Day 6: Modern Neural Networks

Presenter: Auralee Edelen

Day 6

What is a Neural Network?

Artificial Intelligence (AI)

- *How to enable machines to exhibit aspects of "intelligence"*
- *knowledge, learning, planning, reasoning, perception*

Machine Learning (ML)

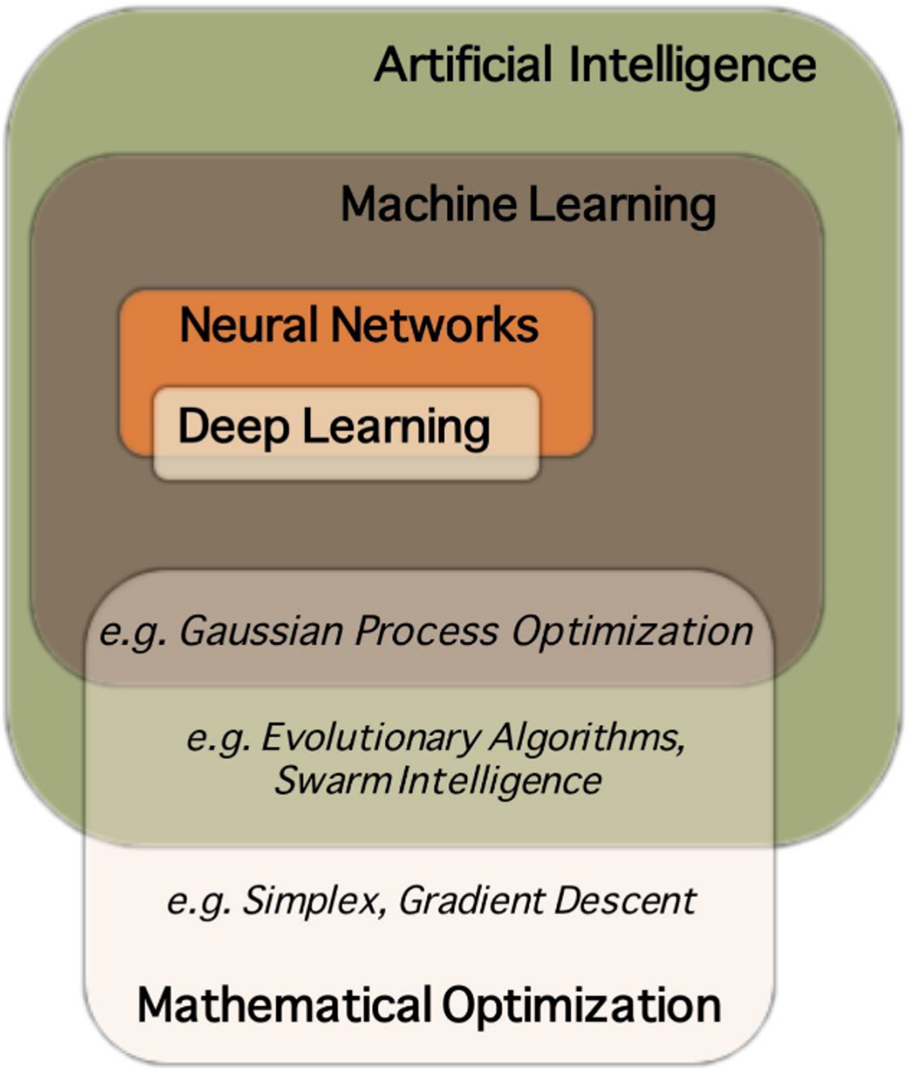
- *Use learned representations to complete tasks without being explicitly programmed*
- Tasks: Regression, Classification, Dimensionality Reduction, etc.

Neural Networks (NNs)

- *Class of ML structures that use many connected processing units to learn input/output maps (used to be called "connectionism")*

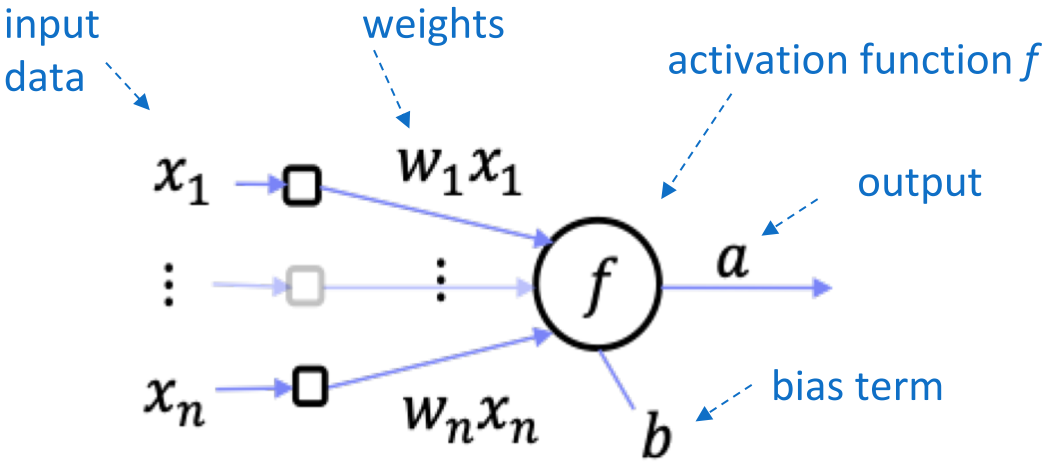
Deep Learning (DL)

- *Learning hierarchical representations*
- Right now, largely synonymous with deep (many-layered) NNs



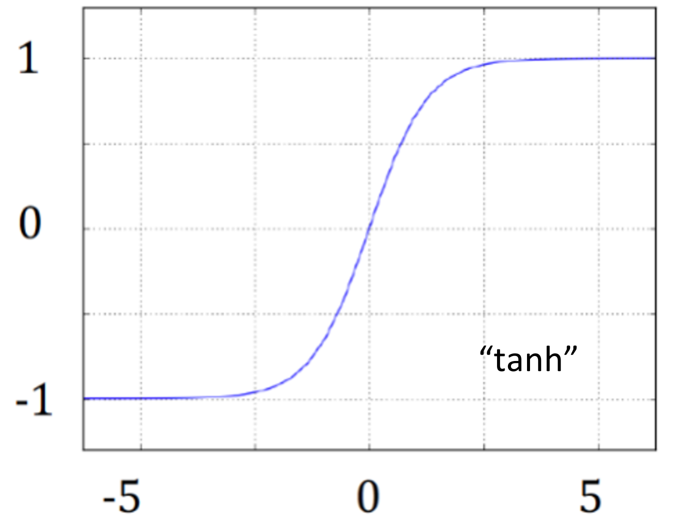
Ingredients of a Neural Network: Individual Nodes

a neuron or node:



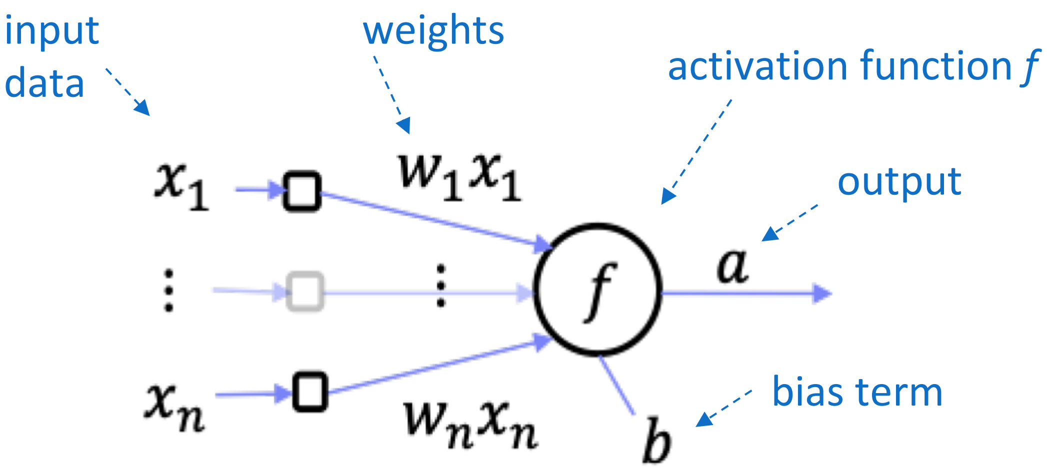
$$f\left(\sum_n w_n x_n + b\right) = a$$

e.g. $f(z) = \frac{2}{(1+e^{-2z})} - 1$

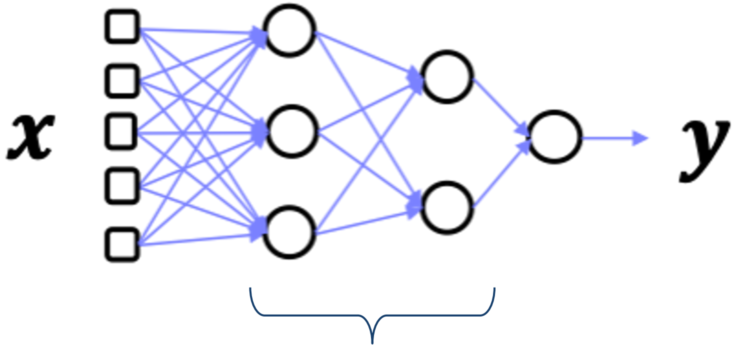


Ingredients of a Neural Network: Connecting Individual Nodes

a neuron or node:



a neural network:



For layer l consisting of j nodes and a previous layer $l-1$ consisting of i nodes the output of the j^{th} node in layer l is

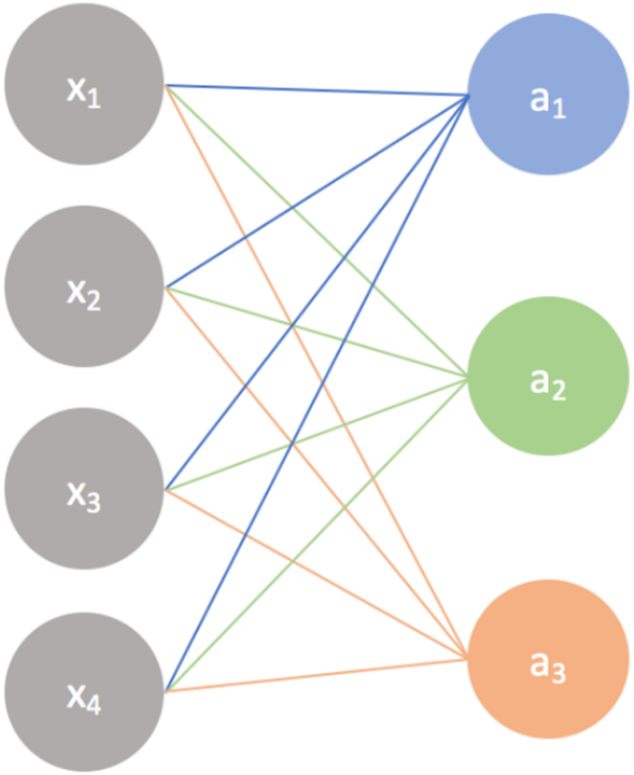
$$a_j^l = f\left(\sum_{i=i}^{n} w_{ji}^l a_i^{l-1} + b_j^l\right)$$

often mathematically expressed by matrices

$$\begin{bmatrix} w_{0,0}^l & w_{1,0}^l \\ w_{0,1}^l & w_{1,1}^l \end{bmatrix} \quad \begin{bmatrix} b_0^l \\ b_1^l \end{bmatrix} \qquad a_j^l = f(w_j^l a^{l-1} + b_j^l)$$

$$A^l = f(W^l A^{l-1} + b^l)$$

Ingredients of a Neural Network: Connecting Individual Nodes



$$A^l = f(W^l A^{l-1} + b^l)$$

$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[\text{activation}]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

https://www.jeremyjordan.me/intro-to-neural-networks/

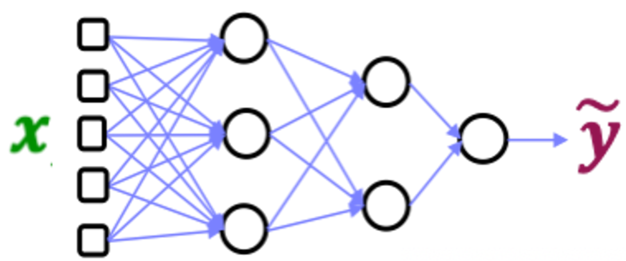
Training

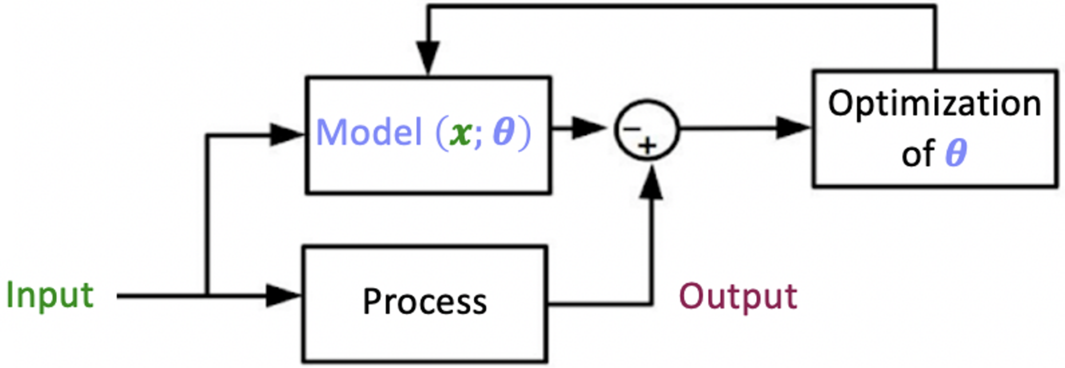
Training → optimization of model parameters

(usually just weights / biases but can include architecture as well)

Data set of N **input** and **output** samples (can be vectors)

$$\left.\begin{array}{c} x_1 \\ \vdots \\ x_n \end{array}\right\} \quad \begin{array}{cc} \boldsymbol{x}_1 & \boldsymbol{y}_1 \\ \vdots & \vdots \\ \boldsymbol{x}_N & \boldsymbol{y}_N \end{array}$$





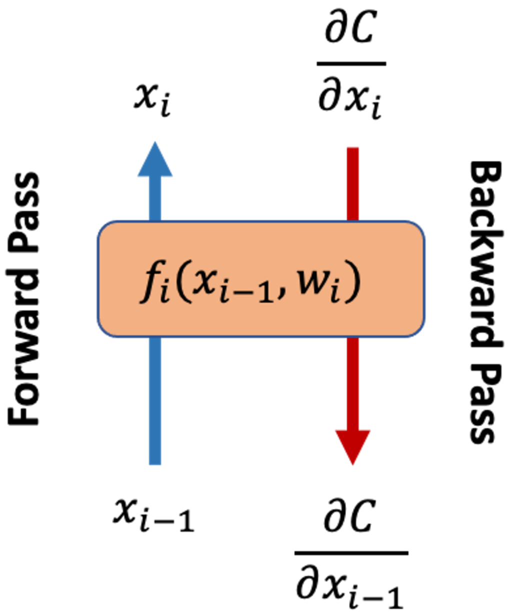
Goal is to find approximate map $\tilde{g}(\boldsymbol{x};\ \boldsymbol{\theta}) = \tilde{\boldsymbol{y}}$

Training: Back-propagation

Backpropagation: propagate the gradient of the cost function backward through the network

→ *essentially, the chain rule*

→ *update each weight and bias according to corresponding contribution to gradient*



$$C(w,b) = \frac{1}{2N}\left[\sum_N (y_N - \tilde{y}_N)^2\right]$$

$$w_k \longrightarrow w'_k = w_k - \alpha \frac{\partial C}{\partial w_k}$$

$$b_k \longrightarrow b'_k = b_k - \alpha \frac{\partial C}{\partial b_k}$$

ML libraries use **automatic differentiation** to make this faster/easier:

-Theano -Tensorflow -Torch

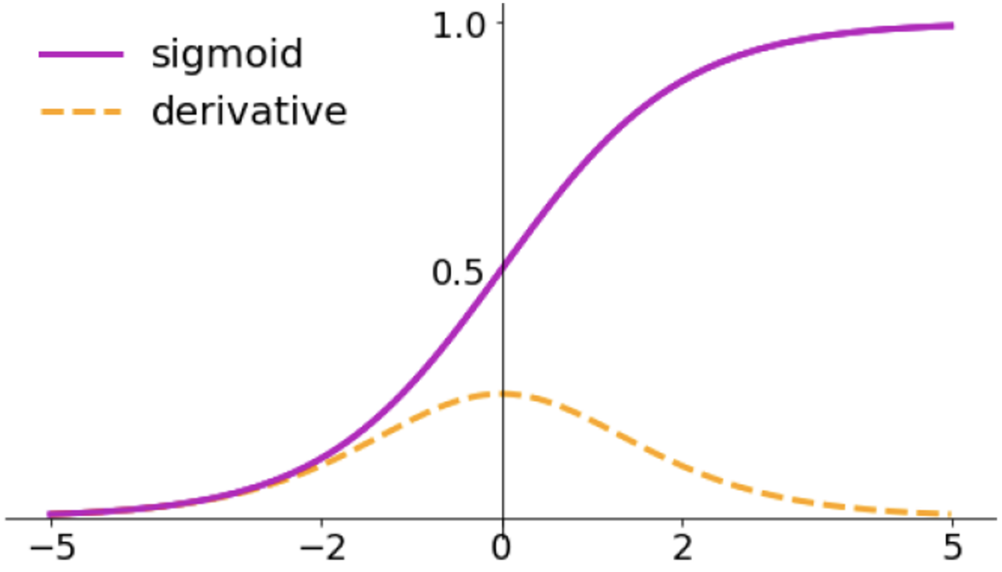
For detailed exposition on backpropagation, see: http://neuralnetworksanddeeplearning.com/chap2.html

Activation Functions

Sigmoid or Logistic

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$



Squashes output to range [0,1]
Historical conceptual appeal: saturation and firing rate of a neuron

Problems:

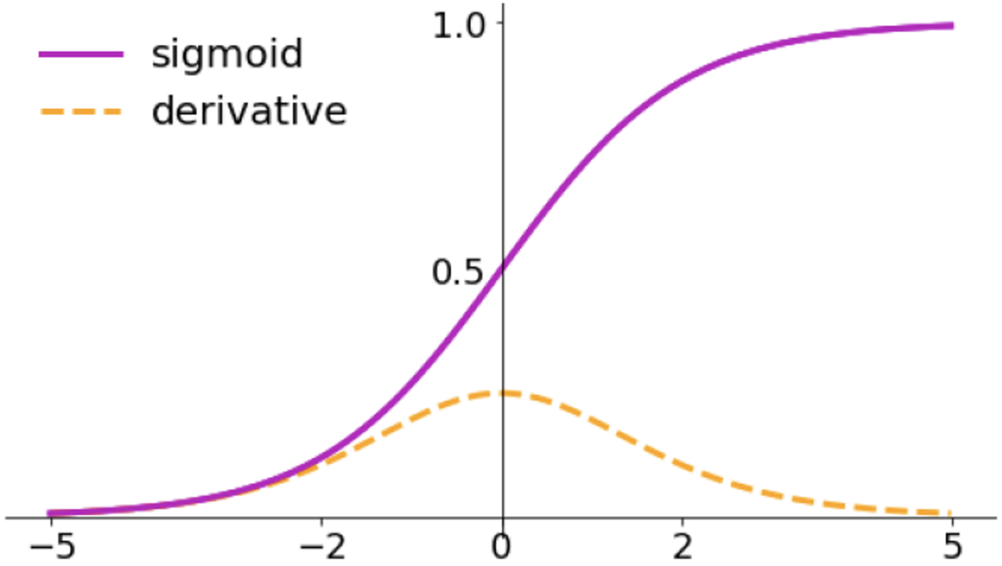
- Saturated neurons kill gradients
- Not zero-centered
- Computational expense

Activation Functions

Sigmoid or Logistic

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

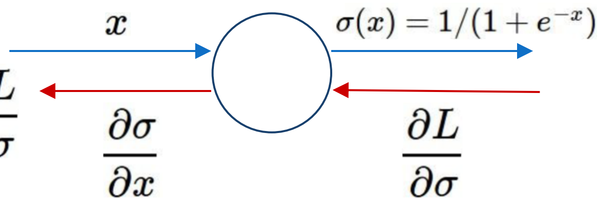


Squashes output to range [0,1]
Historical conceptual appeal: saturation and firing rate of a neuron

Problems:

- Saturated neurons kill gradients
- Not zero-centered
- Computational expense

$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma}$$



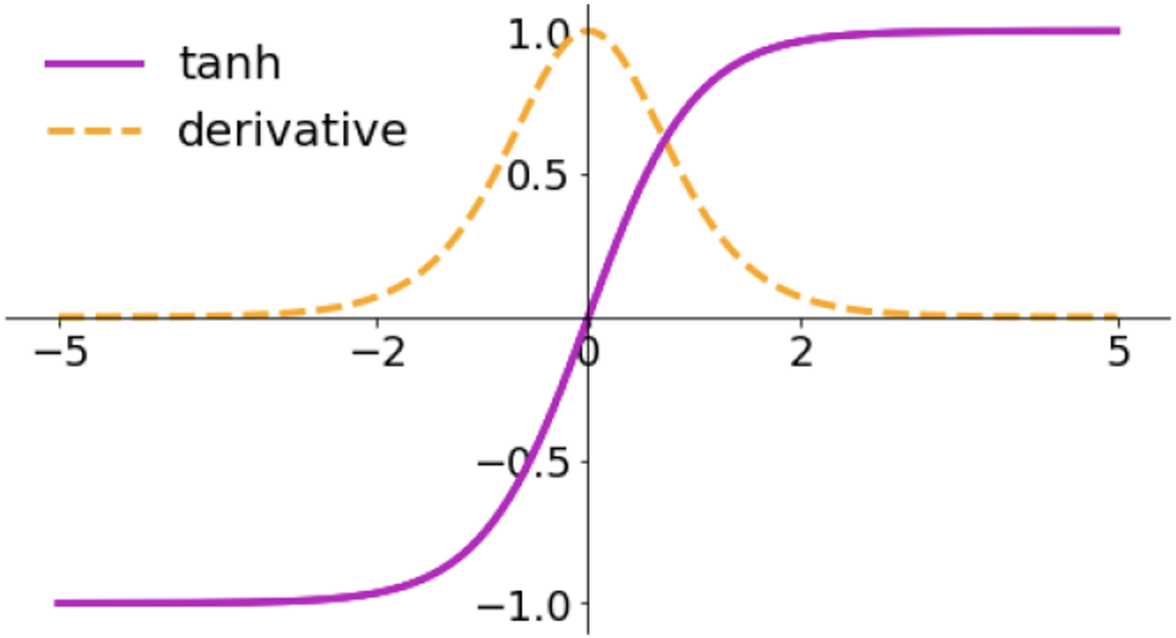
What happens when x = -10?

Activation Functions

Hyperbolic Tangent

$$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

$$f'(x) = 1 - f(x)^2$$



Squashes output to range [-1,1]
Zero-centered

Often approximated version is used to improve computation speed
Still has saturation problem → *important to scale data to -1 to 1 range!*

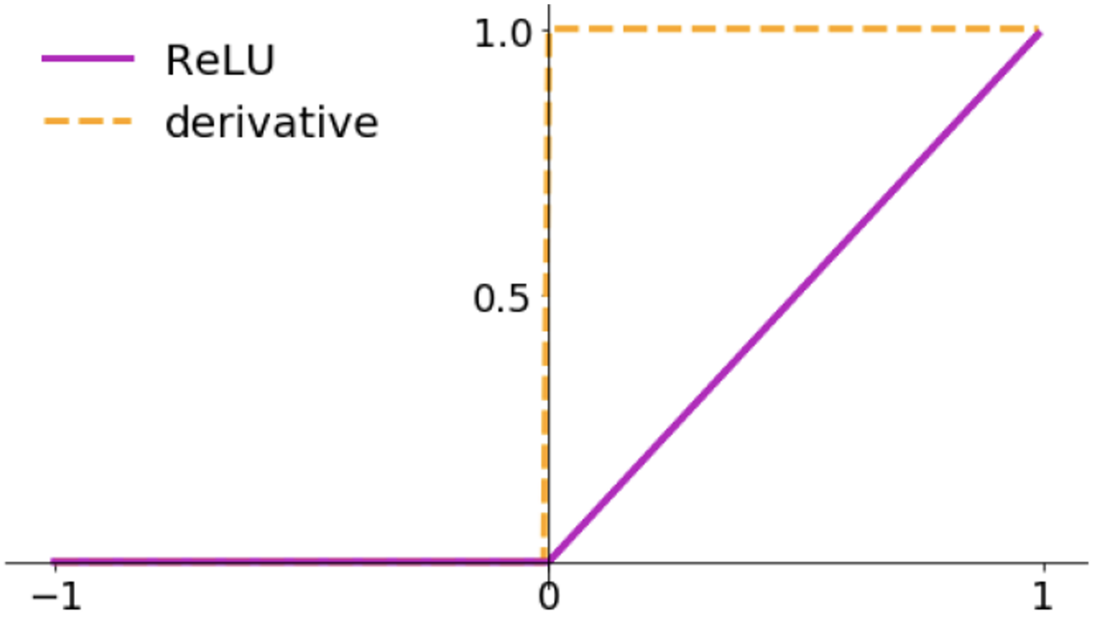
LeCun et al., 1991

Activation Functions

Rectified Linear Unit (ReLU)

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max\{0, x\}$$

$$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$$



Does not saturate for positive values
Computationally efficient
Converges faster than sigmoid/tanh

Problems

- Not zero-centered output
- Dying ReLUs

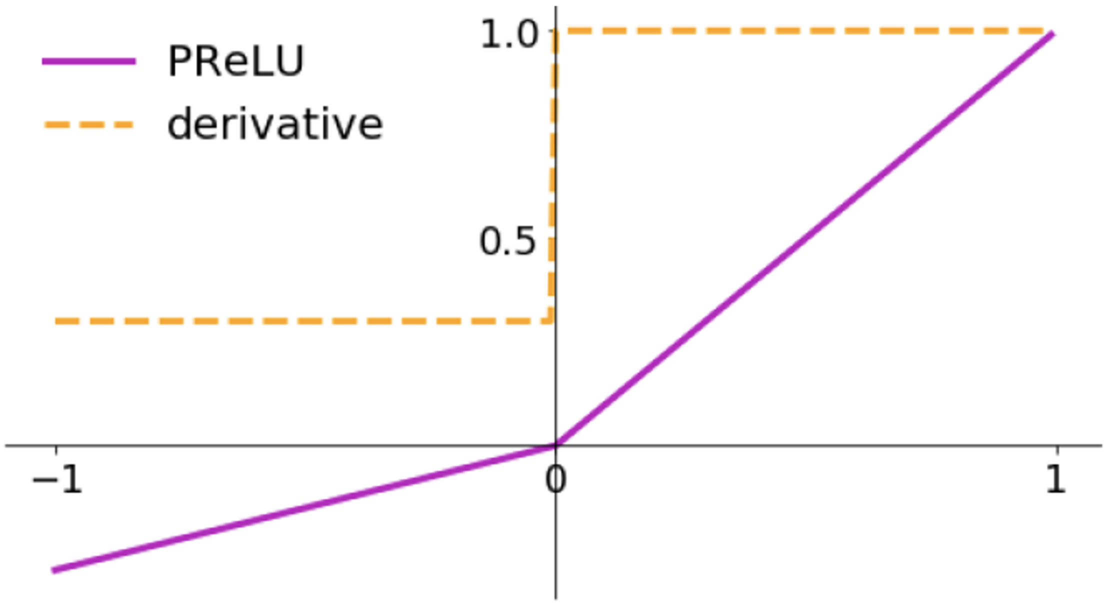
Krizhevsky et al., 2012

Activation Functions

Parameterized Rectified Linear Unit (PReLU)

$$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$



“leaky ReLU” alpha = 0.01

Does not saturate
Computationally efficient
Converges faster than sigmoid/tanh in practice
No dying!

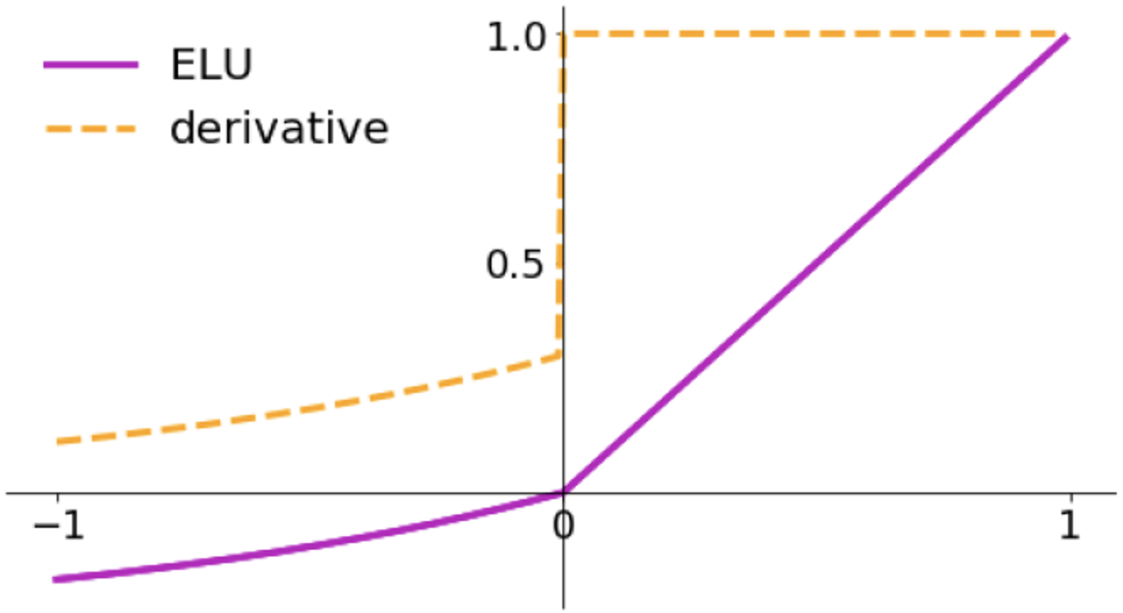
Mass et al., 2013
He et al., 2015

Activation Functions

Exponential Linear Unit (ELU)

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \le 0 \\ x & \text{for } x > 0 \end{cases}$$

$$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \le 0 \\ 1 & \text{for } x > 0 \end{cases}$$



Same benefits as ReLU
Closer to zero mean outputs

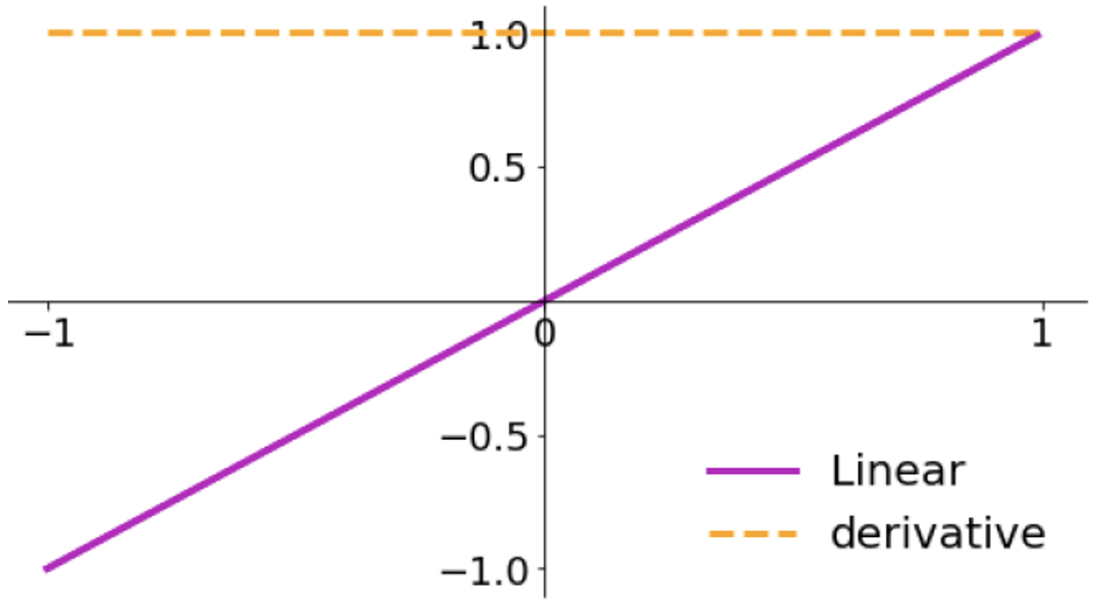
Clevert et al., 2015

Activation Functions

Linear

$$f(x) = x$$

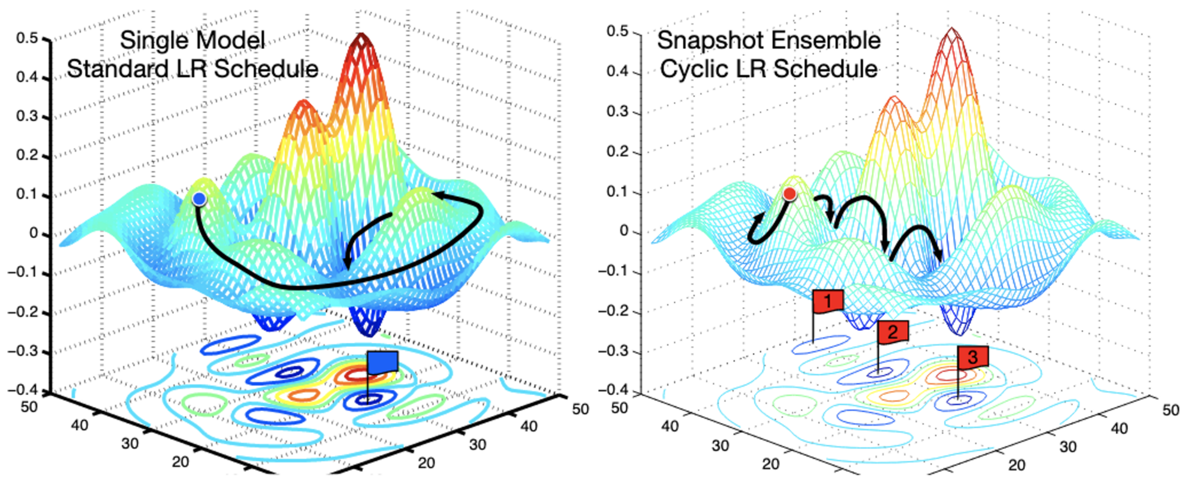
$$f'(x) = 1$$



For unbounded regression: often used on the last layer

Neural networks have many parameters

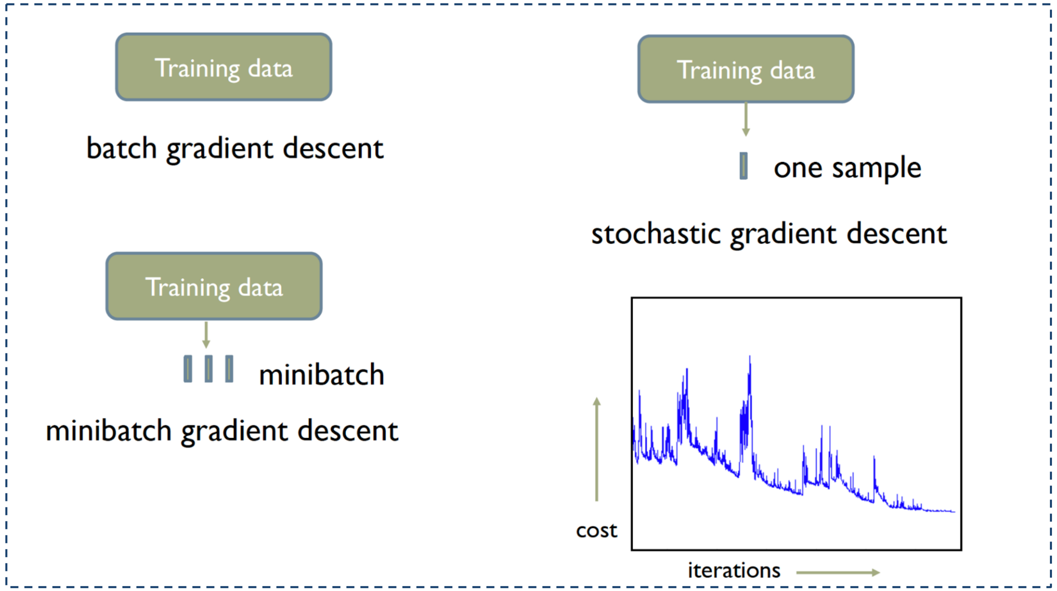
→ *complicated error surfaces with many local minima*



https://arxiv.org/pdf/1704.00109.pdf

Primarily use mini-batch training:

- Gradient noise is useful for jumping out of local minima
- Reduce memory size + compute for high-D data (e.g. images)
- Batch size is a significant training hyperparameter: *some evidence that smaller batches actually help generalization*



Very open area of research over decades:

- how to choose training and initialization techniques that give good generalization?

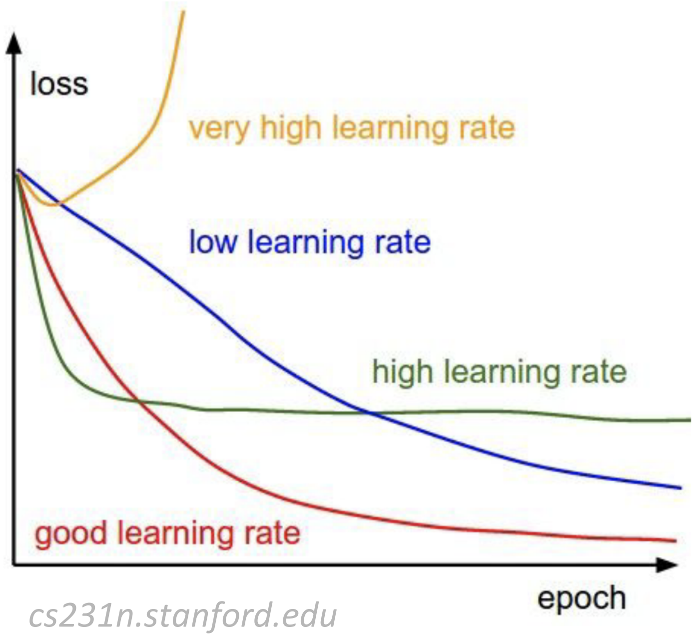
Training: setting the learning rate

Guess the initial learning rate:

- Error worse or oscillating
  → *reduce rate*

- Error decreasing slowly
  → *increase rate*

Too large of a learning rate at the start will make weight magnitudes large
→ *error derivatives in intermediate layers small*



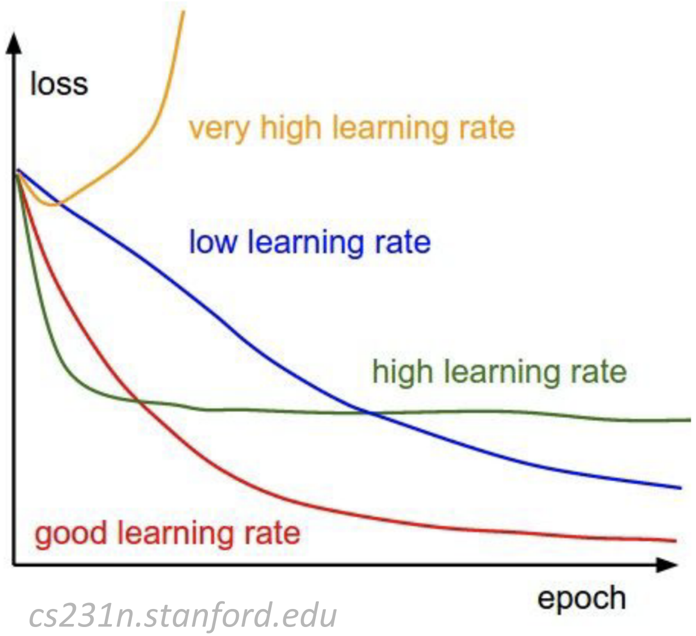
cs231n.stanford.edu

Training: setting the learning rate

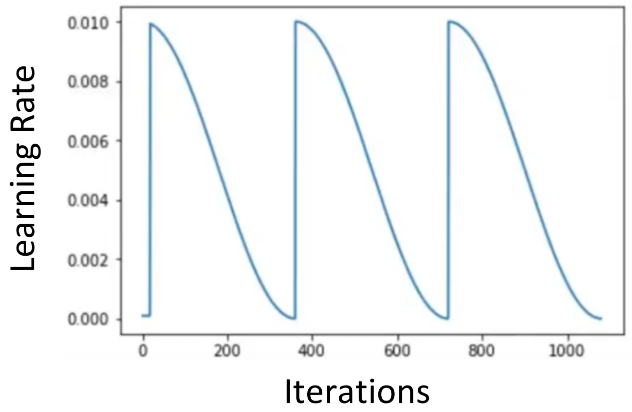
Guess the initial learning rate:

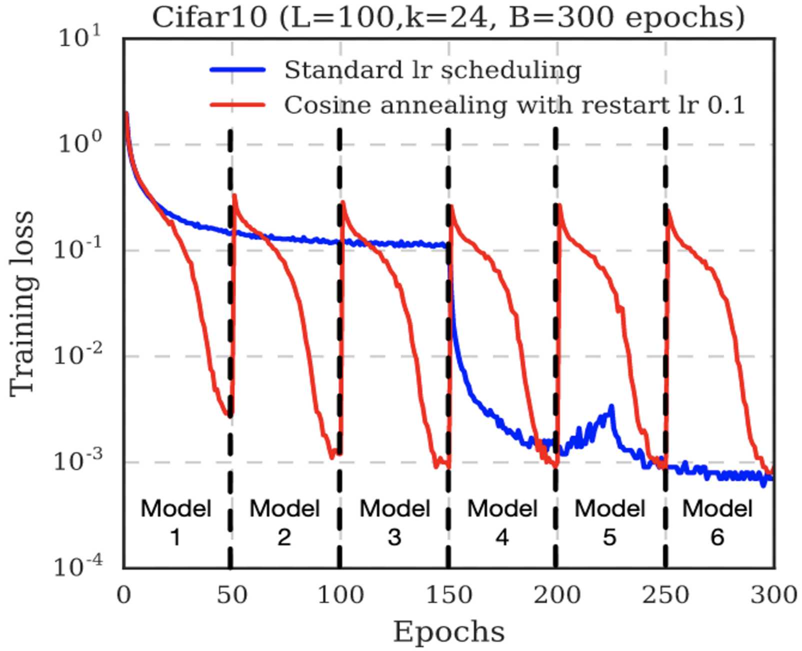
- Error worse or oscillating
  → *reduce rate*
- Error decreasing slowly
  → *increase rate*

Too large of a learning rate at the start will make weight magnitudes large
→ *error derivatives in intermediate layers small*



cs231n.stanford.edu





https://arxiv.org/pdf/1704.00109.pdf

Anneal (reduce learning rate) toward the end of training

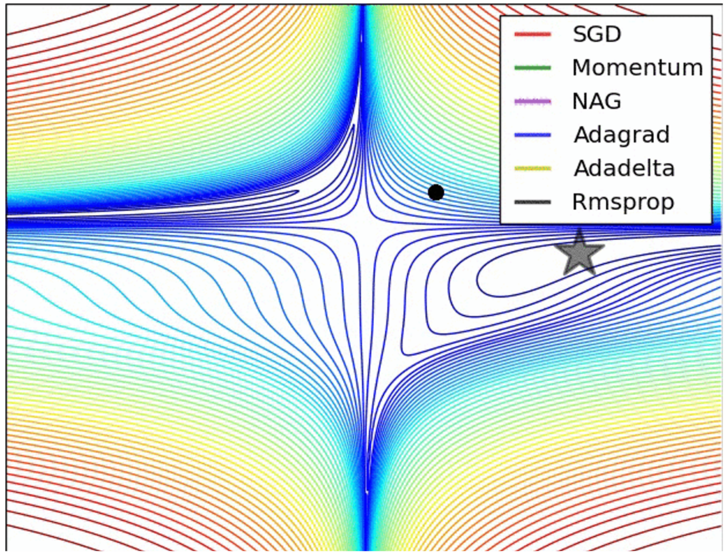
- *Lowers fluctuations due to noise in gradient between mini-batches*
- *Exponential learning rate decay is common*

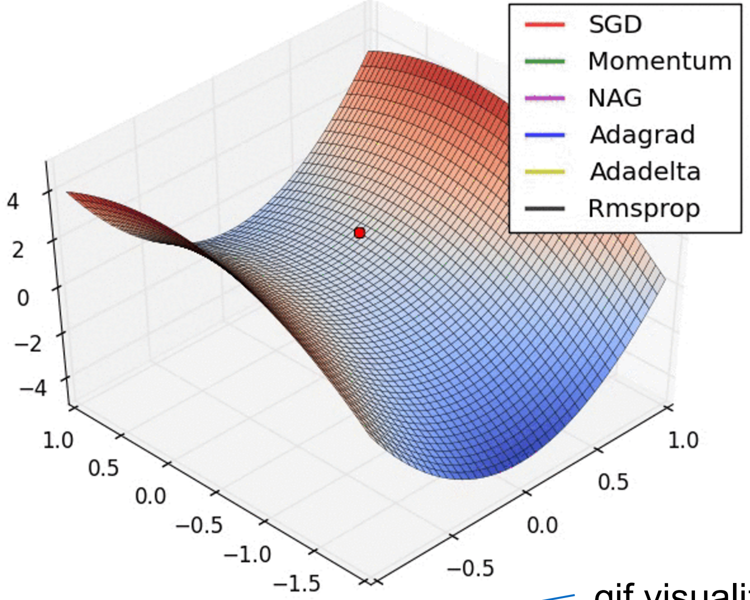
annealing rules = annealing or decay "schedule"

Training: common optimization algorithms for neural networks

Variants of gradient descent

- **Momentum** – *add portion of past update vector to present vector ("add a velocity term")*
- **Nesterov accelerating gradient** – *update with next set of parameters ("look ahead + slow down before a hill")*
- **Adagrad** – *different learning rate for every parameter based on past gradients*
- **Adadelta / RMSProp** – *similar to Adagrad but with decaying influence of past gradients to help stabilize learning rate*
- **Adam (adaptive momentum estimation)** – *adaptive learning rate, decaying average of past gradients + momentum-like term*
- **Nadam** – *Adam with nesterov*





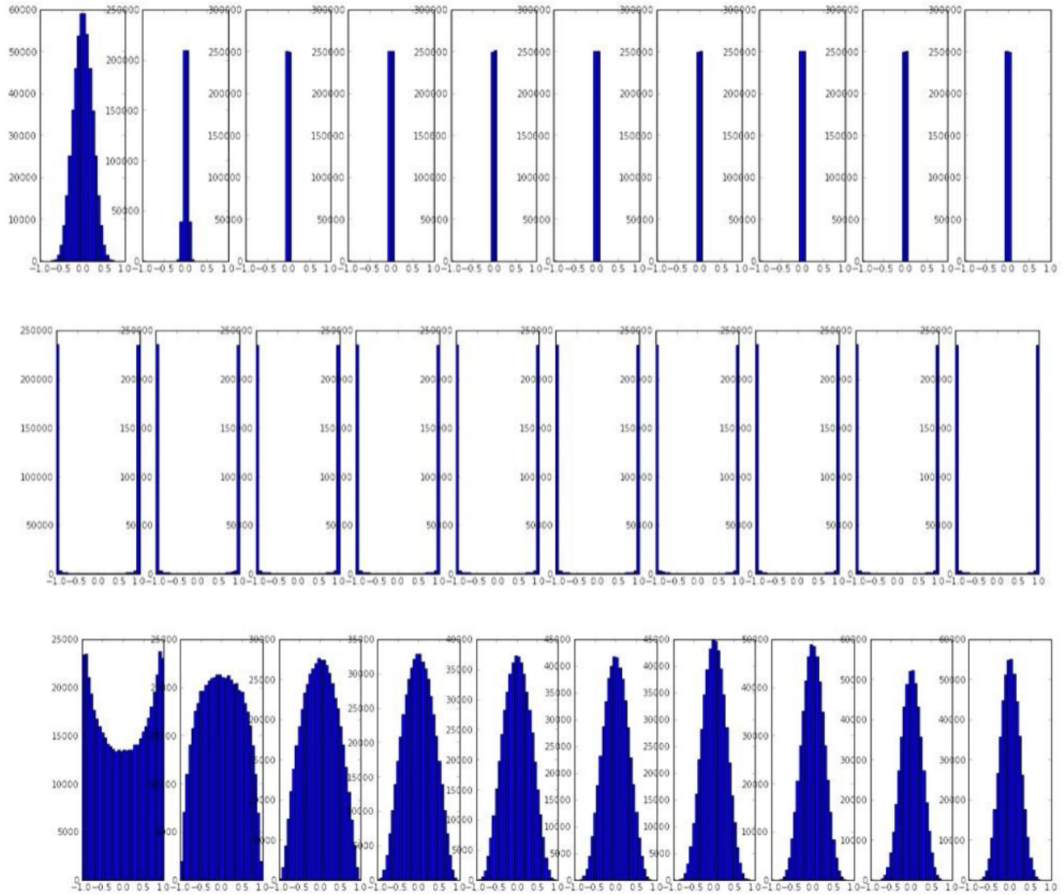
In practice Adam works very well for a lot of problems
Can also use 2nd order (e.g. L-BFGS) → better for smaller networks/ data sets

gif visualizations

For more detail see: http://cs231n.github.io/neural-networks-3/#sgd , https://ruder.io/optimizing-gradient-descent/

Training: weight Initialization

Weight initialization: random values that the weights start at



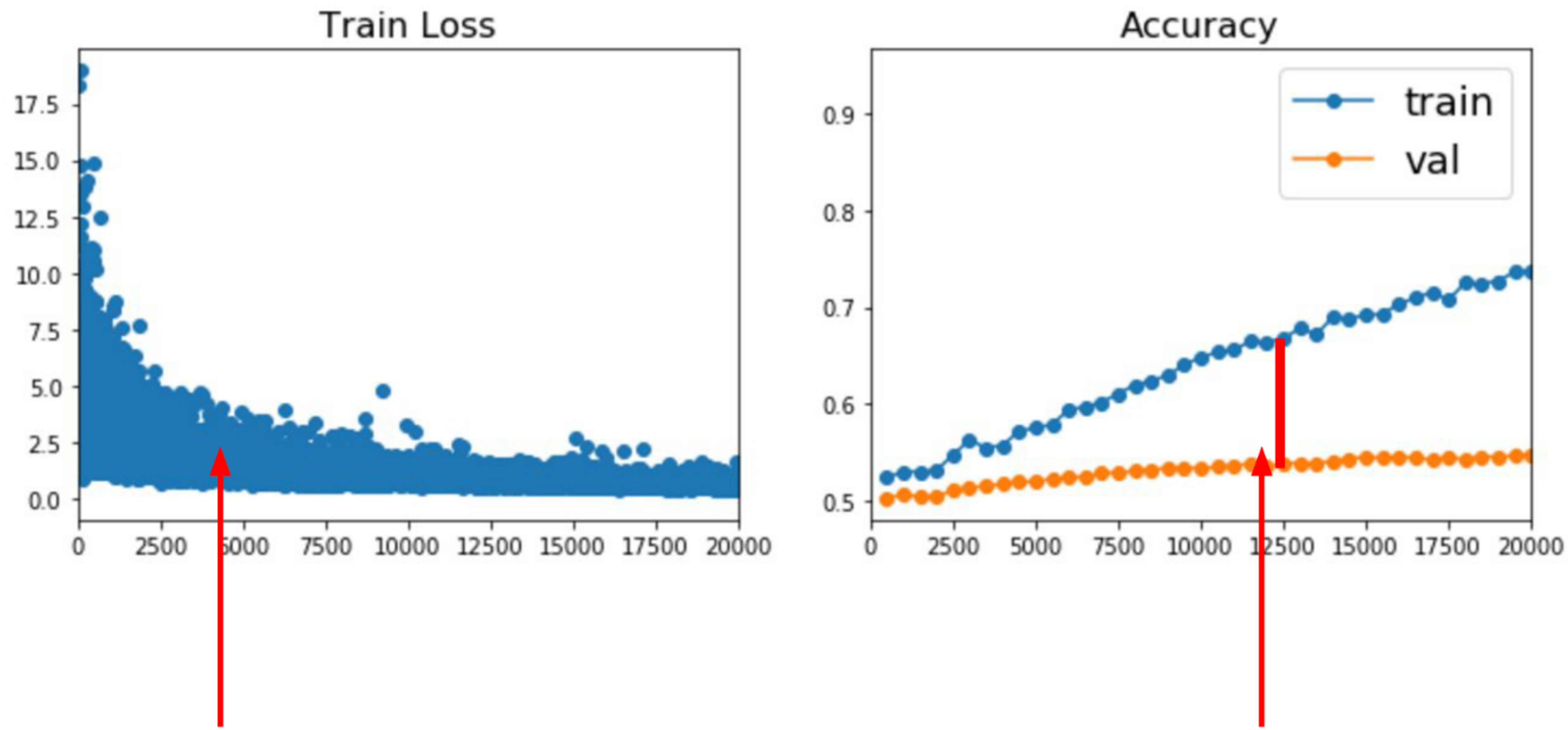
Initialization too small:
Activations go to zero, gradients also zero,
No learning

Initialization too big:
Activations saturate (for tanh),
Gradients zero, no learning

Initialization just right:
Nice distribution of activations at all layers,
Learning proceeds nicely

Slide credit: cs231n.stanford.edu

Training: Generalization and Overfitting

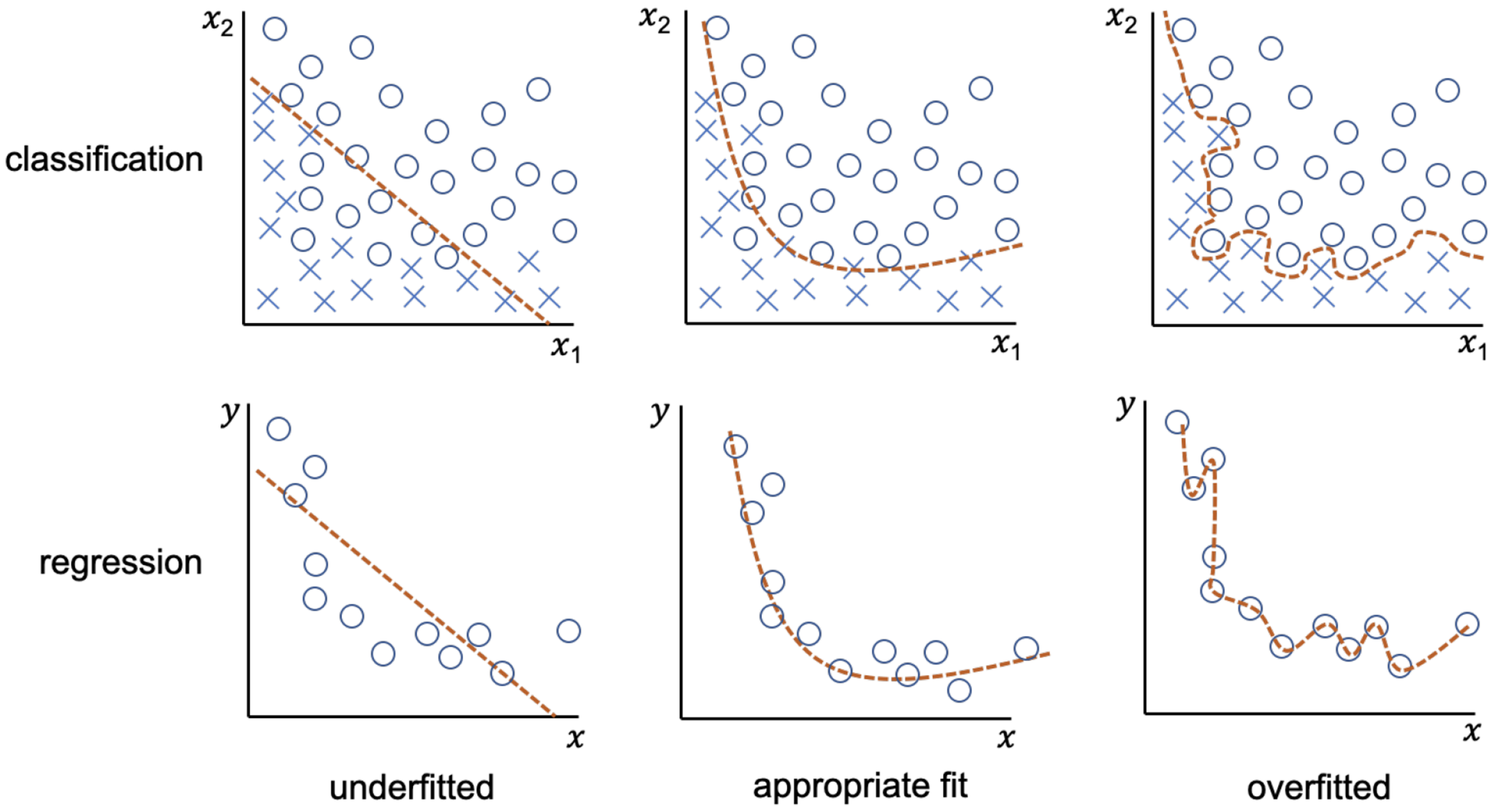


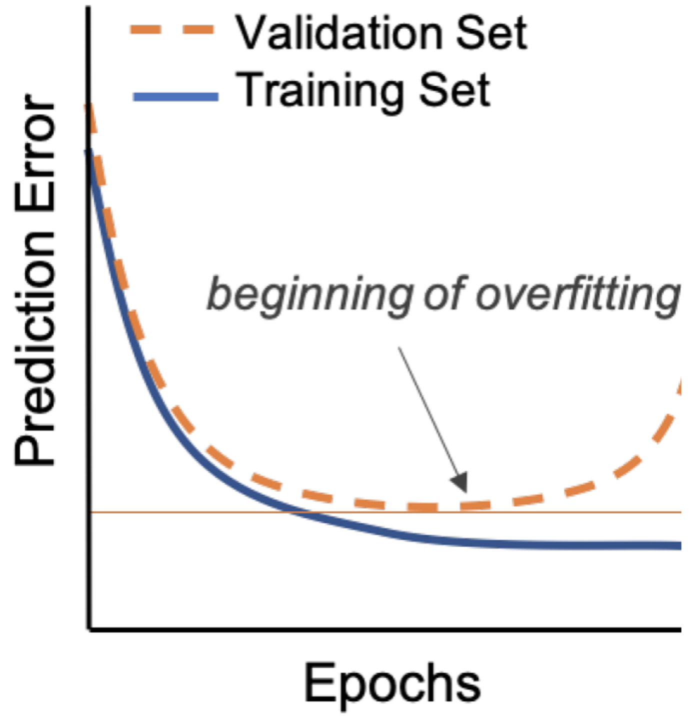
Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

Training: Generalization and Overfitting

Monitor the learning curve to assess overfitting





Does require that training and validation samples are well chosen (and also not oversampled)

Regularization with L1 and L2 norm

Penalize the magnitude of the weights in the cost function

$$||w||_p = \left(\sum_i |w_i|^p\right)^{\frac{1}{p}}$$ p-norm

$$L1 = ||w||_1 = \sum_i |w_i|$$ L1-norm

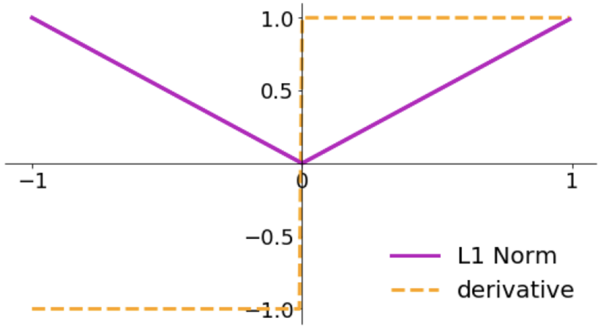
$$L2 = ||w||_2 = \left(\sum_i |w_i|^2\right)^{\frac{1}{2}}$$ L2-norm

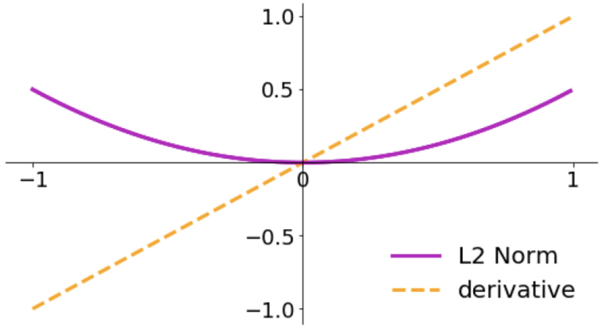
$$C_{reg} = E(y, \tilde{y}) + \lambda ||w||_{(1,2)}$$

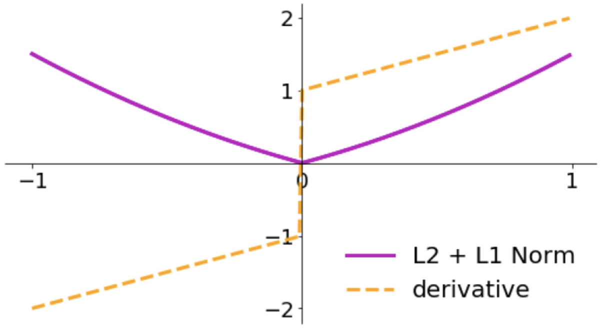
prediction error metric — weight (usually << 0)

L1-norm promotes sparsity → *pushes weights toward 0*

L2-norm promotes weight sharing → *pushes weights to small distribution around 0*



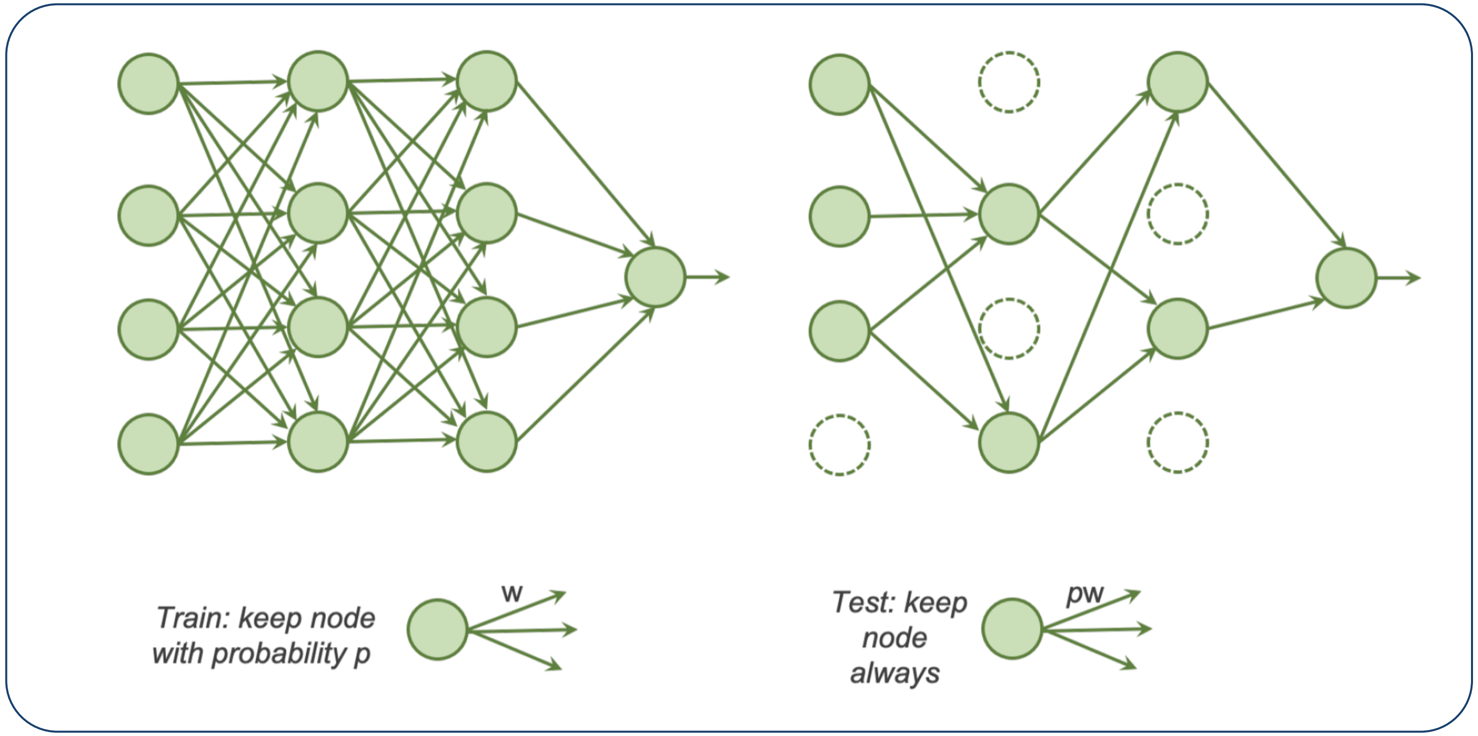




Regularization with Dropout

Dropout

- During each forward pass in training, with some probability drop a given node
- At inference time, retain all nodes and scale according to drop-out probability
- See Srivastava et al. (2014): https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf



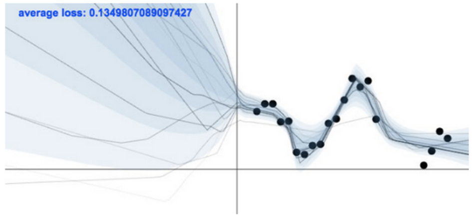
Adapted from Srivastavas et al., (2014)

How does this help us prevent overfitting?

- Encourages representation sharing between nodes → *acts a bit like an ensemble*
- Prevents co-adaptation of features

cs231n.stanford.edu

Bonus: can also be used for uncertainty estimates



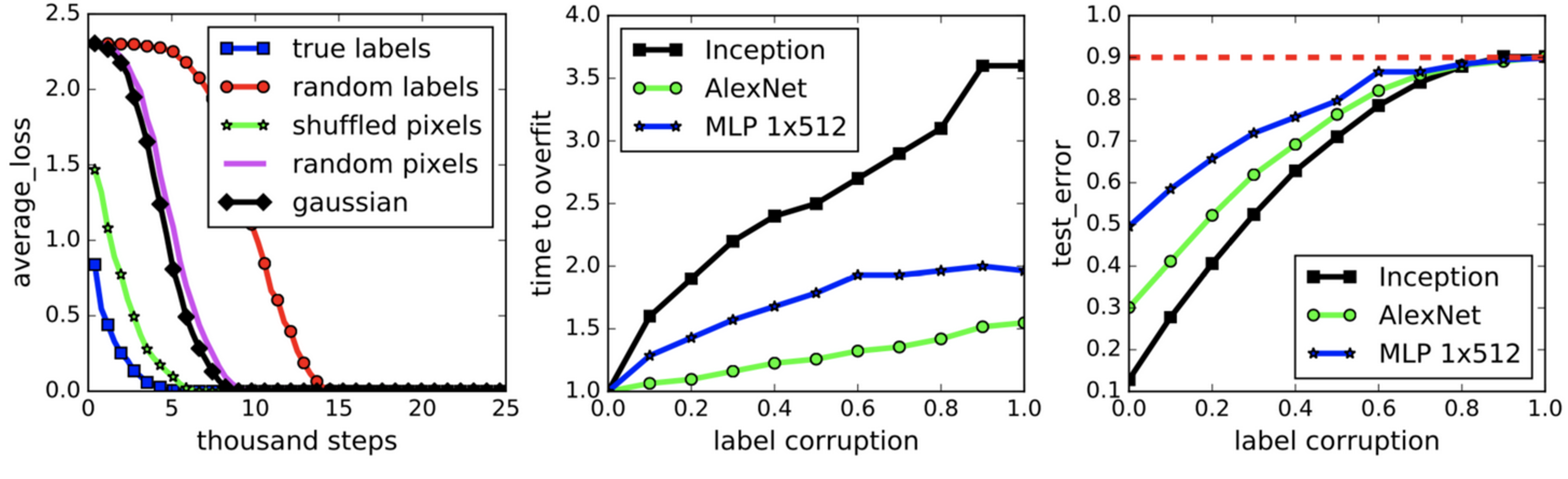
See Yarin Gal's thesis: http://www.cs.ox.ac.uk/people/yarin.gal/website/blog_2248.html

Summary: Combatting Overfitting

Summary of approaches for combating overfitting:

- Penalize weight magnitudes in cost function
- Dropout
- Add noise to each iteration (e.g. noise layers)
- Add more (diverse) data
- Reduce model complexity
- Ensembling (average output of many models)

Aside: Neural networks can learn entirely random noise



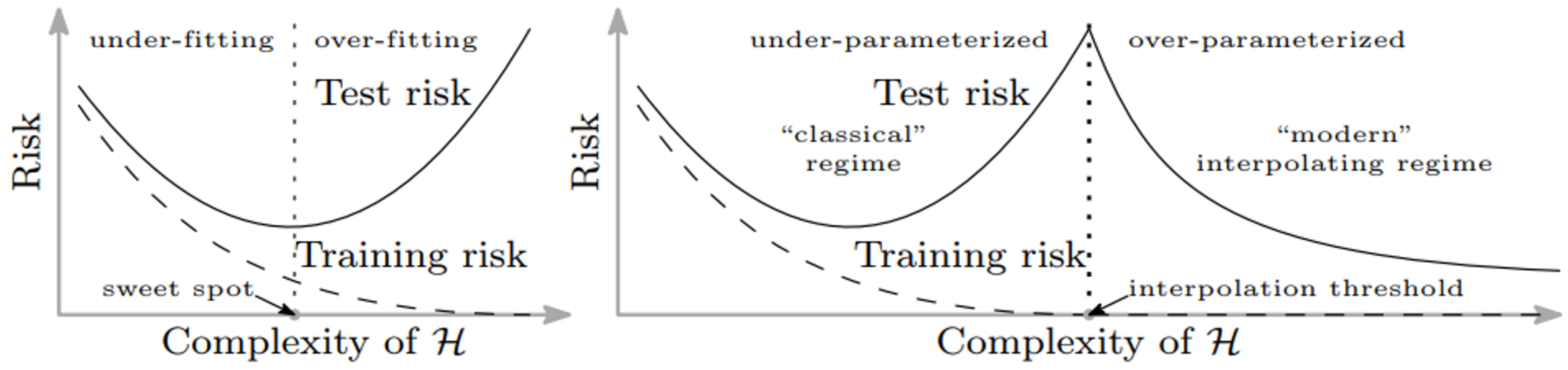
(a) learning curves (b) convergence slowdown (c) generalization error growth

Zhang, Understanding deep learning requires rethinking generalization (2017)
https://arxiv.org/pdf/1611.03530.pdf

Aside: Overparameterization

Belkin et al, (2018): https://arxiv.org/abs/1812.11118

Preetum et al, (2019): https://mltheory.org/deep.pdf



(a) U-shaped "bias-variance" risk curve

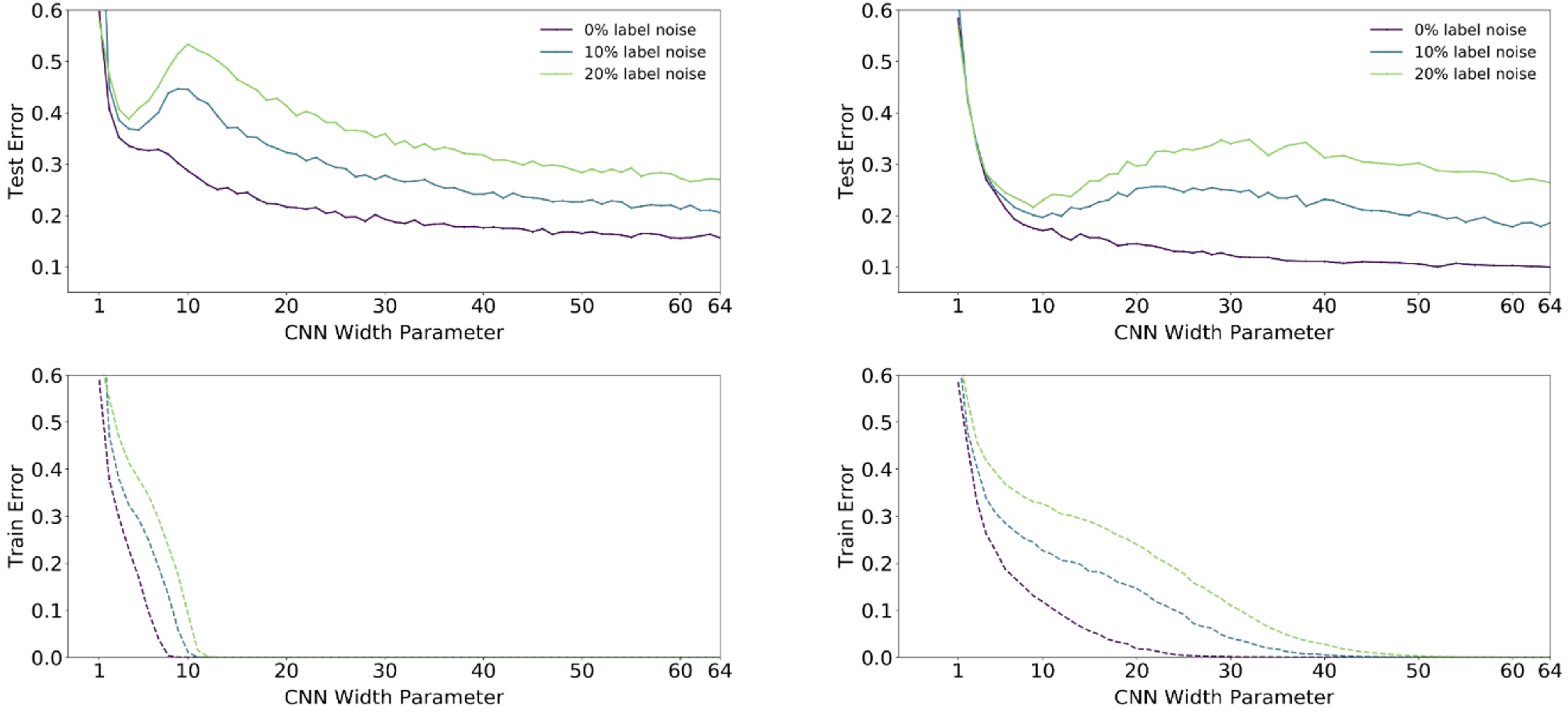
(b) "double descent" risk curve

Figure 1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (b) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high complexity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Aside: Overparameterization

Belkin et al, (2018): https://arxiv.org/abs/1812.11118

Preetum et al, (2019): https://mltheory.org/deep.pdf



(a) Without data augmentation.

(b) With data augmentation.

How to choose the architecture / model hyperparameters?

Variety of Approaches

- Rules of thumb → *then iterate depending on whether overfitting or underfitting*
  - http://dstath.users.uth.gr/papers/IJRS2009_Stathakis.pdf
- Grid search / random search
- Bayesian optimization
- Weight training and architecture search together using heuristic methods
  - Neuro Evolution of Augmenting Topologies (NEAT)
- Neural architecture search is an open area of research

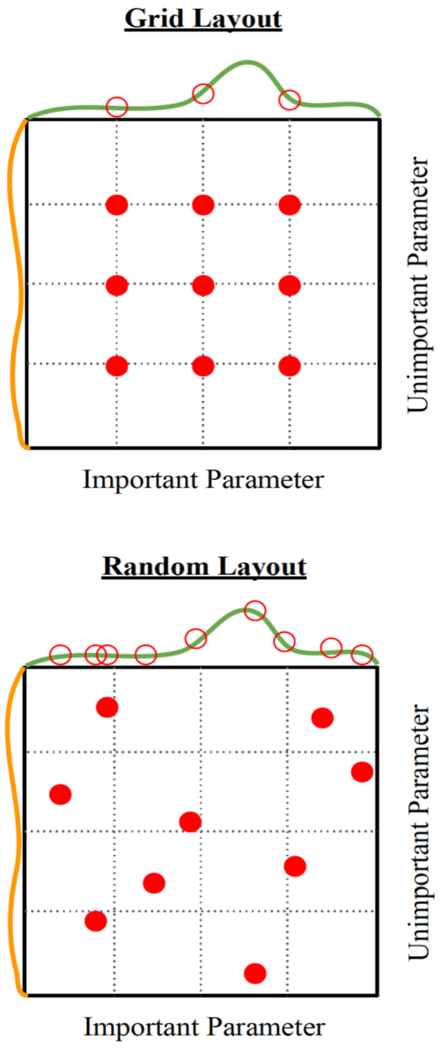
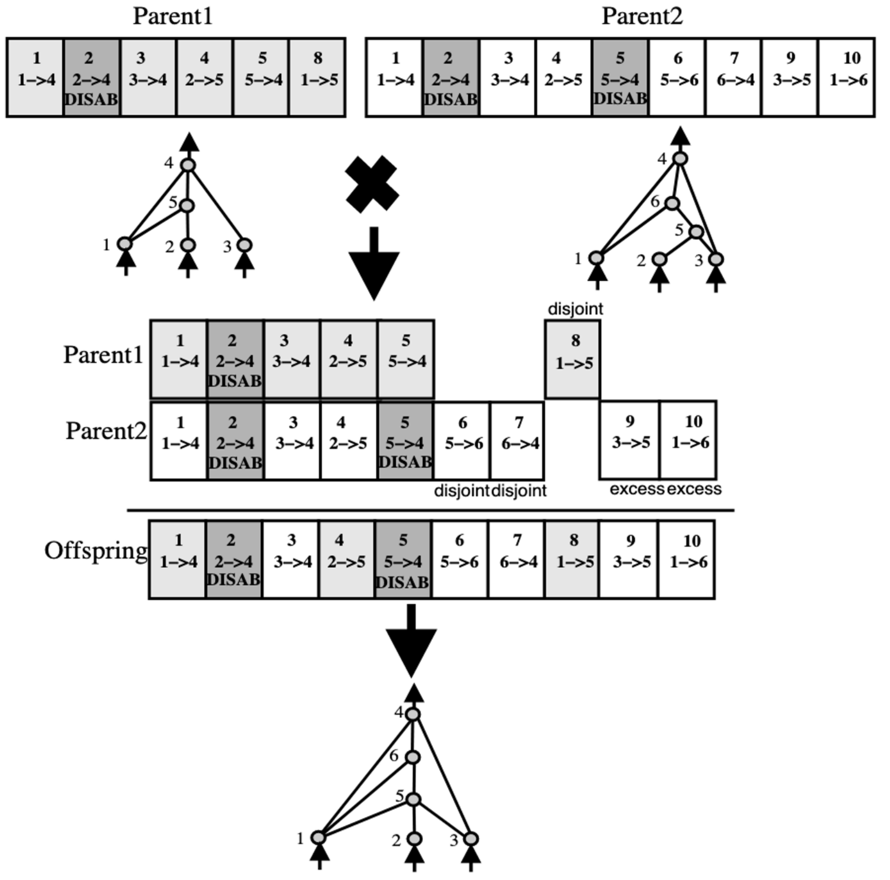


Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017



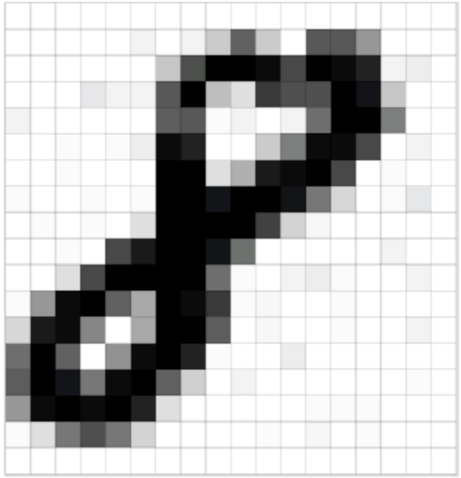
Bergstra, Random Search for Hyperparameter Optimization (2012): https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Stanley, Neuro Evolution of Augmenting Topologies (2002): http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

Summary: hyperparameters to tune

- Network structure (number/types of nodes, layers, activation functions)
- Optimization algorithm (and its hyperparameters)
  - *Learning rate initialization and decay*
  - *Momentum terms*
  - *Update rule*
- Batch size
- Regularization (e.g. noise, l1 and l2 weight penalties, dropout)

Computer Vision

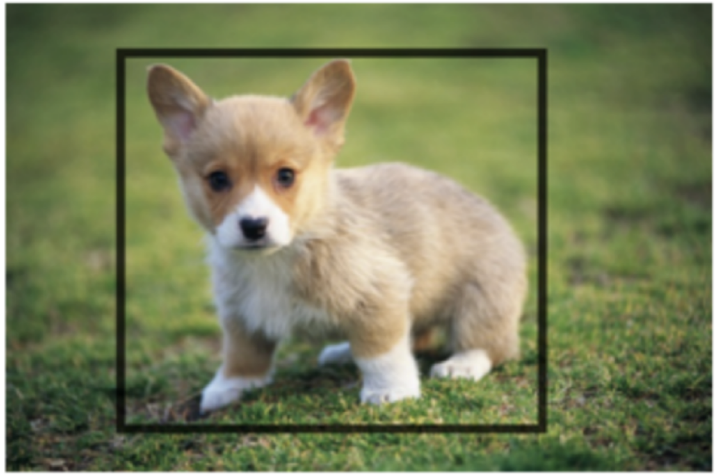


0 to 256
1 channel for grayscale
3 channels for RGB

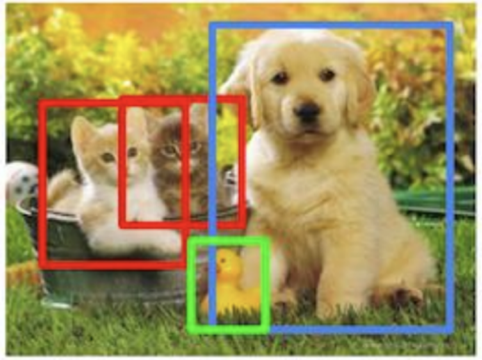
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Some Types of Computer Vision Tasks

Classification



Localization



CAT, DOG, DUCK

Detection (could be multiple classes)

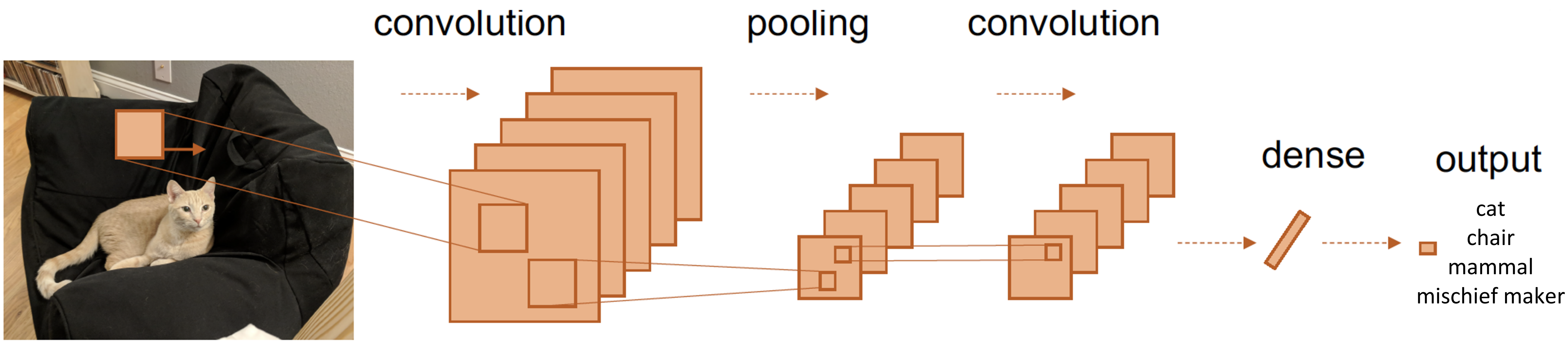
Semantic Segmentation

Computer Vision - can we use a fully-connected neural network?

100 x 100 pixel image

→ 10,000 weights for one neuron!

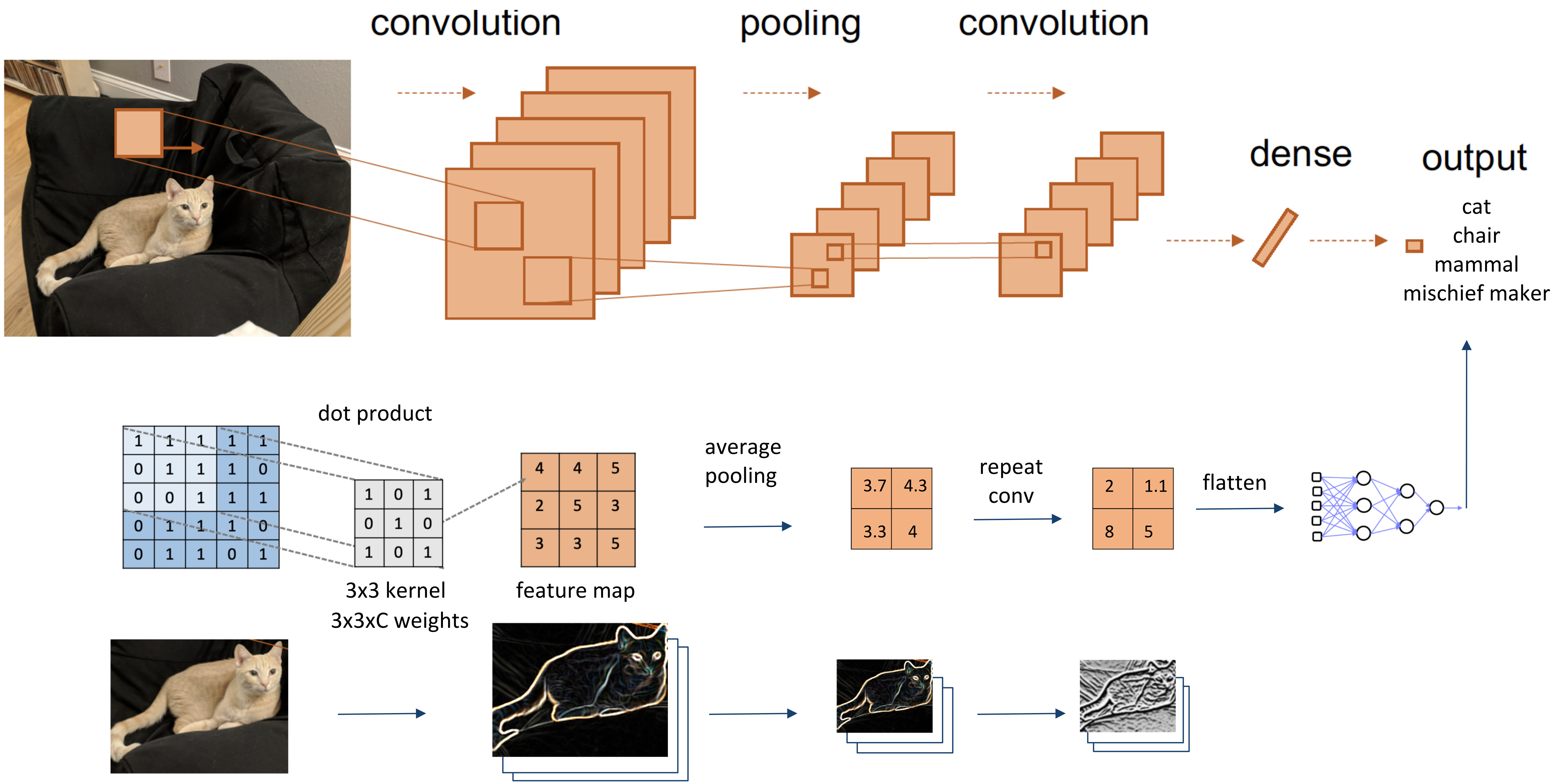
Convolutional Neural Networks (CNNs)



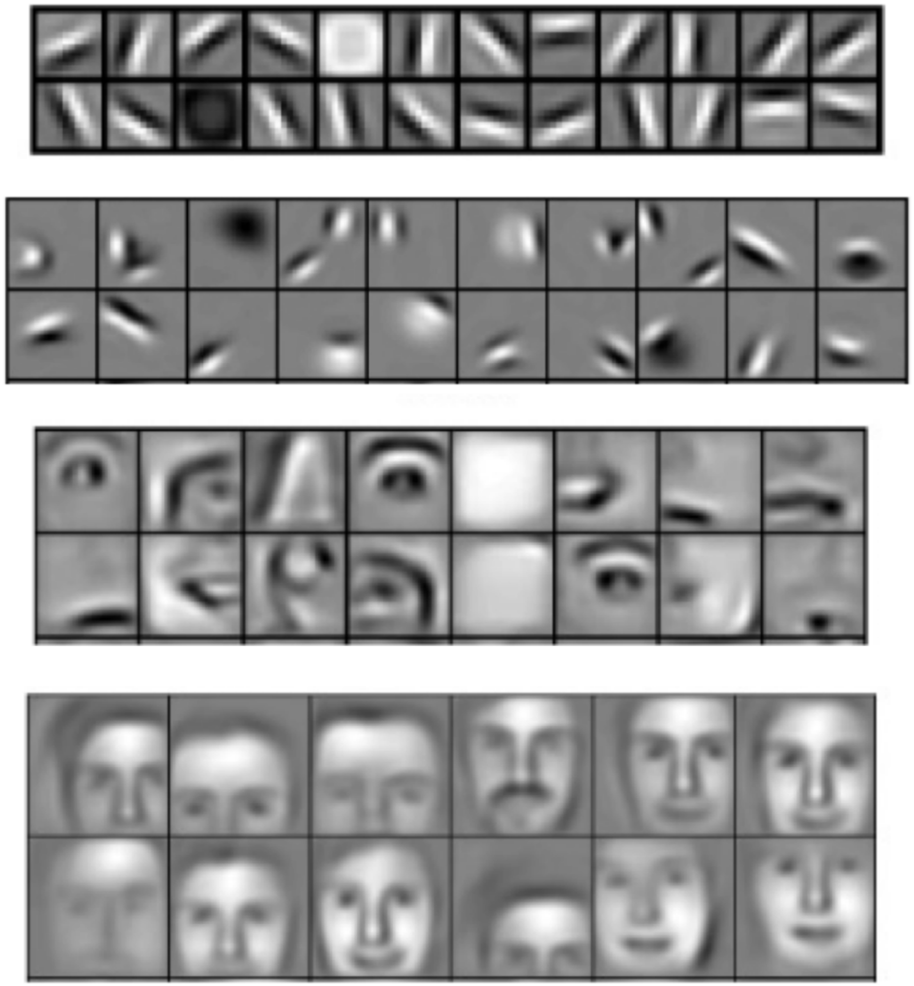
- **Learned filters convolved across image and subsequent feature maps**
- **Learn *local* features that are translation invariant**

- Inspired by structure of visual cortex
- First major use on MNIST data set (ID handwritten digits), 1998
    - http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Convolutional Neural Networks (CNNs)



Convolutional Neural Networks (CNNs)



Hierarchical Representations

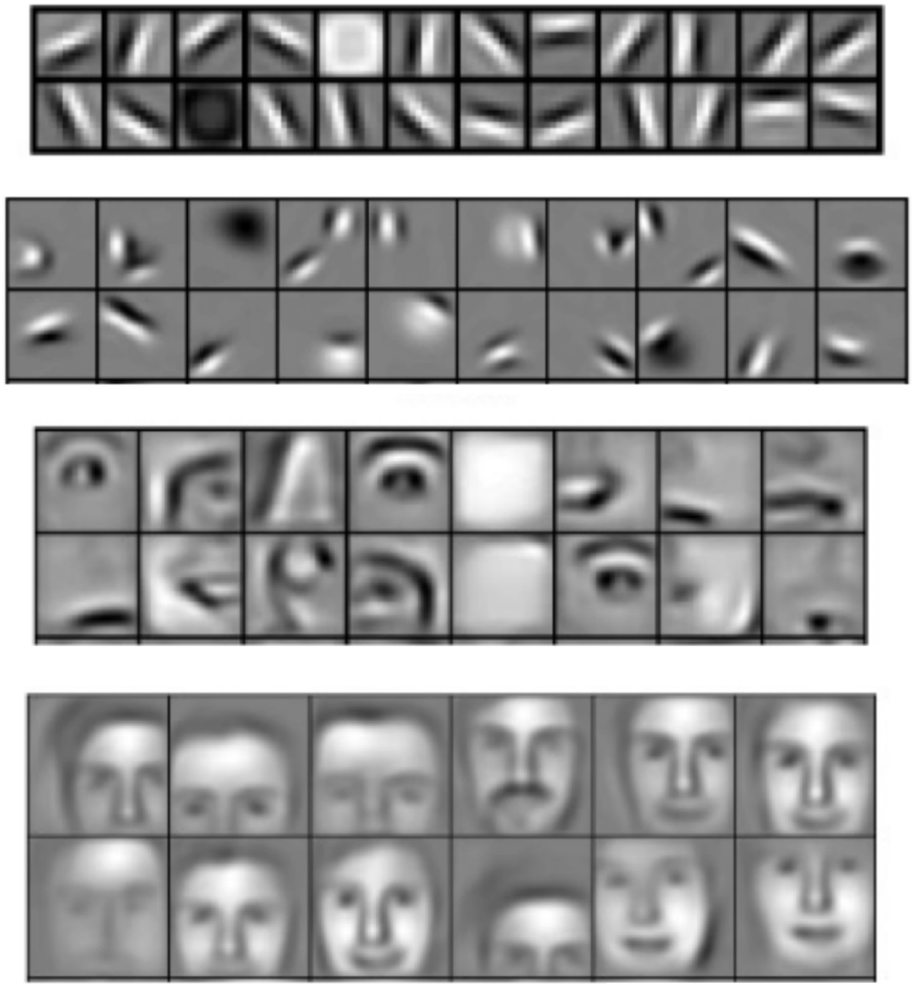
Earlier layers

↓

Later layers

http://web.eecs.umich.edu/~honglak/cacm2011-researchHighlights-convDBN.pdf

Convolutional Neural Networks (CNNs)



http://web.eecs.umich.edu/~honglak/cacm2011-resear

Hierarchical Representations

Earlier layers

Later layers

Natural question:

Can we re-use the more primitive representations?

Transfer Learning

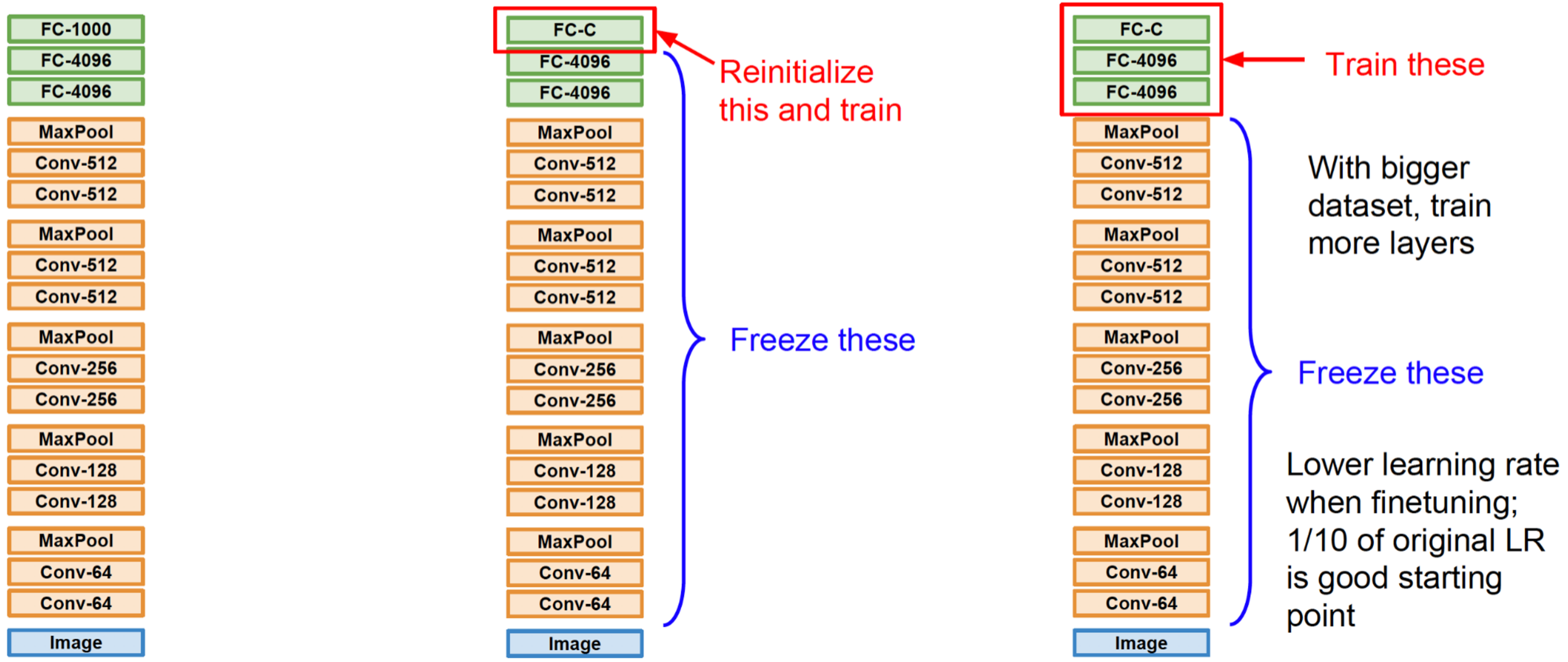
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

1. Train on Imagenet

2. Small Dataset (C classes)

3. Bigger dataset



http://cs231n.stanford.edu/

Transfer Learning

Requires less data by not having to learn primitive features from scratch

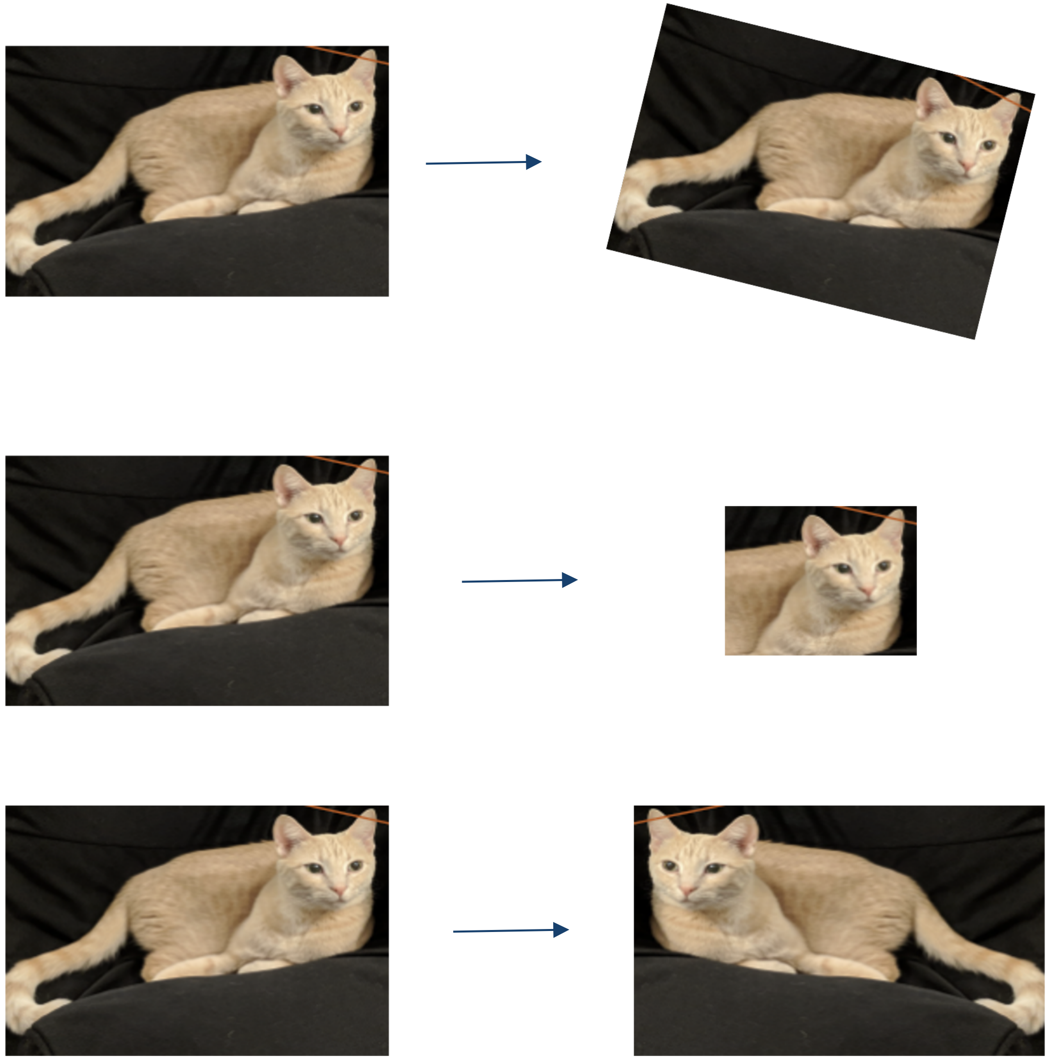
Various “Model Zoos” of pretrained models:

- Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
- TensorFlow: https://github.com/tensorflow/models
- PyTorch: https://github.com/pytorch/vision

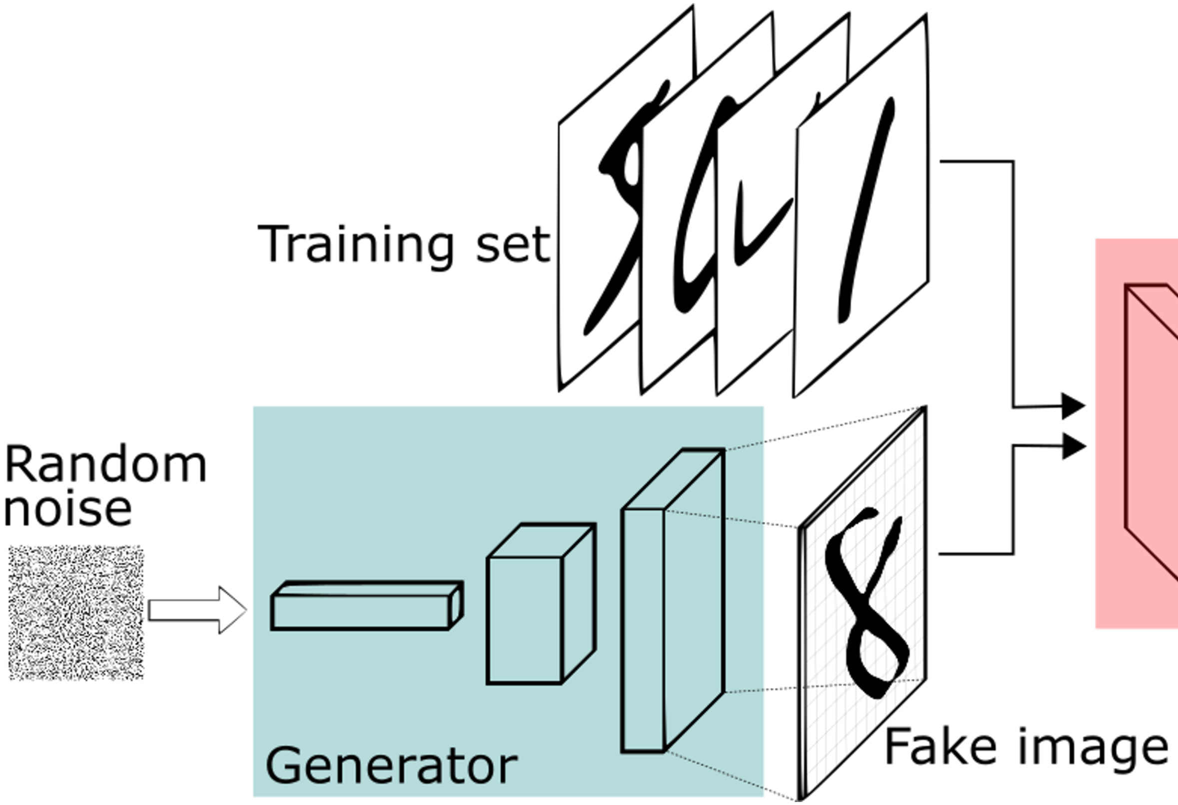
Data Augmentation

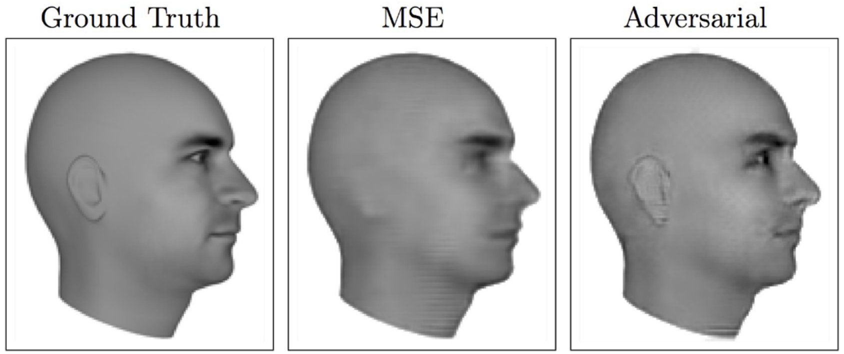
Common practice to artificially increase data set size:

- cropping
- rotations
- noise
- mirroring
- shearing
- etc



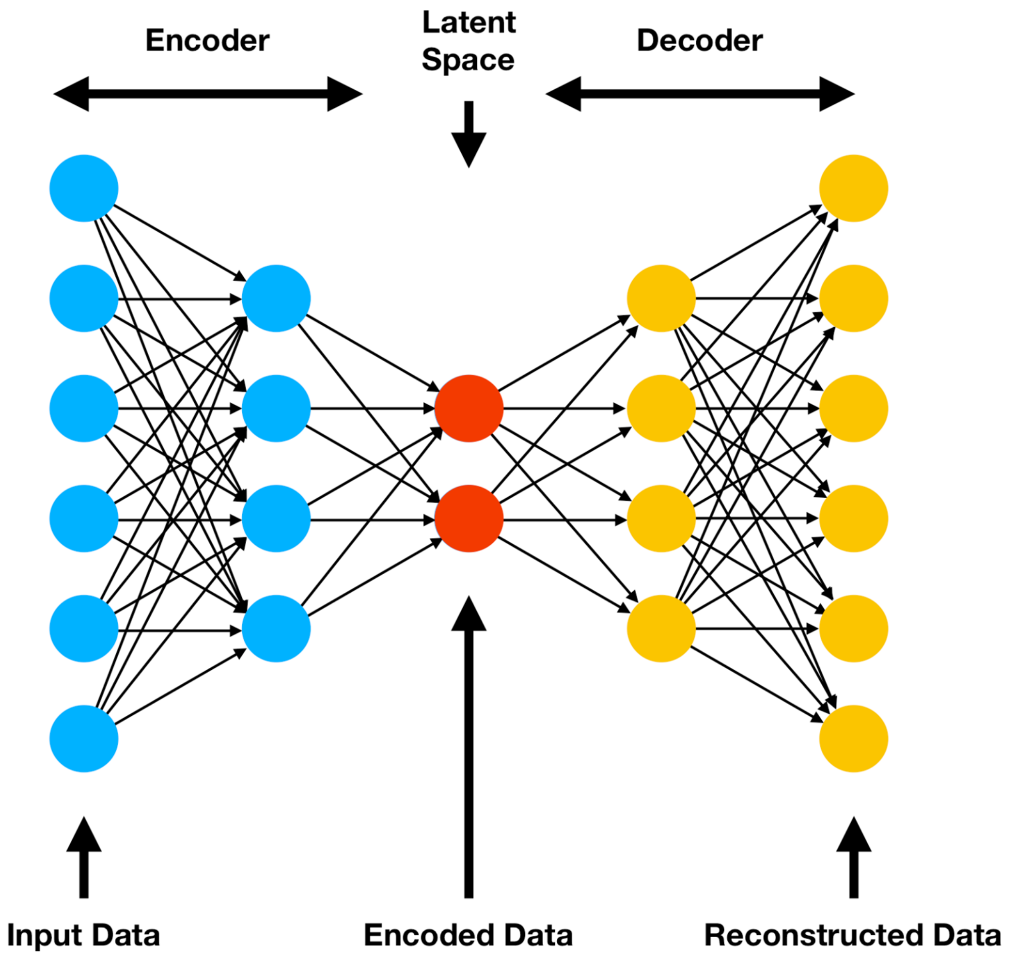
Generative Adversarial Networks (GANs)





See NeurIPS tutorial (2016): https://arxiv.org/pdf/1701.00160.pdf

Auto-Encoders

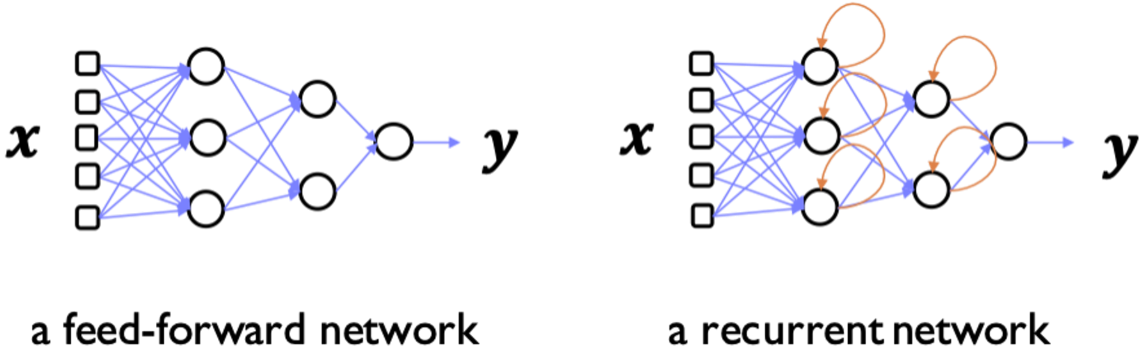


Learn compressed representation (latent space) of the input

Can also have more general "encoder-decoder" style bottleneck architectures that are not auto-encoders

https://www.compthree.com/blog/autoencoder/

Recurrent Neural Networks (RNNs)



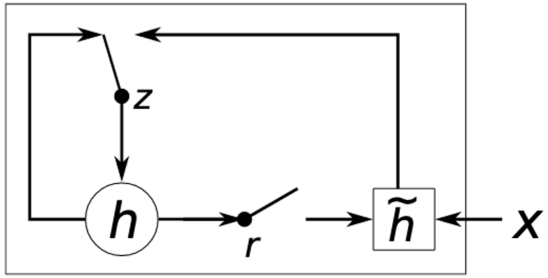
Recurrent connections: previous inputs affect next output

→ can capture series data

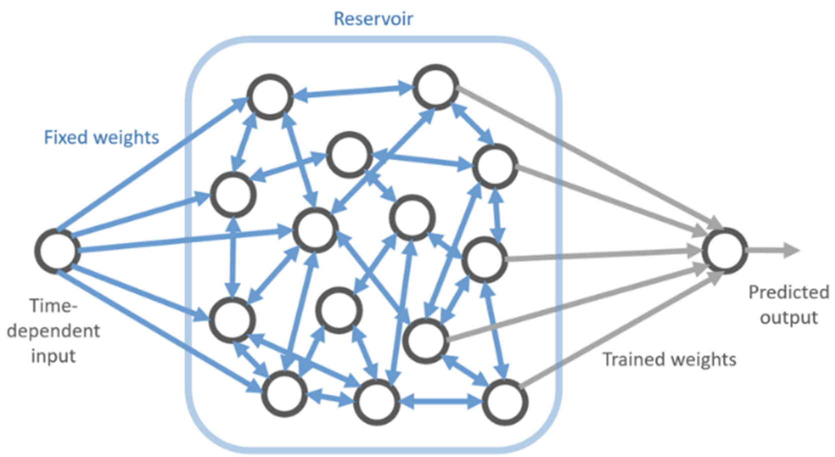
Some use special memory gates to avoid vanishing/exploding gradients:

Long Short-Term Memory (LSTM), Hochreiter et al., 1997
https://www.bioinf.jku.at/publications/older/2604.pdf

Gated Recurrent Unit (GRU), Cho et al., 2014
https://arxiv.org/abs/1406.1078



Reservoir Computing



https://arxiv.org/pdf/1907.00657.pdf

Historical reading:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
https://ieeexplore.ieee.org/document/279181

Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013 http://proceedings.mlr.press/v28/pascanu13.html

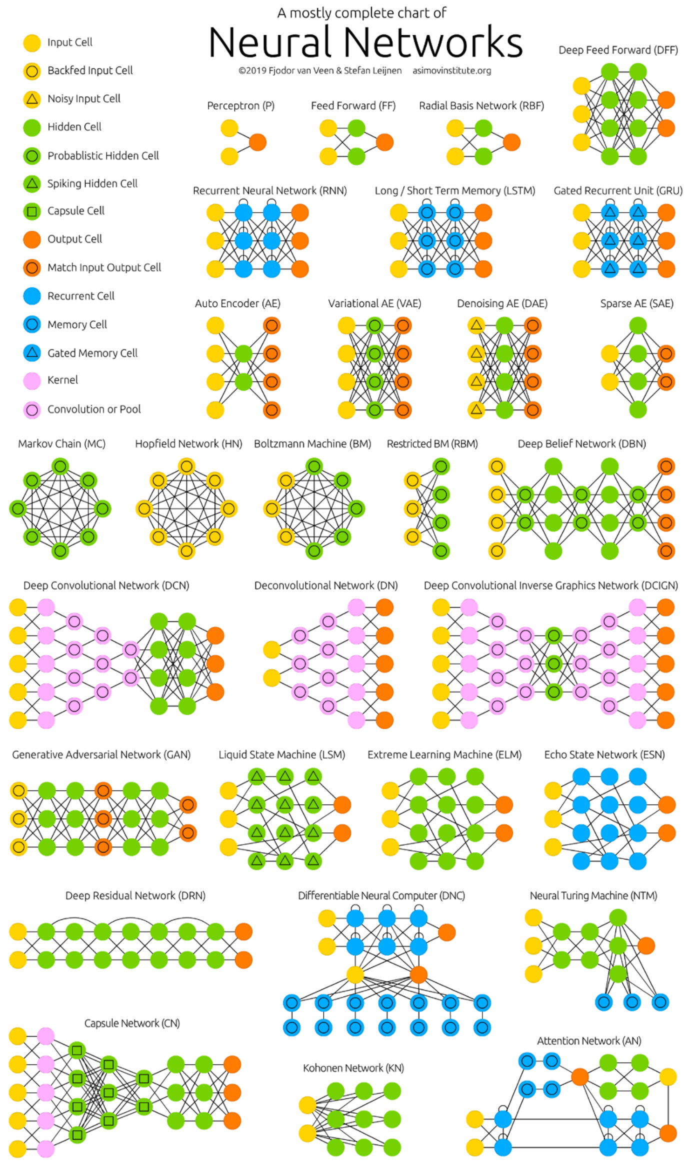
Sutskever et al, Dissertation, 2013,
https://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf

And beyond...

So many specialized neural network architectures!

The “Neural Network Zoo” website can be a good starting point for familiarization

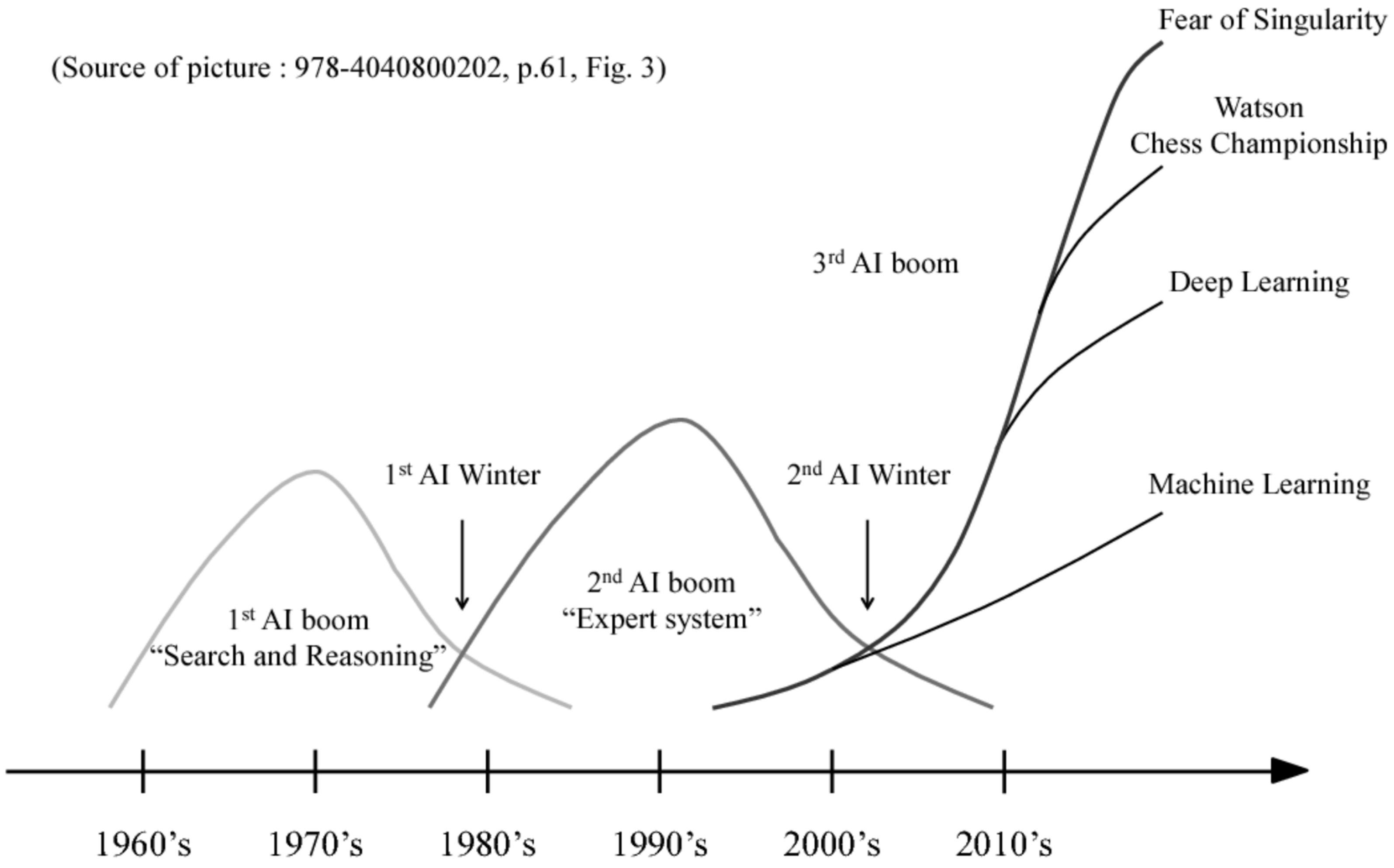
https://www.asimovinstitute.org/neural-network-zoo/



History of Neural Networks and “AI winters”

- 1950s - 1960s: reasoning, search etc
- 1970s: AI winter
- 1980s: “connectionism” i.e. neural networks, knowledge representation
- 1990s: AI winter
- 1997: Deep Blue beats Gary Kasparov in chess
- 2006: Deep learning breakthroughs at University of Toronto
- 2011: IBM Watson wins Jeapordy
- 2015: Deep learning on GPUs
- 2016: Alpha-Go deep learning software beats best players

(Source of picture : 978-4040800202, p.61, Fig. 3)



What’s different now?

Increased computational capability enables more complicated NN architectures and faster training + larger data sets

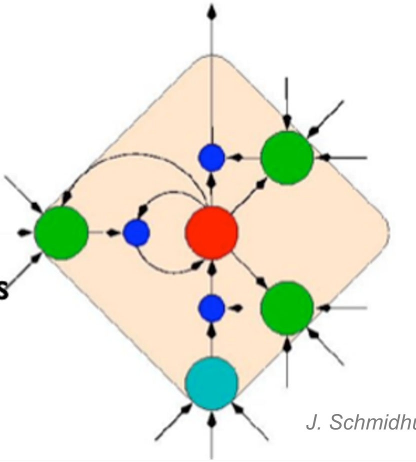
GPUs

NVIDA

Argonne National Lab

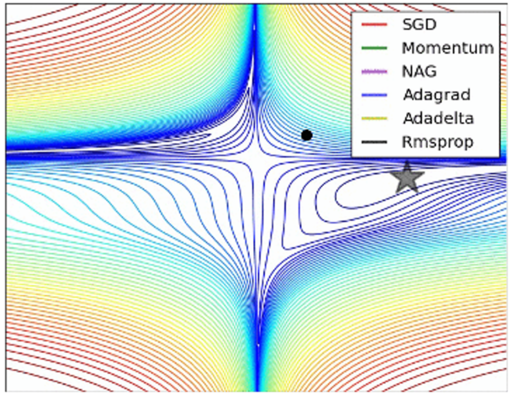
Accessibility of HPC clusters

Neural network architectures and training paradigms,
such as long short term memory (LSTM) networks, generative adversarial networks (GANs)



J. Schmidhuber

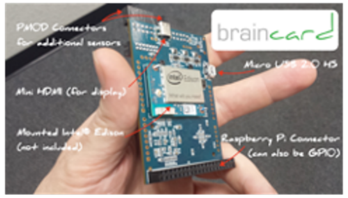
Can **easily share** large data sets, code, and computing setups
(e.g. via cloud computing services)



A. Radford

Better **theoretical understanding** of NNs and improved **optimization methods**

specialized hardware: neuromorphic chips, TPUs



Applications have driven a lot of advancement (both algorithmic and practical/heuristic)

Google

Applications

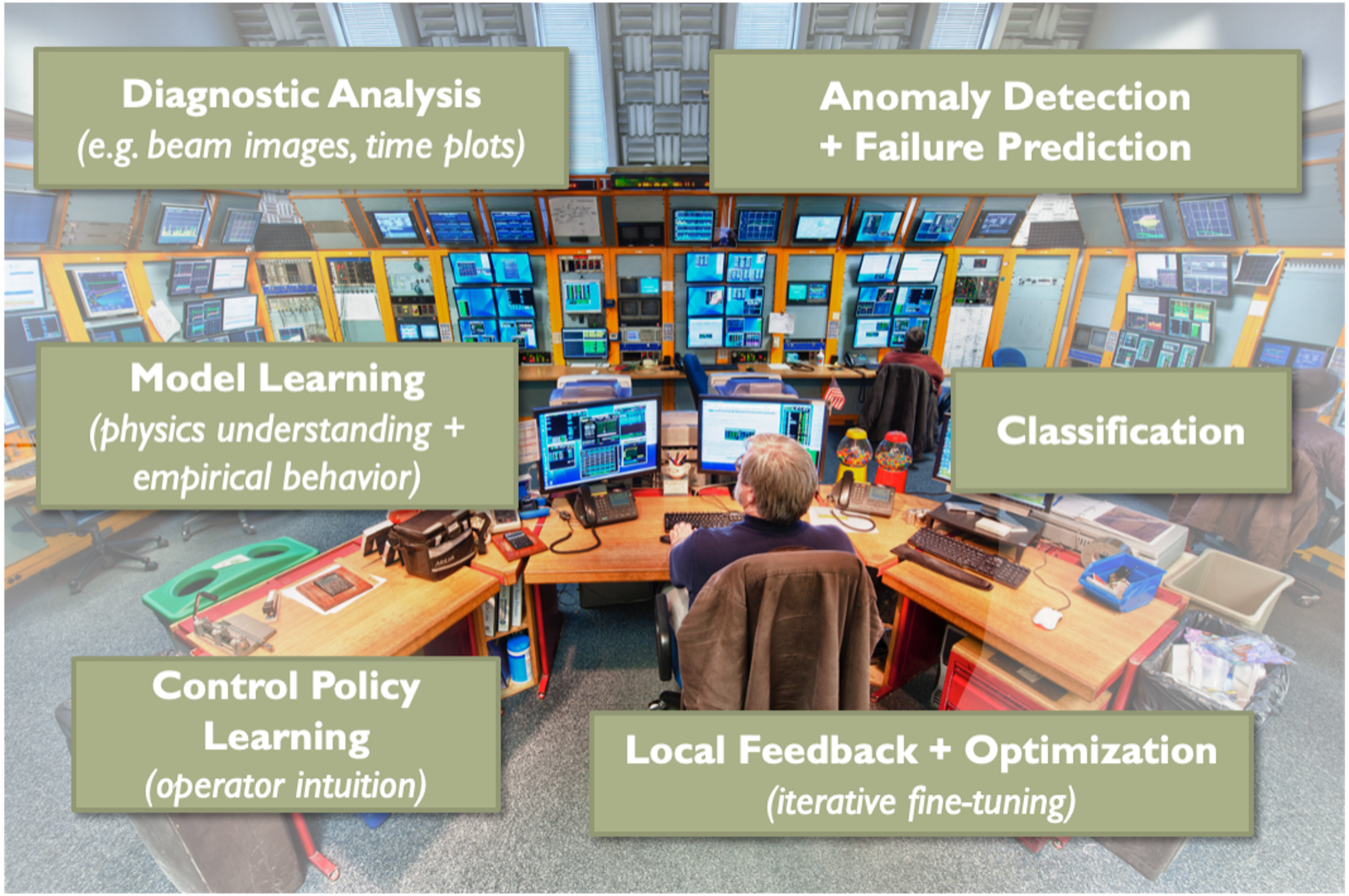
Next: a few examples of how neural networks can be used in particle accelerators

Applications of AI/ML in Accelerators

Fermilab Control Room
(photo: Reidar Hahn, FNAL)

Take inspiration from accelerator operators?

Applications of AI/ML in Accelerators



Fermilab Control Room
(photo: Reidar Hahn, FNAL)

Take inspiration from accelerator operators?

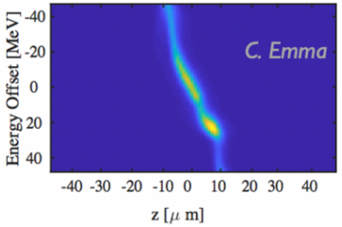
Neural networks can be appealing for some of these individual tasks

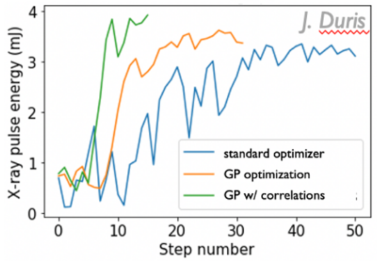
Role of ML

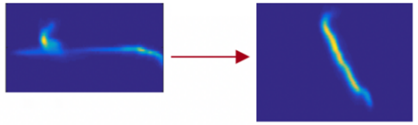
automated control
+ optimization

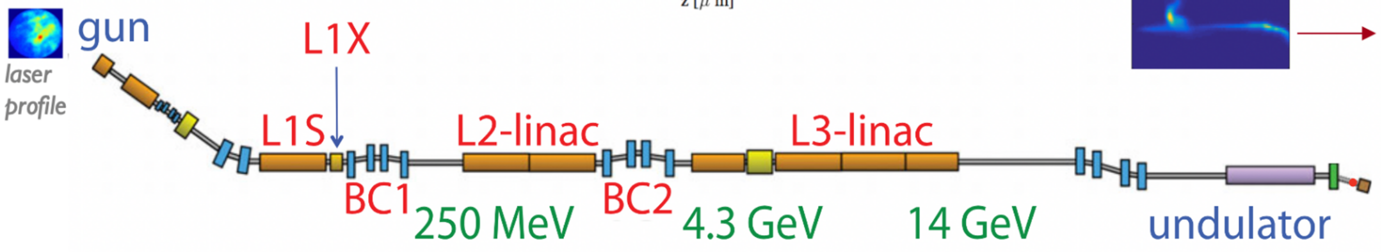
advanced diagnostics
(reconstruct / analyze beam)

anomaly detection
failure prediction

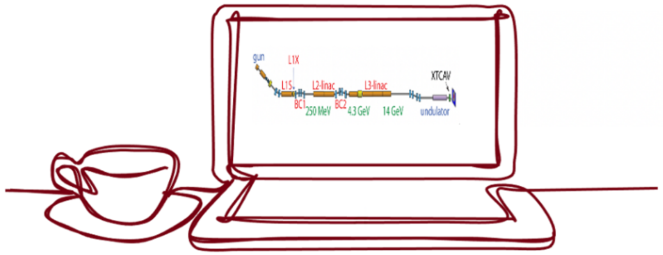








incorporate
physics
information



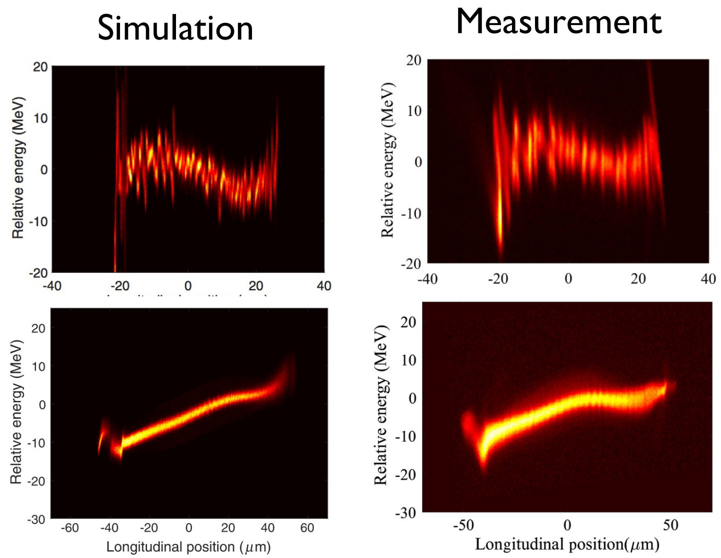
extract unexpected
relationships
(feed into control / design)

digital twins + online modeling
(fast sims, autodiff sims, model calibration)

+ need UQ for all

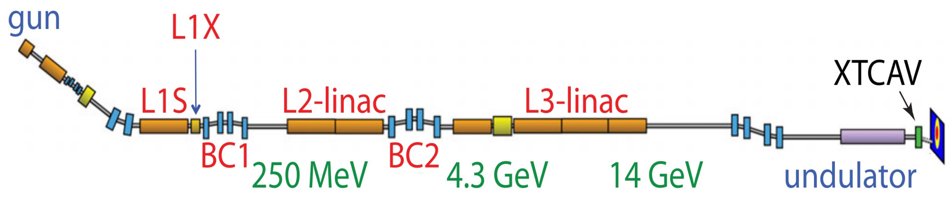
Speeding Up Simulations

Particle accelerator simulations that include nonlinear + collective effects are powerful tools…



J. Qiang, et al., PRSTAB30, 054402, 2017

"10 hours on thousands of cores at the NERSC"



… but they are computationally expensive

Impedes offline start-to-end optimization and control prototyping

Prohibits use as an online model (e.g. diagnostic / control applications)

Difficult to comprehensively calibrate to machine

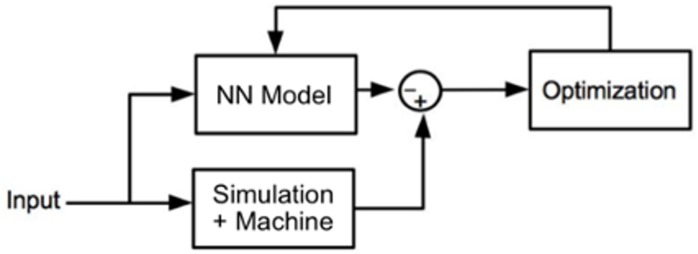
→ very unlikely to achieve sufficient speedup with HPC resources and fundamental improvement in simulation algorithms alone

Speeding Up Simulations

Complementary approach: ML model

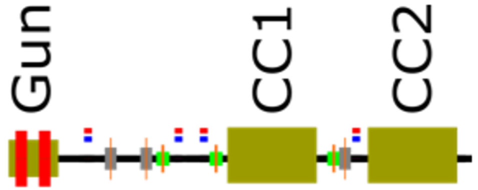
Once trained, neural networks can execute quickly

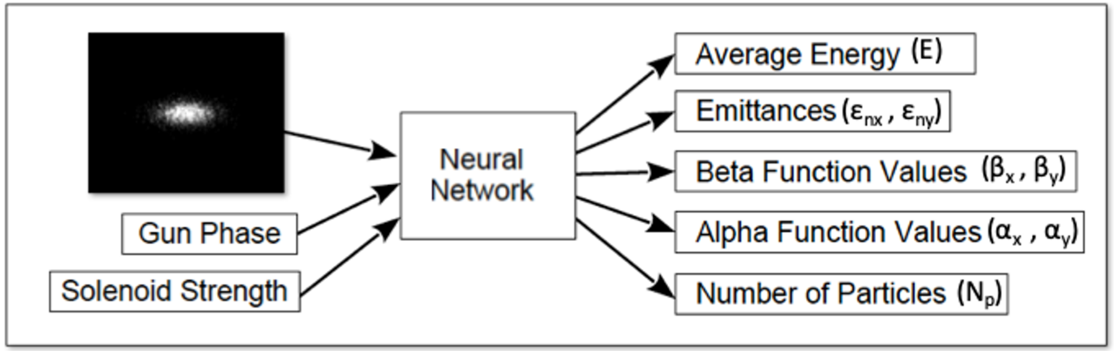
Train on sparse sample from high-fidelity simulations
+
Train on measured data



Initial examples from FAST injector at Fermilab:

PARMELA with 2-D space charge: ~ 20 minutes
Neural network: ~ a millisecond



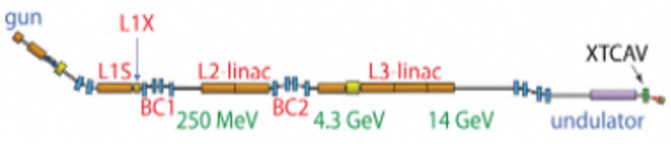


All mean absolute errors between 0.9% and 3.1% of the parameter ranges

A. L. Edelen, J.P. Edelen. D. Edstrom, et al. NAPAC16, TUPOA51
A.L. Edelen, J.P. Edelen, et al., FEL '17 for an example with control

Speeding Up Simulations

LCLS Main Linac



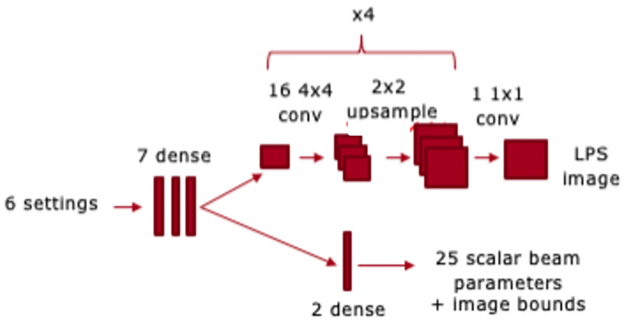
Wide scan of of controllable settings in simulation to generate dataset of beam output prior to the undulator

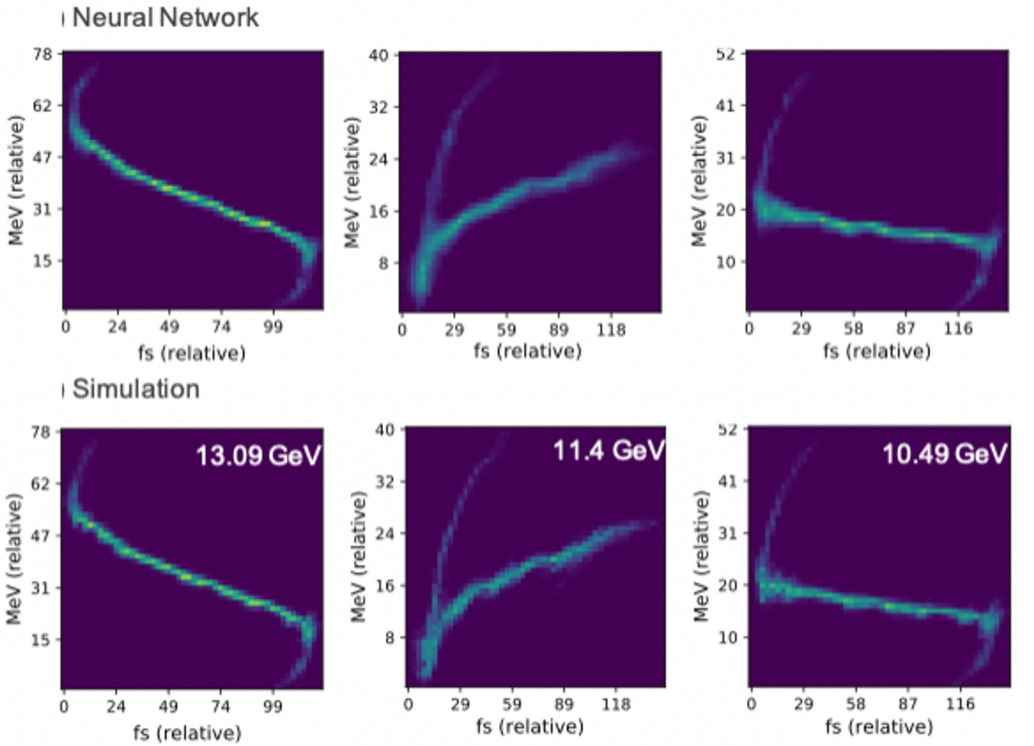
Trained a NN to predict
- **25 scalar beam outputs** (beam size, emittance, energy spread, etc)
- **2D longitudinal phase space** (LPS) projection

Good agreement with simulation and 10^6x faster execution

Scan of 6 settings in simulation

Variable	Min	Max	Nominal	Unit
L1 Phase	-40	-20	-25.1	deg
L2 Phase	-50	0	-41.4	deg
L3 Phase	-10	10	0	deg
L1 Voltage	50	110	100	percent
L2 Voltage	50	110	100	percent
L3 Voltage	50	110	100	percent

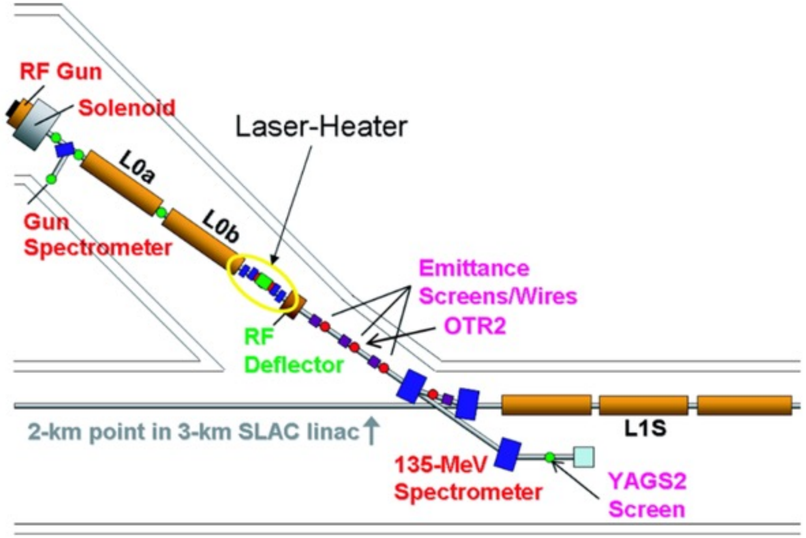




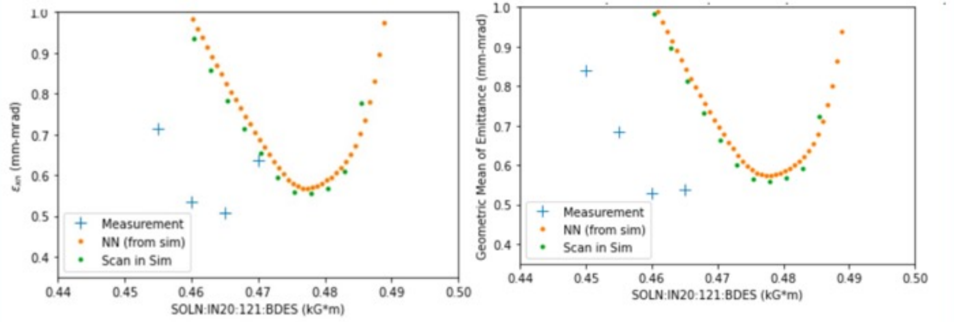
Edelen et al, NeurIPS 2019
https://ml4physicalsciences.github.io/2019/files/NeurIPS_ML4PS_2019_90.pdf

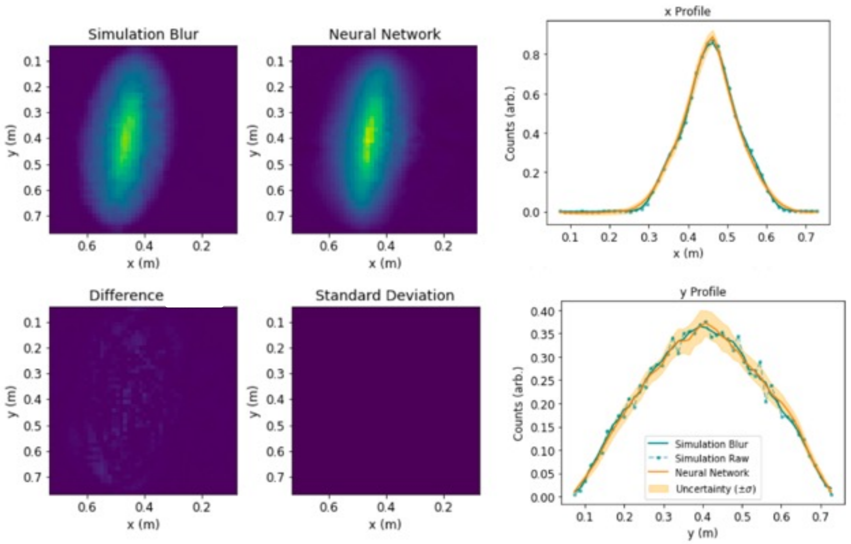
LCLS Injector Surrogate Model

- **Many versions (predict phase space, evolution along z etc); including one with scalar outputs of interest at OTR2**
  - **Inputs:** *laser length + spot size, L0A/B phases, Solenoid, SQ quad, CQ quad, 6matching quads*
  - **Outputs:** *emittances, bunch length, spot sizes, covariances (for Twiss calc), energy*
- Neural network trained on IMPACT-T sims
- Set up to take machine inputs in PV units
- Focused on interpolation to sim vs. exact match to measurements
- Using in tuning algorithm + code testing

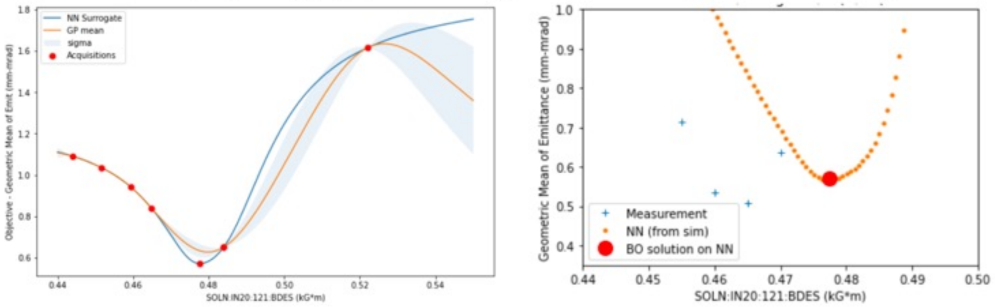


IMPACT-T and SM trained on it deviate from measurements, but similar qualitatively





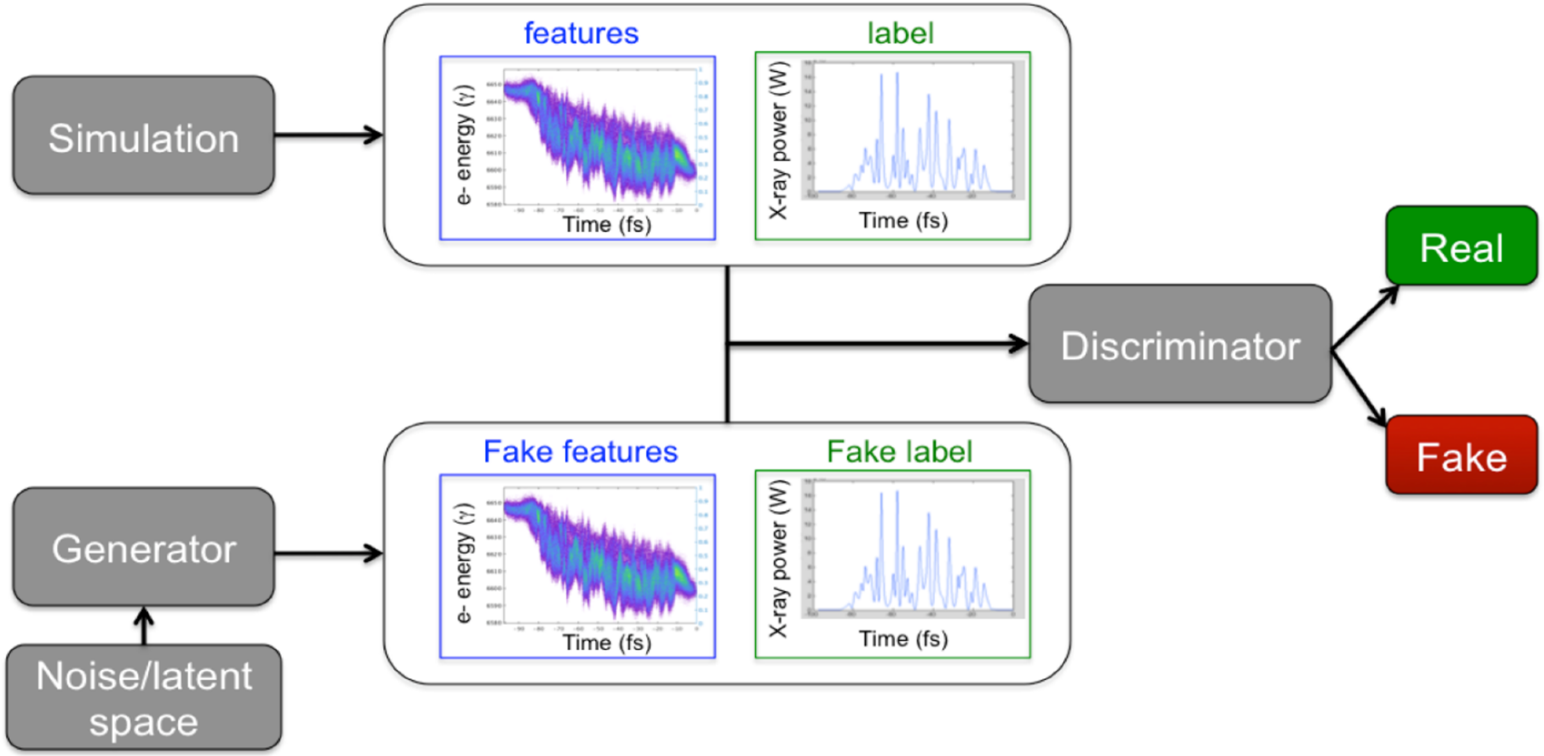
Example prototyping optimization algorithms with SM (GP-BO in this case)

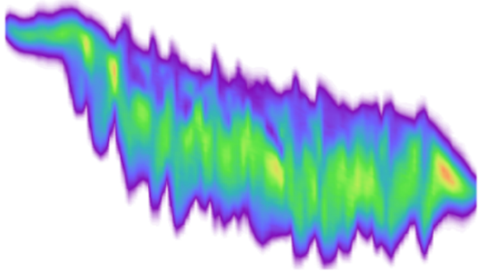


GAN for FEL Pulse Prediction

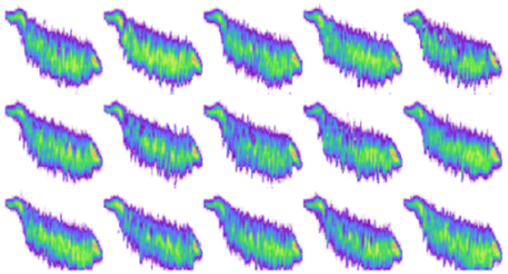
- The FEL process has many stochastic effects that show up as shot-to-shot variation in the output electron and photon beam
- Simulations are slow
- Photon science users would like estimates of the statistical output distributions they can expect (e.g. help with prep for analysis procedures)

→ Can use a GAN to produce examples of FEL longitudinal phase space output that is statistically representative of the real process





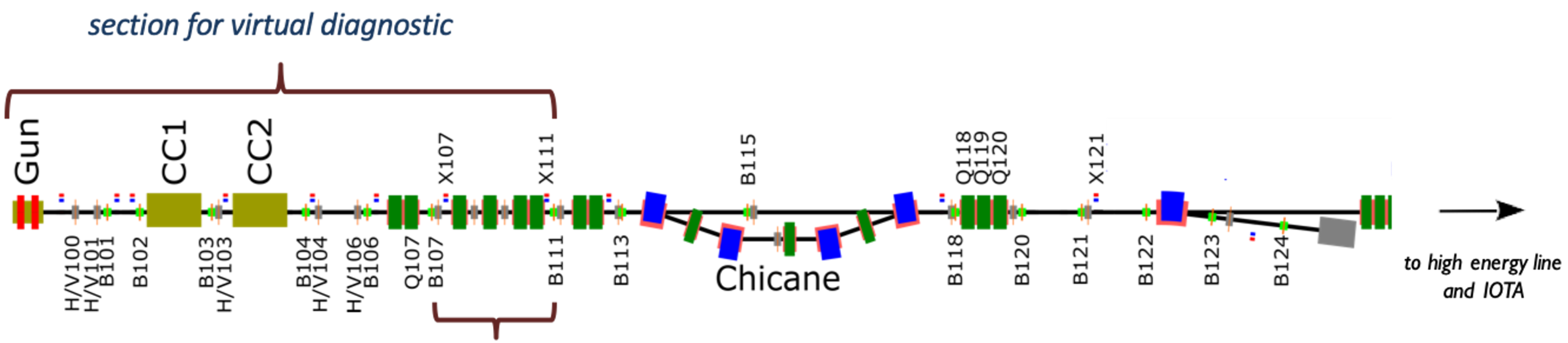
Genesis:
~1000 cpu-sec

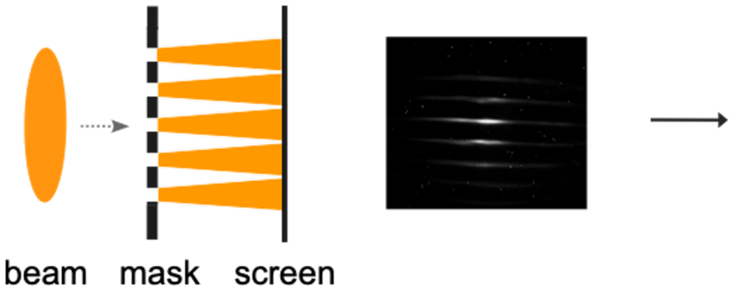


GAN (neural net):
~0.001 gpu-sec

X. Ren

Some diagnostics are destructive to the beam





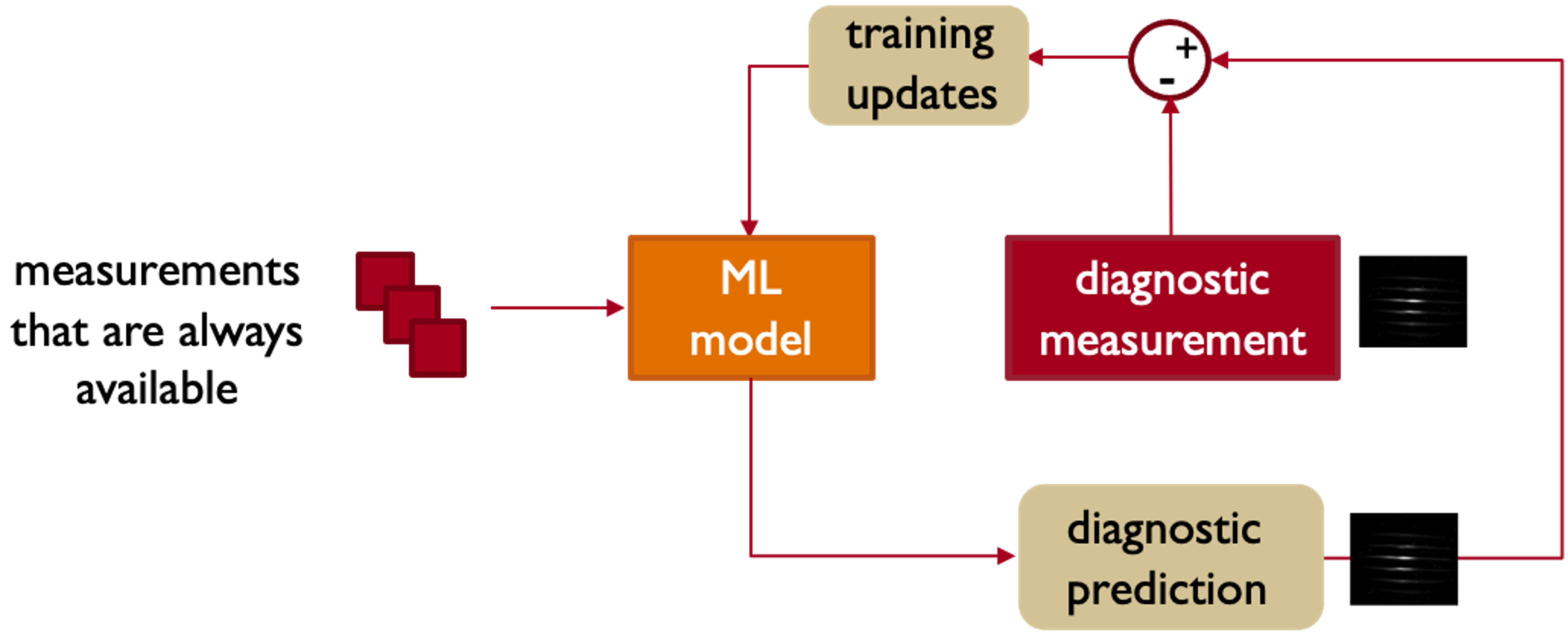
fit to obtain subset of phase space parameters $\langle y^2 \rangle$ $\langle yy' \rangle$ $\langle y'^2 \rangle$

At FAST (Fermilab) multi-slit emittance measurements takes 10-15 seconds in each plane

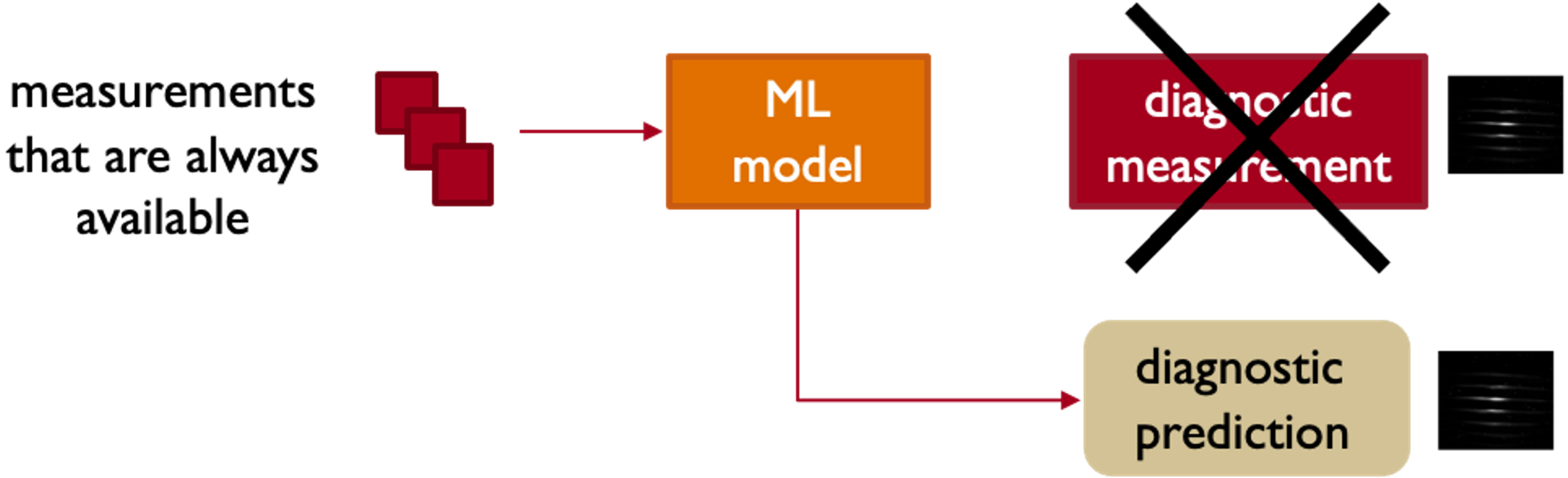
→ can we get a non-destructive prediction of what this diagnostic would show?

A. L. Edelen, J.P. Edelen, D. Edstrom, et al. NAPAC16, TUPOA51
A. L. Edelen, J.P. Edelen, D. Edstrom, et al. 1st ICFA ML Workshop, Feb. 2018

Virtual Diagnostics

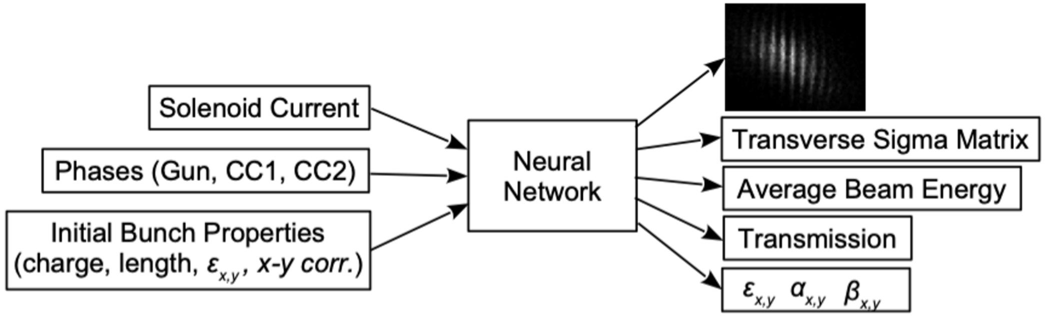


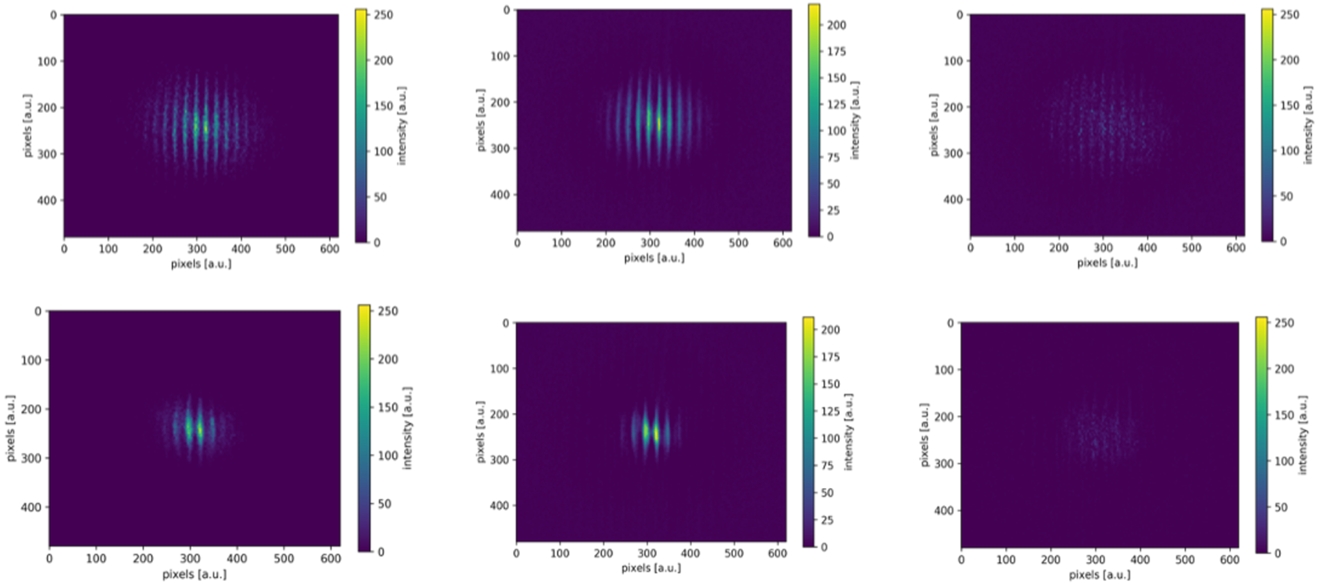
Virtual Diagnostics



Still can have diagnostic prediction for user analysis and system control!

Virtual Diagnostics





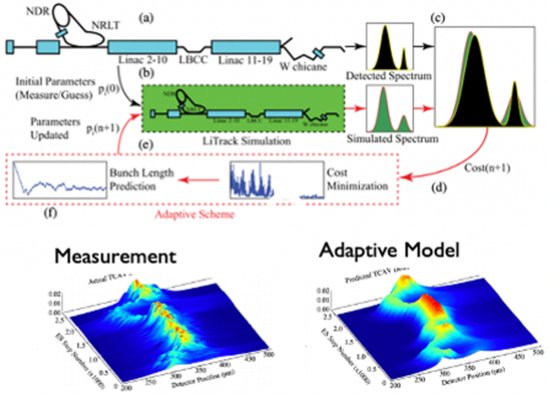
Simulated NN Predictions Difference

A. L. Edelen, J.P. Edelen, D. Edstrom, et al. NAPAC16, TUPOA51
A. L. Edelen, J.P. Edelen, D. Edstrom, et al. 1st ICFA ML Workshop, Feb. 2018

Virtual Diagnostics

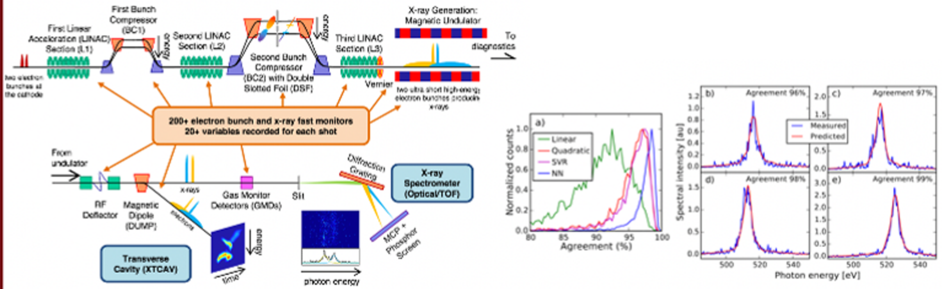
Examples for longitudinal phase space:
mix of adaptively calibrated physics models and ML-based prediction…

Adaptively tune a simple physics model



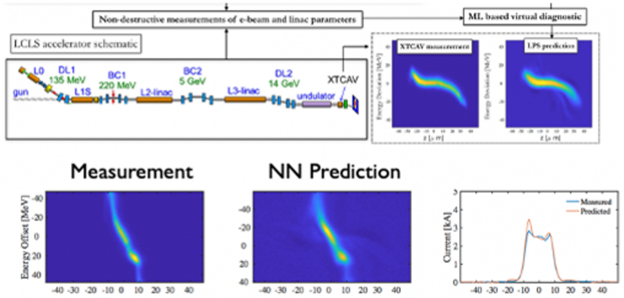
A. Scheinker, S.Gessner, PRAB 18, 102801 (2015)

Fill in shots: use archive data to learn correlation between fast and slow diagnostics



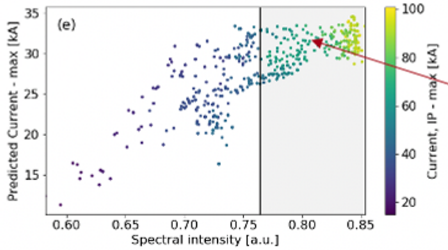
A. Sanchez-Gonzalez, et al., Nature Comms (2017)

Predict with a trained neural network



C. Emma, A. Edelen, et al., PRAB21, 112802 (2018)

Can use spectral information as input to predict beyond typical diagnostic resolution



Shots are beyond the TCAV resolution

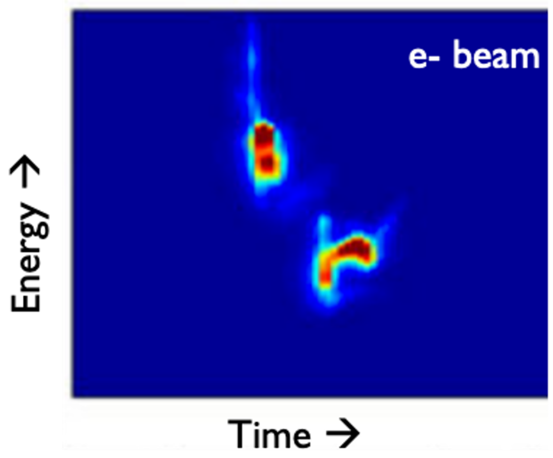
A. Hanuka, et al. 2009.12835 [accepted to Nature Scientific Reports]

Signal Analysis

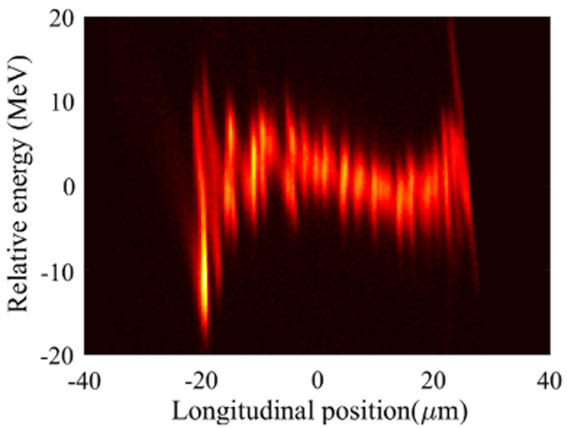
Signals used in feedback control and experimental analysis can be complicated *(e.g. beam images, time series)*

→ *Can use ML to extract more useful information from these signals*

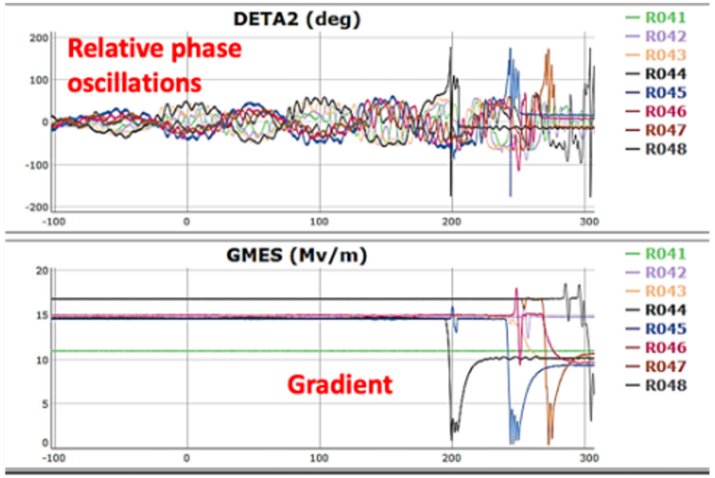
→ *NNs are particularly useful for this*



A. Marinelli, et al., Nat. Commun. **6**, 6369 (2015)

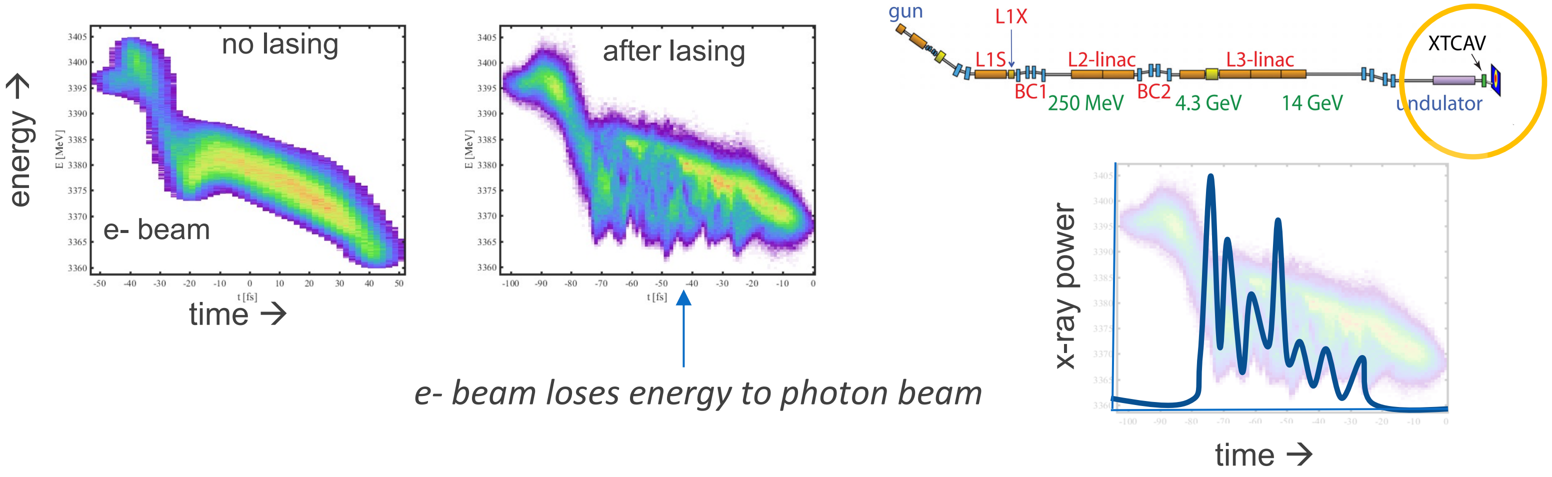


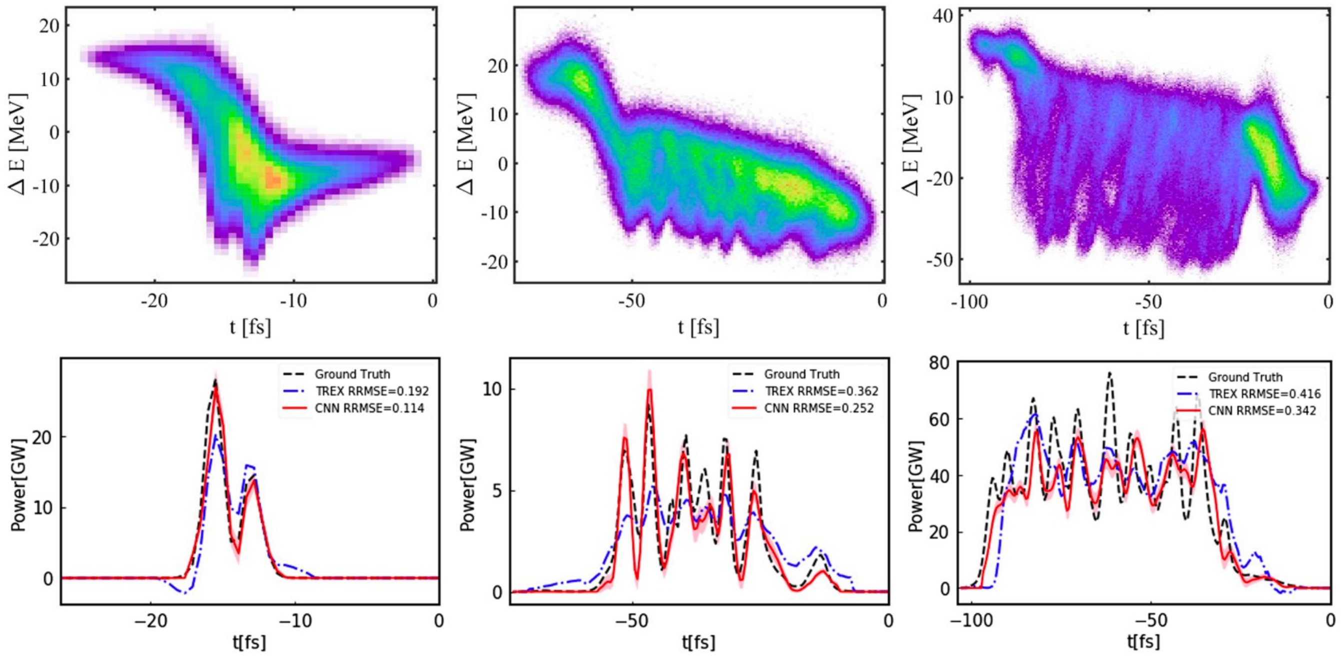
J. Qiang, et al., PRSTAB30, 054402, 2017



A. Solopova, IPAC'19

CNN for Image Analysis





X. Ren, A. Edelen, D. Ratner, et al., PRAB 2020

Free Electron Laser: e- beam loses energy to photon beam

e- beam image before/after lasing process provides critical information to users about photon beam

- *relies on slow, iterative reconstruction algorithm to get X-ray power profile*
- *iterative method doesn't work well for all regimes (e.g. in saturation)*

Instead: use convolutional neural net to get accurate predictions quickly

Classifying Cavity RF Trips

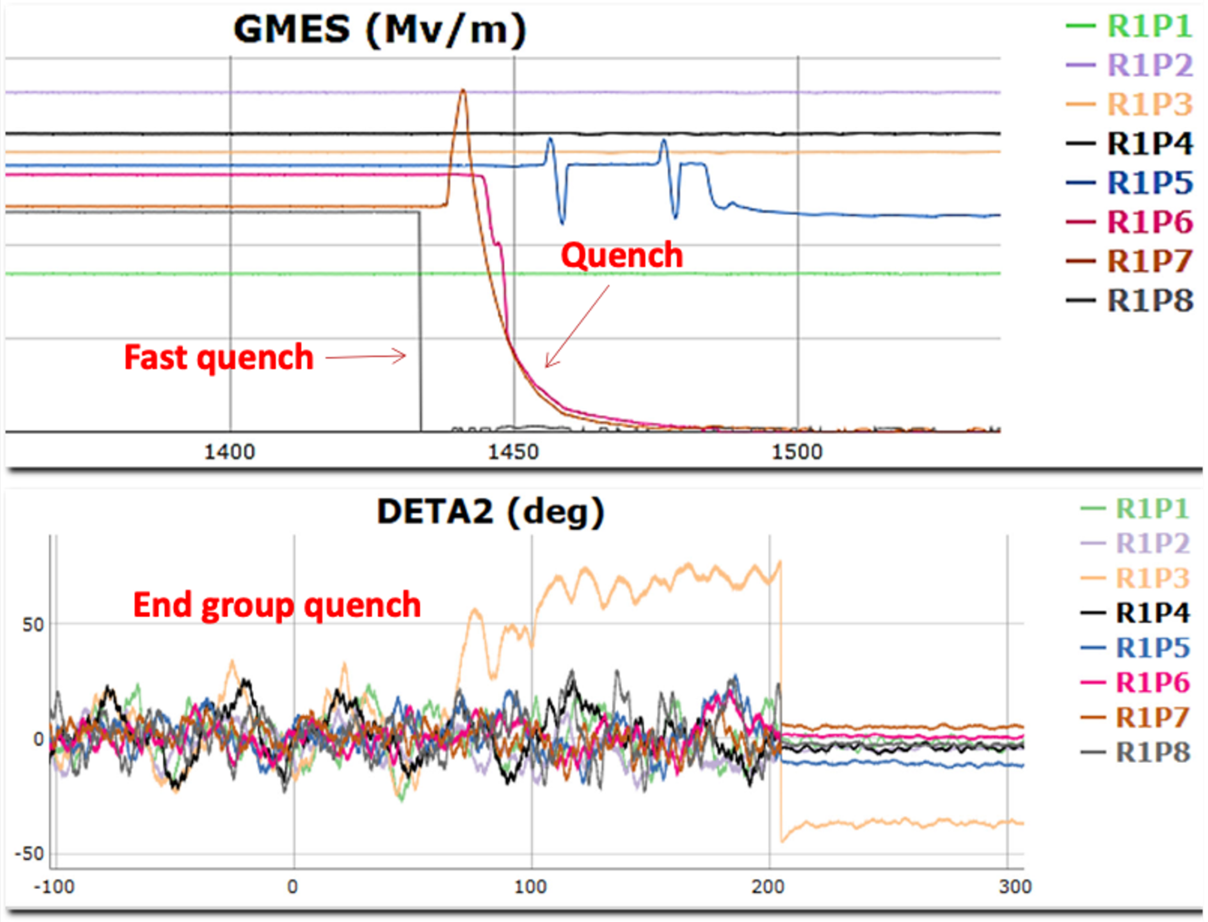
Cavities can trip in a variety of ways

(fast quench, thermal quench, end group quench, microphonics)

Experts identify type of trip from RF waveform data

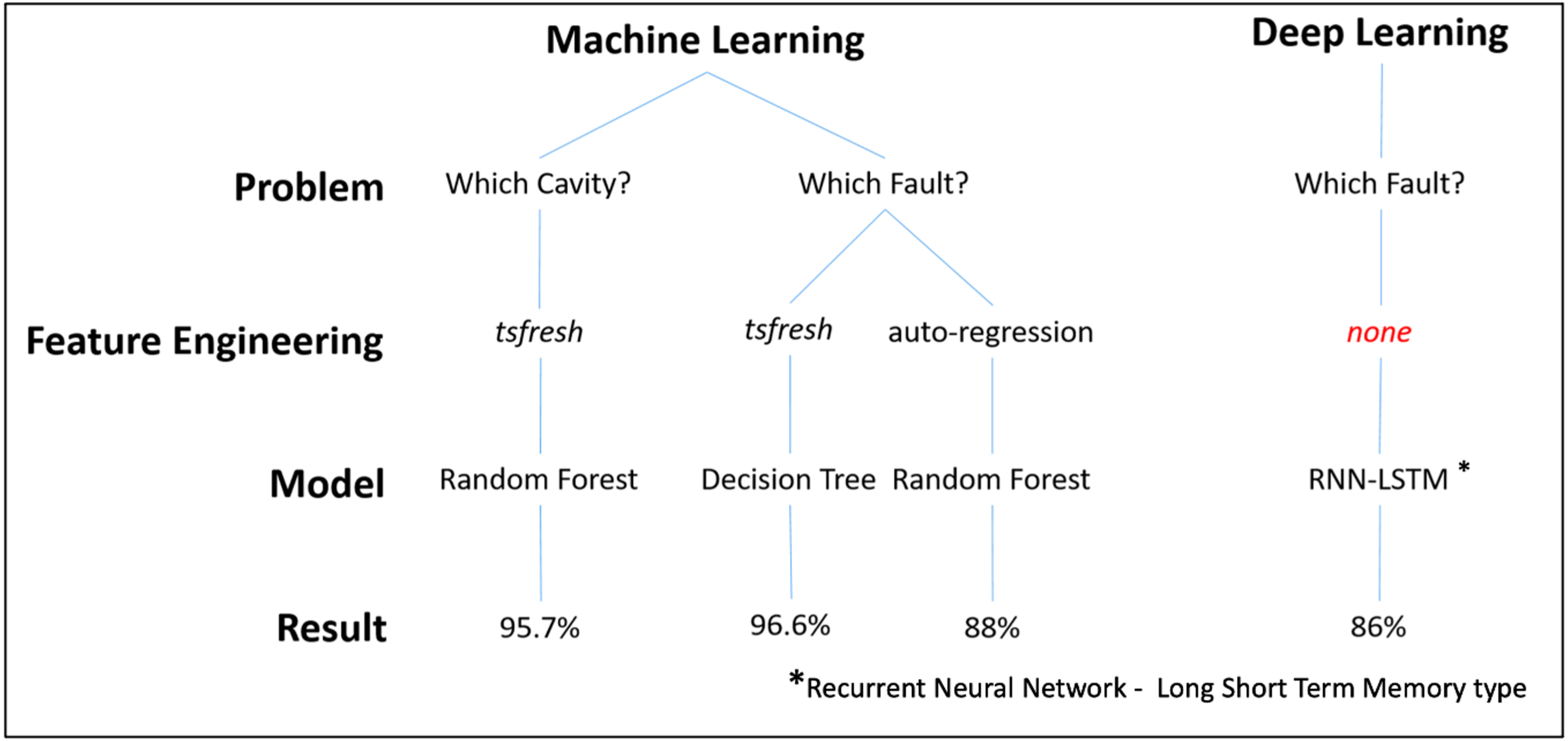
Instead, use automatic classification:

- *Enables more systematic study of trips and effectiveness of recovery strategies*
- *Quickly informs a proper response in the control room*



A. Solopova, et al., IPAC'19

Classifying Cavity RF Trips

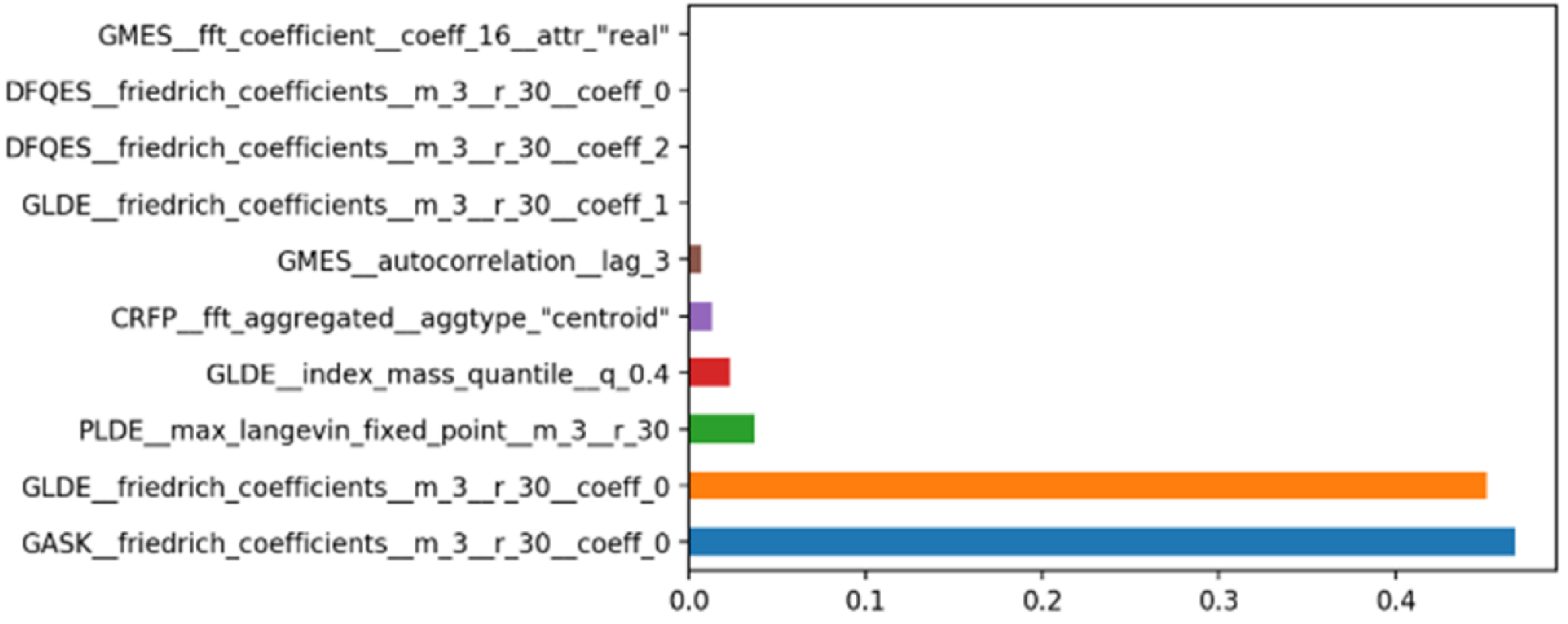


A. Solopova, et al., IPAC'19

Tradeoff between feature engineering, interpretability, and amount of data

Classifying Cavity RF Trips

Example: classification using decision tree also gives feature importance

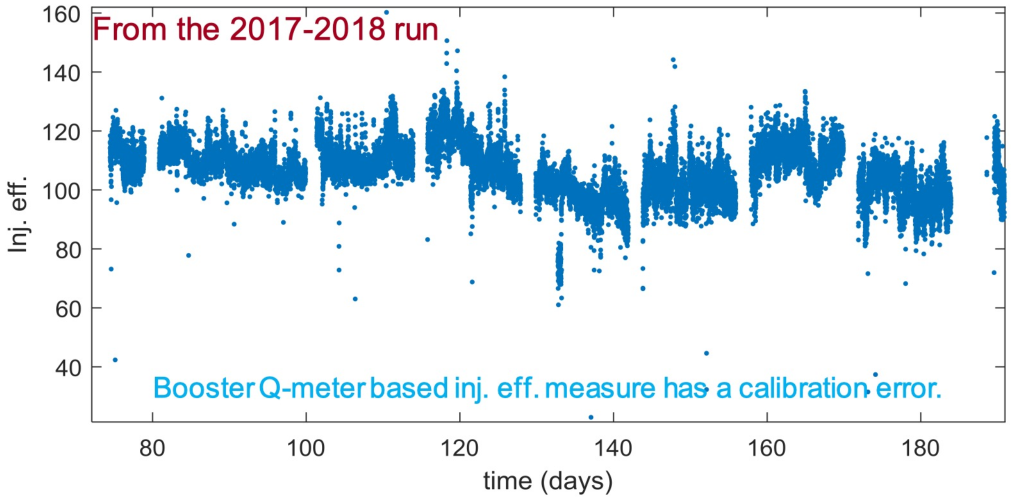


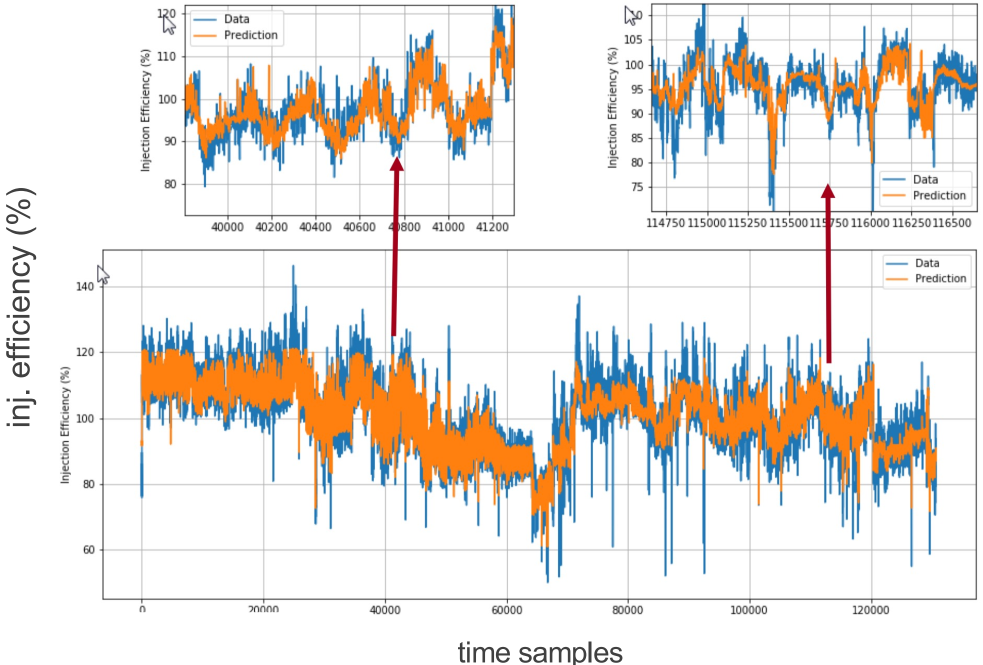
A. Solopova, et al., IPAC'19

Improve system understanding: learn about machine sensitivities

- SPEAR3 storage ring injection efficiency varies → *trajectory feedback settings are frequently optimized to compensate*

- Use NN model to discover what is driving the change *(i.e. find unanticipated parameter dependencies)*

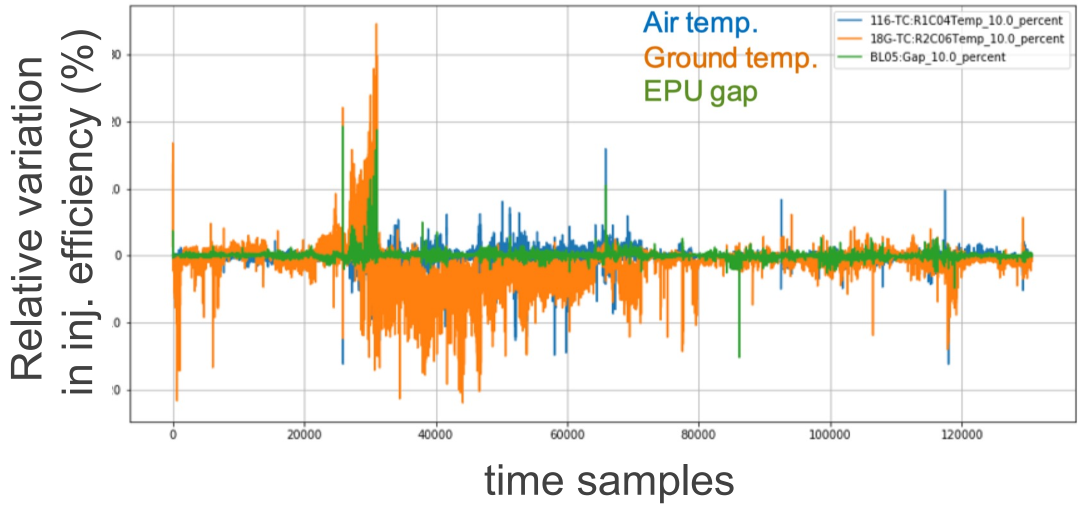




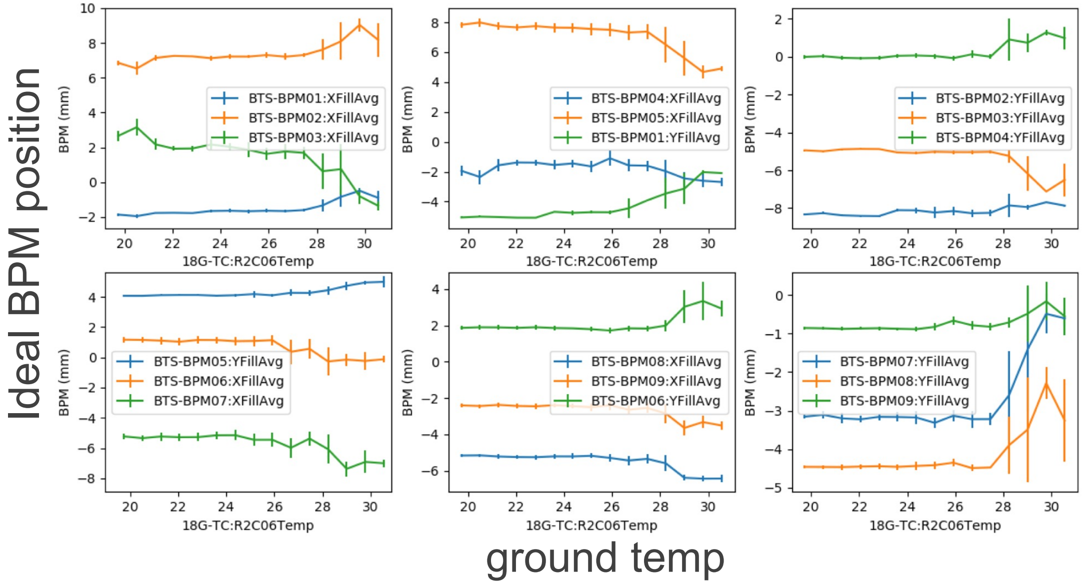
F. Wang, X. Huang, ICFA ML Workshop 2019

Improve system understanding: learn about machine sensitivities

→ *Found ground temperature was a significant factor*



→ *Could now use to predict ideal orbit given ground temperature*



Inverse Models to Help Speed Up Optimization

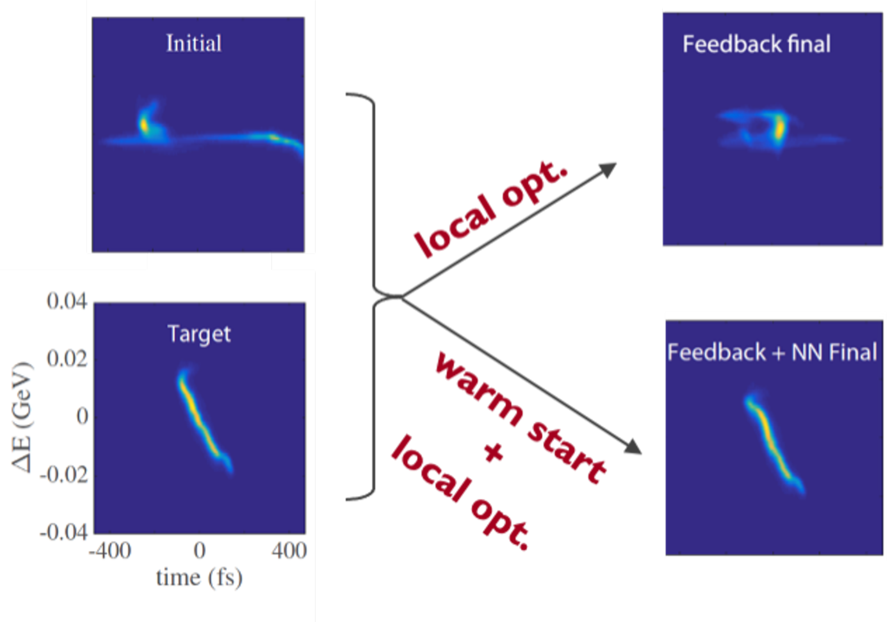
A. Scheinker, A. Edelen, et al., PRL 121, 044801 (2018)
Based on sim study w/ compact FEL: A. Edelen, et al., FEL'17

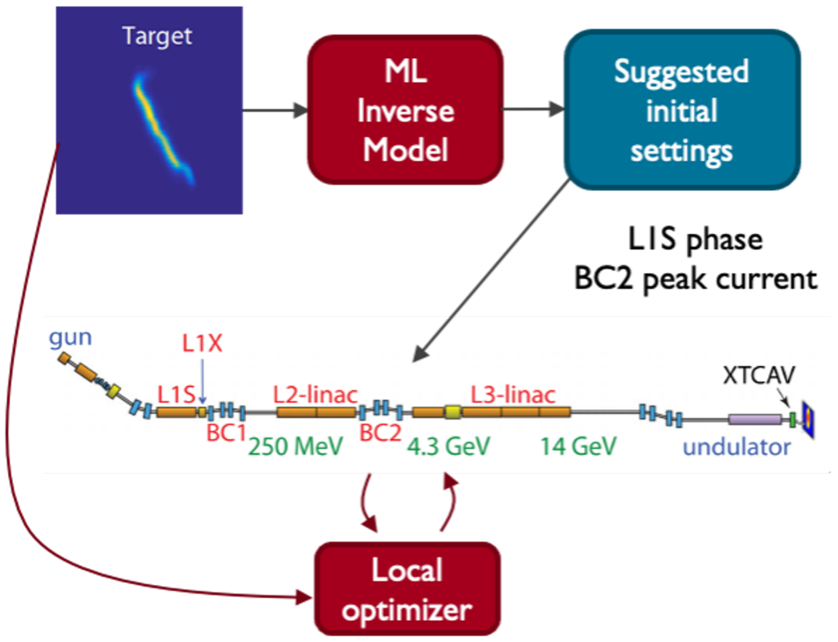
What if we are far away from some target beam parameters and want to switch between configurations quickly?

→ *Use global model to give an initial guess at settings, then refine with local optimization ("warm start")*

Example at LCLS:

- Two settings scanned *(L1S phase, BC2 peak current)*; trained neural network model to map longitudinal phase space to settings
- Compared optimization algorithm with/without warm start





Local optimizer alone was unable to converge → **able to converge after initial settings from neural network**

Other Resources

Excellent visualizations and explanations: https://colah.github.io/

Deep learning textbook (online): www.deeplearningbook.org/

Excellent interactive web book: http://neuralnetworksanddeeplearning.com/

Peer-reviewed tutorials / educational blog: https://distill.pub/

Stanford computer vision course: https://cs231n.github.io/

Interactive report/visualizations for CNN calculations: https://github.com/vdumoulin/conv_arithmetic

Neural network FAQs (old but comprehensive): http://www.faqs.org/faqs/ai-faq/neural-nets/part1/index.html

Questions?