

Vještačka inteligencija

Predavanje 7: Regularizacija i optimizacija

*Pravo obrazovanje znači podsticanje sposobnosti
da se bude zainteresovan za nešto.*

~Sumio Iijima

Odgovorna nastavnica: Vanr. prof. dr Amila Akagić

Univerzitet u Sarajevu



Uvodne informacije

- This work is licensed under a Creative Commons 'Attribution-NonCommercial-ShareAlike 4.0 International' license. EN: <https://creativecommons.org/licenses/by-nc-sa/4.0/>



- Ovaj rad je licenciran pod međunarodnom licencom 'Imenovanje-Nekomercijalno-Dijeli pod istim uvjetima 4.0' od strane Creative Commons. HR: <https://creativecommons.org/licenses/by-nc-sa/4.0/deed.hr>

Aktuelnosti / Novosti

- ❑ Naučnici u Kini razvili alat za automatsko cenzurisanje tekstualnih informacija, koji navodno ima tačnost od 91%.
- ❑ Ne mora se trenirati i ima bolju tačnost nego ranije metode.
- ❑ Koristi se za filtriranje sadržaja na online medijima.
- ❑ Kina ima 900M korisnika interneta! Kineski jezik je jedan od najsloženijih jezika na svijetu, sa preko 10,000 karaktera!
- ❑ Teme koje se cenzurisu uključuju: od pornografije do kultova, zloupotrebe droga, upotrebe vatrenog oružja, terorizma i napada na Komunističku partiju i njenih najviših vođa.

<https://www.scmp.com/news/china/science/article/3129414/chinese-researchers-say-theyve-developed-ai-text-censor-91-cent>



AI pioneer Geoff Hinton: “Deep learning is going to be able to do everything”

Thirty years ago, Hinton's belief in neural networks was contrarian. Now it's hard to find anyone who disagrees, he says.

by **Karen Hao**

November 3, 2020

In the first two years, the best teams had failed to reach even 75% accuracy. But in the third, a band of three researchers—a professor and his students—suddenly blew past this ceiling. They won the competition by a staggering 10.8 percentage points. That professor was Geoffrey Hinton, and the technique they used was called deep learning.

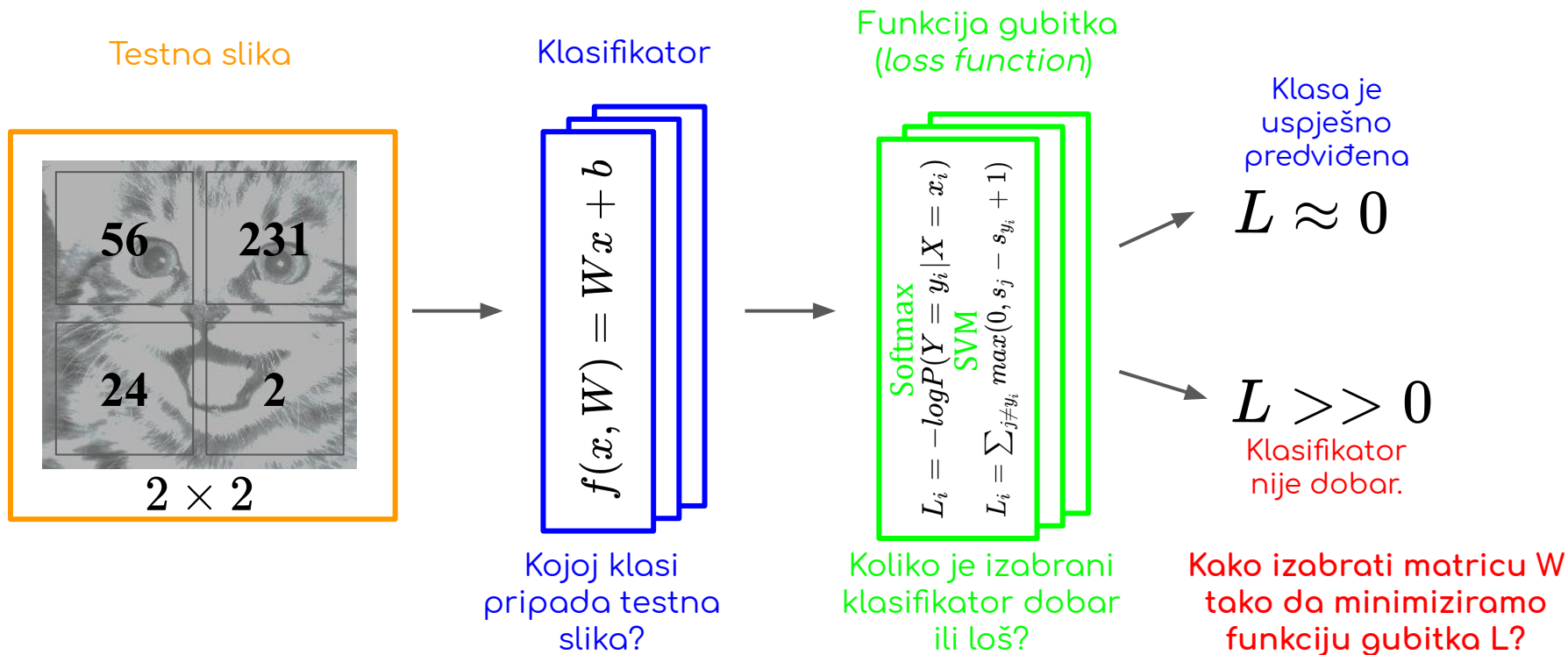
And if we have those breakthroughs, will we be able to approximate all human intelligence through deep learning?

Yes. Particularly breakthroughs to do with how you get big vectors of neural activity to implement things like reason. But we also need a massive increase in scale. The human brain has about 100 trillion parameters, or synapses. What we now call a really big model, like GPT-3, has 175 billion. It's a thousand times smaller than the brain. GPT-3 can now generate pretty plausible-looking text, and it's still tiny compared to the brain.

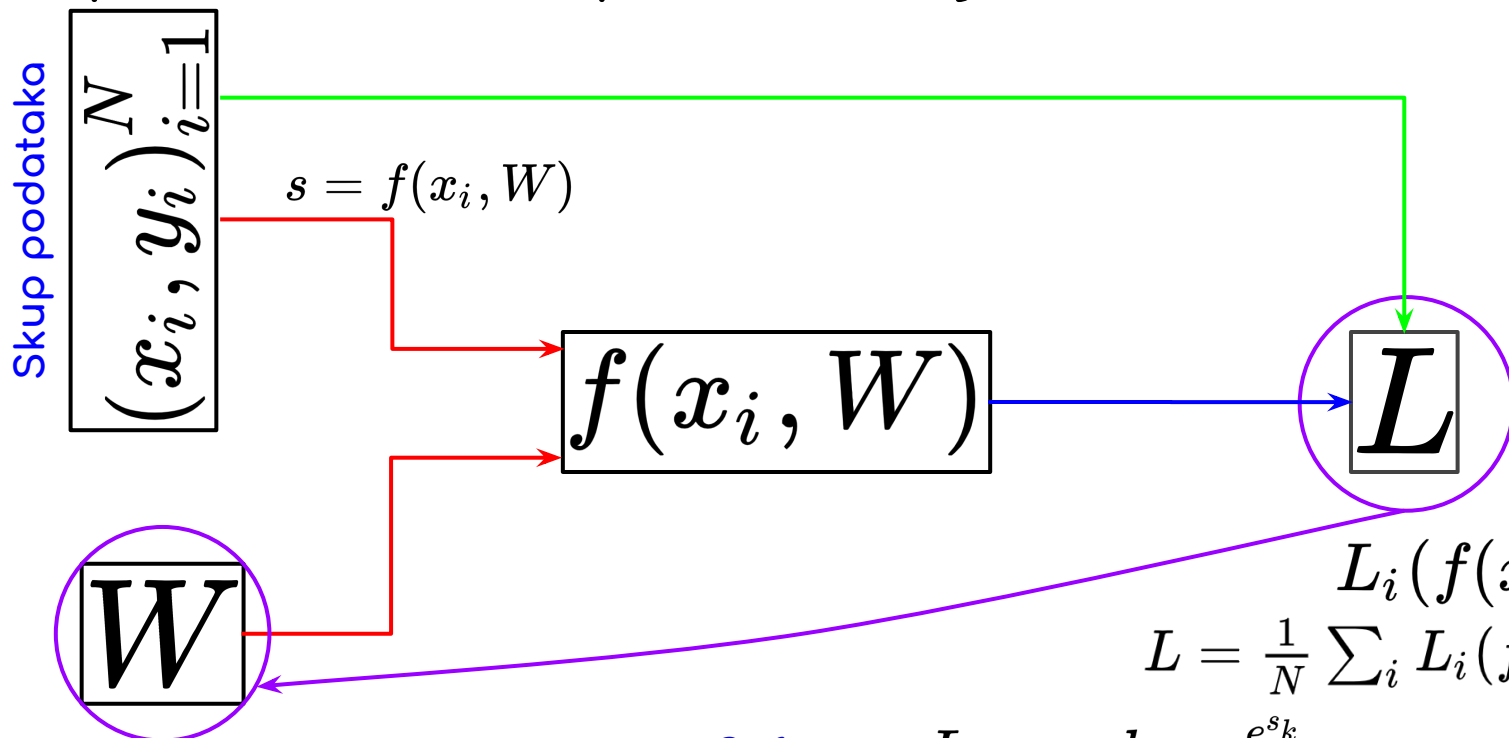


<https://www.technologyreview.com/2020/11/03/1011616/ai-godfather-geoffrey-hinton-deep-learning-will-do-everything/>

Na prethodnom predavanju...



Na prethodnom predavanju...



Da li možemo L koristiti da modifcujemo parametre u matrici W ?

$$L_i(f(x_i, W), y_i)$$
$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Softmax $L_i = -\log \frac{e^{s_k}}{\sum_j e^{s_j}}$

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

Na prethodnom predavanju...

Algebarski pogled

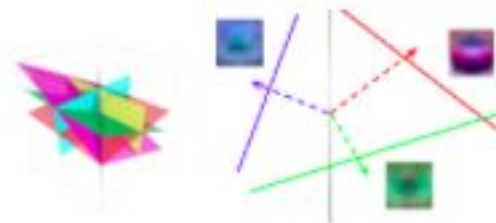


$$f(x, W) = Wx + b$$

Vizuelni pogled



Geometrijski pogled

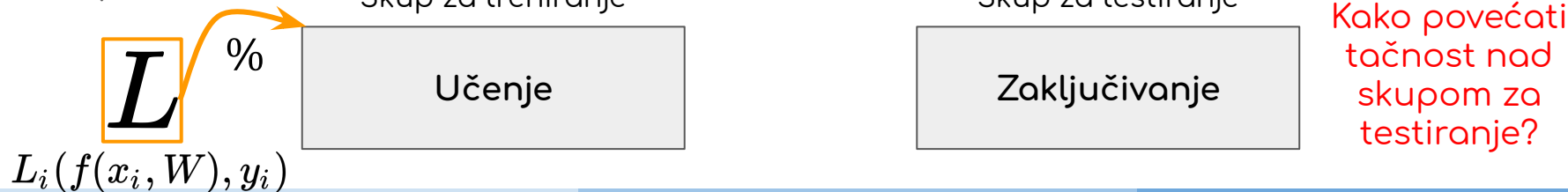


Danas ćemo raditi...

Regularizaciju i optimizaciju

Regularizacija

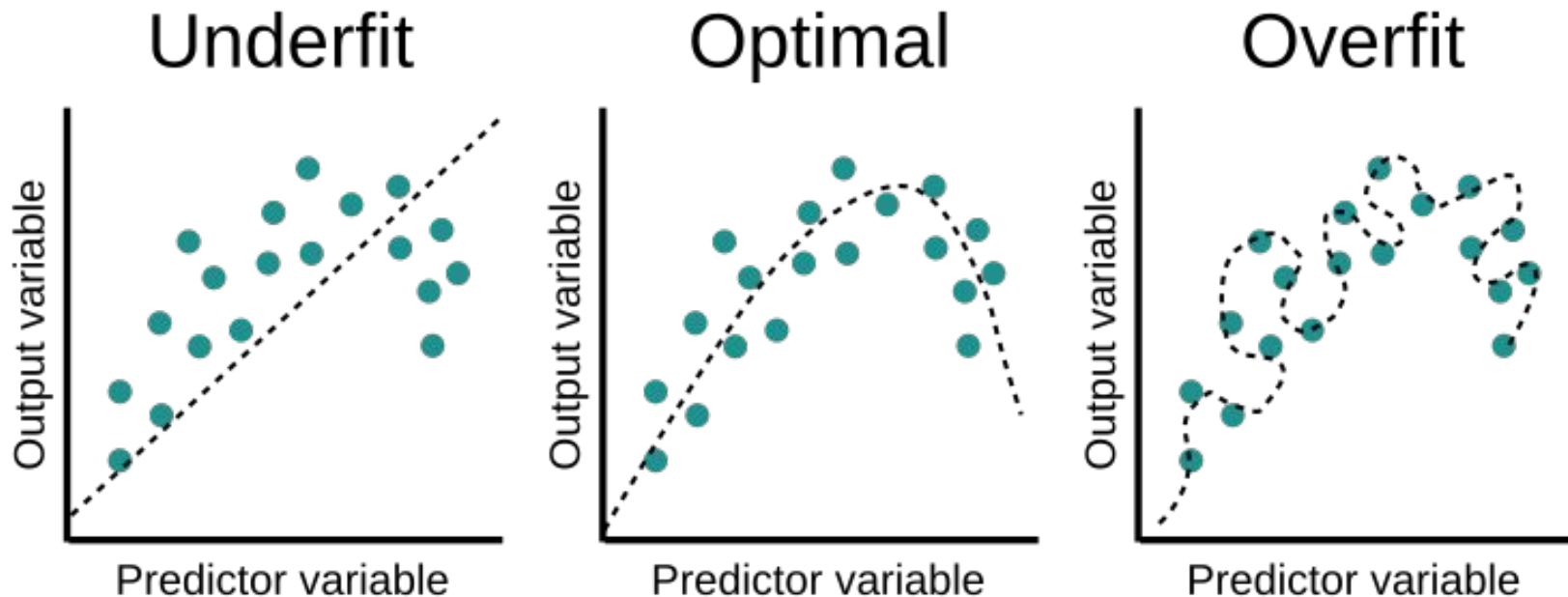
- ❑ Funkcije gubitka (*loss functions*) utiču na povećanje tačnosti nad podacima za treniranje. Međutim, nama je bitnija tačnost nad testnim podacima.
- ❑ Centralni problem u mašinskom učenju je kako napraviti algoritam koji će imati dobre rezultate ne samo na podacima za treniranje, već i na novim ulazima (testiranje), tj. na podacima koje model nikad do sada nije “vidio”.
- ❑ Mnoge strategije koje se koriste u mašinskom učenju izričito su dizajnirane da smanje grešku nad testnim podacima, i to najčešće kao posljedicu imaju *povećanje greške u procesu treniranja*.
- ❑ Ove strategije se nazivaju jednim imenom - **regularizacija**.
- ❑ Razvoj novih strategija za regularizaciju je aktivno polje istraživanja.
- ❑ **Generalizacija** - sposobnost dobrog zaključivanja na prethodno neviđenim podacima.



Regularizacija

- ❑ **Regularizacija:** bilo koja modifikacija nad nekim algoritmom učenja s ciljem smanjivanja njegove generalizacijske greške, ali ne i pogreške u treningu (idealno).
- ❑ Neke strategije regularizacije:
 - ❑ Postavljaju ograničenja na model učenja, npr. postavljanje ograničenja na vrijednostima parametara.
 - ❑ Dodaju novi član u funkciji gubitka (objektivnoj funkciji) koji utiče na promjenu parametarskih vrijednosti na blaži način nego prethodna modifikacija (*soft constraint*).
- ❑ Bez obzira na odabranu strategiju, **krajnji cilj je postizanje boljih performansi na testnom skupu podataka.**
- ❑ U nekim slučajevima, moguće je kodirati neka prethodno stečena znanja kroz postavljanje ograničenja i kazni.
- ❑ Međutim, ova ograničenja i kazne osmišljene su tako da izraze **opću sklonost jednostavnijoj klasi modela** u cilju postizanja generalizacije (uopćenosti) modela.
- ❑ Ostali oblici regularizacije, poznati kao **ansambl (*ensemble*) metode**, kombinuju više hipoteza koje bolje objašnjavaju podatke iz skupa za treniranje.

Problem *overfitting*-a



- ❑ Underfit model rezultira velikim greškama u predviđanju kako za trening tako i za test podatke. Relacija između podataka nije dovoljno dobro naučena.
- ❑ Overfit model daje vrlo nisku pogrešku predviđanja nad podacima za treniranje, ali vrlo visoku pogrešku predviđanja nad podacima za testiranje. Model je memorizirao podatke i nema mogućnost generaliziranja.

Regularizacija

$$L = \underbrace{\frac{1}{N} \sum_i L_i(f(x_i, W), y_i)}_{\text{Gubitak nad podacima: Predviđanja modela trebaju se podudarati s podacima za treniranje.}} + \underbrace{\lambda R(W)}_{\text{Regularizacija: Spriječiti model da *previše dobro* radi nad podacima za treniranje. Ne zavisi od podataka!}}$$

hiperparametar

Jednostavni primjeri

L2 regularizacija

$$R(W) = \sum_{k,l} W_{k,l}^2$$

L1 regularizacija

$$R(W) = \sum_{k,l} |W_{k,l}|$$

Složeniji primjeri

Dropout

Batch normalizacija

Cout, Mixup, Stochastic depth, itd.

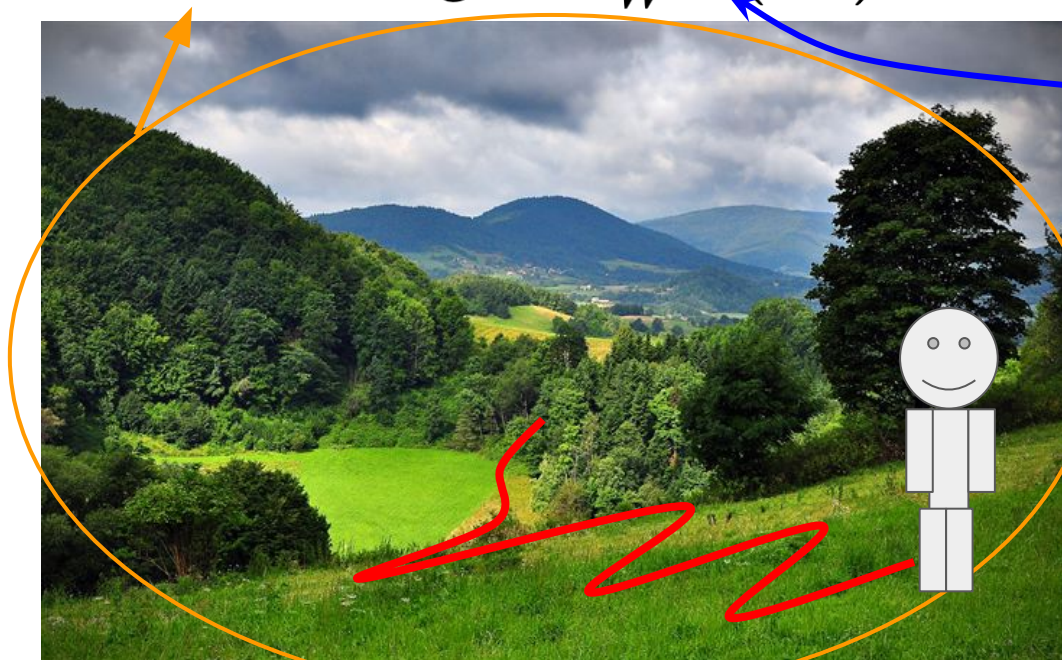
Optimizacija

- ❑ Optimizacija se odnosi na zadatak minimiziranja funkcije $f(x)$ ili maksimiziranja negativa funkcije na način da se mijenja parametar x .
- ❑ Optimizacija: pronaći parametre matrice W na način da se minimizira funkcija gubitka L .

$$W^* = \arg \min_W L(W)$$

Zamisliti 3D prostor u kojem su vrijednost matrice W predstavljene koordinatama (x,y) , dok je visina predstavljena funkcijom gubitka L (pokušavamo doći do najniže tačke).

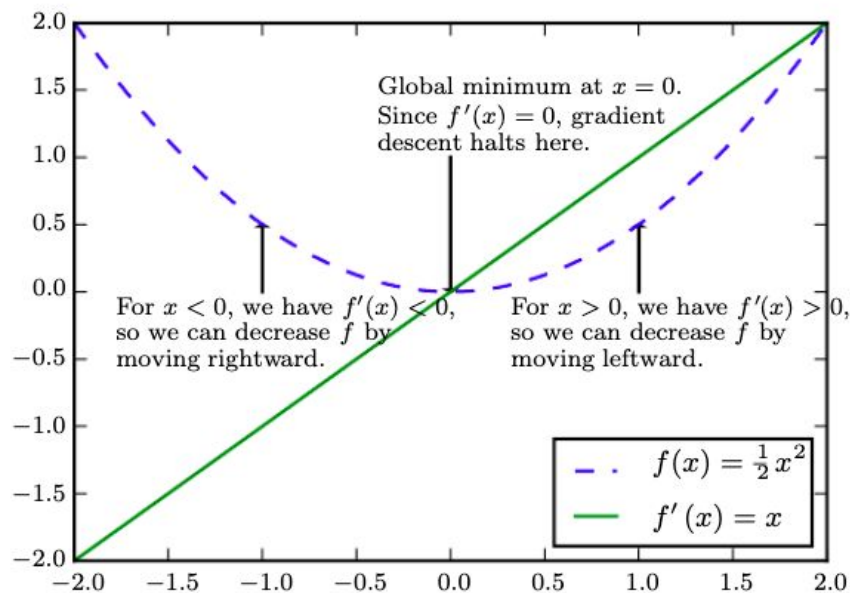
Vrijednost koja minimizira ili maksimizira funkciju često označavamo $*$.



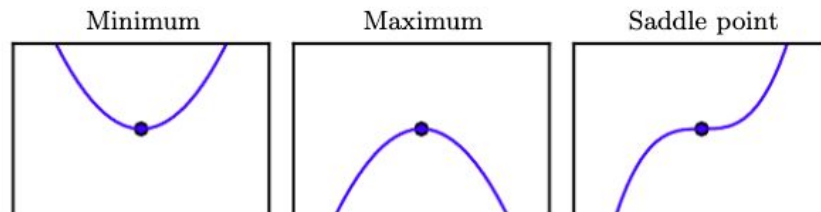
Cilj: pronaći najnižu tačku u prostoru.

Optimizacija

- ❑ Izvod funkcije $f(x)$ se koristi za minimiziranje funkcije: kako promijeniti x kako bismo napravili malo poboljšanje u funkciji $f(x)$?
- ❑ Najčešće korištena metoda je metoda gradijentnog spusta.



Vrste kritičnih tačaka:



Optimizacija

- ❑ Koriste se metode koje iterativno poboljšavaju rješenja s ciljem pronalaženja najniže tačke u prostoru.
- ❑ Postoji nekoliko algoritama:
 - ❑ Nasumično pretraživanje (*Random search*)
 - ❑ Nasumično izaberi parametre matrice W i pamti one parametre koji daju dobre rezultate.
 - ❑ Nasumično lokalno pretraživanje (*Random local search*)
 - ❑ Metoda gradijentnog spusta (*Gradient Descent*)

Ideja #1: Nasumično pretraživanje (Random Search)

Skup podataka: CIFAR10

```
# 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 range(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  
# ... (truncated: continues for 1000 lines)
```


Ideja #1: Nasumično pretraživanje (Random Search)

Skup podataka: CIFAR10

```
# 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
```

Tačnost je oko 15.5% nad testnim podacima!

Problem: Na ovom dataset-u tačnost najboljih algoritama je ~95%

Ideja #2: Nasumično lokalno pretraživanje

- ❑ Pretraži lokalni prostor i pronadi nagib (strminu) i kreći se u tom pravcu ako nagib vodi nizbrdo. Ponavljaj ovaj korak dok ne dođeš do najniže tačke u prostoru.
- ❑ Kako definisati nagib neke 1D funkcije (IM1/IM2)?
- ❑ Kako definisati nagib neke M-D funkcije?

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

U M-D prostoru (višedimenzionalnom prostoru) definišemo **gradijent** je vektor (parcijalni izvod) u odnosu na svaku dimenziju.

Nagib se definiše kao množenje pravca sa gradijentom.

Smjer najstrmijeg spuštanja je negativni gradijent.



Ideja #2: Nasumično lokalno pretraživanje

- ❑ Postoji nekoliko načina određivanja gradijenta. **Egzaktno:**

- ❑ Numeričko određivanje gradijenta

- ❑ Jednostavan način
- ❑ Veoma sporo: $O(\# \text{dimenzija})$
Pogotovo za veće skupove podataka
- ❑ Postupak je aproksimativan

- ❑ Analitičko određivanje gradijenta

- ❑ Brzo
- ❑ Egzaktno
- ❑ Može dovesti do grešaka (zbog procesa ručnog računanja)

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

$$\frac{\partial f}{\partial x_i} = \lim_{h \rightarrow 0} \frac{f(x_i+h, x) - f(x)}{h}$$

Približno (aproksimativno):

$$\frac{df}{dx} \approx \frac{f(x+h) - f(x)}{h}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_i+h, x) - f(x)}{h}$$

U praksi: Uvijek se koristi analitički gradijent, ali implementacija se provjerava numeričkim gradijentom (*debugging tool*).

Postupak se naziva još i **provjera gradijenta (gradient check)**. Postoji nekoliko funkcija koje se mogu koristiti za provjeru gradijenta.

Koristi se samo u slučaju kada se piše novi kod za računanje gradijenta.

```
def grad_check_sparse(f, x, analytic_grad, num_checks=10, h=1e-7):  
    """  
    sample a few random elements and only return numerical  
    in this dimensions.  
    """
```

Ideja #2: Nasumično lokalno pretraživanje

Nasumično
pretraživanje

```
bestloss = float("inf") # Python assigns the highest possible float value
for num in range(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)
```

```
W = np.random.randn(10, 3073) * 0.001 # generate random starting W
bestloss = float("inf")
for i in range(1000):
    step_size = 0.0001
    Wtry = W + np.random.randn(10, 3073) * step_size
    loss = L(Xtr_cols, Ytr, Wtry)
    if loss < bestloss:
        W = Wtry
        bestloss = loss
    print 'iter %d loss is %f' % (i, bestloss)
```

Tačnost je oko 21.4% nad testnim podacima!
Problem: Na ovom dataset-u tačnost najboljih
algoritama je ~95%

Više detalja na <https://cs231n.github.io/optimization-1/>.

Ideja #3: Metoda gradijentnog spusta

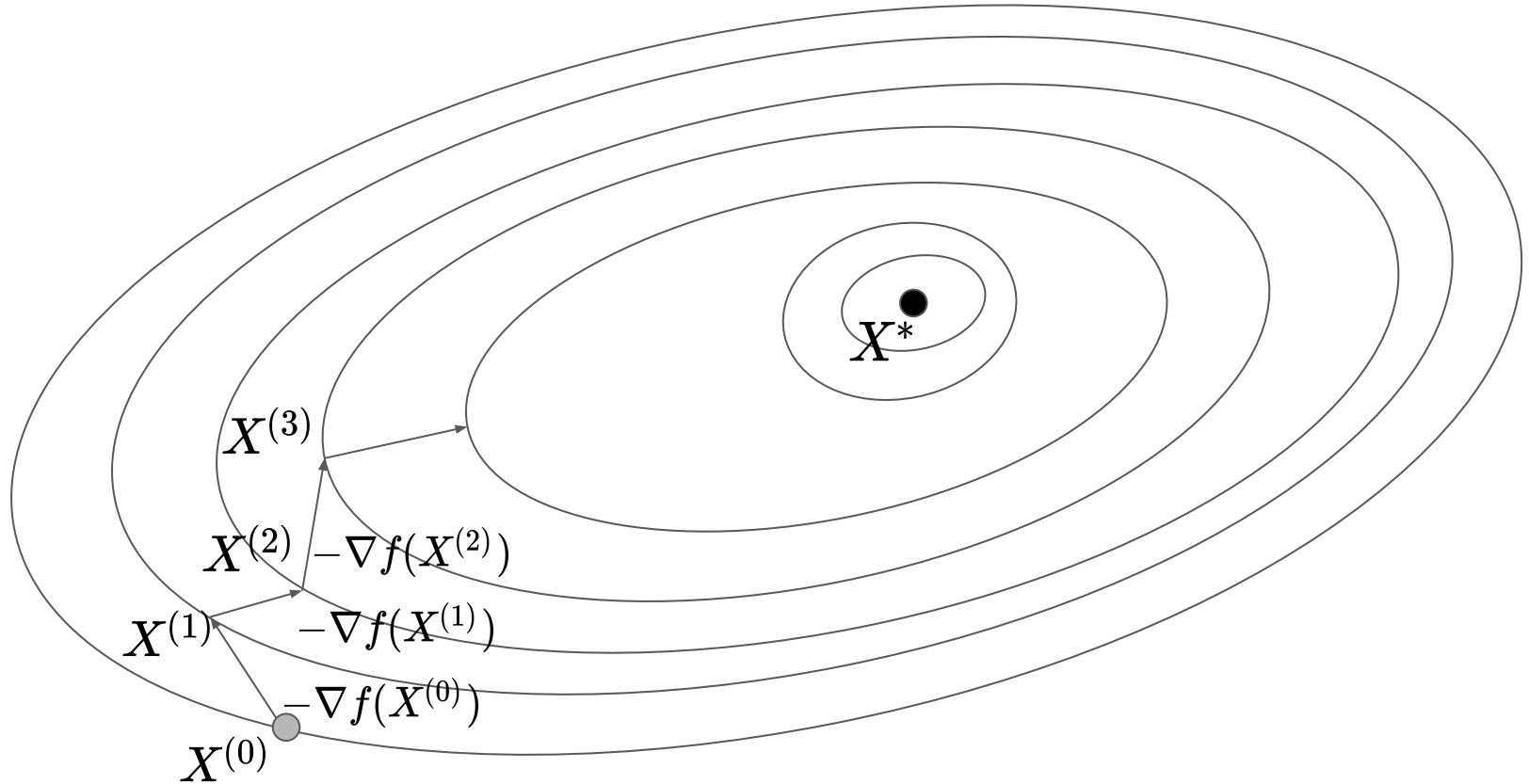
- ❑ Iterativno se kreći u smjeru negativnog gradijenta (lokalni najstrmiji spust).
 - ❑ Koji je smjer lokalnog gradijenta?
 - ❑ Kreći se u suprotnom smjeru.
 - ❑ Ponavljaj ove korake dok ne dođeš do najniže tačke.
- ❑ Lako se implementira (nekoliko linija koda).

```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
    dw = compute_gradient(loss_fn, data, w)
    w -= learning_rate * dw
```

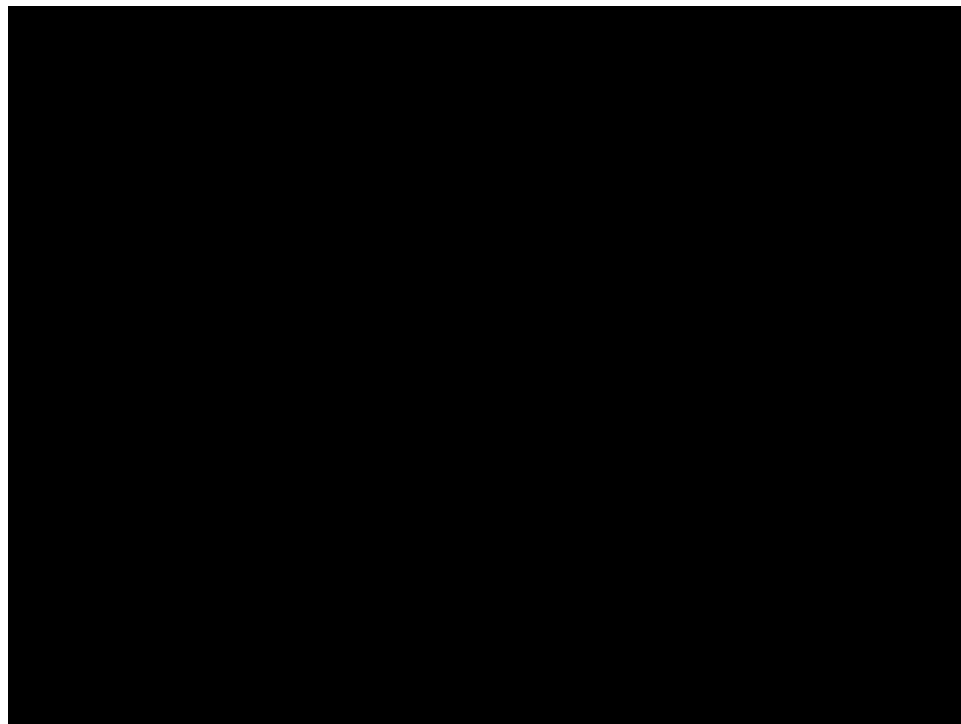
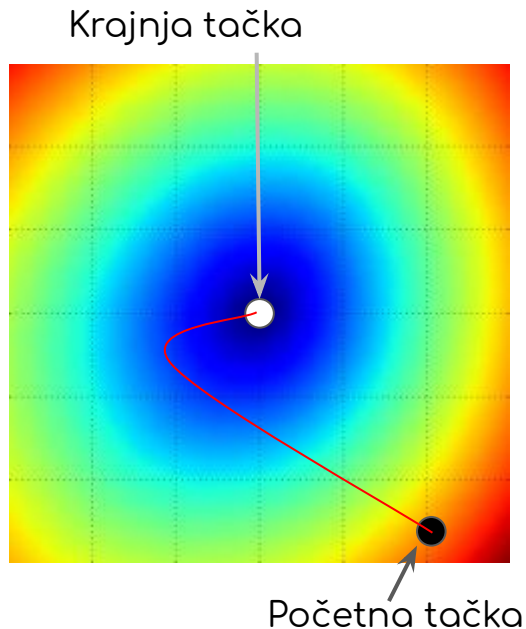
Hiperparametri:

1. Kako se inicijalizira matrica W ? Utiče na brzinu konvergencije.
2. Koji je broj koraka? Zависи od računarskih resursa.
3. “learning_rate”: koliko vjerujemo gradijentu kada se krećemo u njegovom (suprotnom) pravcu?

Ideja #3: Metoda gradijentnog spusta



Ideja #3: Metoda gradijentnog spusta



Prednost: Pravi velike korake kada je daleko od minimuma, a manje koraka kada je blizu minimuma.

Ideja #3: Metoda gradijentnog spusta

- ❑ Algoritam se zaustavlja ukoliko je ispunjen neki kriterijum zaustavljanja (unaprijed poznat).
- ❑ U literaturi se još naziva i *Košijevom metodom*. Augustin-Louis Cauchy (1789–1857)
- ❑ Osnovni koraci metode su:

Broj iteracija



Korak 0. Postaviti $k = 0$. Izabrati dopustivu tačku $X^{(0)} \in R^n$.

Korak 1. Izračunati $d_k = -\nabla f(X^{(k)})$.

Korak 2. Naći α_k kao rešenje problema minimizacije funkcije $g_k(\alpha) = f(X^{(k)} - \alpha \nabla f(X^{(k)}))$ za $\alpha \geq 0$.

Korak 3. Izračunati $X^{(k+1)} = X^{(k)} - \alpha_k d_k$. Postaviti $k = k + 1$ i ići na Korak 1.

Ideja #3: Metoda gradijentnog spusta

- ❑ Postoje tri verzije metode gradijentnog spusta:
 - ❑ Serijski ili ukupni gradijentni spust (Batch Gradient Descent / BGD)
 - ❑ Stohastički gradijentni spust (Stochastic Gradient Descent / SGD)
 - ❑ Mini-serijski gradijentni spust (Mini-batch Gradient Descent)
- ❑ Razlikuju se po količini podataka koja se uzima u jednom koraku da se izračuna gradijent funkcije gubitka.
- ❑ Pravi se kompromis između tačnosti i vremena koje je potrebno da se izračuna gradijent.

Serijski gradijentni spust (Batch Gradient Descent / BGD)

- ❑ N je broj uzoraka u skupu za treniranje, i kreće se od 1 do N (index i).
- ❑ Šta se dešava kada u skupu za treniranje imamo puno elemenata? Npr. 1M, 1B, itd. elemenata?
 - ❑ Suma se izračunava u for petlji, te je realizacija veoma “skupa” (spora).
- ❑ Model se ne može ažurirati *online* (nakon nekoliko uzoraka), nego isključivo *offline*.
- ❑ Konvergencija prema globalnom minimumu je zagarantovana za konveksne površine, a za nekonveksne površine metoda uspešno konvergira ka lokalnom minimumu.

$$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_i \nabla_W L_i(f(x_i, W), y_i) + \lambda \nabla_W R(W)$$

```
for i in range(nb_epochs):  
    params_grad = evaluate_gradient(loss_function, data, params)  
    params = params - learning_rate * params_grad
```

Stohastički gradijentni spust (SGD)

- ❑ Metoda stohastičkog gradijentnog spusta ažurira parametre za svaki element unutar skupa za treniranje. Model se praktično trenira na pojedinačnim elementima iz skupa za treniranje.
- ❑ Prednost je ubrzanje u odnosu na prethodnu verziju (brža konvergencija) i ažuriranje parametara *online*. Zato se metoda još naziva i *Online* gradijentni spust.
- ❑ Mana: zbog čestih ažuriranja, funkcija može da oscilira, tj. da bude nestabilna.
 - ❑ Rješenje: Ako se `learning_rate` postepeno mijenja (tj. nije konstantno) možemo ostvariti sličnu konvergenciju kao kod serijskog gradijentnog spusta.

```
for i in range(nb_epochs):  
    np.random.shuffle(data)  
    for example in data:  
        params_grad = evaluate_gradient(loss_function, example, params)  
        params = params - learning_rate * params_grad
```

Mini-serijski gradijentni spust

- ❑ Parametri se ažuriraju uzimajući mali skup elemenata (tačaka) iz skupa za treniranje nad kojima se izračunava gradijent, te se zatim ažuriraju parametri matrice W .
 - ❑ *Smanjuje se varijansa ažuriranja parametara što može dovesti do brže konvergencije.*
 - ❑ Moguće je koristiti neke strategije za optimizaciju matrice što će dovesti do bržeg računanja gradijenta.
- ❑ Mini-serija (mini-batch) obično sadrži 32 / 64 / 128 elemenata koji se analiziraju u jednom trenutku.


Mini-serijski gradijentni spust

Hiperparametri:

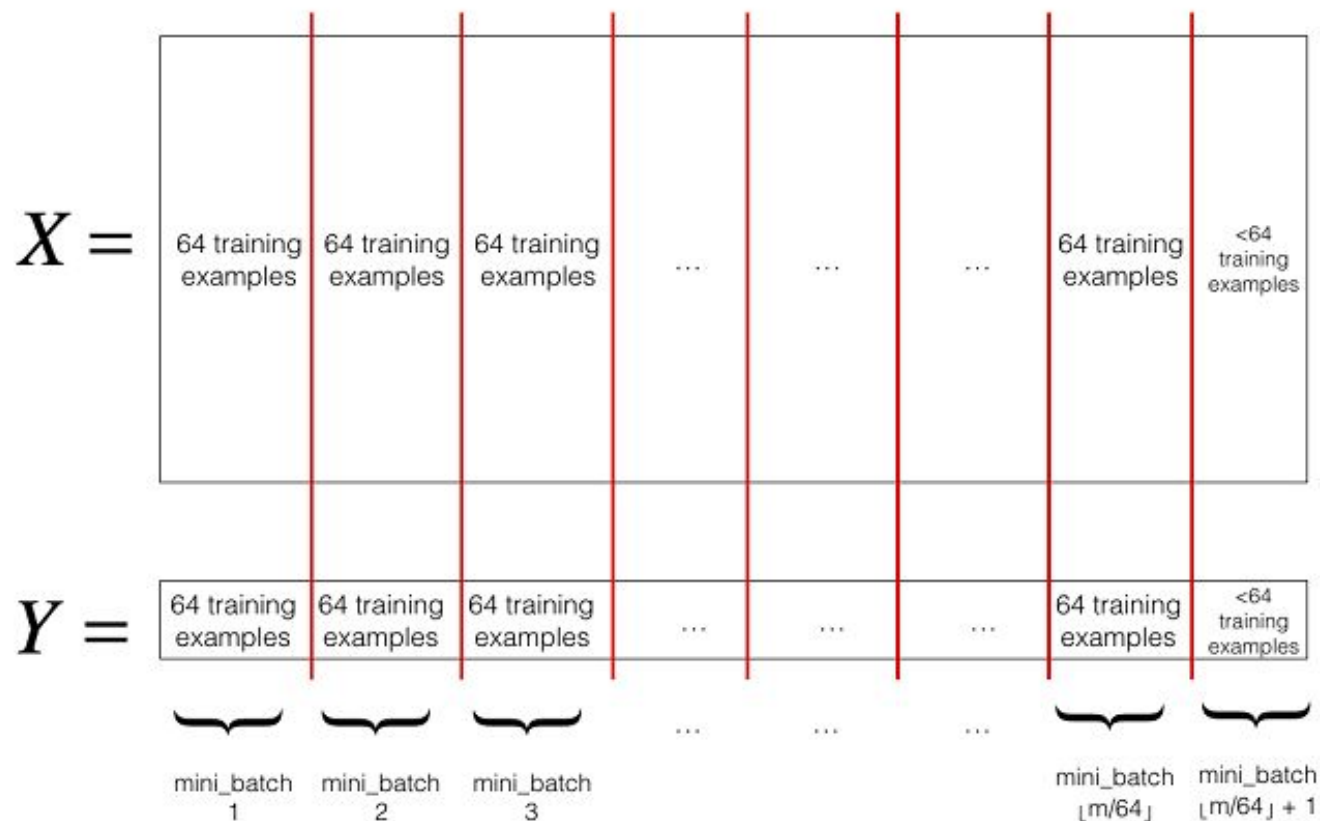
1. Kako se inicijalizira matrica W ? Utiče na brzinu konvergencije.
2. Koji je broj koraka? Zavisi od računarskih resura.
3. "learning_rate": koliko vjerujemo gradijentu kada se krećemo u njegovom (suprotnom) pravcu?
4. Broj elemenata koji se analiziraju u mini-seriji
5. Uzorkovanje podataka

batch_size = N?
batch_size = 1?
batch_size = 32 / 64 / 128?

```
for i in range(nb_epochs):  
    np.random.shuffle(data)  
    for batch in get_batches(data, batch_size=50):  
        params_grad = evaluate_gradient(loss_function, batch, params)  
        params = params - learning_rate * params_grad
```

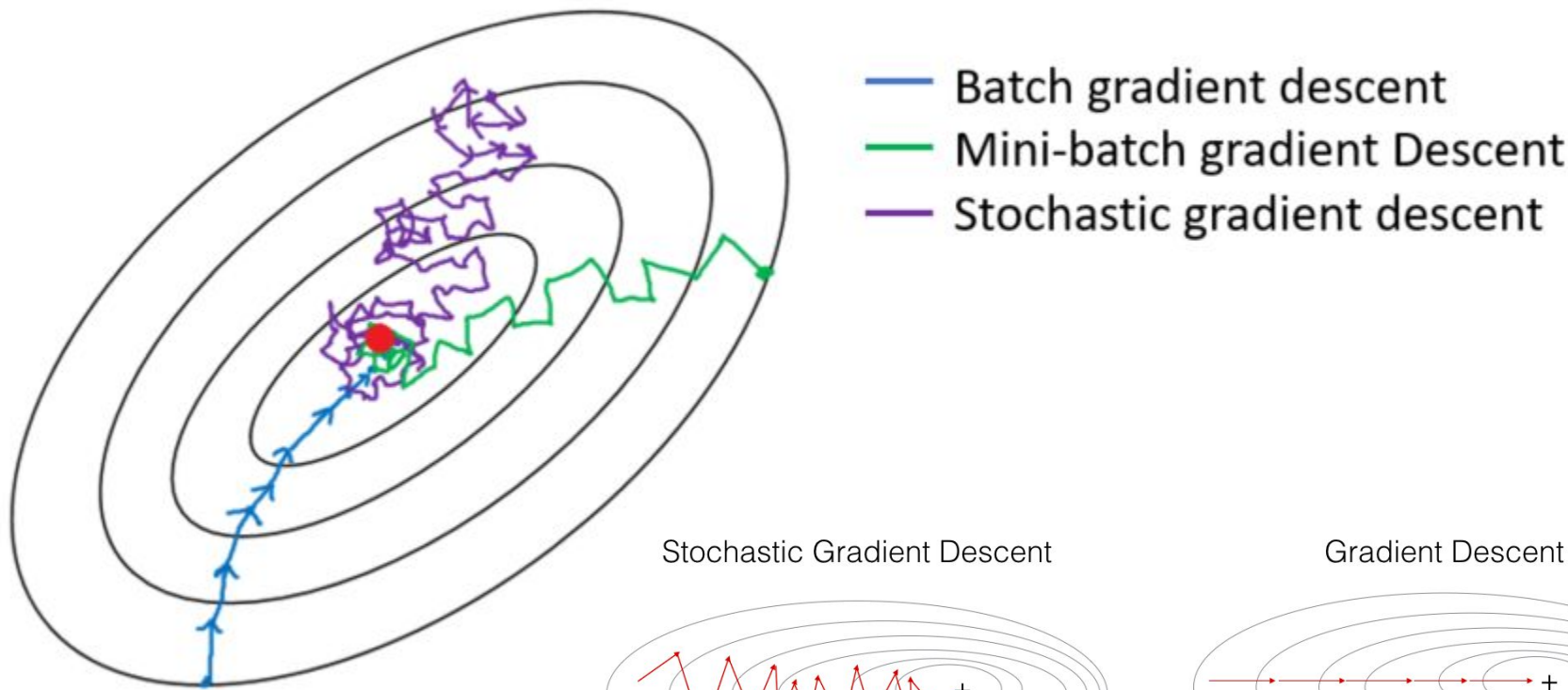


Mini-serijski gradijentni spust



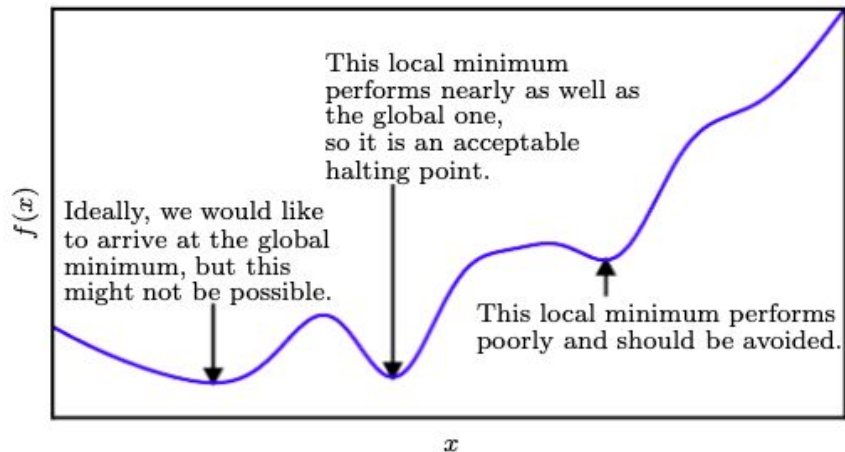
Stohastički gradijentni spust (SGD): problemi

Vrlo sporo napreduje duž jedne dimenzije, a osciluje uz strmi smjer.

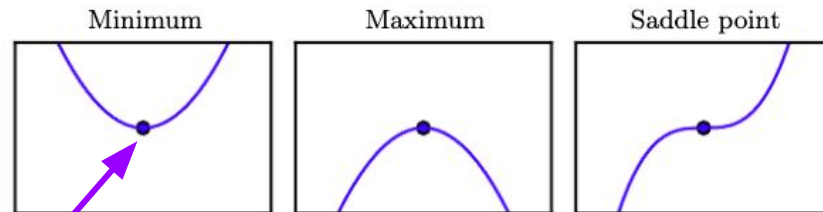


Stohastički gradijentni spust (SGD): problemi

Što ako funkcija gubitka ima lokalni minimum ili sedlastu tačku?

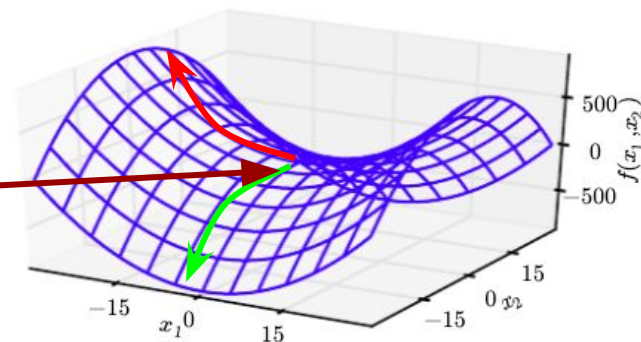


Vrste kritičnih tačaka:



Gradijent je 0!

Potencijalni problem je što ne možemo izaći iz ove tačke, tj. ne možemo naći globalni minimum.



Različite optimizacije SGD-a

- ❑ U praksi postoje neke optimizacije koje rješavaju neke probleme SGD metode:
 - ❑ SGD + Momentum
 - ❑ Nesterov Momentum
 - ❑ Adagrad
 - ❑ Adadelata
 - ❑ RMSprop
 - ❑ Adam
 - ❑ AdaMax
 - ❑ Nadam
 - ❑ AMSGrad

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):  
    dw = compute_gradient(w)  
    w -= learning_rate * dw
```

Vektor brzine

SGD + Momentum

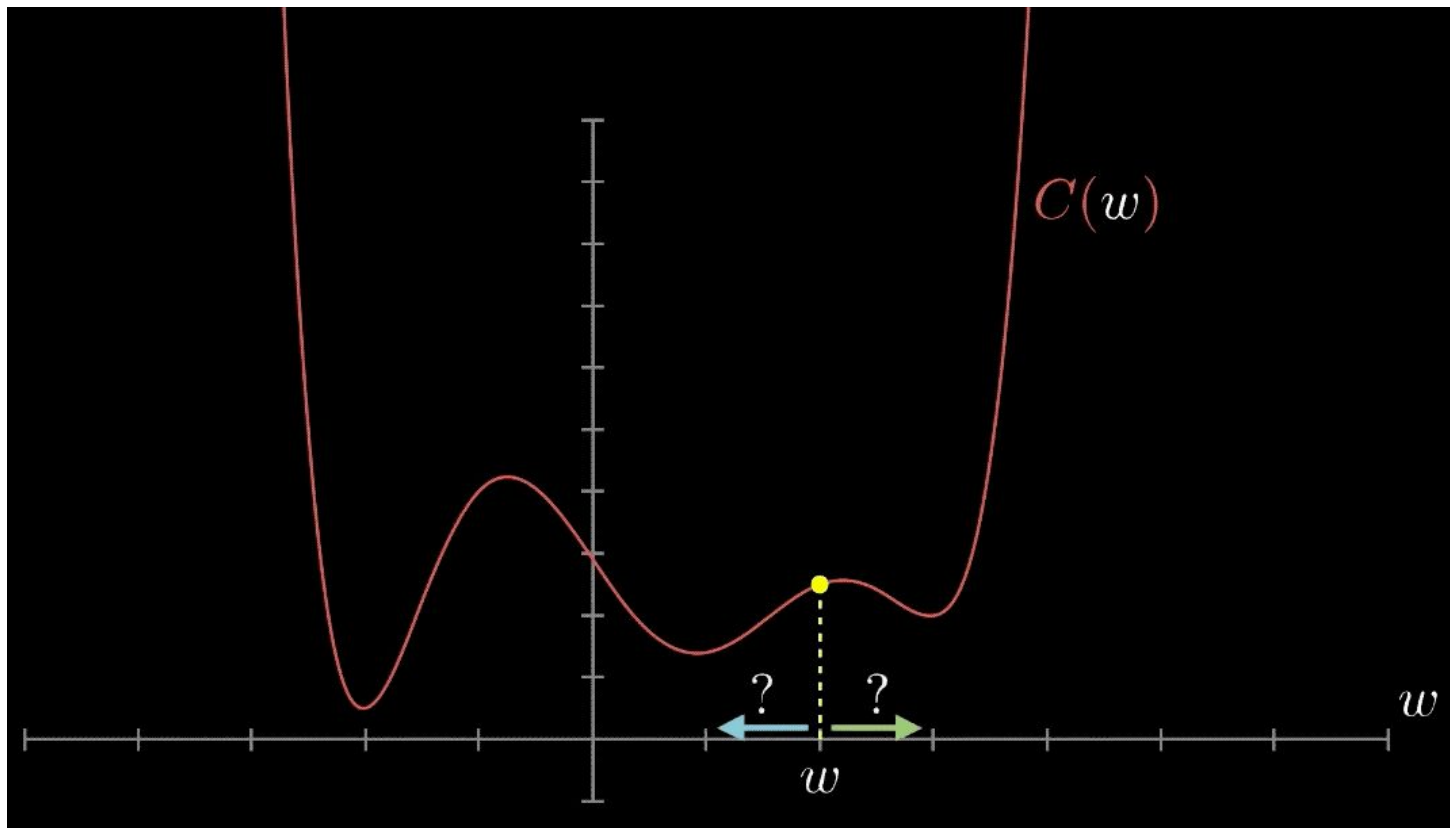
Trenje

$$v_{t+1} = \rho v_t - \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0  
for t in range(num_steps):  
    dw = compute_gradient(w)  
    v = rho * v + dw  
    w -= learning_rate * v
```

- ❑ Momentum ubrzava SGD metod u najkritičnijim tačkama (obično u uvalama oko lokalnog optimuma) i usmjerava u pravom smjeru. Na ovaj način oscilacije koje su ranije bile problem su smanjene.

SGD + Momentum

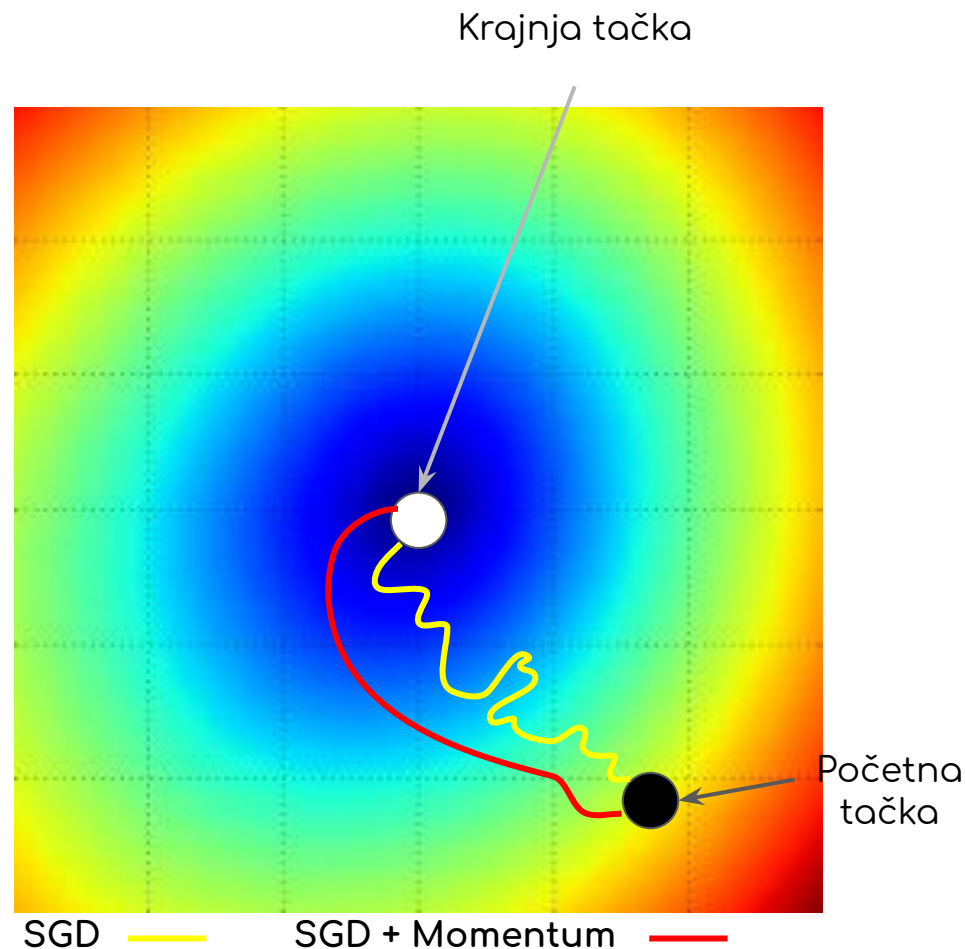
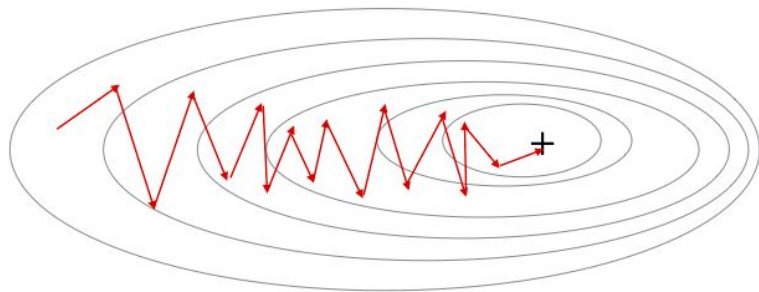


<https://www.youtube.com/watch?v=IHZwWFHWa-w>

SGD + Momentum

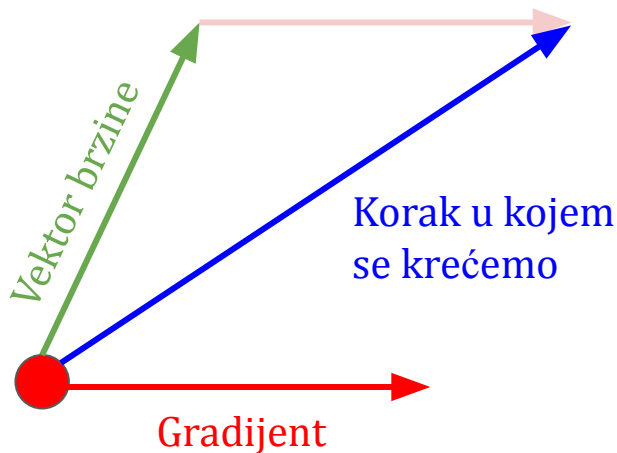


Stochastic Gradient Descent



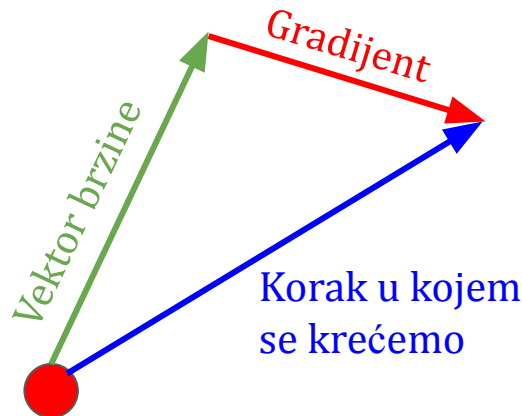
SGD Momentum, Nesterov Momentum

SGD + Momentum



Gradijent trenutne tačke i ubrzanje se kombinuju kako bi se definisao korak koji će ažurirati koeficijente.

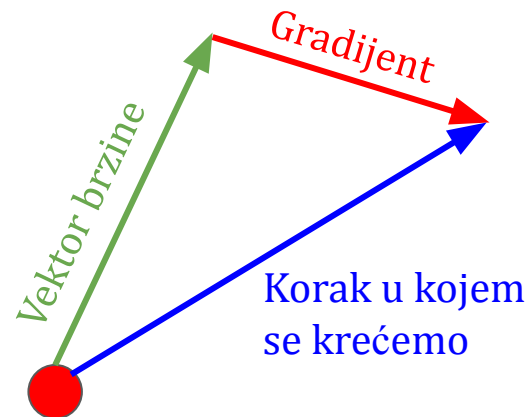
Nesterov Momentum



Kako bi gradijent izgledao ako bismo se kretali u pravcu vektora brzine? Tj. kada bismo malo "zavirili u budućnost"? Izračunaj gradijent u toj tački i kombinuj sa vektorom brzine da bi definisao korak koji će ažurirati koeficijente.

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$



Kako bi gradijent izgledao ako bismo se kretali u pravcu vektora brzine?
Izračunaj gradijent u toj tački i kombinuj sa vektorom brzine da bi definisao korak koji će ažurirati koeficijente.

Nesterov Momentum

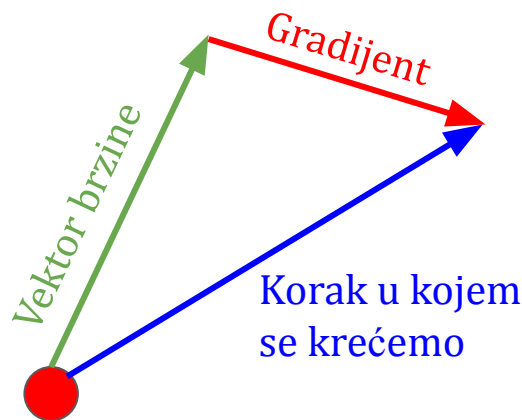
$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
v = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    v = rho * v + dw
    w -= learning_rate * v
```

SGD +
Momentum

Nesterov
Momentum

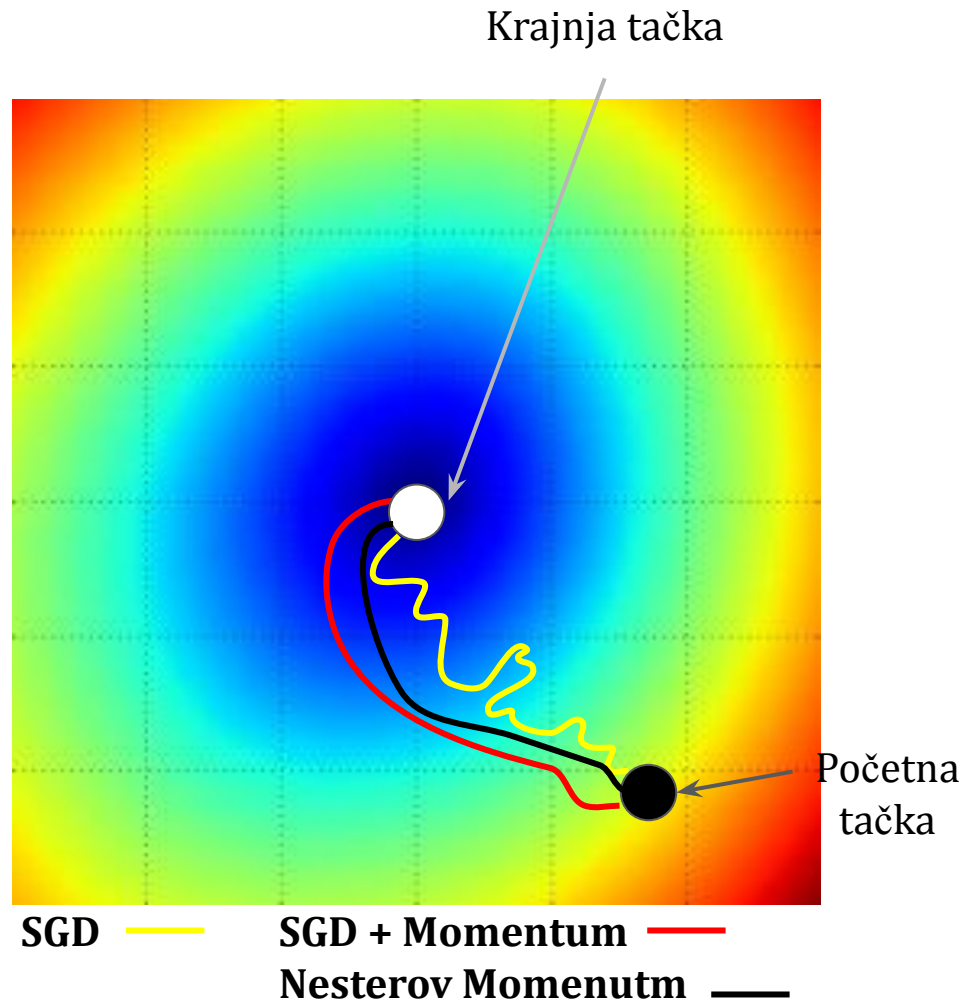
```
v = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    old_v = v
    v = rho * v - learning_rate * dw
    w -= rho * old_v - (1 + rho) * v
```



Kako bi gradijent izgledao ako bismo se kretali u pravcu vektora brzine? Izračunaj gradijent u toj tački i kombinuj sa vektorom brzine da bi definisao korak koji će ažurirati koeficijente.

Nesterov Momentum

- ❑ Ako se vratimo na analogiju sa loptom koja “traži” lokalni minimum neke funkcije, onda se postavlja pitanje da li lopta može imati neko “znanje” o tome kuda ide? I na osnovu toga donijeti zaključak kuda se kretati?
- ❑ Parametre sada ažuriramo ne na osnovu trenutne lokacije, nego na osnovu grube skice/procjene gdje bi parametri mogli biti u sljedećem koraku.
- ❑ Rezultat: brža konvergencija!



AdaGrad

- ❑ Osnovna ideja: prilagođavanje parametra `learning_rate` na osnovu parametara matrice `W` (adaptivna stopa učenja):
 - ❑ Manje ažuriranje (manji `learning_rate`) za parametre koji se povezuju sa značajkama koje se učestalo pojavljuju.
 - ❑ Veće ažuriranje (veći `learning_rate`) za parametre koji se povezuju sa značajkama koje se rijetko pojavljuju.
- ❑ Optimizacija koja je pogodna za rad sa oskudnim podacima (*sparse data*).
- ❑ Prvi put je korišten 2012 od strane Google (prepoznavanje mački iz Youtube video-a).

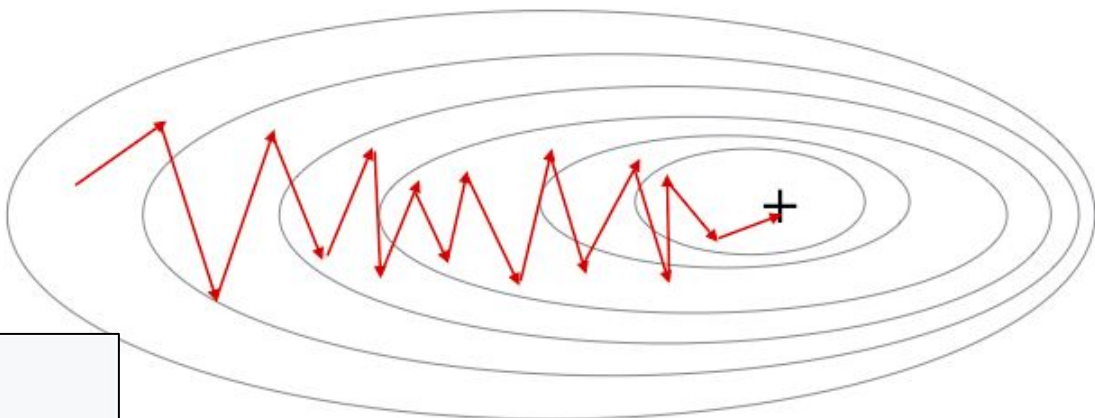
```
grad_squared = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    grad_squared += dw * dw
    w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

<https://www.wired.com/2012/06/google-x-neural-network/>

<http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf>

AdaGrad

Stochastic Gradient Descent



```
grad_squared = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    grad_squared += dw * dw
    w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

Napredak u “strmim” pravcima je prigušen;
ubrzana je napredak u „ravnim“ pravcima.

Glavna slabost Adagrada je nakupljanje kvadratnih gradijenata u nazivniku: Budući da je svaki dodati pojam pozitivan, nakupljena suma raste tokom treninga. To zauzvrat dovodi do smanjenja stope učenja (`learning_rate`) koja na kraju postaje besкраjno mala, u tom trenutku algoritam više nije u mogućnosti steći dodatno znanje. Sljedeći algoritmi imaju za cilj rješavanje ove pogreške.

RMSProp: “Leaky AdaGrad”

```
grad_squared = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    grad_squared += dw * dw
    w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

AdaGrad

RMSProp

```
grad_squared = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
    w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

<https://www.wired.com/2012/06/google-x-neural-network/>

<http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf>

Adam: Momentum + RMSProp

- ❑ **Adam ili Adaptive Moment Estimation:** još jedna metoda koja računa adaptive stope učenja za svaki parametar.

```
moment1 = 0
moment2 = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    moment1 = beta1 * moment1 + (1 - beta1) * dw
    moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
    w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Momentum

```
v = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    old_v = v
    v = rho * v - learning_rate * dw
    w -= rho * old_v - (1 + rho) * v
```

SGD + Momentum

Adam: Momentum + RMSProp

Adam

```
moment1 = 0
moment2 = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    moment1 = beta1 * moment1 + (1 - beta1) * dw
    moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
    w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Momentum

AdaGrad/RMSProp

```
grad_squared = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw
    w -= learning_rate * dw / (grad_squared.sqrt() + 1e-7)
```

RMSProp

Adam: Momentum + RMSProp

Adam

```
moment1 = 0
moment2 = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    moment1 = beta1 * moment1 + (1 - beta1) * dw
    moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
    w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Neka je $\beta_2 = 0.999$. Šta će se desiti u $t=0$?

```
moment1 = 0
moment2 = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    moment1 = beta1 * moment1 + (1 - beta1) * dw
    moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
    moment1_unbias = moment1 / (1 - beta1 ** t)
    moment2_unbias = moment2 / (1 - beta2 ** t)
    w -= learning_rate * moment1_unbias / (moment2_unbias.sqrt() + 1e-7)
```

$\beta_2 = 0.999$
 $\beta_1 = 0.9$
learning_rate =
1e-3, 5e-4, 1e-4

Vizualizacija algoritama

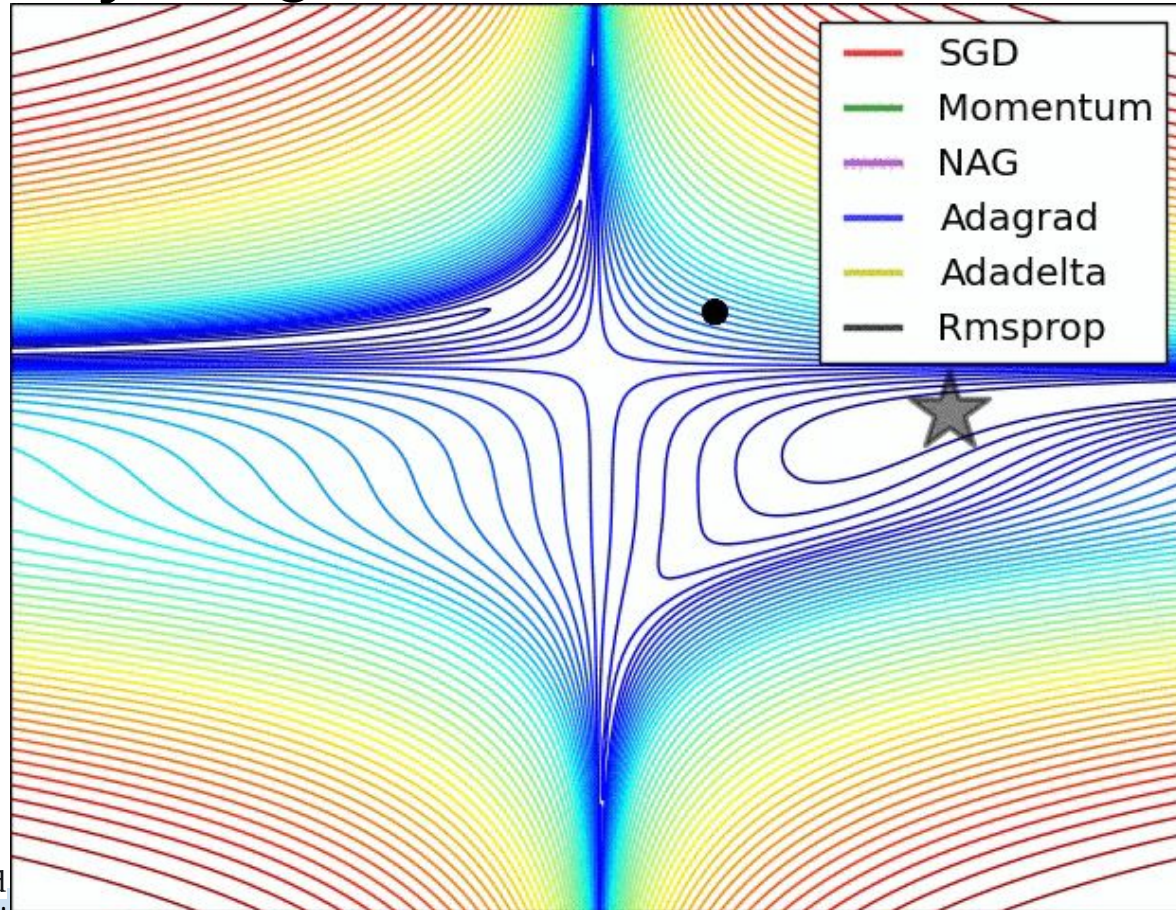


Image credit: Alec Radford

Vizualizacija algoritama

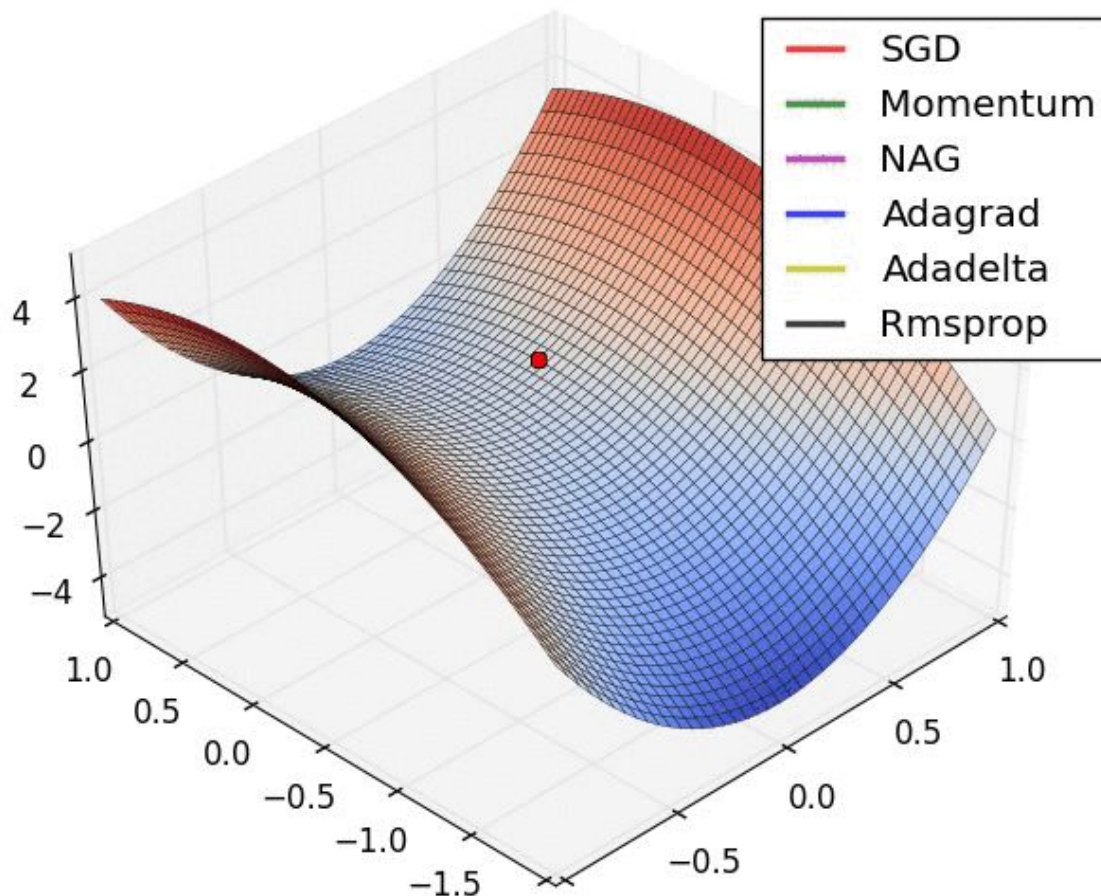
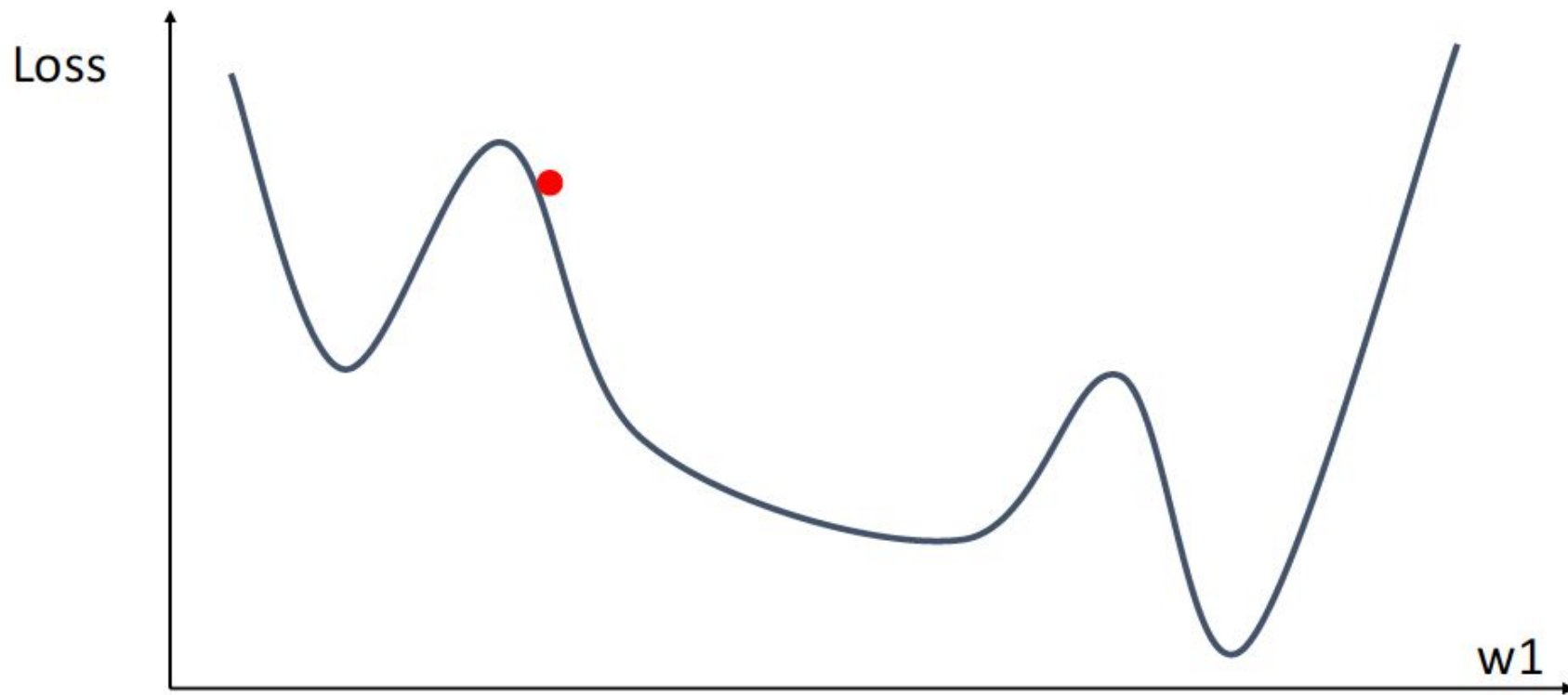


Image credit: Alec Radford

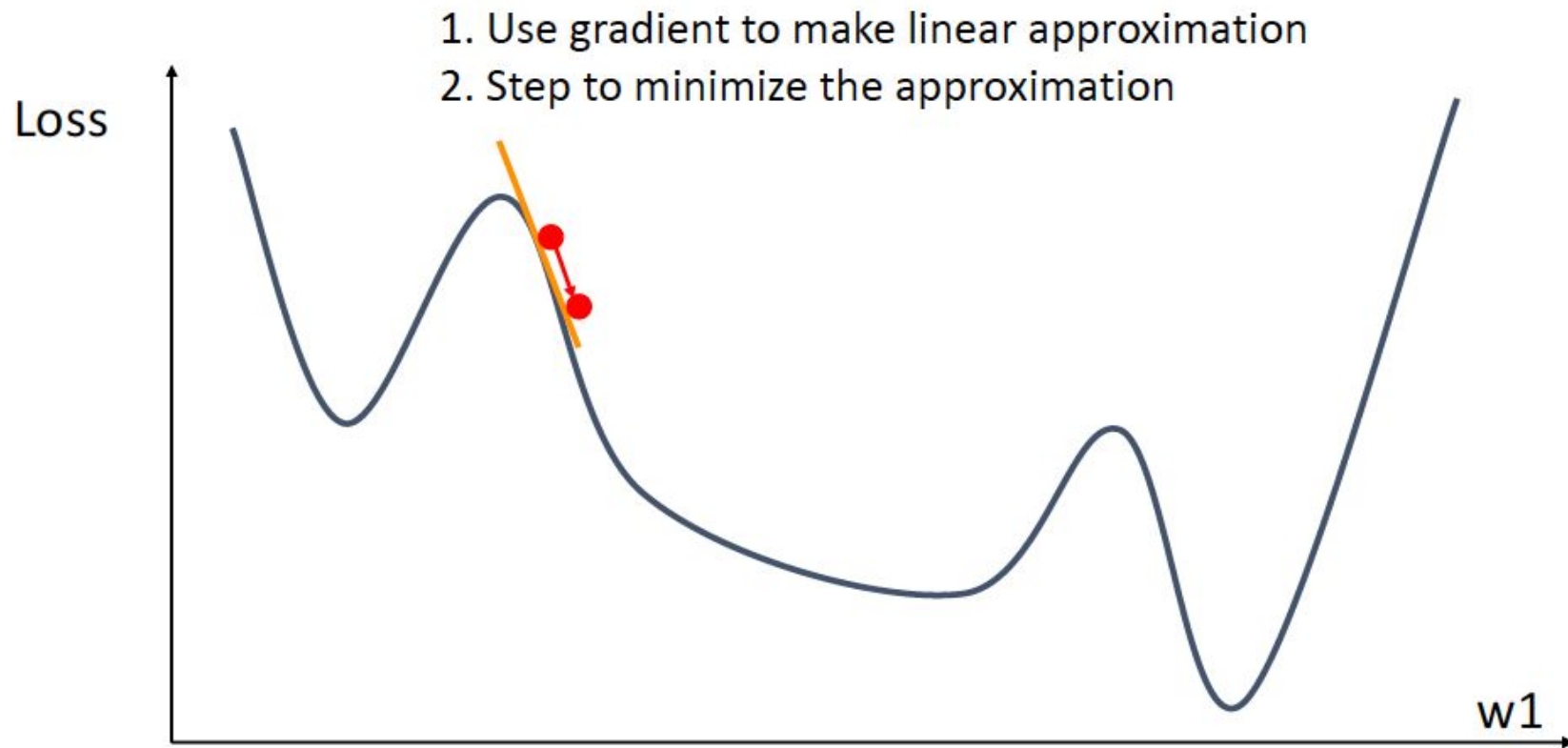
Poređenje algoritama

Algorithm	Tracks first moments (Momentum)	Tracks second moments (Adaptive learning rates)	Leaky second moments	Bias correction for moment estimates
SGD	X	X	X	X
SGD+Momentum	✓	X	X	X
Nesterov	✓	X	X	X
AdaGrad	X	✓	X	X
RMSProp	X	✓	✓	X
Adam	✓	✓	✓	✓

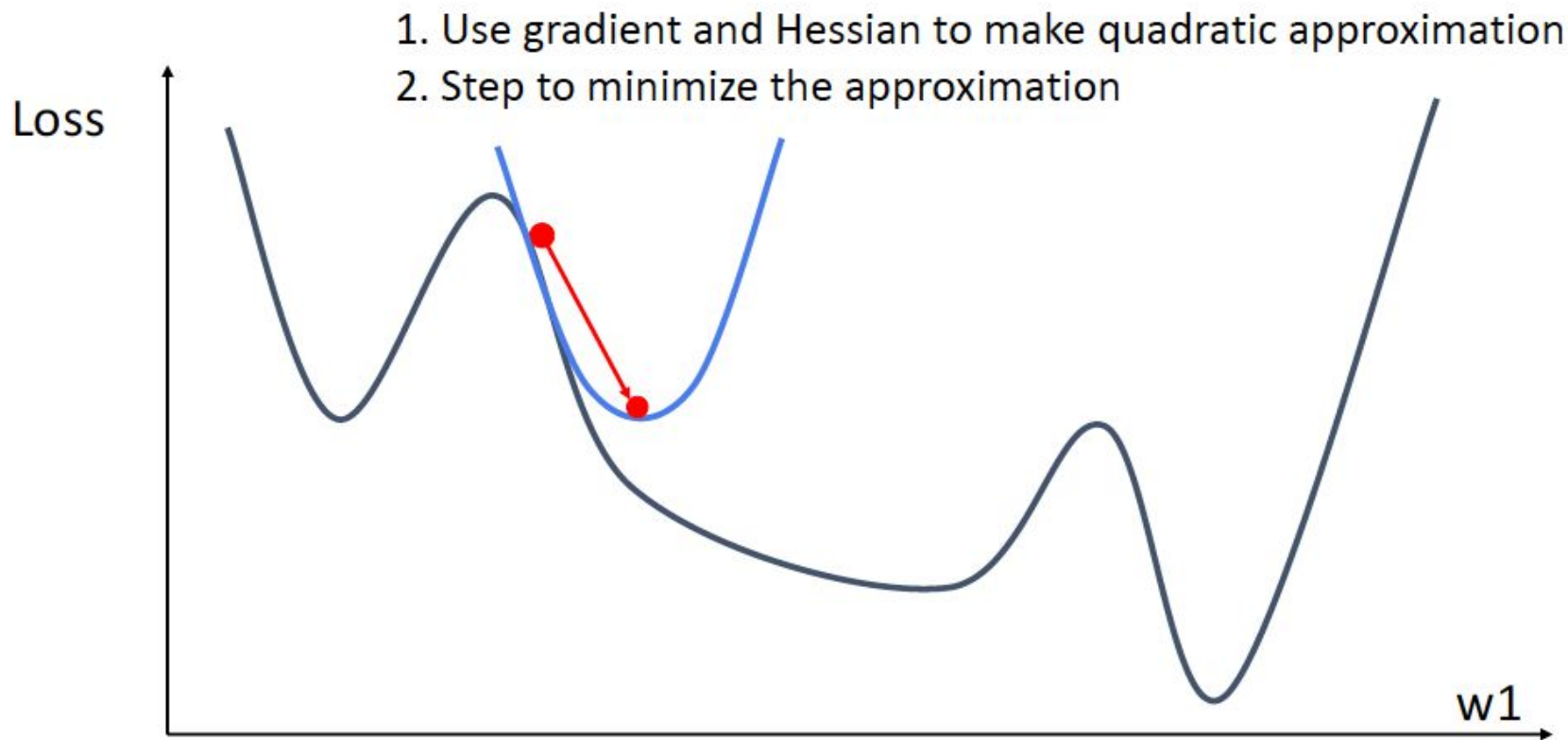
Optimizacije prvog reda



Optimizacije prvog reda



Optimizacije drugog reda



Optimizacije drugog reda

