Theoretical Topics in Deep Learning: Introduction

Daniel Soudry
Electrical Engineering
Technion

Supervised Learning: A Short Recap

Types of Learning

Semi-supervised Learning

Multi-task Learning

Transfer Learning

Active Learning

Bandits

Unsupervised Learning

Data representation
Clustering, dimensionality reduction

Supervised Learning

Mapping inputs to outputs
Classification, regression, time series

Reinforcement Learning

Control, learning control, closed loop
Robotics, game playing, driving

Supervised Learning: binary classification

Data $D_N = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ N - sample size

 → 'cat'

 → 'dog'

$\mathbf{x}_n$ $y_n = \{\text{cat}, \text{fog}\}$

Source $(\mathbf{x}_n, y_n) \sim P_{X,Y}$ i.i.d. The underlying rule/regularity

Modeling: Hypothesis space $\mathcal{F} = \{f: \; f(\mathbf{x}) = y\}$

Linear classifier
Polynomial
Neural network
Gaussian mixture
…

$f($  $)$ ='dog'

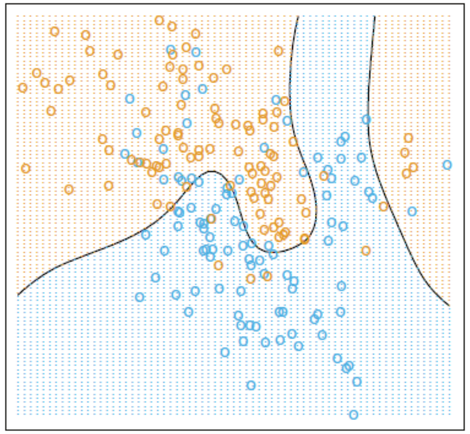
Supervised Learning : A five steps program

Ultimate criterion Minimize probability of error

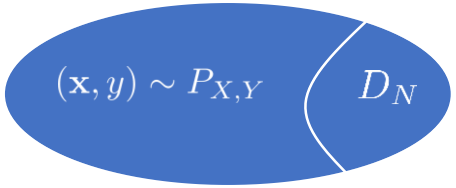
$$\mathcal{L}(f) = \Pr_{(\mathbf{x},y)\sim p_{\mathbf{x},y}}\left(y \neq f(\mathbf{x})\right) \quad = E_{(\mathbf{x},y)\sim p_{\mathbf{x},y}}\left[\mathcal{I}\left[y \neq f(\mathbf{x})\right]\right]$$

Solution: Optimal rule
Bayes classifier

$$f_{\text{Bayes}}(\mathbf{x}) \in \arg\max_y P(y|\mathbf{x})$$



But law $P_{X,Y}$ is unknown!



"Generalization" How f well classifies unobserved examples?

Image: Hastie et al., Elements of Statistical Learning Theory

Supervised Learning : A five steps program

Ultimate criterion

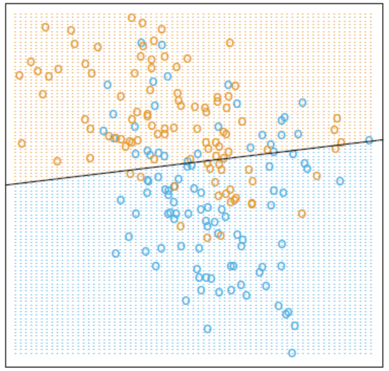
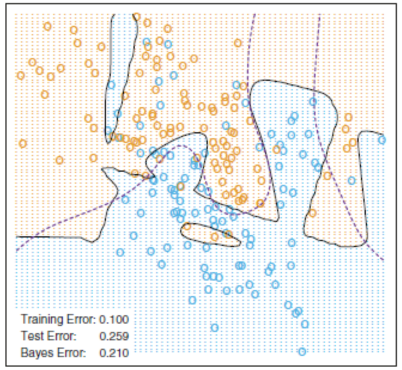
Minimize probability of error $\mathcal{L}(f) = \Pr_{(\mathbf{x},y)\sim P_{X,Y}}(f(x) \neq y)$

Select $\mathcal{F}$, a hypothesis space

Linear classifier, mixture of Gaussians, support vector machine, neural networks (+ architecture, activation functions), …

Best Function in hypothesis space : $f_* \in \arg\min_{f\in\mathcal{F}} \mathcal{L}(f)$

How to Choose a Hypothesis Space? Classical View

	Simple Classifiers	**Complex Classifiers**
Expressivity	 Underfitting	 Overfitting
Optimization	Easy	Hard
Prior knowledge	Incorporate into Hypothesis space	
Better suited to… (Classically)	Small data size	Large data size

Works well in theory, …

Supervised Learning : A five steps program

Ultimate criterion

Minimize probability of error $\mathcal{L}(f) = \mathrm{Pr}_{(\mathbf{x},y)\sim P_{X,Y}}(f(x) \neq y)$

Select $\mathcal{F}$, a hypothesis space

Linear classifier, mixture of Gaussians, support vector machine, neural networks (+ architecture, activation functions), …

Choose a learning criterion

Empirical error, e.g., $\hat{\mathcal{L}}(f) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{I}\left[f(x_n) \neq y_n\right]$

, regularized error, surrogate error

Motivation: generalization, optimization

$$\hat{f} \in \arg\min_{f \in \mathcal{F}} \hat{\mathcal{L}}(f)$$

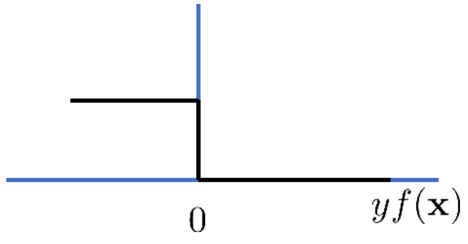
How to Choose a loss Function?

Empiric Error

$Y = \pm 1$

$$\hat{\mathcal{L}}(f) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{I}\left[f(x_n) \neq y_n\right] = \frac{1}{N}\sum_{n=1}^{N}\mathcal{I}\left[f(x_n)y_n < 0\right]$$

Hard to optimize



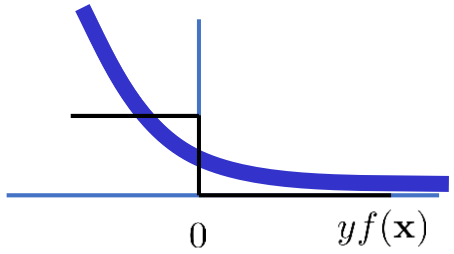
Surrogate error

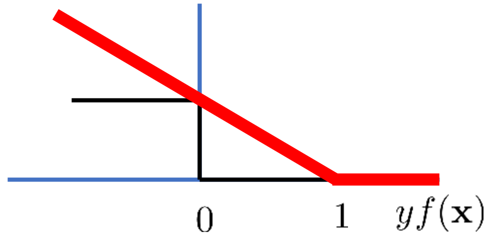
$$\hat{\mathcal{L}}_{\mathrm{sur}}\left(f\right) = \frac{1}{N}\sum_{n=1}^{N}\ell\left(y^{(n)}, f\left(\mathbf{x}^{(n)}\right)\right)$$

A smooth no-negative loss function, with (sub-)gradients

Logistic loss: $\ell\left(y, \hat{y}\right) = \log\left(1 + \exp\left(-y\hat{y}\right)\right)$

Hinge loss: $\ell\left(y, \hat{y}\right) = \max\left(0, 1 - y\hat{y}\right)$





Regularization

$$\hat{\mathcal{L}}_{\mathrm{sur}}(f) + \mathrm{Penalty(f)}$$

Supervised Learning : A five steps program

Ultimate criterion	Minimize probability of error $\mathcal{L}(f) = \Pr_{(\mathbf{x},y)\sim P_{X,Y}}(f(x) \neq y)$
Select $\mathcal{F}$, a hypothesis space	Linear classifier, mixture of Gaussians, support vector machine, neural networks (+ architecture, activation functions), …
Choose a learning criterion	Empirical error, e.g., $\hat{\mathcal{L}}(f) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{I}\left[f(x_n) \neq y_n\right]$, regularized error, smoothed error, surrogate error Motivation: generalization, optimization
Choose how to Optimize	Stochastic gradient descent (SGD), momentum, Adam,… Tune optimization hyper-parameters (e.g., learning rate schedule)
Choose Initialization	Very important in neural networks (typically random). Less important in (strongly) convex models.

Optimization

Parameterized Hypothesis space: $f\left(\mathbf{x};\boldsymbol{\theta}\right)$

Learning rate / step size

"Gradient Descent (GD)": $$\Delta\theta_i = -\frac{\eta}{N}\sum_{n=1}^{N}\frac{\partial\ell(f(\mathbf{x}_n;\boldsymbol{\theta}),\mathbf{y}_n)}{\partial\theta_i}$$

More common
"stochastic Gradient Descent (SGD)": $$\Delta\theta_i = -\eta\frac{\partial\ell(f(\mathbf{x}_n;\boldsymbol{\theta}),\mathbf{y}_n)}{\partial\theta_i}$$

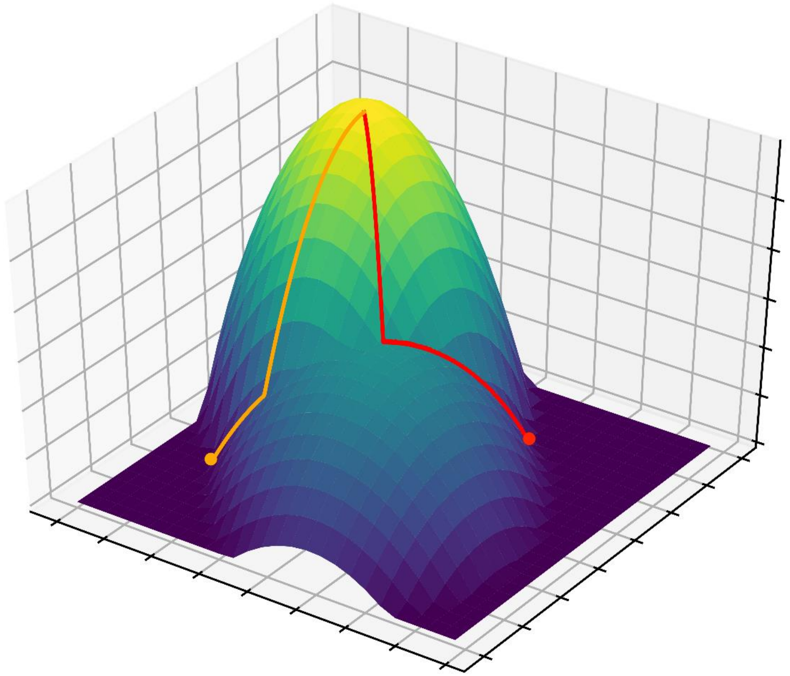
Optimization Path: $\boldsymbol{\theta}^{(0)},\ldots,\boldsymbol{\theta}^{(t)}\xrightarrow{t\to\infty}\boldsymbol{\theta}^{(\infty)}$ ⇨ $f^{(0)},\ldots,f^{(t)}\xrightarrow{t\to\infty}f^{(\infty)}$

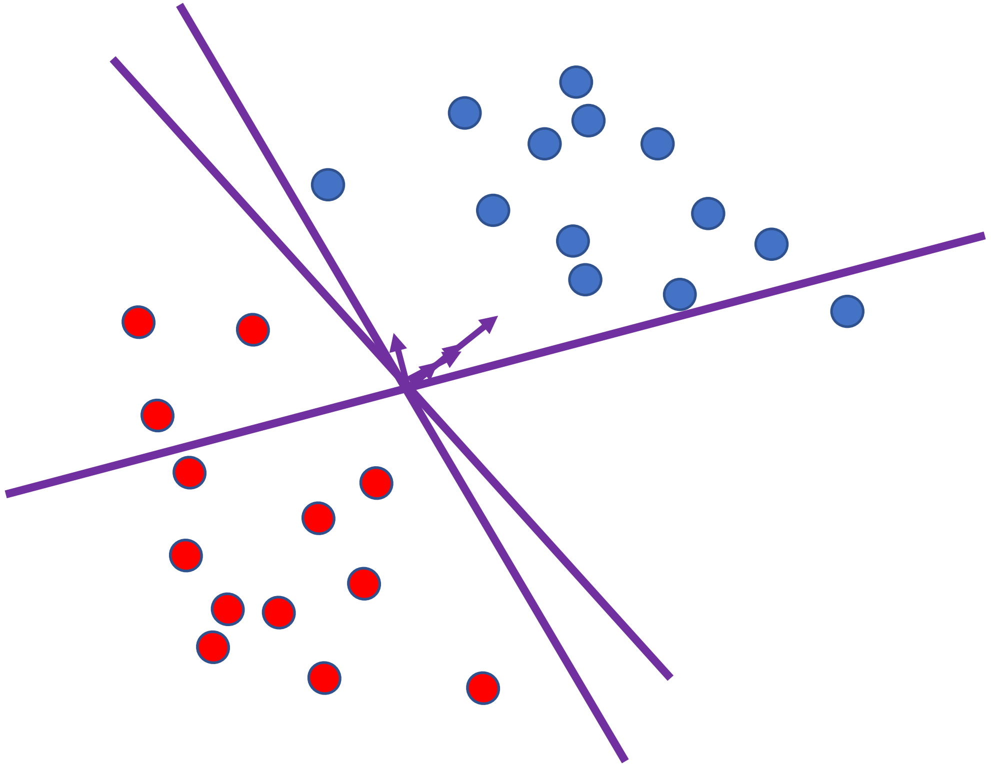
Local methods – might get stuck in "bad" asymptotic solutions, such local minima

Other methods (e.g., Momentum, Adam)?
Issues to consider: scalability (hardware dependent), convergence speed, implicit bias?

Implicit bias

Initialization, optimization, surrogate loss selects a ***specific*** optimum





Different optimization algorithm

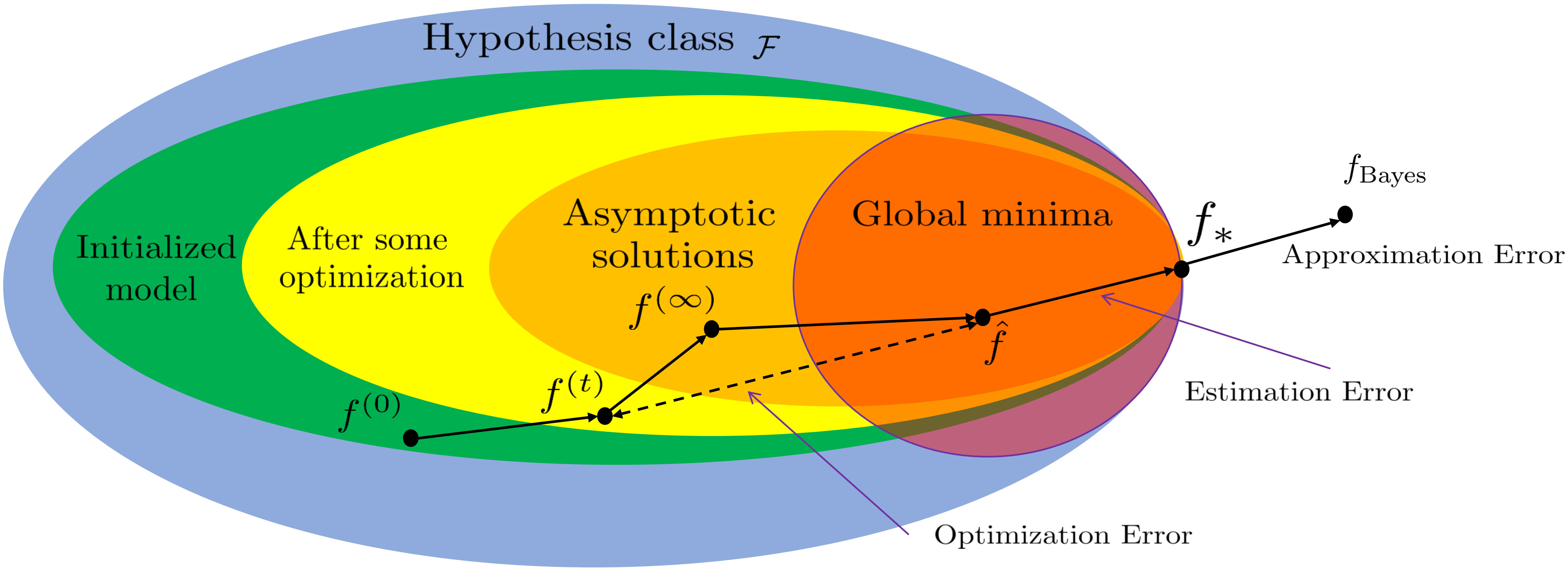
➔ Different optimum reached

➔ Different Inductive bias

➔ Different learning properties

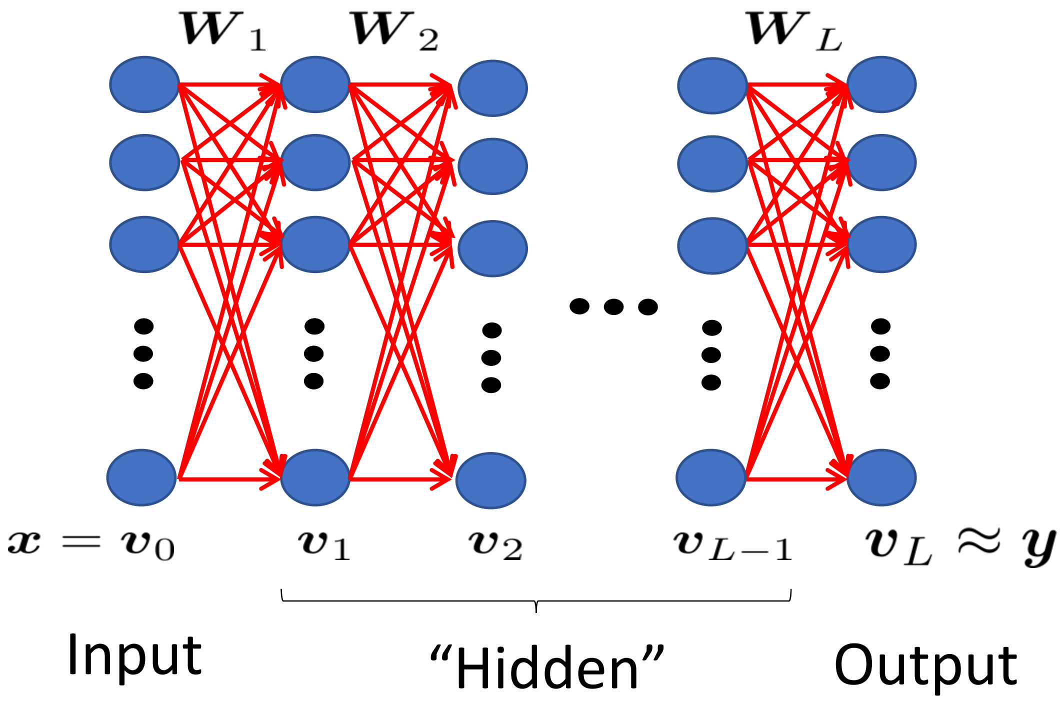
The Sources of Error

$$\mathcal{L}(f^{(t)}) = \underbrace{\left(\mathcal{L}(f^{(t)}) - \mathcal{L}(\hat{f})\right)}_{\text{Optimization Error}} + \underbrace{\left(\mathcal{L}(\hat{f}) - \mathcal{L}(f_*)\right)}_{\text{Estimation Error}} + \underbrace{\left(\mathcal{L}(f_*) - \mathcal{L}(f_{\text{Bayes}})\right)}_{\text{Approximation Error}} + \underbrace{\mathcal{L}(f_{\text{Bayes}})}_{\text{Best Possible}}$$



Neural Networks: An empirical success

What is a neural network?
Basic Example: **Multilayer Neural Networks**



Activation function

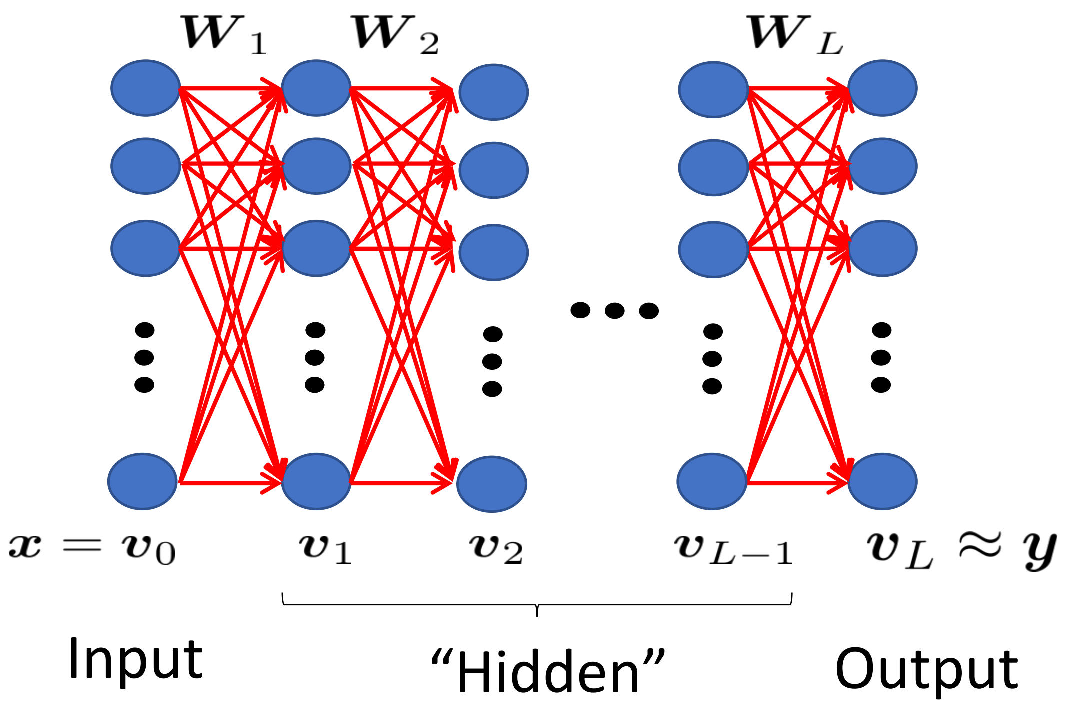
$$v_l = \sigma(W_l v_{l-1} + b_l)$$

Neuron /unit

Weights

Bias

How do we learn the weight?



We test performance on unobserved examples ("validation set" / "test set")

"Gradient Descent (GD)":

$$\Delta w_i \propto -\sum_{n=1}^{N} \frac{\partial \ell(\mathbf{v}_L(\mathbf{x}_n), \mathbf{y}_n)}{\partial w_i}$$

More common "stochastic Gradient Descent (SGD)":

$$\Delta w_i \propto -\frac{\partial \ell(\mathbf{v}_L(\mathbf{x}_n), \mathbf{y}_n)}{\partial w_i}$$

All derivatives can be calculated efficiently using backpropagation (the chain rule "in reverse")

Neural Networks Zoo

- Multilayer Neural Networks (MNNs)
- Convolutional Neural Networks (convnets)
- Recurrent nets
- Attention models
- Auto-encoders
- Ladder nets
- Neural Turing machines
- Normalizing flows
- …

Architectures incorporate **priors about data**:

- sound / speech (2D signal)
- images (3D signal)
- videos (4D signal)

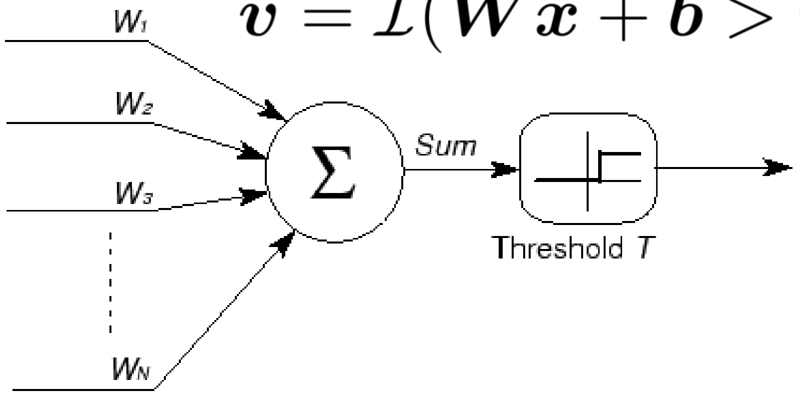
Main theme:

Large, nonlinear, (mostly) differentiable models optimized with SGD (or variants)

The History of Neural Networks I

The “single neuron” age:

$$v = \mathcal{I}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} > 0)$$

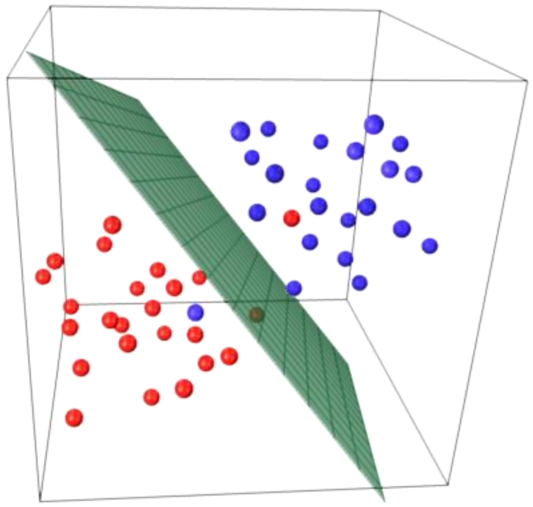


1943 Mcculloch Pitts neuron

Proved: network of these can implement any finite state machine
Generalization [Siegelmann&Sontag 1995]: any Turing machine

1957 The perceptron algorithm by Roessnblat

Algorithms learns the weights of a single neuron to minimize classification error
Proved: convergence in a finite number of iterations



1969 Minsky and Papert:

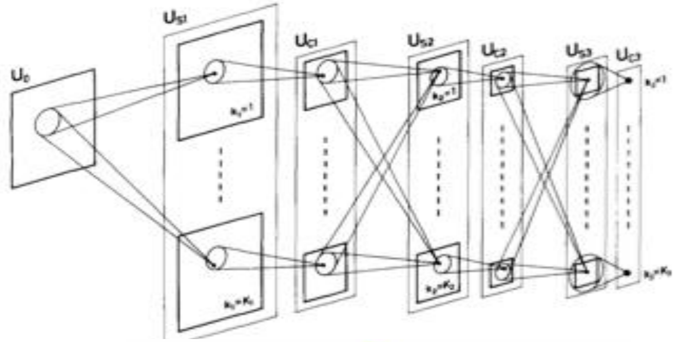
A single layer of neurons can only perform linear separation
However, many datasets are non-linearly separable

1969-1986 The first dark age of neural networks

The History of Neural Networks II

1980 Fukushima: Neocognitron – father of convnets

Multilayer connectivity with structered connecivity (“receptive field”)

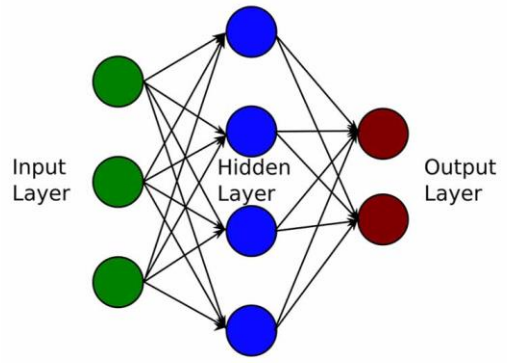


The “Backpropagation” age:

1986 Rumelhart, Hinton and Williams:

Multilayer neural networks – more than linear classifiers

Trained using SGD, after rediscovery of **Backpropagation**

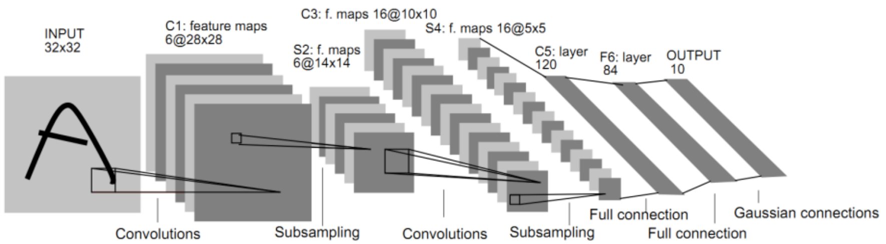


1989 The universal approximation Theorem

Proved: one hidden layer is enough

1989-1998 LeCun: Convnets

Successful industrial applications (optical character recognition)



1998-2012 The second dark ages of neural networks

SVM, boosting & Gaussian processes take over.

Only Hinton, LeCun, Bengio, & Schmidhuber keep trying…

Backpropagation (BackProp)

Implements stochastic gradient descent: $\Delta \mathbf{W}_l = -\eta \nabla_{\mathbf{W}_l} \ell(\mathbf{y}, \mathbf{v}_L)$ $\quad$ (assume $\mathbf{b}_l = 0$)

- Forward Pass

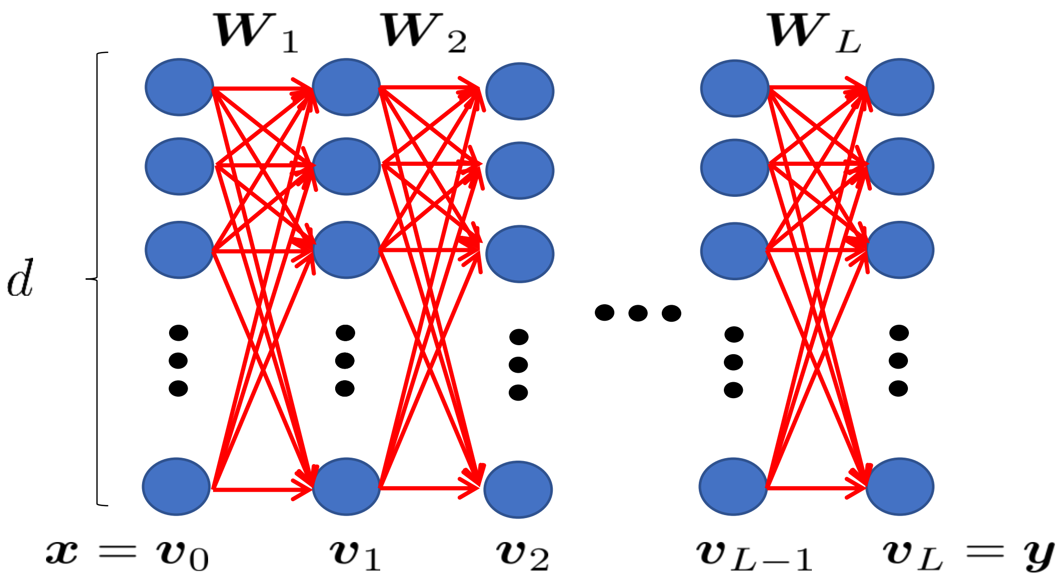
$$\mathbf{u}_l = \mathbf{W}_l \mathbf{v}_{l-1}$$
$$\mathbf{v}_l = \sigma(\mathbf{u}_l)$$

- Backward Pass

$$\boldsymbol{\delta}_{l-1} = \mathbf{W}_l{}^\top \delta_l \odot \sigma'(\mathbf{u}_{l-1})$$

- Update

$$\Delta \mathbf{W}_l = \eta \delta_l \mathbf{v}^\top{}_{l-1}$$



Initialize: $\mathbf{v}_0 = \mathbf{x}$

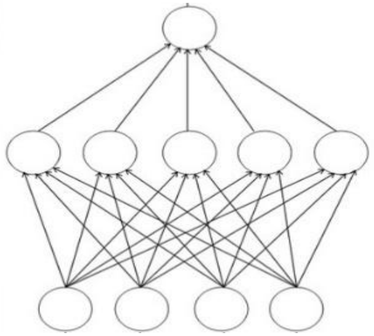
$$\boldsymbol{\delta}_L = \frac{\nabla_{\mathbf{v}} \ell(\mathbf{y}, \mathbf{v})|_{\mathbf{v}=\mathbf{v}_L}}{\partial \mathbf{u}_L}$$

- Origins in control theory of 1960s [Bryson & Ho 1969]

η - learning rate parameter

Universal Approximation Theorem

Theorem A wide enough hidden layer with non-polynom activations can approximate (no learning)



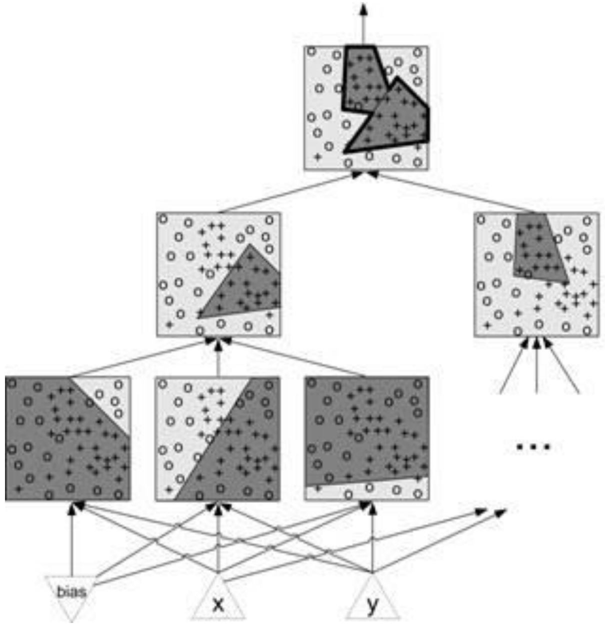
$\sigma\left(w_i^\top x + b\right)$

x

Regression any continuous function

Classification arbitrary decision regions

(“easier” with two hidden layers)



But, may require exponentially many units!

Why did the excitement wane during the 90’?

Neural networks

Advantages:

- Universal approximation property
- Typically had best empirical results in vision datasets

Disadvantages:

- Long training times
- Failure to train very deep networks
- Non-convex: fear of local minima
- Not enough data: fear of overfitting
- Black-box with no guarantees
- Many hyper-parameters to tune

Others methods: SVM, boosting & Gaussian processes

Advantages:

- Elegant theory
- Few tuning parameters
- Convex optimization
- Excellent empirical results in many domains

Disadvantages:

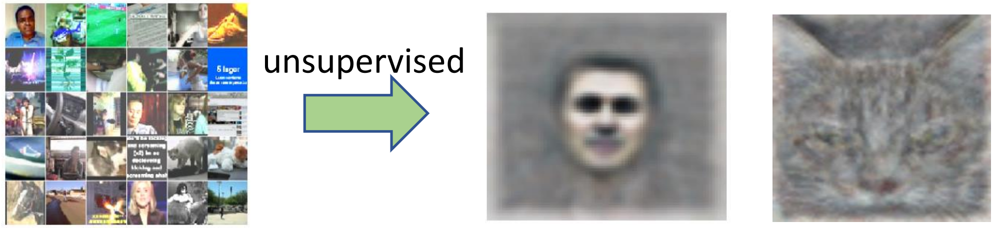
- Required features engineering
- Some methods were not scalable to large datasets

The History of Neural Networks III

2006 Hinton coined term "deep learning"

unsupervised pre-training: significant improvements and excitement

Le et al. 2012: 10^9 weights, 16,000 cores



10^7 Youtube movies

"cat" and "faces" features

The "Big Data" age:

2012 Hinton: AlexNet shatters competition:

Improves ImageNet state-of-the-art by 50%!

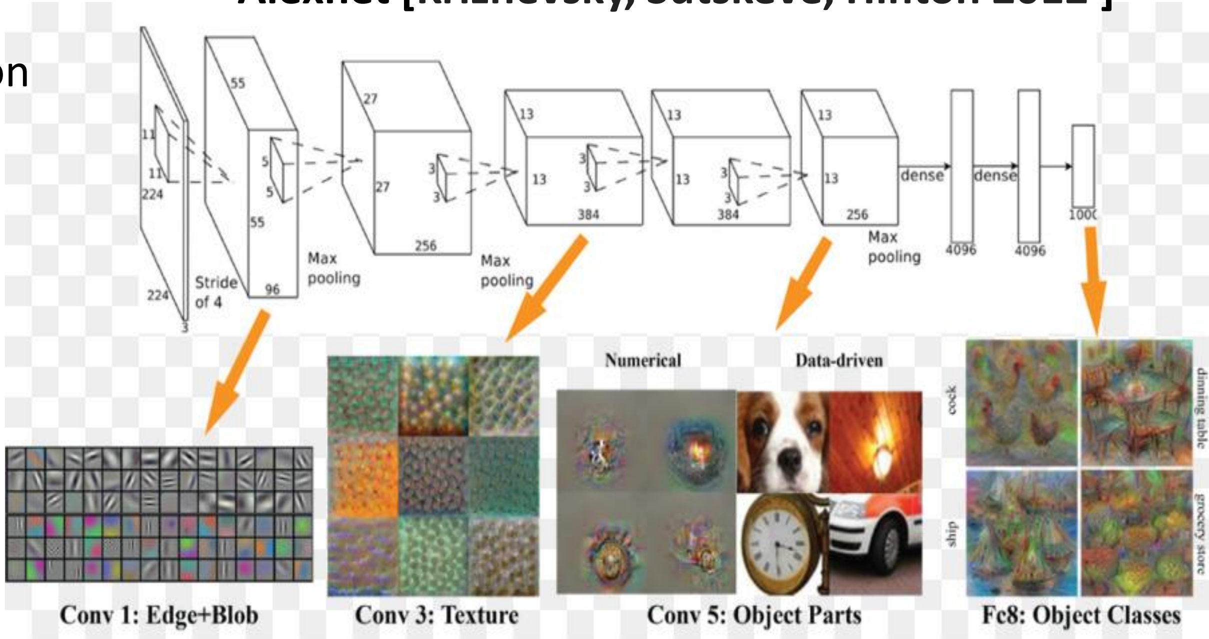
Indicated: large supervised datasets + convnets -> good idea

Many novel ideas: ReLU activations, dropout, GPU parallelization

Also: no need for unsupervised pre-training

…. Deluge

Alexnet [Krizhevsky, Sutskeve, Hinton 2012]



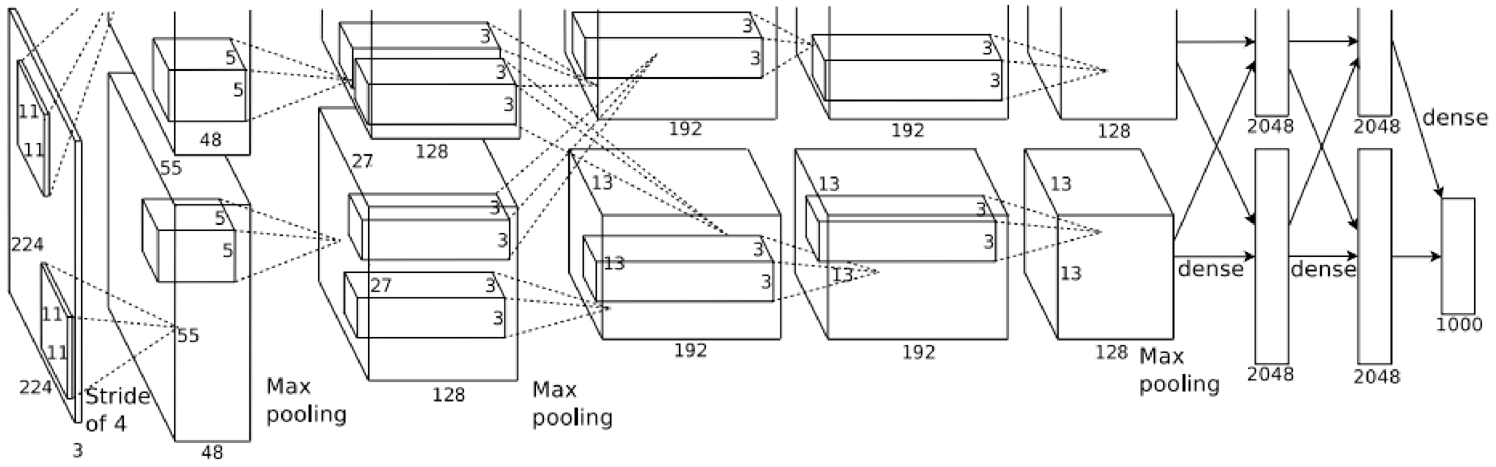
2015 Batch Norm + ResNets

made it easy to train very deep nets

Also: no need for dropout

…. Deluge continues

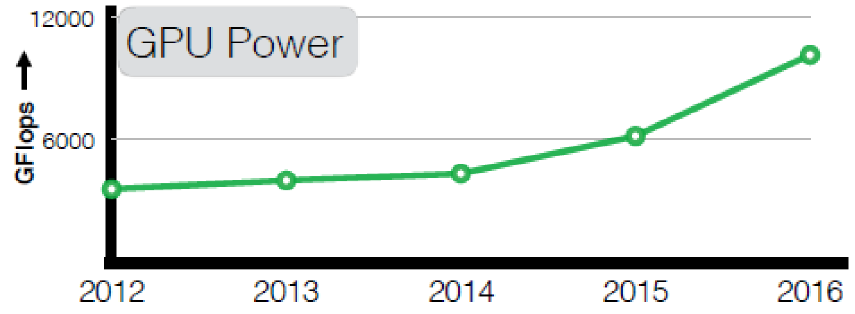
Why AlexNet?



- [Krizhevsky'12] win 2012 ImageNet classification with a **much bigger ConvNet** than before:
  - **deeper**: 7 stages vs 3 before
  - **larger**: 60 million parameters vs 1 million before (and only **1.2M #data samples**)

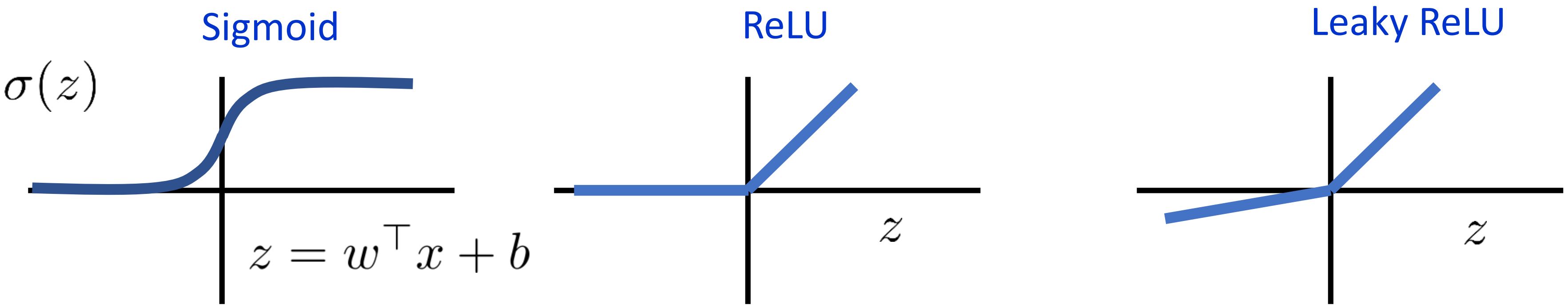
- This was made possible by:
  - **fast hardware**: GPU-optimized code
  - **big dataset**: 1.2 million images vs thousands before
  - **better regularization**: dropout
  - **new activation functions**: ReLU

Since then:



Novel Activation Functions

- Theoretically, any non-polynomial will do
- Practically, ReLU are widely applied



Why?

Heuristic suggestions:

- Piece-wise constant gradient – alleviates vanishing gradient
- Sparse representations

Some Theory:

- Strictly decreasing path to minimum from high enough initializations (Safran & Shamir 2016)
- 1-homogenous functions lead to margin maximization (Wei, Lee, Liu, Ma 2018)

Why ResNet? a case study

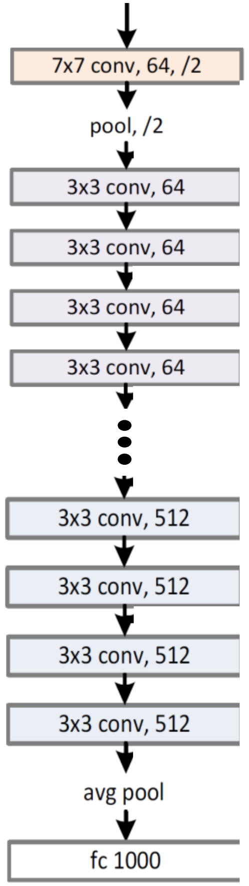
- Deeper -> better
- More parameters -> better
- Training error bottleneck
- Important: skip connections,

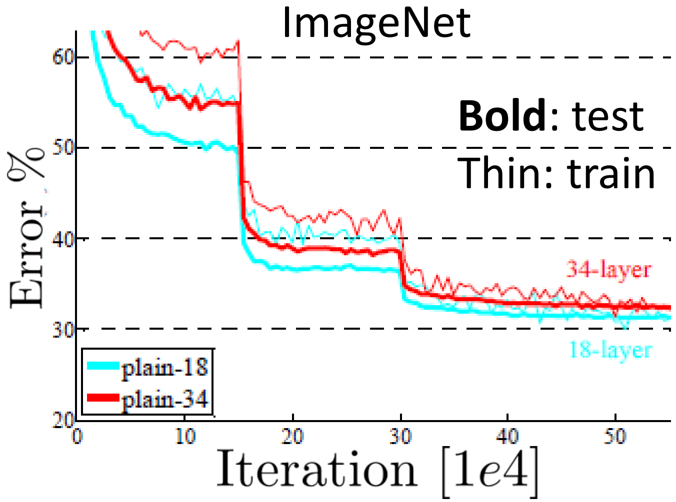
"Batch norm" / Initialization

	# layers	# params	% test error
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

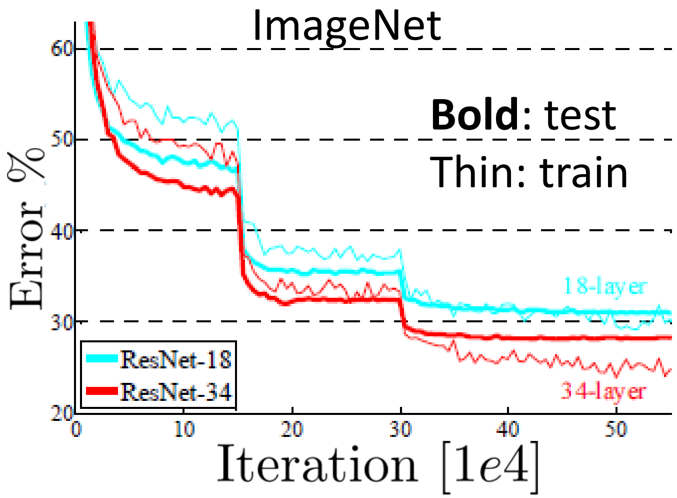
CIFAR-10: 50k training images

ResNet
[He et al. 2015]





add skip connections



Why?

The recent Success of Deep Learning

State-of-the-art results in many fields

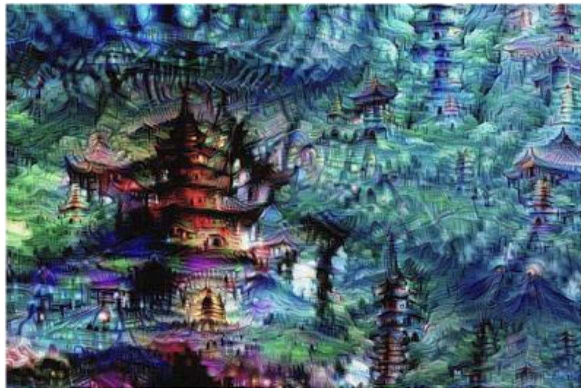
- Object recognition from images
- Image manipulation

[Lample et al. 2017]

- Speech recognition
- Machine Translation
- Atari, Go games

…

- Even abstract art



[“Deep dream” Mordvintsev et al. 2015]

What Has Changed?

- Abundance of **supervised** data
- Computer power – cheap, fast GPUs

Initial phase

- Empiric Claim: Depth allows learning flexible meaningful representations
- Empiric Claim: Even without convexity, somehow SGD still does not get “stuck”
- Transfer learning through pre-training (e.g. train on ImageNet, then on other data)
- Distributed algorithms – multi-cores/computers
- Improved regularization (dropout, batch-norm, data augmentation)
- More efficient activations (ReLU,pooling)

Social aspects

- Large groups in Industry – Google, Facebook, Microsoft, …
- Rapid sharing of information and code through the web
- Free and continually updated software with built in auto-differentiation: just specify model, no need to painfully calculate Backpropagation gradients

Resources – Software

Deep Learning Tools

Non-Symbolic Frameworks:

- **PyTorch** – provides a Matlab-like environment for state-of-the-art machine learning algorithms in C, Lua
- **Caffe** – Deep Learning framework by Berkeley AI Research

Symbolic Computation Frameworks

- **Tensorflow** – An open source software library by Google for numerical computation using data flow graphs.
- **Keras** – High-level Neural Network API (written in Python)

- ..... and many more. For more details see for example
  - http://deeplearning.net/software_links/
  - https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

To Learn About Deep Learning in Practice

Books:

- I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

Online course lectures on YouTube:

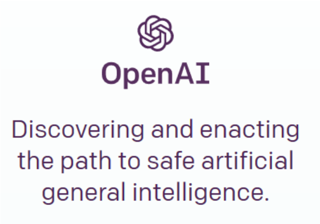
- Oxford University (Nando de Freitas)
- Stanford – CNNs, Deep Learning for NLP

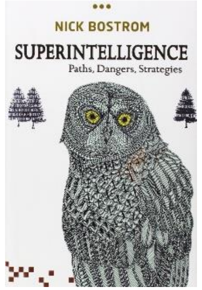
An introductory online course by Google (Udacity, free; also on YouTube):

- https://www.udacity.com/course/deep-learning--ud730

Partial Summary

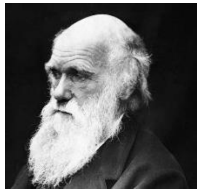
- Progress driven mainly by impressive empirical success
  - Often exceeding human level
- No one predicted this!
  - The opposite was the case



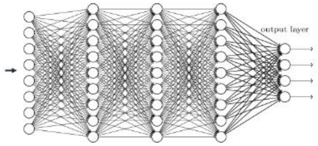


Why?

- Accumulation of many, mostly heuristic, ideas with powerful computers, huge data sets, large groups, software sharing, rapid dissemination of information



- Theory lagging far behind practice
  - Mostly explains why it shouldn't work …
  - Great challenges for theorists!
  - Can theory guide practice?



- Understanding DNNs akin to reverse engineering biological systems

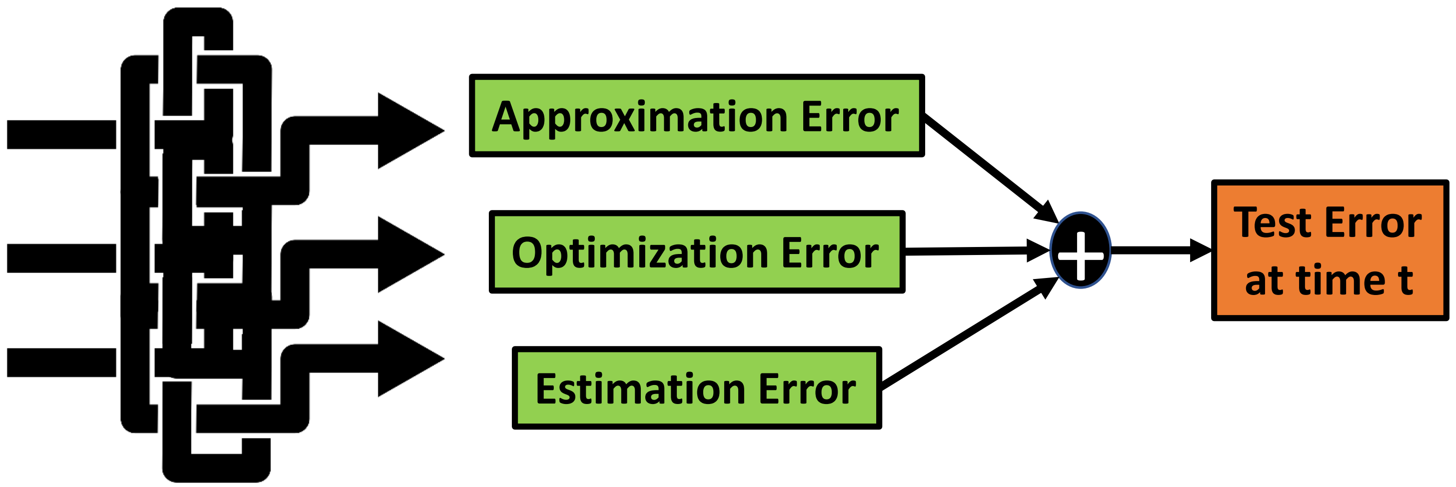
What makes Neural Nets work so well?

How can we make them even better?

Main practical question

How do all the details

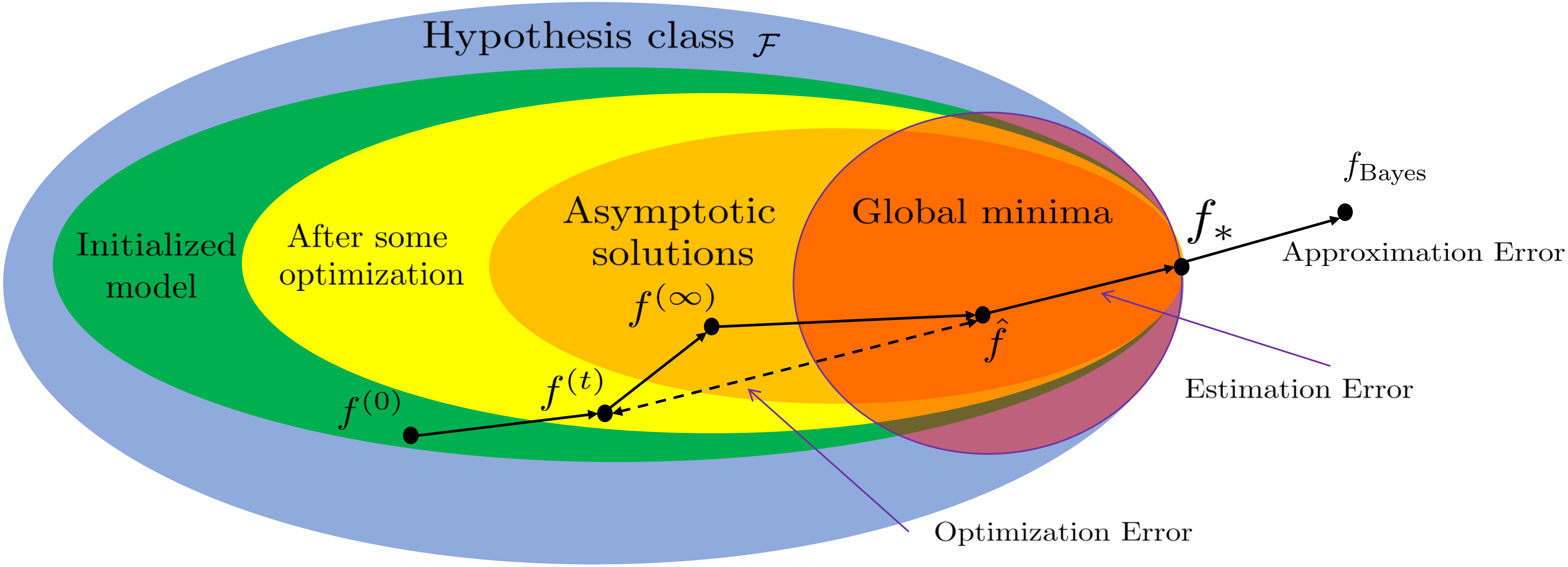
- Initialization
- Architecture
- Surrogate loss function
- Activations functions
- Regularization
- Optimization algorithm
- Hyper-parameters
- Other “tricks” (e.g., batch-norm)



Interact and affect the final test error?

Recall: the Sources of Error

$$\mathcal{L}(f^{(t)}) = \underbrace{\left(\mathcal{L}(f^{(t)}) - \mathcal{L}(\hat{f})\right)}_{\text{Optimization Error}} + \underbrace{\left(\mathcal{L}(\hat{f}) - \mathcal{L}(f_*)\right)}_{\text{Estimation Error}} + \underbrace{\left(\mathcal{L}(f_*) - \mathcal{L}(f_{\text{Bayes}})\right)}_{\text{Approximation Error}} + \underbrace{\mathcal{L}(f_{\text{Bayes}})}_{\text{Best Possible}}$$



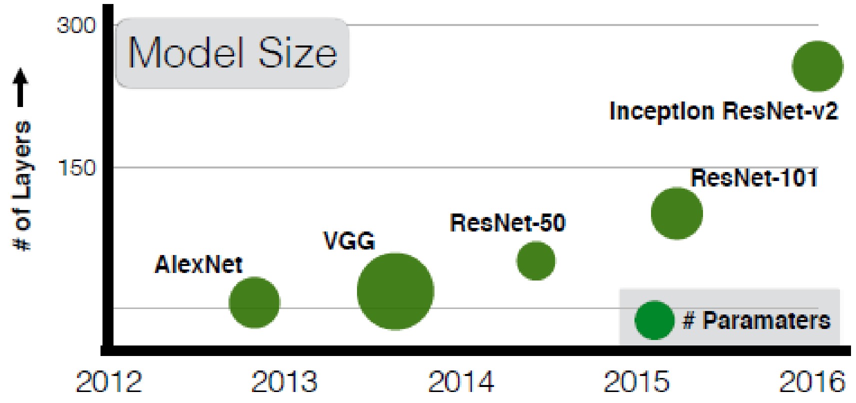
Other practical questions

- Quantify uncertainty in neural net prediction?

- Computational resources : Resource efficient inference/training?

- Decrease the need for labeled data

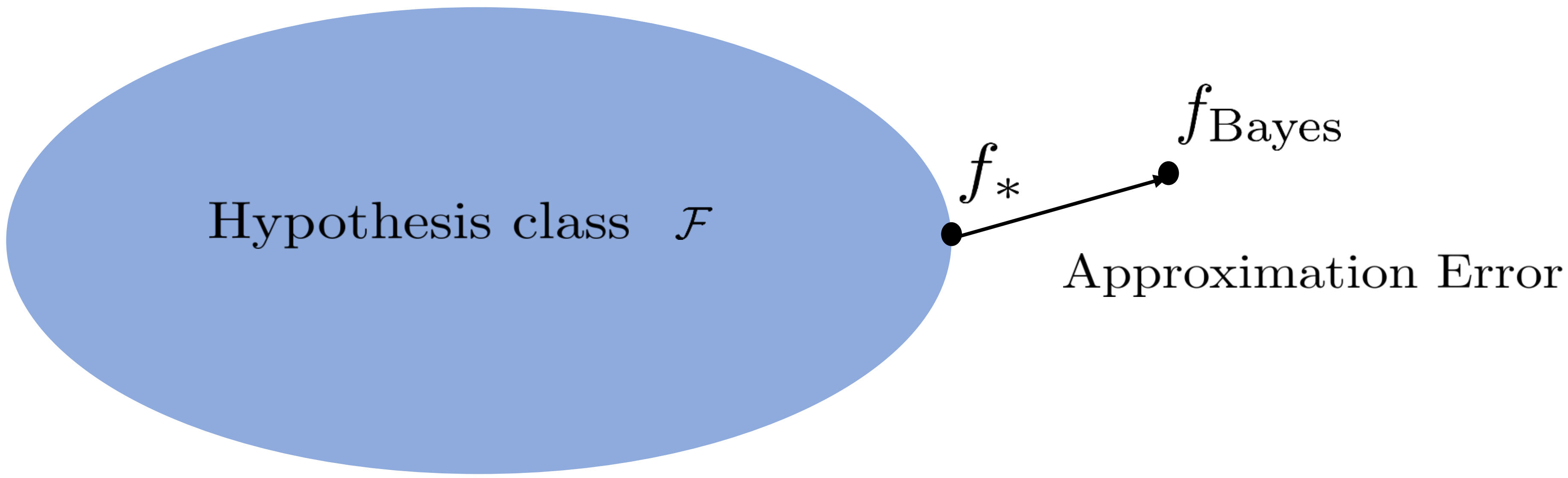
e.g., using transfer learning, unlabeled data?



Approximation Error

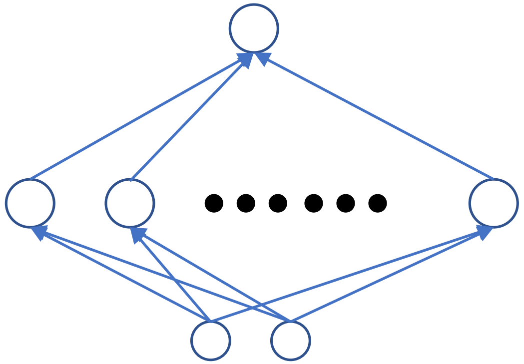
Can we control the approximation error?

- Can it vanish?
- At what rate does it vanish?
- How it is affected by the choices we make
  - Activation function
  - Width
  - **Depth**

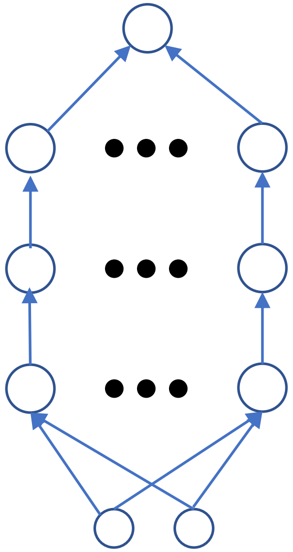


Benefits of Depth in ReLU networks?

Complexity measure: # decision regions



RL units



L layers

R units in each layer

Claim: Exponentially more decision regions for deep nets

Montufar et al., 2014

Fewer units for given setup

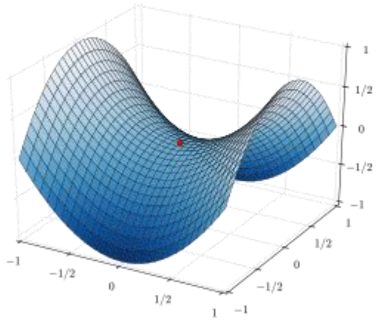
But how does this affect the approximation error?

Benefit of compositionality (hierarchy)

Other activation functions?

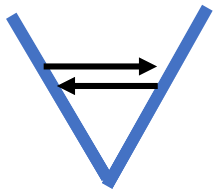
Optimization Error

Optimization convergence guarantees



Classical Guarantees (for smooth loss, and bounded dynamics):

- SGD converges to stationary points (zero gradient) [Bottou 1998]
- Points cannot be strictly saddle (“unstable”) [Pemantle 1990]
- Similar results for Gradient Descent, other first order methods [Lee et al. 2016]



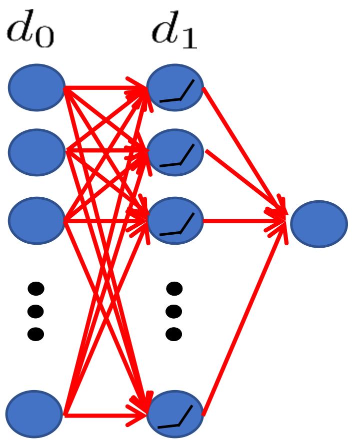
However, for neural nets:

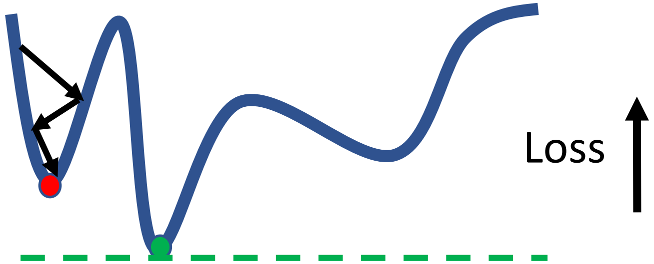
- Non-strict saddle point exist (e.g. for MNN with L>2)
- Smoothness of loss – does not hold for ReLUs [Davis, Drusvyatskiy, ,Kakade, Lee 2018]
- Finite critical points may not exist [Soudry, Hoffer, Shpigel-Nacson, Srebro, ICLR 2018]
- Critical points are generally not global minima

Neural network Landscape

Neural network loss is highly non-convex:

- Multiple local minima exist [Fukumizu & Amari 2000]
  - Even a single neuron can have exponentially many local minima [Sima 2002, Shamir 2016]
- **Naively, SGD should get stuck in “bad” local minima**
- Many hardness results, e.g.
  - NP-hard: $\widehat{\mathrm{MSE}} > \frac{c}{d_1^2}$ [Bartlett&Ben David2002]



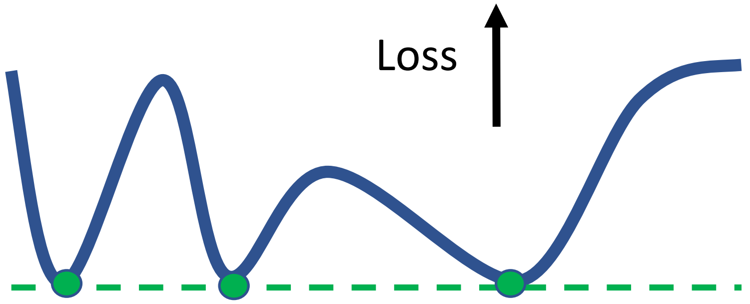


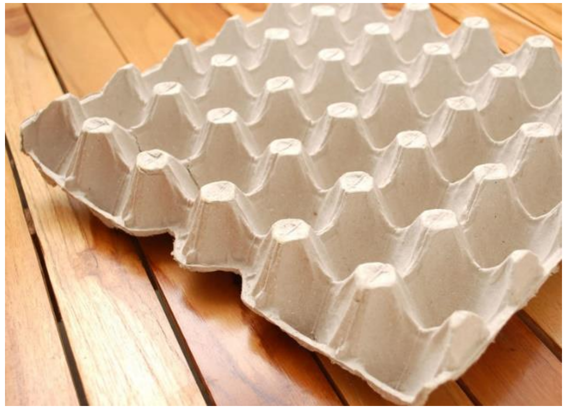
Empirically, training is “well behaved”

Typically, training error:

- Does not depend much on initialization
- Descends on single smooth slope path, no “barriers” [Goodfellow et al. 2014]

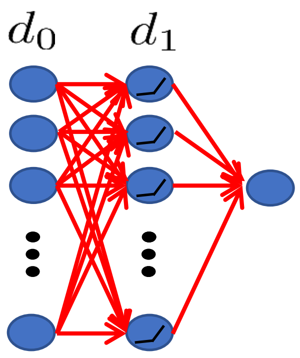
- **Similar training error in all local minima** [Dauphin et al. 2014]



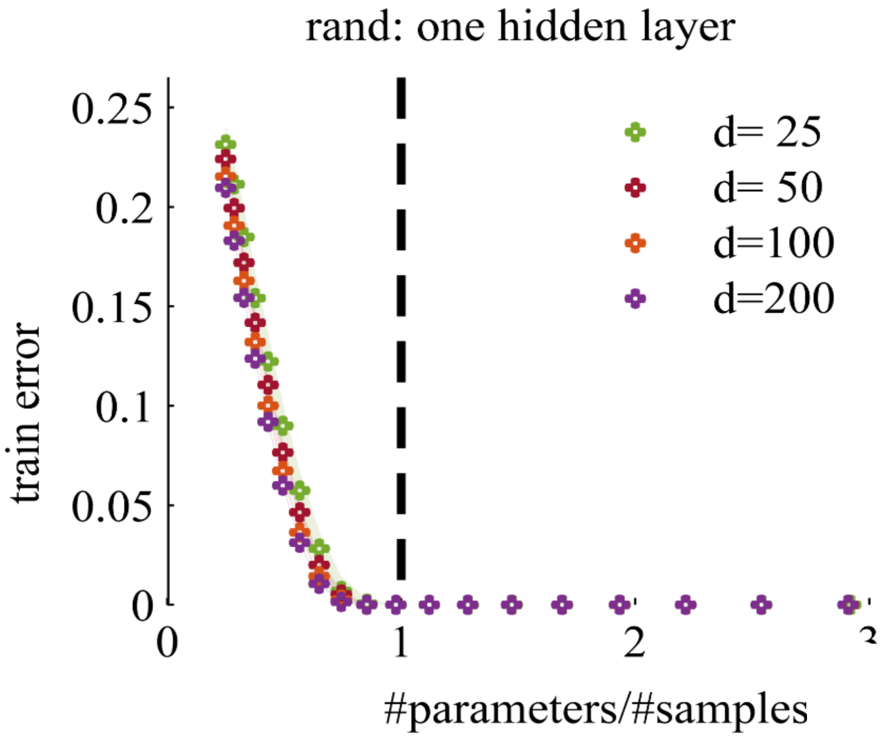


Why?

#parameters ≥ #samples ➡ Low training error



$d_1 = d_0 = d$



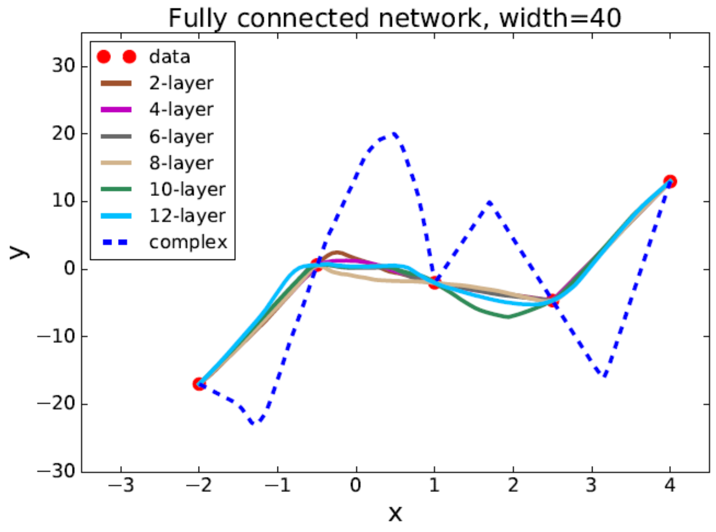
Network can fit random data!

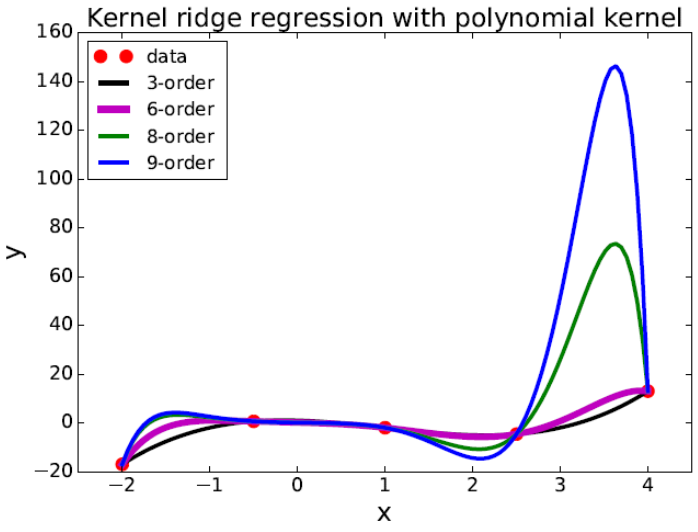
Similar results on other datasets (mnist, CIFAR, ImageNet)

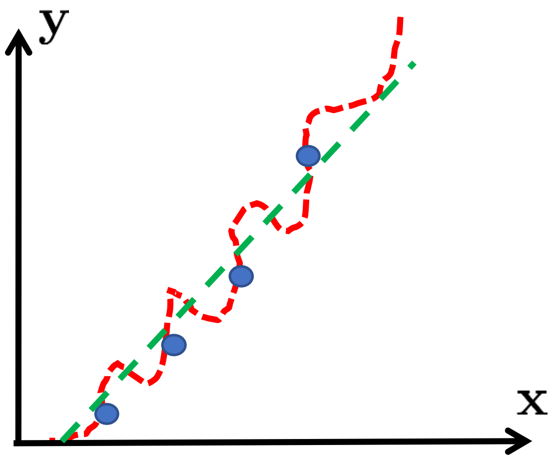
Estimation Error

Why is overfitting not much of a problem?

- Generalization when #parameters >> #samples?
  - Even without explicit regularization [Zhang 2017]

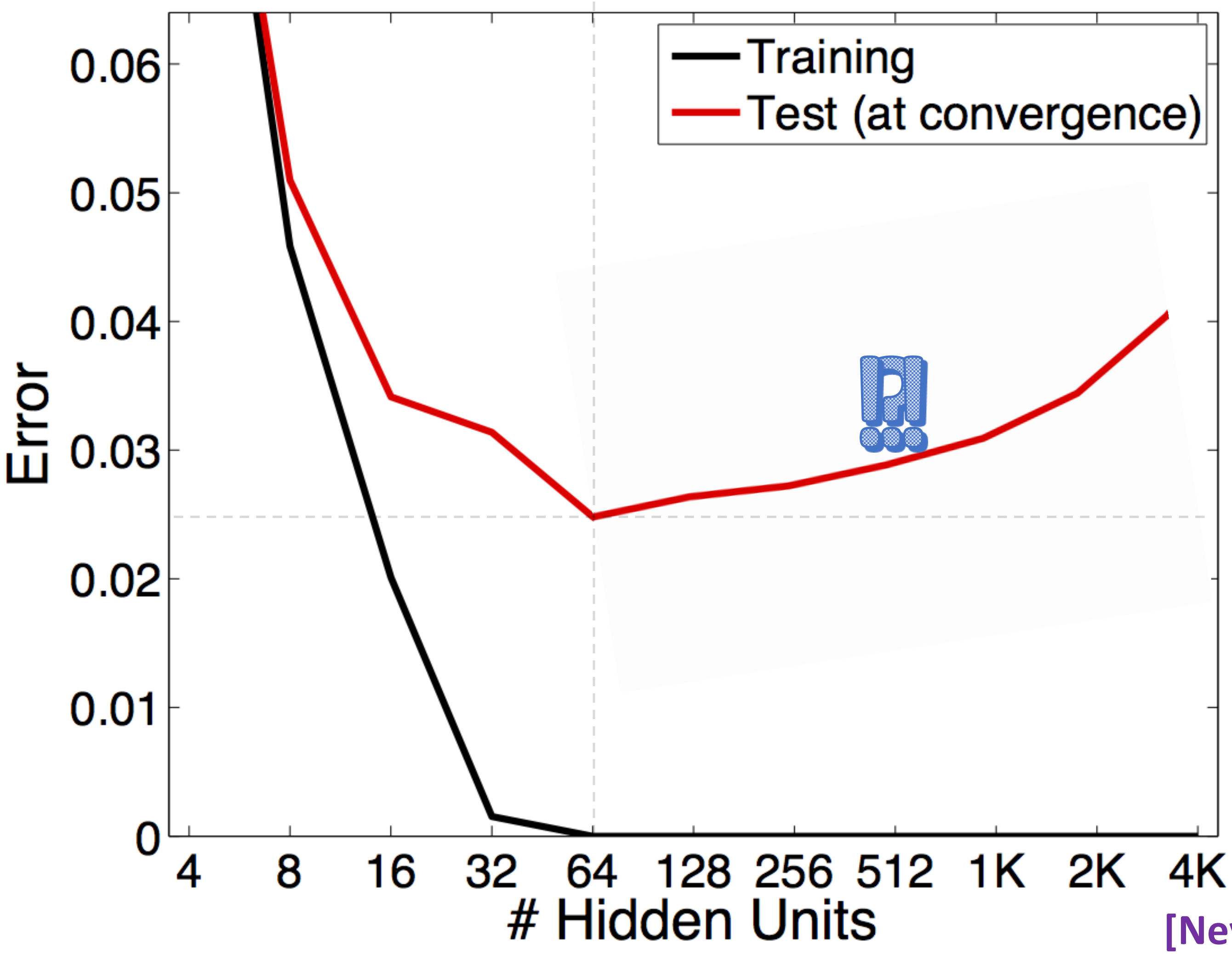






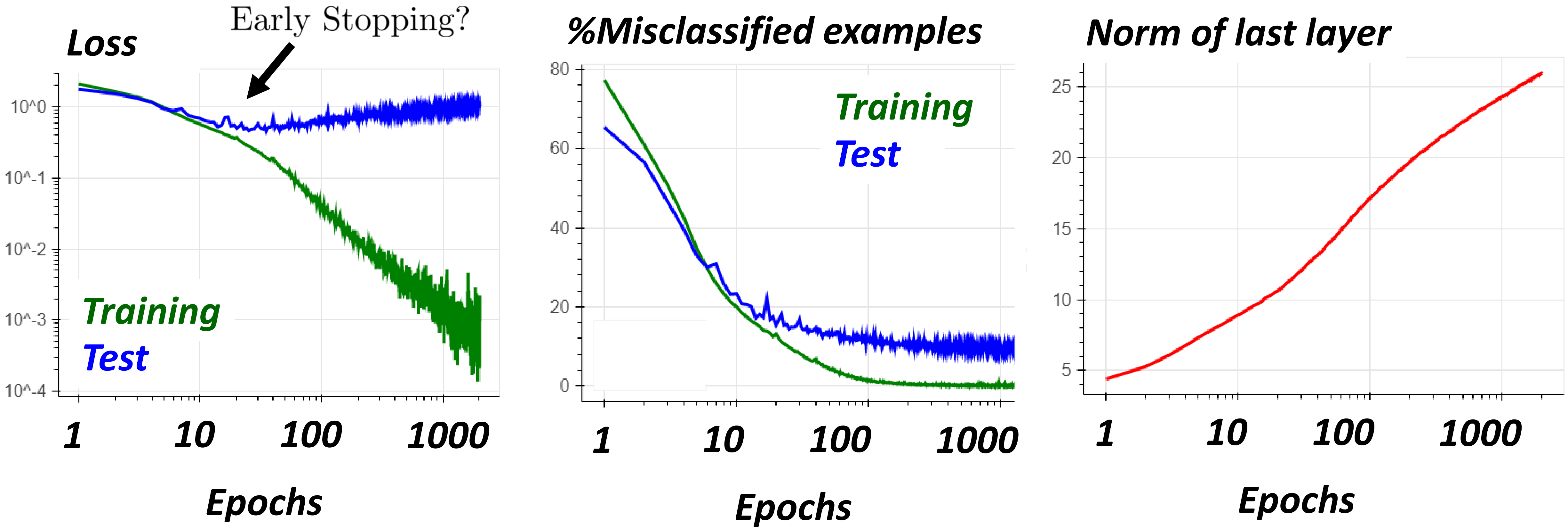
[Wu, Zu & E 2017]

Also for classificaiton



[Neyshabur Tomioka S ICLR’15]

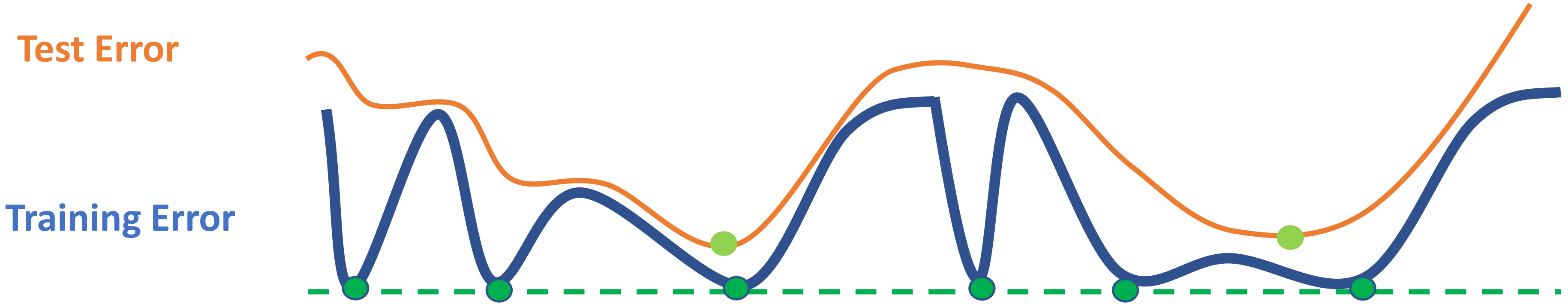
No overfitting – also in dynamics



Dataset: CIFAR10, **Architecture:** Resnet44, **Training**: SGD + momentum + gradient clipping

Generalization (test-train) error ?

- Global bounds useless, since empirically: $\mathcal{L}(f) \leq \hat{\mathcal{L}}(f) + \frac{\Omega(\mathcal{F})}{N^{\alpha}}$



Complexity, e.g.,
parameters,
VC dimension
Practically useless
$\#(\text{params}) > N$

- One non-vacuous (<1) generalization bound [Dziugaite&Roy 2017]: $\mathcal{L}(f) \leq \hat{\mathcal{L}}(f) + \frac{\Omega(f)}{N^{\alpha}}$

MNIST dataset:

Experiment	T-600	T-1200	T-300^2	T-600^2	T-1200^2	T-600^3	R-600 (Random)
Train error	0.001	0.002	0.000	0.000	0.000	0.000	0.007
Test error	0.018	0.018	0.015	0.016	0.015	0.013	0.508
PAC-Bayes bound	0.161	0.179	0.170	0.186	0.223	0.201	1.352

#parameters >> #samples

PAC-Bayes approach. **Numeric, depends on final solution, not much insight…**

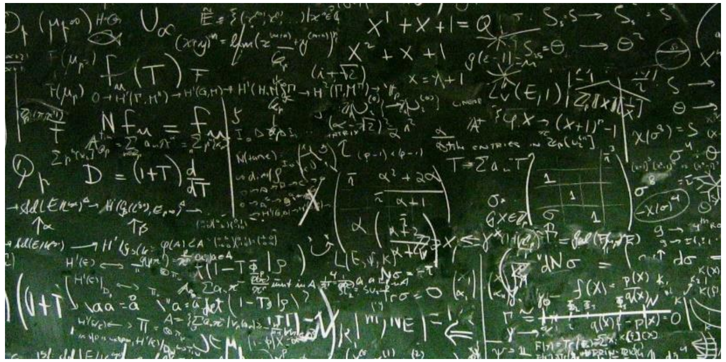
Course Details

Goals of Course

- Present both classical and recent results on the questions we discussed so far.
- Aim to elucidate main ideas and intuition behind results.
- Aim to understand how results combine and interact.
- Help students learn to read and interpret literature

Obstacles:

- Field is rapidly changing
- Challenging literature



Structure of Course

Currently ~24 registered students, this implies the current plan:

- 7 lessons given by me (possibly also my students + guest lectures):
  - Approximation Capabilities of Neural nets
  - Optimization: basics results + Loss Landscape of Neural nets
  - The implicit bias of optimization Algorithms
  - Bayesian neural nets
- 6 lessons given by course students, in groups of 4. Suggested Topics:
  - Hardness results
  - Initializations
  - Analysis of Linear Neural nets
  - Hyper-Parameter Optimization
  - Optimization
  - Normalization Schemes

Grades

- 30%: 3-4 HW Tasks
  - Mainly proofs and derivations
  - Submit in pairs
- 70%: Seminar:
  - ~25 minutes Presentation/Chalk talk
  - ~5 pages written report, combined with other students to create "lesson"

Questions?

Thanks to: Ron Meir, Elad Hoffer, Nati Srebro for some source material