



FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Loss functions and Optimization

**A. Maier**, K. Breininger, L. Mill, N. Ravikumar, T. Würfl

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 25, 2018



# Outline

## Loss Functions

## Optimization



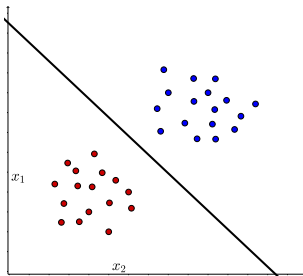
FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Loss Functions

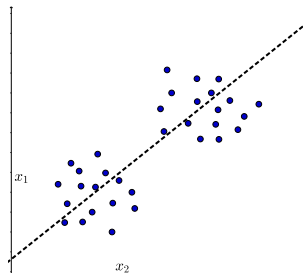


## Regression vs. classification

- **Regression:** Estimate a continuous variable for every input.
- **Classification:** Estimate a discrete variable for every input.



Classification



Regression

## Loss function vs. last activation function in a network

### The last activation function

- is applied on **individual samples**  $x_m$  **of the batch**
- is present at training **and testing**
- produces the output, or prediction
- generally produces a vector

### The loss function

- combines **all  $M$  samples and labels**
- is **only** present at **training**
- produces the loss
- generally produces a scalar

# Maximum Likelihood Estimation Reminder

Assume a

- Conditional probability density function  $p$
- Training set with
  - Observations:  $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_m$
  - and associated labels  $\mathbf{Y} = \mathbf{y}_1, \dots, \mathbf{y}_m$

# Maximum Likelihood Estimation Reminder

Assume a

- Conditional probability density function  $p$
- Training set with
  - Observations:  $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_m$
  - and associated labels  $\mathbf{Y} = \mathbf{y}_1, \dots, \mathbf{y}_m$

## Dataset

- Probability to observe  $\mathbf{y}_m$  given observation  $\mathbf{x}_m$  is  $p(\mathbf{y}_m|\mathbf{x}_m)$
- Joint probability is  $p(\mathbf{y}_m|\mathbf{x}_m) \cdot p(\mathbf{y}_i|\mathbf{x}_i)$  if they are:
  - Independent
  - and **I**dentically **D**istributed
- probability to observe  $\mathbf{Y}$  is  $\prod_{m=1}^M p(\mathbf{y}_m|\mathbf{x}_m)$

## Likelihood function

- $p$  governed by parameters  $\mathbf{w}$

$$\underset{\mathbf{w}}{\text{maximize}} \quad \left\{ \prod_{m=1}^M p(\mathbf{y}_m | \mathbf{x}_m, \mathbf{w}) \right\}$$



## Likelihood function

- $p$  governed by parameters  $\mathbf{w}$

$$\underset{\mathbf{w}}{\text{maximize}} \quad \left\{ \prod_{m=1}^M p(\mathbf{y}_m | \mathbf{x}_m, \mathbf{w}) \right\}$$

## Negative Log Likelihood

- Maximum not affected by a monotonous transformation
- Maximization to minimization by flipping the sign
- $$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ -\ln(L(\mathbf{w})) \right\} = \underset{\mathbf{w}}{\text{minimize}} \quad \left\{ \sum_{m=1}^M -\ln(p(\mathbf{y}_m | \mathbf{x}_m, \mathbf{w})) \right\}$$

## Regression

Assume a **univariate** Gaussian model:

$$p(y|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(\hat{y}(\mathbf{x}, \mathbf{w}), \frac{1}{\beta})$$

# Regression

Assume a **univariate** Gaussian model:

$$p(y|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(\underbrace{\hat{y}(\mathbf{x}, \mathbf{w})}_{\mu}, \underbrace{\frac{1}{\beta}}_{\sigma})$$

## Log Likelihood Function Regression

$$L(\mathbf{w}) = \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{\beta \frac{-(y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}{2}} \right)$$

## Log Likelihood Function Regression

$$\begin{aligned} L(\mathbf{w}) &= \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{\beta \frac{(y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}{2}} \right) \\ &= \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} \right) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \end{aligned}$$

## Log Likelihood Function Regression

$$\begin{aligned} L(\mathbf{w}) &= \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{\beta \frac{(y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}{2}} \right) \\ &= \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} \right) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \\ &= \sum_{m=1}^M \frac{1}{2} (\ln(2\pi) - \ln(\beta)) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \end{aligned}$$

## Log Likelihood Function Regression

$$\begin{aligned} L(\mathbf{w}) &= \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{\beta \frac{(y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}{2}} \right) \\ &= \sum_{m=1}^M -\ln \left( \frac{\sqrt{\beta}}{\sqrt{2\pi}} \right) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \\ &= \sum_{m=1}^M \frac{1}{2} (\ln(2\pi) - \ln(\beta)) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \\ &= \frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\beta) + \frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \end{aligned}$$

## $L^2$ -loss

$$\frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\beta) + \frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2$$



## $L^2$ -loss

$$\frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\beta) + \underbrace{\frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}_{\text{Depends on } \mathbf{w}}$$

## $L^2$ -loss

$$\frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\beta) + \underbrace{\frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}_{\text{Depends on } \mathbf{w}}$$

When optimizing for  $\mathbf{w}$  - eliminate constants and factors:

$$\frac{1}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2$$

## $L^2$ -loss

$$\frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\beta) + \underbrace{\frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}_{\text{Depends on } \mathbf{w}}$$

When optimizing for  $\mathbf{w}$  - eliminate constants and factors:

$$\frac{1}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2$$

This can be generalized to vectors  $\mathbf{y}_m, \hat{\mathbf{y}}$ :

$$\frac{1}{2} \sum_{m=1}^M \|\mathbf{y}_m - \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w})\|_2^2$$

## Classification using an $L$ -norm

### $L_2$ -loss and $L_1$ -loss can be applied for classification

- They correspond to variants of minimizing the **expected misclassification probability**
- They cause **slow convergence** because they don't penalize heavily misclassified probabilities
- They might be advantageous in situations with **extreme label noise**

## Classification

Assume our network provides us with a probabilistic output  $p$ .

## Classification

Assume our network provides us with a probabilistic output  $p$ .

### Bernoulli distribution

$$\mathfrak{B}(y|p) = \begin{cases} p^y(1-p)^{1-y} & \text{if } y \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

## Classification

Assume our network provides us with a probabilistic output  $p$ .

### Bernoulli distribution

$$\mathfrak{B}(y|p) = \begin{cases} p^y(1-p)^{1-y} & \text{if } y \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

### Multi-class generalization: Multinoulli (Categorical, $\mathfrak{C}$ ) distribution

- $\mathbf{y}$ , which is one-hot encoded

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

## Example for $\mathfrak{C}$

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

### Coin example

- We encode head as  $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and tail as  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$



## Example for $\mathfrak{C}$

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

### Coin example

- We encode head as  $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and tail as  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- We have a unfair coin:  $\mathbf{p} = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$  and observe  $\mathbf{y} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

## Example for $\mathfrak{C}$

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

### Coin example

- We encode head as  $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and tail as  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- We have a unfair coin:  $\mathbf{p} = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$  and observe  $\mathbf{y} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- The probability of this is  $\mathfrak{C}(\mathbf{y}|\mathbf{p}) = p_0^0 \cdot p_1^1 = 1 \cdot 0.7 = 0.7$

## Example for $\mathfrak{C}$

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

### Coin example

- We encode head as  $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$  and tail as  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- We have a unfair coin:  $\mathbf{p} = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$  and observe  $\mathbf{y} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- The probability of this is  $\mathfrak{C}(\mathbf{y}|\mathbf{p}) = p_0^0 \cdot p_1^1 = 1 \cdot 0.7 = 0.7$
- So the probability to observe  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$  which is tail for this unfair coin is 70%.

## Maximum Likelihood Estimation for Classification

We convert the scores  $\hat{\mathbf{y}}$  to probabilistic vectors using the Softmax function.

## Maximum Likelihood Estimation for Classification

We convert the scores  $\hat{\mathbf{y}}$  to probabilistic vectors using the Softmax function.

- Assume our labels are categorically distributed
- with probabilities given by our predictions:

$$p(\mathbf{y}|\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w})) = \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))$$

## Maximum Likelihood Estimation for Classification

We convert the scores  $\hat{\mathbf{y}}$  to probabilistic vectors using the Softmax function.

- Assume our labels are categorically distributed
- with probabilities given by our predictions:

$$p(\mathbf{y}|\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w})) = \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))$$

### Negative log Likelihood

$$L(\mathbf{w}) = - \sum_{m=1}^M \ln p(\mathbf{y}_m | \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w}))$$

## Maximum Likelihood Estimation for Classification

We convert the scores  $\hat{\mathbf{y}}$  to probabilistic vectors using the Softmax function.

- Assume our labels are categorically distributed
- with probabilities given by our predictions:

$$p(\mathbf{y}|\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w})) = \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))$$

### Negative log Likelihood

$$L(\mathbf{w}) = - \sum_{m=1}^M \ln p(\mathbf{y}_m | \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w})) = - \sum_{m=1}^M \ln \prod_{k=0}^K \hat{y}_k(\mathbf{x}_m, \mathbf{w})^{y_{k,m}}$$

## Maximum Likelihood Estimation for Classification

We convert the scores  $\hat{\mathbf{y}}$  to probabilistic vectors using the Softmax function.

- Assume our labels are categorically distributed
- with probabilities given by our predictions:

$$p(\mathbf{y}|\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w})) = \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))$$

### Negative log Likelihood

$$\begin{aligned} L(\mathbf{w}) &= - \sum_{m=1}^M \ln p(\mathbf{y}_m | \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w})) = - \sum_{m=1}^M \ln \prod_{k=0}^K \hat{y}_k(\mathbf{x}_m, \mathbf{w})^{y_{k,m}} \\ &= - \sum_{m=1}^M \sum_{k=0}^K \ln (\hat{y}_k^{y_{k,m}}) \end{aligned}$$



## Maximum Likelihood Estimation for Classification

We convert the scores  $\hat{\mathbf{y}}$  to probabilistic vectors using the Softmax function.

- Assume our labels are categorically distributed
- with probabilities given by our predictions:

$$p(\mathbf{y}|\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w})) = \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))$$

### Negative log Likelihood

$$\begin{aligned} L(\mathbf{w}) &= - \sum_{m=1}^M \ln p(\mathbf{y}_m | \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w})) = - \sum_{m=1}^M \ln \prod_{k=0}^K \hat{y}_k(\mathbf{x}_m, \mathbf{w})^{y_{k,m}} \\ &= - \sum_{m=1}^M \sum_{k=0}^K \ln (\hat{y}_k^{y_{k,m}}) \end{aligned}$$

$$= - \sum_{m=1}^M \ln(\hat{y}_k(\mathbf{x}_m, \mathbf{w}))|_{y_{k,m}=1}$$

## Relation to the Kullback Leibler Divergence

$$\text{KL}(p, q) = \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx$$

## Relation to the Kullback Leibler Divergence

$$\begin{aligned} \text{KL}(p, q) &= \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx \\ &= \int_{-\infty}^{+\infty} p(x) \ln p(x) - \int_{-\infty}^{+\infty} p(x) \ln q(x) dx \end{aligned}$$

## Relation to the Kullback Leibler Divergence

$$\begin{aligned} \text{KL}(p, q) &= \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx \\ &= \underbrace{\int_{-\infty}^{+\infty} p(x) \ln p(x) dx}_{-\text{Entropy } H(p)} - \underbrace{\int_{-\infty}^{+\infty} p(x) \ln q(x) dx}_{\text{Cross Entropy } H(p, q)} \end{aligned}$$

## Relation to the Kullback Leibler Divergence

$$\begin{aligned} \text{KL}(p, q) &= \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx \\ &= \underbrace{\int_{-\infty}^{+\infty} p(x) \ln p(x) dx}_{-\text{Entropy } H(p)} - \underbrace{\int_{-\infty}^{+\infty} p(x) \ln q(x) dx}_{\text{Cross Entropy } H(p, q)} \end{aligned}$$

We know that our ML estimation for a single sample has the form of cross-entropy:

$$-\sum_{k=0}^K \ln(\hat{y}_k^{y_k}) = H(\mathbf{y}, \hat{\mathbf{y}})$$

and therefore is equal to minimizing the KL-divergence.

## Can we also use cross-entropy for regression?

## Can we also use cross-entropy for regression?

- Of course. We just have to make sure  $\hat{y}_k \in [0, 1] \forall k$
- This can be achieved using a sigmoid activation function
- $\mathbf{y}$  is simply no longer one-hot encoded
- As we've seen before this is equivalent to minimizing KL-divergence

# Summary



## Summary

- $L_2$ -loss can be used for **regression**
- Cross-entropy-loss can be used for **classification**

## Summary

- $L_2$ -loss can be used for **regression**
- Cross-entropy-loss can be used for **classification**
- $L_2$ -loss and Cross-entropy-loss can be derived as **ML-Estimators** from **strict** probabilistic assumptions
- In absence of more domain knowledge they are your **first choices**
- They are both intrinsically **multi-variate**

## Back to the Perceptron - again!

How does the Perceptron criterion fit into this?

$$\text{minimize } \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here  $y_m \in -1, 1$  instead of  $y_m \in 0, 1$

## Back to the Perceptron - again!

How does the Perceptron criterion fit into this?

$$\text{minimize } \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here  $y_m \in -1, 1$  instead of  $y_m \in 0, 1$
- Note that the sign function does not appear in the criterion

## Back to the Perceptron - again!

How does the Perceptron criterion fit into this?

$$\text{minimize } \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here  $y_m \in -1, 1$  instead of  $y_m \in 0, 1$
- Note that the sign function does not appear in the criterion
- What if it was in?

## Back to the Perceptron - again!

How does the Perceptron criterion fit into this?

$$\text{minimize } \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here  $y_m \in -1, 1$  instead of  $y_m \in 0, 1$
- Note that the sign function does not appear in the criterion
- What if it was in?
- Than we would just count the number of misclassifications

## Back to the Perceptron - again!

How does the Perceptron criterion fit into this?

$$\text{minimize } \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here  $y_m \in -1, 1$  instead of  $y_m \in 0, 1$
- Note that the sign function does not appear in the criterion
- What if it was in?
- Than we would just count the number of misclassifications
- ... and the gradient would vanish almost everywhere

## Back to the Perceptron - again!

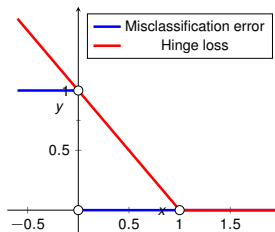
How does the Perceptron criterion fit into this?

$$\text{minimize } \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here  $y_m \in -1, 1$  instead of  $y_m \in 0, 1$
- Note that the sign function does not appear in the criterion
- What if it was in?
- Than we would just count the number of misclassifications
- ... and the gradient would vanish almost everywhere
- Sounds familiar?
- What did we do about that last time?



# Hinge loss



$$L(\mathbf{w}) = \sum_{m=1}^M \max(0, 1 - y_m \hat{y}_m(\mathbf{x}_m, \mathbf{w}))$$

- Classification depends only on the sign
- If the signs match we get a positive value and classify correct
- Hinge loss is a convex approximation to the misclassification loss
- But what about the gradient?

## Subgradients

Suppose we have a convex, differentiable function. Then we have:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

## Subgradients

Suppose we have a convex, differentiable function. Then we have:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

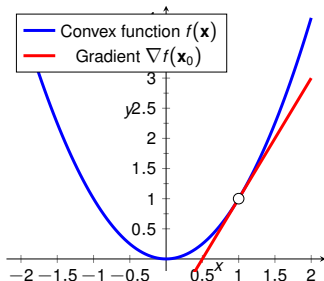
In words: If we follow the gradient from any point of a convex function and check against the function its value at the same  $\mathbf{x}$  will be higher.

## Subgradients

Suppose we have a convex, differentiable function. Then we have:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

In words: If we follow the gradient from any point of a convex function and check against the function its value at the same  $\mathbf{x}$  will be higher.



## Subgradients

- We now define something which just keeps this property but is not necessarily a gradient

## Subgradients

- We now define something which just keeps this property but is not necessarily a gradient
- A vector  $\mathbf{g}$  is a subgradient of a **convex** function  $f$  at point  $\mathbf{x}_0 \in \mathcal{X}$  if:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \mathbf{g}^T(\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

## Subgradients

- We now define something which just keeps this property but is not necessarily a gradient
- A vector  $\mathbf{g}$  is a subgradient of a **convex** function  $f$  at point  $\mathbf{x}_0 \in \mathcal{X}$  if:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \mathbf{g}^T(\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

- This is not unique! We get a set of subgradients which we call a subdifferential:

$$\partial f(\mathbf{x}_0) := \{\mathbf{g}\}$$

## Subgradients

- We now define something which just keeps this property but is not necessarily a gradient
- A vector  $\mathbf{g}$  is a subgradient of a **convex** function  $f$  at point  $\mathbf{x}_0 \in \mathcal{X}$  if:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \mathbf{g}^T(\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

- This is not unique! We get a set of subgradients which we call a subdifferential:

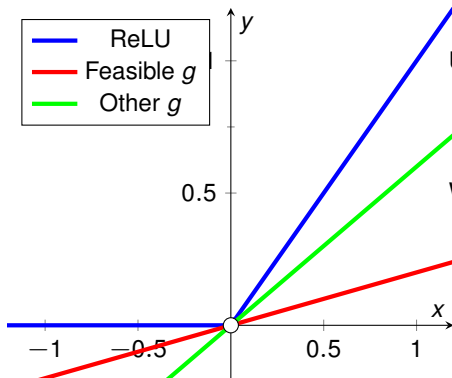
$$\partial f(\mathbf{x}_0) := \{\mathbf{g}\}$$

- If  $f$  is differentiable at  $\mathbf{x}_0$ :

$$\partial f(\mathbf{x}_0) = \{\nabla f(\mathbf{x}_0)\}$$



# Subgradients



Using:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \mathbf{g}^T(\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

We get:

$$\partial f(x_0) = \begin{cases} 1 & \text{if } x_0 > 0 \\ 0 & \text{if } x_0 < 0 \\ g \in [0, 1] & \text{if } x_0 = 0 \end{cases}$$

- We already used this for the ReLU!
- Gradient descent was implicitly generalized to the subgradient algorithm

# Summary

## Summary

- Subgradients are a generalization of gradients for **convex, non-smooth functions**
- The gradient descent algorithm is replaced by the subgradient algorithm for these functions

## Summary

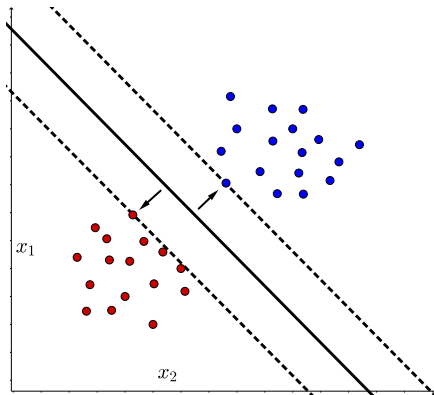
- Subgradients are a generalization of gradients for **convex, non-smooth functions**
- The gradient descent algorithm is replaced by the subgradient algorithm for these functions
- For piecewise continuous functions you just choose a particular subgradient and don't even notice a difference
- This is basically just the solid math why this works

## Summary

- Subgradients are a generalization of gradients for **convex, non-smooth functions**
- The gradient descent algorithm is replaced by the subgradient algorithm for these functions
- For piecewise continuous functions you just choose a particular subgradient and don't even notice a difference
- This is basically just the solid math why this works
- We use this for the ReLU and Hinge loss so far

# Isn't an SVM far more desirable?

## SVM reminder

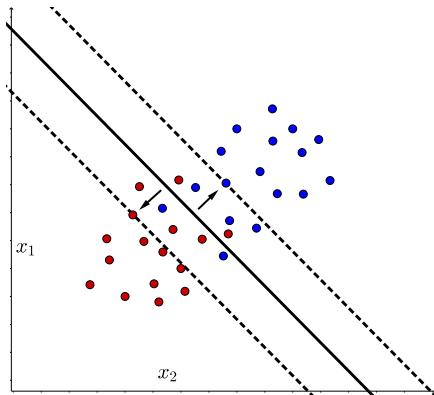


$$\min \quad \frac{1}{2} \|\mathbf{w}\|_2^2$$

$$s.t. \quad \forall n : -(y_i \cdot (\mathbf{w}^T \mathbf{x}_m) - 1) \leq 0$$

# Isn't an SVM far more desirable?

## SVM reminder



$$\begin{aligned}
 \min \quad & \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_i \xi_m \\
 \text{s.t.} \quad & \forall n : -(y_i \cdot (\mathbf{w}^T \mathbf{x}_m) - 1 + \xi_m) \leq 0 \\
 & \forall n : -\xi_m \leq 0
 \end{aligned}$$

## Isn't a SVM far more desirable?

- We construct the Lagrangian dual function

$$L(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \xi_m + \sum_{m=1}^M \lambda_m (-y_m \cdot (\mathbf{w}^T \mathbf{x}_m) + 1 - \xi_m) - \sum_{m=1}^M \nu_m \xi_m$$



## Isn't a SVM far more desirable?

- We construct the Lagrangian dual function

$$\begin{aligned} L(\mathbf{w}) &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \xi_m + \sum_{m=1}^M \lambda_m (-y_m \cdot (\mathbf{w}^T \mathbf{x}_m) + 1 - \xi_m) - \sum_{m=1}^M \nu_m \xi_m \\ &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{m=1}^M (\gamma \xi_m - \nu_m \xi_m - \lambda_m \xi_m) + \sum_{m=1}^M \lambda_m (1 - y_m \cdot (\mathbf{w}^T \mathbf{x}_m)) \end{aligned}$$

## Isn't a SVM far more desirable?

- We construct the Lagrangian dual function
- Remember:  $\lambda_m \geq 0$

$$\begin{aligned} L(\mathbf{w}) &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \xi_m + \sum_{m=1}^M \lambda_m (-y_m \cdot (\mathbf{w}^T \mathbf{x}_m) + 1 - \xi_m) - \sum_{m=1}^M \nu_m \xi_m \\ &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{m=1}^M (\gamma \xi_m - \nu_m \xi_m - \lambda_m \xi_m) + \sum_{m=1}^M \lambda_m (1 - y_m \cdot (\mathbf{w}^T \mathbf{x}_m)) \end{aligned}$$

## Isn't a SVM far more desirable?

- We construct the Lagrangian dual function
- Remember:  $\lambda_m \geq 0$

$$\begin{aligned} L(\mathbf{w}) &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \xi_m + \sum_{m=1}^M \lambda_m (-y_m \cdot (\mathbf{w}^T \mathbf{x}_m) + 1 - \xi_m) - \sum_{m=1}^M \nu_m \xi_m \\ &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{m=1}^M (\gamma \xi_m - \nu_m \xi_m - \lambda_m \xi_m) + \sum_{m=1}^M \lambda_m (1 - y_m \cdot (\mathbf{w}^T \mathbf{x}_m)) \\ &\approx \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \max(1 - y_m \cdot (\mathbf{w}^T \mathbf{x}_m)) \end{aligned}$$

## Isn't a SVM far more desirable?

- We construct the Lagrangian dual function
- Remember:  $\lambda_m \geq 0$
- Equivalent "up to an overall multiplicative constant"[1]

$$\begin{aligned} L(\mathbf{w}) &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \xi_m + \sum_{m=1}^M \lambda_m (-y_m \cdot (\mathbf{w}^T \mathbf{x}_m) + 1 - \xi_m) - \sum_{m=1}^M \nu_m \xi_m \\ &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{m=1}^M (\gamma \xi_m - \nu_m \xi_m - \lambda_m \xi_m) + \sum_{m=1}^M \lambda_m (1 - y_m \cdot (\mathbf{w}^T \mathbf{x}_m)) \\ &\approx \frac{1}{2} \underbrace{\|\mathbf{w}\|_2^2}_{\text{L2 regularizer}} + \gamma \sum_{m=1}^M \underbrace{\max(1 - y_m \cdot (\mathbf{w}^T \mathbf{x}_m))}_{\text{Hinge loss}} \end{aligned}$$

## Open points

## Open points

### Outliers are punished linearly

## Open points

### Outliers are punished linearly

- A variant of the hinge loss which penalizes outliers more strongly [4]:

$$L(\mathbf{w}) = \sum_{m=1}^M (\max(0, 1 - y_m \hat{y}_m(\mathbf{x}_m, \mathbf{w})))^2$$

## Open points

### Outliers are punished linearly

- A variant of the hinge loss which penalizes outliers more strongly [4]:

$$L(\mathbf{w}) = \sum_{m=1}^M (\max(0, 1 - y_m \hat{y}_m(\mathbf{x}_m, \mathbf{w})))^2$$

### How to apply SVMs to multi-class problems?



## Open points

### Outliers are punished linearly

- A variant of the hinge loss which penalizes outliers more strongly [4]:

$$L(\mathbf{w}) = \sum_{m=1}^M (\max(0, 1 - y_m \hat{y}_m(\mathbf{x}_m, \mathbf{w})))^2$$

### How to apply SVMs to multi-class problems?

- A Hinge loss for multi-class problems[9]:

$$L(\mathbf{w}) = \sum_{m=1}^M \sum_{k \neq c}^K \max(0, 1 - \hat{y}_c(\mathbf{x}_m, \mathbf{w}) + \hat{y}_k(\mathbf{x}_m, \mathbf{w}))$$

## Summary

- We have seen we can incorporate a SVM into a neural network
- See [4] for a reference using this
- We've learned before how to deal with the non-smooth objective



FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Optimization



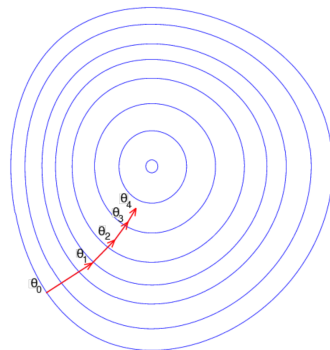
# Gradient Descent revisited

**Goal:** Optimize empirical risk:

$$\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}(\mathbf{x}, \mathbf{y})} [L(\mathbf{w}, \mathbf{x}, \mathbf{y})] = \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

$$\mathbf{w}^{k+1} = \mathbf{w}^{(k)} - \eta \nabla L(\mathbf{w}^{(k)}, \mathbf{x}, \mathbf{y})$$

- Step size defined by learning rate  $\eta$
- Gradient with respect to **every** sample
- Guaranteed to converge to a **local minimum**



## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples

## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples
  - Preferred option for convex problems
  - Updates are guaranteed to **decrease** the error

## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples
  - Preferred option for convex problems
  - Updates are guaranteed to **decrease** the error
  - Problem non-convex anyway & memory limitations

## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples
  - Preferred option for convex problems
  - Updates are guaranteed to **decrease** the error
  - Problem non-convex anyway & memory limitations
- Stochastic (Online) Gradient Descent (SGD): Use 1 sample
  - No longer necessarily decreases the empirical risk in every iteration
  - **Inefficient** because of transfer latency to GPU



## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples
  - Preferred option for convex problems
  - Updates are guaranteed to **decrease** the error
  - Problem non-convex anyway & memory limitations
- Stochastic (Online) Gradient Descent (SGD): Use 1 sample
  - No longer necessarily decreases the empirical risk in every iteration
  - **Inefficient** because of transfer latency to GPU

## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples
  - Preferred option for convex problems
  - Updates are guaranteed to **decrease** the error
  - Problem non-convex anyway & memory limitations
- Stochastic (Online) Gradient Descent (SGD): Use 1 sample
  - No longer necessarily decreases the empirical risk in every iteration
  - **Inefficient** because of transfer latency to GPU
- Mini-Batch SGD: Use  $B \ll M$  **random** samples

$$\mathbf{g}^{(k)} := \nabla L(\mathbf{w}^{(k)}) = \frac{1}{B} \nabla \sum_{b=1}^B L(\mathbf{w}^{(k)}, \mathbf{x}_b)$$

## Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all  $M$  samples
  - Preferred option for convex problems
  - Updates are guaranteed to **decrease** the error
  - Problem non-convex anyway & memory limitations
- Stochastic (Online) Gradient Descent (SGD): Use 1 sample
  - No longer necessarily decreases the empirical risk in every iteration
  - **Inefficient** because of transfer latency to GPU
- Mini-Batch SGD: Use  $B \ll M$  **random** samples

$$\mathbf{g}^{(k)} := \nabla L(\mathbf{w}^{(k)}) = \frac{1}{B} \nabla \sum_{b=1}^B L(\mathbf{w}^{(k)}, \mathbf{x}_b)$$

- Small batches offer regularization effect  $\rightarrow$  need smaller  $\eta$
- Regains efficiency  $\rightarrow$  the standard case in deep learning

## How can this even work?

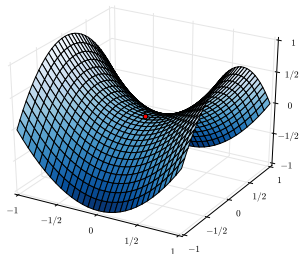
- Optimization problem is non-convex
- Exponential number of local minima

## How can this even work?

- Optimization problem is non-convex
- Exponential number of local minima

## Possible Answers (Choromanska et al. 2015, Dauphin et al. 2014)

- High dimensional function
  - Local minima exist but very close to global minima
  - ... and many of those are equivalent
- Presumably more critical: saddle points
- Local minimum might be better than global minima (overfitting!)



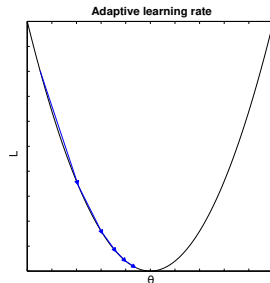
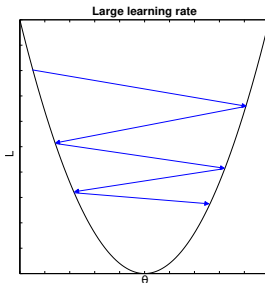
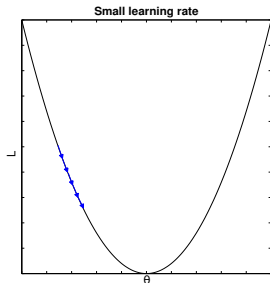
Source: [https://upload.wikimedia.org/wikipedia/commons/1/1e/Saddle\\_point.svg](https://upload.wikimedia.org/wikipedia/commons/1/1e/Saddle_point.svg)

## Another possible answer

### Possible answer (Percy Liang, NIPS 2016)

- “overprovisioning”
- Many different ways how a network can approximate the desired relationship
- Only needs to find one
- This has been verified experimentally by learning **random** labels[10]

## SGD – Learning Rate Choice



- $\eta$  too small: long training time
- $\eta$  too large: miss optima
- Practice: “learning rate decay”: adapt  $\eta$  gradually (e.g.: start with  $\eta = 0.01$  and divide every  $x$  epoch by 10)

# Can't we get rid of this magic $\eta$ ?



# Can't we get rid of this magic $\eta$ ?

## By performing line search?

## Can't we get rid of this magic $\eta$ ?

### By performing line search?

- Multiple evaluations necessary, while we could take multiple steps
- The direction is extremely noisy anyway
- Still people have presented methods[8]

## Can't we get rid of this magic $\eta$ ?

### By performing line search?

- Multiple evaluations necessary, while we could take multiple steps
- The direction is extremely noisy anyway
- Still people have presented methods[8]

### By second order methods?

$$\mathbf{w}^{k+1} = \mathbf{w}^{(k)} - H\left(L(\mathbf{w}^{(k)})\right)^{-1} \nabla L(\mathbf{w}^{(k)})$$

## Can't we get rid of this magic $\eta$ ?

### By performing line search?

- Multiple evaluations necessary, while we could take multiple steps
- The direction is extremely noisy anyway
- Still people have presented methods[8]

### By second order methods?

$$\mathbf{w}^{k+1} = \mathbf{w}^{(k)} - H\left(L(\mathbf{w}^{(k)})\right)^{-1} \nabla L(\mathbf{w}^{(k)})$$

- The Hessian matrix  $H\left(L(\mathbf{w}^{(k)})\right)$  is too expensive to calculate

## Can't we get rid of this magic $\eta$ ?

### By performing line search?

- Multiple evaluations necessary, while we could take multiple steps
- The direction is extremely noisy anyway
- Still people have presented methods[8]

### By second order methods?

$$\mathbf{w}^{k+1} = \mathbf{w}^{(k)} - H\left(L(\mathbf{w}^{(k)})\right)^{-1} \nabla L(\mathbf{w}^{(k)})$$

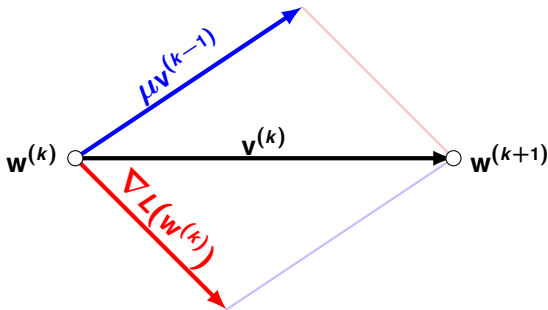
- The Hessian matrix  $H\left(L(\mathbf{w}^{(k)})\right)$  is too expensive to calculate
- L-BFGS doesn't perform well outside of batch settings
- A report on this was presented by Google[7]

## What can we do?

Idea: Accelerate in directions with persistent gradients

## What can we do?

Idea: Accelerate in directions with persistent gradients



## Momentum

- Parameter update based on current and past gradients:

$$\mathbf{v}^{(k)} = \underbrace{\mu}_{\text{momentum}} \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)})$$
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{v}^{(k)}$$



## Momentum

- Parameter update based on current and past gradients:

$$\mathbf{v}^{(k)} = \underbrace{\mu}_{\text{momentum}} \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)})$$
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{v}^{(k)}$$

- commonly:  $\mu = \{0.9, 0.95, 0.99\}$  (or adaptive: small  $\rightarrow$  large)

## Momentum

- Parameter update based on current and past gradients:

$$\mathbf{v}^{(k)} = \underbrace{\mu}_{\text{momentum}} \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)})$$
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{v}^{(k)}$$

- commonly:  $\mu = \{0.9, 0.95, 0.99\}$  (or adaptive: small  $\rightarrow$  large)
- + Overcomes poor Hessian & variance in SGD  $\rightarrow$  dampened oscillations
- + Acceleration

## Momentum

- Parameter update based on current and past gradients:

$$\mathbf{v}^{(k)} = \underbrace{\mu}_{\text{momentum}} \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)})$$
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{v}^{(k)}$$

- commonly:  $\mu = \{0.9, 0.95, 0.99\}$  (or adaptive: small  $\rightarrow$  large)
- + Overcomes poor Hessian & variance in SGD  $\rightarrow$  dampened oscillations
- + Acceleration
- Still learning rate decay needed!

## Nesterov Accelerated Gradient (NAG) / Nesterov Momentum

- "Look ahead" - compute the gradient in the direction we're going anyway!

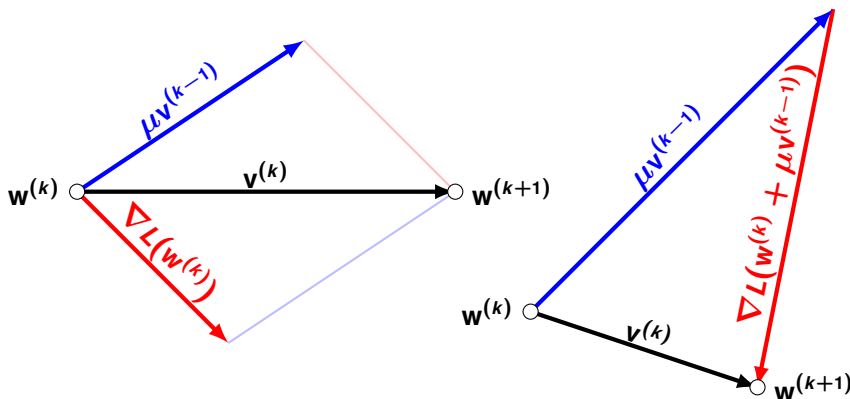
$$\mathbf{v}^{(k)} = \mu \mathbf{v}^{(k-1)} - \eta \nabla L(\underbrace{\mathbf{w}^{(k)} + \mu \mathbf{v}^{(k-1)}}_{\text{approx. of next parameters}})$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{v}^{(k)}$$

- We can rewrite this to use the conventional gradient:

$$\begin{aligned}\mathbf{v}^{(k)} &= \mu \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)}) \\ \mathbf{w}^{(k+1)} &= -\mu \mathbf{v}^{(k-1)} + (1 + \mu) \mathbf{w}^{(k)}\end{aligned}$$

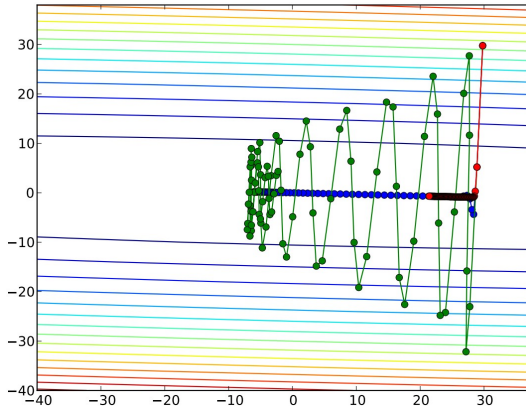
## How does this compare to momentum?



Momentum

Nesterov Momentum

## Example for an advantage of NAG



GD (red), momentum (green), NAG (blue)

Source: Sutskever "Training Recurrent Neural Networks", p. 76

## What if our features have different needs?

## What if our features have different needs?

- Suppose some features are activated very infrequently
- ... while others are updated very often



## What if our features have different needs?

- Suppose some features are activated very infrequently
- ... while others are updated very often
- We'd need individual learning rates for every parameter in the network
- Large (small) learning rates for infrequent (frequent) parameters and parameters with small (large) gradient magnitudes

## AdaGrad

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} + \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\eta}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}$$

- **Adaptive Gradient**
- Adaption based on all past squared gradients
- We use  $\odot$  to emphasize the element-wise multiplication

## AdaGrad

$$\begin{aligned}\mathbf{g}^{(k)} &= \nabla L(\mathbf{w}^{(k)}) \\ \mathbf{r}^{(k)} &= \mathbf{r}^{(k-1)} + \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)} \\ \mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} - \frac{\eta}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}\end{aligned}$$

- **Adaptive Gradient**
  - Adaption based on all past squared gradients
  - We use  $\odot$  to emphasize the element-wise multiplication
- + Individual learning rates

## AdaGrad

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} + \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\eta}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}$$

- **Adaptive Gradient**
- Adaption based on all past squared gradients
- We use  $\odot$  to emphasize the element-wise multiplication
- + Individual learning rates
- Learning rate decreases too aggressively

## RMSProp

$$\begin{aligned}\mathbf{g}^{(k)} &= \nabla L(\mathbf{w}^{(k)}) \\ \mathbf{r}^{(k)} &= \rho \mathbf{r}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)} \\ \mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} - \frac{\eta}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}\end{aligned}$$

- Hinton suggests  $\rho = 0.9$ ,  $\eta = 0.001$
- + The aggressive decrease is fixed
- We still have to set the learning rate

## Adadelta

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{r}^{(k)} = \rho \mathbf{r}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\mathbf{h}^{(k)} = \rho \mathbf{g}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\sqrt{\mathbf{h}^{(k-1)}} + \epsilon}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}$$

- Suggested:  $\rho = 0.95$

+ No learning rate

# Adam

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{v}^{(k)} = \mu \mathbf{v}^{(k-1)} + (1 - \mu) \mathbf{g}^{(k)}$$

$$\mathbf{r}^{(k)} = \rho \mathbf{r}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\text{Bias correction: } \hat{\mathbf{v}}^{(k)} = \frac{\mathbf{v}^{(k)}}{1 - \mu^k} \quad \hat{\mathbf{r}}^{(k)} = \frac{\mathbf{r}^{(k)}}{1 - \rho^k}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \frac{\hat{\mathbf{v}}^{(k)} + \epsilon}{\sqrt{\hat{\mathbf{r}}^{(k)} + \epsilon}}$$

- Short for **A**daptive **M**oment Estimation
- Suggested:  $\mu = 0.9, \rho = 0.999, \eta = 0.001$
- + Robustness
- Combination w. NAG exists (“Nadam”)

## AMSGrad

- Adam empirically observed to fail to converge to an optimal/good solution



## AMSGrad

- Adam empirically observed to fail to converge to an optimal/good solution
- Recent insight by Reddi et al. [5]: Adam (and similar methods) **do not guarantee** convergence for convex problems (error in original convergence proof)
- AMSGrad [5] “fixes” Adam to ensure non-increasing step size:

$$\hat{\mathbf{v}}^{(k)} = \max(\hat{\mathbf{v}}^{(k-1)}, \mathbf{v}^{(k)})$$

## AMSGrad

- Adam empirically observed to fail to converge to an optimal/good solution
- Recent insight by Reddi et al. [5]: Adam (and similar methods) **do not guarantee** convergence for convex problems (error in original convergence proof)
- AMSGrad [5] “fixes” Adam to ensure non-increasing step size:

$$\hat{\mathbf{v}}^{(k)} = \max(\hat{\mathbf{v}}^{(k-1)}, \mathbf{v}^{(k)})$$

- Effect has to be shown in larger experiments
- Lesson: Keep your eyes open!

## Summary

- SGD + Nesterov momentum + learning rate decay
  - + Often converges most reliably
  - + Still used in many state-of-the-art papers
  - Learning rate decay needs to be adjusted
- Adam
  - + Individual learning rates
  - + Learning rate very well behaved
  - Loss curves harder to interpret
- **Not discussed:** Distributed gradient descend

## Practical recommendations

- Start by using minibatch SGD with momentum
- Mostly keep to the default momentum
- Give Adam a try when you have a feeling for your data
- When in need for individual learning rates use Adam
- Start by using the default parameters for Adam
- Adjust the learning rate first
- Keep your eyes open for unusual behavior (see AMSGrad)

**NEXT TIME**

**ON DEEP LEARNING**

## Coming Up

- How can we deal with spatial correlation in features?
- Why do we hear so much about convolution in neural networks?
- How can we incorporate invariances into network architectures?

## Comprehensive Questions

- What are our standard loss functions for classification and regression?
- What assumptions do our standard loss functions imply?
- What is a subdifferential at a point  $\mathbf{x}_0$ ?
- How can we optimize a non-smooth convex function?
- What if somebody tells you, to use an SVM because it is superior?
- What is Nesterov Momentum?
- Describe Adam.

## Further Reading

- [Link](#) - for details on Maximum Likelihood estimation and the basic loss functions.
- [Link](#) - [6] for insights about some loss functions
- [Link](#) - [10] for a troubling insight, that deep networks can learn arbitrary random labels





FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# References



## References I

- [1] Christopher M. Bishop.  
Pattern Recognition and Machine Learning (Information Science and Statistics)  
Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [2] Anna Choromanska, Mikael Henaff, Michael Mathieu, et al. “The Loss Surfaces of Multilayer Networks.”. In: AISTATS. 2015.
- [3] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: Advances in neural information processing systems. 2014, pp. 2933–2941.
- [4] Yichuan Tang. “Deep learning using linear support vector machines”. In: arXiv preprint arXiv:1306.0239 (2013).

## References II

- [5] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. “On the Convergence of Adam and Beyond”. In: International Conference on Learning Representations. 2018.
- [6] Katarzyna Janocha and Wojciech Marian Czarnecki. “On Loss Functions for Deep Neural Networks in Classification”. In: arXiv preprint arXiv:1702.05659 (2017).
- [7] Jeffrey Dean, Greg Corrado, Rajat Monga, et al. “Large scale distributed deep networks”. In: Advances in neural information processing systems. 2012, pp. 1223–1231.
- [8] Maren Mahsereci and Philipp Hennig. “Probabilistic line searches for stochastic optimization”. In: Advances In Neural Information Processing Systems. 2015, pp. 181–189.

## References III

- [9] Jason Weston, Chris Watkins, et al. "Support vector machines for multi-class pattern recognition.". In: ESANN. Vol. 99. 1999, pp. 219–224.
- [10] Chiyuan Zhang, Samy Bengio, Moritz Hardt, et al. "Understanding deep learning requires rethinking generalization". In: arXiv preprint arXiv:1611.03530 (2016).