Analyses of Deep Learning

STATS385

Stanford University

D. Donoho, V. Papyan, Y. Zhong

Details about course

- Wed 3:00-4:20 PM in Bishop Auditorium
- Sept 25 - Dec 6 (10 Weeks)
- Website: http://stats385.github.io
- Instructors:
  - David Donoho: donoho@stanxxx.edu
  - Yiqiao Zhong: yiqiaoz@stanxxx.edu
  - Vardan Papyan: papyan@stanxxx.edu

Analyses of Deep Learning (STATS 385)

Stanford University, Fall 2019

Deep learning is a transformative technology that has delivered impressive improvements in image classification and speech recognition. Many researchers are trying to better understand how to improve prediction performance and also how to improve training methods. Some researchers use experimental techniques; others use theoretical approaches. In this course we will review both experimental and theoretical analyses of deep learning. We will have 8 guest lecturers as well as graded projects for those who take the course for credit.

Instructors and office hours:

David Donoho
Sequoia 128

Vardan Papyan
Sequoia 208

Yiqiao Zhong
Packard 239

Analyses of **Deep Learning** (STATS 385)

https://stats385.github.io/

Stanford University

List of speakers

October 2: Stefano Soatto

October 9: Tengyu Ma

October 16: Jeffrey Pennington

October 23: Song Mei

October 30: Arthur Jacot

November 6: Aleksander Madry

November 13: Nati Srebro

November 20: Andrew Saxe

December 6: Vardan Papyan

Why are we here?

- **Unprecedented success** of deepnets
- **Tremendous media** attention
- **Dramatic investments** in deepnet technology
- **Purely empirical** understanding
- **Perception:** massive stakes
- Please see slides **2017 Theories of Deep Learning**

Standard Notations in Deep Learning

Linear regression from a deep learning perspective

Linear regression:

$$\min_{\beta} \sum_i ||y_i - \beta^T x_i||_2^2$$

Logistic regression:

$$\min_{\beta} \sum_i \rho(\beta^T x_i y_i), \quad \rho(t) = -\ln(1+e^{-t})$$

Multinomial logistic regression (cross-entropy loss):

$$\min_{\beta} \sum_i \rho(\underbrace{\beta^T x_i}_{z \in \mathbb{R}^C}), \quad \rho(z) = -\sum_c y_c \log\left(\frac{\exp\{z_c\}}{\sum_{c'} \exp\{z_{c'}\}}\right)$$

General:

$$\min_{\beta} \mathrm{Loss}(f(x_i;\beta), y_i), \quad f(x_i;\beta) = \beta^T x_i$$

Linear regression from deep learning perspective

```python
import math
import torch
import torch.nn as nn
import torch.optim as optim

n = 100000
p = 512
X = torch.randn(n, p)
beta = torch.randn(p, 1) / math.sqrt(p)
y = torch.mm(X, beta) + 0.5 * torch.randn(n, 1)

class LinearReg(nn.Module):
    def __init__(self):
        super(LinearReg, self).__init__()
        self.linear = torch.nn.Linear(p, 1)
    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
```

Stochastic gradient descent (SGD)

$$\beta_{t+1} = \beta_t - \eta_t \sum_{i \in \mathcal{B}} \nabla_\beta \operatorname{Loss}_i(f(x_i; \beta_t), y_i)$$

- Batch size
- Epoch
- Learning rate = step size
- Learning rate scheduler
- Variations:
    - Weight decay = ridge regularization
    - Momentum
- Beyond SGD:
    - RMSProp, Adagrad, Adam

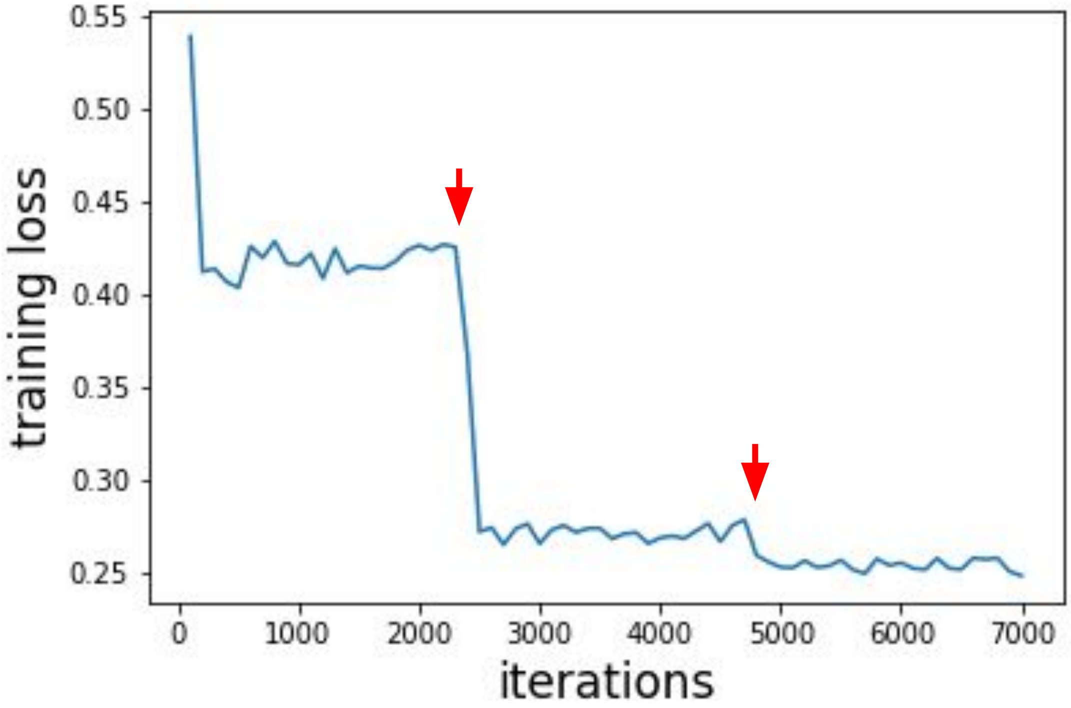
Optimizing linear regression using SGD

```
from torch.optim.lr_scheduler import MultiStepLR
batch_sz = 128
epochs = 9
model = LinearReg()
optimizer = optim.SGD(model.parameters(), lr=0.01,
                      momentum=0.9, weight_decay=5e-4)
scheduler = MultiStepLR(optimizer, milestones=[3,6], gamma=0.2)
Loss = nn.MSELoss()

for epoch in range(epochs):
    scheduler.step()
    for idx in range(n // batch_sz):
        min_idx = batch_sz*idx
        max_idx = batch_sz*idx+batch_sz
        x_batch, y_batch = X[min_idx:max_idx,:], y[min_idx:max_idx]

        optimizer.zero_grad()
        Loss(model(x_batch), y_batch).backward()
        optimizer.step()
```

AUTOGRAD!

Optimizing linear regression using SGD

Learning rate initialized to 0.01 and decreased by factor of 0.2 at ⅓ and ⅔ of iterations



From linear regression to feedforward fully connected neural network

Regression:

$$f(x_i; \beta) = \beta^T x_i$$

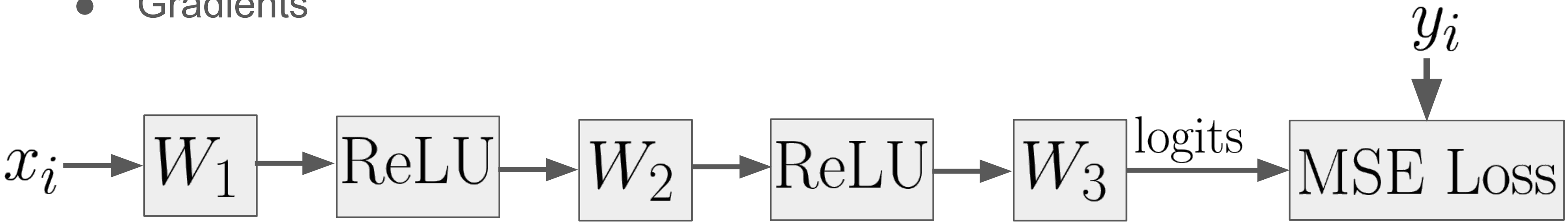
Fully connected feedforward neural network:

$$f(x_i; \beta) = W_3\, \sigma(W_2\, \sigma(W_1 x_i)) \qquad \sigma(x) = \max(x, 0)$$

A cascade of linear and non-linear operators.

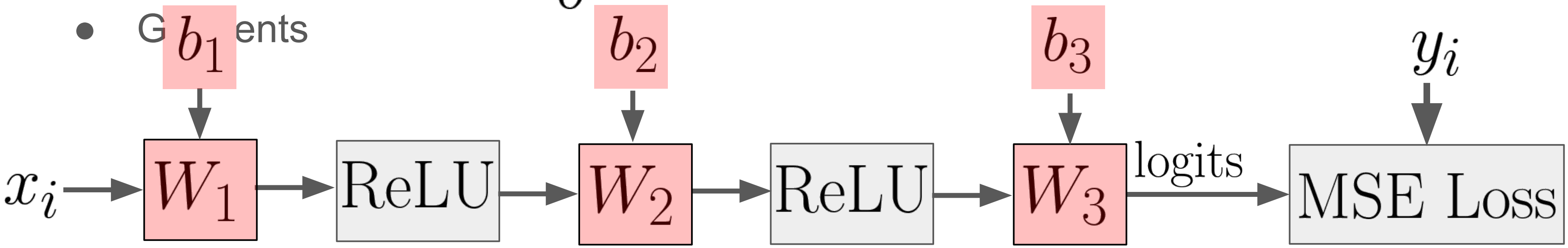
Terminology (use these words & symbols)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients



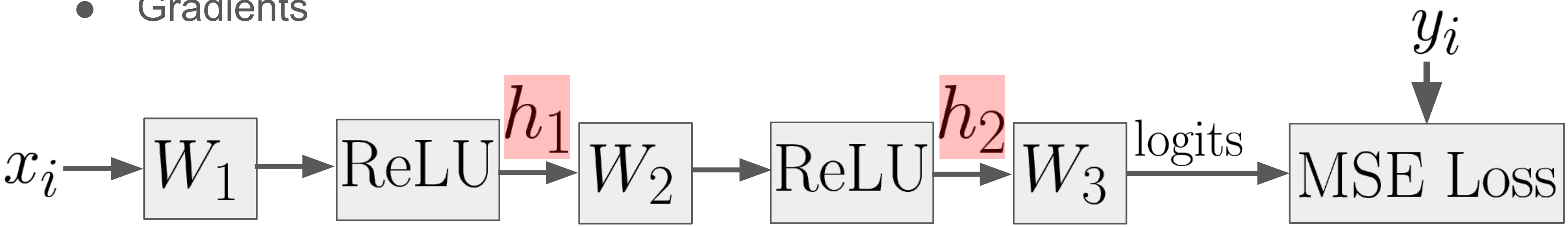
Terminology (use these words & symbols)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- G ents



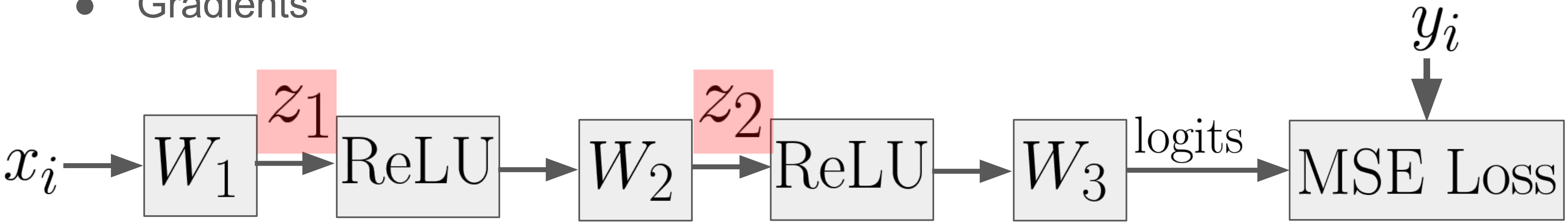
Terminology (use these words & symbols)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients



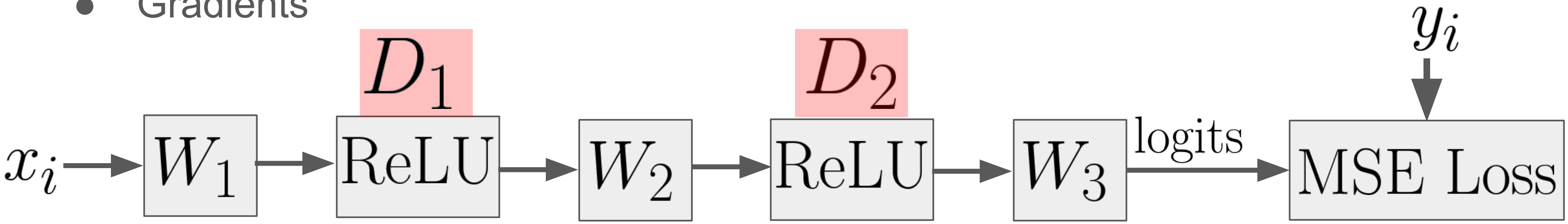
Terminology (use these words & symbols)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients



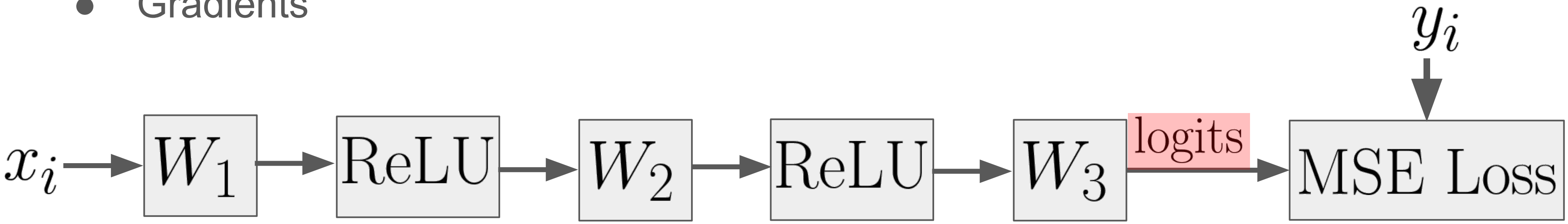
Terminology (use these words & symbols)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients



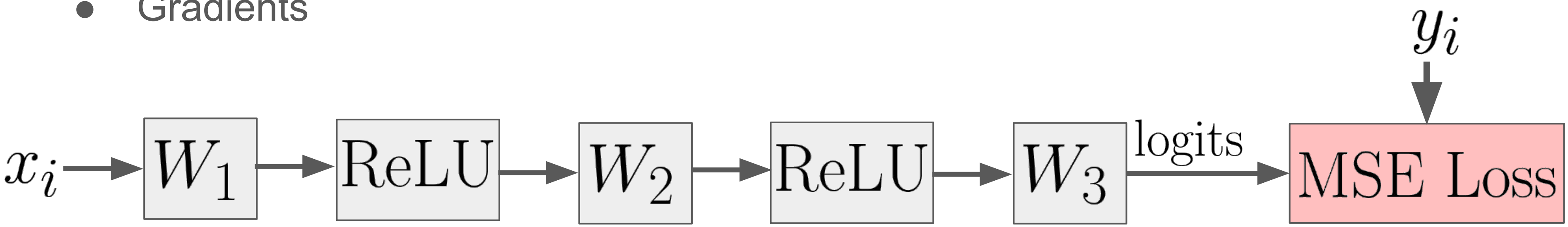
Terminology (use these words)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients



Terminology (use these words)

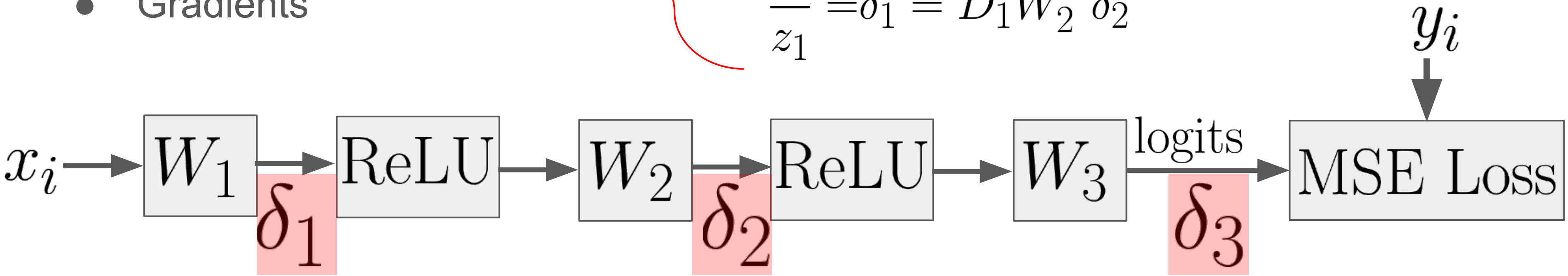
- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta) = \min \| W_3\, \sigma(W_2\, \sigma(W_1 x_i)) - y_i \|_2^2$
- Backpropagated errors: δ
- Gradients



Terminology (use these words)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients

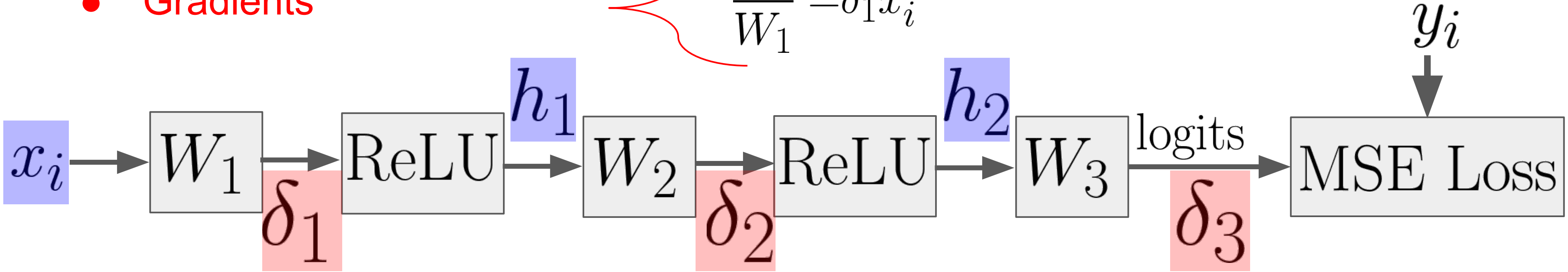
$$\delta \left\{ \begin{aligned} \frac{\mathcal{L}}{\text{logits}} &= \delta_3 = \text{logits} - y_i \\ \frac{\mathcal{L}}{z_2} &= \delta_2 = D_2 W_3^T \delta_3 \\ \frac{\mathcal{L}}{z_1} &= \delta_1 = D_1 W_2^T \delta_2 \end{aligned} \right.$$



Terminology (use these words)

- Weights: W Biases: b
- Features (post-activations): h
- Pre-activations: z
- Masks: D
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: δ
- Gradients

$$\left\{\begin{aligned} \frac{\mathcal{L}}{W_3} &= \delta_3 h_2^T \\ \frac{\mathcal{L}}{W_2} &= \delta_2 h_1^T \\ \frac{\mathcal{L}}{W_1} &= \delta_1 x_i^T \end{aligned}\right.$$

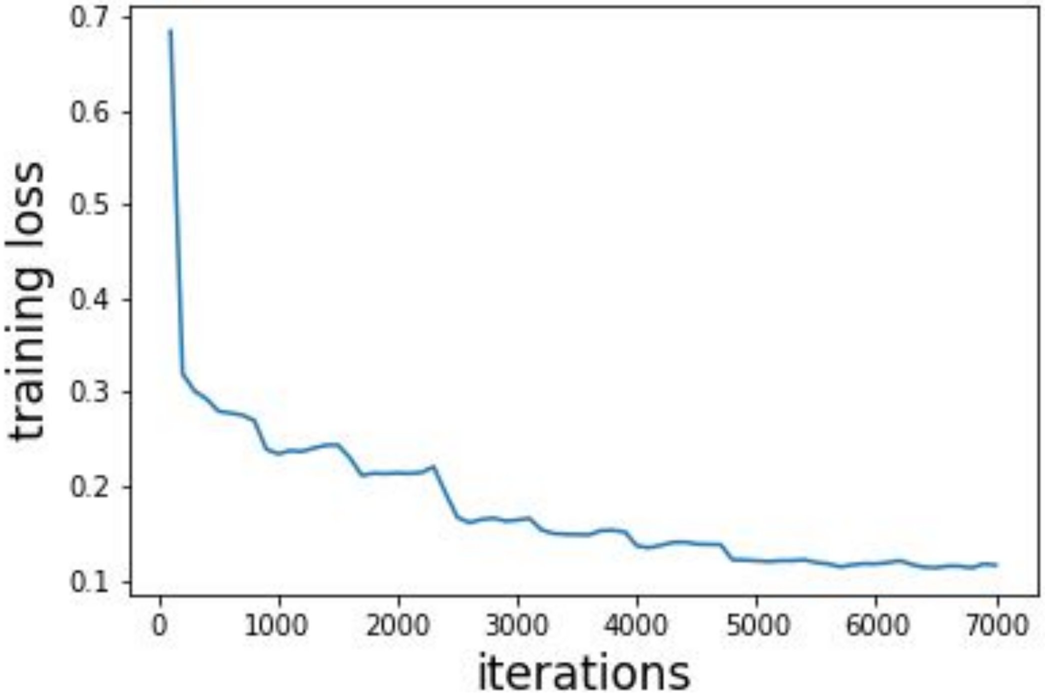


Feedforward fully connected network in PyTorch

```
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(p, p)
        self.fc2 = nn.Linear(p, p)
        self.fc3 = nn.Linear(p, 1)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

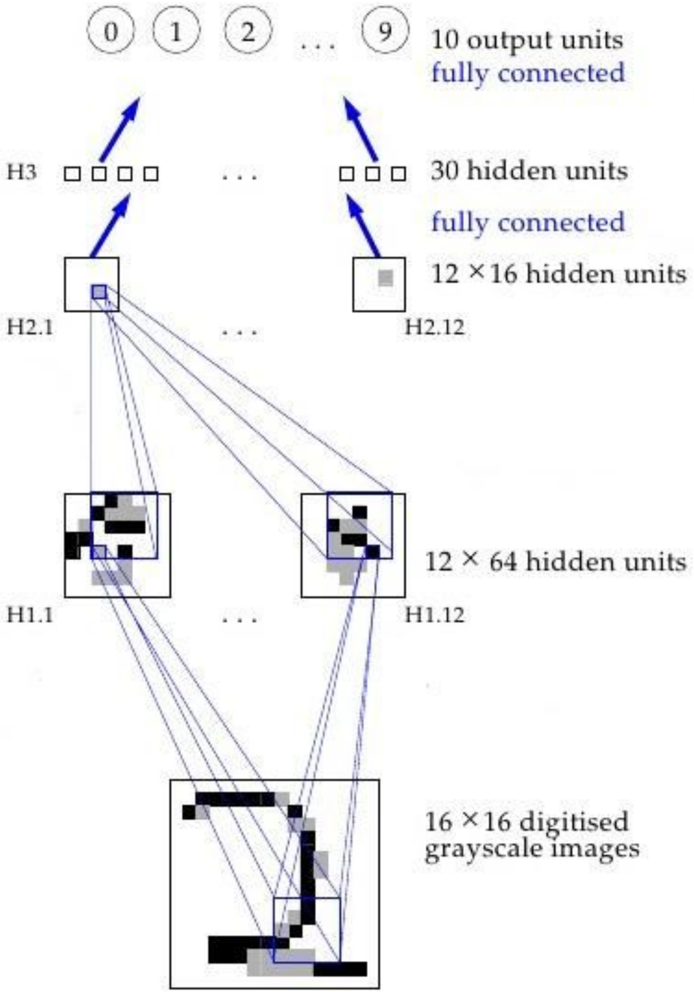
Linear regression trained with a neural net

Learning rate initialized to 0.01 and decreased by factor of 0.2 at ⅓ and ⅔ of iterations



LeNet: first success of CNNs

- Fully connected layers are not enough for computer vision
- Backpropagation applied to handwritten ZIP code recognition (1989)
- Convolutional layers
- Pooling layers

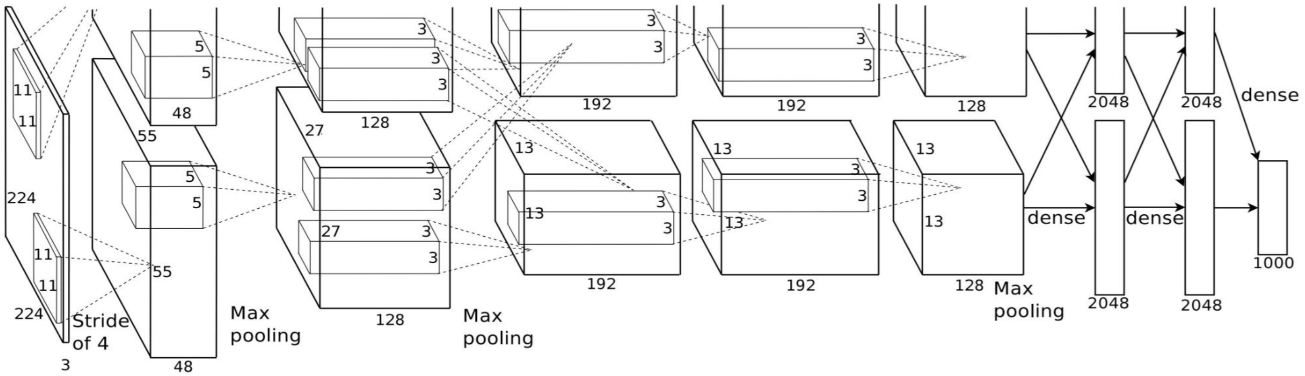


ImageNet

- Created by Fei-Fei Li
- Crowdsourced annotations
- More than 20,000 classes
- Image size: variable-resolution, often 224 × 224 × 3 after cropping
- ILSVRC competition: Subset of 1000 classes from ImageNet; training set contains 1.2 million images.

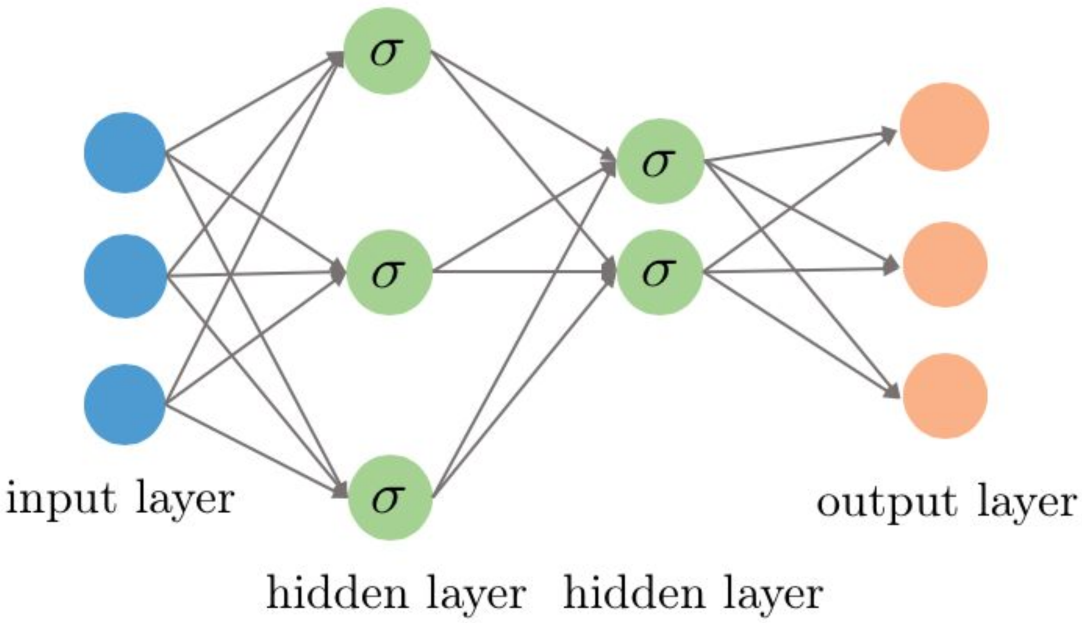
AlexNet

- First use of ReLU
- Dropout 0.5 (explained later)
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when validation accuracy plateaus
- 5e-4 weight decay



Remedy to optimization/generalization problems

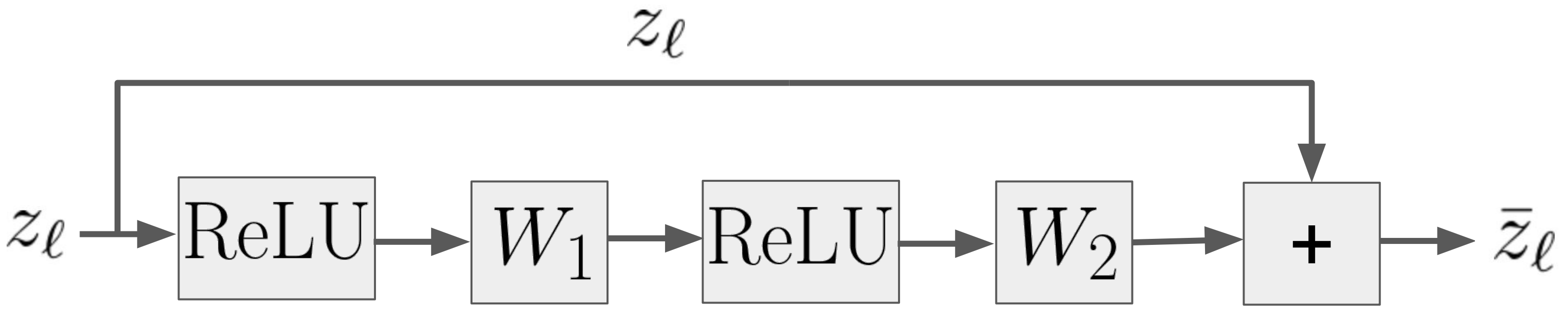
- Dropout
- Skip connections
- Batch normalization



During training, randomly zeros/removes a fraction of nodes for each iteration

Remedy to optimization/generalization problems

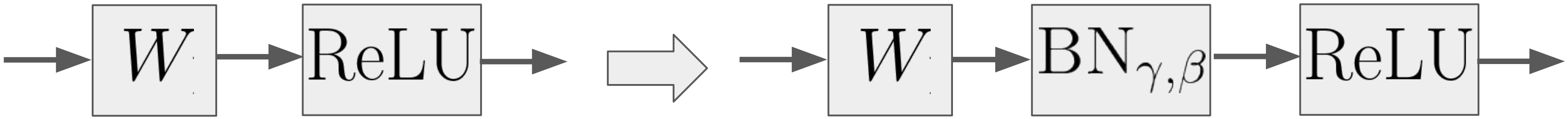
- Dropout
- Skip connections
- Batch normalization



General form: $\bar{z}_\ell = z_\ell + \mathcal{F}(z_\ell, \theta_\ell)$

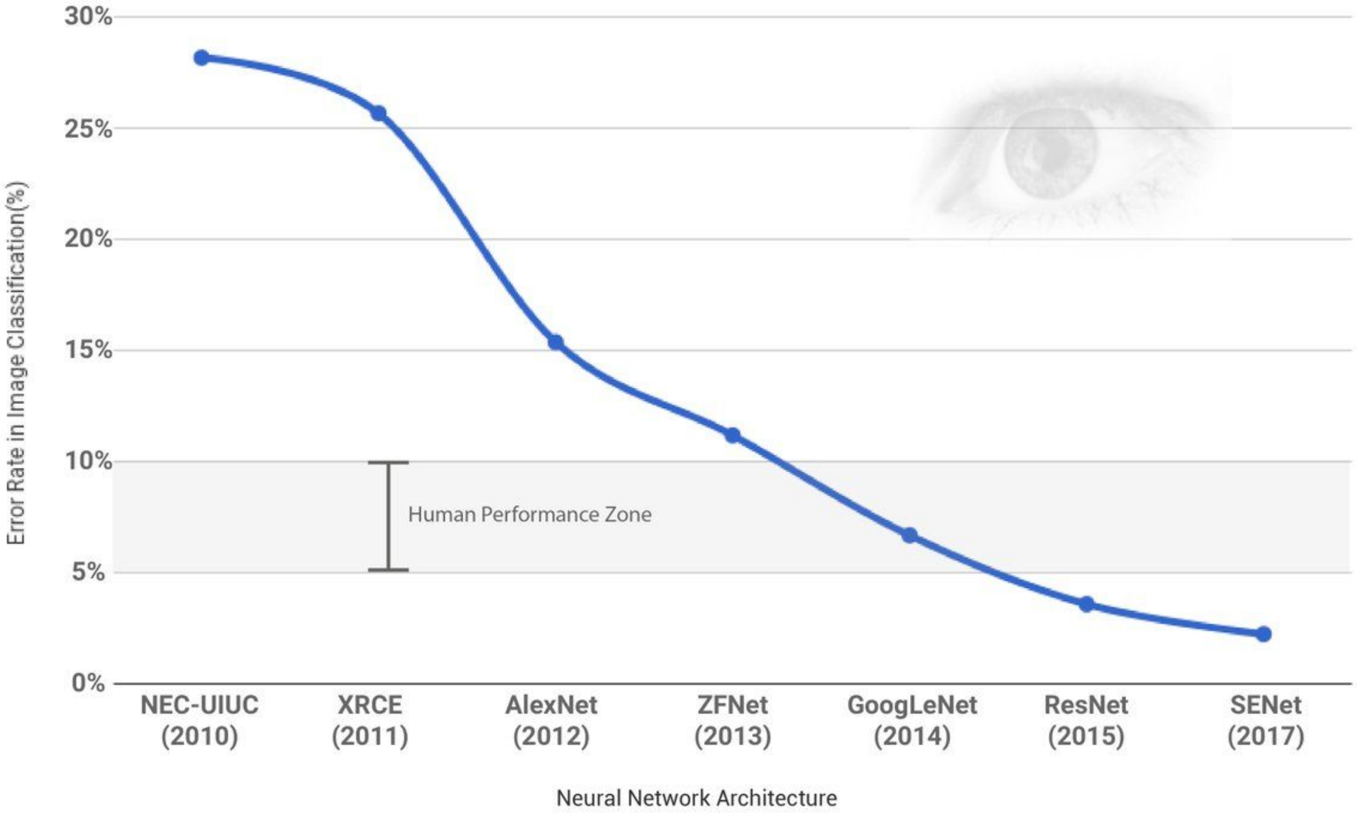
Remedy to optimization/generalization problems

- Dropout
- Skip connections
- Batch normalization



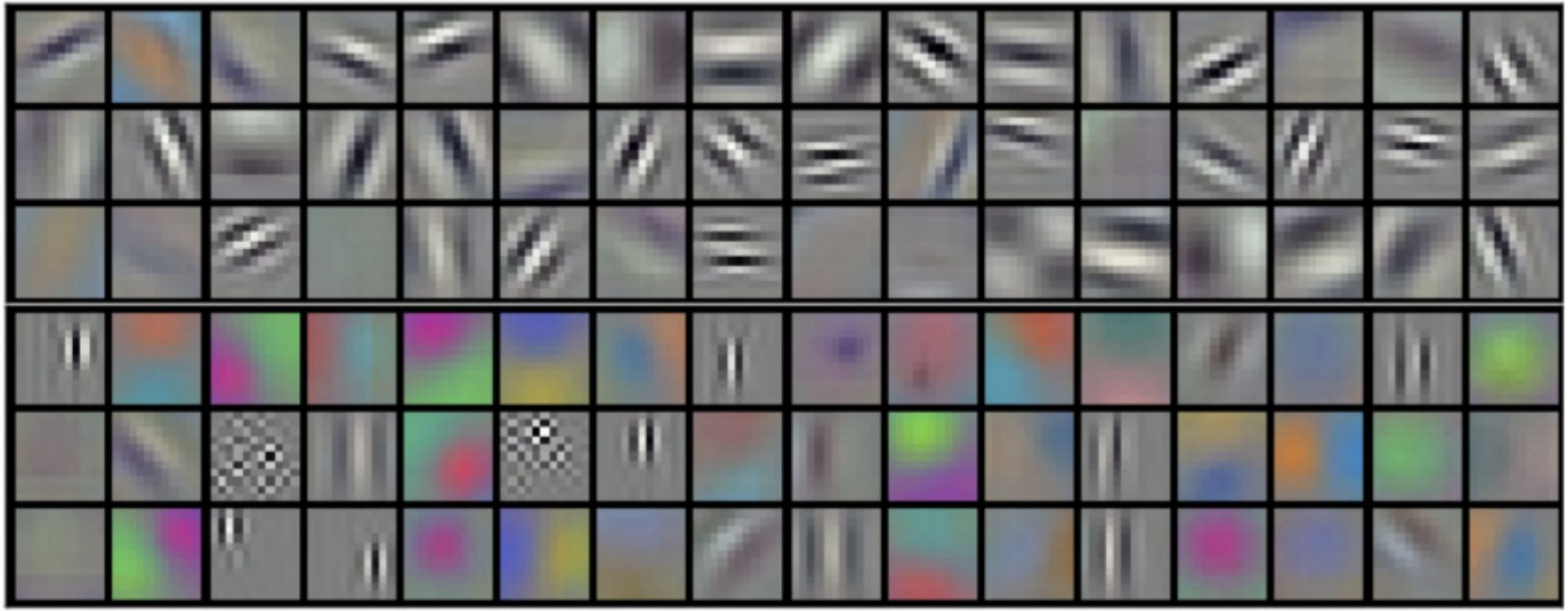
$$\mathrm{BN}_{\gamma,\beta}(x) = \gamma\hat{x} + \beta, \text{ where } \hat{x} = \frac{x - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

ILSVRC competition

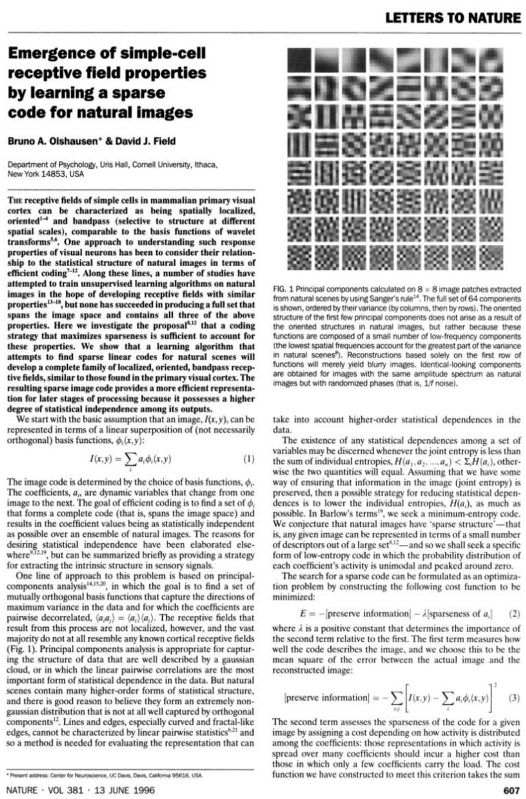


Intriguing **phenomena** in deep learning

First layer weights of AlexNet



Same filters obtained from sparse coding



LETTERS TO NATURE

Emergence of simple-cell receptive field properties by learning a sparse code for natural images

Bruno A. Olshausen* & David J. Field

Department of Psychology, Uris Hall, Cornell University, Ithaca, New York 14853, USA

The receptive fields of simple cells in mammalian primary visual cortex can be characterized as being spatially localized, oriented[1–4] and bandpass (selective to structure at different spatial scales), comparable to the basis functions of wavelet transforms[5,6]. One approach to understanding such response properties of visual neurons has been to consider their relationship to the statistical structure of natural images in terms of efficient coding[7–12]. Along these lines, a number of studies have attempted to train unsupervised learning algorithms on natural images in the hope of developing receptive fields with similar properties[13–18], but none has succeeded in producing a full set that spans the image space and contains all three of the above properties. Here we investigate the proposal[8,12] that a coding strategy that maximizes sparseness is sufficient to account for these properties. We show that a learning algorithm that attempts to find sparse linear codes for natural scenes will develop a complete family of localized, oriented, bandpass receptive fields, similar to those found in the primary visual cortex. The resulting sparse image code provides a more efficient representation for later stages of processing because it possesses a higher degree of statistical independence among its outputs.

We start with the basic assumption that an image, $I(x,y)$, can be represented in terms of a linear superposition of (not necessarily orthogonal) basis functions, $\phi_i(x,y)$:

$$I(x,y) = \sum_i a_i \phi_i(x,y) \qquad (1)$$

The image code is determined by the choice of basis functions, ϕ_i. The coefficients, a_i, are dynamic variables that change from one image to the next. The goal of efficient coding is to find a set of ϕ_i that forms a complete code (that is, spans the image space) and results in the coefficient values being as statistically independent as possible over an ensemble of natural images. The reasons for desiring statistical independence have been elaborated elsewhere[9,12,19], but can be summarized briefly as providing a strategy for extracting the intrinsic structure in sensory signals.

One line of approach to this problem is based on principal-components analysis[14,15,20], in which the goal is to find a set of mutually orthogonal basis functions that capture the directions of maximum variance in the data and for which the coefficients are pairwise decorrelated, $\langle a_i a_j \rangle = \langle a_i \rangle \langle a_j \rangle$. The receptive fields that result from this process are not localized, however, and the vast majority do not at all resemble any known cortical receptive fields (Fig. 1). Principal components analysis is appropriate for capturing the structure of data that are well described by a gaussian cloud, or in which the linear pairwise correlations are the most important form of statistical dependence in the data. But natural scenes contain many higher-order forms of statistical structure, and there is good reason to believe they form an extremely non-gaussian distribution that is not at all well captured by orthogonal components[12]. Lines and edges, especially curved and fractal-like edges, cannot be characterized by linear pairwise statistics[6,21] and so a method is needed for evaluating the representation that can take into account higher-order statistical dependences in the data.

FIG. 1 Principal components calculated on 8 × 8 image patches extracted from natural scenes by using Sanger's rule[14]. The full set of 64 components is shown, ordered by their variance (by columns, then by rows). The oriented structure of the first few principal components does not arise as a result of the oriented structures in natural images, but rather because these functions are composed of a small number of low-frequency components (the lowest spatial frequencies account for the greatest part of the variance in natural scenes[8]). Reconstructions based solely on the first row of functions will merely yield blurry images. Identical-looking components are obtained for images with the same amplitude spectrum as natural images but with randomized phases (that is, 1/*f* noise).

The existence of any statistical dependences among a set of variables may be discerned whenever the joint entropy is less than the sum of individual entropies, $H(a_1, a_2, \ldots, a_n) < \Sigma_i H(a_i)$, otherwise the two quantities will equal. Assuming that we have some way of ensuring that information in the image (joint entropy) is preserved, then a possible strategy for reducing statistical dependences is to lower the individual entropies, $H(a_i)$, as much as possible. In Barlow's terms[19], we seek a minimum-entropy code. We conjecture that natural images have 'sparse structure'—that is, any given image can be represented in terms of a small number of descriptors out of a large set[8,12]—and so we shall seek a specific form of low-entropy code in which the probability distribution of each coefficient's activity is unimodal and peaked around zero.

The search for a sparse code can be formulated as an optimization problem by constructing the following cost function to be minimized:

$$E = -[\text{preserve information}] - \lambda[\text{sparseness of } a_i] \qquad (2)$$

where λ is a positive constant that determines the importance of the second term relative to the first. The first term measures how well the code describes the image, and we choose this to be the mean square of the error between the actual image and the reconstructed image:

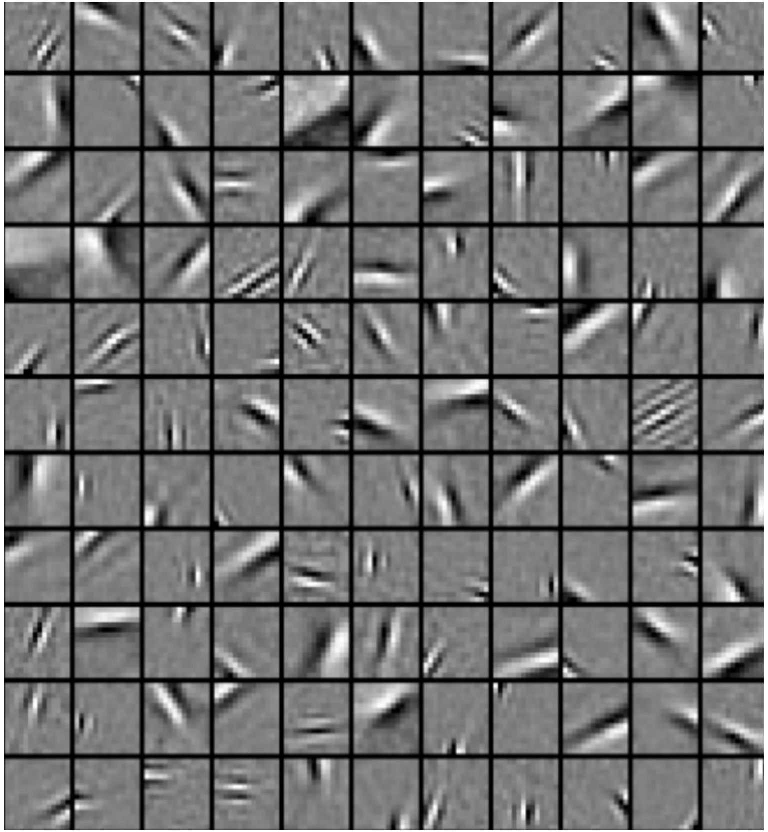
$$[\text{preserve information}] = -\sum_{x,y} \left[I(x,y) - \sum_i a_i \phi_i(x,y) \right]^2 \qquad (3)$$

The second term assesses the sparseness of the code for a given image by assigning a cost depending on how activity is distributed among the coefficients: those representations in which activity is spread over many coefficients should incur a higher cost than those in which only a few coefficients carry the load. The cost function we have constructed to meet this criterion takes the sum

* Present address: Center for Neuroscience, UC Davis, Davis, California 95616, USA.

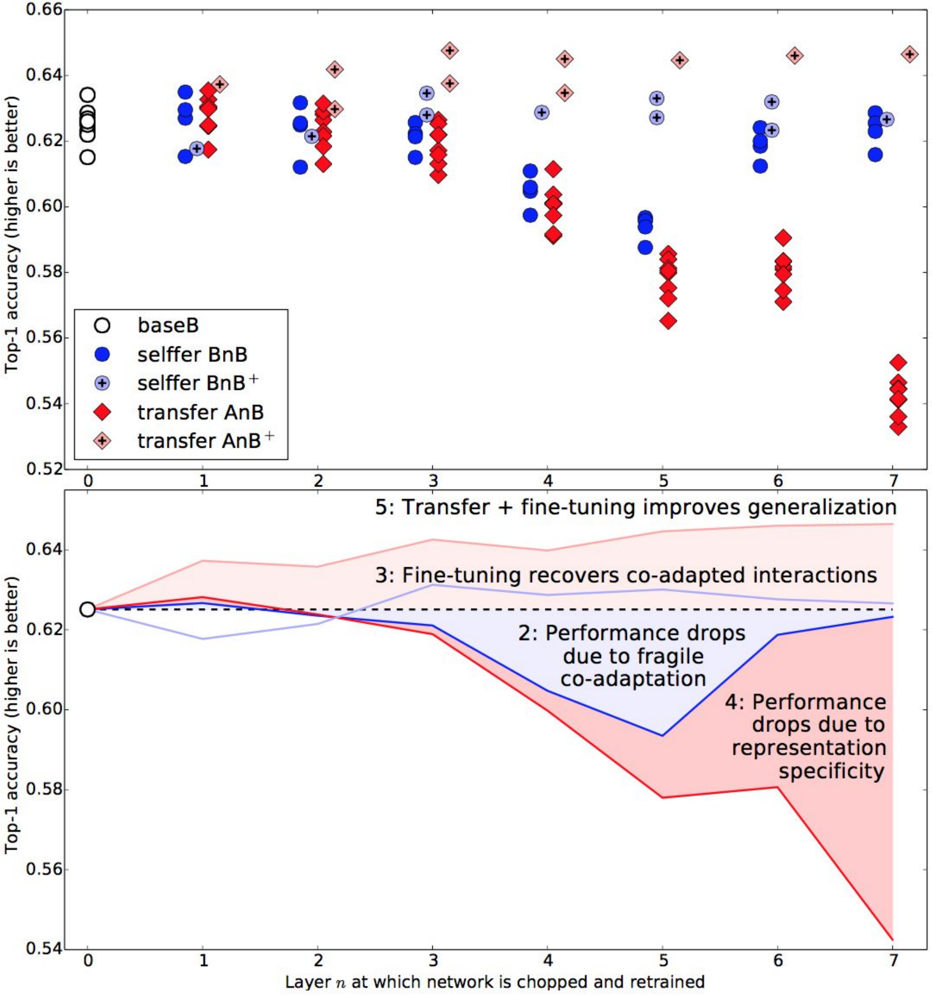
NATURE · VOL 381 · 13 JUNE 1996

607



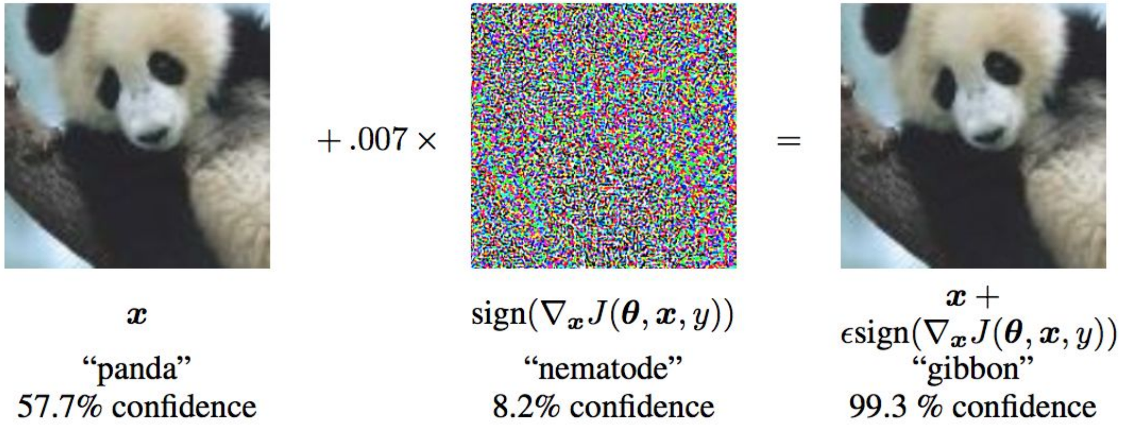
Transfer learning

- Low layer features learned on one dataset can be *transferred* to another similar dataset.
- Yosinski et al., *How transferable are features in deep neural networks?*, NeurIPS, 2014.



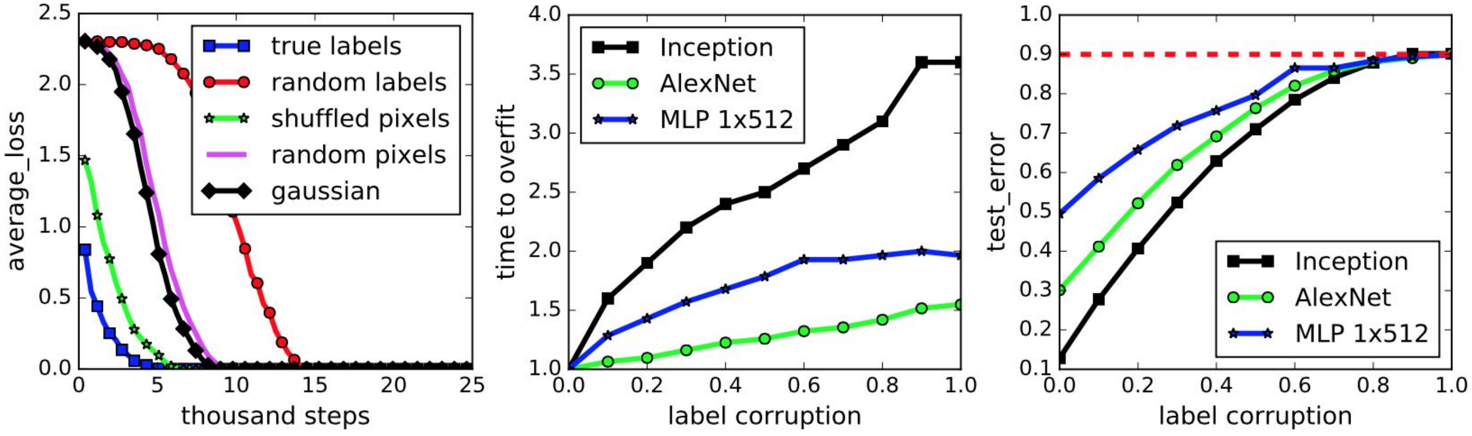
Adversarial examples (Aleksander Madry, Nov. 6th)

- Adding small perturbations can change drastically prediction results
- Szegedy et al., *Intriguing properties of neural networks*, 2013.
- Goodfellow et al., *Explaining and harnessing adversarial examples*, ICLR, 2015.



Interpolation regime

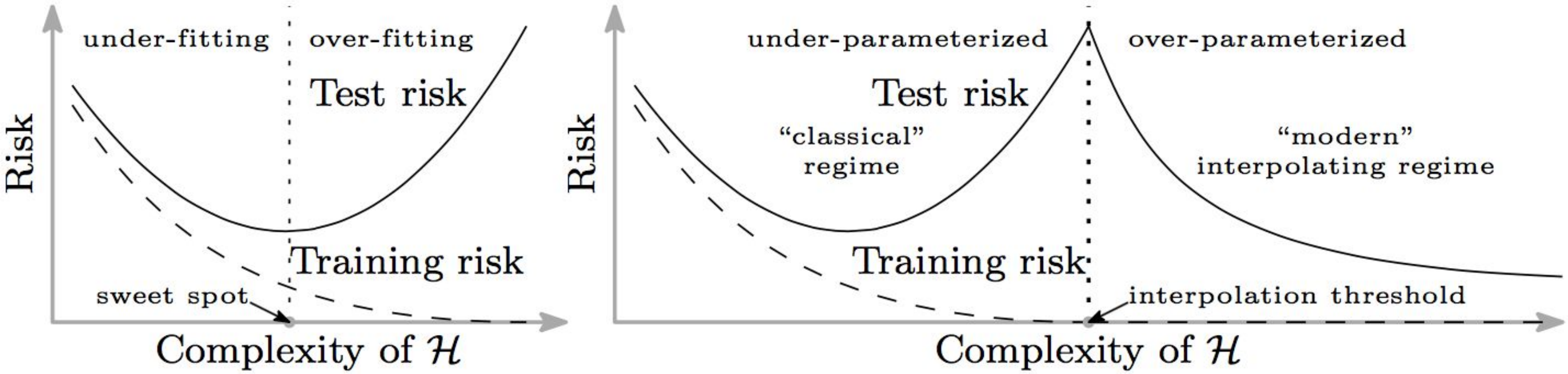
- Overparameterization can memorize & generalize.
- Zhang et al., *Understanding deep learning requires rethinking generalization*, ICLR, 2017.



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

Double descent curve (Song Mei, Oct. 23rd)

- Under overparameterization, new phenomenon beyond “bias-variance” tradeoff.
- Belkin et al., 2018.
- Hastie et al., 2019
- Montanari and Mei, 2019.

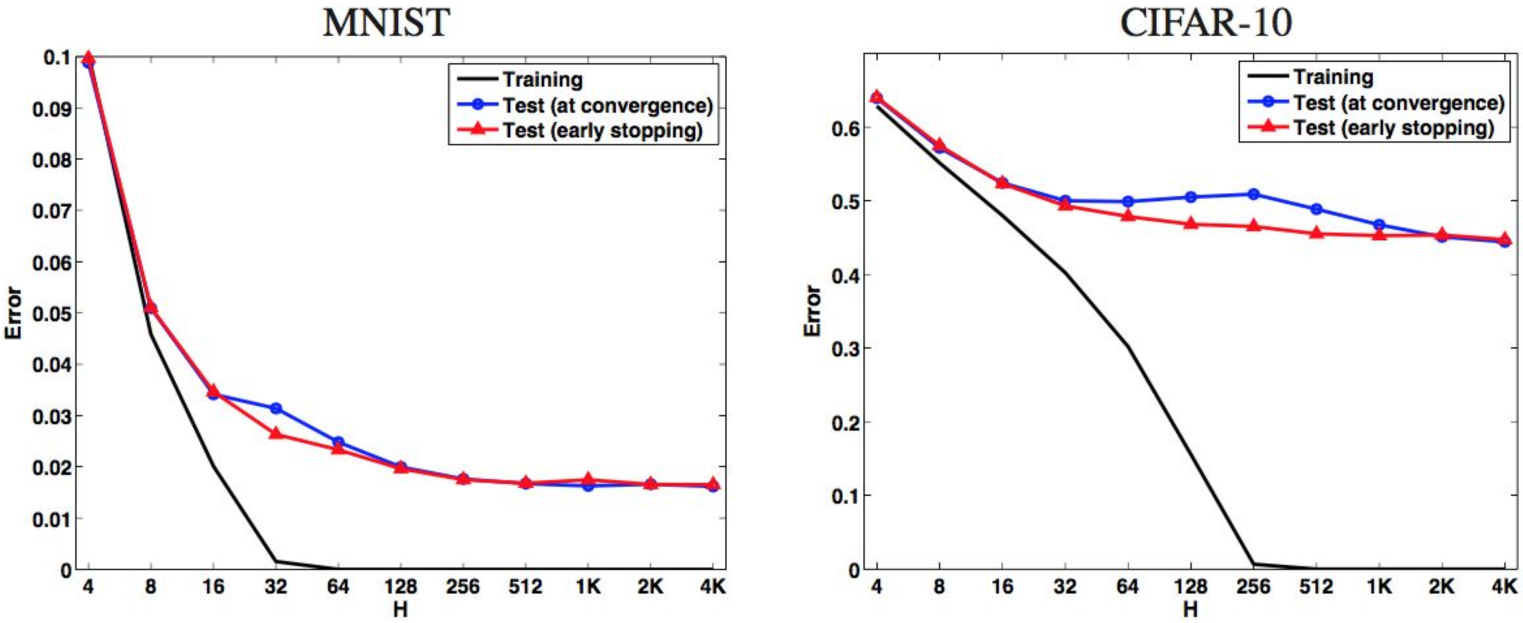


(a) U-shaped “bias-variance” risk curve

(b) “double descent” risk curve

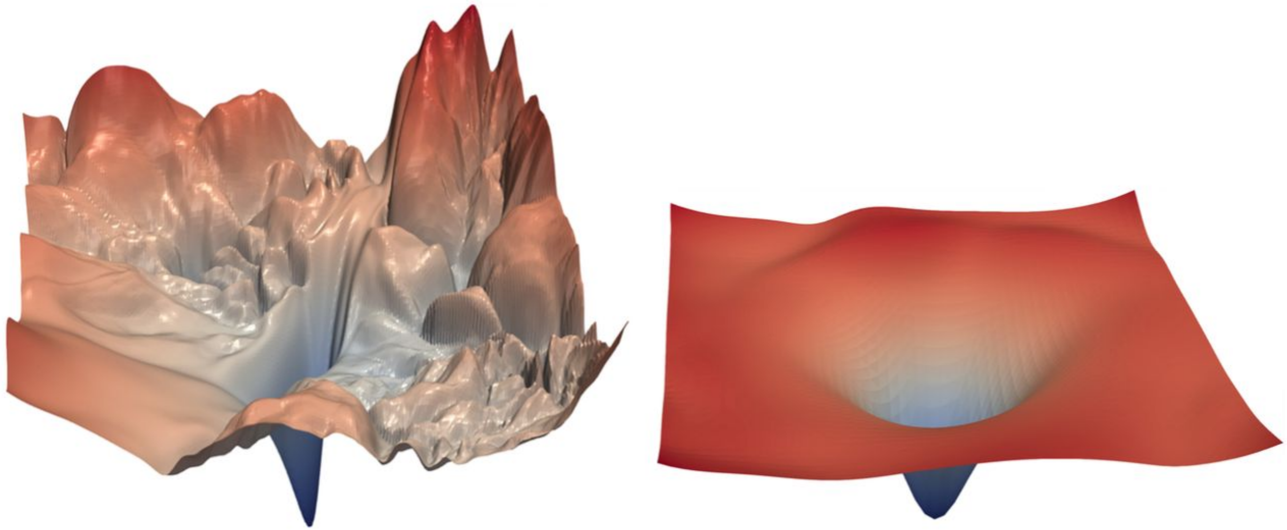
Implicit bias (Srebro, Nov. 13th, Tengyu Ma Oct. 9th)

- Neyshabur et al., In search of the real inductive bias, ICLR, 2015.



Optimization landscape

- Choromanska et al., *The loss surfaces of multilayer networks*, AISTATS, 2015.
- Li et al., *Visualizing the loss landscape of neural nets*, NeuIPS, 2018.

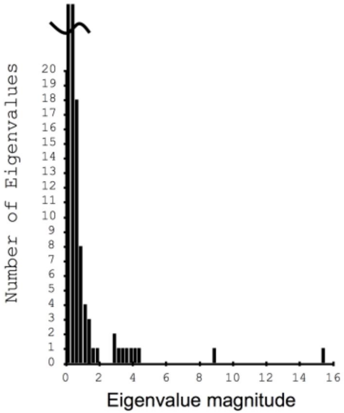


(a) without skip connections　　(b) with skip connections

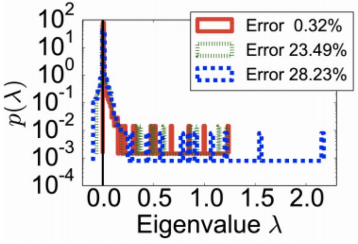
Deepnet spectra (Vardan Papyan Dec. 6)

- **Hessian:** LeCun et al. (1998), Dauphin et al. (2014), Sagun et al. (2016,2017), Papyan (2019), Ghorbani (2019), Li et al. (2019), Granziol et al. (2019), Pfahler and Morik (2019), Alain et al. (2019)
- **Weights:** Martin and Mahoney (2018)
- **Fisher information matrix:** Papyan (2019), Li et al. (2019)
- **Gradients:** Gur-Ari et al. (2018)
- **Features:** Verma et al. (2019)
- **Backpropagated errors:** Oymak et al. (2019)

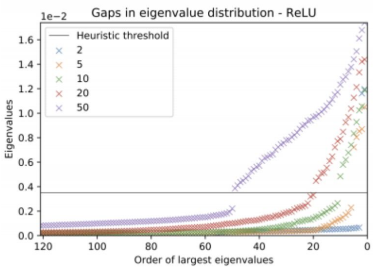
Deepnet spectra: Hessian



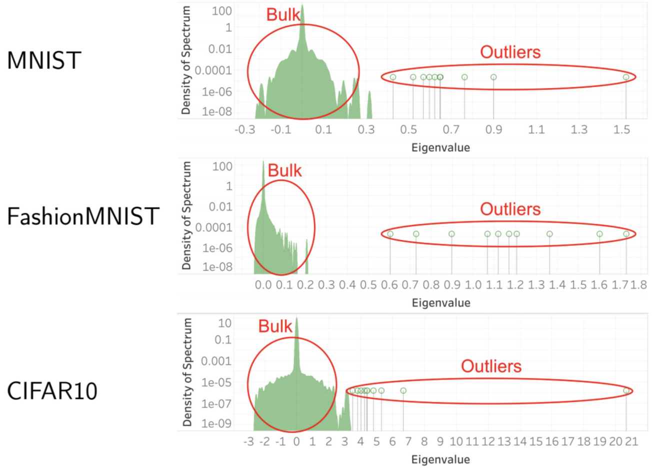
LeCun et al. (1998)



Dauphin et al. (2014)
MNIST downsampled to 10×10 pixels



Sagun et al. (2017)
100 dimensional input, two hidden layers, 30 hidden units each



Papyan (2019)
VGG11, 28 million parameters

Towards understanding phenomena

Generalization (Srebro, Nov. 13th, Tengyu Ma Oct. 9th)

- Generalization error bounds based on different complexity measures.
- Uniform control over a function class, no generative model.
- Bartlett et al., 2017.
- Neyshabur et al., 2018.
- Arora et al., 2018.
- Wei and Ma, 2019.

$$\mathcal{R}_S(\mathcal{F}) = \frac{1}{n}\mathbb{E}_{\xi\sim\{\pm 1\}^n}\left[\sup_{f\in\mathcal{F}}\sum_{i=1}^{n}\xi_i f(x_i)\right]$$

Kernels (Arthur Jacot, Oct. 30,Jeffrey Pennington, Oct. 16)

- Under certain limits, training and inference is characterized by kernels.
- Jacot, el al., *Neural tangent kernel: Convergence and generalization in neural networks*, NeurIPS, 2018.
- Lee et al., *Deep neural networks as Gaussian processes*, ICLR, 2018.
- Ghorbani et al., *Limitations of Lazy Training of Two-layers Neural Networks*, 2019.

First-order expansion:

$$f(x;\theta) \approx f(x;\theta_0) + \langle \theta - \theta_0, \nabla_\theta f(x;\theta_0) \rangle$$

Induced Kernel:

$$K(x,x') = \langle \nabla_\theta f(x;\theta_0), \nabla_\theta f(x';\theta_0) \rangle$$

Mean-field perspective (Song Mei, Oct. 23rd)

- Understanding training dynamics (SGD trajectory)
- Mei et al., *A mean field view of the landscape of two-layer neural networks*, PNAS, 2018.

$$\hat{y}(x;\theta) = \frac{1}{N}\sum_{i=1}^{N}\sigma_*(x;\theta) \xrightarrow{N\to\infty} \int \sigma_*(x;\theta)\,\rho(d\theta)$$

SGD Dynamics of θ Gradient Flow of ρ

Summary

- Summary of basic concepts.
- Intriguing phenomena:
  - First layer filters.
  - Transfer learning.
  - Adversarial examples.
  - Interpolation regime.
  - Double descent curve.
  - Implicit bias.
  - Optimization landscape.
  - Structure in deepnet spectra.
- Attempts at understanding phenomena:
  - Generalization bounds.
  - Neural tangent kernels.
  - Mean-field approach.