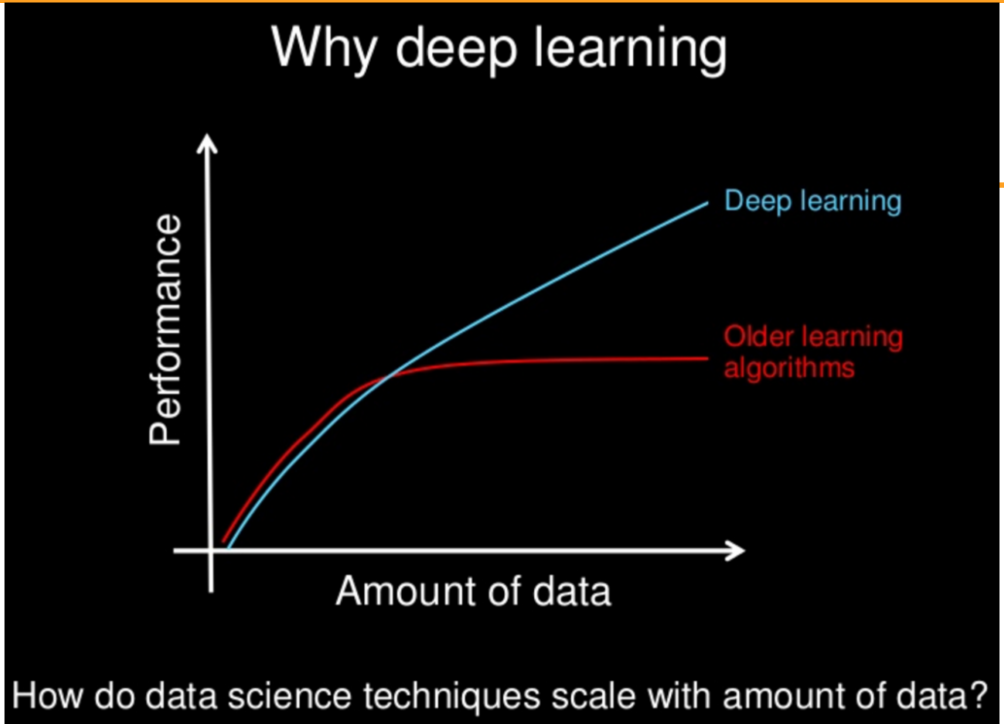
“deep neural networks and all of its friends”

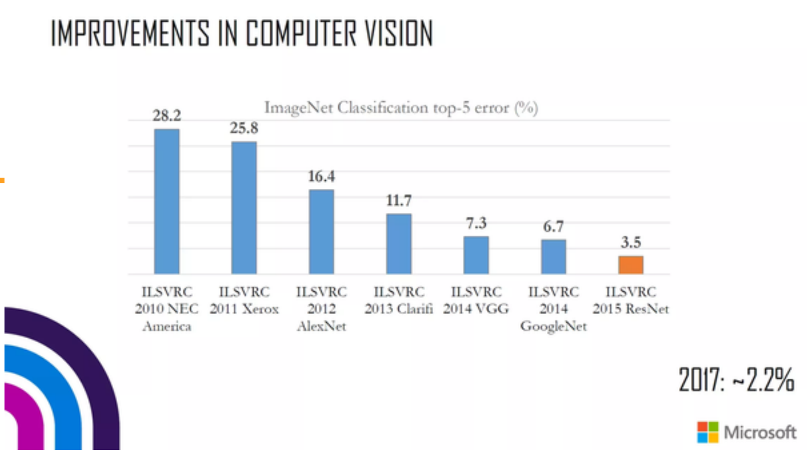
R. Michael Churchill, *PPPL*

PPPL PRINCETON PLASMA PHYSICS LABORATORY

PRINCETON UNIVERSITY

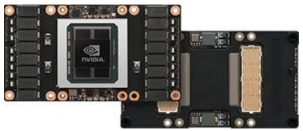
Artificial intelligence is the science and engineering of making computers behave in ways that, until recently, we thought required human intelligence – Andrew Moore, *Forbes Magazine 2017*


Why deep learning
Performance
Deep learning
Older learning algorithms
Amount of data
How do data science techniques scale with amount of data?


IMPROVEMENTS IN COMPUTER VISION
ImageNet Classification top-5 error (%)
28.2
25.8
16.4
11.7
7.3
6.7
3.5
ILSVRC 2010 NEC America
ILSVRC 2011 Xerox
ILSVRC 2012 AlexNet
ILSVRC 2013 Clarifi
ILSVRC 2014 VGG
ILSVRC 2014 GoogleNet
ILSVRC 2015 ResNet
2017: ~2.2%
Microsoft


TRAVERSE

NVIDIA



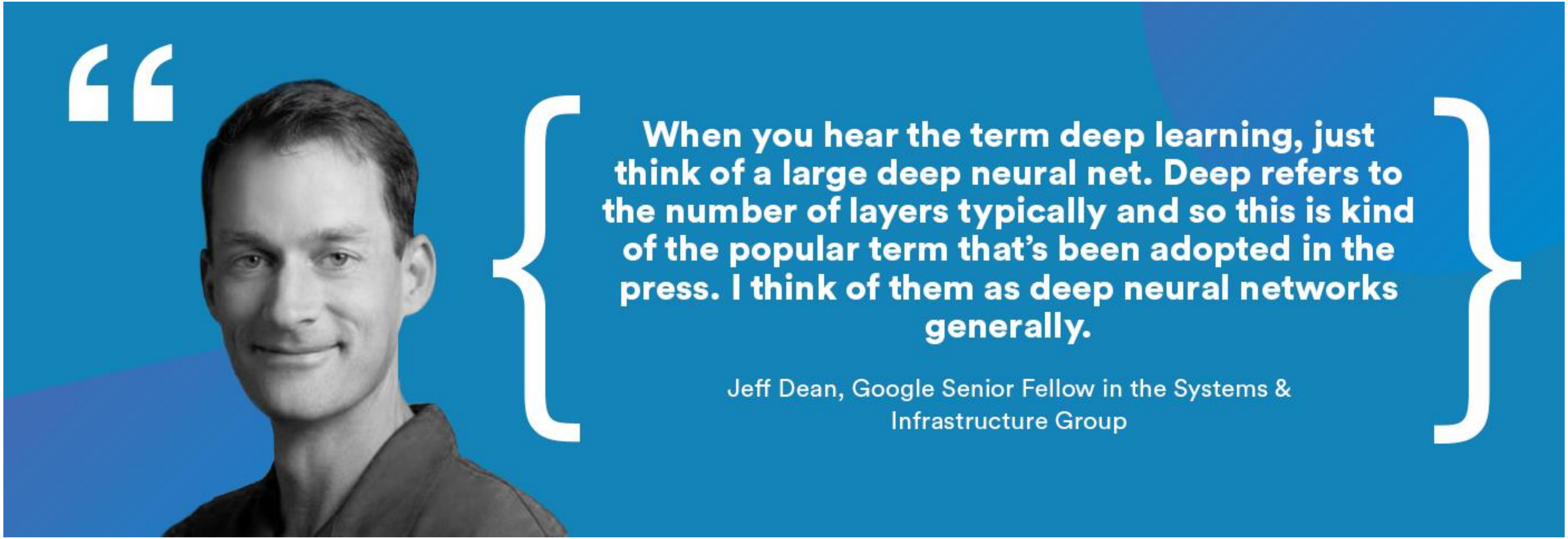
Deep learning / Neural Network primer

Building blocks of deep neural networks

- Model/architecture
- Data
- Loss function and optimizer
- Compute

Building blocks of deep neural networks

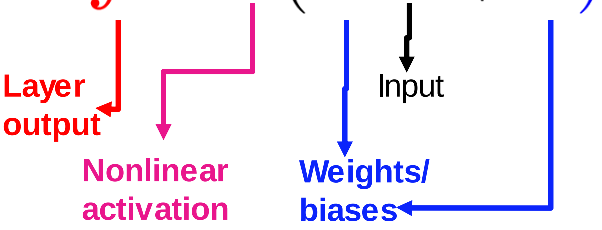
- Model/architecture
- Data
- Loss function and optimizer
- Compute


When you hear the term deep learning, just think of a large deep neural net. Deep refers to the number of layers typically and so this is kind of the popular term that's been adopted in the press. I think of them as deep neural networks generally.
Jeff Dean, Google Senior Fellow in the Systems & Infrastructure Group

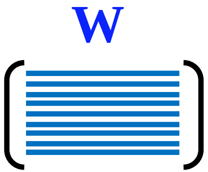
Building blocks: model layer equations

Layer equation

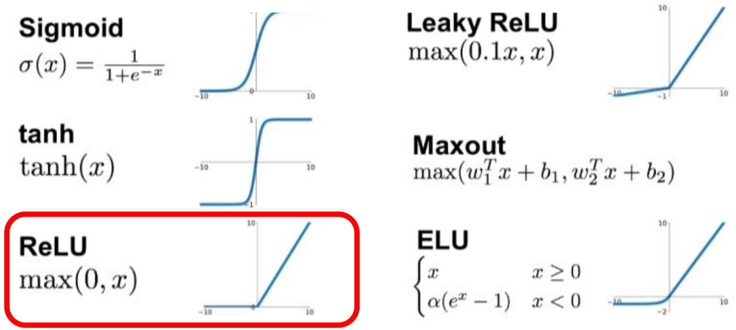
$$\mathbf{y} = \sigma(W\mathbf{x} + \mathbf{b})$$

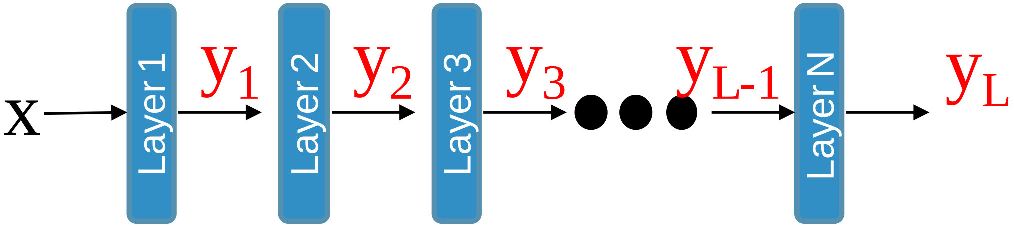


Fully-connected Neural Network



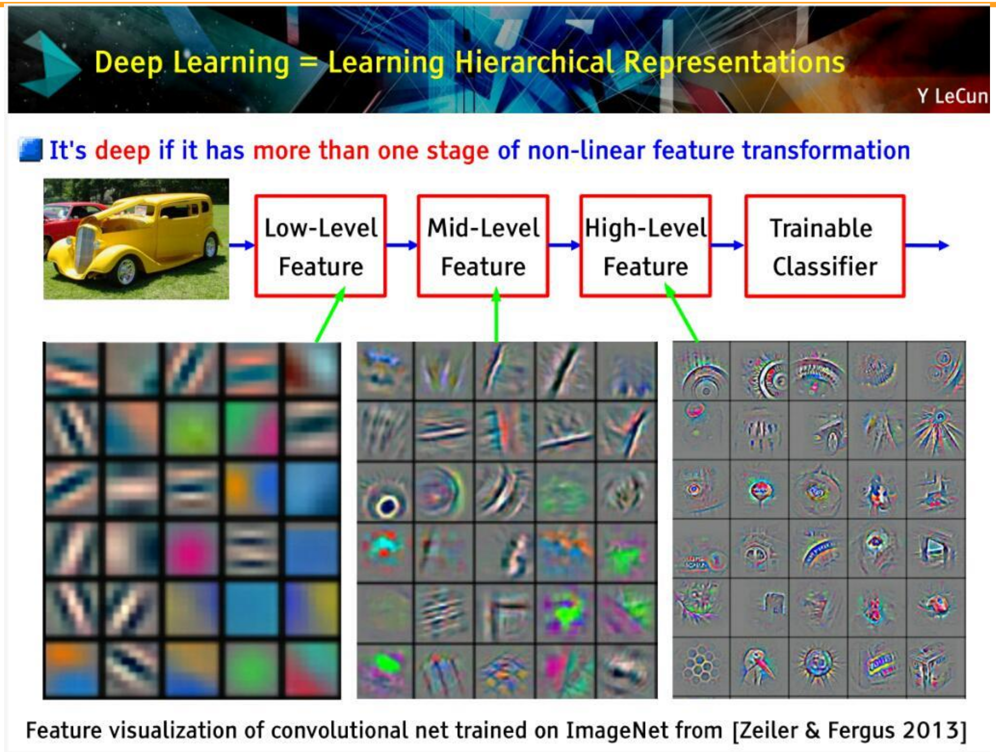
Nonlinear activation functions





Mehta, V., et. al. (2020). http://arxiv.org/abs/2006.12682

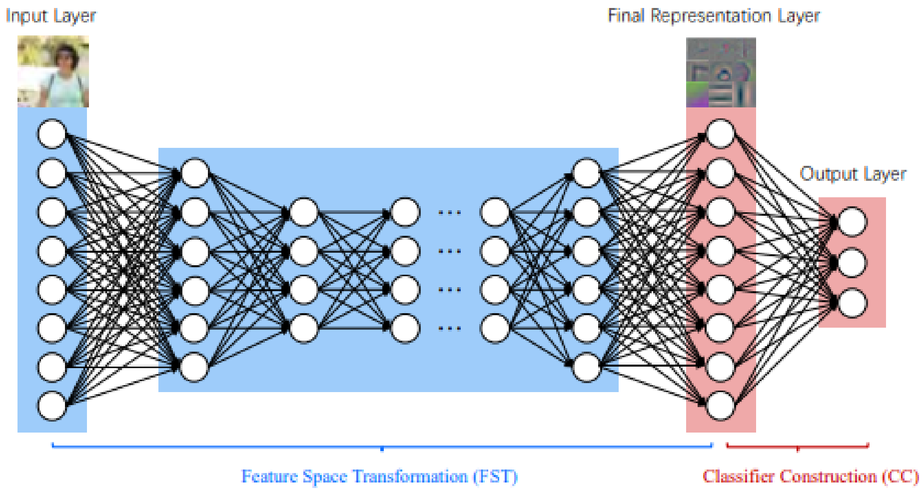
Why go deep?
Learn hierarchical, composable features



Important that these features are learned *jointly*, i.e. can not train layers separately and get the same result

Features, representations, latent space

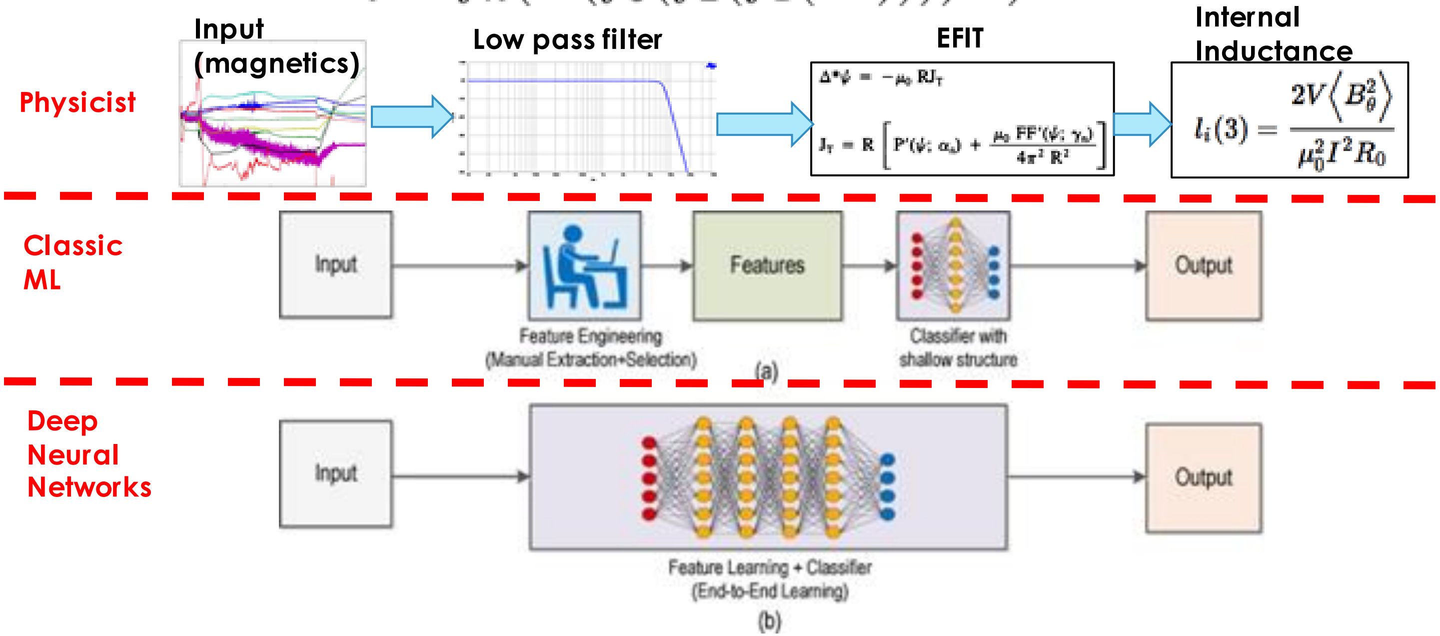
- One way to view deep NN is that they learn “features” or “representations”, with a final layer for classification or regression
- The feature space also often referred to as the latent space; data compressed to a space which latent random variables define



https://towardsdatascience.com/overparameterized-but-generalized-neural-network-420fe646c54c

Deep learning enables working with complex, high-dimensional data

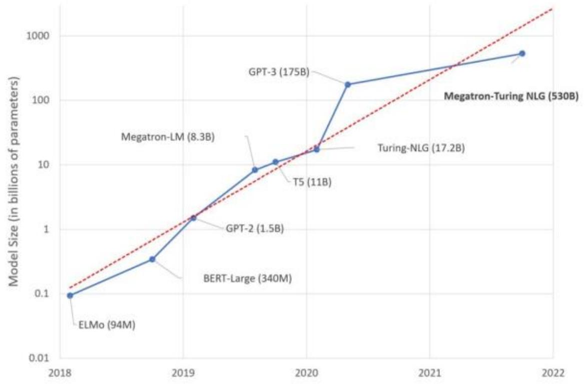
$$\ell_i = f_n(\ldots(f_3(f_2(f_1(M))))\ldots)$$

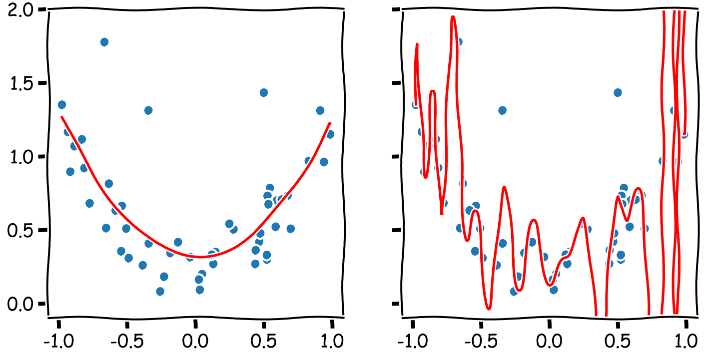


Overparameterization and generalization

- Current largest neural networks nearing 1 trillion parameters
  - Human brain has 100 trillion synapses
- Number of learnable parameters in modern neural networks is often much larger than the training data points
  - Yet they still can generalize well (!?)
  - Different from traditional experience with statistics e.g. regression
  - Still open question as to why and how

→ Role of engineering currently critical in deep NN





Building blocks of deep neural networks

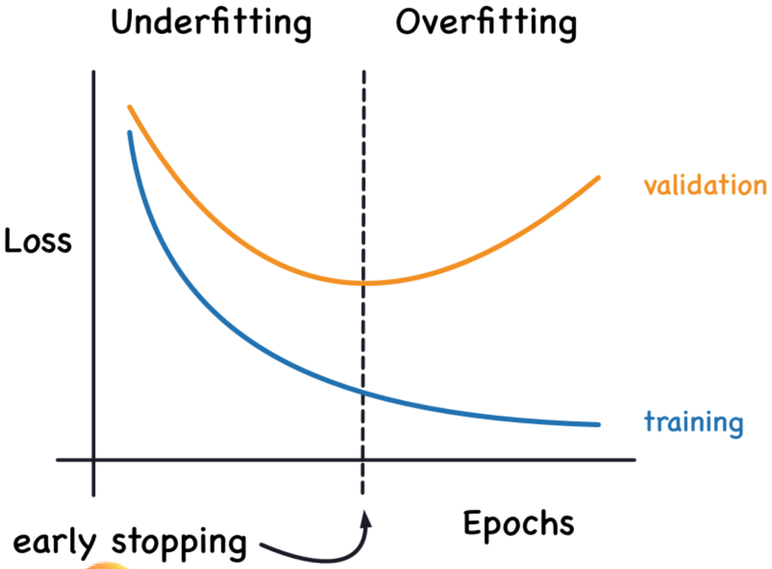
- Model/architecture
- Data
- Loss function and optimizer
- Compute


"Data is the food for AI"
Andrew Ng
CEO, Landing, AI

Building blocks: data

- Due to large number of parameters, deep neural networks are data hungry.
- How much data do you need for your problem?
  - Answer is always "it depends" (complexity of the problem, size of the network, etc.), but more is (almost) always better
  - Rule of thumb ~5k examples per category for classification
- Typical "supervised learning" setup involves gathering input data and the targeted output data (e.g. input: pictures of cats/dogs; output: label for each picture whether cat/dog)

Training/validation/test split

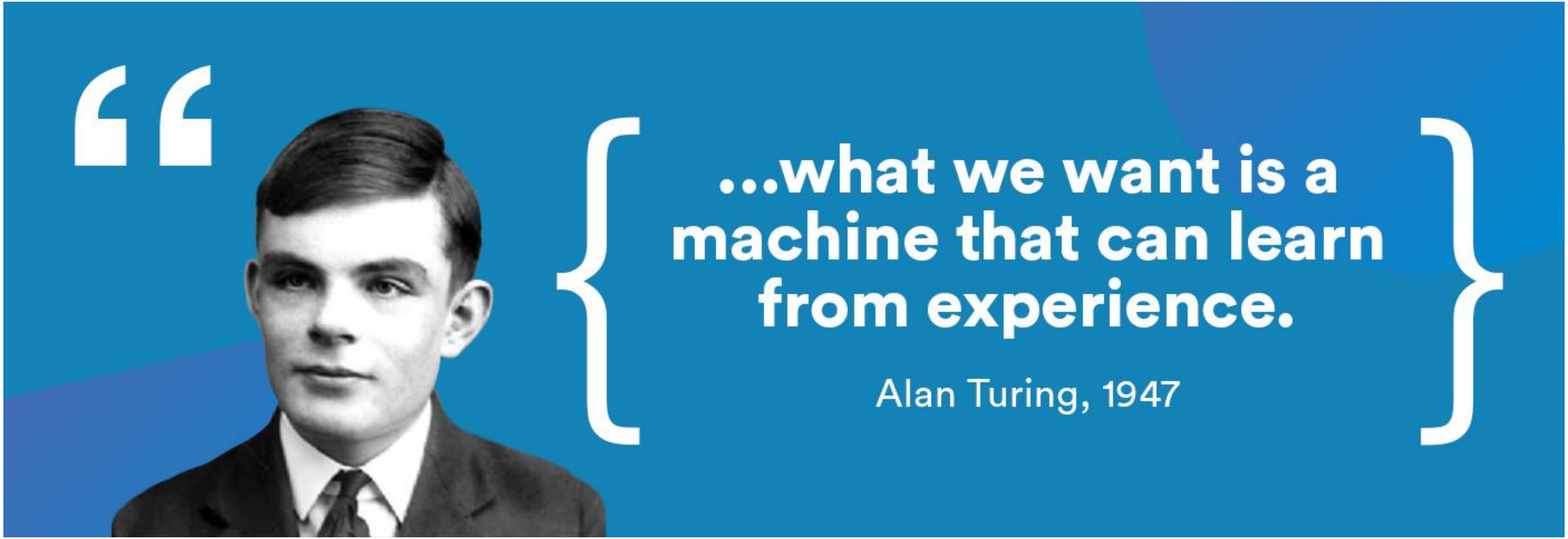


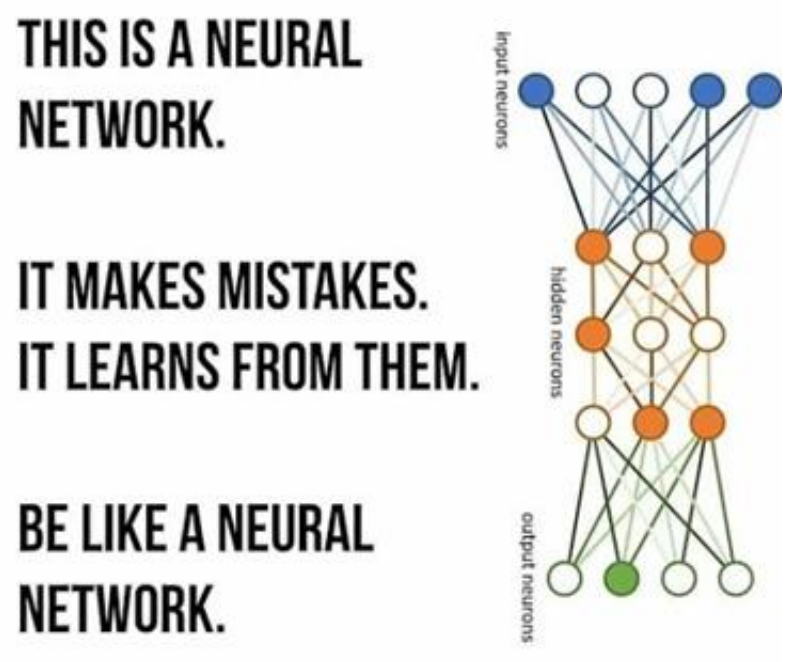
Dataset splitting

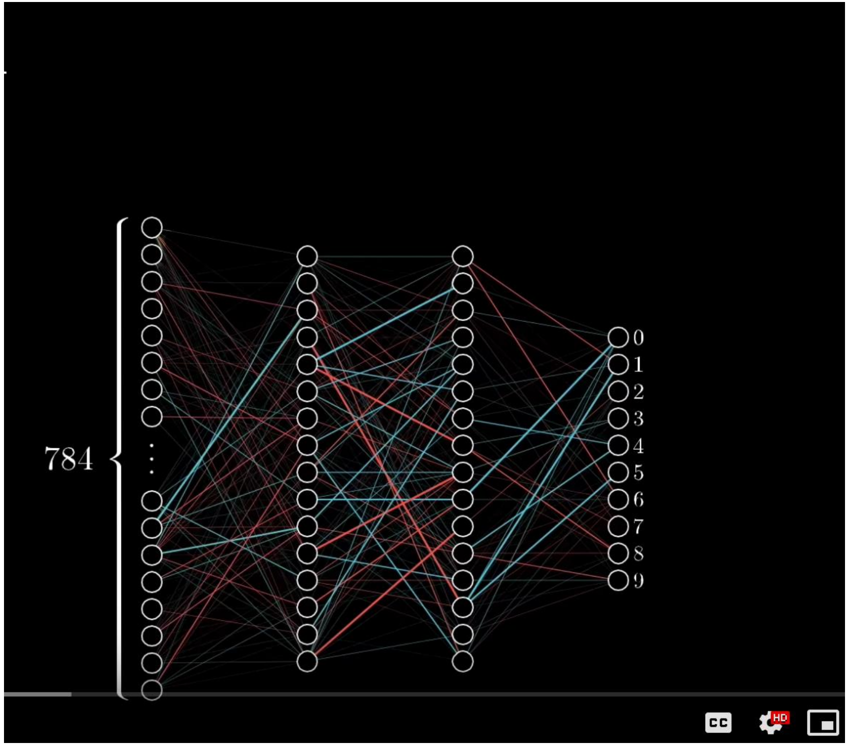
Training	Validation	Test
Trains model, used to update gradients/weights	Monitor performance as training progresses on out-of-sample data, validates model/hyperparameter choices, never updates gradients/weights	Test final trained model on out-of-sample data

Building blocks of deep neural networks

- Model/architecture
- Data
- Loss function and optimizer
- Compute

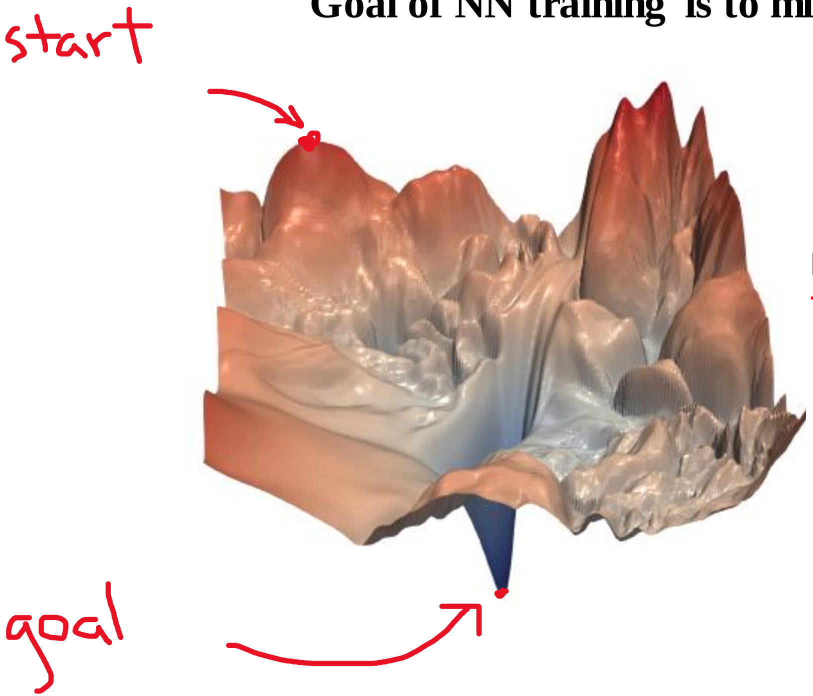

...what we want is a machine that can learn from experience.
Alan Turing, 1947


THIS IS A NEURAL NETWORK.
IT MAKES MISTAKES.
IT LEARNS FROM THEM.
BE LIKE A NEURAL NETWORK.
input neurons
hidden neurons
output neurons


784
0
1
2
3
4
5
6
7
8
9

Building blocks: Loss function for gradient-based optimization

Goal of NN training is to minimize the loss function for the dataset



$$f_\theta(x) = f^{(L)}\left(f^{(L-1)}\left(f^{(N-2)}\left(\ldots\left(f^{(2)}\left(f^{(1)}(x)\right)\right)\right)\right)\right)$$

NN output

Target output

Loss function

$$\ell\left(f_\theta(x), y\right) = \begin{cases} \frac{1}{2}\left(f_\theta(x) - y\right)^2 & \text{MSE loss} \\ -\sum_i^C y_i \log f_\theta(x) & \text{Cross-entropy loss} \\ \ldots & \end{cases}$$

NN weight update

Learning rate

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \ell}{\partial \boldsymbol{\theta}}$$

Backpropagation: the learning algorithm

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \ell}{\partial \boldsymbol{\theta}}$$

- Backpropagation uses chain rule to determine how weights should change given an output loss. Propagate error backwards, calculating weight updates
- Most deep learning frameworks use autodifferentiation to accurately calculate gradients, no need to specify by hand

$$\boldsymbol{\theta} = \left\{ W^{(1)}, \mathbf{b}^{(1)}, \ldots, W^{(L)}, \mathbf{b}^{(L)} \right\}$$

$$\mathbf{z}^{(k)} = W^{(k)} \mathbf{y}^{(k-1)} + \mathbf{b}^{(k)}$$

$$\frac{\partial \ell}{\partial W^{(k)}} = \frac{\partial \ell}{\partial \mathbf{z}^{(k)}} \cdot \left[\mathbf{y}^{(k-1)} \right]^T$$

$$\frac{\partial \ell}{\partial \mathbf{b}^{(k)}} = \frac{\partial \ell}{\partial \mathbf{z}^{(k)}}$$

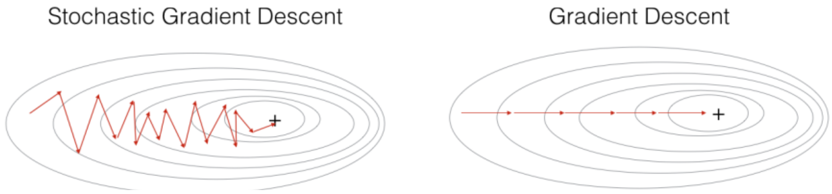
$$\frac{\partial \ell}{\partial \mathbf{z}^{(k-1)}} = \left[W^{(k)} \right]^T \cdot \frac{\partial \ell}{\partial \mathbf{z}^{(k)}} \odot \sigma^{(k-1)\prime}(\mathbf{z}^{(k-1)})$$

Rumelhart, "Learning representations by back-propagating errors", Nature 1986

Stochastic Gradient Descent (SGD)

- Performs optimization steps using part (or "batches") of dataset for gradient (instead of entire dataset)
  - "stochastic" because random samples used in mini-batches
  - Spend more time processing more data instead of minimizing optimization steps
  - "the best optimization algorithms are not necessarily the best learning algorithms" [Bottou, NeurIPS 2007]

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell(f_\theta(x_i), y_i)}{\partial \boldsymbol{\theta}}$$



$n = N_{batch}$ $(N_{batch} < N)$

$n = N$

- Variants most commonly used:
  - SGD with momentum
    - Faster convergence
  - Adam
    - Easy default hyperparameters

Hyperparameter tuning

- Many parameters chosen (not learned), called "hyperparameters"
- Many ways to find optimal hyperparameters
  - Grid search can be employed to scan, but often too expensive.
  - Heuristics most often employed ("what worked before")
  - Bayesian optimization (with several packages e.g. RayTune and Optuna implementing) can find optimal hyperparameter setting with fewer training runs

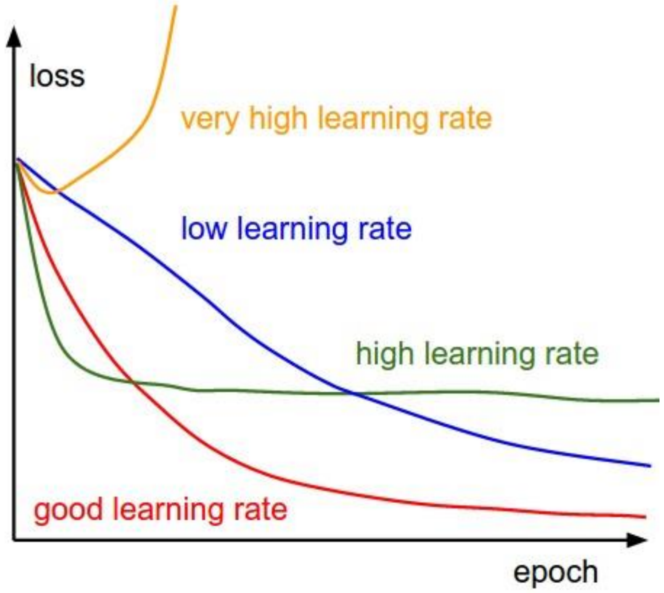
Hyperparameter	Approximate sensitivity
Learning rate	High
Optimizer choice	Low
Other optimizer params (e.g., Adam beta1)	Low
Batch size	Low
Weight initialization	Medium
Loss function	High
Model depth	Medium
Layer size	High
Layer params (e.g., kernel size)	Medium
Weight of regularization	Medium
Nonlinearity	Low

tune

OPTUNA

Example: Hyperparameter tuning of learning rate

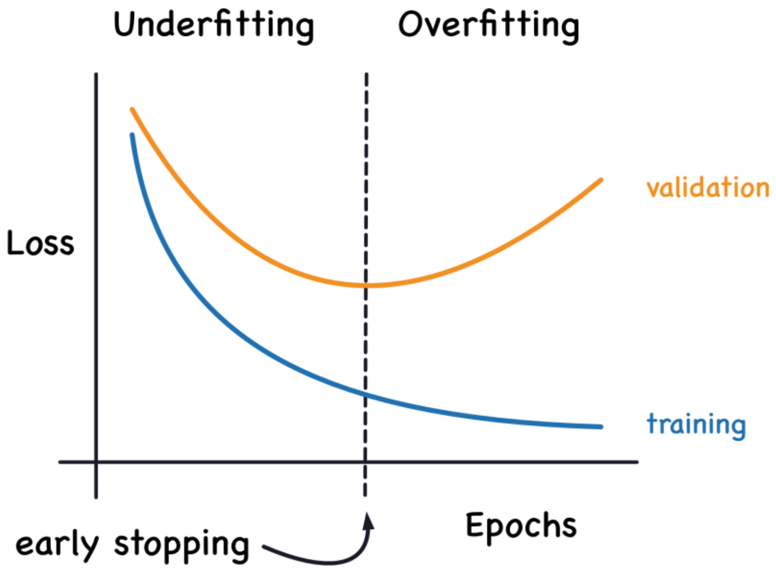
- Learning rate (LR) is one of the most important hyperparameters to tune
- For each LR, train the neural network over several epochs, monitor training loss to select optimal LR



Loss: measure of the "error"
1 Epoch = 1 pass through ALL data

Example: Hyperparameter tuning of learning rate

- With the optimal LR, the training loss will continue to drop
- (usually) when the validation loss begins to rise, the neural network begins to overfit
- Early stopping of the training is often used to save the neural network at the optimal level



Loss: measure of the "error"
1 Epoch = 1 pass through ALL data

Building blocks of deep neural networks

- Model/architecture
- Data
- Loss function and optimizer
- Compute

The biggest lesson that can be read from 70 years of AI research is that **general methods that leverage computation are ultimately the most effective**, and by a large margin. …Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation… the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation.

-Richard Sutton "*The Bitter Lesson*", 2019

GPUs are (mostly) the driver for compute improvements in deep neural networks

- Specifically CUDA parallel programming model made it easy to leverage GPUs for parallel processing, accelerating the tensor operations needed in neural networks
- Many frameworks exist to implement deep neural networks; all make it seamless to leverage GPUs (no CUDA programming required)
- Key for fastest performance is pipelining the workflow to ensure GPUs don't sit idle
  - e.g. load next data batch using CPUs concurrent with GPU operations on other batch of data with pin_memory in Pytorch

NVIDIA CUDA

PyTorch

TensorFlow

Pytorch example

```python
# Load the dataset.
dataset = MyDataset(filepath)

# Create the dataloader.
dataloader = DataLoader(dataset, batch_size=32, shuffle=True, num_workers=4)

# Create the model.
model = resnet50(pretrained=True)

# Create the optimizer.
optimizer = Adam(model.parameters(), lr=0.001)

#create device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

#start training loop
for epoch in range(1, 100):
    for i, (images, labels) in enumerate(dataloader):
        # Move the images and labels to the device.
        images = images.to(device); labels = labels.to(device)
        # Forward pass.
        outputs = model(images)
        loss = CrossEntropyLoss()(outputs, labels)

        # Backward pass.
        optimizer.zero_grad()
        loss.backward()

        # Update weights.
        optimizer.step()

        # Print the loss.
        if i % 100 == 0:
            print(loss.item())
```

Written by: GitHub Copilot

Inductive bias

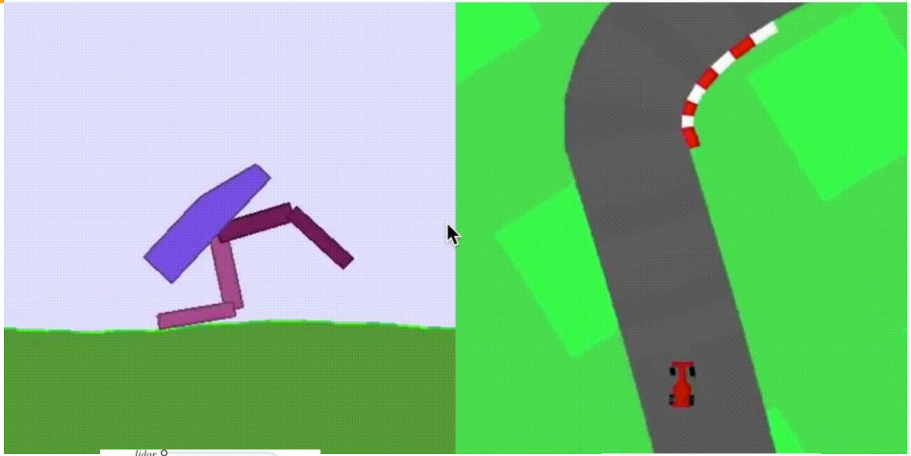
"Encode our knowledge and assumptions about the world"

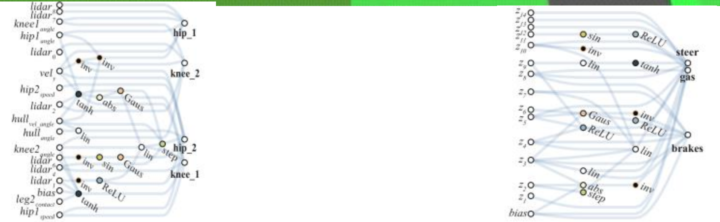
Weight Agnostic Neural Networks

ADAM GAIER
Google Brain
DAVID HA
Google Brain

June 12
2019
Download
PDF
NeurIPS 2019
Slides

- Instead of updating weights, modify *structure* of neural network
- Showed that with the right *structure*, learning was possible, despite single parameter weight





"Not all neural network architectures are created equal, some perform much better than others for certain tasks. But **how important are the weight parameters of a neural network compared to its architecture?**"

https://arxiv.org/abs/1906.04358

Convolutional structure in neural networks a strong inductive bias for locality

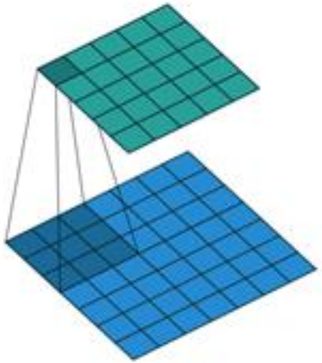
Layer equation

$$\mathbf{y} = \sigma(W\mathbf{x} + \mathbf{b})$$

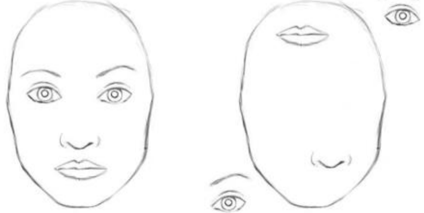
CNN weight matrix sparse connectivity enforces **translation invariance,** useful for natural images. But also cons, e.g. one con is the "Picasso effect", default CNNs can't distinguish global and relative relationships

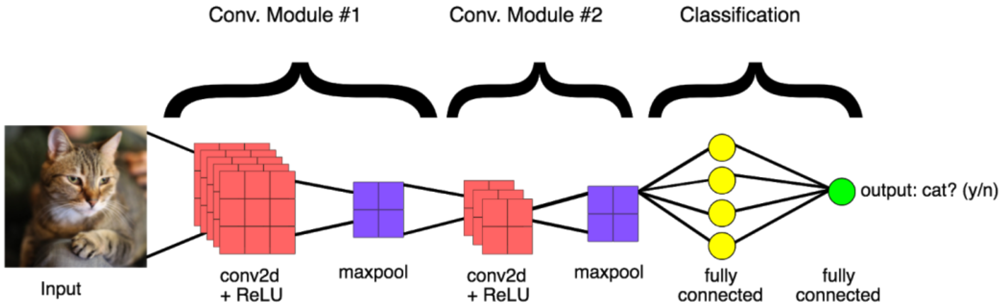
Fully-connected Neural Network

Convolutional Neural Network

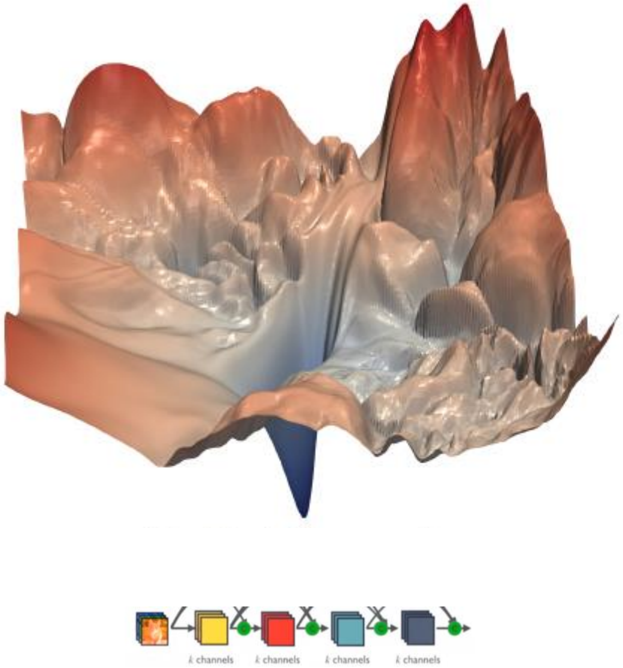


"Face" "Face"

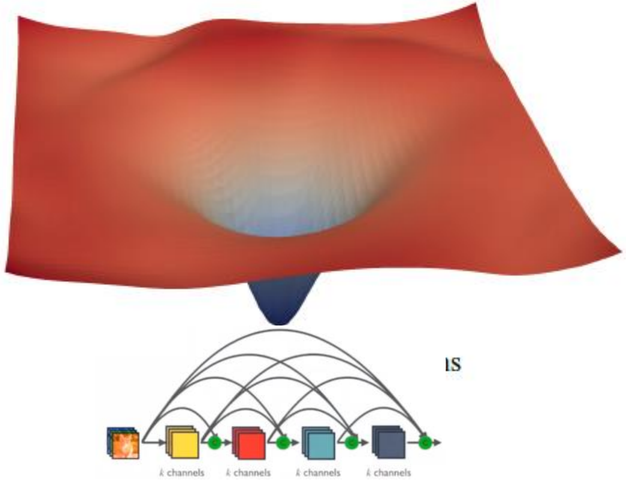




Architecture choices have dramatic effect on loss landscape -> ease of training



No Residual connections



With Residual connections

Li, H., *et. al.*(2018). https://arxiv.org/pdf/1712.09913.pdf

Architectures and Techniques Related to Scientific Deep Neural Networks

Resource for exploring current models:

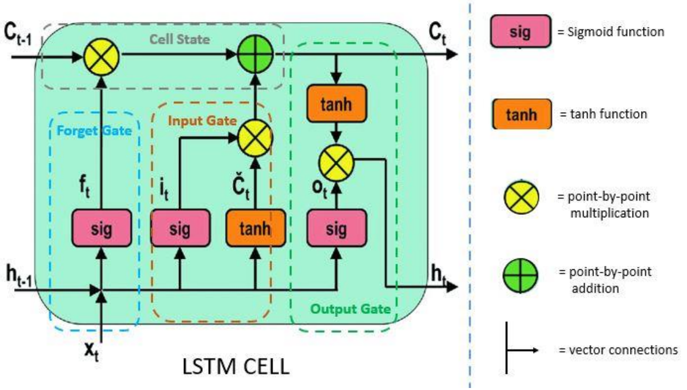
Papers With Code

https://paperswithcode.com/

Sequential Models

Sequential models: Long Short-Term Memory (LSTM)

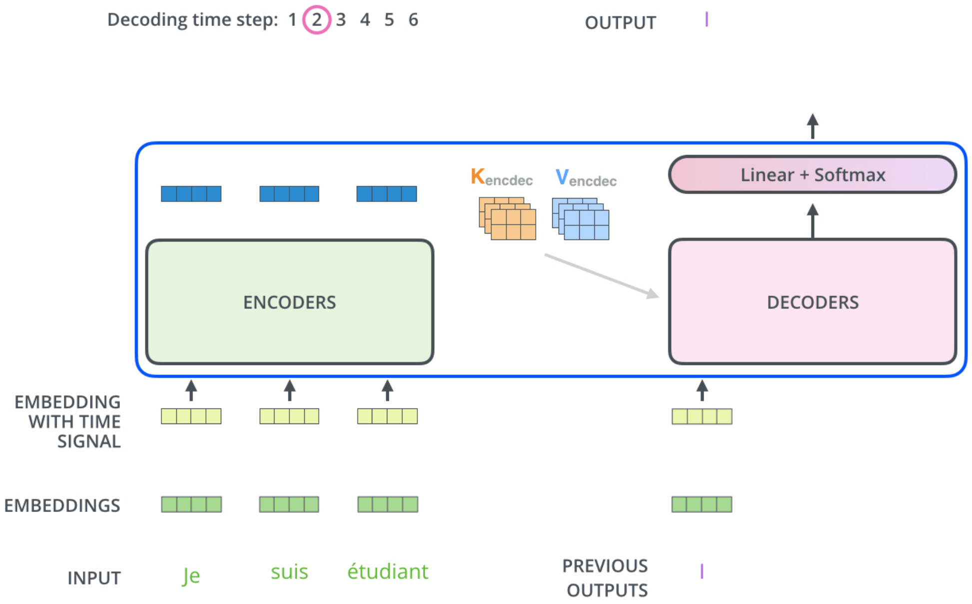
- Sequential models solve time-dependent problems (e.g. audio transcribing, text translation, time-series prediction, etc.)



An **LSTM** is a type of recurrent neural network that addresses the vanishing gradient problem in vanilla RNNs through additional cells, input and output gates. Intuitively, vanishing gradients are solved through additional *additive* components, and forget gate activations, that allow the gradients to flow through the network without vanishing as quickly.

Hochreiter and Schmidhuber, Neural Computation 9 (8): 1735-1780, 1997

Sequential models: Transformer



A **Transformer** is a model architecture that eschews recurrence and instead relies entirely on an attention mechanism to draw global dependencies between input and output. Before Transformers, the dominant sequence transduction models were based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The Transformer also employs an encoder and decoder, but removing recurrence in favor of attention mechanisms allows for significantly more parallelization than methods like RNNs and CNNs.

Sequential models: Transformer cont., the attention mechanism

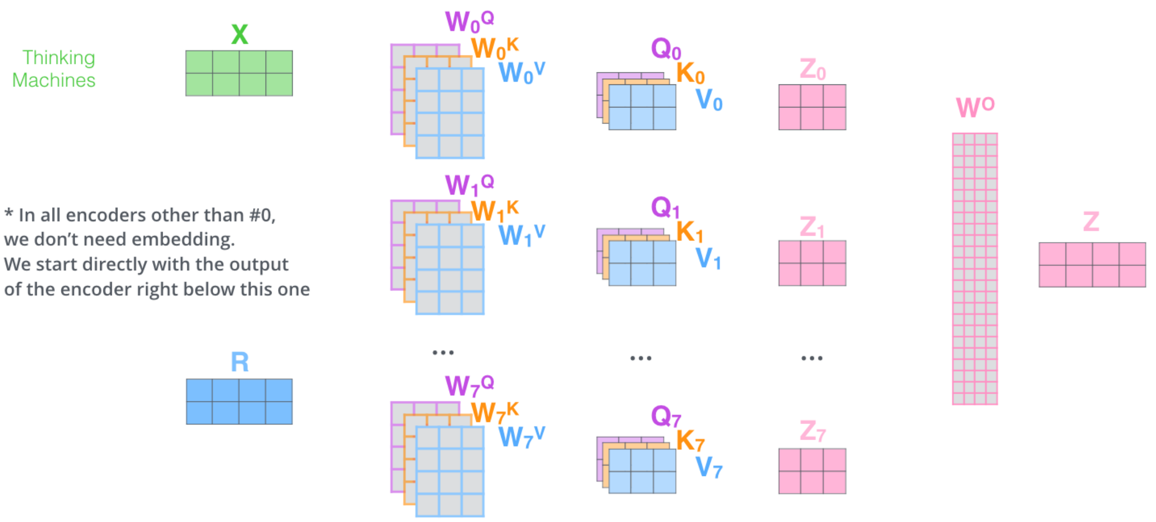
1) This is our input sentence*

2) We embed each word*

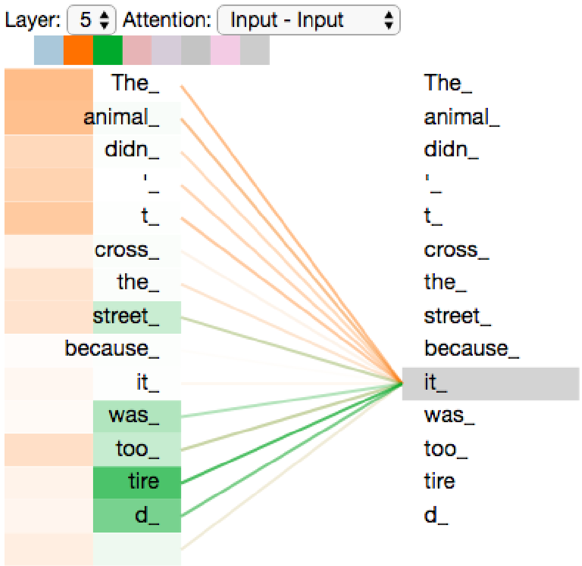
3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

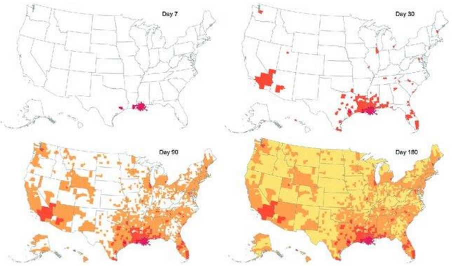


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

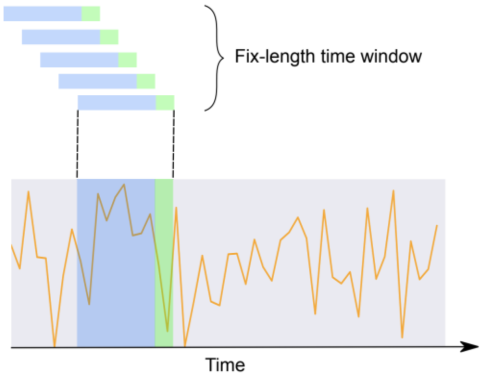


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Sequence models for predicting influenza spread



Transformer can often benefit from better modeling of long-term dependencies vs recurrent architectures
(CNNs with dilated convolutions also designed for long-term dependencies, see e.g. TCN [Bai 2018])



ith respect to baseline model.

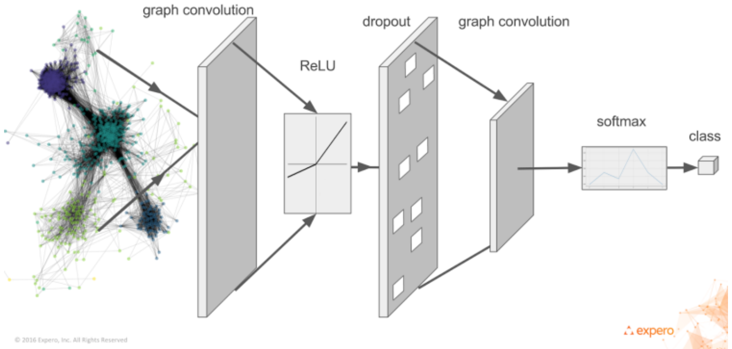
Model	Pearson Correlation	RMSE
ARIMA	0.769 (+0 %)	1.020 (-0 %)
LSTM	0.924 (+19.9 %)	0.807 (-20.9 %)
Seq2Seq+attn	0.920 (+19.5 %)	0.642 (-37.1 %)
Transformer	0.928 (+20.7 %)	0.588 (-42.4 %)

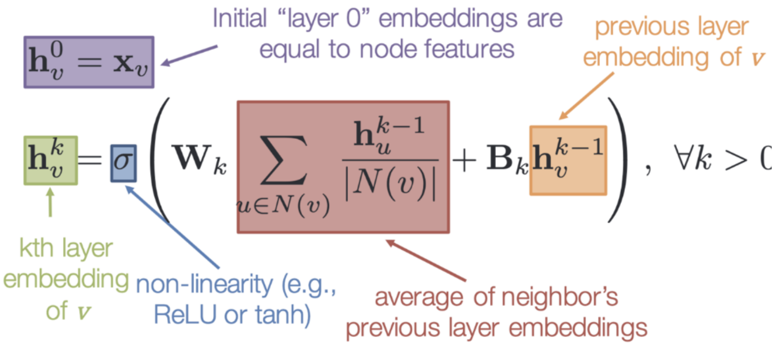
Wu, "Deep Transformer Models for Time Series Forecasting" https://arxiv.org/pdf/2001.08317.pdf

Graph Neural Networks

Graph Neural Networks

- GNNs operate on graph structures with nodes/edges
  - Perform better with fewer layers

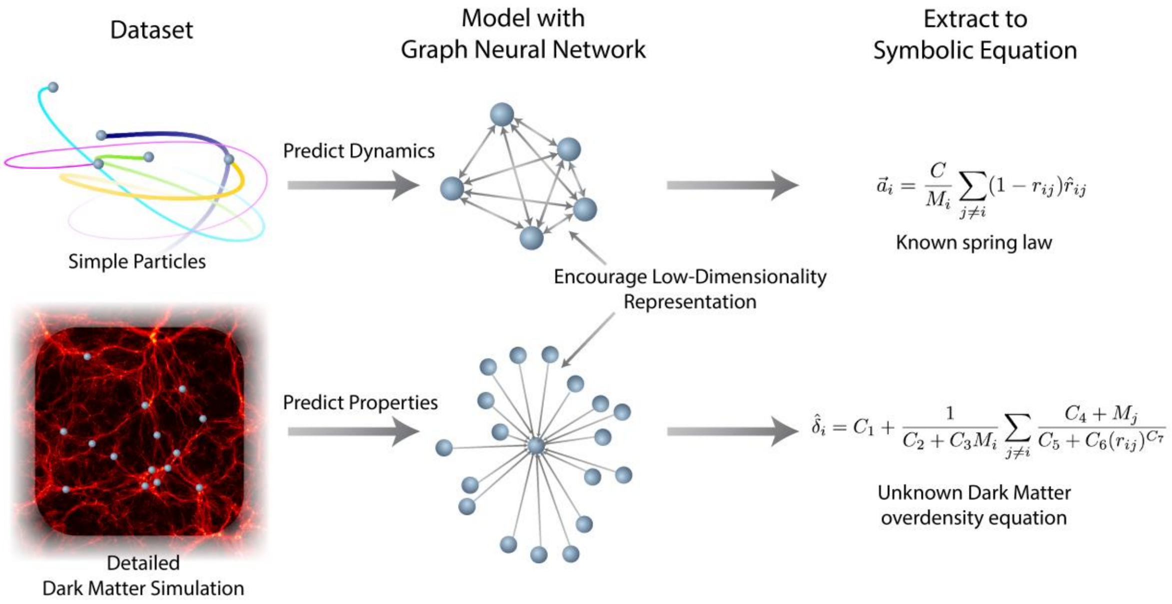




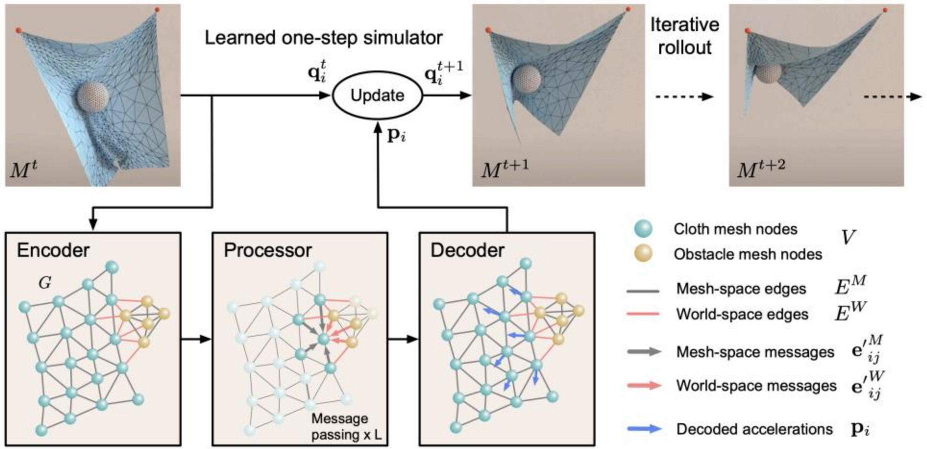
Savannah Thais, Graph Neural Networks

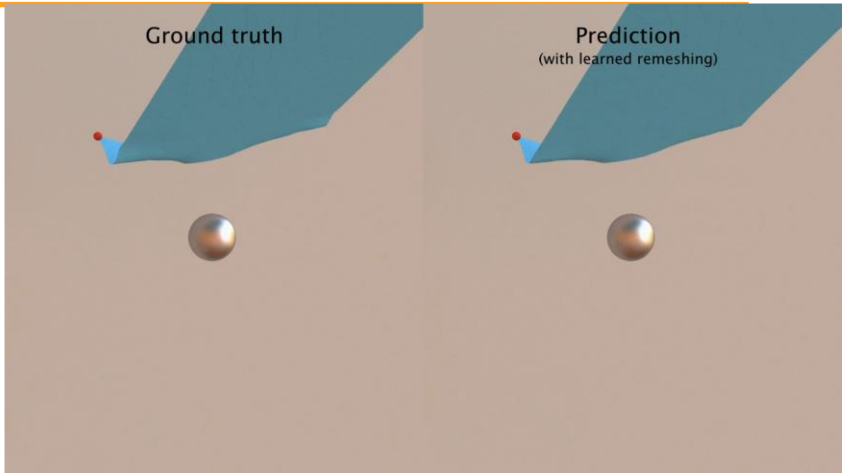
https://ericmjl.github.io/essays-on-data-science/machine-learning/graph-nets/
https://theaisummer.com/graph-convolutional-networks/
https://theaisummer.com/gnn-architectures/

Graph Neural Networks for learning N-body problems and dark matter is cosmology



M. Cranmer, https://astroautomata.com/paper/symbolic-neural-nets/

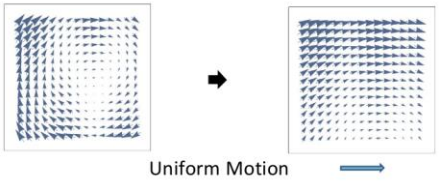




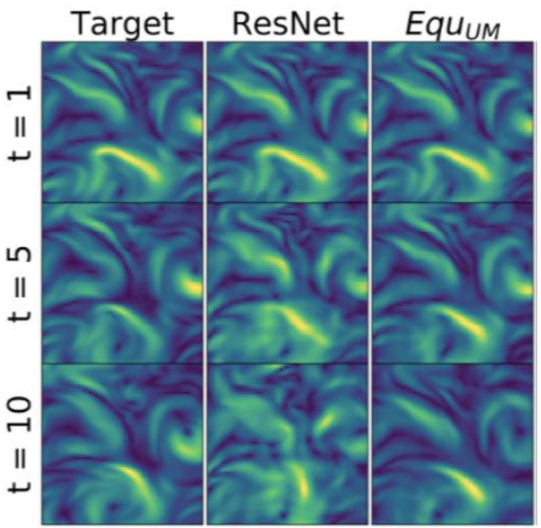
https://sites.google.com/view/meshgraphnets

PDE solving

Equivariant Neural Networks



- Modeling PDEs such as Navier-Stokes can be made more accurate by using NN architecture which guarantee symmetries of the underlying PDE are satisfied
- Ex: Ocean data flow prediction enforcing Uniform Motion performs much better over long time



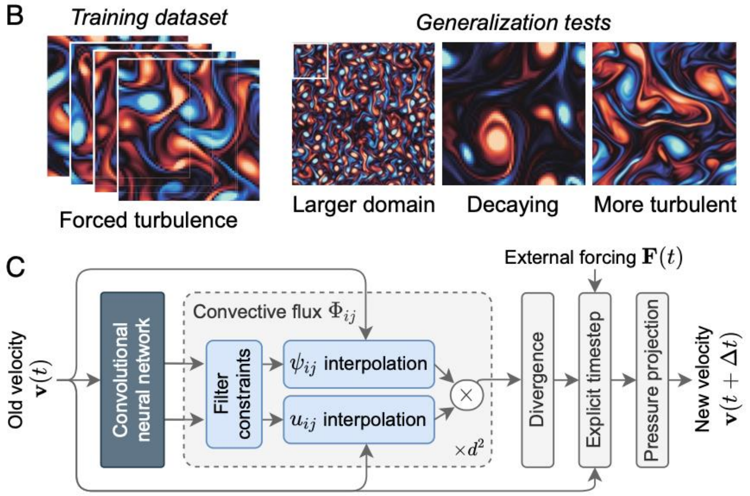
Robin Walters, "Incorporating Symmetry into Deep Dynamics Models for Improved Generalization." ICLR 2021.

Solving on coarse grain grids, leveraging differentiable simulators

- Hybrid approaches can use NN to target specific parts of numerical PDE algorithms (e.g. local operator for convective fluxes)
- Learn to replicate high-res simulations on coarse, limited grid, inference on fine, expanded grid
- With a fully differentiable simulator, can optimize end-to-end through multiple steps of simulation. Help stability.

$$\frac{\partial \mathbf{u}}{\partial t} = -\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) + \frac{1}{Re}\nabla^2 \mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{f}$$

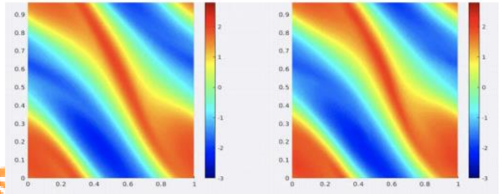
$$\nabla \cdot \mathbf{u} = 0,$$

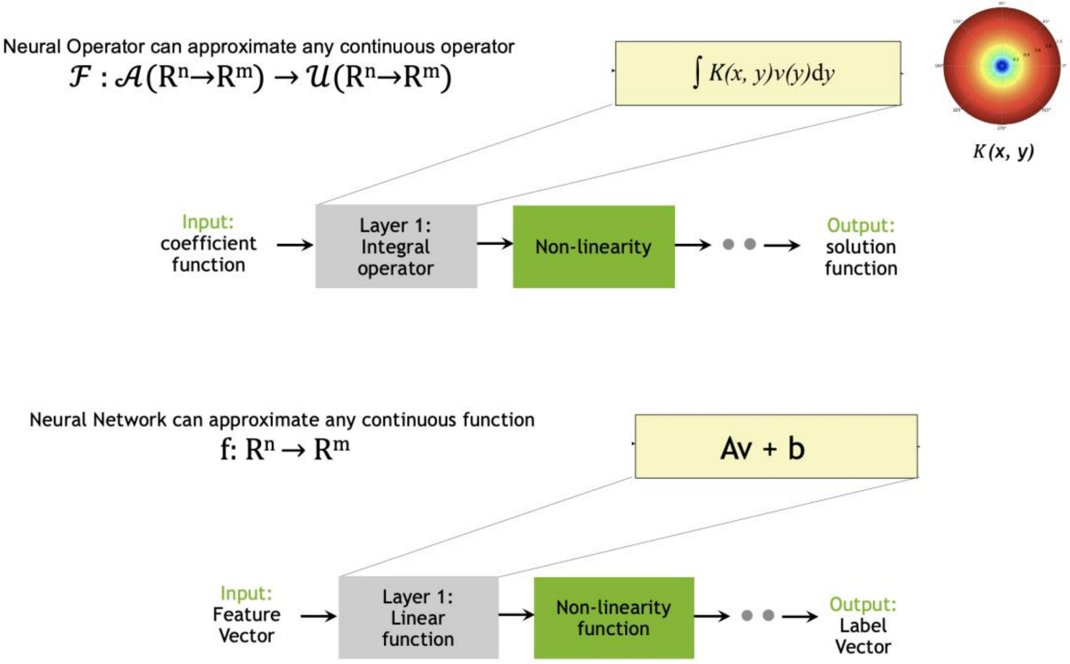


Kochkov, PNAS 2021

Learning operators instead of functions

- Replacing linear function with an integral operator enables better generalization to unseen data
- e.g. Fourier Neural Operator (FNO) for fluid flow, 1000x faster





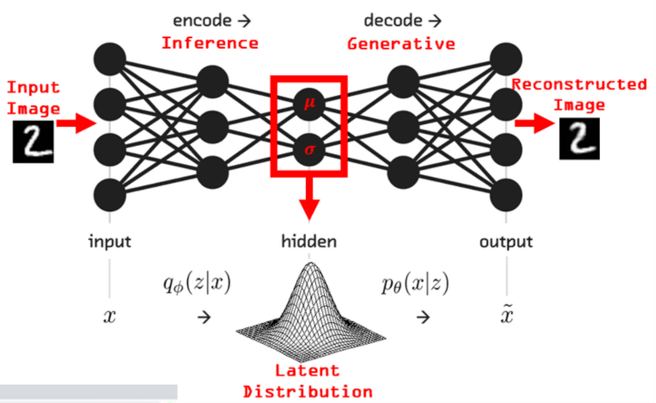
Anima Anandkumar, GTC2021

Generative modeling

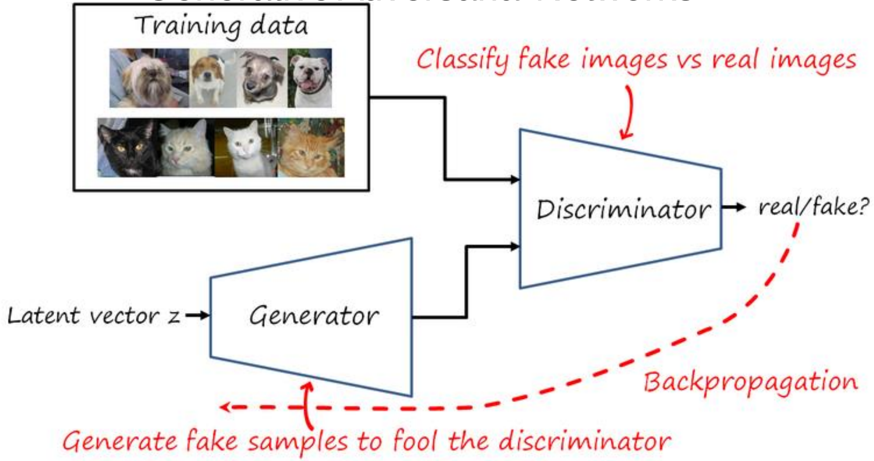
Generative architectures

- Learn joint-distribution p(x,y) instead of discriminative distribution p(y|x)

Variational Autoencoder (VAE)



Generative Adversarial Networks



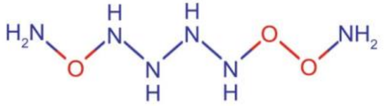
Normalizing flows for scientific generative modeling

$$f : R^n \rightarrow R^n \quad \text{such that} \quad x = f(z) \quad \text{and} \quad z = f^{-1}(x)$$

$$p_X(x) = p_Z\left(f^{-1}(x)\right)\left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right| = p_Z(z)\left|\det\left(\frac{\partial f}{\partial z}\right)\right|^{-1}$$

$$\log p_X(x) = \log p_Z(z) - \log\left|\det\left(\frac{\partial f}{\partial z}\right)\right|$$

- GAN and VAE don't learn probabilit density p(x) of data directly
- Normalizing flow-based algorithms learn p(x) explicitly, by design being invertible (and cheaply for computational reasonableness)
  - Can more accurately capture data distribution
- Used in inverse molecule design to learn from molecule database, and then invert to specify properties to generate new molecules

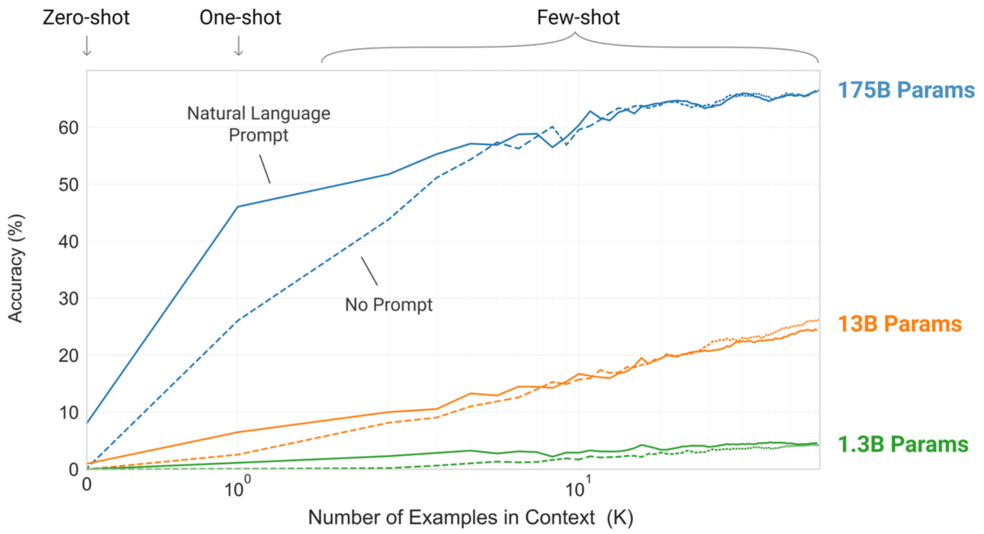


https://www.scirp.org/journal/paperinformation.aspx?paperid=112258
https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Pretraining models
(“Foundation models”)

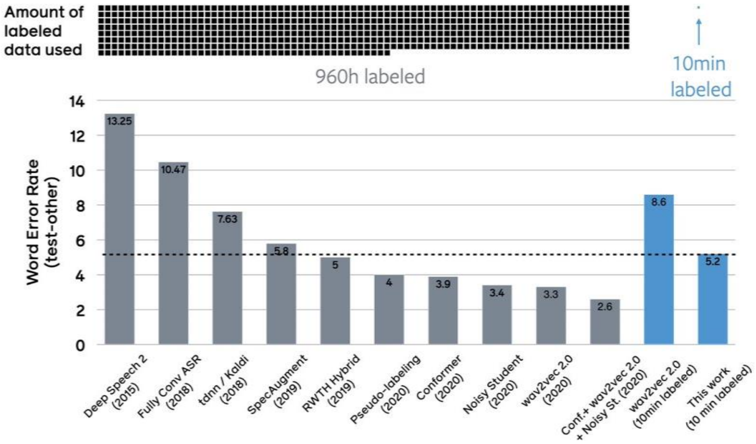
Large-scale pretraining unsupervised leads to general, flexible neural networks

- Train on unlabelled data, simply predicting the next word in the sentence
- GPT-3 showed scaling the parameter size of transformer (with required data) results in flexible neural networks, that are few-shot learners

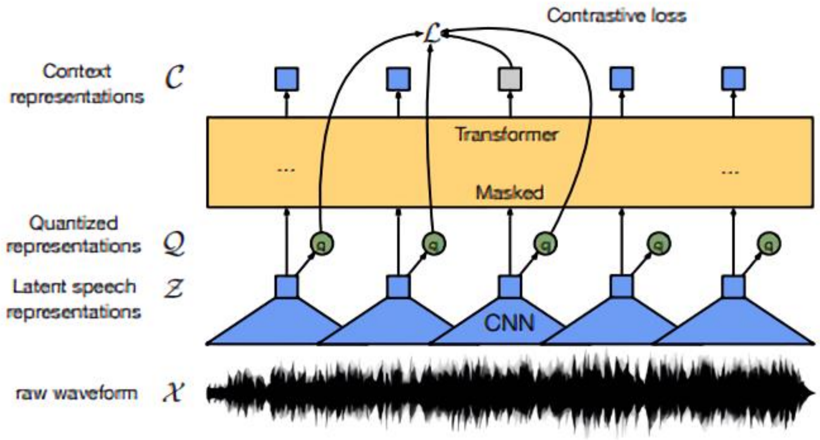


Brown, https://arxiv.org/abs/2005.14165

Pre-training speech recognition models with contrastive loss drastically reduces needed labelled data



Librispeech benchmark, WER on test-other



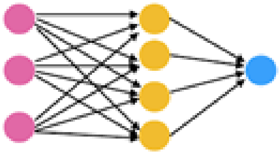
Baevski https://arxiv.org/abs/2006.11477

SUMMARY

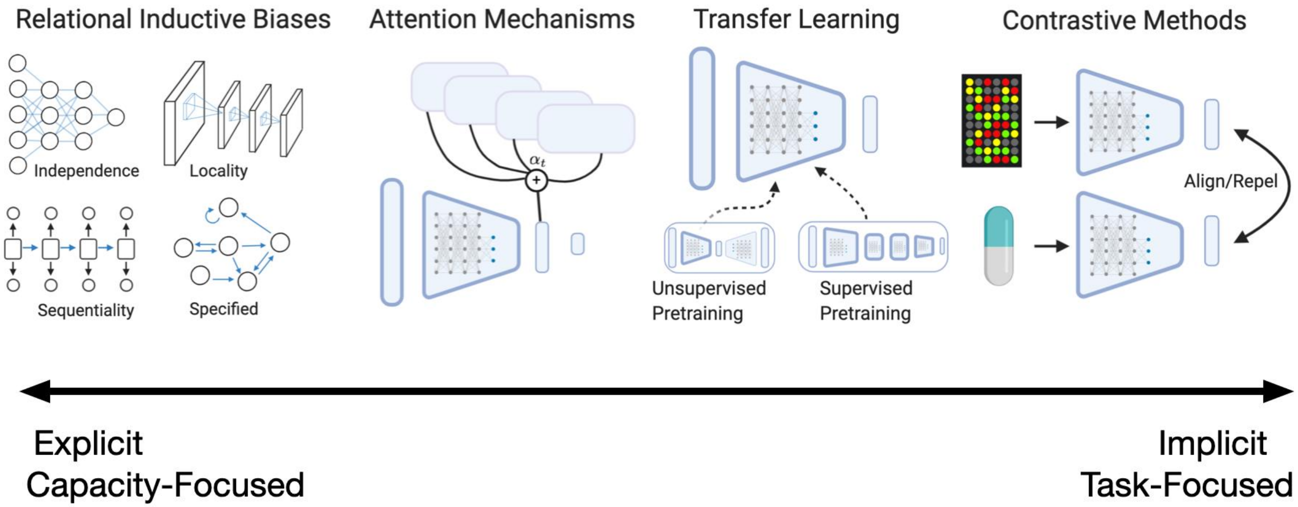
Fusion researcher

AI/ML

+



Backup

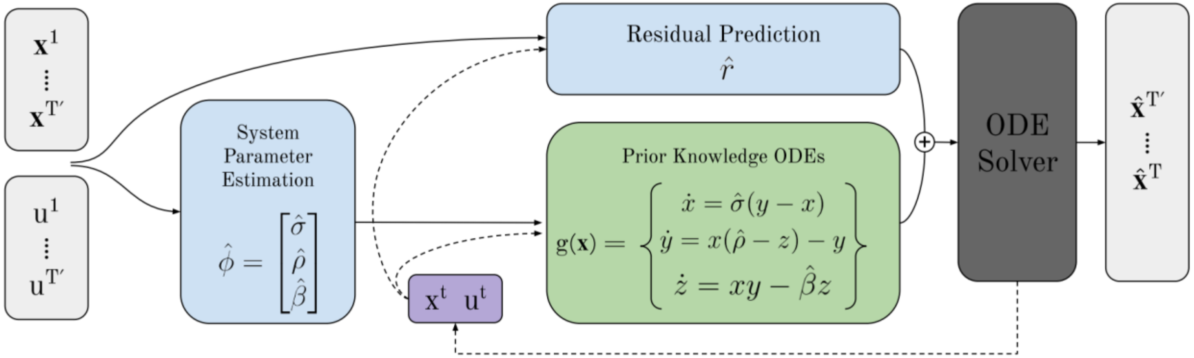


https://sgfin.github.io/2020/06/22/Induction-Intro/

Control / RL

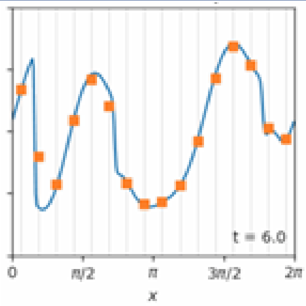
Structure ("inductive bias") can be included in neural networks to further enhance performance

"Physics"-informed

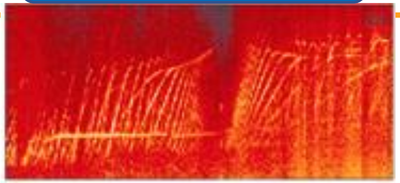


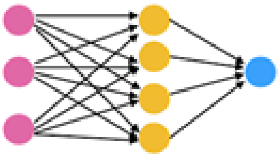
Mehta, V., et. al. (2020). http://arxiv.org/abs/2006.12682

Code Acceleration

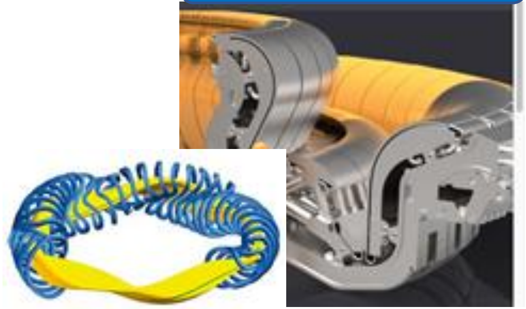


Prediction and detection

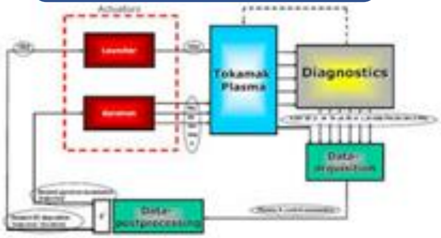




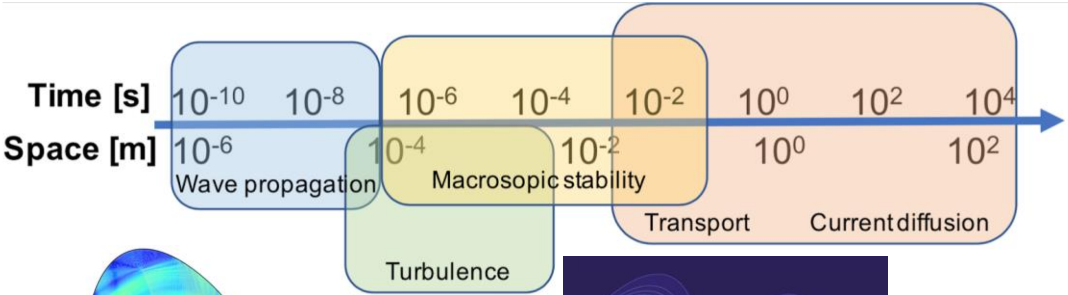
Design

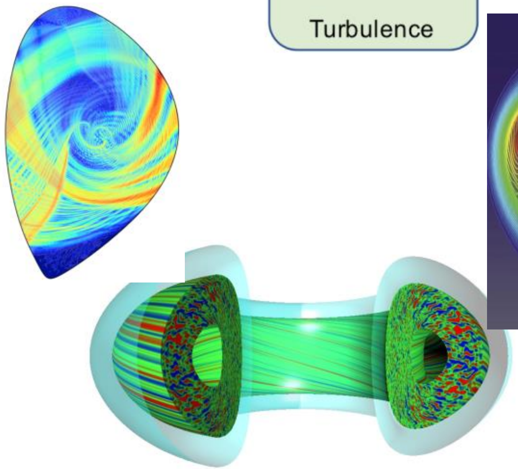


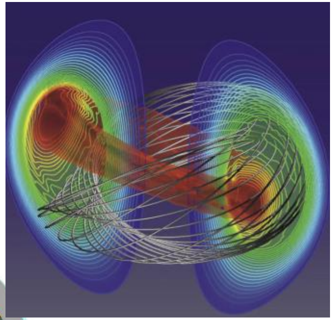
Control

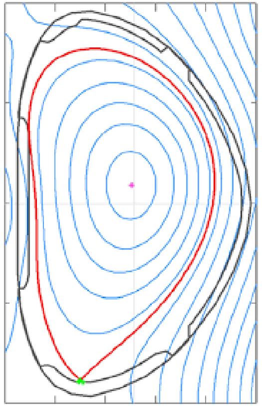


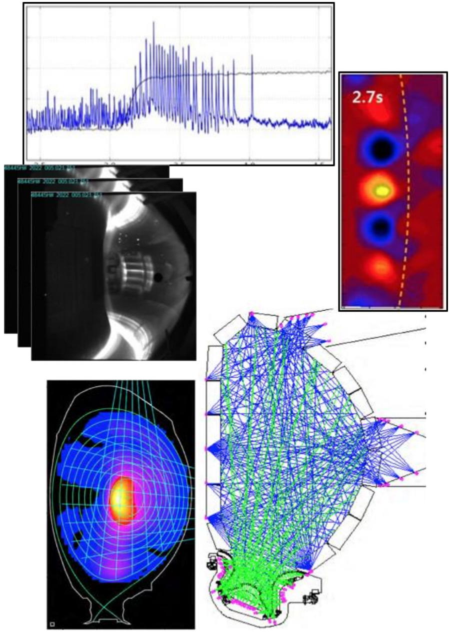
Fusion plasmas have a range of phenomena, manifest over multiple time and spatial scales









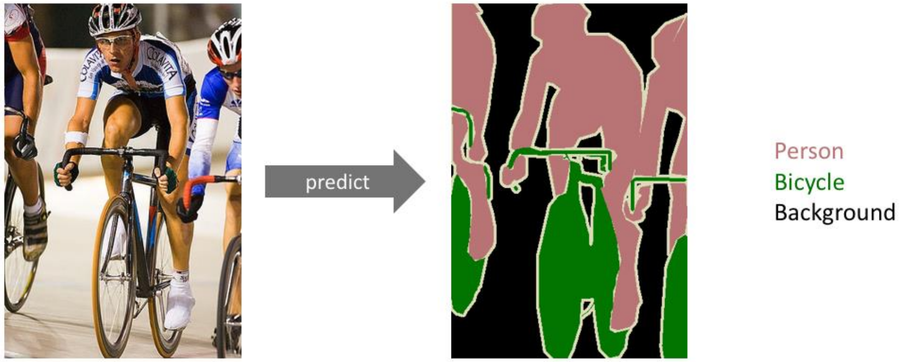


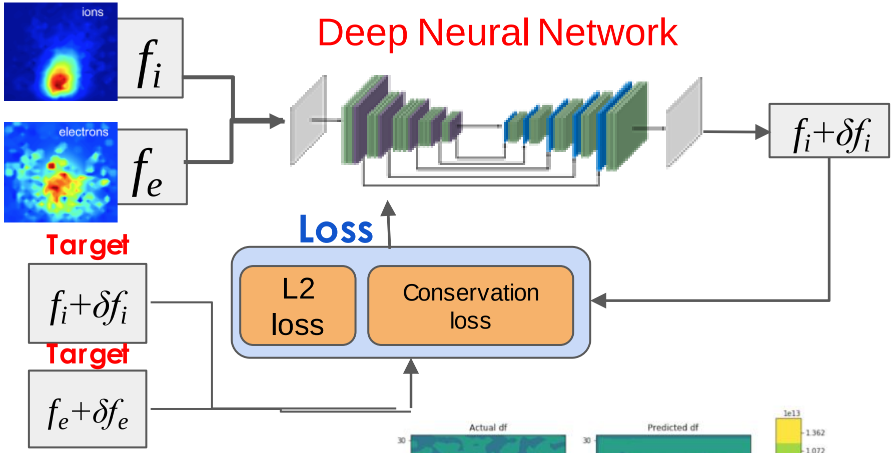
Accelerating massively parallel XGC code with machine learning allows including more physics

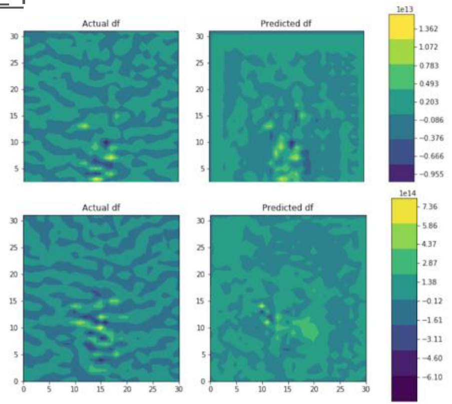
$$\left.\frac{\mathrm{d} f_a}{\mathrm{d}t}\right|_{col} = \sum_b C_{ab}(f_a, f_b')$$

$$= -\sum_b \frac{e_a^2 e_b^2 \ln \Lambda_{ab}}{8\pi \epsilon_0^2 m_a} \nabla \cdot \left[\int \mathrm{d}^3 v' \, \underline{\boldsymbol{U}} \cdot \left(\frac{f_a}{m_b} \nabla' f_b' - \frac{f_b'}{m_a} \nabla f_a \right) \right]$$

$$\sum_a \int \mathrm{d}^3 v \left[\phi_a \sum_b C_{ab}(f_a, f_b') \right] = -\sum_{a,b} \int \mathrm{d}^3 v \, \phi_a \nabla \cdot \boldsymbol{J}_{ab}$$







References

- AI quote: https://www.forbes.com/sites/peterhigh/2017/10/30/carnegie-mellon-dean-of-computer-science-on-the-future-of-ai/#3a6ad4f82197
- Gradient Descent, How Networks Learn https://www.youtube.com/watch?v=IHZwWFHWa
- Megatron 530 billion parameter: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/
- Backpropagation https://www.youtube.com/watch?v=3Kb0QS6z7WA
- More backpropagation https://brilliant.org/wiki/backpropagation/
- Autodesk generative design https://www.youtube.com/watch?v=CtYRfMzmWFU
- GAN http://www.lherranz.org/2018/08/07/imagetranslation/
- Quotes from Alan Turing, Jeff Dean https://www.import.io/post/history-of-deep-learning/

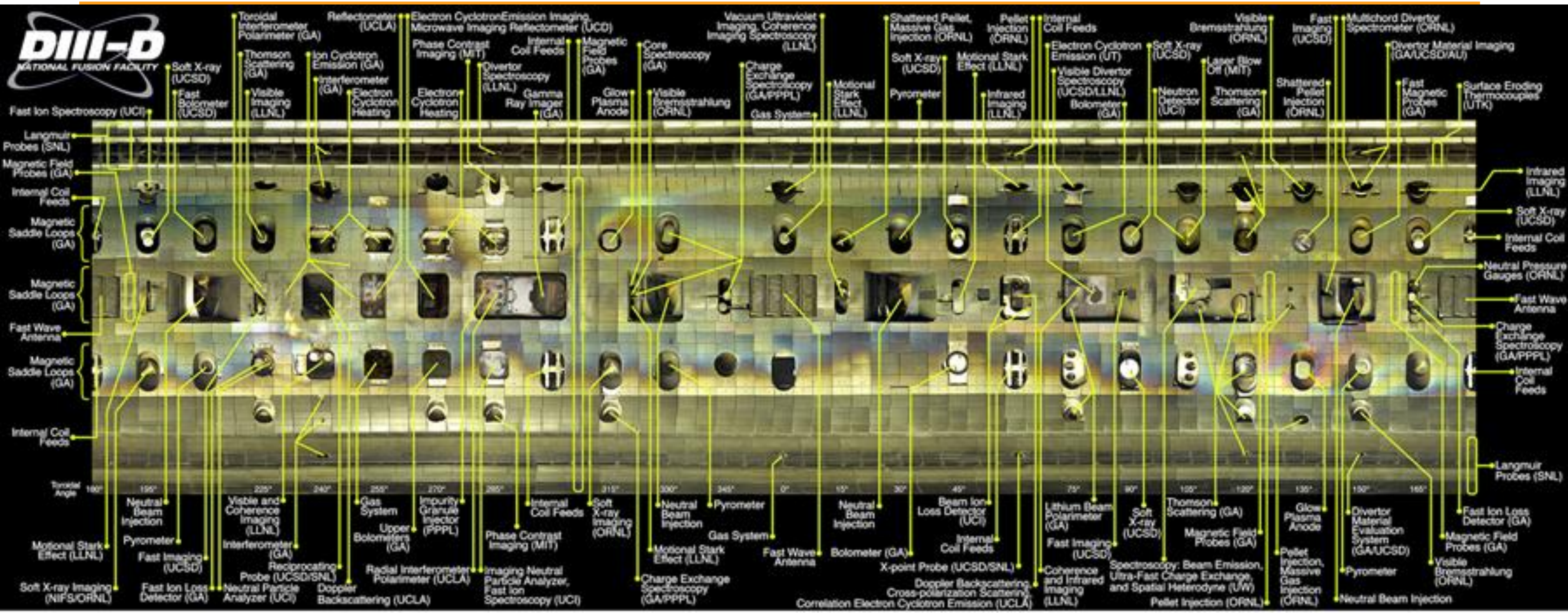
References Cont.

- Meena, Google chatbot https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html
- Hyperparameter list https://wandb.ai/site/articles/running-hyperparameter-sweeps-to-pick-the-best-model-using-w-b
- Hyperparameter and deep learning training rules of thumb https://jeffmacaluso.github.io/post/DeepLearningRulesOfThumb/
- Whats wrong with CNNs? https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b
- Transformer https://jalammar.github.io/illustrated-transformer/
- Transformer: "Attention is all you need" https://arxiv.org/abs/1706.03762
- Li, H., *et. al.* (2018). https://arxiv.org/pdf/1712.09913.pdf
- Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks" http://karpathy.github.io/2015/05/21/rnn-effectiveness/

References Cont.

- LSTM: Hochreiter and Schmidhuber, Neural Computation 9 (8): 1735-1780, 1997, https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory
- LSTM: https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
- Graph NN:
  - https://ericmjl.github.io/essays-on-data-science/machine-learning/graph-nets/
  - https://theaisummer.com/graph-convolutional-networks/
  - https://theaisummer.com/gnn-architectures/
  - Battaglia, https://arxiv.org/abs/1806.01261
  - Savannah Thais, https://drive.google.com/file/d/1b7T6Z-MoUId0qk5mBMY98IXM0CXy50RG/view
- PDE solving:
  - Pfaff https://arxiv.org/pdf/2010.03409.pdf

- Robin Walters, "Incorporating Symmetry into Deep Dynamics Models for Improved Generalization." ICLR 2021.
- Hoyer, https://www.pnas.org/content/pnas/118/21/e2101784118.full.pdf
- Wu, “Deep Transformer Models for Time Series Forecasting” https://arxiv.org/pdf/2001.08317.pdf


DIII-D
NATIONAL FUSION FACILITY
Fast Ion Spectroscopy (UCI)
Langmuir Probes (SNL)
Magnetic Field Probes (GA)
Internal Coil Feeds
Magnetic Saddle Loops (GA)
Fast Wave Antenna
Toroidal Interferometer Polarimeter (GA)
Thomson Scattering (GA)
Soft X-ray (UCSD)
Fast Bolometer (UCSD)
Visible Imaging (LLNL)
Reflectometer (UCLA)
Ion Cyclotron Emission (GA)
Interferometer (GA)
Electron Cyclotron Heating
Electron Cyclotron Emission Imaging, Microwave Imaging Reflectometer (UCD)
Phase Contrast Imaging (MIT)
Divertor Spectroscopy (LLNL)
Gamma Ray Imager (GA)
Magnetic Field Probes (GA)
Core Spectroscopy (GA)
Glow Plasma Anode
Visible Bremsstrahlung (ORNL)
Vacuum Ultraviolet Imaging, Coherence Imaging Spectroscopy (LLNL)
Charge Exchange Spectroscopy (GA/PPPL)
Gas System
Motional Stark Effect (LLNL)
Shattered Pellet, Massive Gas Injection (ORNL)
Pellet Injection (ORNL)
Soft X-ray (UCSD)
Motional Stark Effect (LLNL)
Pyrometer
Infrared Imaging (LLNL)
Internal Coil Feeds
Electron Cyclotron Emission (UT)
Visible Divertor Spectroscopy (UCSD/LLNL)
Bolometer (GA)
Soft X-ray (UCSD)
Neutron Detector (UCI)
Visible Bremsstrahlung (ORNL)
Laser Blow Off (MIT)
Thomson Scattering (GA)
Fast Imaging (UCSD)
Shattered Pellet Injection (ORNL)
Multichord Divertor Spectrometer (ORNL)
Divertor Material Imaging (GA/UCSD/AU)
Fast Magnetic Probes (GA)
Surface Eroding Thermocouples (UTK)
Infrared Imaging (LLNL)
Soft X-ray (UCSD)
Internal Coil Feeds
Neutral Pressure Gauges (ORNL)
Fast Wave Antenna
Charge Exchange Spectroscopy (GA/PPPL)
Internal Coil Feeds
Langmuir Probes (SNL)
Toroidal Angle
Neutral Beam Injection
Motional Stark Effect (LLNL)
Pyrometer
Fast Imaging (UCSD)
Soft X-ray Imaging (NIFS/ORNL)
Fast Ion Loss Detector (GA)
Visible and Coherence Imaging (LLNL)
Interferometer (GA)
Reciprocating Probe (UCSD/SNL)
Neutral Particle Analyzer (UCI)
Gas System
Upper Bolometers (GA)
Doppler Backscattering (UCLA)
Impurity Granule Injector (PPPL)
Radial Interferometer Polarimeter (UCLA)
Phase Contrast Imaging (MIT)
Imaging Neutral Particle Analyzer, Fast Ion Spectroscopy (UCI)
Internal Coil Feeds
Soft X-ray Imaging (ORNL)
Neutral Beam Injection
Motional Stark Effect (LLNL)
Charge Exchange Spectroscopy (GA/PPPL)
Pyrometer
Gas System
Fast Wave Antenna
Neutral Beam Injection
Bolometer (GA)
Beam Ion Loss Detector (UCI)
Internal Coil Feeds
X-point Probe (UCSD/SNL)
Doppler Backscattering, Cross-polarization Scattering
Correlation Electron Cyclotron Emission (UCLA)
Lithium Beam Polarimeter (GA)
Fast Imaging (UCSD)
Coherence and Infrared Imaging (LLNL)
Soft X-ray (UCSD)
Thomson Scattering (GA)
Magnetic Field Probes (GA)
Spectroscopy: Beam Emission, Ultra-Fast Charge Exchange, and Spatial Heterodyne (UW)
Pellet Injection (ORNL)
Glow Plasma Anode
Pellet Injection, Massive Gas Injection (ORNL)
Divertor Material Evaluation System (GA/UCSD)
Pyrometer
Neutral Beam Injection
Fast Ion Loss Detector (GA)
Magnetic Field Probes (GA)
Visible Bremsstrahlung (ORNL)