Optimization and Stochastic Gradient Descent

Applied Machine Learning
Derek Hoiem

Dall-E

Machine learning optimization

	Optimization	Solution Depends on Initialization or Randomized Optimization?	Optimization Strategy is Important to Effectiveness?
KNN	N/A	No	No
K-means	Coordinate Descent	Yes	No
Linear Regression	Iterative	No	No
Logistic Regression	Iterative	No	No
Linear SVM	Iterative	No	No
Kernelized SVM	Iterative	No	No
EM Algorithm	Coordinate Descent	Yes	No
Decision Tree	Greedy selection	No	No

- For methods we learned so far, one optimizer may be faster or more memory efficient than other, but they will generally be able to achieve similar solutions.

Machine learning optimization

	Optimization	Solution Depends on Initialization or Randomized Optimization?	Optimization Strategy is Important to Effectiveness?
KNN	N/A	No	No
K-means	Coordinate Descent	Yes	No
Linear Regression	Iterative	No	No
Logistic Regression	Iterative	No	No
Linear SVM	Iterative	No	No
Kernelized SVM	Iterative	No	No
EM Algorithm	Coordinate Descent	Yes	No
Decision Tree	Greedy selection	No	No
MLPs, Deep Networks	**Iterative**	**Yes**	**Yes**

- For methods we learned so far, one optimizer may be faster or more memory efficient than other, but they will generally be able to achieve similar solutions.
- For MLPs and deep networks, optimization is an important part of design.

This lecture

1. Batch gradient descent
2. PEGASOS: Stochastic Gradient Descent for SVM
3. Perceptrons

Gradient descent

```
gradient_descent(f’(x), x0, lr, niter)
  x = x0
  for t in range(niter):
    x = x – lr*f’(x)
  return x
```

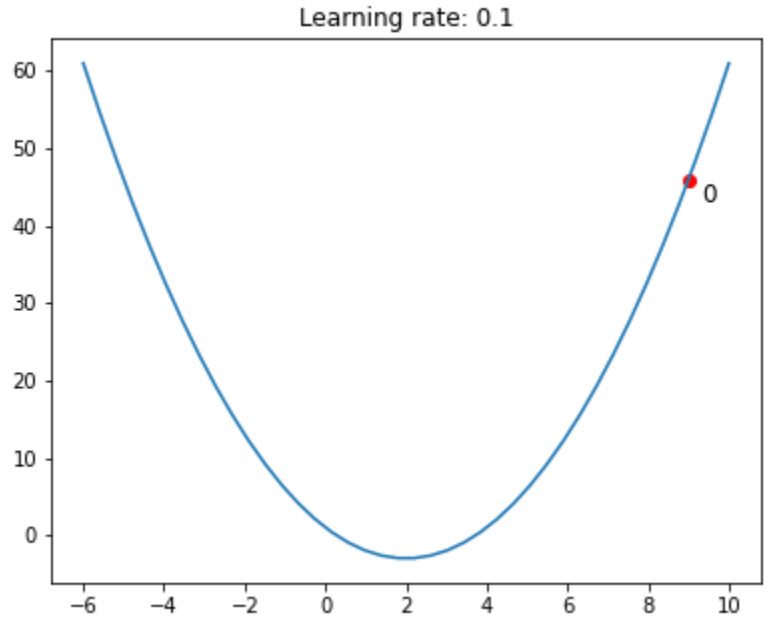
Gradient descent

```
gradient_descent(f’(x), x0, lr, niter)
   x = x0
   for t in range(niter):
      x = x - lr*f’(x)
   return x
```

Example:

$$f(x) = x^2 - 4x + 1$$

$$f'(x) = 2x - 4$$



Example: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

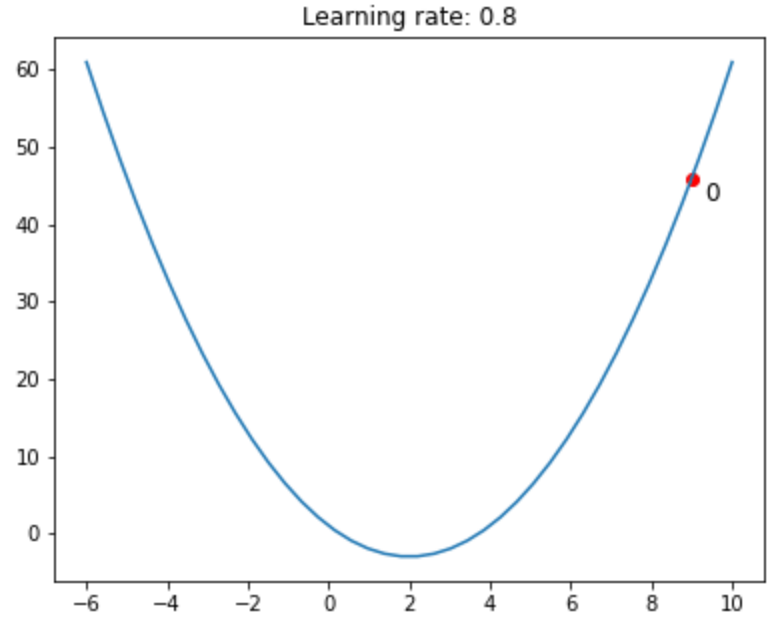
Gradient descent

```
gradient_descent(f’(x), x0, lr, niter)
   x = x0
   for t in range(niter):
      x = x - lr*f’(x)
   return x
```

Example:

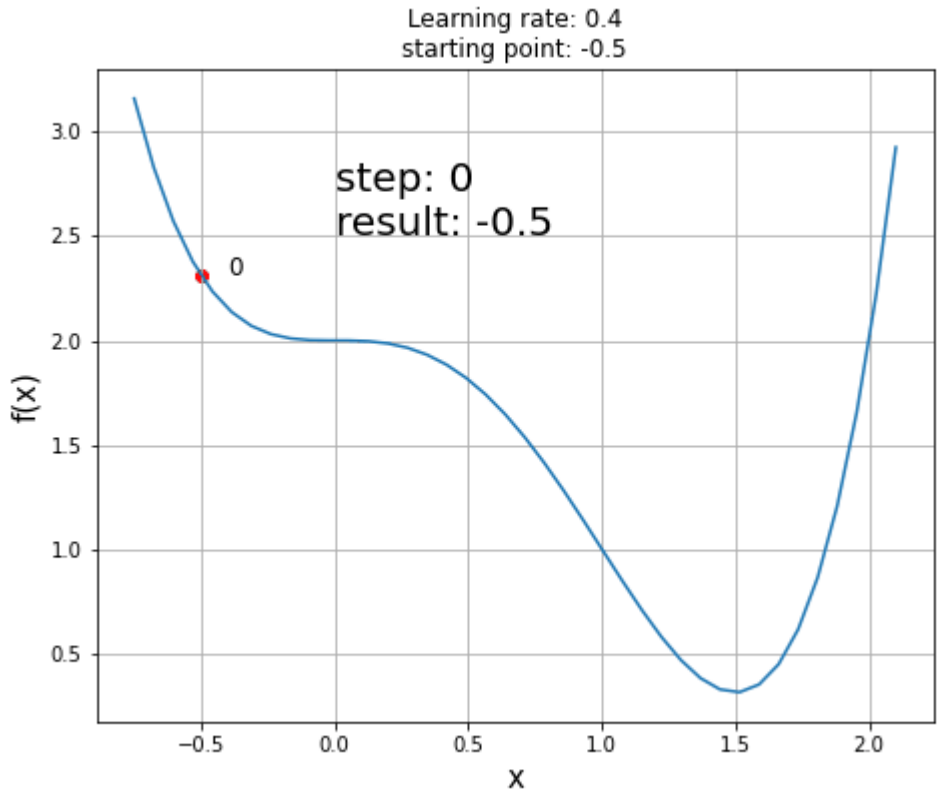
$$f(x) = x^2 - 4x + 1$$

$$f'(x) = 2x - 4$$

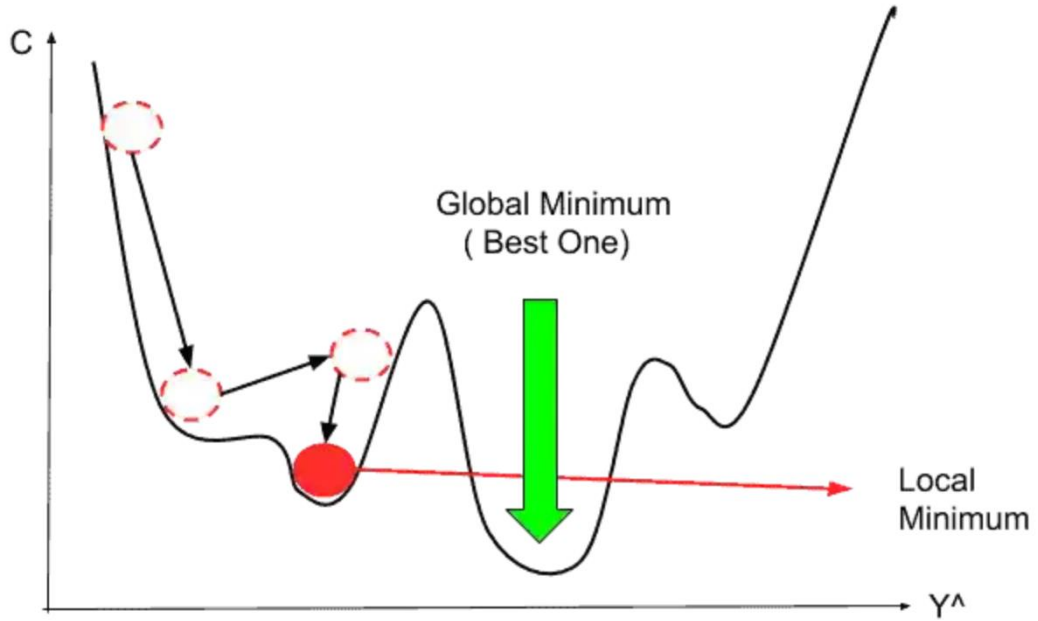


Example: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Gradient descent challenge cases



Saddle points (gradient = 0 in some parts of solution space)



Multiple local minima

Many models we've learned so far (e.g., SVM, logistic regression, linear regression) are convex, so they don't have these challenges.

Example: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21 Fig: https://www.mltut.com/stochastic-gradient-descent-a-super-easy-complete-guide/

Gradient Descent Visualization with Local Minima

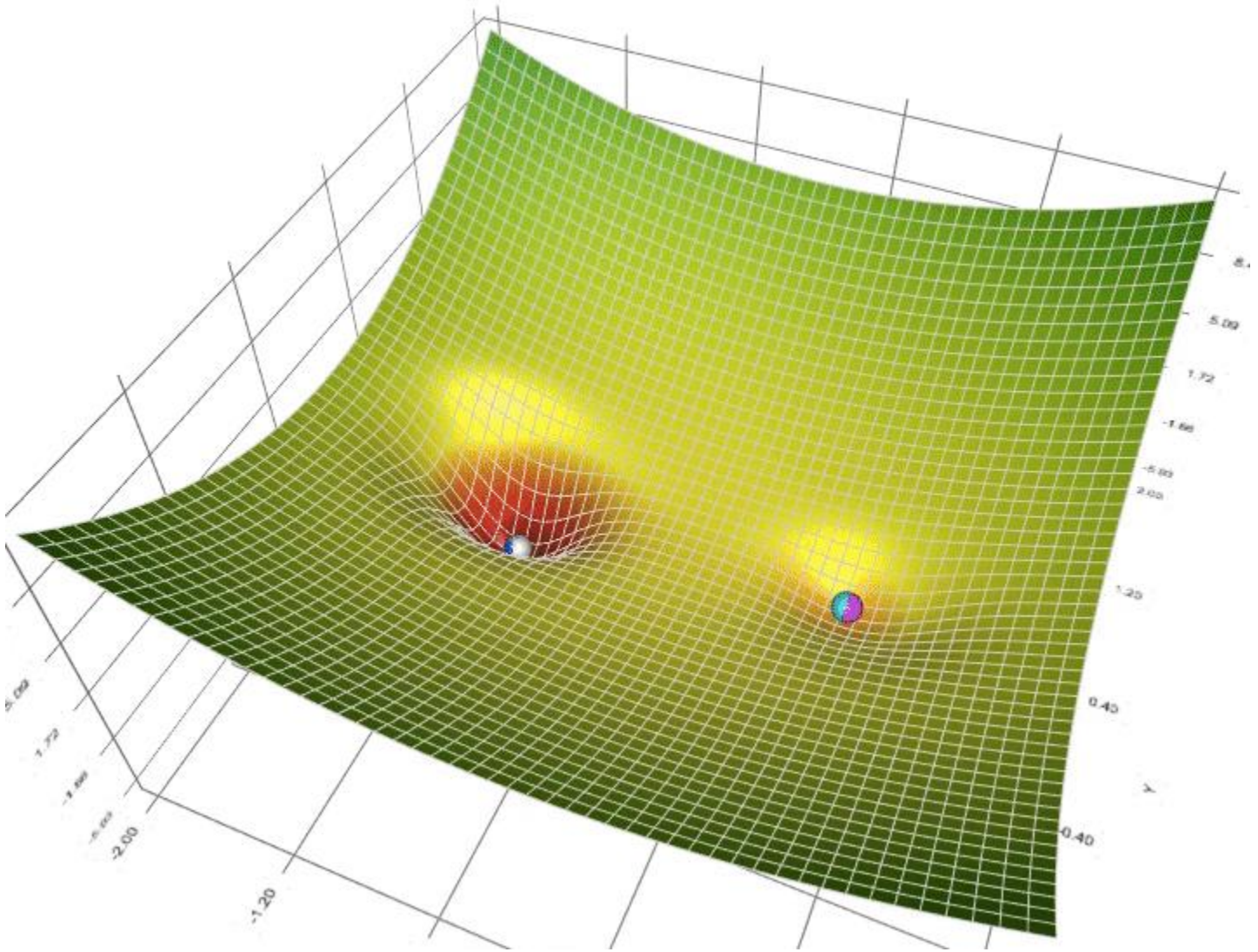


Figure source

Learning rates and learning schedules

- Learning rate = step size that is multiplied by gradient direction/magnitude
  - Large rate allows big movements toward optimum but might over-step
  - Small rate is less likely to over-step but could take longer

- Learning schedule: change learning rate over time
  - Constant
  - Exponential decay, e.g. lr = lr * 0.95
  - Linear, e.g. lr = lr0 *(1 – iter / max_iter)

SVM Formulation

Prediction

$$y_n = \text{sign}(\boldsymbol{w}^T x_n + b)$$

Optimization

Known as “hinge loss”
Penalty is paid if margin is less than 1

$$w^* = \underset{\boldsymbol{w}}{\text{argmin}} \left[\|w\|^2 + C \sum_n^N \max(0, 1 - y_n(\boldsymbol{w}^T x_n + b)) \right]$$

Here, $y \in \{-1,1\}$ which is a common convention that simplifies notation for binary classifiers

Gradient descent with SVM

```
gradient_descent(f’(w,i,x,y), lr, niter)
  w = zeros(x.shape[1],)
  for t in range(niter):
    for i in range(len(w)):
      w[i] = w[i] – lr*f’(w,i,x,y)
  return x
```

$$f(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{w}\|^2 + C \sum_{n}^{N} \max(0, 1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n))$$

$$f'(\boldsymbol{w}, i, \boldsymbol{x}, \boldsymbol{y}) = 2w_i + C \sum_{n} -\delta(y_n(\boldsymbol{w}^T \boldsymbol{x}_n) < 1) y_n x_{ni}$$

Only examples with score of correct answer less than 1 contribute to the gradient

Pegasos: Primal Estimated sub-GrAdient SOlver for SVM (2011)

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{(\mathbf{x},y)\in S} \ell(\mathbf{w};(\mathbf{x},y))$$

$$\ell(\mathbf{w};(\mathbf{x},y)) = \max\{0, 1 - y\,\langle \mathbf{w},\mathbf{x}\rangle\}$$

SVM problem that we want to solve (Minimize weights square + sum of hinge losses on all samples)

$$f(\mathbf{w};i_t) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \ell(\mathbf{w};(\mathbf{x}_{i_t},y_{i_t}))$$

Problem in terms of **one** sample

$$\nabla_t = \lambda\,\mathbf{w}_t - \mathbb{1}[y_{i_t}\,\langle \mathbf{w}_t,\mathbf{x}_{i_t}\rangle < 1]\,y_{i_t}\mathbf{x}_{i_t}$$

Gradient in terms of one sample

- Direction to move to improve solution

https://home.ttic.edu/~nati/Publications/PegasosMPB.pdf

Pegasos algorithm: Stochastic Gradient Descent (SGD)

INPUT: S, λ, T

INITIALIZE: Set $\mathbf{w}_1 = 0$

FOR $t = 1, 2, \ldots, T$

Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.

Set $\eta_t = \frac{1}{\lambda t}$

If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:

Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$

Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):

Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$

OUTPUT: $\mathbf{w}_{T+1}$

Notation

S: training set

λ: regularization weight

T: number iterations

$\boldsymbol{w}_t$: model weights

$\boldsymbol{x}_{i_t}$: features for example i_t

y_{i_t}: label for example i_t

η_t: step size (“learning rate”)

Pegasos with **mini-batch**

- Calculating gradient based on multiple examples reduces variance of gradient estimate

INPUT: S, λ, T, k
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
  Choose $A_t \subseteq [m]$, where $|A_t| = k$, uniformly at random
  Set $A_t^+ = \{i \in A_t : y_i \langle \mathbf{w}_t, \mathbf{x}_i \rangle < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \frac{\eta_t}{k} \sum_{i \in A_t^+} y_i \mathbf{x}_i$

OUTPUT: $\mathbf{w}_{T+1}$

k: batch size
m: number of training samples
A_t: batch of examples
A_t^+: examples within margin

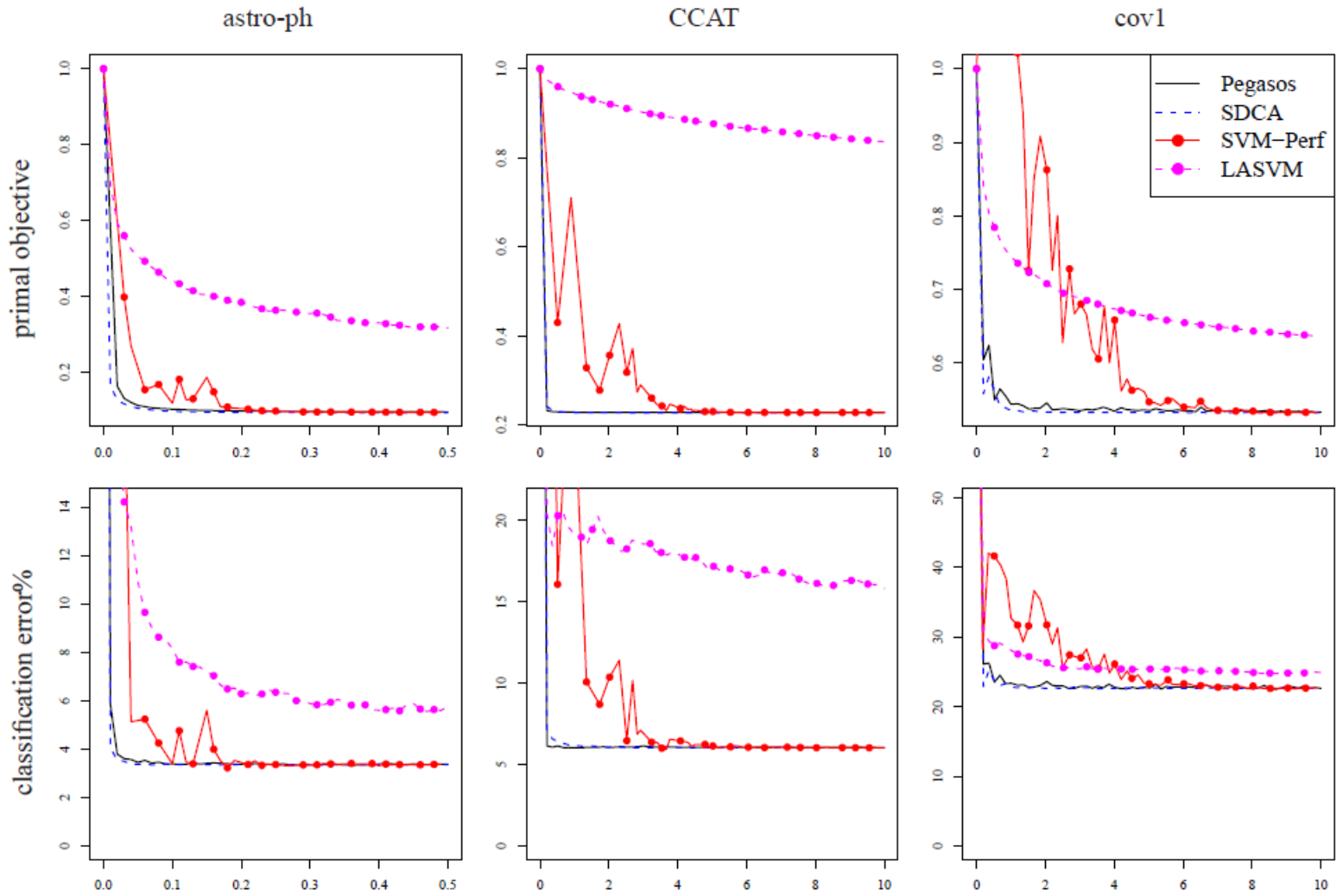
S: training set
λ: regularization weight
T: number iterations
$\boldsymbol{w}_t$: model weights
$\boldsymbol{x}_i$: features for example i
y_i: label for example i
η_t: step size (“learning rate”)

SGD applies to many losses

	Loss function	Subgradient
SVM (hinge loss)	$\ell(z, y_i) = \max\{0, 1 - y_i z\}$	$\mathbf{v}_t = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i z < 1 \\ \mathbf{0} & \text{otherwise} \end{cases}$
Logistic regression / sigmoid loss	$\ell(z, y_i) = \log(1 + e^{-y_i z})$	$\mathbf{v}_t = -\frac{y_i}{1+e^{y_i z}} \mathbf{x}_i$
Hinge L1 regression	$\ell(z, y_i) = \max\{0, \lvert y_i - z \rvert - \epsilon\}$	$\mathbf{v}_t = \begin{cases} \mathbf{x}_i & \text{if } z - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - z > \epsilon \\ \mathbf{0} & \text{otherwise} \end{cases}$
Margin loss between scores of most likely and correct label	$\ell(z, y_i) = \max_{y \in \mathcal{Y}} \delta(y, y_i) - z_{y_i} + z_y$	$\mathbf{v}_t = \phi(\mathbf{x}_i, \hat{y}) - \phi(\mathbf{x}_i, y_i)$ where $\hat{y} = \arg\max_y \delta(y, y_i) - z_{y_i} + z_y$
Variant of a logistic loss	$\ell(z, y_i) = \log\left(1 + \sum_{r \neq y_i} e^{z_r - z_{y_i}}\right)$	$\mathbf{v}_t = \sum_r p_r \phi(\mathbf{x}_i, r) - \phi(\mathbf{x}_i, y_i)$ where $p_r = e^{z_r} / \sum_j e^{z_j}$

z is the score for y=1

SGD is fast compared to other optimization approaches



Dataset	Training Size	Testing Size	Features	Sparsity	λ
astro-ph	29882	32487	99757	0.08%	5×10^{-5}
CCAT	781265	23149	47236	0.16%	10^{-4}
cov1	522911	58101	54	22.22%	10^{-6}

SDCA = stochastic dual coordinate descent, another form of sub-gradient optimization that chooses learning rate dynamically

Fig. 4 Comparison of linear SVM optimizers. Primal suboptimality (top row) and testing classification error (bottom row), for one run each of Pegasos, stochastic DCA, SVM-Perf, and LASVM, on the astro-ph (left), CCAT (center) and cov1 (right) datasets. In all plots the horizontal axis measures runtime in seconds.

Experiments with Linear SVM

Dataset	Training Size	Testing Size	Features	Sparsity	λ
astro-ph	29882	32487	99757	0.08%	5×10^{-5}
CCAT	781265	23149	47236	0.16%	10^{-4}
cov1	522911	58101	54	22.22%	10^{-6}

Training time and test error

Dataset	Pegasos	SDCA	SVM-Perf	LASVM
astro-ph	$0.04s$ (3.56%)	$0.03s$ (3.49%)	$0.1s$ (3.39%)	$54s$ (3.65%)
CCAT	$0.16s$ (6.16%)	$0.36s$ (6.57%)	$3.6s$ (5.93%)	$> 18000s$
cov1	$0.32s$ (23.2%)	$0.20s$ (22.9%)	$4.2s$ (23.9%)	$210s$ (23.8%)

Effect of mini-batch size

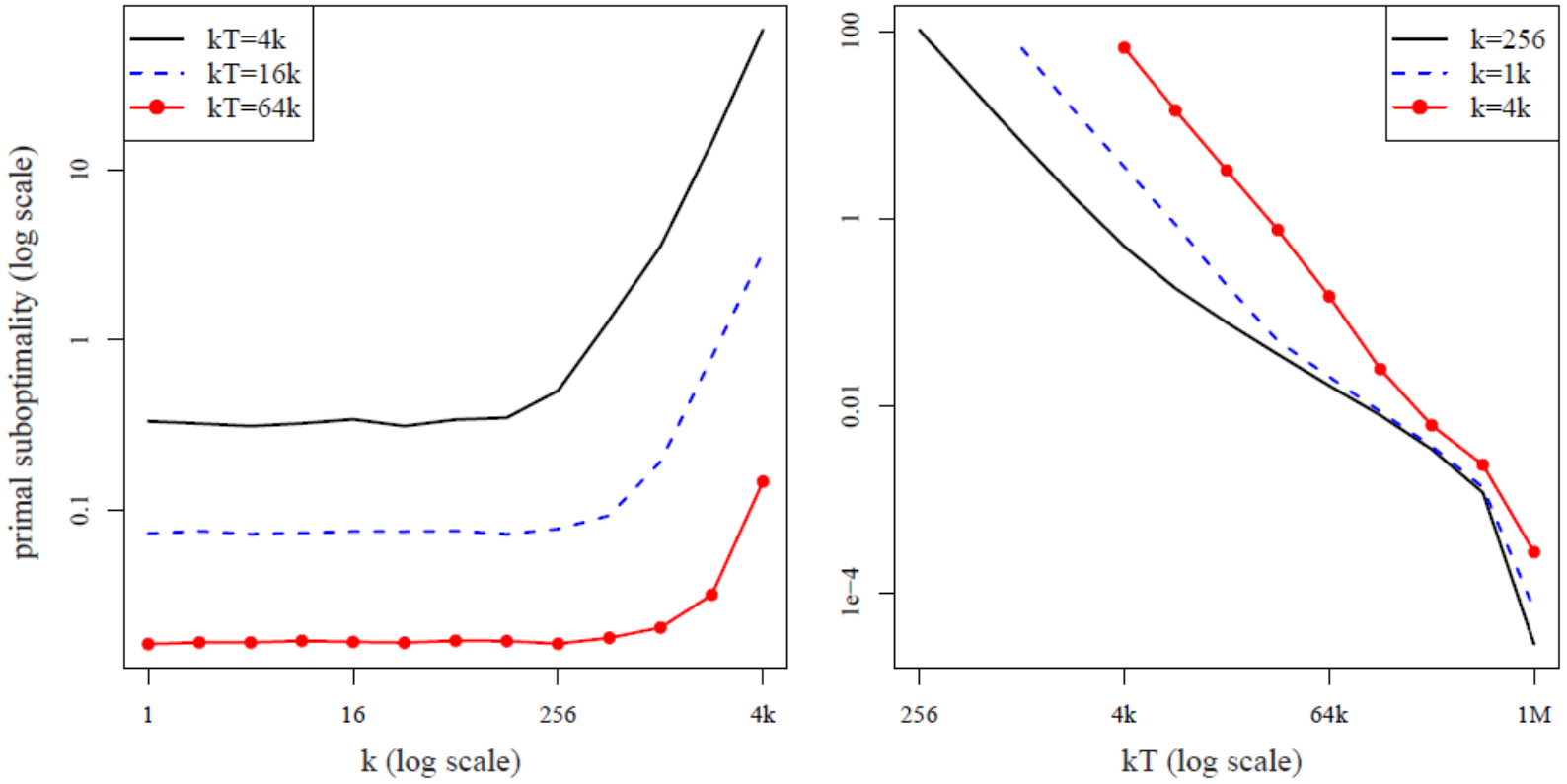
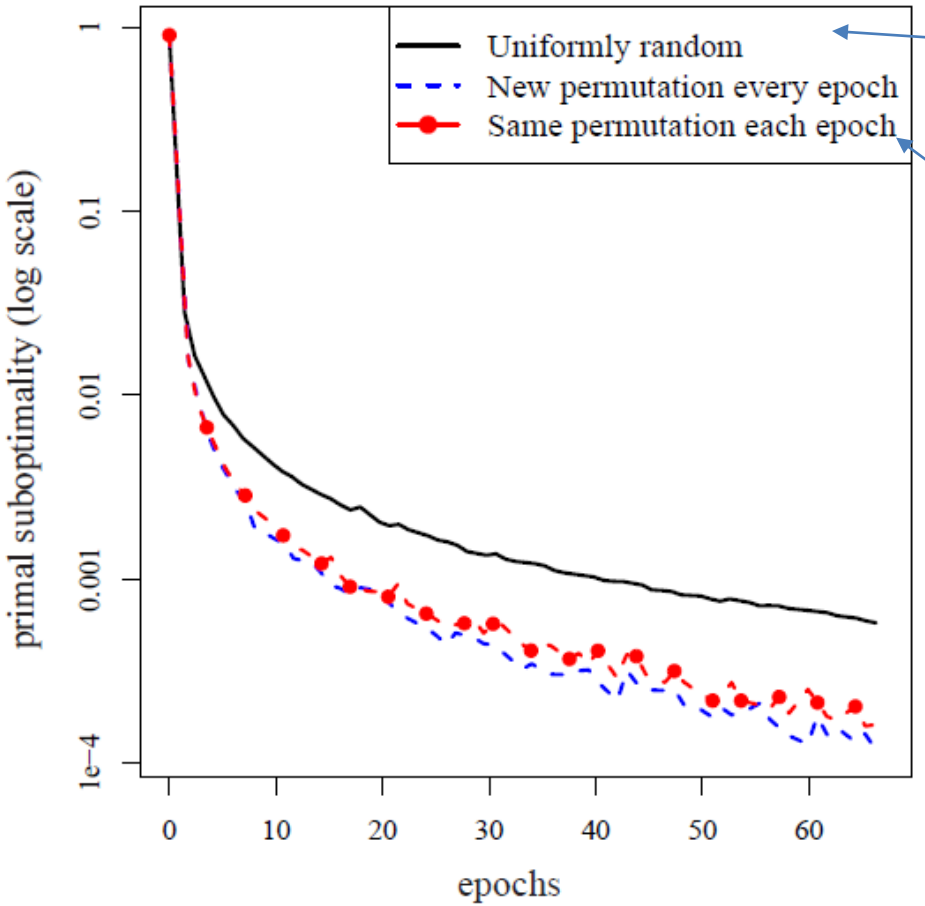


Fig. 7 The effect of the mini-batch size on the runtime of Pegasos for the astro-ph dataset. The first plot shows the primal suboptimality achieved for certain fixed values of overall runtime kT, for various values of the mini-batch size k. The second plot shows the primal suboptimality achieved for certain fixed values of k, for various values of kT. Very similar results were achieved for the CCAT dataset.

Effect of sampling procedure: randomly ordered epochs is best



Sampling with replacement

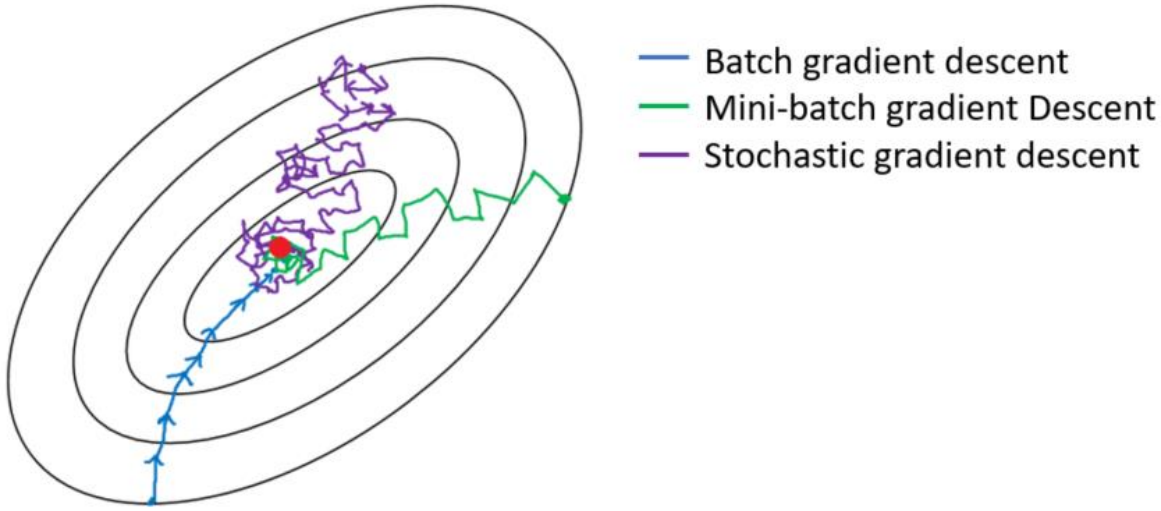
Use different random order for each “epoch”

Use same order for each epoch

Epoch: one run through the training set

Mini-Batch SGD vs. Full Batch Gradient Descent

- Mini-batch is faster
  - Time to compute gradient is $O(B)$ for batch size B, but standard error of gradient direction is $O\left(1/\sqrt{B}\right)$
  - E.g. batch size of 10000 vs 100 will take 100 times longer but reduce standard deviation by factor of 10
- Full batch is more stable, but the instability of SGD can help escape local minima
- We'll discuss enhancements to SGD, such as momentum later
- SGD training is highly parallelizable (good for GPU processing)



https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4

Pegasos: take-ways and surprising facts

- SGD is very simple and effective optimization algorithm – step toward better solution based on a small sample of training data
- Not very sensitive to mini-batch size (but larger batches can be much faster with GPU parallel processing)
- The same learning schedule is effective across several problems
- A *larger training set* makes it *faster* to obtain the same test performance

Review questions

http://tinyurl.com/cs441sgd

(participation in all remaining forms worth total of 20 points)

True or False:

- Unlike SVM, linear logistic regression loss always adds a non-zero penalty over all training data points.

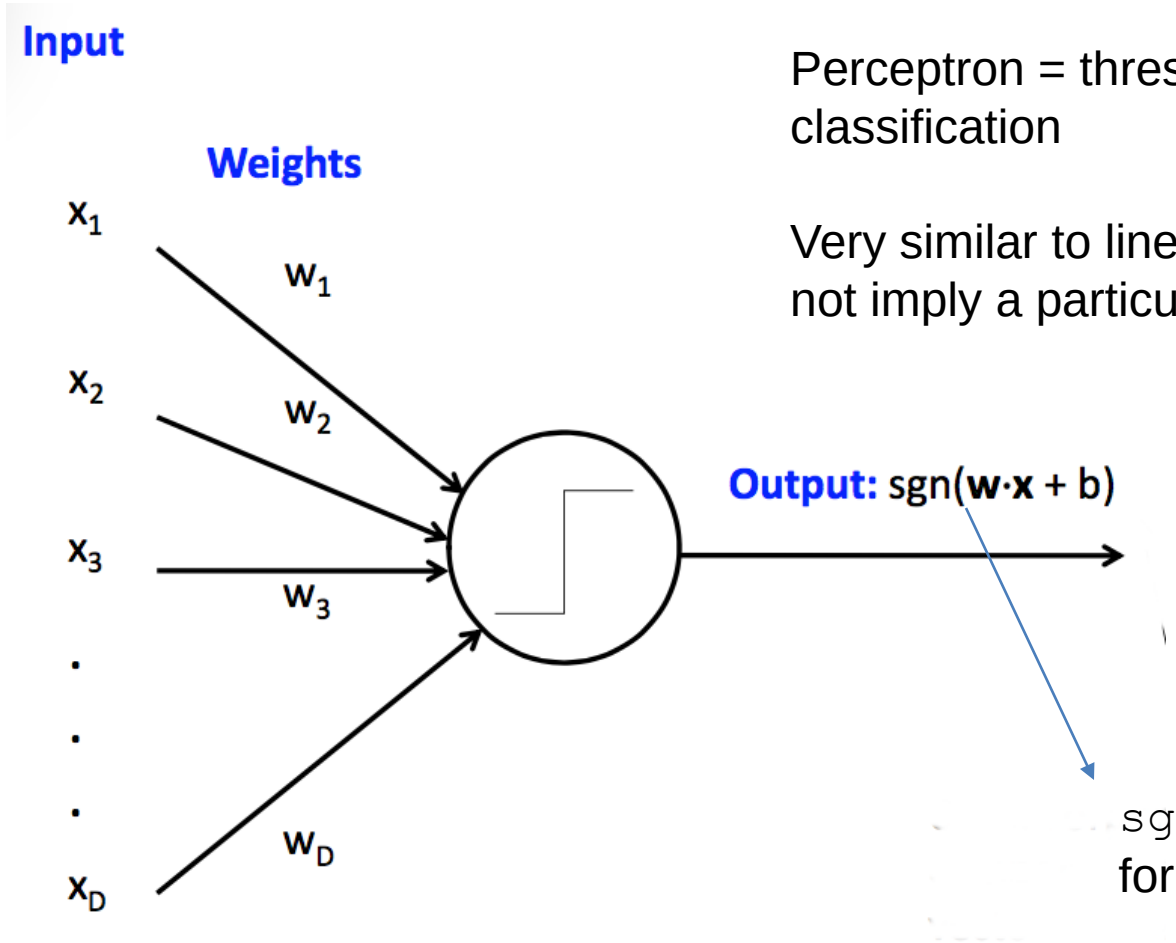
- The PEGASUS algorithm computes the gradient for the optimization algorithm using only one sample out of the training data points – instead of using the whole dataset – thus increasing its computational efficiency.

- PEGASUS has the disadvantage that the larger the training dataset, the slower it can be optimized to reach a particular test error.

Which of these algorithms' objective functions have a single local optimum?

– Logistic regression, linear regression, linear SVM, kernelized SVM, EM algorithm, K-means

Perceptron



Perceptron = thresholded linear prediction model for classification

Very similar to linear logistic regression, though perceptron does not imply a particular error or training objective

`sgn` returns -1 for negative inputs and +1 for positive inputs

Fig source: CS 440

Perceptron Update Rule

Prediction: $f(\boldsymbol{x}) = w_0 x_0 + w_1 x_1 + \ldots w_m x_m + b$

Error: $E(\boldsymbol{x}) = (f(\boldsymbol{x}) - y)^2$

prediction target

Update w_i: take a step to decrease $E(\boldsymbol{x})$

$$\frac{\partial E(\boldsymbol{x})}{\partial w_i} = 2(f(\boldsymbol{x}) - y)\left[\frac{\partial (f(\boldsymbol{x}) - y)}{\partial w_i}\right]$$

$$\frac{\partial E(\boldsymbol{x})}{\partial w_i} = 2(f(\boldsymbol{x}) - y)x_i$$

$$w_i = w_i - \eta(f(\boldsymbol{x}) - y)x_i$$

Make error *lower* Learning rate

Chain Rule:
$h(x) = f(g(x))$, then
$h'(x) = f'\big(g(x)\big)g'(x)$

(the 2 is folded into the learning rate)

Perceptron Optimization by SGD

Randomly initialize weights, e.g. w ~ Gaus(mu=0, std=0.05)

For each iteration t:

Split data into batches

$\eta = 0.1/t$

For each batch X_b:

For each weight w_i:

$$w_i = w_i - \eta \frac{1}{|X_b|} \sum_{\boldsymbol{x}_n \in X_b} (f(\boldsymbol{x}_n) - y_n) x_{ni}$$

With different loss, the update changes accordingly

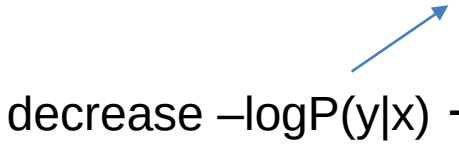
Logistic loss:

$$f(\boldsymbol{x}) = w_0 x_0 + w_1 x_1 + \ldots w_m x_m + b$$

$$P(y|\boldsymbol{x}) = \frac{1}{1+\exp(-yf(x))}, y \in \{-1,1\}$$

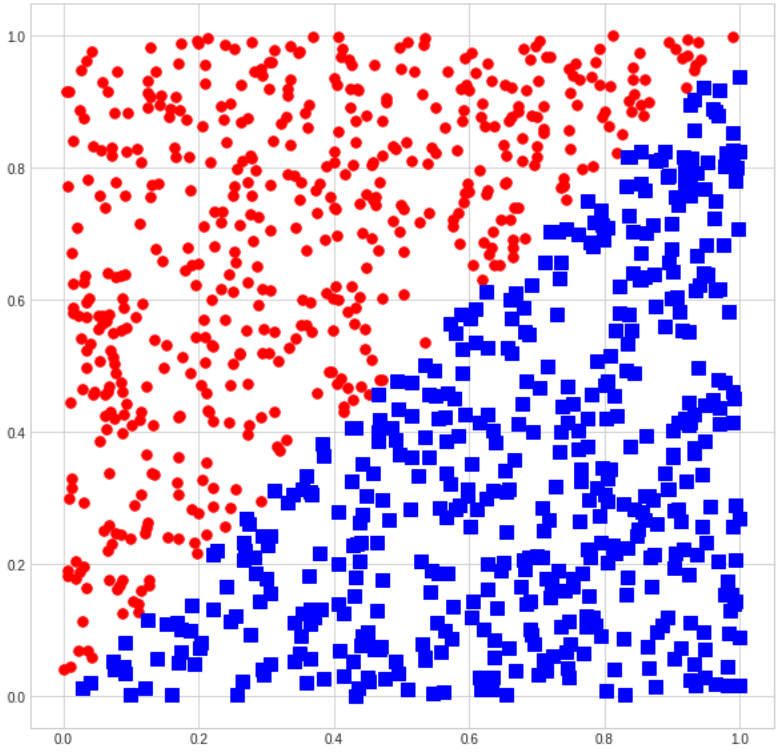
$$E(\boldsymbol{x}) = -\log P(y|\boldsymbol{x})$$

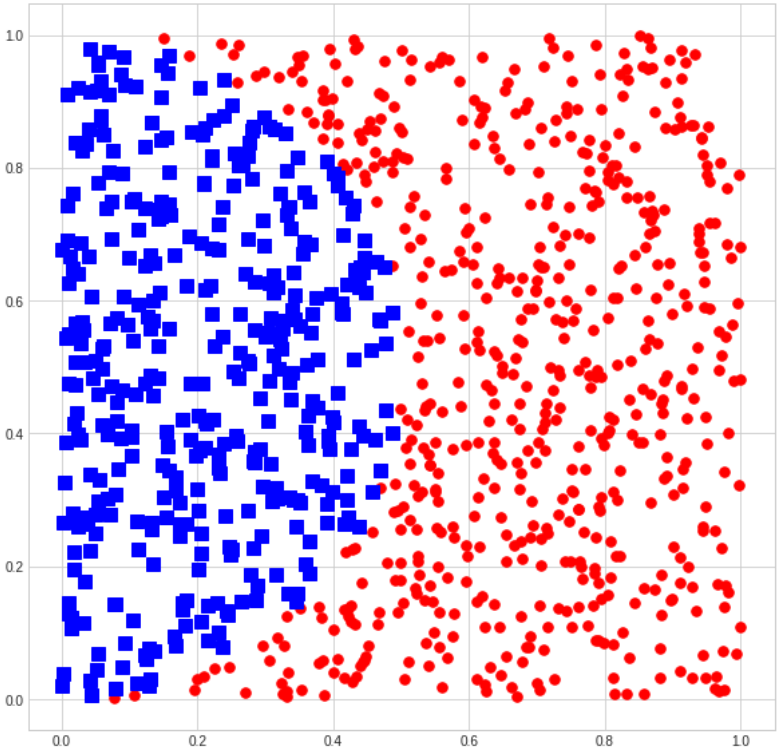
$$w_i = w_i + \eta \frac{1}{|X_b|} \sum_{\boldsymbol{x}_n \in X_b} y_n x_{ni} (1 - P(y = y_n|x_n))$$

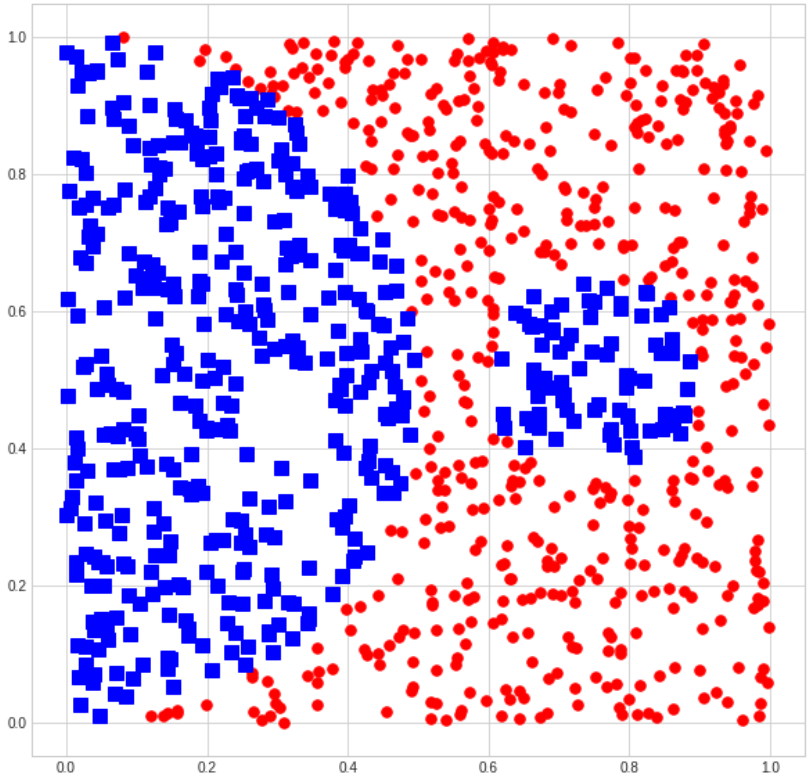


Is a perceptron enough?

Which of these can a perceptron solve (fit with zero training error)?







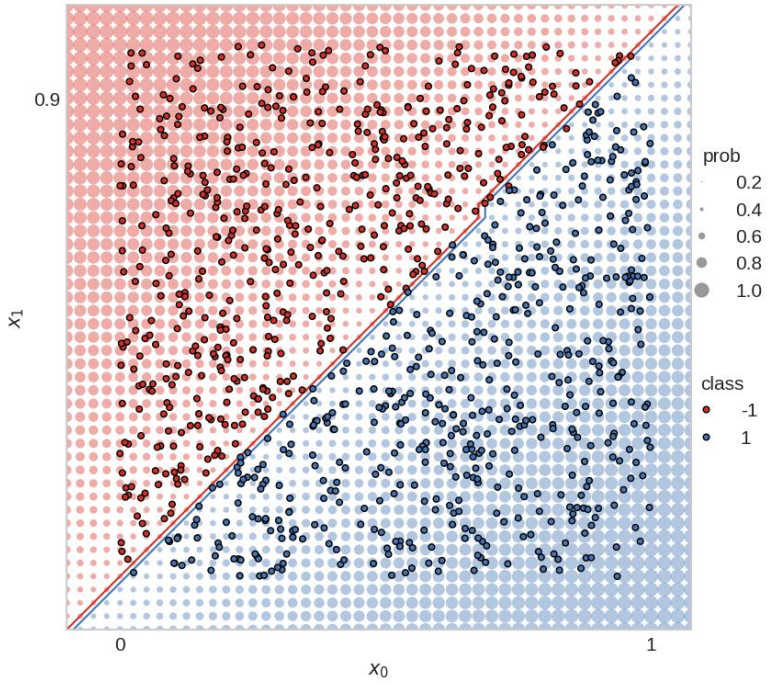
Demo

https://colab.research.google.com/drive/1nKNJyolqgzW53Rz59M2BZtyQM8bbrExb?usp=sharing

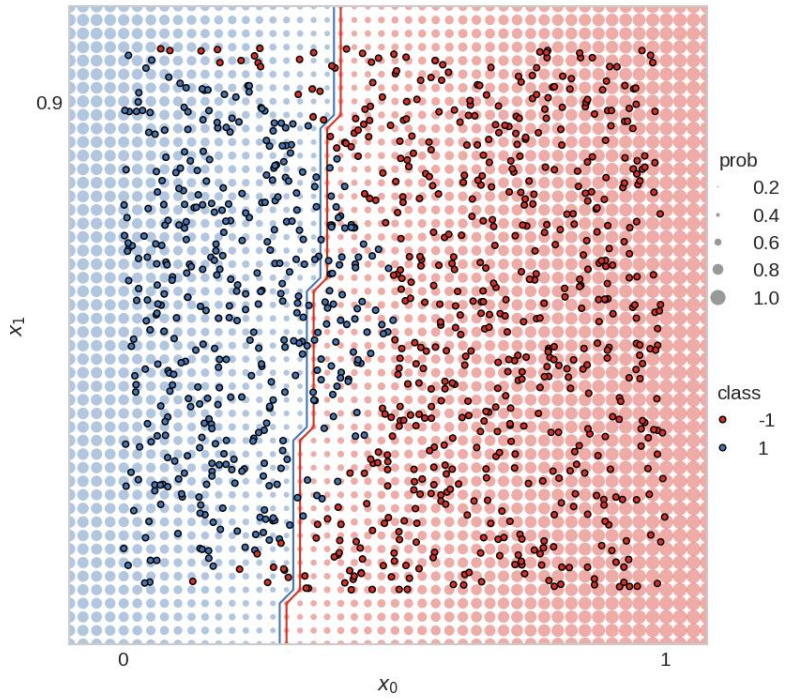
Perceptron is often not enough

- Perceptron is linear, but we often need a non-linear prediction function

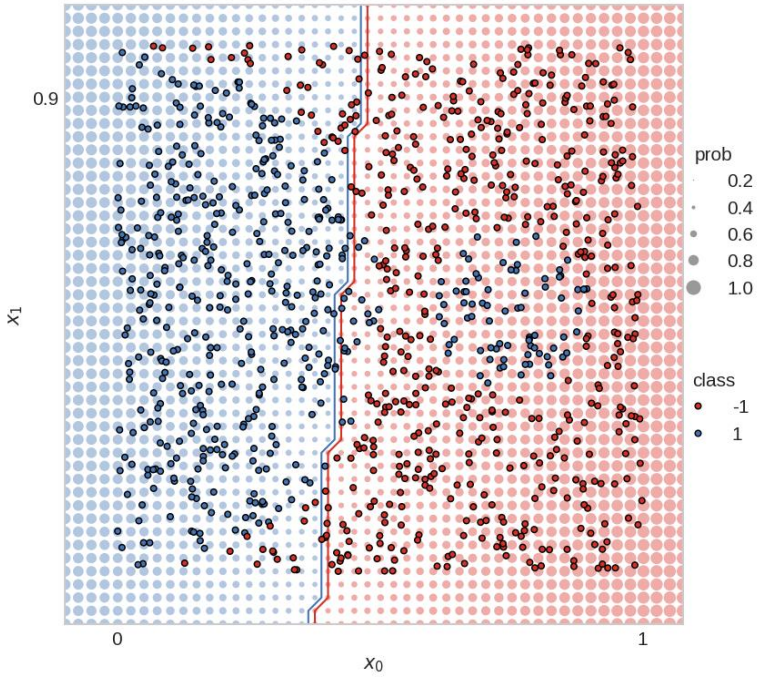
Which of these can a perceptron solve (fit with zero training error)?



Yes



No



Not even close

Review questions

http://tinyurl.com/cs441sgd

- Which of these are true about gradient descent or SGD?
  - Full-batch gradient descent (GD) guarantees an improvement in the objective at every step
  - GD offers more stable progression toward the objective than mini-batch SGD
  - SGD is able to reach better solutions for linear models than GD
  - SGD optimization tends to be faster than GD
  - SGD is better at escaping local minima than GD
  - Understanding SGD is critical for deep learning, even if you don't care about efficiency

Next week

- Tuesday: Recap and review for midterm
- Thursday: Exam – no lecture