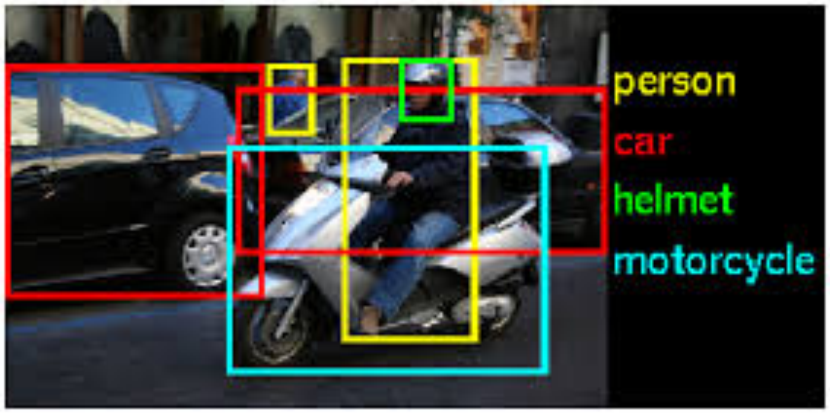
Optimization Methods for Supervised Machine Learning: From Linear Models to Deep Learning. Part I

Katya Scheinberg
jointly with Frank Curtis
LEHIGH UNIVERSITY

ML applications

- Computer vision
- Machine translation
- Speech recognition
- Text categorization
- Recommender systems
- Ranking web search results
- Next word prediction
- Video content classification
- Anomaly detection

Popular applications of deep learning models



Data on Google scale today

- ImageNet Large Scale Visual Recognition Competition: **1.2 million 224x224 images**
- **300** hours of video are uploaded to YouTube every minute! **819,417,600** hours of video total **> 93,000 years**.
- Google's big initiative is: **next billion users**.
  - Next word prediction in texts.
  - Machine translation
  - Image classification and recognition

LEARNING PROBLEM, SETUP

Supervised learning problem

- Given a sample data set S of n (input, label) pairs, written

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}.$$

- each pair is an observation of the random variables (x, y) with some unknown distribution $P(x, y)$ over $\mathcal{X}, \mathcal{Y}$.

- each pair (x_i, y_i) is an independent sample

- Find a hypothesis (predictor) $p(w, \cdot)$ such that $p(w, x) \approx y$, i.e.,

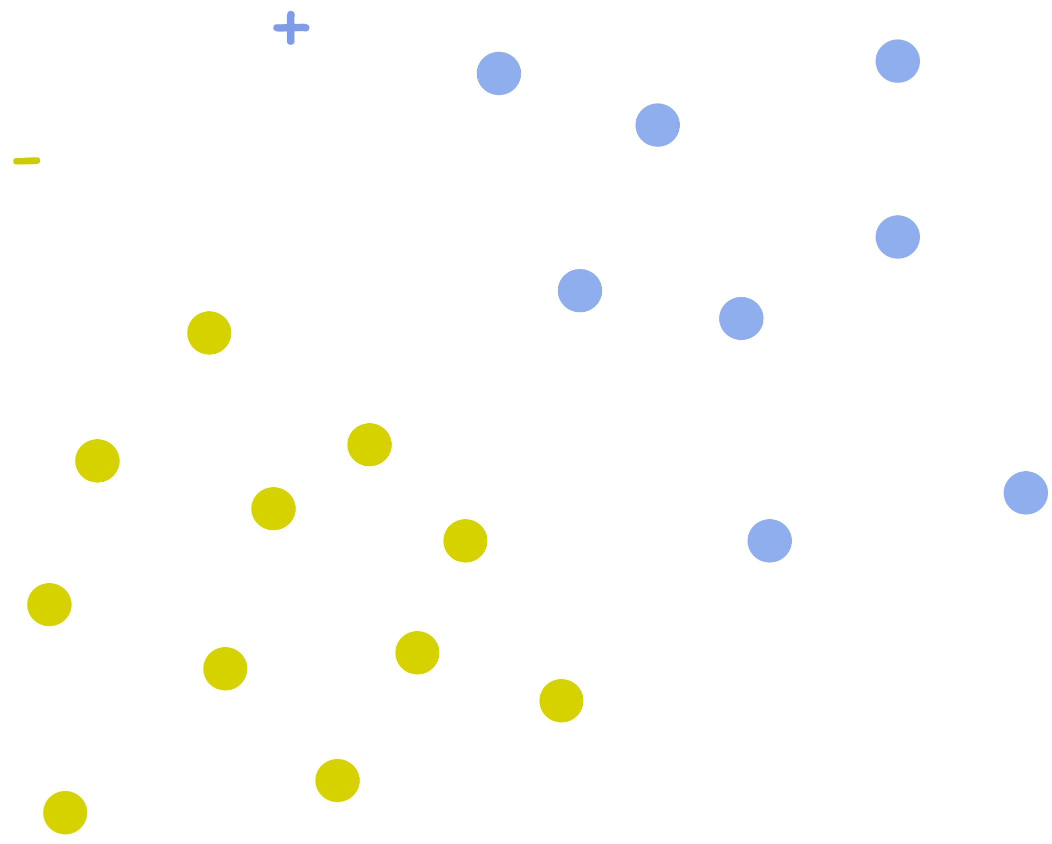
$$\max_w \int_{\mathcal{X}\times\mathcal{Y}} \mathbb{1}[p(w, x) \approx y] dP(x, y).$$

For example, *x* is the image of a letter and *y* is the letter label.
What should *p(w,x)* be?

Binary classification problem

$Y \in \{-1,+1\}$

Two sets of
labeled points



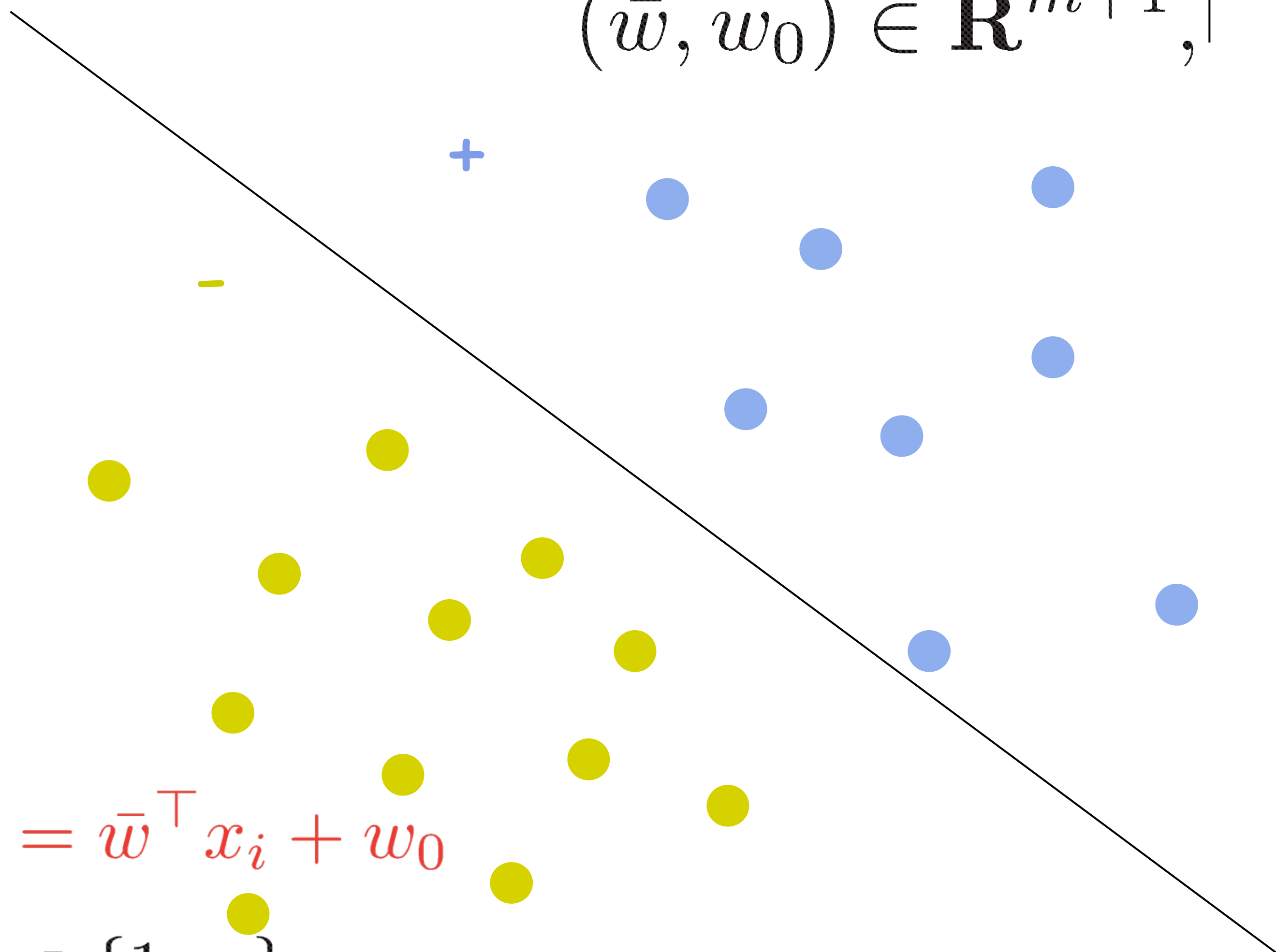
Linear classifier

y∈ {-1,+1}

$$\bar{w}^\top x + w_0 = 0$$

$$(\bar{w}, w_0) \in \mathbf{R}^{m+1},$$

Like this:



$$p(w, x_i) = \bar{w}^\top x_i + w_0$$

$$\forall i \in \{1..n\}$$

Binary Classification Objective

Expected risk: ideal objective

$$\max_{w} \int_{\mathcal{X}\times\mathcal{Y}} \mathbb{1}[yp(w,x) > 0] dP(x,y).$$

$\Updownarrow$

$$\min_{w} f_{01}(w) = \mathbb{E}[\ell_{01}(p(w,x),y)]$$

$$\ell_{01}(p(w,x),y) = \begin{cases} 0 & \text{if } yp(w,x) > 0 \\ 1 & \text{if } yp(w,x) \leq 0, \end{cases}$$

Usually an intractable problem

Binary Classification Objective

Expected risk: ideal objective

$$\max_{w} \int_{\mathcal{X}\times\mathcal{Y}} \mathbb{1}[yp(w,x) > 0]dP(x,y).$$

$$\updownarrow$$

$$\min_{w} f_{01}(w) = \mathbb{E}[\ell_{01}(p(w,x),y)]$$

$$\ell_{01}(p(w,x),y) = \begin{cases} 0 & \text{if } yp(w,x) > 0 \\ 1 & \text{if } yp(w,x) \leq 0, \end{cases}$$

Empirical risk: realizable objective

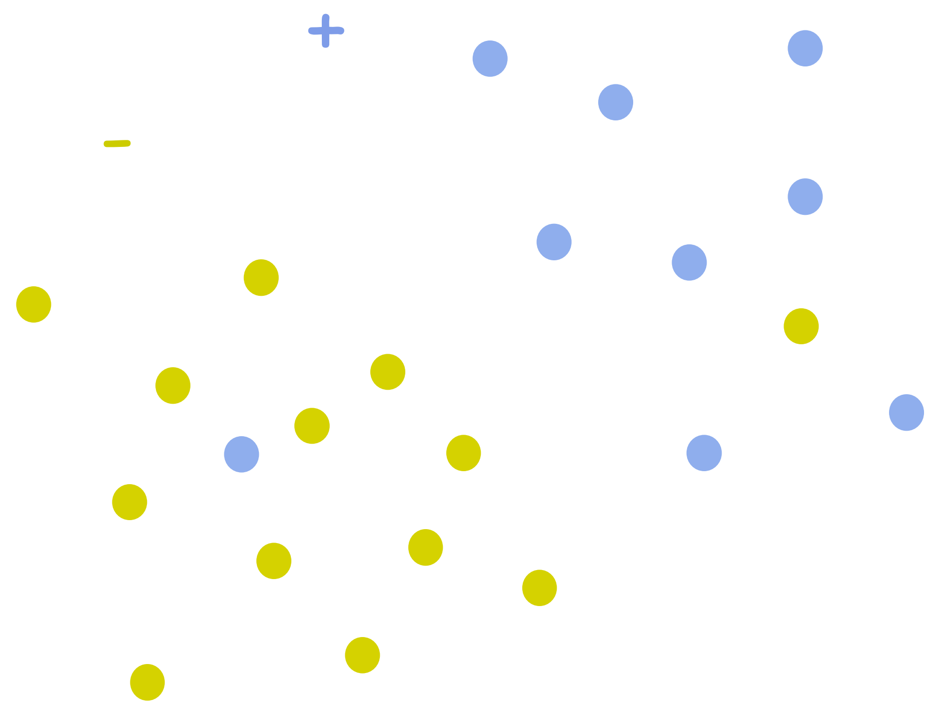
$$\min_{w} \hat{f}_{01}(w) = \frac{1}{n}\sum\nolimits_{i=1}^{n} \ell_{01}(p(w,x_i),y_i)$$

Finite, but NP hard problem

Handling outliers, logistic regression

$$\ln\left(\frac{P(Y=y|x)}{1-P(Y=y|x)}\right) = yp(w,x).$$

$$P(Y=y|x) = \frac{e^{yp(w,x)}}{1+e^{yp(w,x)}} = \frac{1}{1+e^{-yp(w,x)}}. \quad (1)$$



Logistic Regression Model

Expected loss: ideal objective

$$\min_{w} \; f(w) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(p(w,x),y)dP(x,y) = \mathbb{E}[\ell(p(w,x),y)], \qquad (1)$$

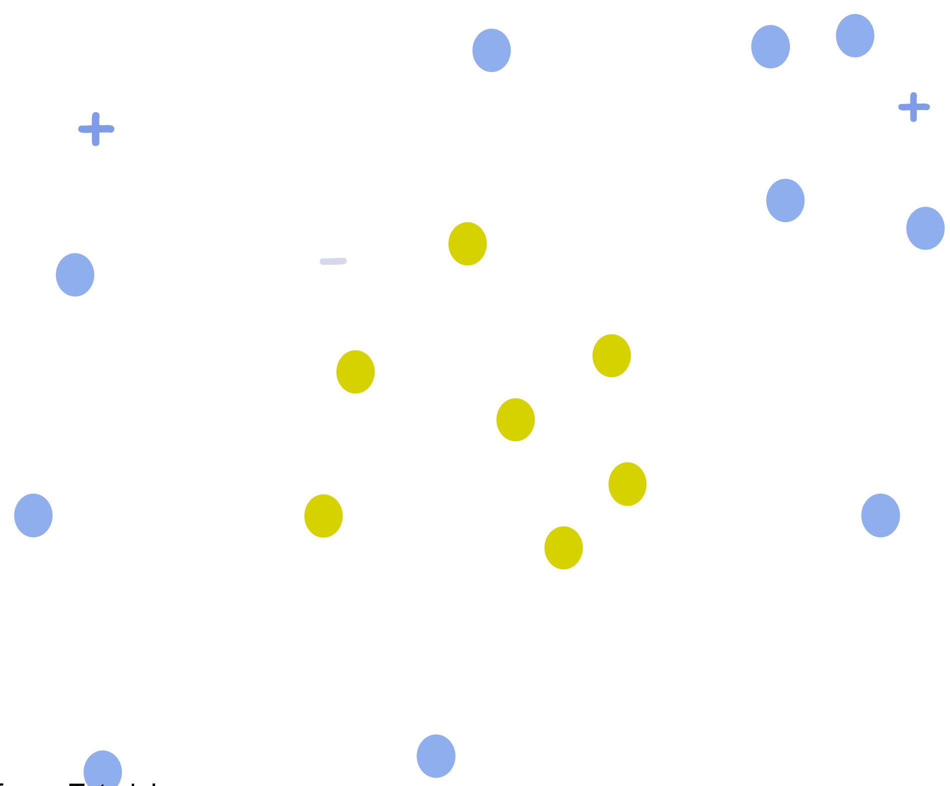
$$\ell(p(w,x),y) = \log(1+e^{-yp(w,x)}).$$

Empirical loss: realizable objective

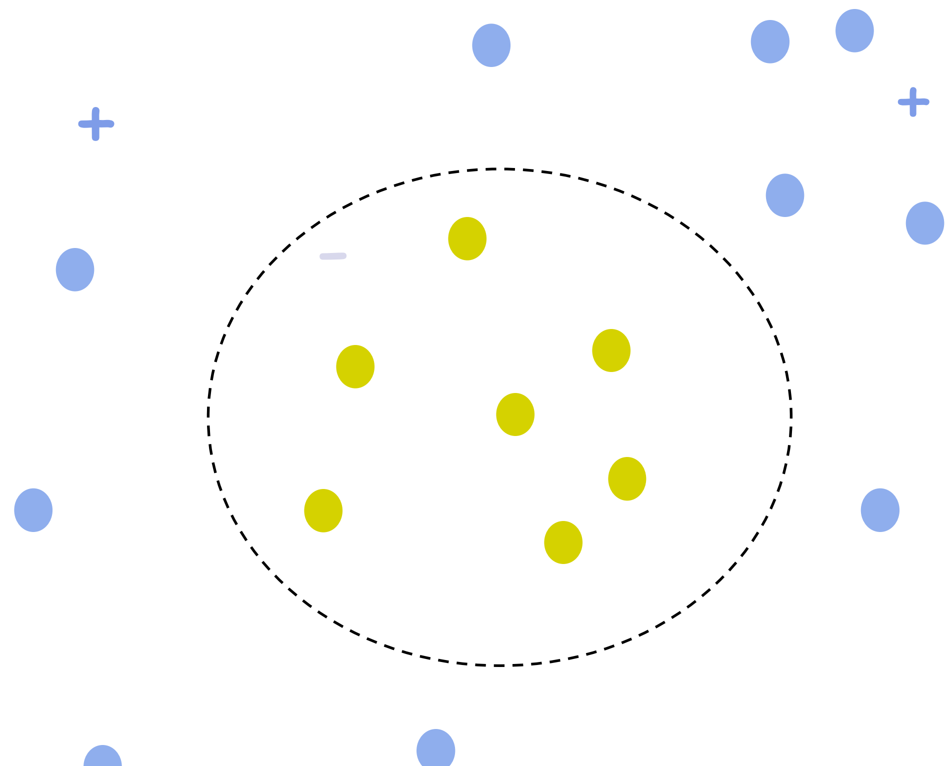
$$\min_{w} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1+e^{-yp(w,x)})$$

This is a convex function when *p(w,x)* is linear in *w*

Linear classifier, generalization



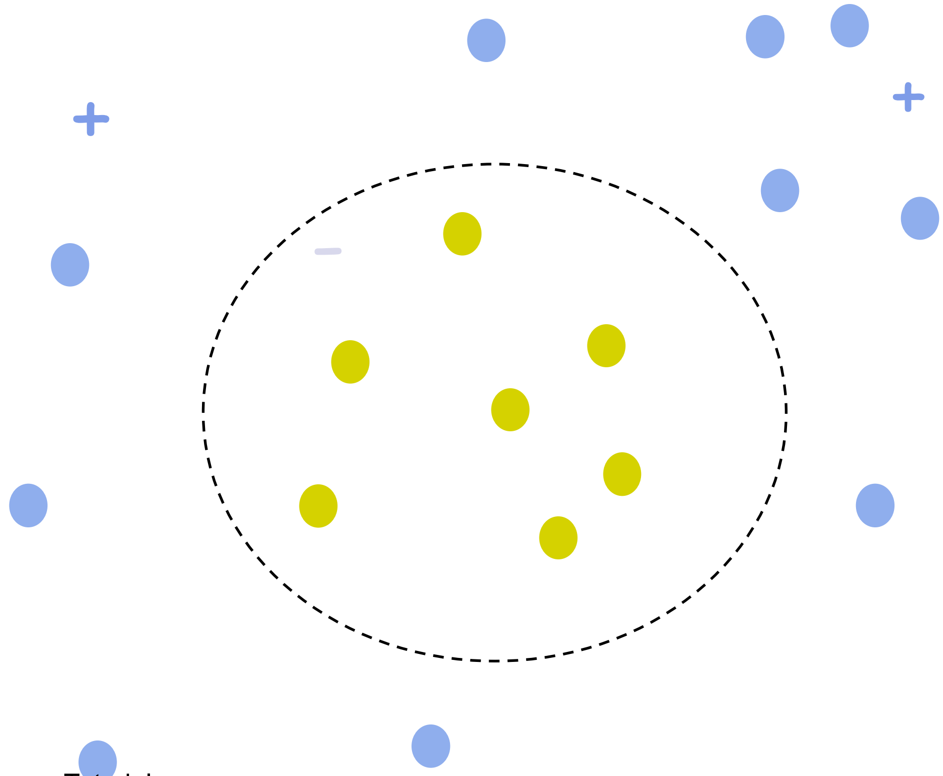
Linear classifier, generalization



Linear classifier, generalization

$$w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2 + w_0$$

$$\bar{w}^\top \phi(x) + w_0, \;\; \phi(x) = (x_1, x_2, x_1^2, x_1 x_2, x_2^2) \in \mathbf{R}^5$$



Learning guarantees

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

Problem is well defined, even if it may be difficult to solve

Learning guarantees

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

Problem is well defined, even if it may be difficult to solve.

However, we are really interested in

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

Learning guarantees

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

Problem is well defined, even if it may be difficult to solve.

However, we are really interested in

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

How does $\hat{w}_{opt}$ behave on the unseen data?

$$|\mathbb{E}[\ell(p(w_{opt}, x), y)] - \frac{1}{n}\sum_{i=1}^{n} \ell(p(w_{opt}, x_i), y_i)| \leq O\left(\sqrt{\frac{\mathcal{C}(p(\cdot,\cdot))}{n}}\right)$$

$\mathcal{C}(p(\cdot,\cdot))$ is a measure of complexity of the class of predictors

Learning guarantees

$$\hat{w}_{opt} = \arg \min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

Problem is well defined, even if it may be difficult to solve.

However, we are really interested in

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

How does $\hat{w}_{opt}$ behave on the unseen data?

$$|\mathbb{E}[\ell_{01}(p(w_{opt}, x), y)] - \frac{1}{n}\sum_{i=1}^{n} \ell_{01}(p(w_{opt}, x_i), y_i)| \leq O\left(\sqrt{\frac{\mathcal{C}(p(\cdot,\cdot))}{n}}\right)$$

$\mathcal{C}(p(\cdot,\cdot))$ is a measure of complexity of the class of predictors

Learning guarantees

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

Problem is well defined, even if it may be difficult to solve.

However, we are really interested in

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

How do $\hat{w}_{opt}$ and w_{opt} relate?

$$|\mathbb{E}[\ell(p(w_{opt}, x), y] - \mathbb{E}[\ell(p(\hat{w}_{opt}, x), y)]| \leq O\left(\sqrt{\frac{\mathcal{C}(p(\cdot, \cdot))}{n}}\right)$$

$\mathcal{C}(p(\cdot, \cdot))$ is a measure of complexity of the class of predictors

Learning guarantees

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

Problem is well defined, even if it may be difficult to solve.

However, we are really interested in

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

How do $\hat{w}_{opt}$ and w_{opt} relate?

$$|\mathbb{E}[\ell_{01}(p(w_{opt}, x), y] - \mathbb{E}[\ell_{01}(p(\hat{w}_{opt}, x), y)]| \leq O\left(\sqrt{\frac{\mathcal{C}(p(\cdot, \cdot))}{n}}\right)$$

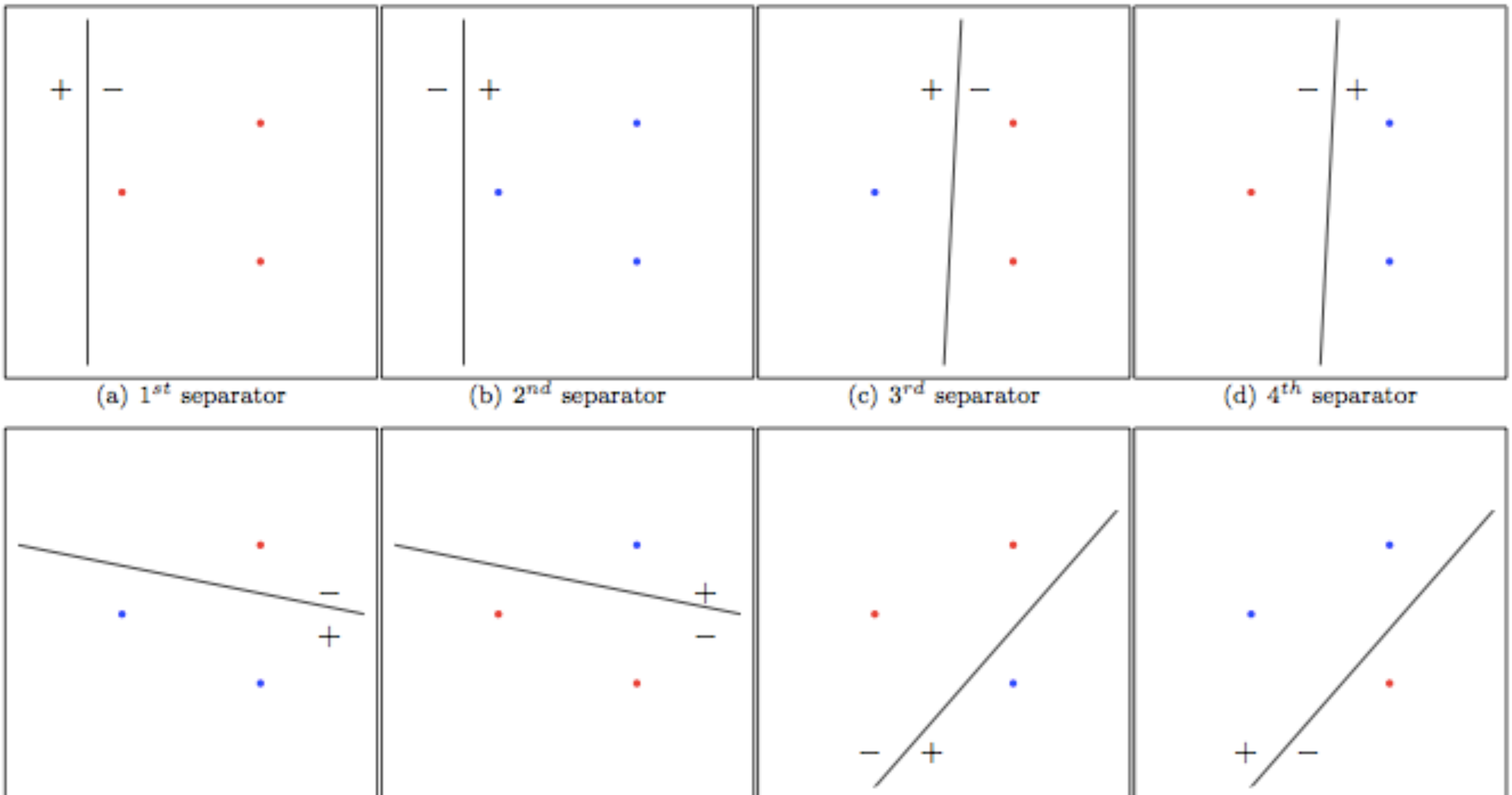
$\mathcal{C}(p(\cdot, \cdot))$ is a measure of complexity of the class of predictors

Learning guarantees via Vapnik-Chervonenkis (VC)-dimension

$$|\mathbb{E}[\ell_{01}(p(w,x),y] - \sum_{i=1}^{n} \ell_{01}(p(w,x_i),y_i)| \leq O\left(\sqrt{\frac{VC(p(\cdot,\cdot))}{n}}\right)$$

- ***VC(p(·,·)*** **- VC dimension of a set of classifiers – is the maximum number of points** ***x*** **such that any labeling can be separated by a classifier from this set.**

Vapnik-Chervonenkis (VC)-dimension, example



(a) 1^{st} separator (b) 2^{nd} separator (c) 3^{rd} separator (d) 4^{th} separator

(e) 5^{th} separator (f) 6^{th} separator (g) 7^{th} separator (h) 8^{th} separator

Learning guarantees via Vapnik-Chervonenkis (VC)-dimension

$$|\mathbb{E}[\ell_{01}(p(w,x),y] - \sum_{i=1}^{n} \ell_{01}(p(w,x_i),y_i)| \leq O\left(\sqrt{\frac{VC(p(\cdot,\cdot))}{n}}\right)$$

- *VC(p(·,·)* - VC dimension of a set of classifiers – is the maximum number of points *x* such that any labeling can be separated by a classifier from this set.
- VC(linear classifiers) = *m+1*
- Conclusion: large dimension of w require large data sets.

OPTIMIZATION METHODS FOR LOGISTIC REGRESSION

Gradient descent with line search

$$\min_{w \in \mathbf{R}^m} F(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

$$\nabla F(w_k) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w) \tag{1}$$

Algorithm 1 Gradient descent with line search

Parameters: the intial step size α_0, backtracking parameter $\gamma \in (0,1)$.
Initialize: w_0
Iterate:
for $k = 1, 2, \ldots$ **do**
  for $j = 0, 1, 2, \ldots$ **do**
    $w_{k+1} = w_k - \gamma^j \alpha_0 \nabla F(w_k)$
    Compute $F(w_k)$, if it satifies sufficient decrease condition, then **Break**
  end for
end for

Gradient descent with line search

$$\min_{w \in \mathbf{R}^m} F(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

$$\nabla F(w_k) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w) \tag{1}$$

Convergence rate *$O(\log(1/\epsilon))$* (strongly convex case)

Algorithm 1 Gradient descent with line search

Parameters: the intial step size α_0, backtracking parameter $\gamma \in (0, 1)$.
Initialize: w_0
Iterate:
for $k = 1, 2, \ldots$ **do**
  for $j = 0, 1, 2, \ldots$ **do**
    $w_{k+1} = w_k - \gamma^j \alpha_0 \nabla F(w_k)$
    Compute $F(w_k)$, if it satifies sufficient decrease condition, then **Break**
  end for
end for

Stochastic Gradient Descent

Choose a subset of *{1, ⋯, n}*, $S_{k,}$ uniformly at random

$$\nabla_{S_k} F(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(x)$$

$$\mathbb{E}[\nabla_{S_k} F(w_k)] = \nabla F(w_k)$$

Parameters: the step size sequence $\eta_k > 0$ and the minibatch size s
Initialize: w_0
for $k = 1, 2, \ldots$ **do**
  Generate S_k, $|S_k| = s$, uniformly from $\{1, \ldots, n\}$
  $w_{k+1} = w_k - \eta_k \nabla_{S_k} F(w_k)$
end for

Stochastic Gradient Descent

Choose a subset of *{1, ⋯, n}*, $S_{k,}$ uniformly at random

$$\nabla_{S_k} F(w_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(x)$$

$$\mathbb{E}[\nabla_{S_k} F(w_k)] = \nabla F(w_k)$$

Parameters: the step size sequence $\eta_k > 0$ and the minibatch size s
Initialize: w_0
for $k = 1, 2, \ldots$ **do**
  Generate S_k, $|S_k| = s$, uniformly from $\{1, \ldots, n\}$
  $w_{k+1} = w_k - \eta_k \nabla_{S_k} F(w_k)$
end for

Convergence rate *O(1/*ϵ*)* (strongly convex case)

Work per-iteration is O(sm)<<O(nm), but convergence sensitive to η_k and *s*

Stochastic Gradient Descent with Momentum

Parameters: learning rate $\eta > 0$; momentum weight $\theta \in (0,1)$; mini-batch size $s \in \mathbb{N}$

Initialize: $w_0 \in R^n$; $v_0 = 0 \in R^n$

for $k = 1, 2, \ldots$ **do**

  Generate S_k with $|S_k| = s$ uniformly from $\{1, \ldots, n\}$

  Set $v_k \leftarrow \theta v_{k-1} + \nabla_{S_k} F(w_k)$

  Set $w_{k+1} \leftarrow w_k - \eta v_k$

end for

Work per-iteration is O(sm), less sensitive to η but no convergence theory

Stochastic Variance Reducing Gradient method

Parameters: learning rate $\eta > 0$; mini-batch size $s \in \mathbb{N}$; inner loop size $m \in \mathbb{N}$

Initialize: $\tilde{w}_0 \in R^n$

for $k = 1, 2, \ldots$ **do** — Outer Loop

  Set $w_0 \leftarrow \tilde{w}_{k-1}$

  Set $v_0 \leftarrow \nabla F(w_0)$

  Set $w_1 \leftarrow w_0 - \eta v_0$

  for $t = 1, \ldots, m-1$ **do** — Inner Loop

    Generate S_t with $|S_t| = s$ uniformly from $\{1, \ldots, n\}$

    Set $v_t \leftarrow \nabla_{S_t} F(w_t) - \nabla_{S_t} F(w_0) + v_0$

    Set $w_{t+1} \leftarrow w_t - \eta v_t$

  end for

  Set $\tilde{w}_k = w_t$ with t chosen uniformly from $\{0, \ldots, m\}$

end for

[Johnson & Zhang, 2013]

SARAH (momentum version of SVRG)

Parameters: learning rate $\eta > 0$; mini-batch size $s \in \mathbb{N}$; inner loop size $m \in \mathbb{N}$

Initialize: $\tilde{w}_0 \in R^n$

for $k = 1, 2, \ldots$ **do** — Outer Loop

  Set $w_0 \leftarrow \tilde{w}_{k-1}$

  Set $v_0 \leftarrow \nabla F(w_0)$

  Set $w_1 \leftarrow w_0 - \eta v_0$

  for $t = 1, \ldots, m-1$ **do** — Inner Loop

    Generate S_t with $|S_t| = s$ uniformly from $\{1, \ldots, n\}$

    Set $v_t \leftarrow \nabla_{S_t} F(w_t) - \nabla_{S_t} F(w_{t-1}) + v_{t-1}$

    Set $w_{t+1} \leftarrow w_t - \eta v_t$

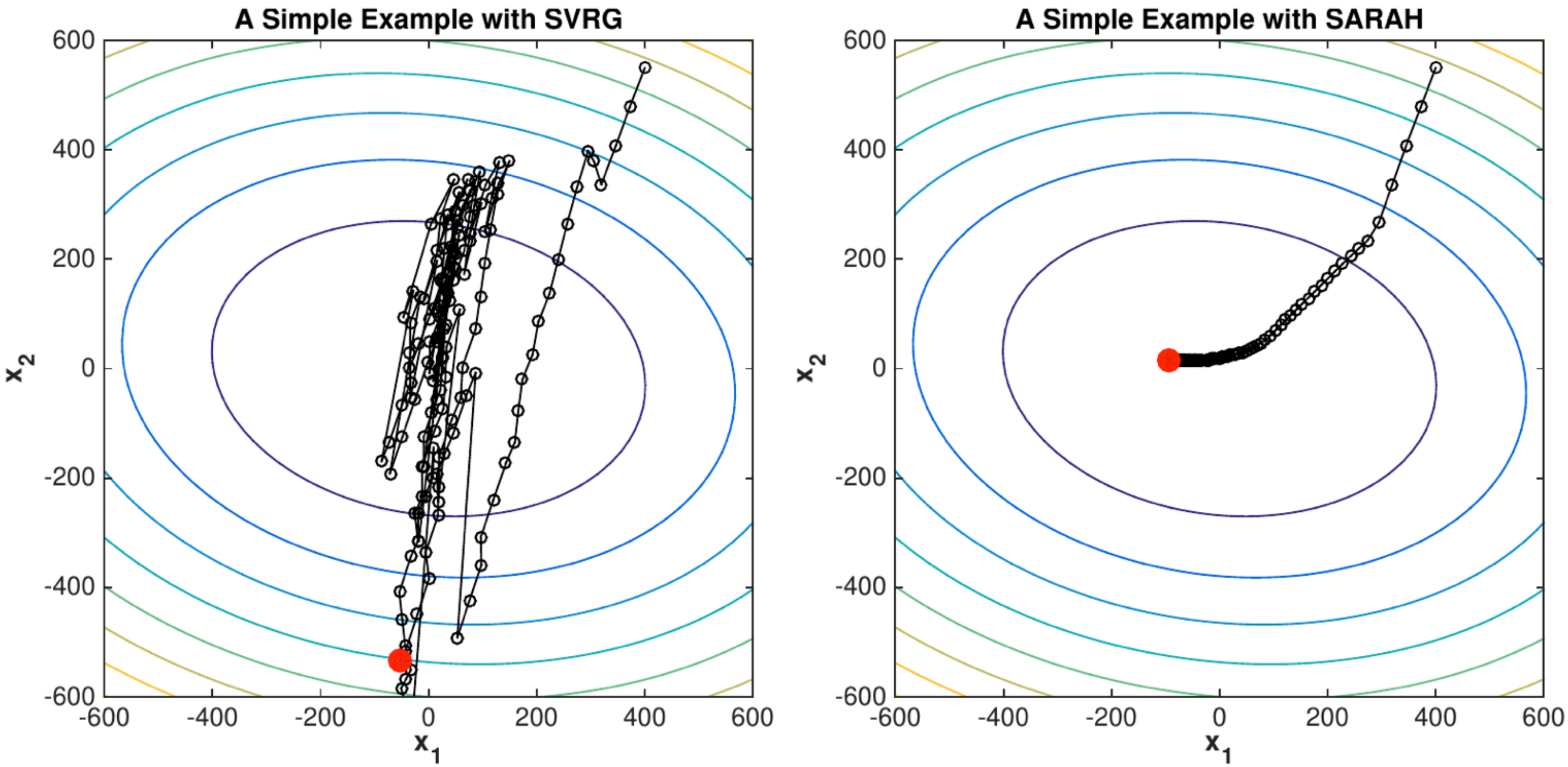
  end for

  Set $\tilde{w}_k = w_t$ with t chosen uniformly from $\{0, \ldots, m\}$

end for

[Nguyen, Liu, S & Takac, 2017]

Regular SG vs. Momentum SG



Convergence rates comparisons

For strongly convex functions, κ is the condition number

Method	Complexity
GD	$\mathcal{O}\left(n\kappa \log\left(1/\epsilon\right)\right)$
SGD	$\mathcal{O}\left(1/\epsilon\right)$
SVRG	$\mathcal{O}\left(\left(n+\kappa\right)\log\left(1/\epsilon\right)\right)$
SARAH	$\mathcal{O}\left(\left(n+\kappa\right)\log\left(1/\epsilon\right)\right)$

For convex functions

Method	Complexity
GD	$\mathcal{O}\left(n/\epsilon\right)$
SGD	$\mathcal{O}\left(1/\epsilon^2\right)$
SVRG	$\mathcal{O}\left(n+\left(\sqrt{n}/\epsilon\right)\right)$
SARAH	$\mathcal{O}\left(\left(n+\left(1/\epsilon\right)\right)\log(1/\epsilon)\right)$

What's so good about stochastic gradient method?

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

$$\hat{w}_\epsilon : \hat{f}(w_\epsilon) - \hat{f}(\hat{w}_{opt}) \leq \epsilon$$

$$\begin{aligned} |\hat{f}(\hat{w}_\epsilon) - f(w_{opt})| &\leq |\hat{f}(\hat{w}_\epsilon) - f(\hat{w}_\epsilon)| \\ &+ |\hat{f}(\hat{w}_\epsilon) - \hat{f}(\hat{w}_{opt})| \\ &+ |\hat{f}(\hat{w}_{opt}) - f(\hat{w}_{opt})| \\ &+ |f(\hat{w}_{opt}) - f(w_{opt})|. \end{aligned}$$

What's so good about stochastic gradient method?

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

$$\hat{w}_\epsilon : \hat{f}(w_\epsilon) - \hat{f}(\hat{w}_{opt}) \leq \epsilon$$

$$\begin{aligned} |\hat{f}(\hat{w}_\epsilon) - f(w_{opt})| &\leq O\left(\frac{1}{\sqrt{n}}\right) \\ &+ \epsilon \\ &+ O\left(\frac{1}{\sqrt{n}}\right) \\ &+ O\left(\frac{1}{\sqrt{n}}\right) \end{aligned}$$

Bousquet and Bottou '08

What's so good about stochastic gradient method?

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

$$\hat{w}_\epsilon : \ \hat{f}(w_\epsilon) - \hat{f}(\hat{w}_{opt}) \leq \epsilon$$

$$\epsilon \sim O\left(\frac{1}{\sqrt{n}}\right)$$

$$n \sim O\left(\frac{1}{\epsilon^2}\right)$$

Strongly convex case

Method	Complexity
GD	$\mathcal{O}\left(1/\epsilon^2 \kappa \log\left(1/\epsilon\right)\right)$
SGD	$\mathcal{O}\left(1/\epsilon\right)$
SVRG	$\mathcal{O}\left(1/\epsilon^2 + \kappa\right) \log\left(1/\epsilon\right)\right)$
SARAH	$\mathcal{O}\left(1/\epsilon^2 + \kappa\right) \log\left(1/\epsilon\right)\right)$

Bousquet and Bottou '08

What's so good about stochastic gradient method?

$$\hat{w}_{opt} = \arg\min_{w \in R^m} \hat{f}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

$$w_{opt} = \arg\min_{w} f(w) = \mathbb{E}[\ell(p(w, x), y)]$$

$$\hat{w}_\epsilon : \hat{f}(w_\epsilon) - \hat{f}(\hat{w}_{opt}) \leq \epsilon$$

$$\epsilon \sim O\left(\frac{1}{\sqrt{n}}\right)$$

$$n \sim O\left(\frac{1}{\epsilon^2}\right)$$

Convex case

Method	Complexity
GD	$\mathcal{O}\left(1/\epsilon^3\right)$
SGD	$\mathcal{O}\left(1/\epsilon^2\right)$
SVRG	$\mathcal{O}\left(1/\epsilon^2\right))$
SARAH	$\mathcal{O}\left((1/\epsilon^2 + 1/\epsilon)\log(1/\epsilon)\right)$

Is stochastic gradient the best we can do? And what about nonconvex problems...?

To be continued by Frank Curtis

Optimization Methods for Supervised Machine Learning: From Linear Models to Deep Learning, Part II

Frank E. Curtis, Lehigh University

joint work with

Katya Scheinberg, Lehigh University

INFORMS Annual Meeting, Houston, TX, USA

23 October 2017

Outline

Outline

Deep Neural Networks

Nonconvex Optimization

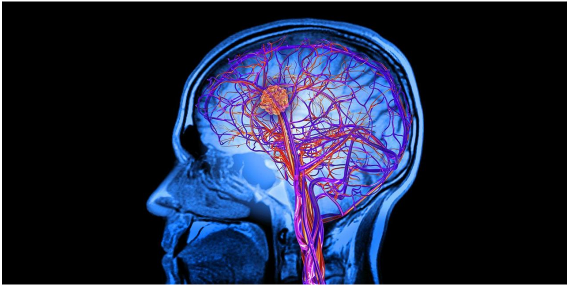
Second-Order Methods

Thanks

What is a neural network?

What is a neural network?

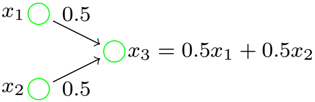
- A computer brain (artificial intelligence!)



http://www.bbc.com/future/tags/brain

What is a neural network?

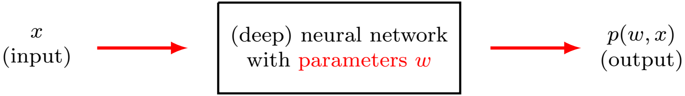
- A computer brain (artificial intelligence!)
- A computational graph
  - . . . defined using neuroscience jargon (e.g., node $\equiv$ neuron)



https://www.extremetech.com/extreme/151696-ibm-on-track-to-building-artificial-synapses

What is a neural network?

- A computer brain (artificial intelligence!)
- A computational graph
  - ...defined using neuroscience jargon (e.g., node ≡ neuron)
- A function! ...defined by some parameters



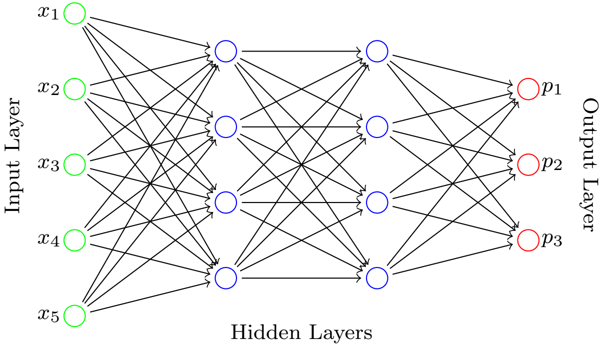
Learning

Neural networks do not learn on their own.

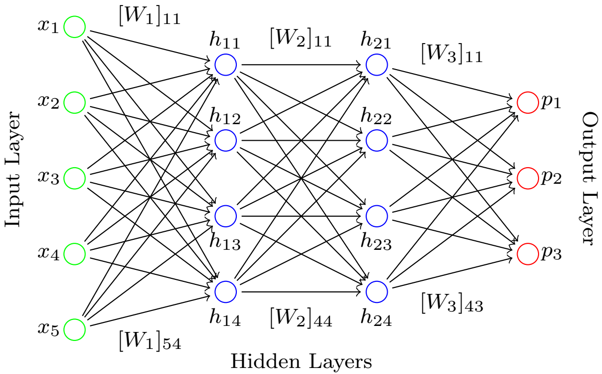
- In supervised learning, we train them by giving them inputs...
- ...and use optimization to better match their outputs to known outputs.
- (After, we hope they give the right outputs when they are unknown!)

We optimize the parameters ≡ weights ≡ decision variables.

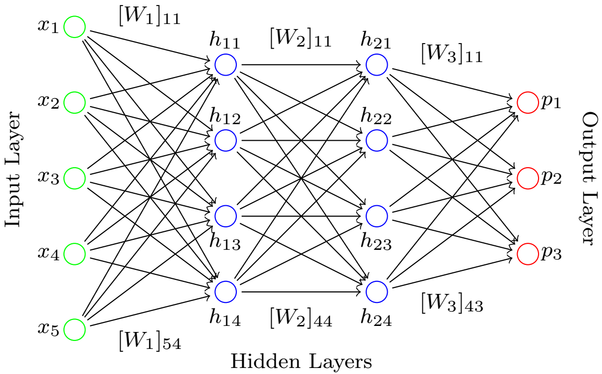
Feed forward network, fully connected



Feed forward network, fully connected

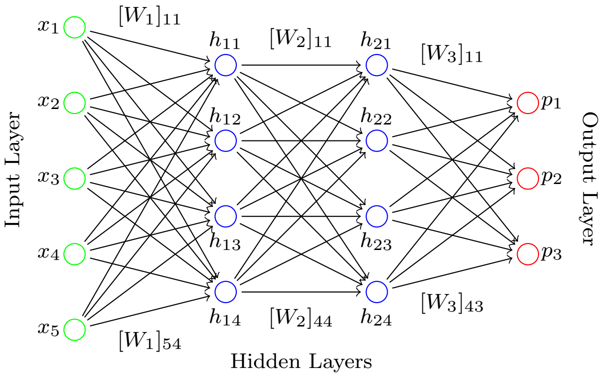


Feed forward network, fully connected



$$h_1 = s_1(W_1 x + \omega_1)$$

Feed forward network, fully connected



$$p(w, x) = s_3(W_3(s_2(W_2(s_1(W_1 x + \omega_1)) + \omega_2)) + \omega_3)$$

Training

As before, we have an optimization problem of the form

$$\min_{w \in \mathcal{W}} \mathbb{E}[\ell(p(w, x), y)]$$

or, with training data, of the form

$$\min_{w \in \mathcal{W}} \frac{1}{n} \sum_{i=1}^{n} \ell(p(w, x_i), y_i)$$

where

$$p(w, x) = s_3(W_3(s_2(W_2(s_1(W_1 x + \omega_1)) + \omega_2)) + \omega_3)$$

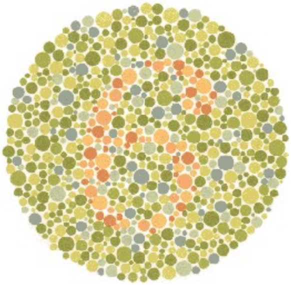
Example: Image classification

Humans can easily determine digits/letters from arrangements of pixels

Bottou et al., Optimization Methods for Large-Scale Machine Learning, *SIAM Review* (to appear)

Example: Image classification

Humans can easily determine digits/letters from arrangements of pixels



. . . for the most part.

(I'm told there's a number there!)

---

https://colormax.org/color-blind-test/

Convolutional neural networks

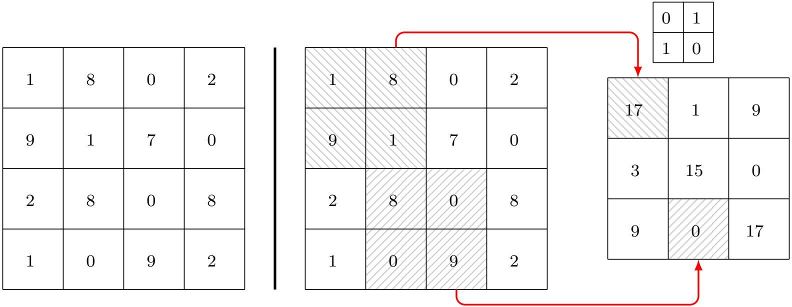
A modern tool for image classification is a convolutional neural network (CNN)

- These work by trying to capture spatial relationships between input values

Convolutional neural networks

A modern tool for image classification is a convolutional neural network (CNN)

- These work by trying to capture spatial relationships between input values
- For example, in the example below, a filter is applied—to compute the sum of elementwise products—to look for a diagonal pattern



- Here, the data is a matrix, but these can be translated to vector operations.

Anjelica Huston (not Houston!)

A random filter simply blurs the data, which doesn't help

https://www.filmcomment.com/article/interview-anjelica-huston

Anjelica Huston (not Houston!)

. . . but certain filters can reveal edges and other features

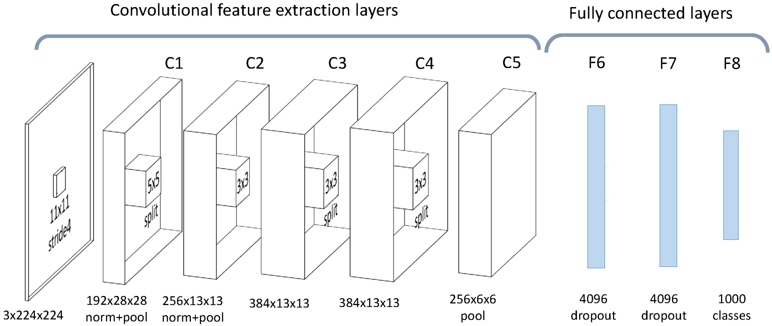
(There are plenty of Python tools for playing around like this.)

https://www.filmcomment.com/article/interview-anjelica-huston

Large-scale network

A full large-scale network involves various other components/tools:

- rectification, normalization, pooling, regularization, etc.

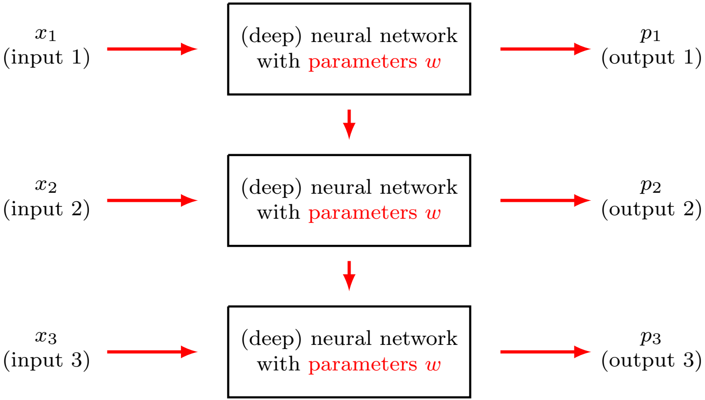


This network involves over 60 million parameters. Need good algorithms!

Bottou et al., Optimization Methods for Large-Scale Machine Learning, *SIAM Review* (to appear)

Recurrent neural networks

These try to capture temporal relationships between input values.



- Video classification, speech recognition, text classification, etc.

Outline

Deep Neural Networks

Nonconvex Optimization

Second-Order Methods

Thanks

Back propagation

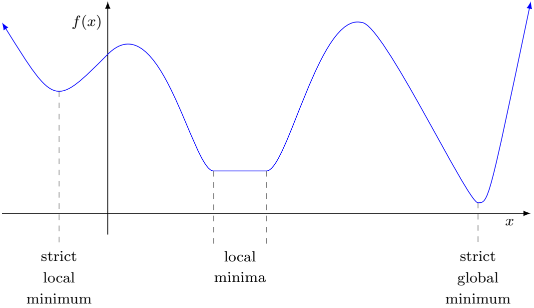
How do we optimize?

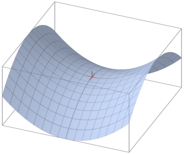
- Same as always!
- Compute derivatives, but how?
- Back propagation, i.e., automatic differentiation
- Then we need an optimization algorithm in which to use them.

Main challenges:

- "Full gradient" involves loop over all data, which is expensive
- . . . so consider stochastic methods, as previously mentioned.
- However, these problems are large-scale and nonconvex.

Global minima, local minima, saddle points, etc.





- These textbook illustrations might be misleading.
- The “landscape” of the objective function defined by a deep neural network is something of great interest these days.

https://upload.wikimedia.org/wikipedia/commons/4/40/Saddle_point.png

How to optimize?

It is not clear where a gradient-based method might converge.

- However, (stochastic) gradient-based methods seem to work well!
- They provably avoid saddle points with high probability.
- ... and often converge to "good" stationary points.

How to optimize?

It is not clear where a gradient-based method might converge.

- However, (stochastic) gradient-based methods seem to work well!
- They provably avoid saddle points with high probability.
- . . . and often converge to "good" stationary points.

Open questions:

- How to characterize the behavior of different methods?
- How to characterize the generalization properties of solutions?
- What algorithms are the most effective at finding points with good generalization properties?

We will not answer these; instead, we'll simply describe/motivate some methods.

Outline

Deep Neural Networks

Nonconvex Optimization

Second-Order Methods

Thanks

First- versus second-order

First-order methods follow a steepest descent methodology:

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f(w_k)$$

Second-order methods follow Newton's methodology:

$$w_{k+1} \leftarrow w_k - \alpha_k [\nabla^2 f(w_k)]^{-1} \nabla f(w_k),$$

which one should view as minimizing a quadratic model of f at w_k:

$$f(w_k) + \nabla f(w_k)^T (w - w_k) + \tfrac{1}{2}(w - w_k)^T \nabla^2 f(w_k)(w - w_k)$$

First- versus quasi-second-order

First-order methods follow a steepest descent methodology:

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f(w_k)$$

Second-order methods follow Newton's methodology:

$$w_{k+1} \leftarrow w_k - \alpha_k M_k \nabla f(w_k),$$

which one should view as minimizing a quadratic model of f at w_k:

$$f(w_k) + \nabla f(w_k)^T (w - w_k) + \tfrac{1}{2}(w - w_k)^T H_k (w - w_k)$$

Might also replace the Hessian with an approximation H_k with inverse M_k

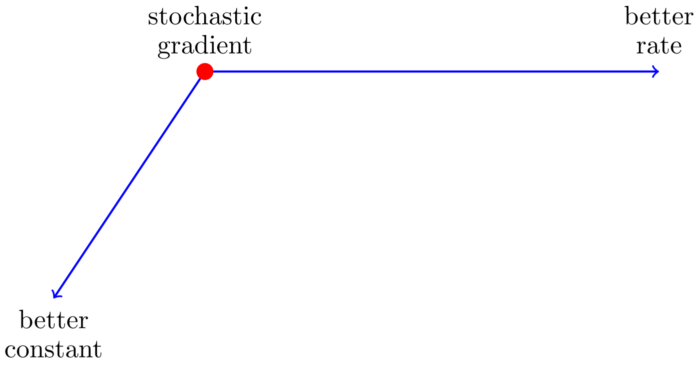
Why second-order?

Second-order methods are expensive!

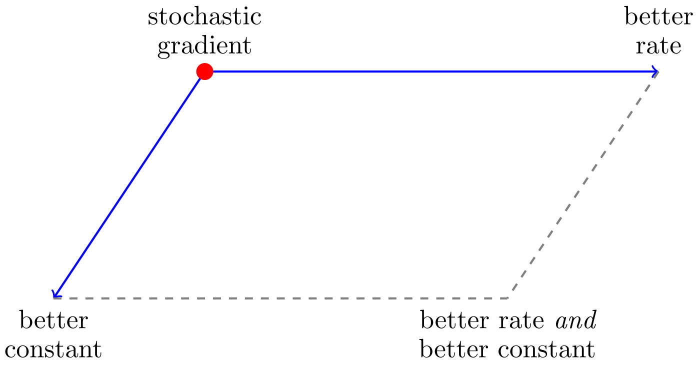
- Yes, but judicious use of second-order information can help
- . . . and the resulting methods can be made nearly as cheap as SG.

Overall, there are various ways to improve upon SG. . .

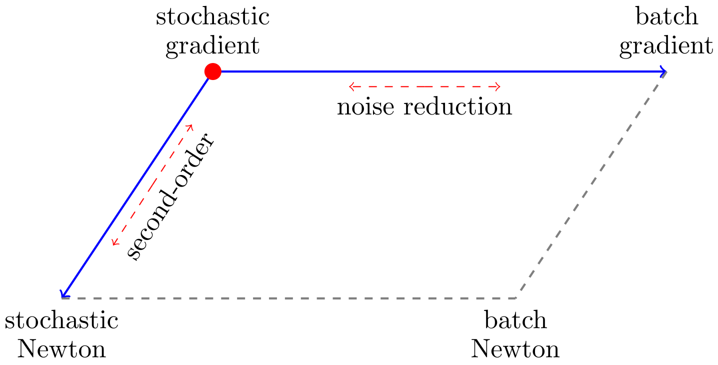
What can be improved?



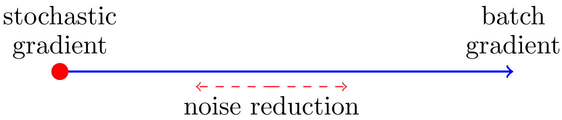
What can be improved?



Two-dimensional schematic of methods

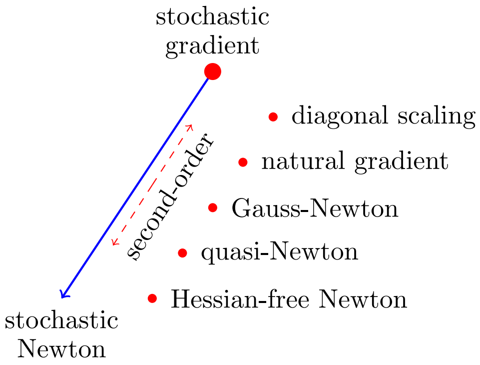


2D schematic: Noise reduction methods



- dynamic sampling
- gradient aggregation
- iterate averaging

2D schematic: Second-order methods



So, why second-order?

Traditional motivation: fast local convergence guarantees

- Hard to achieve in large-scale stochastic settings

So, why second-order?

Traditional motivation: fast local convergence guarantees

- Hard to achieve in large-scale stochastic settings

Recent motivation (last few years): better complexity properties

- Many are no better than first-order methods in terms of complexity
- ...and ones with better complexity aren't necessarily best in practice (yet)

So, why second-order?

Traditional motivation: fast local convergence guarantees

- Hard to achieve in large-scale stochastic settings

Recent motivation (last few years): better complexity properties

- Many are no better than first-order methods in terms of complexity
- ...and ones with better complexity aren't necessarily best in practice (yet)

Other reasons?

- Adaptive, natural scaling (gradient descent $\approx 1/L$ while Newton ≈ 1)
- Mitigate effects of ill-conditioning
- Easier to tune parameters(?)
- Better at avoiding saddle points(?)
- Better trade-off in parallel and distributed computing settings
- New algorithms! Not analyzing the same old

Framework #1: Matrix-free (Gauss-)Newton

Compute each step by applying an iterative method to solve

$$H_k s_k = -g_k$$

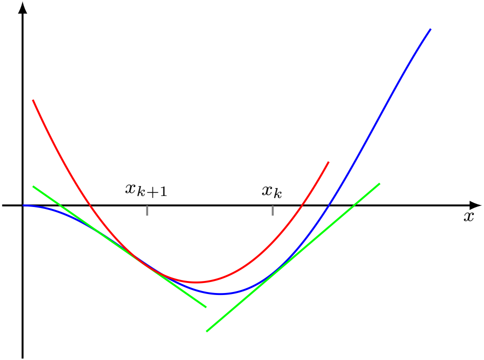
potentially with regularization, within a trust region, etc.

This can be computationally efficient since

- H_k can be defined by a subsample of data.
- Matrix-vector products can be computed without forming the matrix
- ... using similar principles as in back propagation.
- The linear system need not be solved exactly.

Quasi-Newton (deterministic setting)

Only *approximate* second-order information with gradient displacements:



Secant equation $H_k y_k = s_k$ to match gradient of f at w_k, where

$$s_k := w_{k+1} - w_k \quad \text{and} \quad y_k := \nabla f(w_{k+1}) - \nabla f(w_k)$$

Framework #2: Quasi-Newton

How can this idea be adapted to the stochastic setting?

- Idea #1: Replace y_k by displacement using the same sample, i.e.,

$$\nabla_{\mathcal{S}_k} f(w_{k+1}) - \nabla_{\mathcal{S}_k} f(w_k).$$

  (This doubles the number of stochastic gradients, but maybe worthwhile?)

- Idea #2: Replace y_k by action on a (subsampled) Hessian, i.e.,

$$\nabla^2_{\mathcal{S}_k^H} f(w_{k+1}) s_k$$

  (This requires matrix-vector products with a Hessian.)

- ...other ideas?

Outline

OptML @ Lehigh

Please visit the OptML @ Lehigh website!

- http://optml.lehigh.edu