# Funkcija Greške i Optimizacija

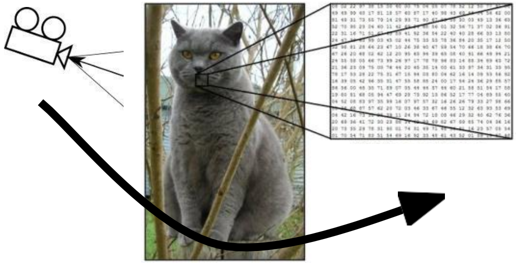
Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

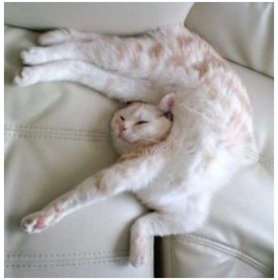
# Sa prošlog predavanja … Izazovi kod klasifikacije slika

Pozcija kamere



Osvetljenje

Deformacija

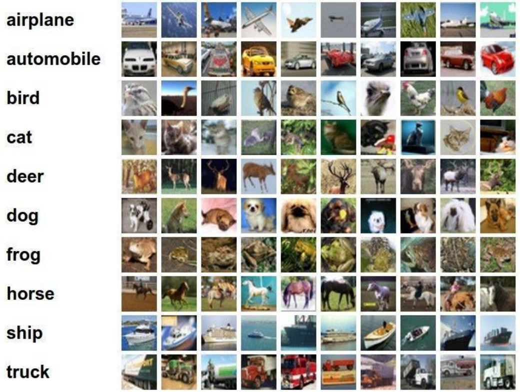


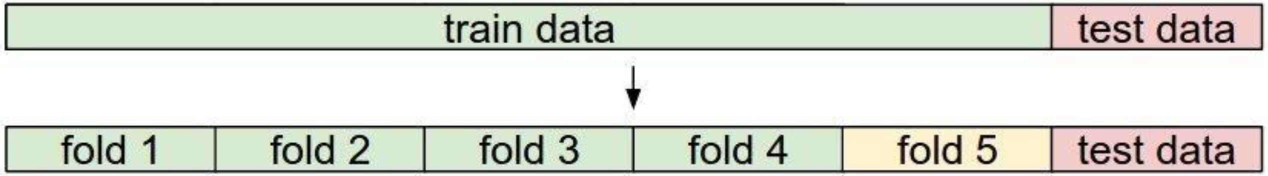
Okluzija

Pretrpana pozadina

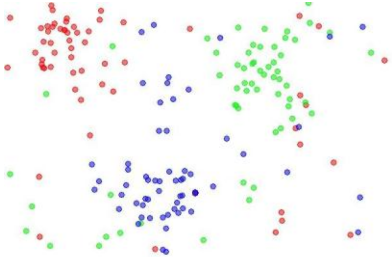
Varijabilnost unutra klase

# **Sa prošlog predavanja…** nadgledano učenje, kNN

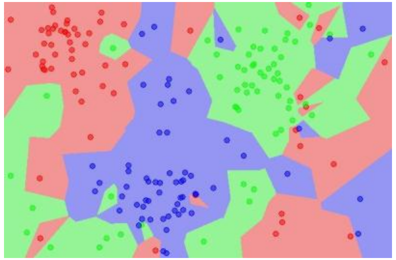


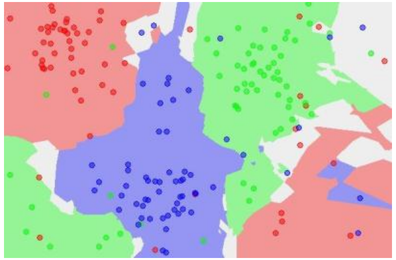

the data

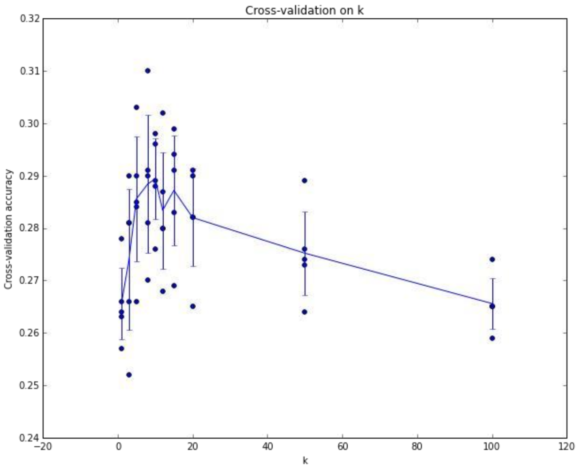


NN classifier



5-NN classifier





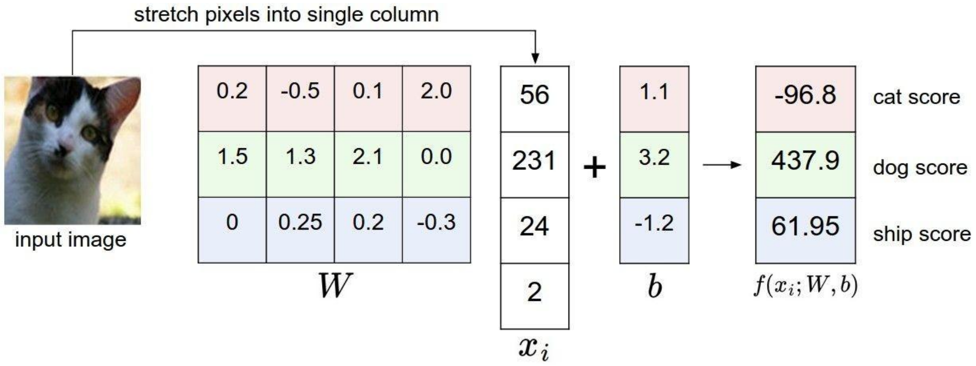
# **Sa prošlog predavanja …** Linearni klasifikator

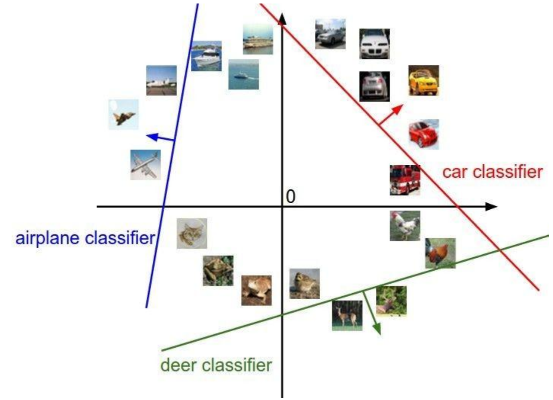
**[32x32x3]**
niz od 3072 brojeava

slika parametri

f(**x**,**W**)

**10** brojeva koji su skorovi klasa




# Sa prošlog predavanja … Danas: Funkcija greške /Optimizacija

| | | | |
|---|---|---|---|
| **airplane** | -3.45 | -0.51 | 3.42 |
| **automobile** | -8.87 | **6.04** | 4.64 |
| **bird** | 0.09 | 5.31 | 2.65 |
| **cat** | **2.9** | -4.22 | 5.1 |
| **deer** | 4.48 | -4.19 | 2.64 |
| **dog** | 8.02 | 3.58 | 5.55 |
| **frog** | 3.78 | 4.49 | **-4.34** |
| **horse** | 1.06 | -4.37 | -1.5 |
| **ship** | -0.36 | -2.09 | -4.79 |
| **truck** | -0.72 | -2.93 | 6.14 |

TODO:

1. Definisanje funkcije greške (***loss function***) koja kvantifikuje kvalitet skorova za slike iz obučavajućeg skupa

2. Nađemo način da automatski odredimo W i b tako da minimizujemo funkciju greške. **(optimizacija)**

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x,W) = Wx$ su:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$

funkcija greške za SVM je oblika:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | | | |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$

funkcija greške za SVM je oblika:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

$= \max(0, 5.1 - 3.2 + 1)$
$\quad + \max(0, -1.7 - 3.2 + 1)$
$= \max(0, 2.9) + \max(0, -3.9)$
$= 2.9 + 0$
$= 2.9$

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$

funkcija greške za SVM je oblika:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | 10.9 |

funkcija greške za SVM je oblika:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x,W) = Wx$ su:

| |  |  |  |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | 10.9 |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$
funkcija greške za SVM je oblika:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Greška za sve tri slike je prosek grešaka:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 + 0 + 10.9)/3
= **4.6**

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | 10.9 |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$
funkcija greške za SVM je oblika:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Šta bi bilo ko bi $L_i$ računali za sve klase tj. i za $j=y_i$?

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x,W) = Wx$ su:

| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | 10.9 |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$
funkcija greške za SVM je oblika:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Šta bi bilo ako bi korsitili:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x,W) = Wx$ su:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | 10.9 |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i,y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$
funkcija greške za SVM je oblika:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Koje su min i max vrednosti za funkciju greške?

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Greške: | 2.9 | 0 | 10.9 |

## Fukcija greške modela SVM za više klasa:

Za primer $(x_i, y_i)$
gde je $x_i$ slika
gde je $y_i$ klasa (ceo broj) ,
Gde se skor računa kao: $s = f(x_i, W)$
funkcija greške za SVM je oblika:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Obično se vrednosti W inicijalizuju na jako male brojeve ~= 0.
Koje su tada vrednosti funcije greške?

numpy kod za SVM primer:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

# Naša funkcija greške ima bag:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

Naša funkcija greške ima bag:

$$f(x,W) = Wx$$

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1)$$

Recimo da smo pronašli W tako da je L = 0.
Da li je to W jedinstveno?

Na primer: 3 slike u obučavajućem skupu, 3 klase.
Za neko W skorovi $f(x, W) = Wx$ su:



| | | | |
|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 |
| car | 5.1 | **4.9** | 2.5 |
| frog | -1.7 | 2.0 | **-3.1** |
| Losses: | 2.9 | 0 | |

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

**Ranije:**

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

**Sa duplo većim W:**

= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

# Regularizacija težina

lambda = jačina regularizacije
(hiper-parametar)

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Tipično se koristi:

**L2 regularizacija** $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 **regularizacija** $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Dropout (kasnije tokom kursa)

# L2 regularizacija: motivacija

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

**skorovi = nenormalizovane log verovatnoće klasa**
**(nenormalizovane znači da se ne sabiraju na 1)**

$$\boxed{s = f(x_i; W)}$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

**skorovi = nenormalizovane log verovatnoće klasa**

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad s = f(x_i; W)$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

**skorovi = nenormalizovane log verovatnoće klasa**

$$P(Y=k|X=x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad s = f(x_i; W)$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Softmax funkcija

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

**skorovi = nenormalizovane log verovatnoće klasa**

$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}} \quad s=f(x_i;W)$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Maksimizujemo log verovatnost (log likelihood) – obično se minimizuje negativna log verovatnost:

$$L_i=-\log P(Y=y_i|X=x_i)$$

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

**skorovi = nenormalizovane log verovatnoće klasa**

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad s = f(x_i; W)$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Maksimizujemo log verovatnost (log likelihood) – obično se minimizuje negativna log verovatnost:

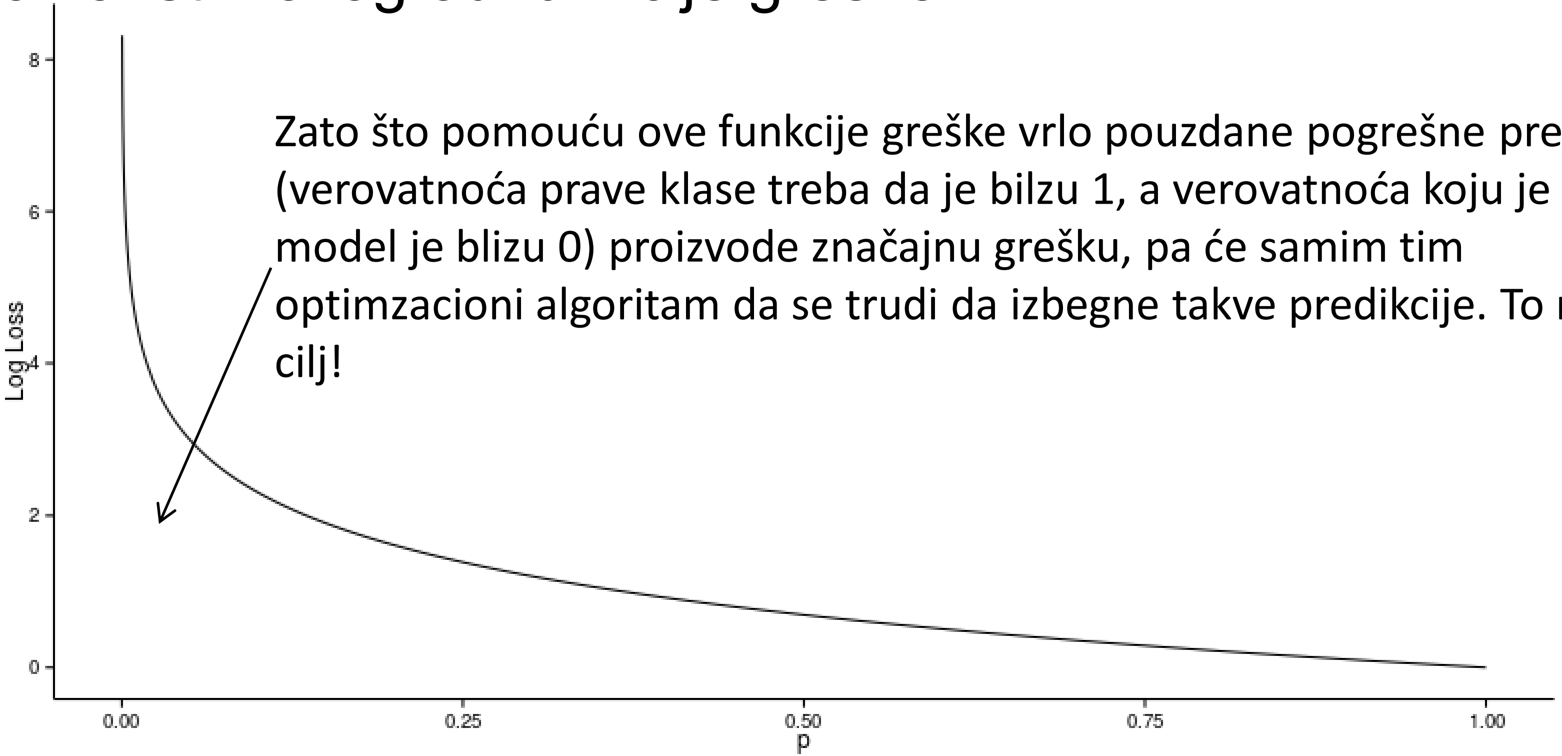
$$L_i = -\log P(Y=y_i|X=x_i)$$

---

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

## Zašto koristimo log od funkcije greške?




Zato što pomouću ove funkcije greške vrlo pouzdane pogrešne predikcije (verovatnoća prave klase treba da je bilzu 1, a verovatnoća koju je dao model je blizu 0) proizvode značajnu grešku, pa će samim tim optimzacioni algoritam da se trudi da izbegne takve predikcije. To nam je i cilj!

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

nenormalizovane log verovatnoće

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

nenormalizovane verovatnoće

| | | | |
|---|---|---|---|
| cat | **3.2** | | **24.5** |
| car | 5.1 | $\xrightarrow{\text{exp}}$ | 164.0 |
| frog | -1.7 | | 0.18 |

nenormalizovane log verovatnoće

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

nenormalizovane verovatnoće

| | | | | |
|---|---|---|---|---|
| cat | **3.2** | | **24.5** | **0.13** |
| car | 5.1 | exp → | 164.0 | normalizacija → 0.87 |
| frog | -1.7 | | 0.18 | 0.00 |

nemormalizovane log verovatnoće

verovatnoće

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

nenormalizovane verovatnoće

| | | exp | | normalizacija | | |
|---|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** | → L_i = -log(0.13) = **0.89** |
| car | 5.1 | | 164.0 | | 0.87 | |
| frog | -1.7 | | 0.18 | | 0.00 | |

nenormalizovane log verovatnoće

verovatnoće

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Koje su moguće min i max vrednosti za L_i?

nenormalizovane verovatnoće

| | nenormalizovane log verovatnoće | exp | normalizacija → verovatnoće |
|---|---|---|---|
| cat | **3.2** | **24.5** | **0.13** |
| car | 5.1 | 164.0 | 0.87 |
| frog | -1.7 | 0.18 | 0.00 |

→ L_i = -log(0.13)
= **0.89**

# **Softmax Klasifikator** (Multinominalna Logistička Regresija)

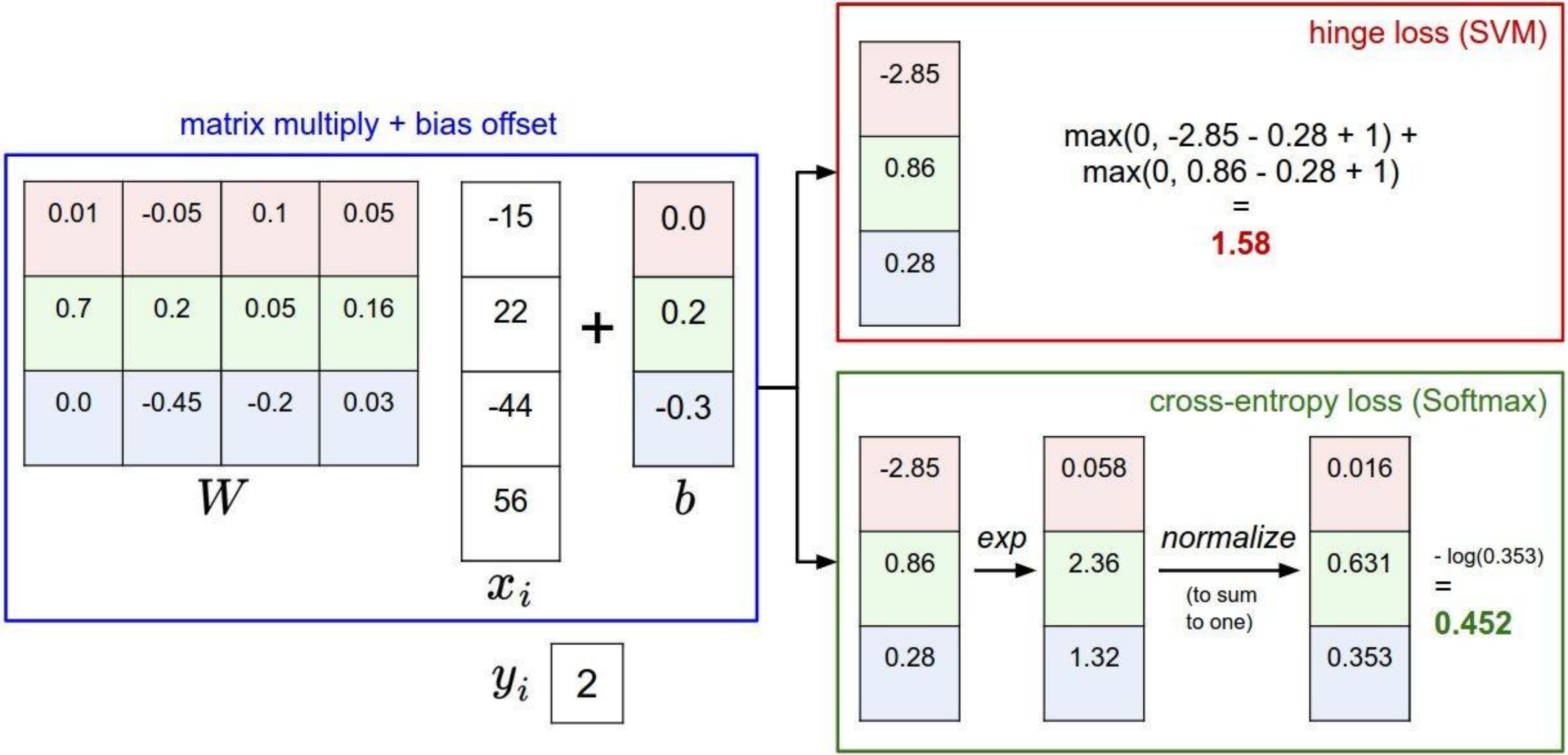
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Obično se vrednosti W inicijalizuju na jako male brojeve ~= 0. Koje su tada vrednosti funcije greške?

nemormalizovane verovatnoće

| | nemormalizovane log verovatnoće | exp | normalizacija | verovatnoće |
|---|---|---|---|---|
| cat | **3.2** | **24.5** | | **0.13** |
| car | 5.1 | 164.0 | | 0.87 |
| frog | -1.7 | 0.18 | | 0.00 |

→ L_i = -log(0.13) = **0.89**


matrix multiply + bias offset
0.01
-0.05
0.1
0.05
0.7
0.2
0.05
0.16
0.0
-0.45
-0.2
0.03
W
-15
22
-44
56
$x_i$
+
0.0
0.2
-0.3
b
$y_i$
2
hinge loss (SVM)
-2.85
0.86
0.28
max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum
to one)
0.016
0.631
0.353
- log(0.353)
=
0.452

# Softmax vs. SVM

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

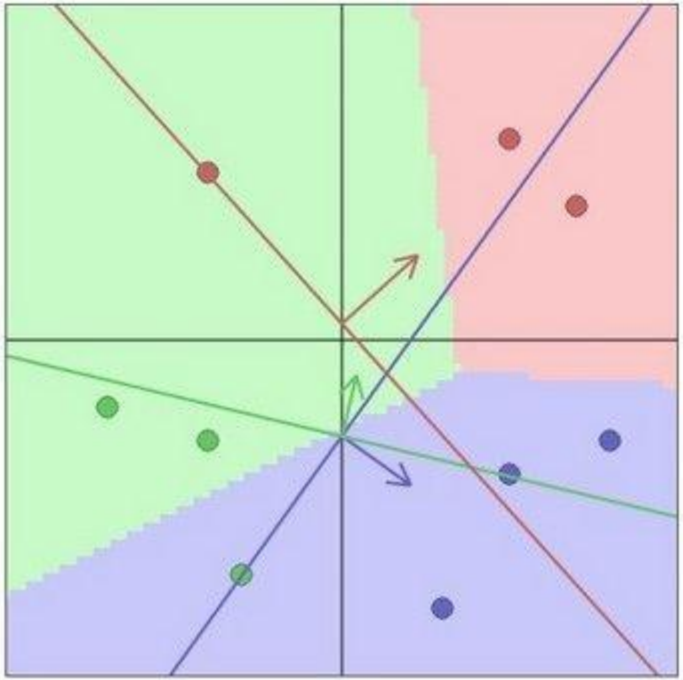
# Softmax vs. SVM

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad\qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

recimo da imamo skorove:
[10, -2, 3]
[10, 9, 9]
[10, -100, -100]
and $\boxed{y_i = 0}$

Šta se dešava sa obe funkcije greške ako uzmemo jednu tačku i malo je pomeramo u prostoru?

# Interaktivni Web Demo



| W[0,0] | W[0,1] | b[0] |
|---|---|---|
| 2.06 / -0.12 | 1.48 / 0.11 | -0.42 / 0.00 |

| W[1,0] | W[1,1] | b[1] |
|---|---|---|
| 0.44 / 0.19 | -1.82 / -0.37 | 0.52 / 0.11 |

| W[2,0] | W[2,1] | b[2] |
|---|---|---|
| 2.27 / 0.17 | -2.04 / 0.14 | -0.10 / -0.11 |

Step size: 0.10000

Single parameter update

Start repeated update

Stop repeated update

Randomize parameters

| x[0] | x[1] | y | s[0] | s[1] | s[2] | L |
|---|---|---|---|---|---|---|
| 0.50 | 0.40 | 0 | 1.20 | 0.01 | 0.22 | 0.02 |
| 0.80 | 0.30 | 0 | 1.67 | 0.33 | 1.10 | 0.44 |
| 0.30 | 0.80 | 0 | 1.38 | -0.80 | -1.05 | 0.00 |
| -0.40 | 0.30 | 1 | -0.80 | -0.20 | -1.62 | 0.39 |
| -0.30 | 0.70 | 1 | -0.01 | -0.88 | -2.21 | 1.87 |
| -0.70 | 0.20 | 1 | -1.57 | -0.15 | -2.10 | 0.00 |
| 0.70 | -0.40 | 2 | 0.43 | 1.55 | 2.31 | 0.25 |
| 0.50 | -0.60 | 2 | -0.28 | 1.83 | 2.26 | 0.57 |
| -0.40 | -0.50 | 2 | -1.98 | 1.26 | 0.01 | 2.24 |
| | | | | | mean: | 0.64 |

Total data loss: 0.64
Regularization loss: 1.92
Total loss: 2.57

L2 Regularization strength: 0.10000

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

# Optimizacija

# Rezime

- Imamo skup parova (x,y)
- Imamo **skor funkciju:**
- Imamo **funkciju greške**:

$$s = f(x; W) \overset{\text{npr.}}{=} Wx$$

Softmax

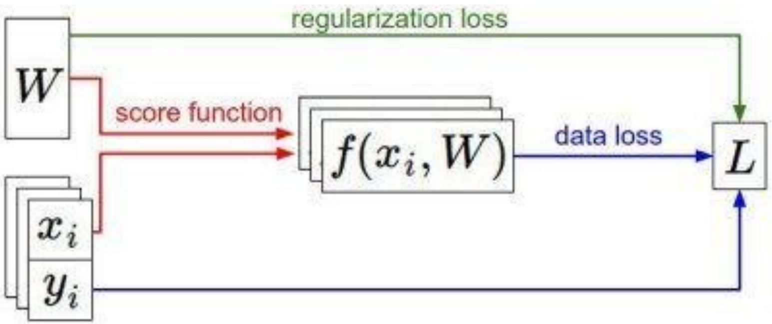
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$

Greška + Regularizacija



# Optimizacija - Strategija #1: **Random pretraga**

```python
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (trunctated: continues for 1000 lines)
```

# Šta dobijamo na test skupu...

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% tačnost! nije loše!
(State-of-the-Art, SOTA je ~95%)

# Strategija #2: **Pratimo nagib (*slope*)**

Kod funkcije jedne promenljive, izvod funkcije dat je sa:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Kad imamo više promenljivih (atributa), **gradijent** je vektor parcijalnih izvoda po svim atributima.

**W:**

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25347**

**gradijent dW:**

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

| **W:** | **W + h** (za prvi atribut)**:** | **gradijent dW:** |
|---|---|---|
| [0.34, | [0.34 + **0.0001**, | [?, |
| -1.11, | -1.11, | ?, |
| 0.78, | 0.78, | ?, |
| 0.12, | 0.12, | ?, |
| 0.55, | 0.55, | ?, |
| 2.81, | 2.81, | ?, |
| -3.1, | -3.1, | ?, |
| -1.5, | -1.5, | ?, |
| 0.33,…] | 0.33,…] | ?,…] |
| **greška 1.25347** | **greška 1.25322** | |

**W:**

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25347**

**W + h** (za prvi atribut)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25322**

**gradijent dW:**

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

?,
?,…]

| **W:** | **W + h** (drugi atribut)**:** | **gradijent dW:** |
|---|---|---|
| [0.34, | [0.34, | [-2.5, |
| -1.11, | -1.11 + **0.0001**, | ?, |
| 0.78, | 0.78, | ?, |
| 0.12, | 0.12, | ?, |
| 0.55, | 0.55, | ?, |
| 2.81, | 2.81, | ?, |
| -3.1, | -3.1, | ?, |
| -1.5, | -1.5, | ?, |
| 0.33,…] | 0.33,…] | ?,…] |
| **greška 1.25347** | **greška 1.25353** | |

**W:**

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25347**

**W + h** (drugi atribut)**:**

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25353**

## gradijent dW:

[-2.5,
**0.6**,
?,
?,

(1.25353 - 1.25347)/0.0001
= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

?,…]

| **W:** | **W + h** (treći atribut)**:** | **gradijent dW:** |
|---|---|---|
| [0.34, | [0.34, | [-2.5, |
| -1.11, | -1.11, | 0.6, |
| 0.78, | 0.78 + **0.0001**, | ?, |
| 0.12, | 0.12, | ?, |
| 0.55, | 0.55, | ?, |
| 2.81, | 2.81, | ?, |
| -3.1, | -3.1, | ?, |
| -1.5, | -1.5, | ?, |
| 0.33,…] | 0.33,…] | ?,…] |
| **greška 1.25347** | **greška 1.25347** | |

**W:**

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25347**

**W + h** (treći atribut):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**greška 1.25347**

**gradijent dW:**

[-2.5,
0.6,
**0**,
?,

(1.25347 - 1.25347)/0.0001
= 0

$$\boxed{\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

?,…]

# Numerički izvod

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

```python
def eval_numerical_gradient(f, x):
  """
  a naive implementation of numerical gradient of f at x
  - f should be a function that takes a single argument
  - x is the point (numpy array) to evaluate the gradient at
  """

  fx = f(x) # evaluate function value at original point
  grad = np.zeros(x.shape)
  h = 0.00001

  # iterate over all indexes in x
  it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
  while not it.finished:

    # evaluate function at x+h
    ix = it.multi_index
    old_value = x[ix]
    x[ix] = old_value + h # increment by h
    fxh = f(x) # evalute f(x + h)
    x[ix] = old_value # restore to previous value (very important!)

    # compute the partial derivative
    grad[ix] = (fxh - fx) / h # the slope
    it.iternext() # step to next dimension

  return grad
```

# Numerički izvod

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- aproksimacija
- vrlo sporo

```python
def eval_numerical_gradient(f, x):
  """
  a naive implementation of numerical gradient of f at x
  - f should be a function that takes a single argument
  - x is the point (numpy array) to evaluate the gradient at
  """

  fx = f(x) # evaluate function value at original point
  grad = np.zeros(x.shape)
  h = 0.00001

  # iterate over all indexes in x
  it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
  while not it.finished:

    # evaluate function at x+h
    ix = it.multi_index
    old_value = x[ix]
    x[ix] = old_value + h # increment by h
    fxh = f(x) # evalute f(x + h)
    x[ix] = old_value # restore to previous value (very important!)

    # compute the partial derivative
    grad[ix] = (fxh - fx) / h # the slope
    it.iternext() # step to next dimension

  return grad
```

Numerički izvod nije dobar pristup ovde. Greška je funkcija od W, pa možemo da koristimo analitički izvod.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

treba nam $\nabla_W L$

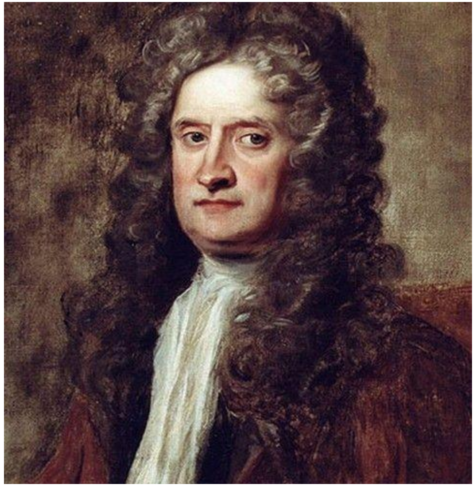
Numerički izvod nije dobar pristup ovde. Greška je funkcija od W, pa možemo da koristimo analitički izvod.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

treba nam $\nabla_W L$



Numerički izvod nije dobar pristup ovde. Greška je funkcija od W, pa možemo da koristimo analitički izvod.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

treba nam $\nabla_W L$

Analiza

Numerički izvod nije dobar pristup ovde. Greška je funkcija od W, pa možemo da koristimo analitički izvod.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

$\nabla_W L$ = ...

**W:**

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

**loss 1.25347**

dW = ...
(neka funkcija
od W)

**gradijent dW:**

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

# Rezime:

- Numerički izvod: aproksimacija, spor, ali jednostavna formula
- Analitički izvod: tačan, brz, nije ga uvek lako naći, pa postoji mogućnost greške

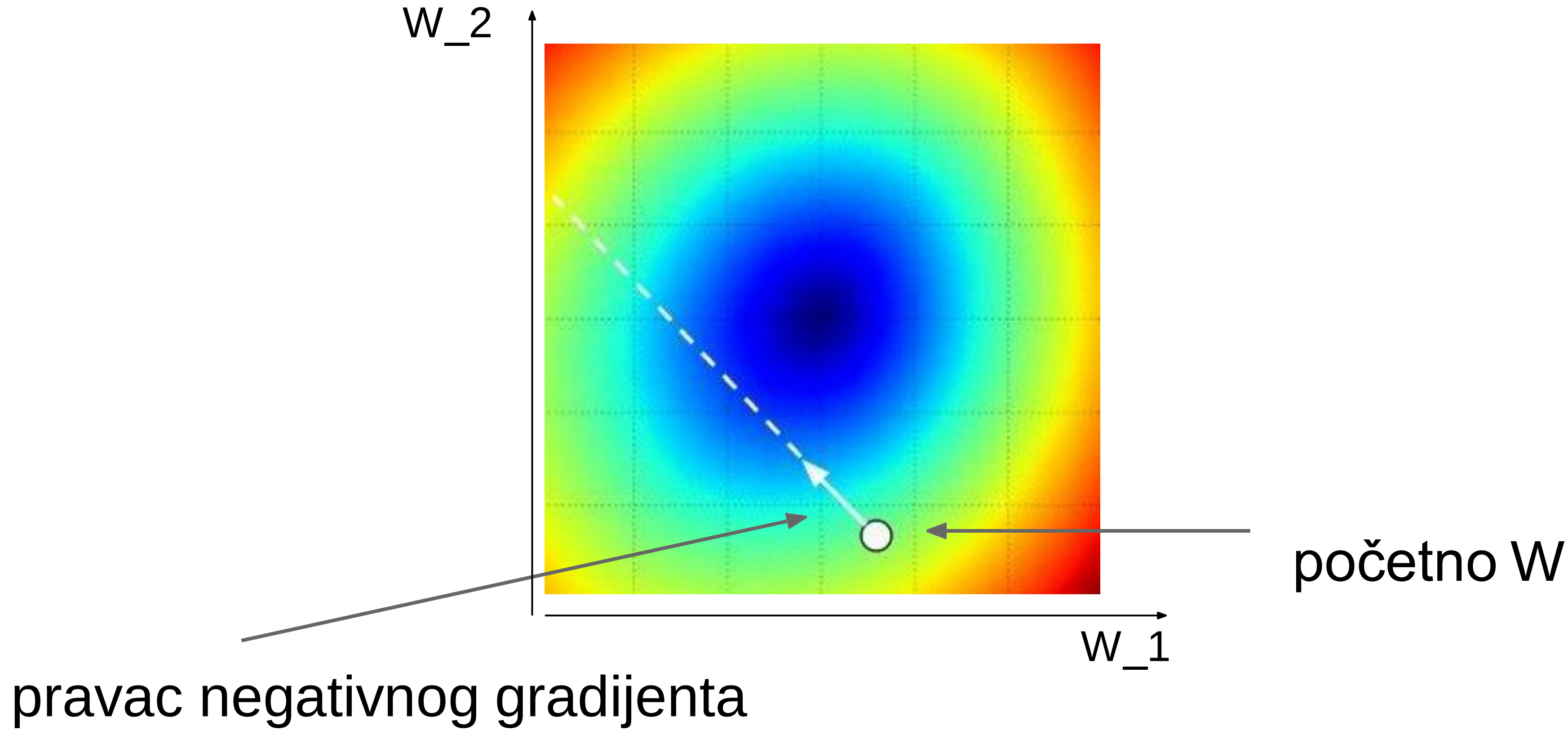
=>

U praksi: Uvek koristimo analitički izvod, ali ga proverimo numerički. Ovo se zove provera gradijenta (***gradient check***).

# Gradijetni Supst (*Gradient Descent*)

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```


W_2
W_1
početno W
pravac negativnog gradijenta

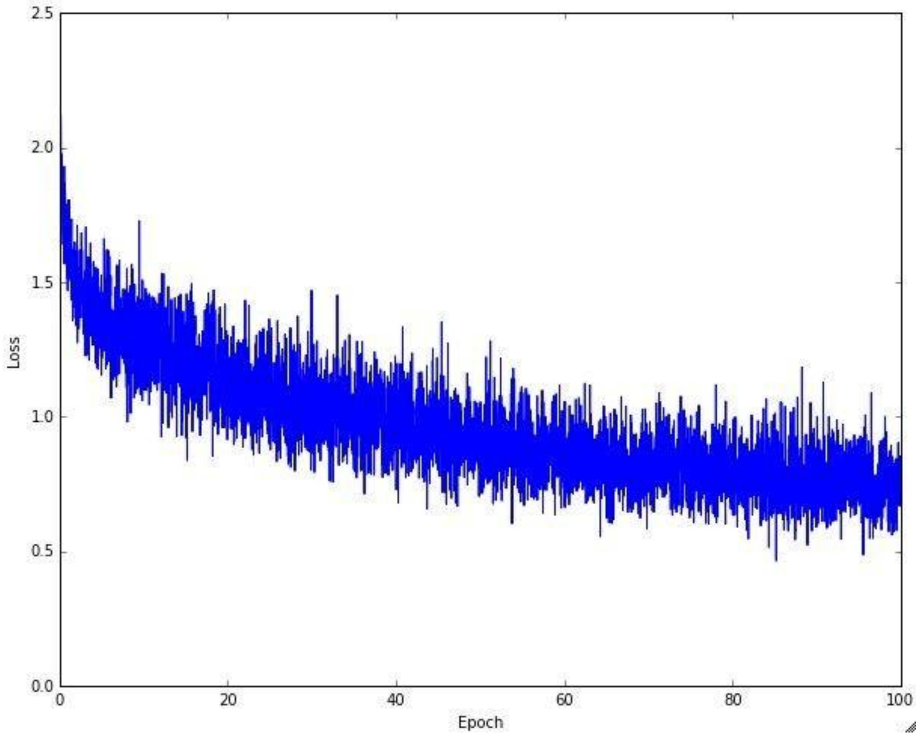
# Gradijetni Spust sa Mini-Podskupovima
# *Mini-batch Gradient Descent*

- koristimo samo deo obučavajućeg skupa da izračunamo gradijent.

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

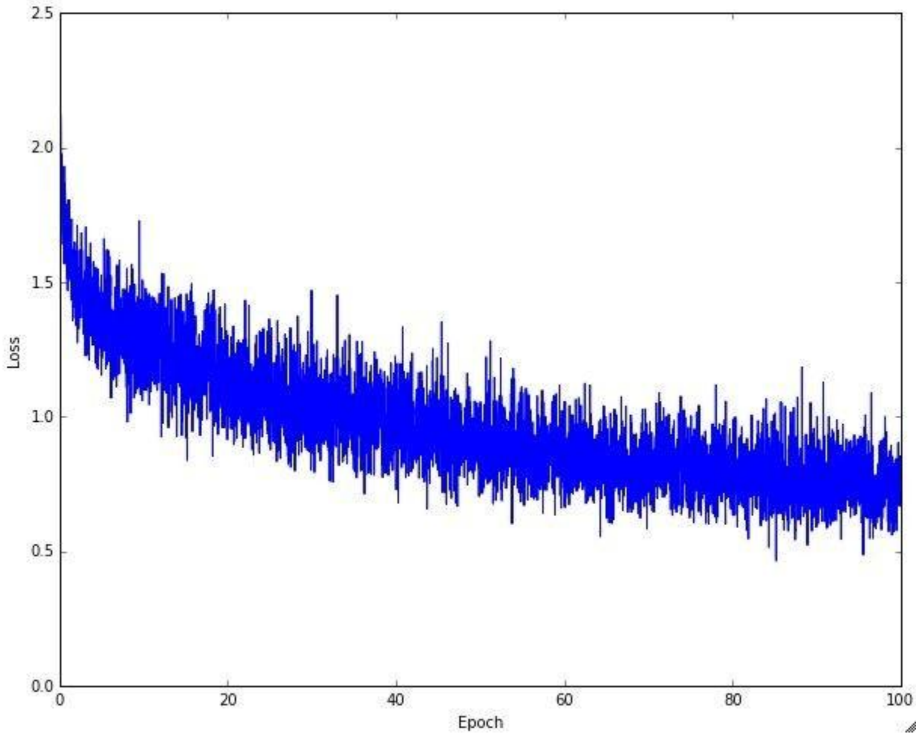
Tipične veličine podskupova su 32/64/128/256 primera

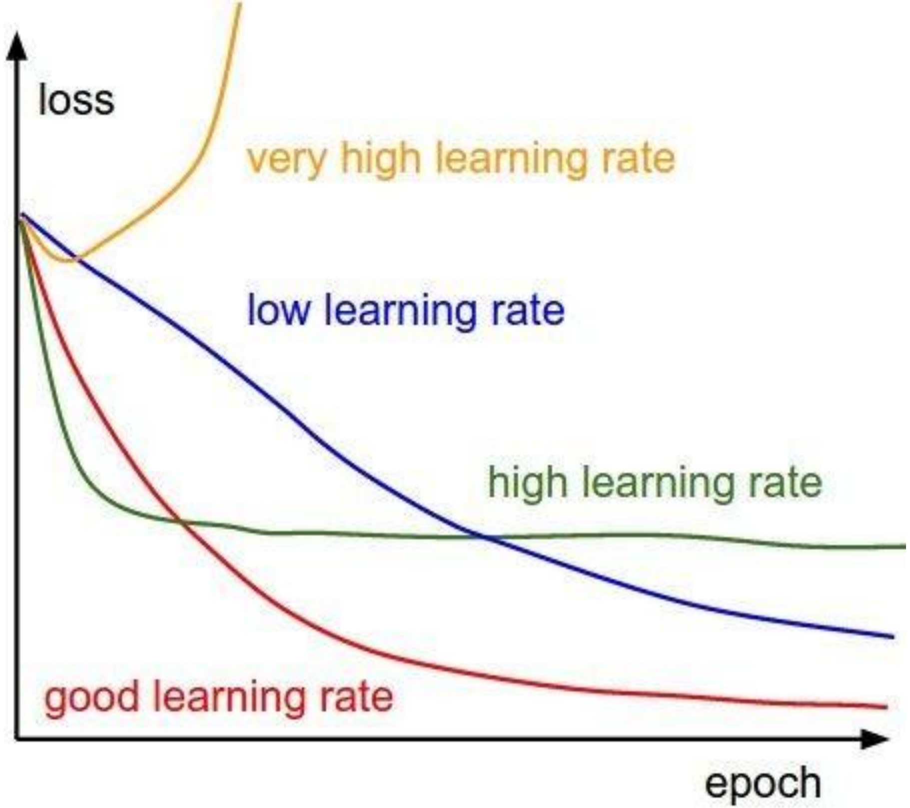



Primer optimizacije funkcije greške.

(Funkcija greške opada kroz iteracije - epohe.)

# Efekat veličine koraka (ili tempa učenja - *learning rate*)



# Gradijetni Spust sa Mini-Podskupovima

# *Mini-batch Gradient Descent*

- koristimo samo deo obučavajućeg skupa da izračunamo gradijent.

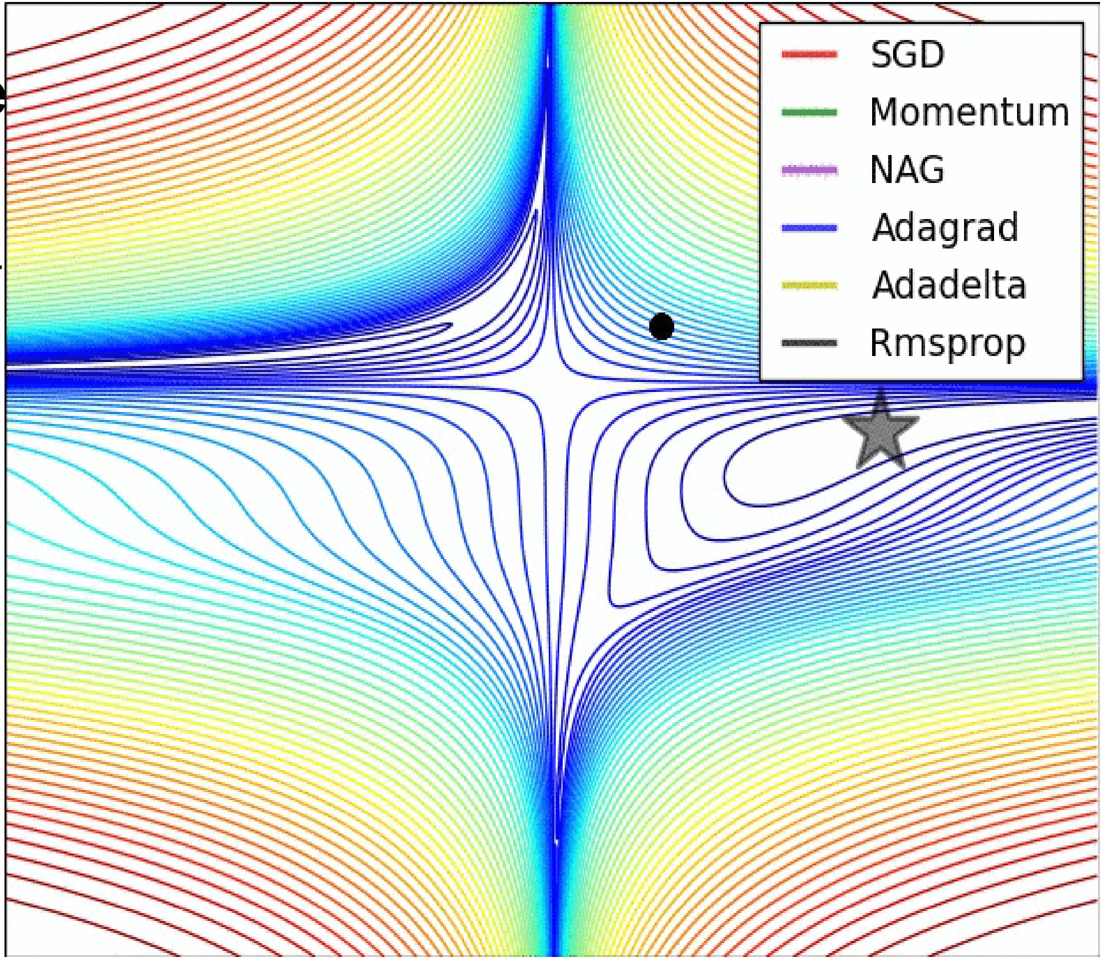
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Tipične veličine podskupova su 32/64/128/256 primera

Kasnije tokom kursa radićemo novije metode za promenu težina (momentum, Adagrad, RMSProp, Adam, …)

# Ponašanje različitih formula za promene težina

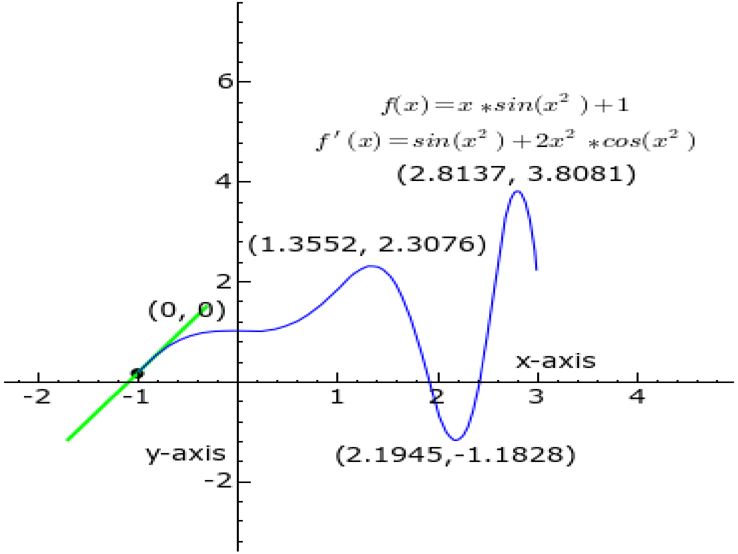


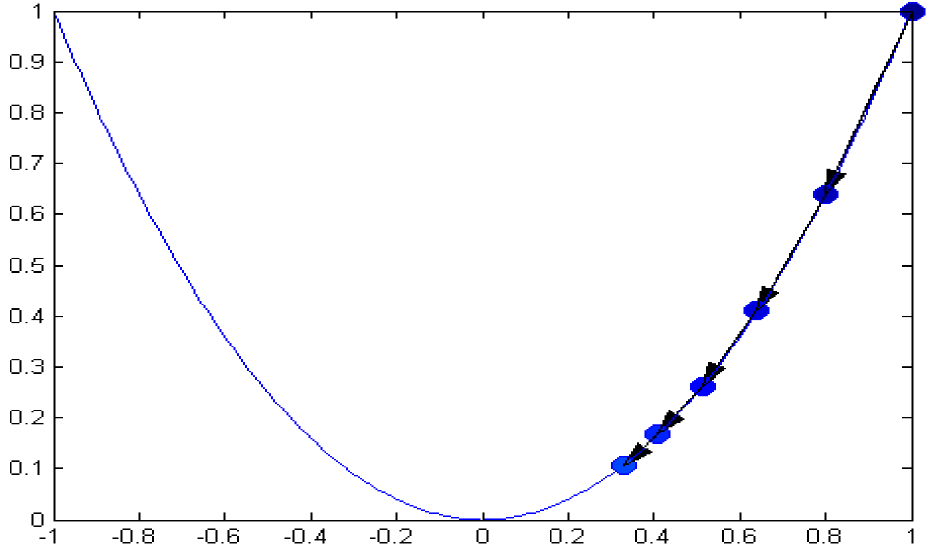

(image credits to Alec Radford)

# Gradijetni Spust ponavljanje

# Kako pronaći minimum funkcije?

- Izvod funkcije predstavlja nagib tangente na krivu funkcije
- Ideja: iterativno ćemo se pomerati ka minimumu
  - levo od minimuma: gradijent je negativan – pomeramo se u desno
  - desno od minimuma: gradijent je pozitivan – pomeramo se u levo



# Gradient descent

- **Optimizaciona tehnika**: data je (proizvoljna) funkcija $J(\theta)$. Želimo da pronađemo $\min_{\theta} J(\theta)$

| | |
|---|---|
| **Ulaz** | • $J(\theta)$ – funkcija koja se optimizuje<br>• $\theta_{\mathbf{0}}$ – početno rešenje<br>• α – *learning rate* (veličina koraka)<br>• *maxIters* – maksimalan broj iteracija |
| **Postupak** | for $t$ = 1, 2,…, maxIters:<br>for $d$ = 1,2,…,$D$<br>• $\theta_d{}^{(t+1)} = \theta_d{}^{(t)} - \boxed{\alpha \frac{\partial}{\partial \theta_d} J(\theta)}$ |
| **Izlaz** | $\theta$ – tačka u kojoj funkcija $J(\theta)$ ima minimum |

Gradijent – određuje smer pretrage

GD_visualization_quick_convergence.m

# Primena na linearnu regresiju

- Fitovanje modela: želimo da pronađemo parametre θ za koje funkcija greške ima najmanju vrednost

$$\mathrm{J}(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(\theta_1 x^{(i)} + \theta_0 - y^{(i)}\right)^2$$

- Ovo možemo uraditi primenom *gradient descent* algoritma:
    - Ponavljati do konvergencije:

$$\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha\frac{\partial}{\partial\theta_d}J(\theta) \text{ za } d \in \{0,1\}$$

$$\frac{\partial J}{\partial\theta_0} = \frac{1}{N}\sum_{i=1}^{N}(\theta_1 x^{(i)} + \theta_0 - y^{(i)}) \qquad = \frac{1}{N}\sum_{i=1}^{N}(h_\theta(x^{(i)}) - y^{(i)}) \cdot 1$$

$$\frac{\partial J}{\partial\theta_1} = \frac{1}{N}\sum_{i=1}^{N}\left(\theta_1 x^{(i)} + \theta_0 - y^{(i)}\right)x^{(i)} \qquad = \frac{1}{N}\sum_{i=1}^{N}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

# Primena na linearnu regresiju

$$X = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \dots & \dots \\ 1 & x^{(N)} \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \qquad h_\theta(x) = X\theta$$

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot 1$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

$$\longrightarrow \quad \frac{\partial J}{\partial \theta_d} = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_d^{(i)}$$

$$\theta_d^{(t+1)} = \theta_d^{(t)} - \frac{\alpha}{N} \sum_{i=1}^{N} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_d^{(i)}$$

# Gradient descent

- Ako bismo imali samo jedan primer u skupu podataka, pravilo za ažuriranje parametara je:

$$\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha(h_\theta(x) - y)x_d$$

- Ovo pravilo se naziva LMS (Least Mean Squares) update rule ili Widrow-Hoff learning rule
- Magnituda promene je proporcionalna greški $h_\theta(x) - y$
  - ako za dati primer naš prediktor daje vrednost veoma sličnu tačnoj vrednosti – nećemo mnogo menjati parametar
  - ako za dati primer prediktor ima veliku grešku – promena parametra će biti velika

# Batch GD vs. Stohastic GD

- *Batch GD*: u svakom koraku istovremeno ažuriramo parametre modela koristeći *sve* trening podatke

for $t$ = 1, 2,..., maxIters:

for $d$ = 1,2,...,$D$

- $\theta_d^{(t+1)} = \theta_d^{(t)} - \frac{\alpha}{N}\sum_{i=1}^{N}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_d{}^{(i)}$

moramo da izračunamo grešku za sve primere pa tek onda radimo promenu

- *Stohastic GD* (ili *Incremental GD*): više puta prolazimo kroz skup podataka. Kad god naiđemo na trening podatak, ažuriramo gradijent na osnovu tog (jednog) trening podatka

for $t$ = 1, 2,..., maxIters:

for $d$ = 1,2,...,$D$

for $i$ = 1,2,..., $N$

- $\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_d{}^{(i)}$

nema sume ovde, promenu radimo odmah za svaki primer

# Mini-Batch Stohastic GD

- *Batch GD*: u svakom koraku istovremeno ažuriramo parametre modela koristeći *sve* trening podatke

batch_size = 32

for $t$ = 1, 2,…, maxIters:

mini_batch = slučjano odabrana 32 primera iz skupa podataka

for $d$ = 1,2,…,$D$

- $\theta_d^{(t+1)} = \theta_d^{(t)} - \frac{\alpha}{batch\_size}\sum_{i=1}^{batch\_size}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right){x_d}^{(i)}$

ovde koristimo samo primere iz skupa mini_batch

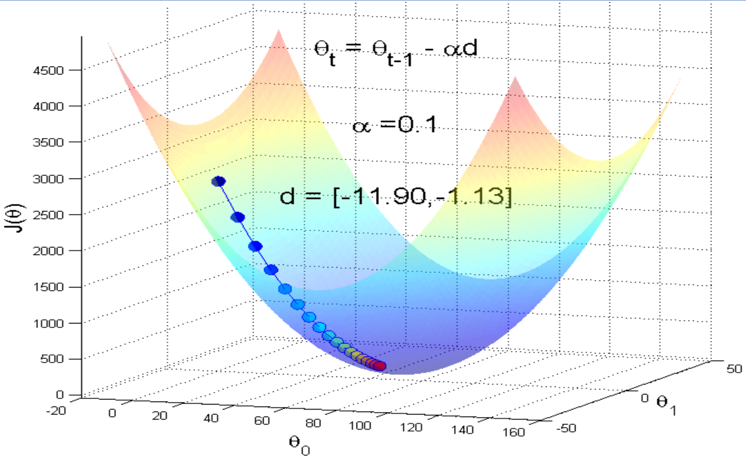
# Batch GD vs. Stohastic GD

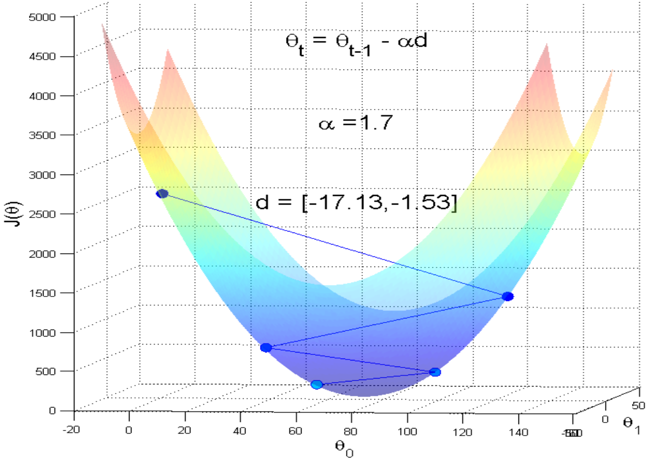
- *Batch GD* mora da skenira ceo skup podataka da bi napravio jedan korak
  - Ovo je skupa operacija ako je broj podataka *N* velik
- *Stohastic GD* može da napreduje odmah, i napreduje sa svakim uočenim trening podatkom
- Često, *stohastic GD* dovede θ „blizu“ minimuma mnogo brže od *batch GD*
- Međutim, dešava se da nikada ne „konvergira“ u minimum (parametri θ osciluju oko minimuma)
  - U praksi, tačke „blizu“ minimuma su dovoljno dobre
- Iz ovih razloga, kada je skup podataka velik, preferiramo *stohastic GD*

# Mini-Batch Stohastic GD

- Predstavlja ravnotežu koja ispravlja mane prethodna dva algoritma
- Najčešće se koristi u praksi

# Odabir α



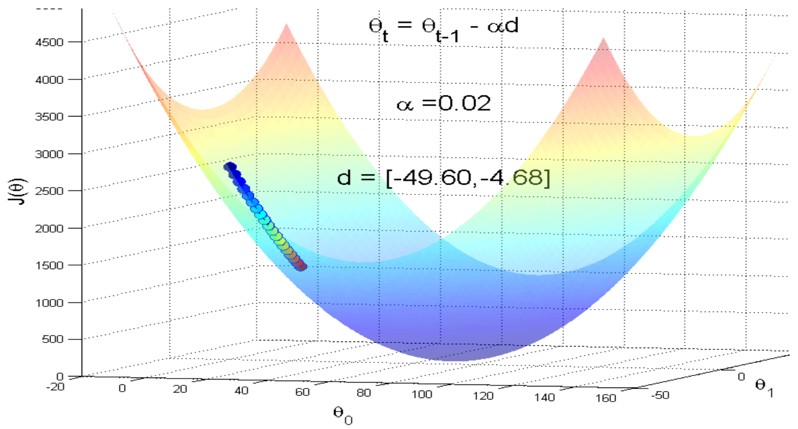

$$\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha \frac{\partial}{\partial \theta_d} J(\theta)$$

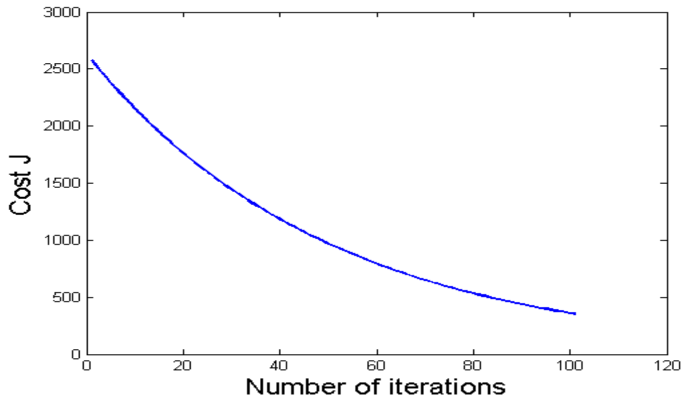
- *Gradient descent* može da konvergira u minimum za fiksiranu vrednost α: kako se približavamo minimumu koraci su sve manji jer je gradijent (nagib) sve manji

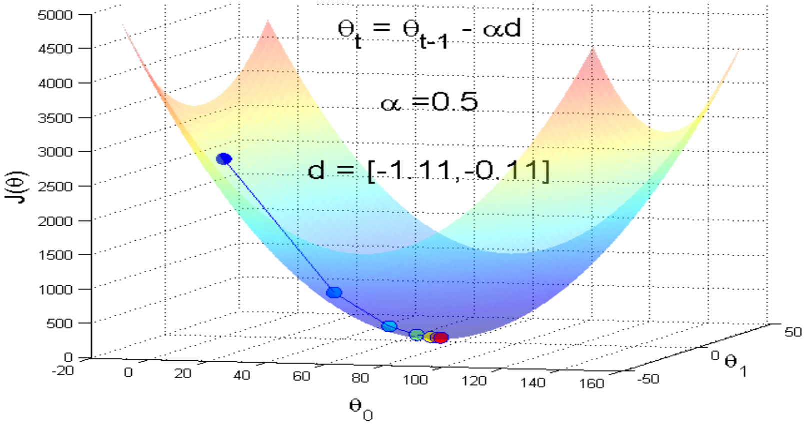
- Ako se nalazimo levo od minimuma vrednost gradijenta je negativna: θ će da raste
- Ako se nalazimo desno od minimuma vrednost gradijenta je pozitivna: θ se smanjuje

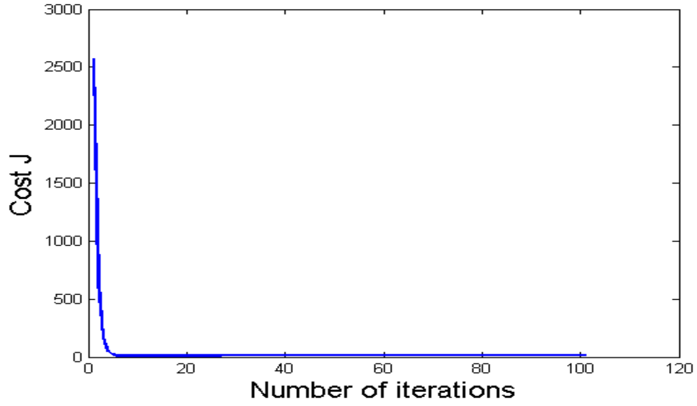
# Odabir α

- Za male vrednosti α *gradient descent* je spor



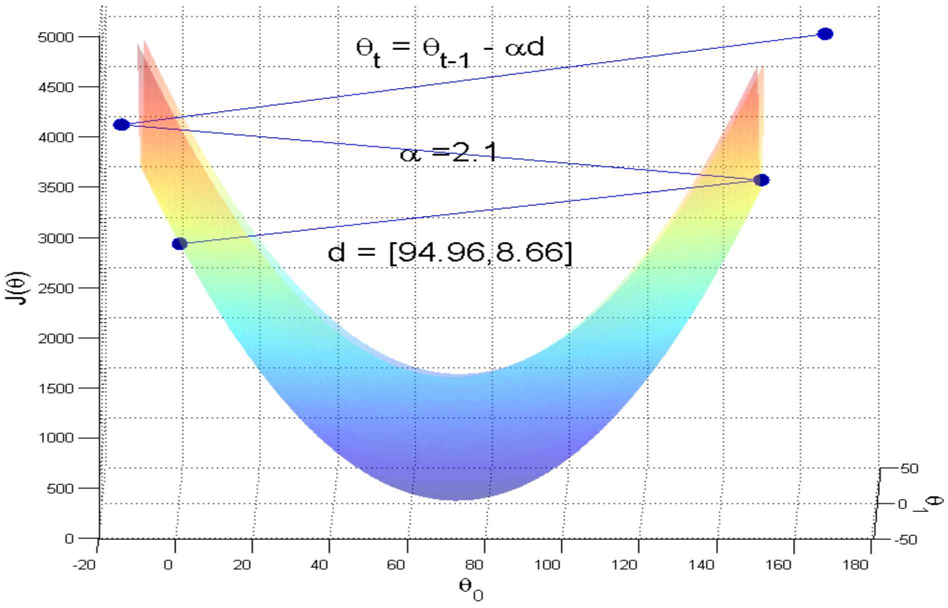



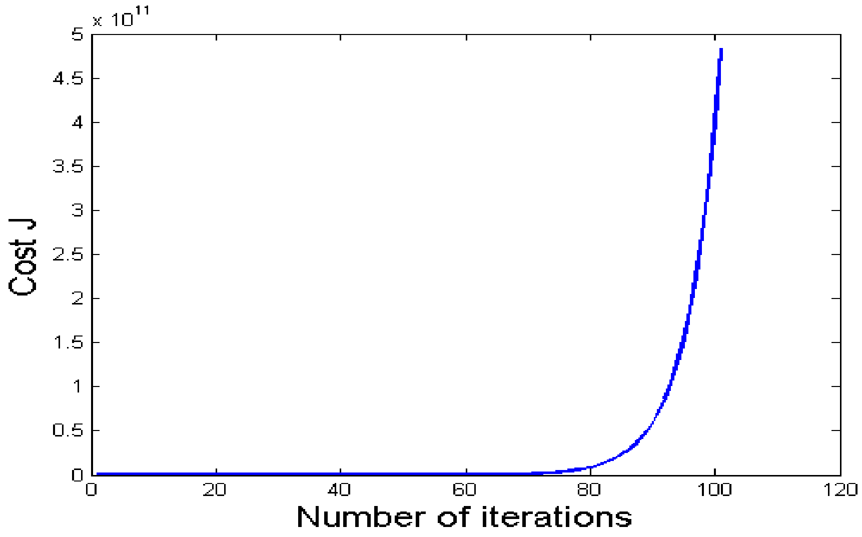



# Odabir α

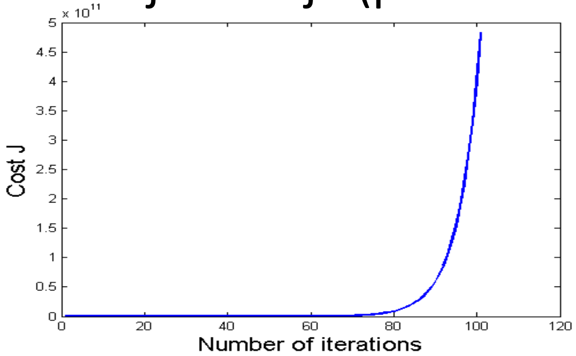
- Za preveliko α *gradient descent* ne konvergira, a može čak i da divergira



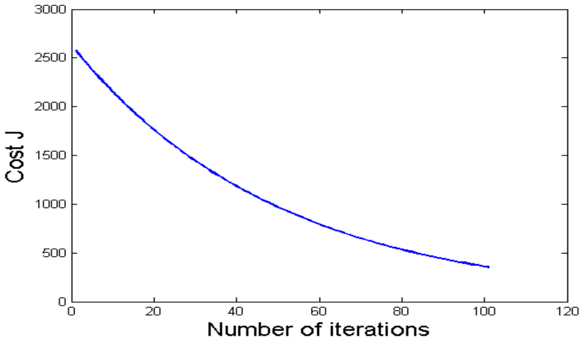



# Odabir α - zaključak

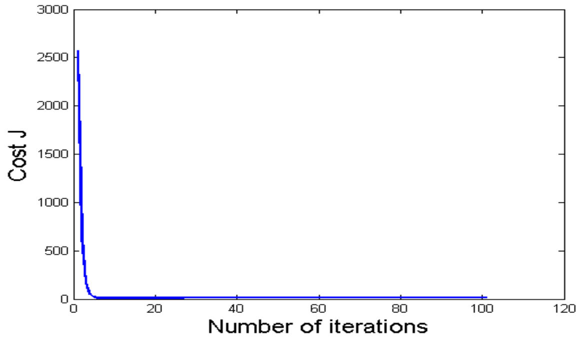
- Da bismo odredili α, najbolje je posmatrati grafik funkcije greške $J(\theta)$ u odnosu na broj iteracija (probati α = 0.001, α = 0.01, α = 0.1, α = 1,...):




Došlo je do divergencije. Smanjiti α




Algoritam je spor. Nije konvergirao u zadatom broju iteracija. Povećati α




Dobra vrednost α. Algoritam je brzo konvergirao.

- Pored fiksne vrednosti α (koje radi kada je funkcija "*strongly convex*"), čest izbor je i "*stepsize schedule*" – smanjivanje koraka α sa brojem iteracija:
  - $\eta_t = \alpha / t$ ili $\eta_t = \alpha / \sqrt{t}$