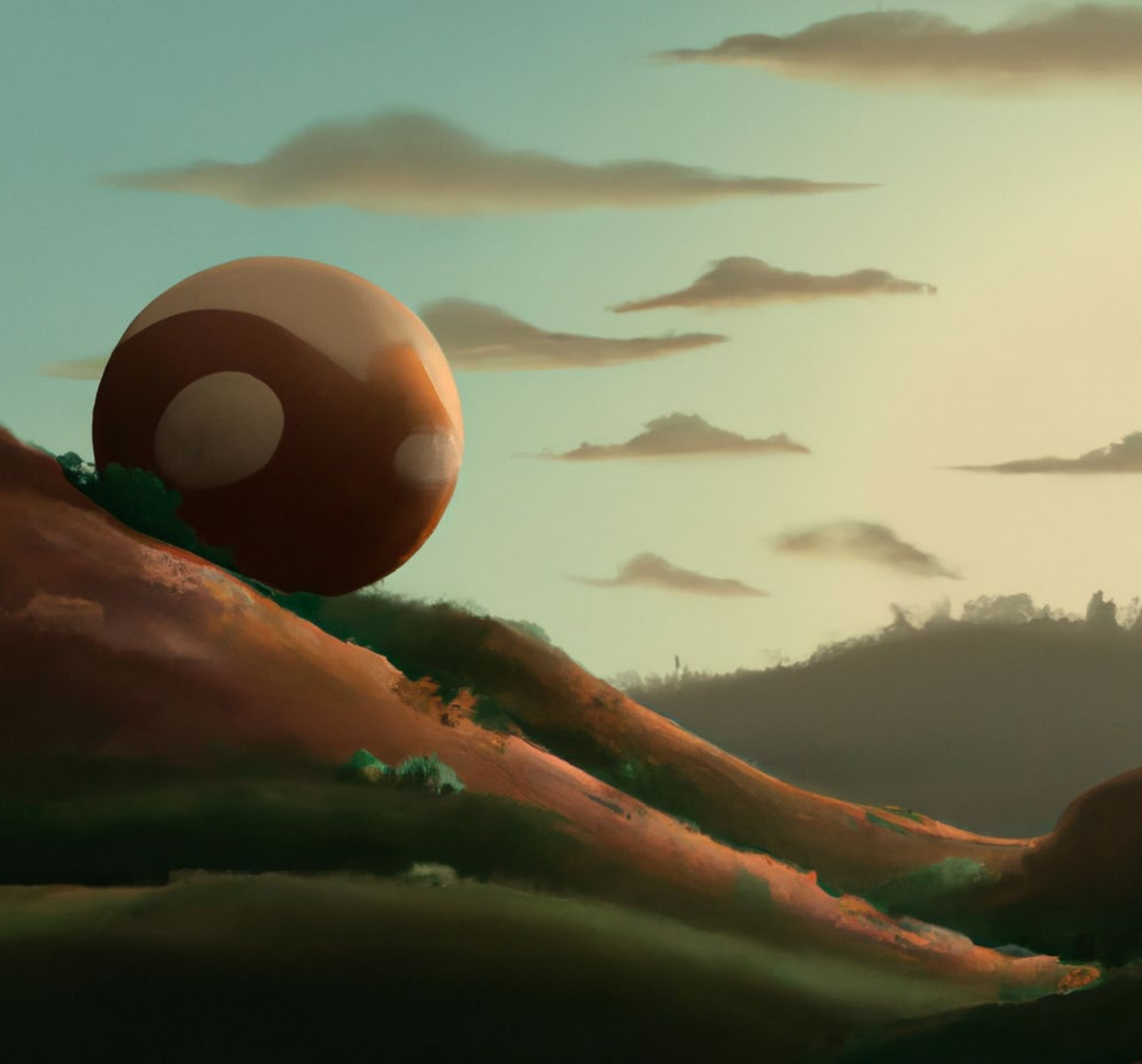




Optimization and Stochastic Gradient Descent

Applied Machine Learning
Derek Hoiem

Talk to a neighbor

1. Think for 2 minutes.
 - What is an idea for an application built around ML?
 - How will someone use the application?
 - What is the input and output of the ML model that is within the application?
2. Share with your neighbor(s)
3. Write down someone else's idea here

<https://tinyurl.com/441AML-L14>



Deep Learning

Deep learning is a way of learning effective representations, especially when the inputs have important structures, such as images, audio, text

- Today: Stochastic Gradient Descent (SGD)
- Thurs: MLPs and backpropagation
- Oct 20: Convolutional networks, residual blocks, advanced SGD
- Oct 22: Training and adapting deep networks, computer vision
- Oct 27: Representing words, transformer blocks
- Oct 29: More transformers, use in vision and language
- Nov 3: Foundation models: CLIP and GPT

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 another, 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

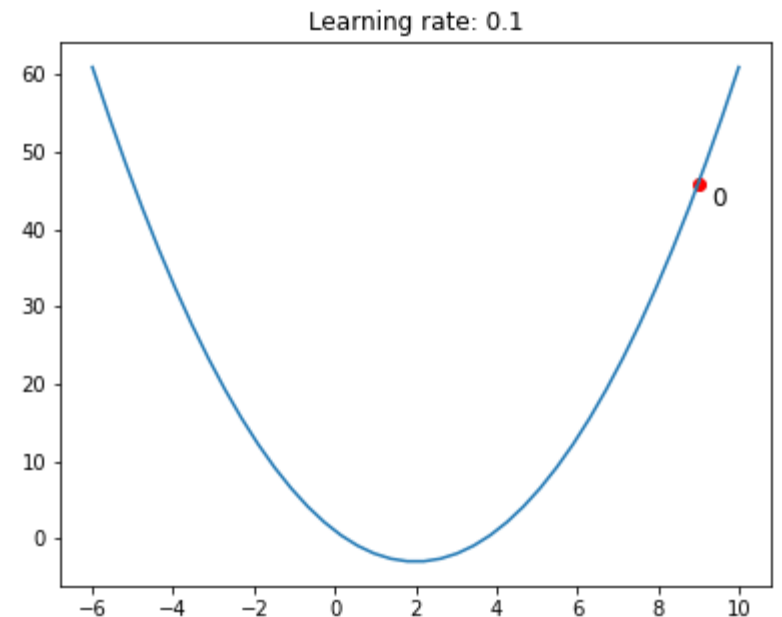
```
gradient_descent(f'(x), x0, lr, niter)
    x = x0
    for t in range(niter):
        x = x - lr*f'(x)
    return x
```

Example:

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

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

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



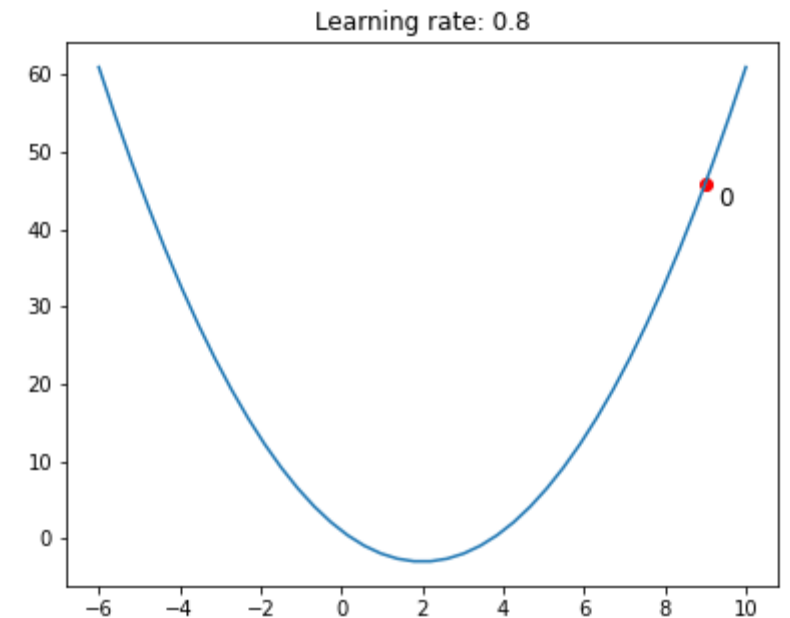
Gradient descent

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

Example:

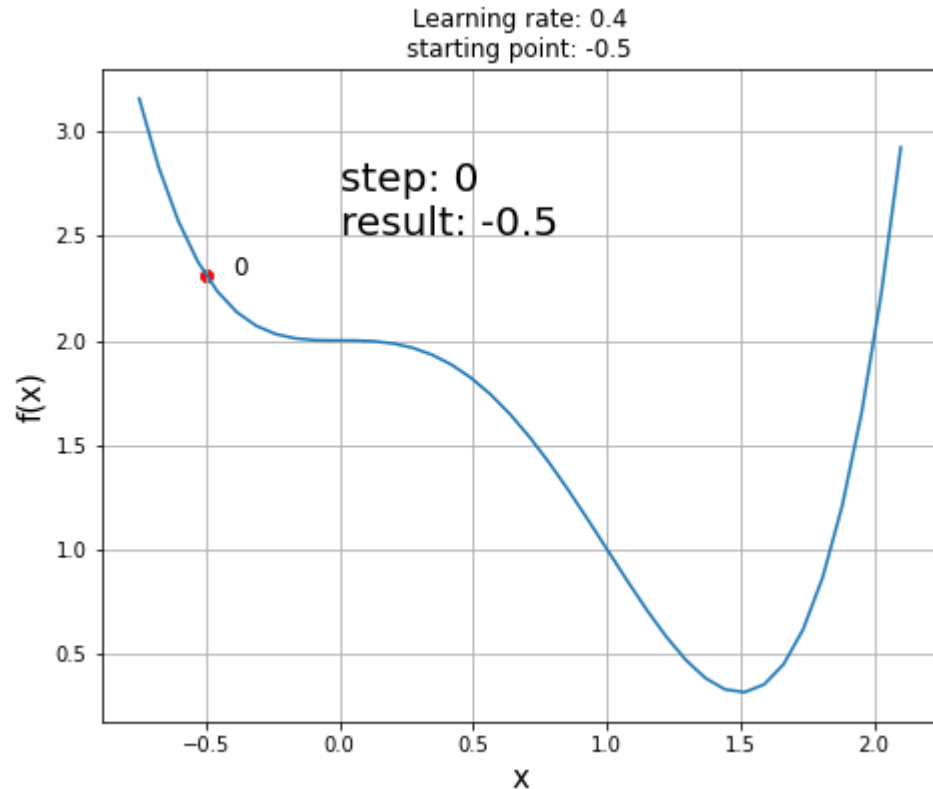
$$f(x) = x^2 - 4x + 1$$

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

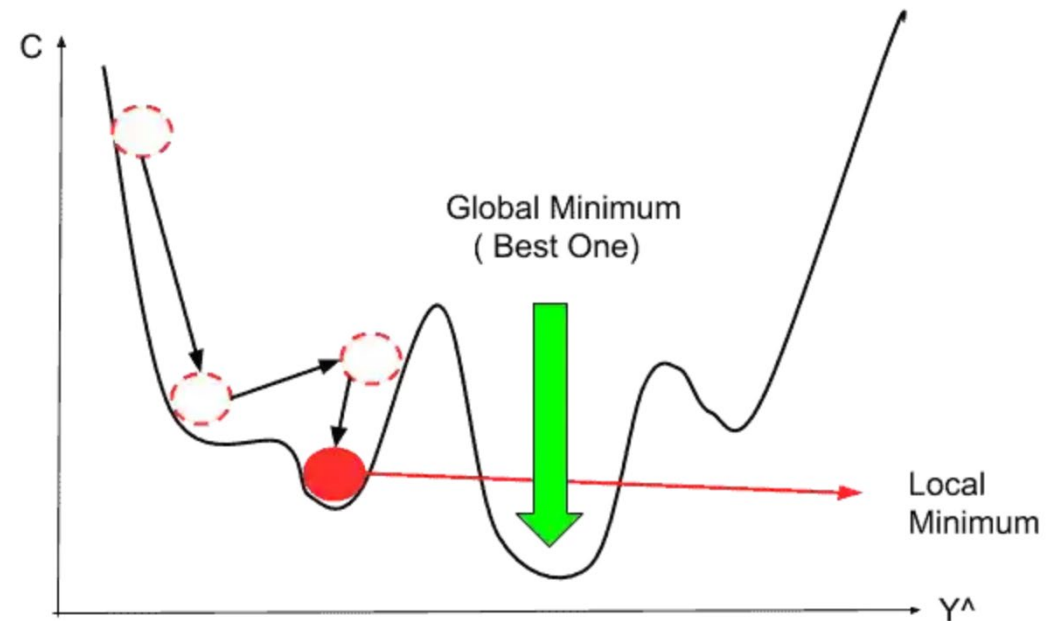


Example: <https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21>

Gradient descent challenge cases



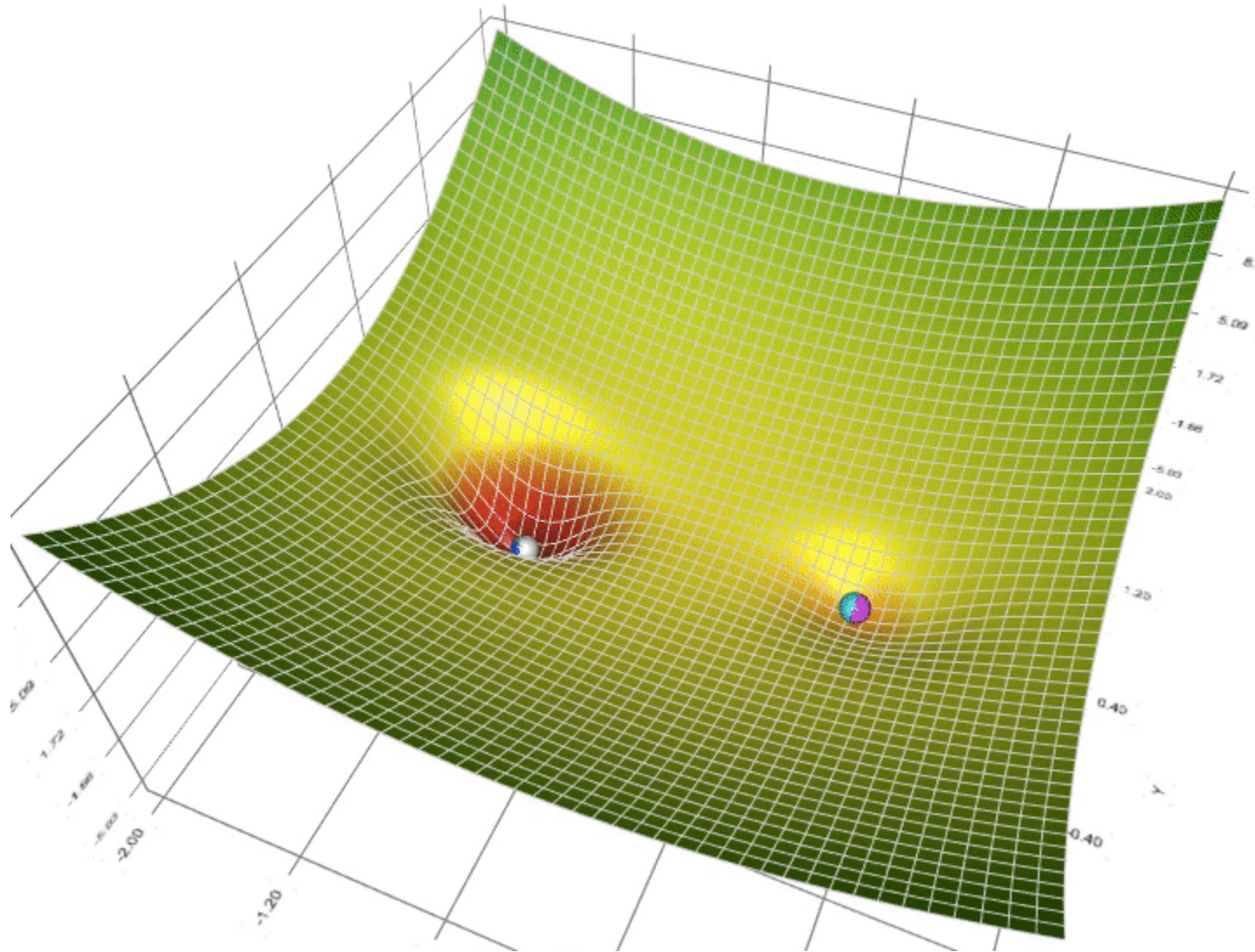
Saddle points (gradient = 0 in some parts of solution space)



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.

Gradient Descent Visualization with Local Minima



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 or get stuck in local minima
- Learning schedule: change learning rate over time
 - Constant
 - Exponential decay, e.g. $lr = lr * 0.95$
 - Linear, e.g. $lr = lr_0 * (1 - \text{iter} / \text{max_iter})$

SVM Formulation

Prediction

$$y_n = \text{sign}(\mathbf{w}^T \mathbf{x}_n + b)$$

Optimization

$$\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}} \left[\frac{1}{2} \lambda \|\mathbf{w}\|^2 + \frac{1}{N} \sum_n^N \max(0, 1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)) \right]$$

Known as “hinge loss”

Penalty is paid if margin is less than 1

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 w
```

$$f(\mathbf{w}, \mathbf{x}, \mathbf{y}) = \frac{1}{2} \lambda \|\mathbf{w}\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(\mathbf{w}^T \mathbf{x}_n))$$

$$f'(\mathbf{w}, i, \mathbf{x}, \mathbf{y}) = \lambda w_i + \frac{1}{N} \sum_n -\delta(y_n(\mathbf{w}^T \mathbf{x}_n) < 1) y_n x_{ni}$$

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

$$\frac{\partial (x_{n0}w_0 + x_{n1}w_1 + \dots + x_{ni}w_i + \dots)}{\partial w_i} = x_{ni}$$

Slow with large datasets, because need to compute scores for all examples in each step

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

Pegasos algorithm: Stochastic Gradient Descent (SGD)

```
INPUT:  $S, \lambda, T$ 
INITIALIZE: Set  $\mathbf{w}_1 = 0$ 
FOR  $t = 1, 2, \dots, T$ 
    Choose  $i_t \in \{1, \dots, |S|\}$  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

$\mathbf{w}_t$: model weights

$\mathbf{x}_{i_t}$: 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:  $S, \lambda, T, k$ 
INITIALIZE: Set  $\mathbf{w}_1 = 0$ 
FOR  $t = 1, 2, \dots, T$ 
    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

$\mathbf{w}_t$: model weights

$\mathbf{x}_i$: features for example i

y_i : label for example i

η_t : step size (“learning rate”)

SGD applies to many losses

z is the score
for $y=1$

SVM (hinge loss)

$$\ell(z, y_i) = \max\{0, 1 - y_i z\}$$

$$\mathbf{v}_t = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i z < 1 \\ 0 & \text{otherwise} \end{cases}$$

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

$$\ell(z, y_i) = \max\{0, |y_i - z| - \epsilon\}$$

$$\mathbf{v}_t = \begin{cases} \mathbf{x}_i & \text{if } z - y_i > \epsilon \\ -\mathbf{x}_i & \text{if } y_i - z > \epsilon \\ 0 & \text{otherwise} \end{cases}$$

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 $\hat{y} = \arg \max_y \delta(y, y_i) - z_{y_i} + z_y$

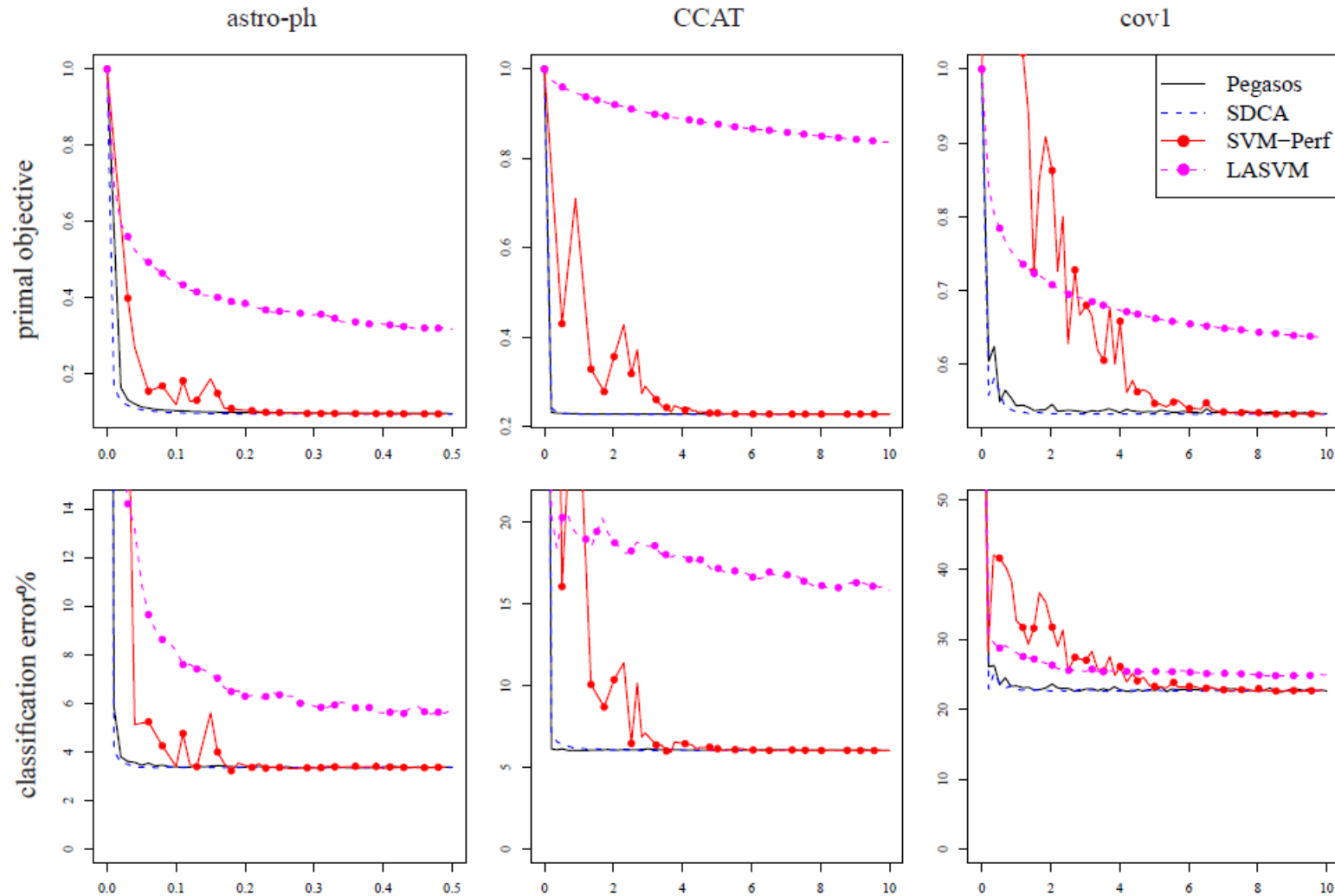
Variant of a logistic loss

$$\ell(z, y_i) = \log \left(1 + \sum_{r \neq y_i} e^{z_r - z_{y_i}} \right)$$

$$\mathbf{v}_t = \sum_r p_r \phi(\mathbf{x}_i, r) - \phi(\mathbf{x}_i, y_i)$$

where $p_r = e^{z_r} / \sum_j e^{z_j}$

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 stochastic 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

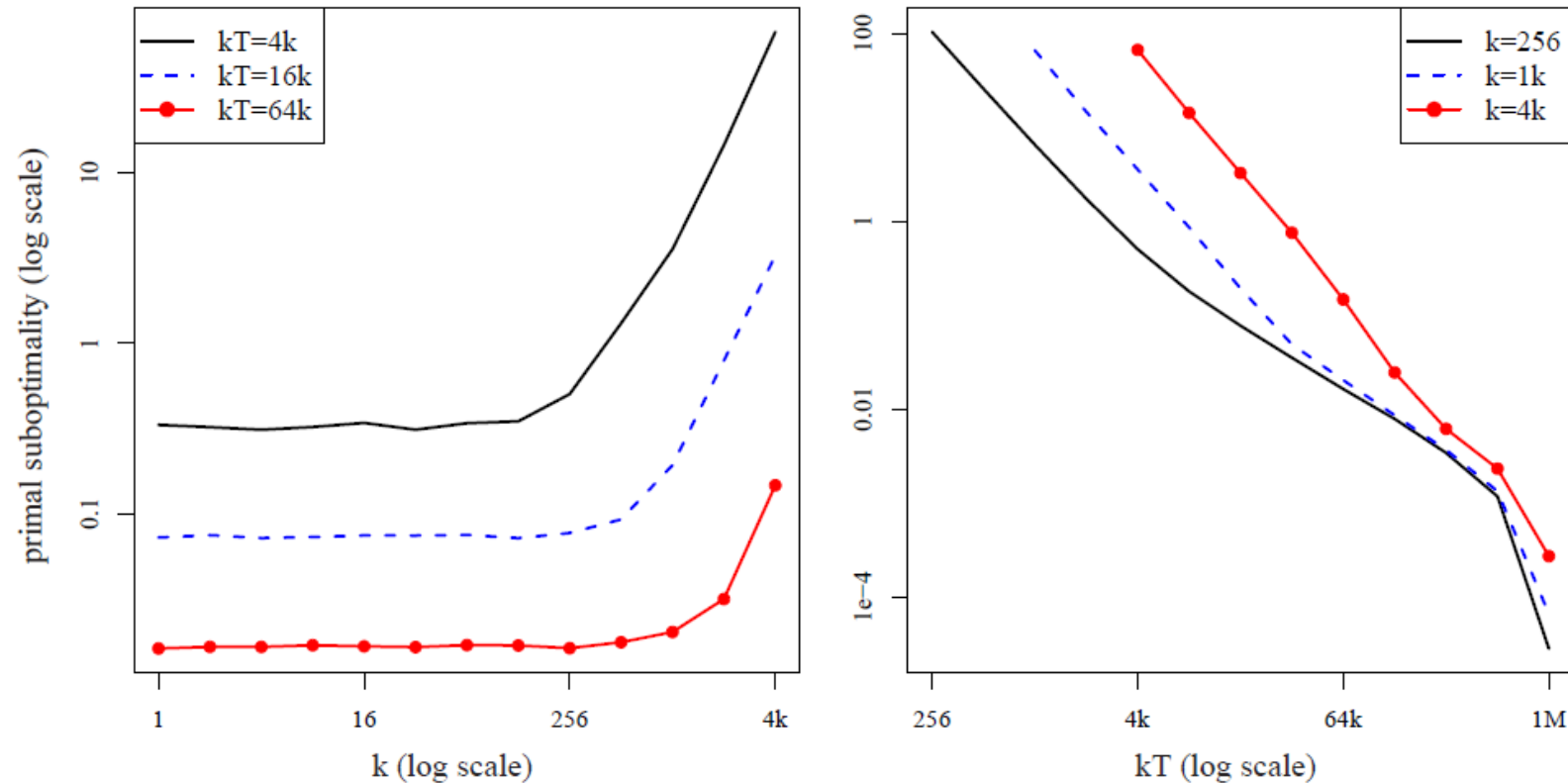
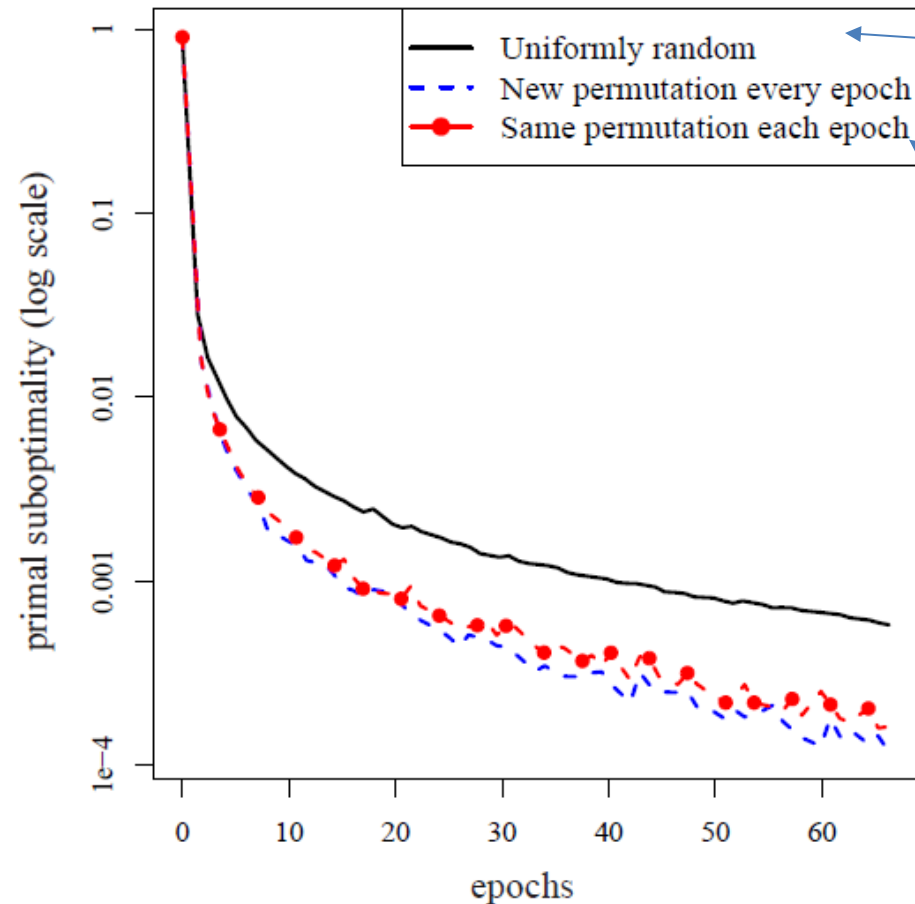


Fig. 7 The effect of the mini-batch size on the runtime of Pegasos for the astro-ph dataset. The first plot shows the primal suboptimality achieved for certain fixed values of overall runtime kT , for various values of the mini-batch size k . The second plot shows the primal suboptimality achieved for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

With fixed computation, a batch of 1-256 gets the lowest loss

(but with GPUs, large batches can sometimes reach lower loss in less wall clock time due to parallelization)

Effect of sampling procedure: randomly ordered epochs is best



Sampling with replacement

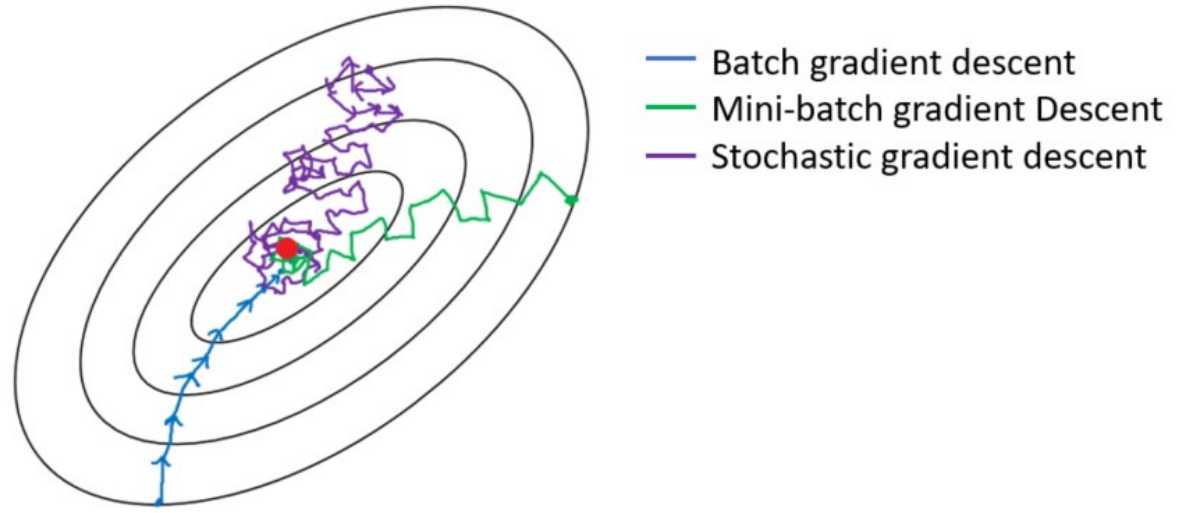
Use different random order for each "epoch"

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 $O(1/\sqrt{B})$
 - 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>

Terminology: when people say “SGD” they usually mean mini-batch SGD

Pegasos: take-aways 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

Q1-Q2

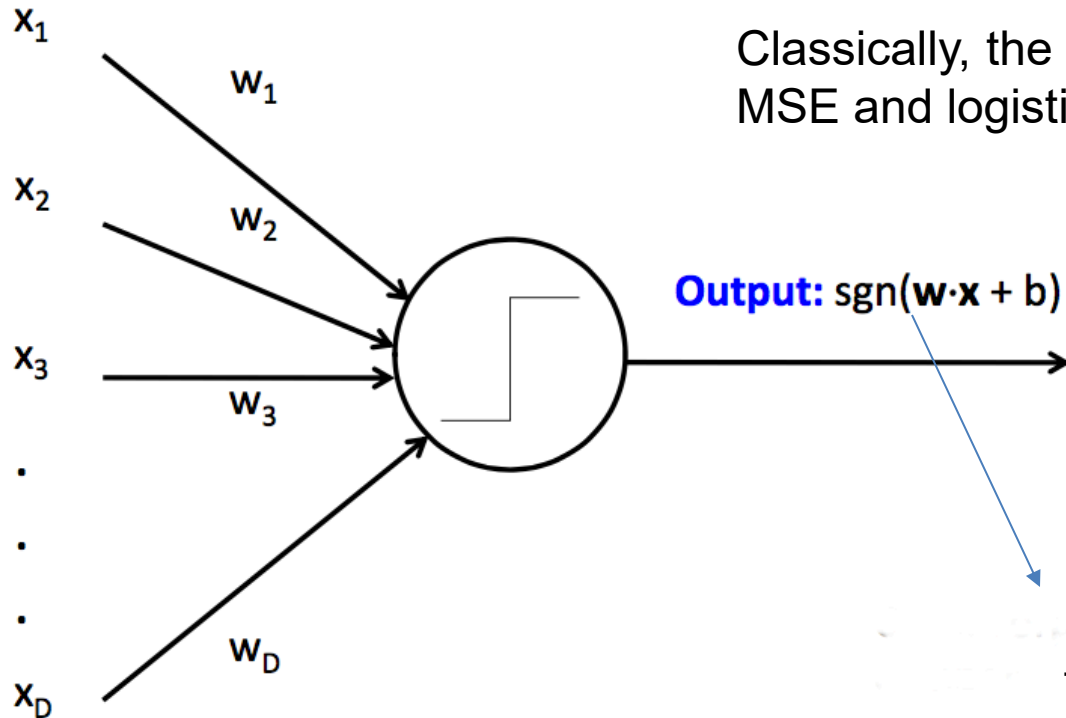
<https://tinyurl.com/441AML-L14>



Perceptron

Input

Weights



Perceptron = thresholded linear prediction model for classification

Classically, the loss is a hinge loss (like SVM), but we'll consider MSE and logistic losses

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

Perceptron Update Rule with MSE Loss

Prediction: $f(\mathbf{x}) = w_0x_0 + w_1x_1 + \dots w_mx_m + b$

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

prediction target

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

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

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

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

Make error *lower*

Learning rate

Chain Rule:

$h(x) = f(g(x))$, then

$$h'(x) = f'(g(x))g'(x)$$

(the 2 is folded into the learning rate)

Perceptron Optimization by SGD (MSE Loss)

Randomly initialize weights, e.g. $w \sim \text{Gaus}(\mu = 0, \sigma = 0.05)$

For each iteration t :

Split data into batches

$$\eta = 0.1/t$$

For each batch X_b :

For each weight w_i :

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

With different loss, the update changes accordingly

Logistic loss:

$$f(\mathbf{x}) = w_0x_0 + w_1x_1 + \dots w_mx_m + b$$

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

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

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


decrease $-\log P(y|\mathbf{x}) \rightarrow$ increase $\log P(y|\mathbf{x})$

Q3

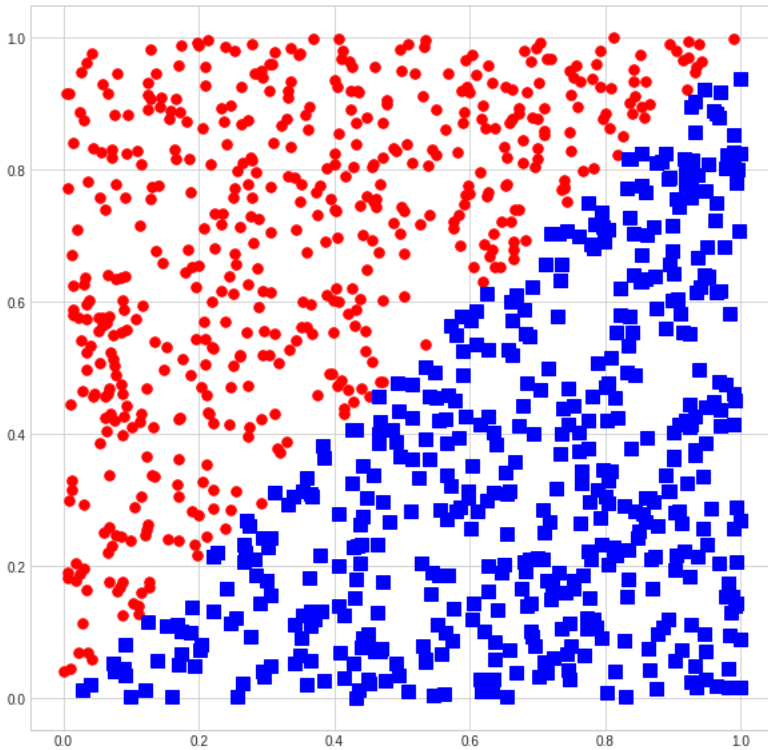
<https://tinyurl.com/441AML-L14>



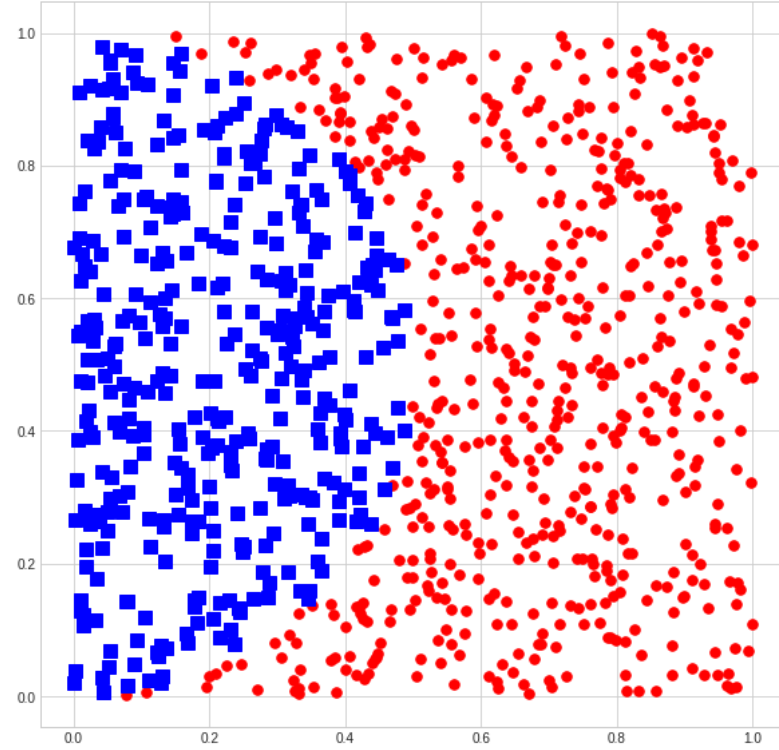
Demo

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

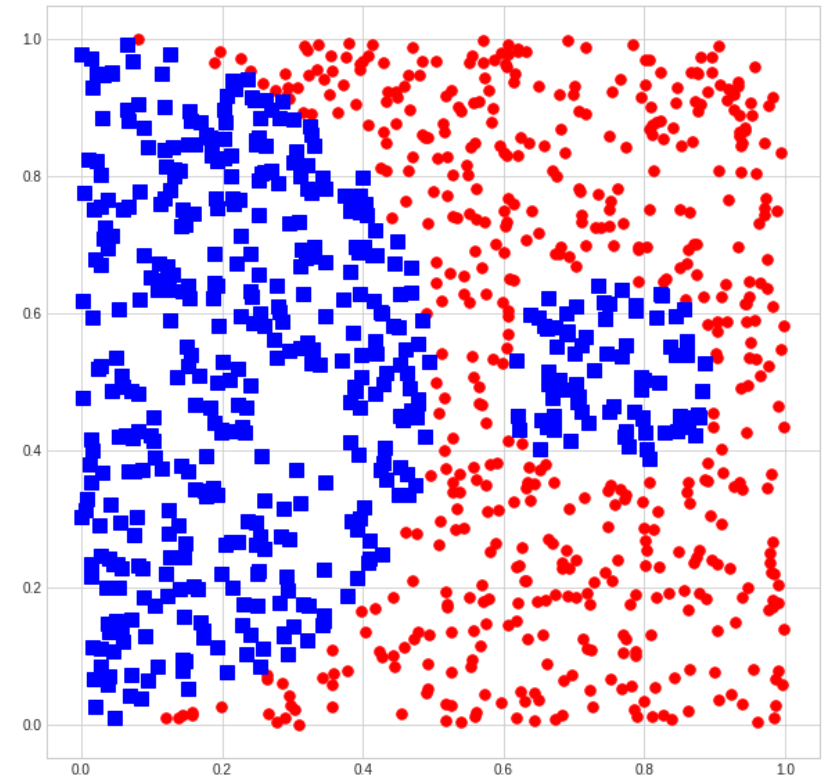
Which of these can a perceptron solve (fit with zero training error)?



(a)



(b)

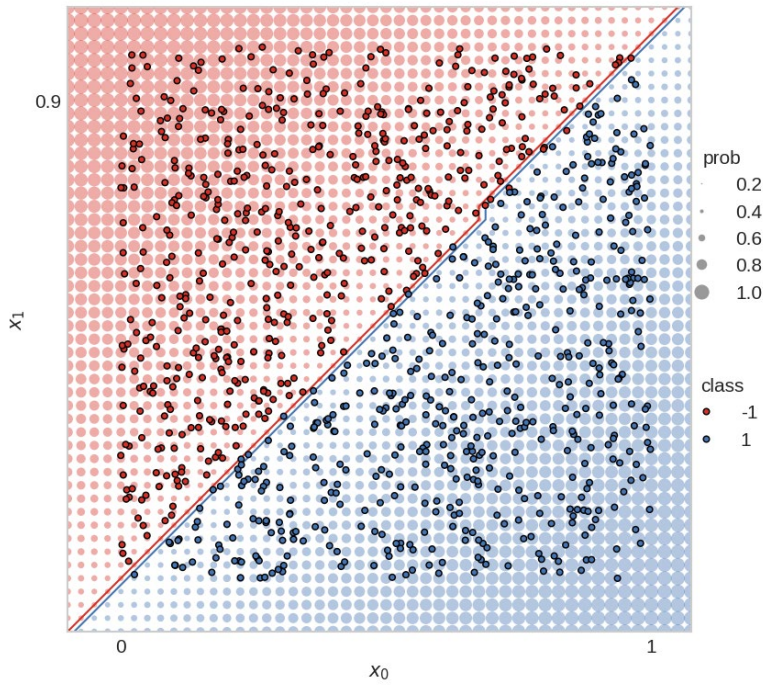


(c)

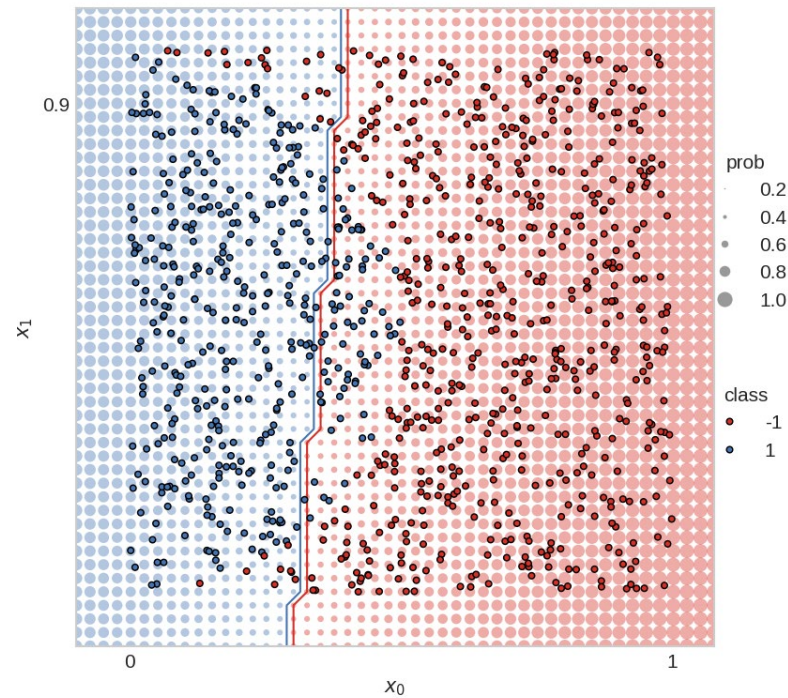
Perceptron is often not enough

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

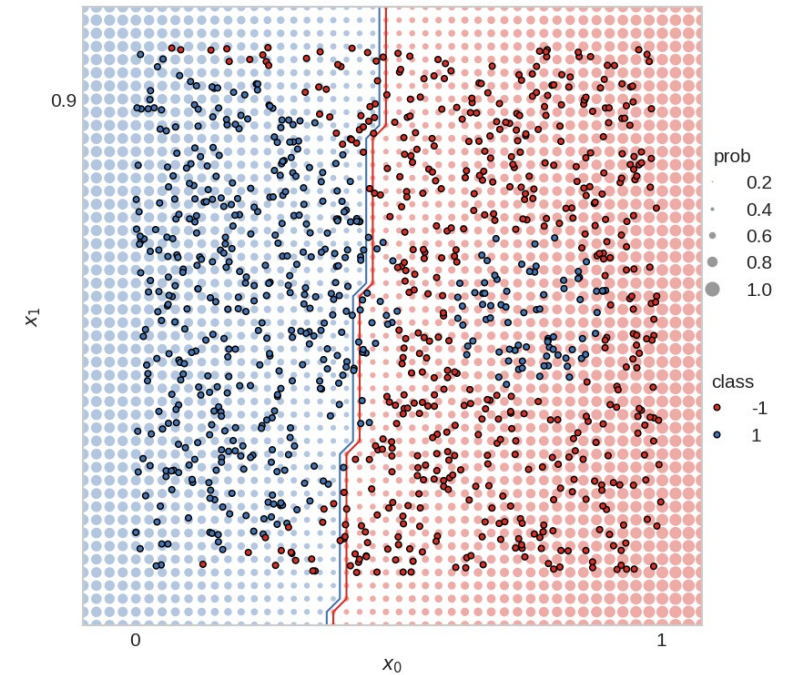
Which of these can a perceptron solve (fit with zero training error)?



Yes



No



Not even close

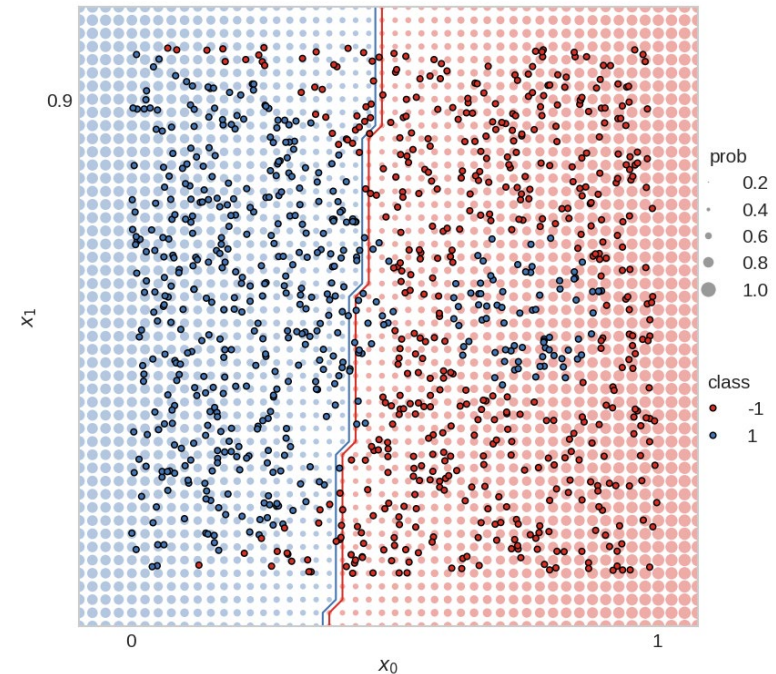
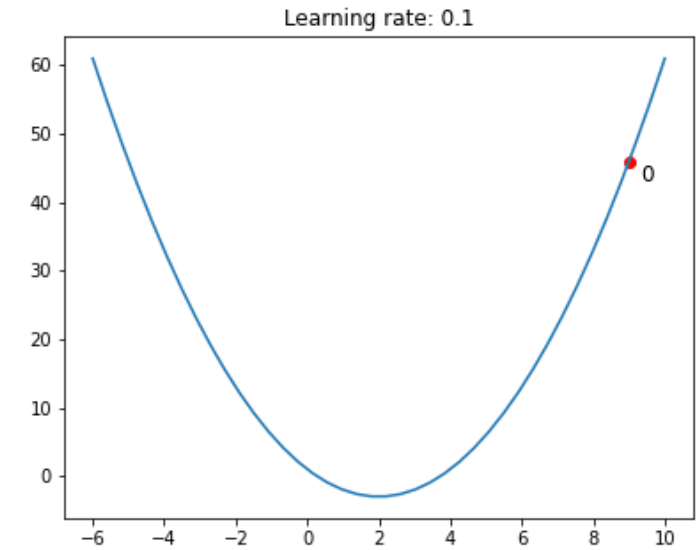
Q4

<https://tinyurl.com/441AML-L14>



Things to Remember

- Gradient descent iteratively steps in direction of negative gradient of loss
- Stochastic gradient descent estimates gradient using small batches of samples
 - Faster than full gradient descent
- Linear models have limited ability to fit the data – often need non-linear models like multilayer networks



Coming up

- Thursday: MLPs