ECE 18-898G: Special Topics in Signal Processing: Sparsity, Structure, and Inference

Neural Networks: A brief touch

Yuejie Chi

Department of Electrical and Computer Engineering

Carnegie Mellon University

Spring 2018

Outline

- introduction to deep learning
- perceptron model (a single neuron)
- 1-hidden-layer (2-layer) neural network

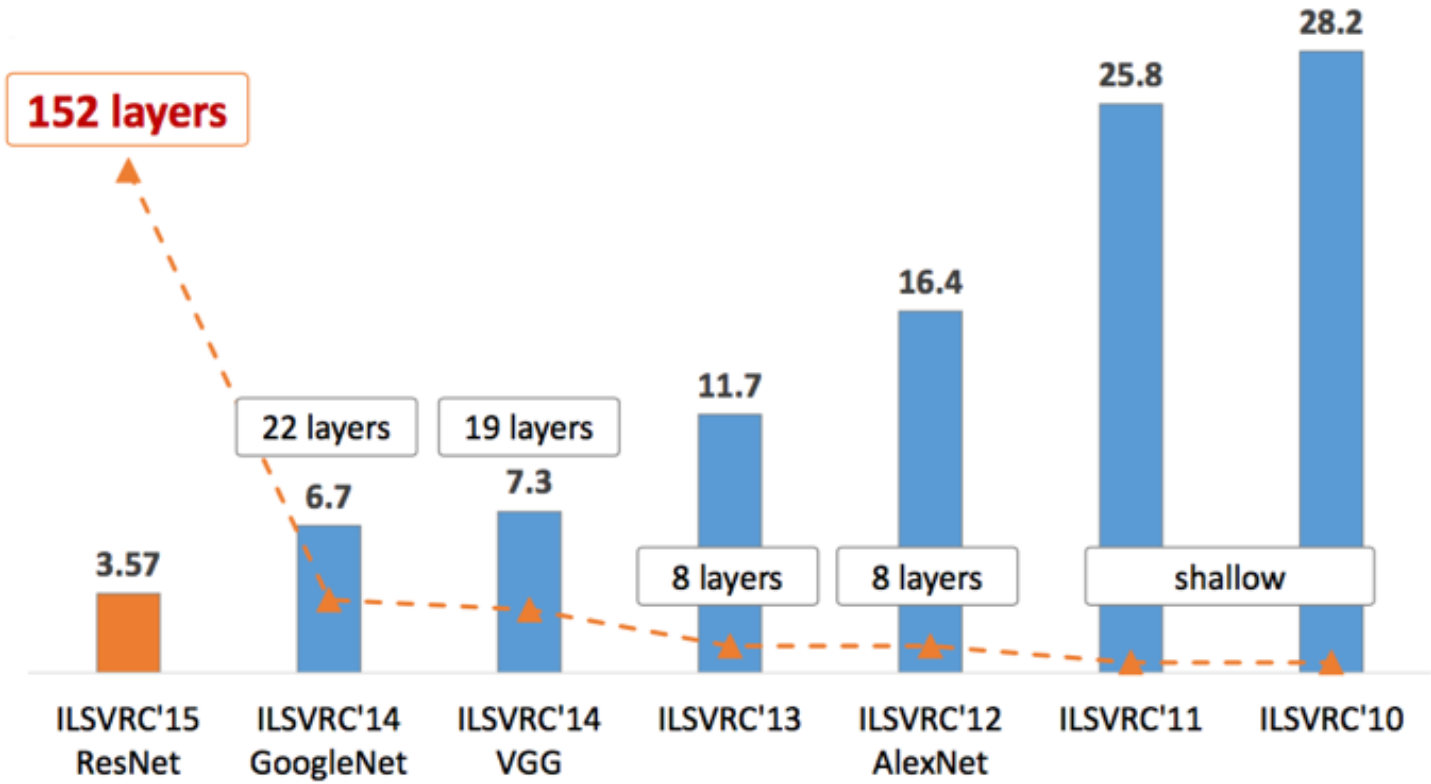
The rise of deep neural networks

ImageNet Large Scale Visual Recognition Challenge (ILSVRC): Led by Prof. Fei-Fei Li (Stanford).

Total number of non-empty synsets (categories): 21841;
Total number of images: 14,197,122

The rise of deep neural networks

The deeper, the better?



AlexNet

- Won the 2012 LSVRC competition by a large margin: top-1 and top-5 error rates of 37.5% and 17.0%.

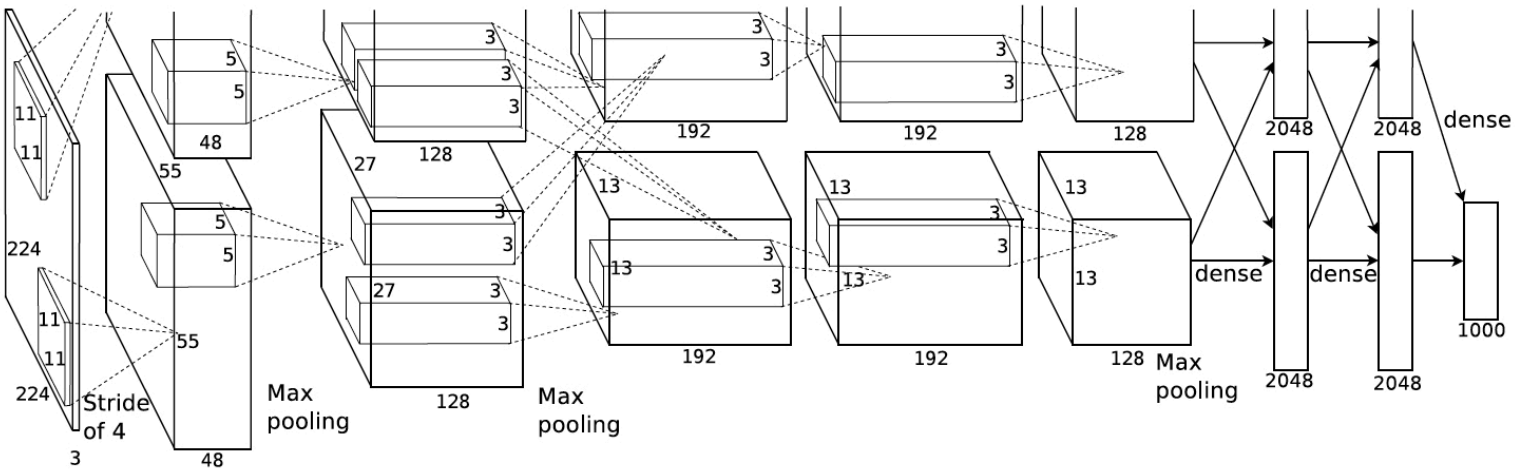
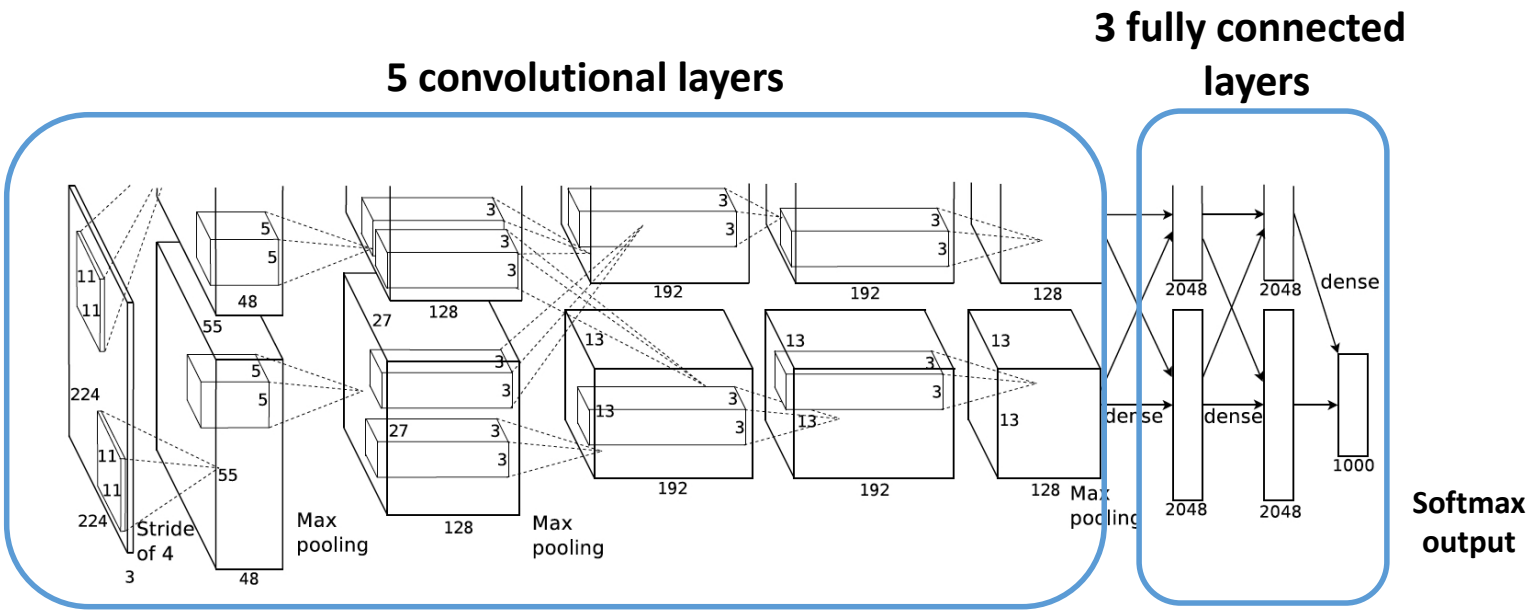


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

AlexNet

- 60 million parameters and 650,000 neurons.
- takes 5-6 days to train on two GTX 580 3GB GPUs.

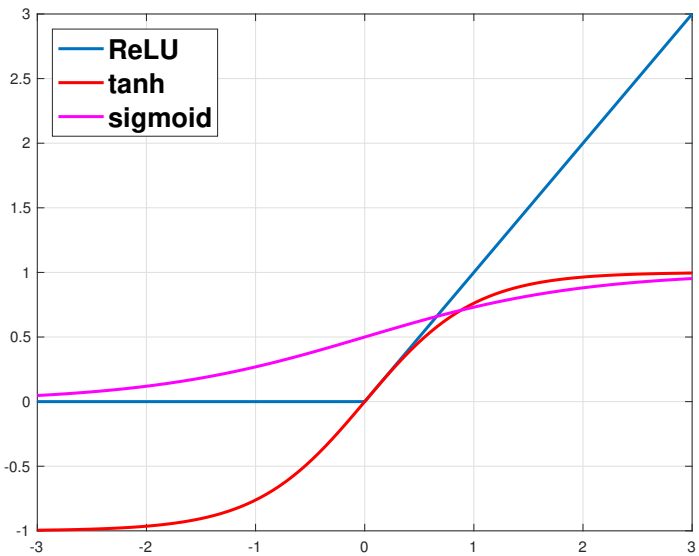


Faster training with ReLU

- Rectified linear units (ReLU):

$$y = \max(0, x)$$

- compared to $\tanh$ and sigmoid, training is much faster.



ReLu doesn't saturate.

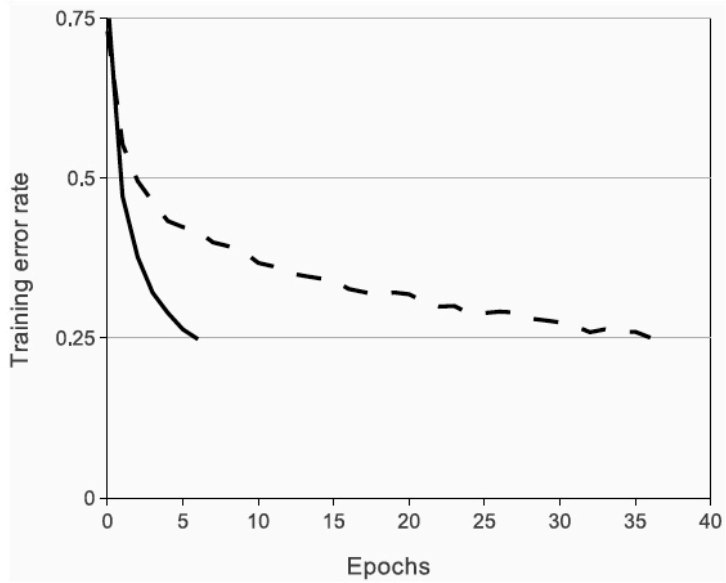


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with $\tanh$ neurons **(dashed line)**. The learning rates for each net-

Reduce overfitting

Important to reduce overfitting since:

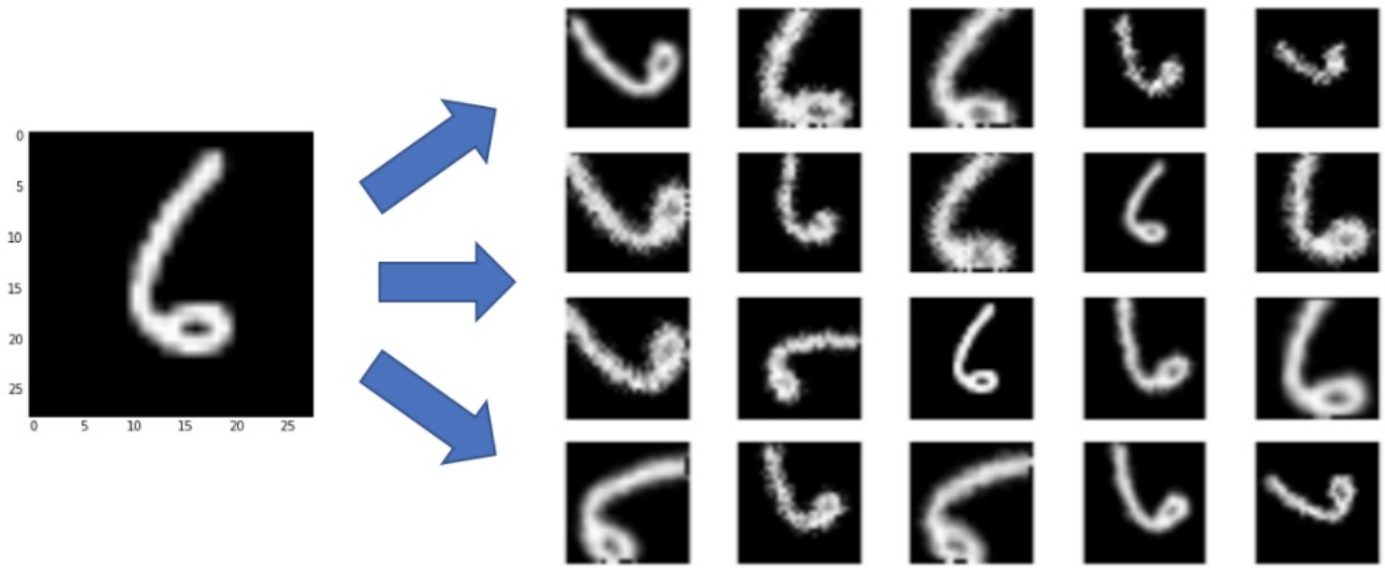
$$\text{number of training data} \ll \text{number of parameters}$$

Reduce overfitting

Important to reduce overfitting since:

$$\text{number of training data} \ll \text{number of parameters}$$

- Data augmentation: apply label-invariant transforms

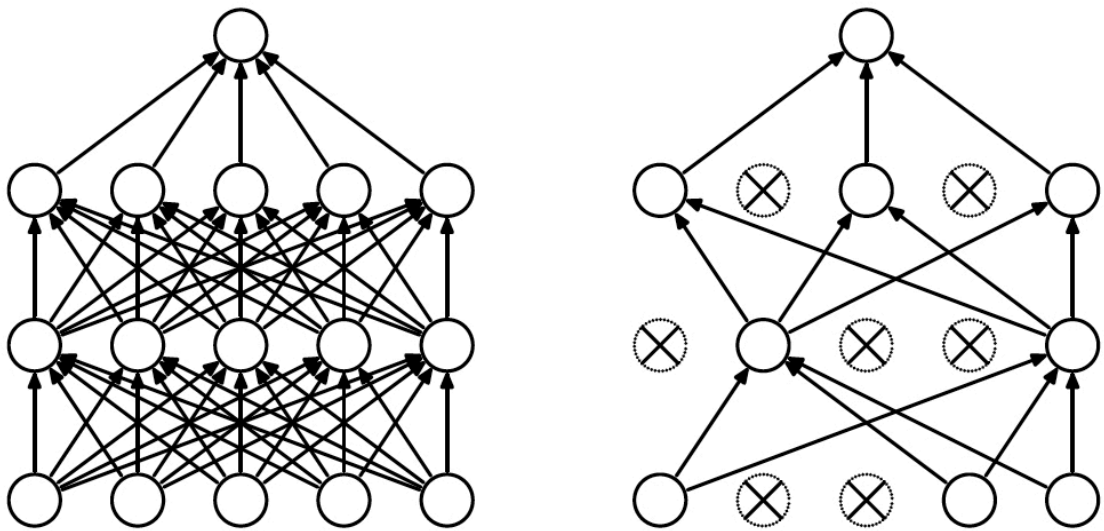


Reduce overfitting

Important to reduce overfitting since:

$$\text{number of training data} \ll \text{number of parameters}$$

- Dropout



(a) Standard Neural Net

(b) After applying dropout.

Reduce overfitting

Important to reduce overfitting since:

$$\text{number of training data} \ll \text{number of parameters}$$

- Other ways of "implicit regularization":
  - early stopping
  - weight decay (ridge regression)
  - ....

Learned hierarchical representations

Learned representations using CNN trained on ImageNet:

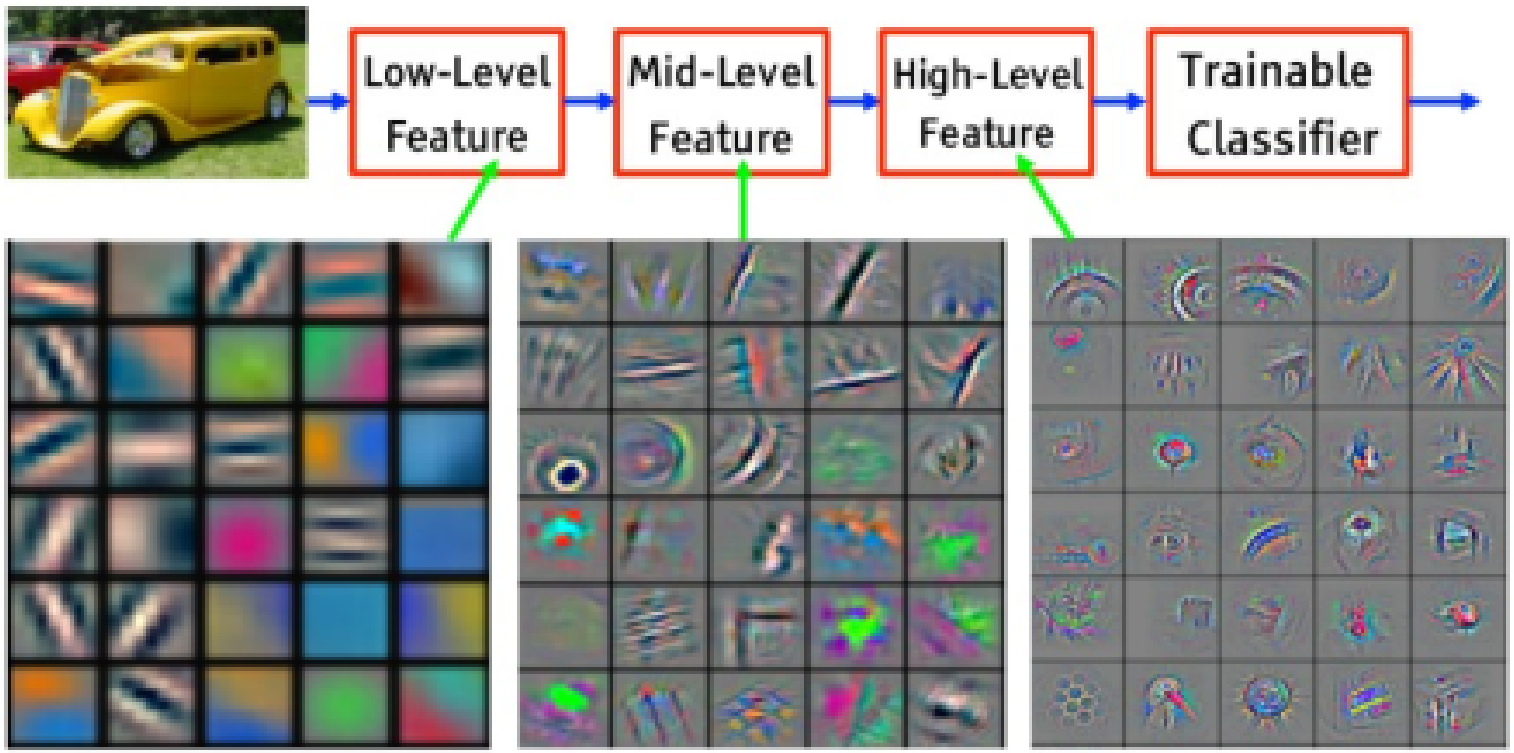


Figure credit: Y. Lecun's slide with research credit to, Zeiler and Fergus, 2013.

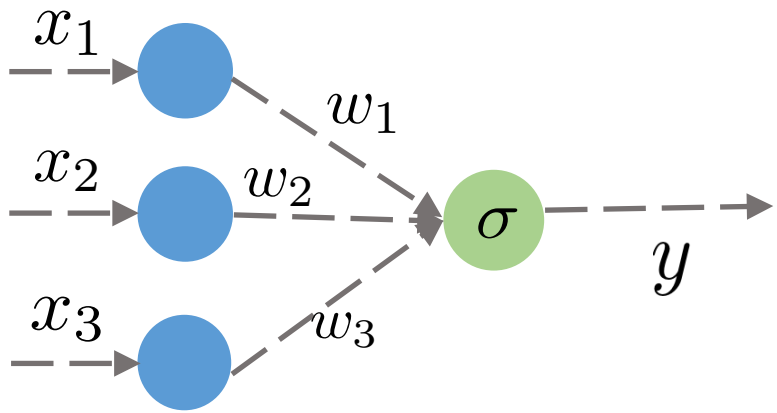
single-layer networks (perceptron)

Perceptron

Input $\boldsymbol{x} = [x_1, \ldots, x_d] \in \mathbb{R}^d$, weight $\boldsymbol{w} = [x_1, \ldots, x_d] \in \mathbb{R}^d$, output $y \in \mathbb{R}$;

$$y = \sigma\left(\boldsymbol{w}^\top \boldsymbol{x}\right) = \sigma\left(\sum_{i=1}^{d} w_i x_i\right)$$

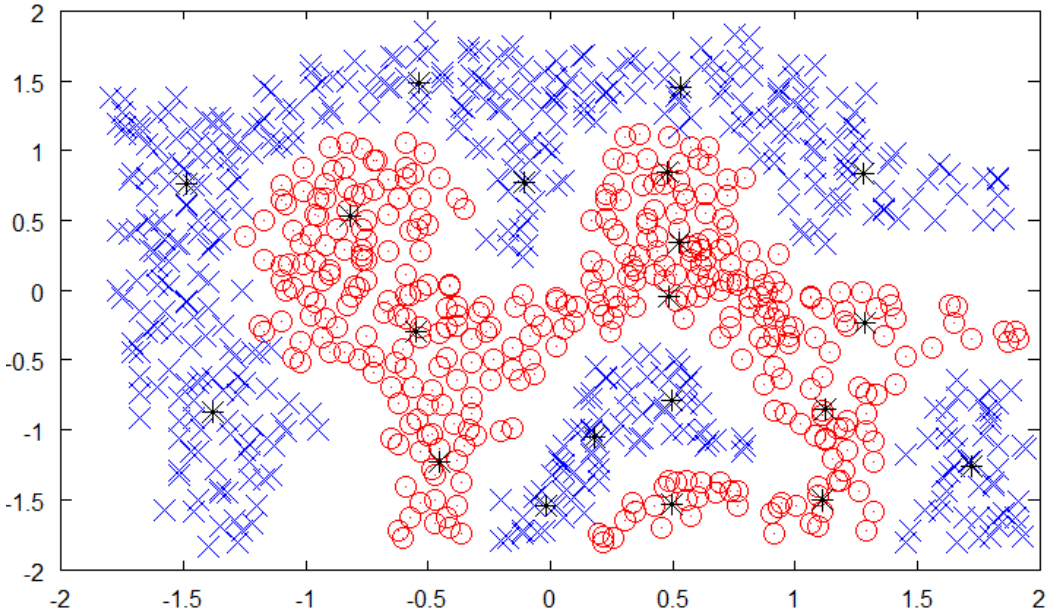
where $\sigma(\cdot)$ is a nonlinear activation function, e.g. $\sigma(z) = \text{sign}(z)$ (hard thresholding) or $\sigma(z) = \text{sigmoid}(z) = \frac{1}{1+e^{-z}}$ (soft thresholding).



Decision making at test stage: given a test sample $\boldsymbol{x}$, calculate y.

Nonlinear activation

Nonlinear activation is critical for complex decision boundary.



Training of a perceptron

Empirical risk minimization: Given training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, find the weight vector $\boldsymbol{w}$:

$$\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

- find the weight parameter $\boldsymbol{w}$ that best fits the data;
- popular choice for loss function: quadratic, cross entropy, hinge, etc..

  $$\ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \Big(y_i - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)\Big)^2$$

  we'll use the quadratic loss and sigmoid activation as an example...

Training via (stochastic) gradient descent

$$\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{2n}\sum_{i=1}^{n}\Big(y_i - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)\Big)^2 := \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \ell_n(\boldsymbol{w})$$

- Gradient descent:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla \ell_n(\boldsymbol{w}_t)$$

  where η_t is the step-size or learning rate.

Training via (stochastic) gradient descent

$$\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{2n}\sum_{i=1}^{n}\Big(y_i - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)\Big)^2 := \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \ell_n(\boldsymbol{w})$$

- Gradient descent:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla \ell_n(\boldsymbol{w}_t)$$

  where η_t is the step-size or learning rate.
- The gradient can be calculated via chain rule.
  - call $\hat{y}_i = \hat{y}_i(\boldsymbol{w}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)$, then

$$\begin{aligned}\frac{d}{d\boldsymbol{w}}\frac{1}{2}(y_i - \hat{y}_i)^2 = (\hat{y}_i - y_i)\frac{d\hat{y}_i(\boldsymbol{w})}{d\boldsymbol{w}} &= (\hat{y}_i - y_i)\sigma'(\boldsymbol{w}^\top \boldsymbol{x}_i)\boldsymbol{x}_i \\ &= \underbrace{(\hat{y}_i - y_i)\hat{y}_i(1-\hat{y}_i)}_{\text{scalar}}\boldsymbol{x}_i\end{aligned}$$

  where we used $\sigma'(z) = \sigma(z)(1-\sigma'(z))$. This is called "delta rule".

Stochastic gradient descent

Stochastic gradient descent uses only a mini-batch of data every iteration.

At every iteration t,

1. Draw a mini-batch of data indexed by $\mathcal{S}_t \in \{1, \cdots, n\}$;
2. Update

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \sum_{i \in \mathcal{S}_t} \nabla \ell(\boldsymbol{w}_t; \boldsymbol{x}_i, y_i)$$

Detour: Backpropagation

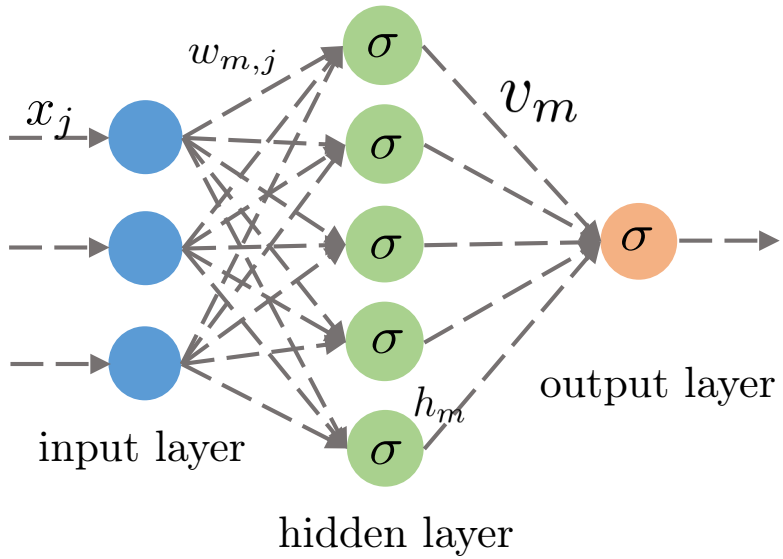
- Backpropagation is the basic algorithm to train neural network, rediscovered several times in the literature in the 1970-80's, but popularized by the 1986 paper by Rumelhart, Hinton, and Williams.
- Assuming node operations take unit time, backpropagation takes linear time, specifically, $O(\text{Network Size}) = O(V + E)$ to compute the gradient, where V is the number of vertices and E is the number of edges in the neural network.

main idea: chain rule from calculus.

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

Let's illustrate the process with single-output, 2-layer NN

Derivations of backpropagation



network output:

$$\hat{y} = \sigma\left(\sum_m v_m h_m\right) = \sigma\left(\sum_m v_m \sigma\left(\sum_j w_{m,j} x_j\right)\right)$$

loss function: $f = \frac{1}{2}(y - \hat{y})^2$.

Backpropogation I

Optimize the weights for each layer, starting with the layer closest to outputs and working back to the layer closest to inputs.

1. To update v_m's: realize

$$\begin{aligned}\frac{df}{dv_m} &= \frac{df}{d\hat{y}}\frac{d\hat{y}}{dv_m} \\ &= (\hat{y}-y)\frac{d\hat{y}}{dv_m} \\ &= (\hat{y}-y)\sigma'\left(\sum_m v_m h_m\right)h_m \\ &= \underbrace{(\hat{y}-y)\hat{y}(1-\hat{y})}_{\delta}\, h_m.\end{aligned}$$

   This is the same as updating the perceptron.

Backpropogation II

2. To update $w_{m,j}$'s: realize

$$\begin{aligned}\frac{df}{dw_{m,j}} &= \frac{df}{d\hat{y}}\frac{d\hat{y}}{dh_m}\frac{dh_m}{dw_{m,j}} \\ &= (\hat{y}-y)\hat{y}(1-\hat{y})v_m h_m(1-h_m)x_j \\ &= \delta v_m h_m(1-h_m)x_j\end{aligned}$$

Questions we may ask (in general)

- **Representation:** how well can a given network (fixed activation) approximate / explain the training data?
- **Generalization:** how well can the learned $\widehat{\boldsymbol{w}}$ behave in prediction during testing?
- **Optimization:** how does the output of (S)GD $\boldsymbol{w}_t$ relate to $\widehat{\boldsymbol{w}}$? (or we should really plug in $\boldsymbol{w}_t$ in the previous two questions!)

Nonconvex landscape of perceptron can be very bad

SGD converges to *local minimizers*. Are they global?

Theorem 12.1 (Auer et al., 1995)

Let $\sigma(\cdot)$ be sigmoid and $\ell(\cdot)$ be the quadratic loss function. There exists a sequence of training samples $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ such that $\ell_n(\boldsymbol{w})$ has $\lfloor \frac{n}{d} \rfloor^d$ distinct local minima.

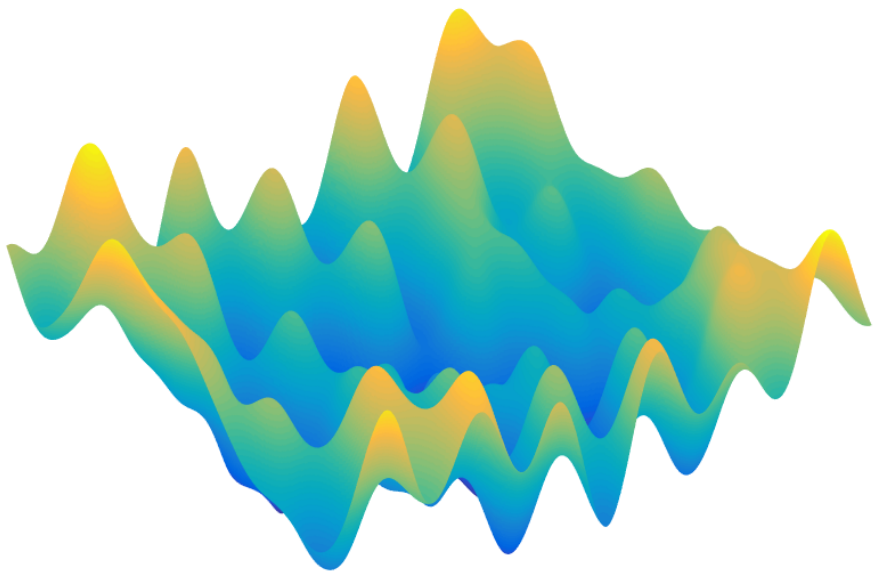
Nonconvex landscape of perceptron can be very bad

SGD converges to *local minimizers*. Are they global?

Theorem 12.1 (Auer et al., 1995)

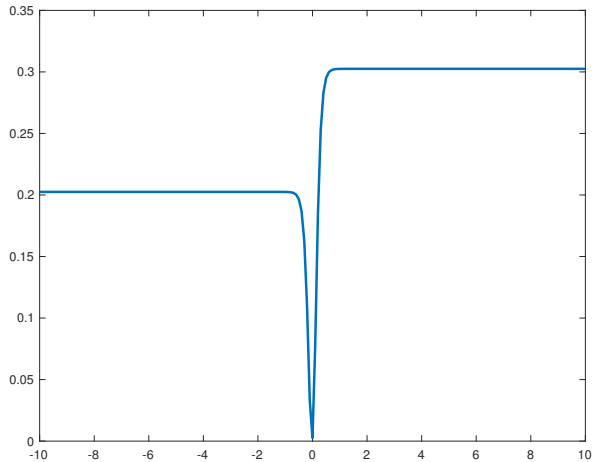
Let $\sigma(\cdot)$ be sigmoid and $\ell(\cdot)$ be the quadratic loss function. There exists a sequence of training samples $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ such that $\ell_n(\boldsymbol{w})$ has $\lfloor \frac{n}{d} \rfloor^d$ distinct local minima.

Consequence: there may exist exponentially many bad local minima with arbitrary data! — curse of dimensionality

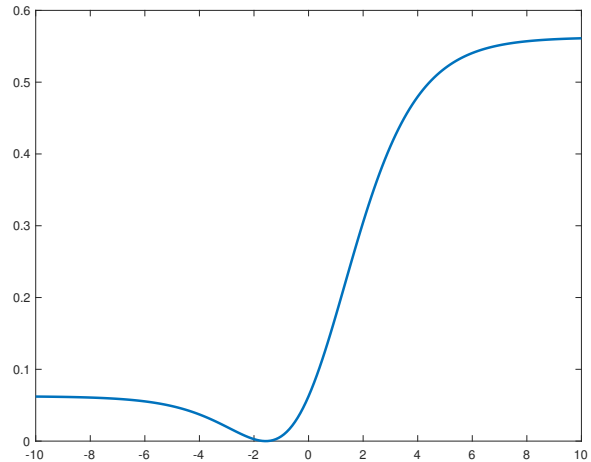


Why?

- **saturation** of the sigmoid



$\ell(w; 10, 0.55)$

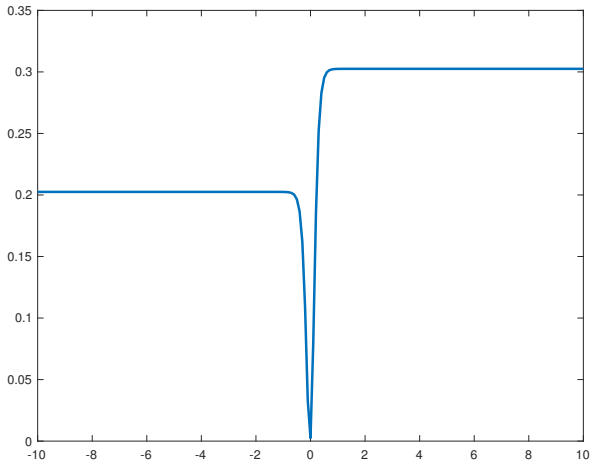


$\ell(w; 0.7, 0.25)$

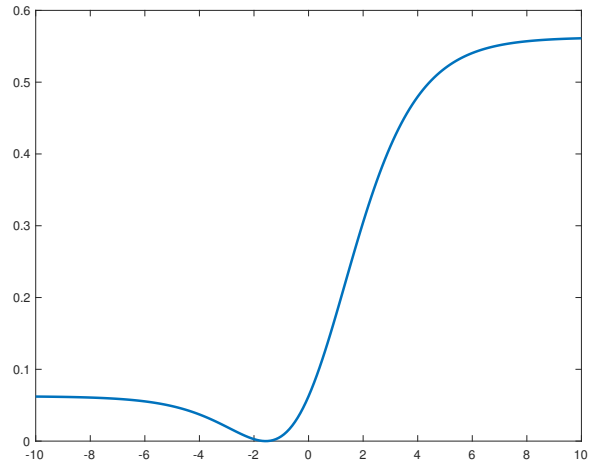
  - each sample produces a local min + flat surfaces away from the minimizer

Why?

- **saturation** of the sigmoid



$\ell(w; 10, 0.55)$

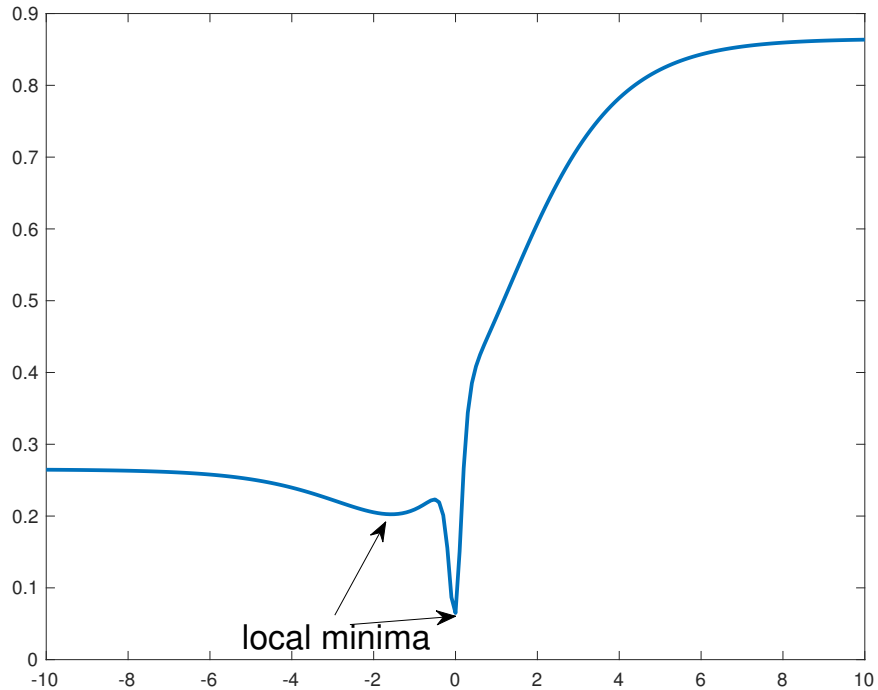


$\ell(w; 0.7, 0.25)$

  - each sample produces a local min + flat surfaces away from the minimizer
  - if the local min of sample A falls into the flat region of sample B (and vice versa), the sum of sample losses preserve both minima.

Why?

- We get one local minimum per sample in 1D.



- Curse of dimensionality: we construct the samples to get $\lfloor \frac{n}{d} \rfloor^d$ distinct local minima in d dim.

Statistical models come to rescue

Data/measurements follow certain **statistical models** and hence are not worst-case instances.

$$\text{minimize}_{\boldsymbol{w}} \ \ell_n(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} \ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

Statistical models come to rescue

Data/measurements follow certain **statistical models** and hence are not worst-case instances.

$$\text{minimize}_{\boldsymbol{w}}\ \ell_n(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\ell(\boldsymbol{w};\boldsymbol{x}_i,y_i) \quad \overset{m\to\infty}{\Longrightarrow} \quad \mathbb{E}[\ell(\boldsymbol{w};\boldsymbol{x},y)] := \ell(\boldsymbol{w})$$

Statistical models come to rescue

Data/measurements follow certain **statistical models** and hence are not worst-case instances.

$$\text{minimize}_{\boldsymbol{w}} \ \ell_n(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \quad \stackrel{m\to\infty}{\Longrightarrow} \quad \mathbb{E}[\ell(\boldsymbol{w}; \boldsymbol{x}, y)] := \ell(\boldsymbol{w})$$

empirical risk $\approx$ population risk

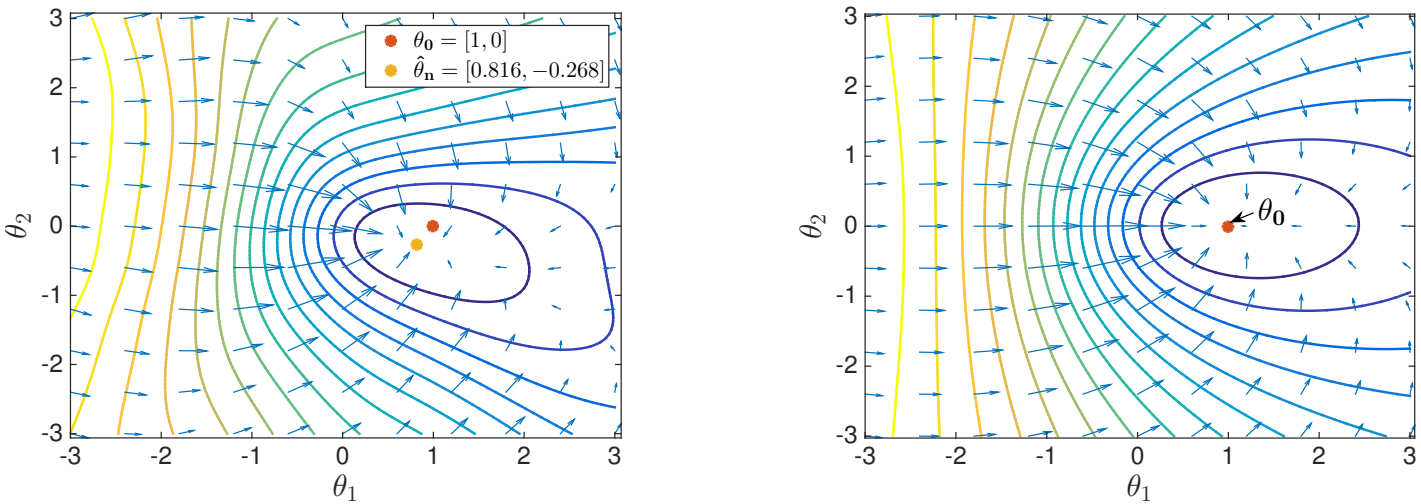


Figure credit: Mei, Bai and Montanari

Statistical models of training data

Assume the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ is *i.i.d.* drawn from some *distribution*:

$$(\boldsymbol{x}, y) \sim p(\boldsymbol{x}, y)$$

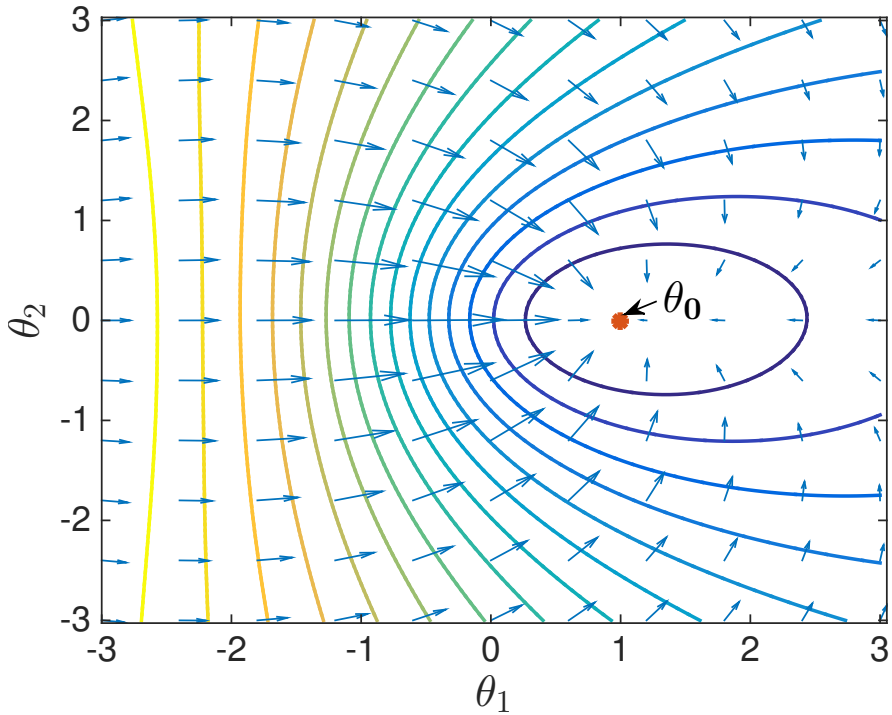
We are using neural networks to fit $p(\boldsymbol{x}, y)$.

- A planted-truth model: let $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and the label y is drawn as
  - *regression model:*
  $$y_i = \sigma(\boldsymbol{w}^{\star\top} \boldsymbol{x}_i)$$
  - *classification model:* $y_i \in \{0, 1\}$, where
  $$\mathbb{P}(y_i = 1) = \sigma(\boldsymbol{w}^{\star\top} \boldsymbol{x}_i)$$
- **Parameter recovery:** can we recover $\boldsymbol{w}^\star$ using $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$?

Roadmap

1. Step 1: Verify the landscape properties of population loss;
2. Step 2: translate properties of population loss to empirical loss;
3. Step 3: argue $\widehat{\boldsymbol{w}}$ (minimizer of empirical loss) is close to $\boldsymbol{w}^\star$ (minimizer of population loss).

Step 1: population risk



- $\boldsymbol{w}^\star$ is the unique local minimizer that is also global. No bad local minima!
- strongly convex near global optima ; large gradient elsewhere

Nonconvex landscape: from population to empirical

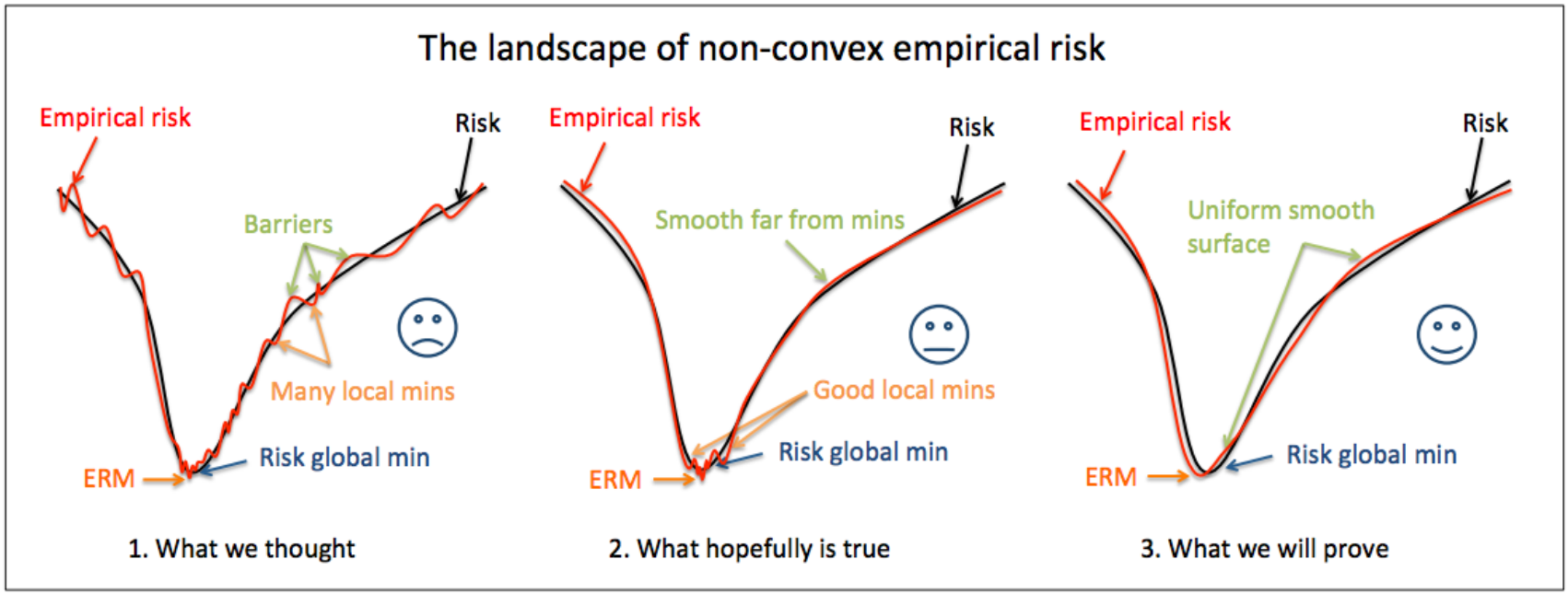


Figure 1: Possibe behaviors of non-convex empirical risk.

Figure credit: Mei, Bai and Montanari

Step 2: uniform convergence of gradients & Hessian

Theorem 12.2 (Bai, Song, Montanari, 2017)

Under suitable assumptions, for any $\delta > 0$, there exists a positive constant C depending on (R, δ) but independent of n and d, such that as long as $n \geq Cd \log d$, we have

1. *preservation of gradient:*

$$\mathbb{P}\left(\sup_{\|\boldsymbol{w}\| \leq R} \|\nabla \ell_n(\boldsymbol{w}) - \nabla \ell(\boldsymbol{w})\|_2 \leq \sqrt{\frac{Cd \log n}{n}} \right) \geq 1 - \delta.$$

2. *preservation of Hessian:*

$$\mathbb{P}\left(\sup_{\|\boldsymbol{w}\| \leq R} \left\|\nabla^2 \ell_n(\boldsymbol{w}) - \nabla^2 \ell(\boldsymbol{w})\right\| \leq \sqrt{\frac{Cd \log n}{n}} \right) \geq 1 - \delta.$$

Step 3: establish rate of convergence for ERM

By the mean-value theorem, there exists some $\boldsymbol{w}'$ between $\widehat{\boldsymbol{w}}$ and $\boldsymbol{w}^\star$ such that

$$\begin{aligned}\ell_n(\widehat{\boldsymbol{w}}) &= \ell_n(\boldsymbol{w}^\star) + \langle \nabla \ell_n(\boldsymbol{w}^\star), \widehat{\boldsymbol{w}} - \boldsymbol{w}^\star \rangle + \frac{1}{2}(\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star)^\top \nabla^2 \ell_n(\boldsymbol{w}')(\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star) \\ &\leq \ell_n(\boldsymbol{w}^\star)\end{aligned}$$

where the last line follows by optimality of $\widehat{\boldsymbol{w}}$. Then

$$\begin{aligned}\frac{1}{2}\lambda_{\min}(\nabla^2 \ell_n(\boldsymbol{w}'))\, \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star\|_2^2 &\leq \frac{1}{2}(\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star)^\top \nabla^2 \ell_n(\boldsymbol{w}')(\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star) \\ &\leq |\langle \nabla \ell_n(\boldsymbol{w}^\star), \widehat{\boldsymbol{w}} - \boldsymbol{w}^\star \rangle| \\ &\leq \|\nabla \ell_n(\boldsymbol{w}^\star)\| \cdot \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star\|\end{aligned}$$

$$\to \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^\star\|_2 \leq \frac{2\|\nabla \ell_n(\boldsymbol{w}^\star)\|}{\lambda_{\min}(\nabla^2 \ell_n(\boldsymbol{w}'))} \lesssim \sqrt{\frac{Cd \log n}{n}}.$$

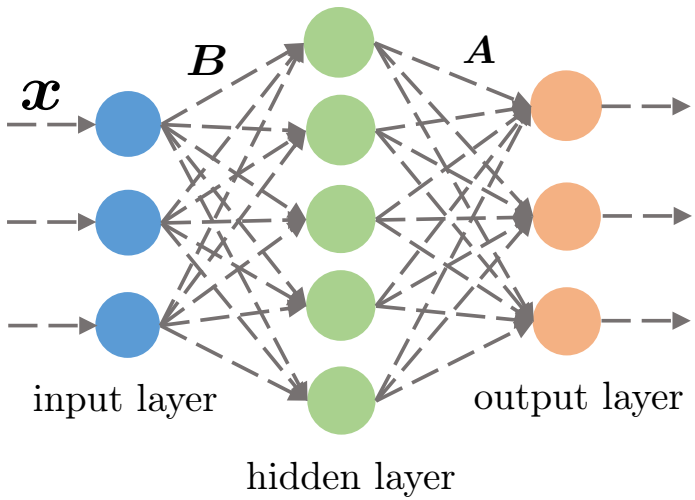
two-layer networks

Two-layer linear network

Given *arbitrary* data $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^n$, $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^d$, fit the two-layer linear network with quadratic loss:

$$f(\boldsymbol{A}, \boldsymbol{B}) = \sum_{i=1}^{n} \|\boldsymbol{y}_i - \boldsymbol{A}\boldsymbol{B}\boldsymbol{x}_i\|_2^2$$

where $\boldsymbol{B} \in \mathbb{R}^{p \times d}$, $\boldsymbol{A} \in \mathbb{R}^{d \times p}$, where $p \leq d$.



- special case: auto-association (auto-encoding, identity mapping), where $\boldsymbol{y}_i = \boldsymbol{x}_i$, for e.g. image compression.

Landscape of 2-layer linear network

- bears some similarity with the nonconvex matrix factorization problem;
- Lack identifiability: for any invertible $\boldsymbol{C}$, $\boldsymbol{AB} = (\boldsymbol{AC})(\boldsymbol{C}^{-1}\boldsymbol{B})$.
- Define $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$ and $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_n]$, then

$$f(\boldsymbol{A}, \boldsymbol{B}) = \|\boldsymbol{Y} - \boldsymbol{ABX}\|_{\mathrm{F}}^2$$

- Let $\boldsymbol{\Sigma}_{XX} = \sum_{i=1}^n \boldsymbol{x}_i \boldsymbol{x}_i^\top = \boldsymbol{X}\boldsymbol{X}^\top$, $\boldsymbol{\Sigma}_{YY} = \boldsymbol{Y}\boldsymbol{Y}^\top$, $\boldsymbol{\Sigma}_{XY} = \boldsymbol{X}\boldsymbol{Y}^\top$, and $\boldsymbol{\Sigma}_{YX} = \boldsymbol{Y}\boldsymbol{X}^\top$.
- When $\boldsymbol{X} = \boldsymbol{Y}$, any optimum $\boldsymbol{AB} = \boldsymbol{U}\boldsymbol{U}^\top$, where $\boldsymbol{U}$ is the top p eigenvectors of $\boldsymbol{\Sigma}_{XX}$.

Landscape of 2-layer linear network

Theorem 12.3 (Baldi and Hornik, 1989)

Suppose $\boldsymbol{X}$ *is full rank and hence* $\boldsymbol{\Sigma}_{XX}$ *is invertible. Further assume*

$$\boldsymbol{\Sigma} := \boldsymbol{\Sigma}_{YX}\boldsymbol{\Sigma}_{XX}^{-1}\boldsymbol{\Sigma}_{XY}$$

is full rank with d *distinct eigenvalues* $\lambda_1 > \cdots > \lambda_d > 0$*. Then* $f(\boldsymbol{A}, \boldsymbol{B})$ *has no spurious local minima, except for equivalent versions of global minimum due to invertible transformations.*

- no bad local min!
- generalizable to multi-layer linear networks.

Critical points of two-layer linear networks

Lemma 12.4 (critical points)

Any critical point satisfies

$$\boldsymbol{AB} = \mathcal{P}_{\boldsymbol{A}} \boldsymbol{\Sigma}_{YX} \boldsymbol{\Sigma}_{XX}^{-1},$$

where $\boldsymbol{A}$ *satisfies*

$$\mathcal{P}_{\boldsymbol{A}} \boldsymbol{\Sigma} = \mathcal{P}_{\boldsymbol{A}} \boldsymbol{\Sigma} \mathcal{P}_{\boldsymbol{A}} = \boldsymbol{\Sigma} \mathcal{P}_{\boldsymbol{A}},$$

where $\mathcal{P}_{\boldsymbol{A}}$ *is the ortho-projector projecting onto the column span of the sub-indexed matrix.*

Critical points of two-layer linear networks

Let the EVD of $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_{YX}\boldsymbol{\Sigma}_{XX}^{-1}\boldsymbol{\Sigma}_{XY}$ be $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$.

Lemma 12.5 (critical points)

At any critical point, $\boldsymbol{A}$ can be written in the form

$$\boldsymbol{A} = [\boldsymbol{U}_{\mathcal{J}}, \boldsymbol{0}_{d\times(p-r)}]\boldsymbol{C}$$

where rank$(\boldsymbol{A}) = r \le p$, $\mathcal{J} \subset \{1, \ldots, d\}$, $|\mathcal{J}| = r$ and $\boldsymbol{C}$ is invertible. Correspondingly,

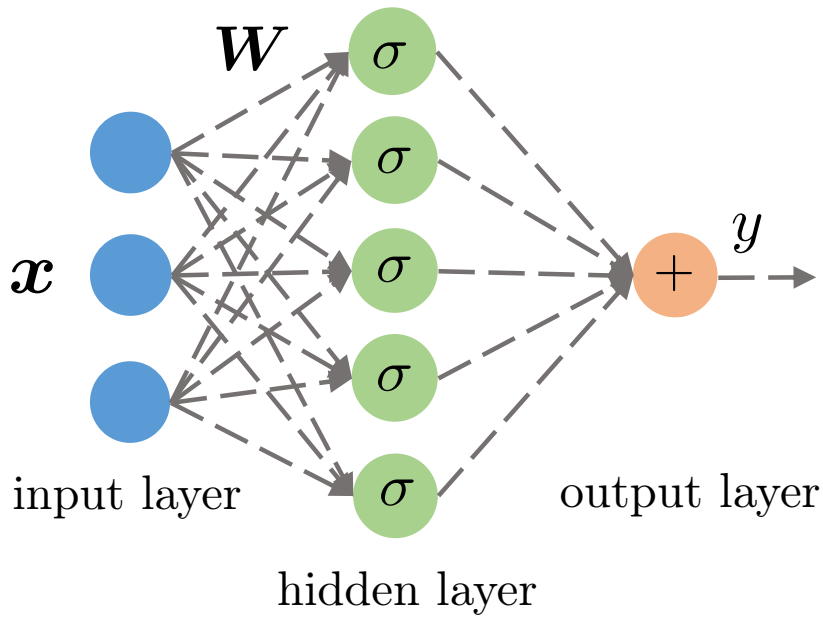
$$\boldsymbol{B} = \boldsymbol{C}^{-1}\begin{bmatrix} \boldsymbol{U}_{\mathcal{J}}^\top \boldsymbol{\Sigma}_{YX}\boldsymbol{\Sigma}_{XX}^{-1} \\ \text{last } p-r \text{ rows of } \boldsymbol{C}\boldsymbol{L} \end{bmatrix},$$

where $\boldsymbol{L}$ is any $d \times d$ matrix.

Verify $(\boldsymbol{A}, \boldsymbol{B})$ is global optima if and only if $\mathcal{J} = \{1, \ldots, p\}$.

Two-layer nonlinear network

Local strong convexity under the Gaussian model [Fu et al., 2018].



Reference I

[1] "*ImageNet*," `www.image-net.org/`.

[2] "*Deep learning*," Lecun, Bengio, and Hinton, *Nature*, 2015.

[3] "*ImageNet classification with deep convolutional neural networks*," Krizhevsky et al., *NIPS*, 2012.

[4] "*Learning representations by back-propagating errors*," Rumelhart, Hinton, and Williams, *Nature*, 1986.

[5] "*Dropout: A simple way to prevent neural networks from overfitting*," Srivastava et al., *JMLR*, 2014.

[6] "*Dropout Training as Adaptive Regularization*," Wager et al., *NIPS*, 2013.

[7] "*On the importance of initialization and momentum in deep learning*," Sutskever et al., *ICML*, 2013.

[8] "*Exponentially many local minima for single neurons*," P. Auer, M. Herbster and K. Warmuth, *NIPS*, 1995.

Reference II

[9] "*The landscape of empirical risk for non-convex losses*," S. Mei, Y. Bai, and A. Montanari, *arXiv preprint arXiv:1607.06534*, 2016.

[10] "*Neural networks and principal component analysis: Learning from examples without local minima*," P. Baldi and K. Hornik, *Neural Networks*, 1989.

[11] "*Local Geometry of One-Hidden-Layer Neural Networks for Logistic Regression*," Fu, Chi, Liang, *arXiv preprint arXiv:1802.06463*, 2018.