Obučavanje Neuronskih Mreža
Drugi Deo

Predavač: Aleksandar Kovačević

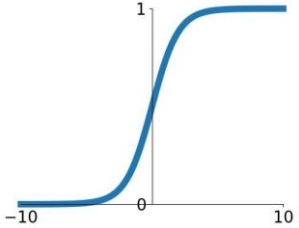
Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

Prošli put: Funkcije Aktivacije

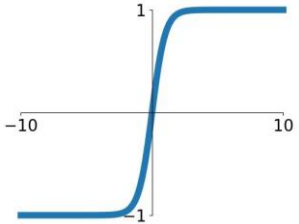
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

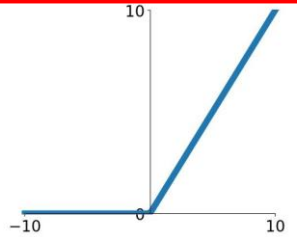
$$\tanh(x)$$



ReLU

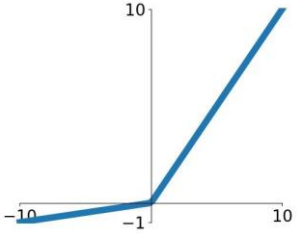
$$\max(0, x)$$

Dobar *default* izbor



Leaky ReLU

$$\max(0.1x, x)$$

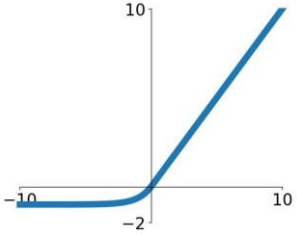


Maxout

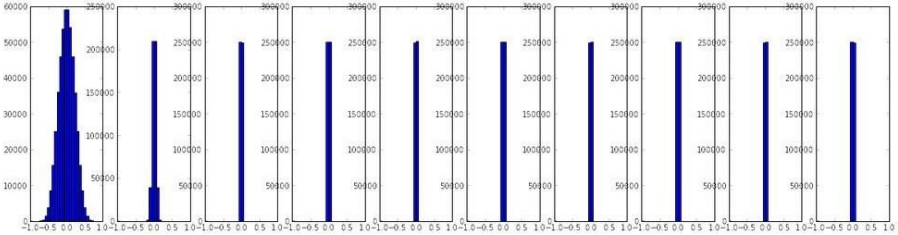
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

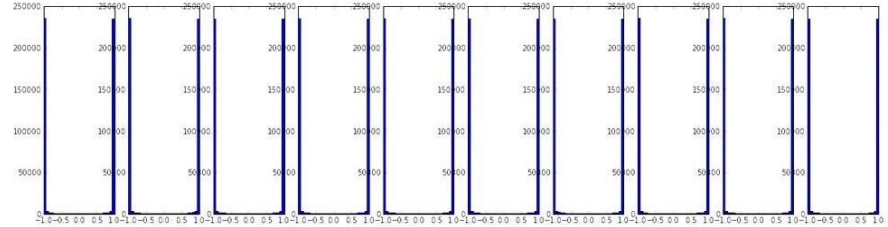
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



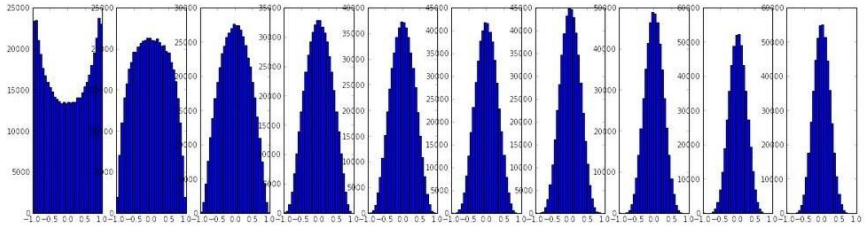
Prošli put: Inicijalizacija težina



inicijalne težine previše male:
Aktivacije i gradijenti postaju 0, nema obučavanja

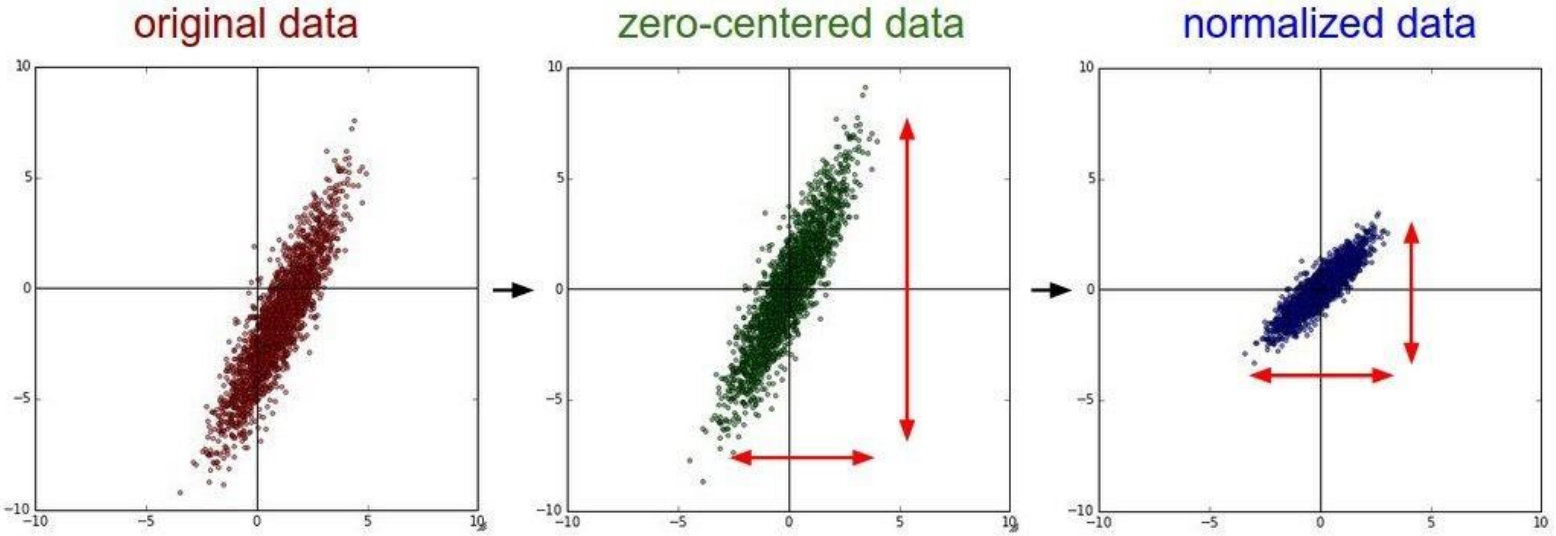


inicijlane težine previše velike:
Aktivacije su u saturaciji (za tanh ili sigmoid), gradijenti su 0, nema obučavanja



inicijalne težine dobro izabrane:
Dobre distribucije aktivacija po slojevima, obučavanje dobro napreduje

Prošli put: Predprocesiranje Podataka



Prošli put: Batch Normalizacija

Ulaz: $x : N \times D$

Parametri koji se uče:

$$\gamma, \beta : D$$

Vrednosti koje se računaju tokom obučavanja:

$$\mu, \sigma : D$$

$$\hat{x} : N \times D$$

Izlaz: $y : N \times D$

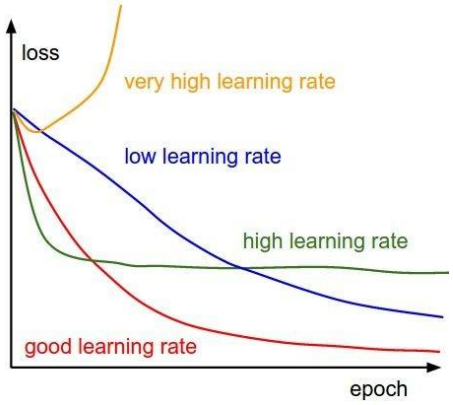
$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

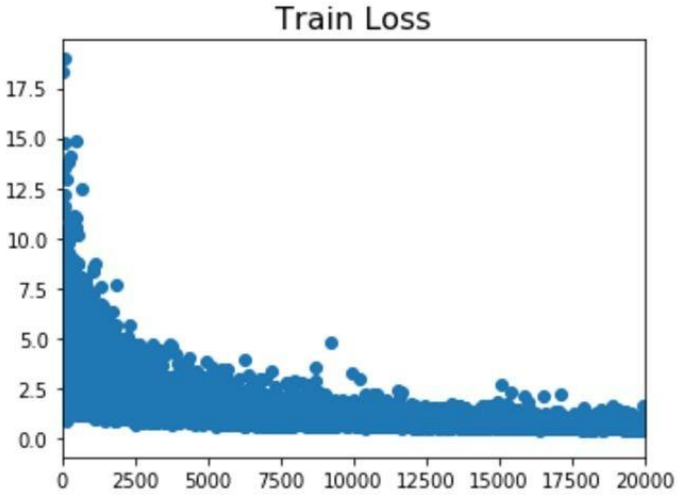
$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

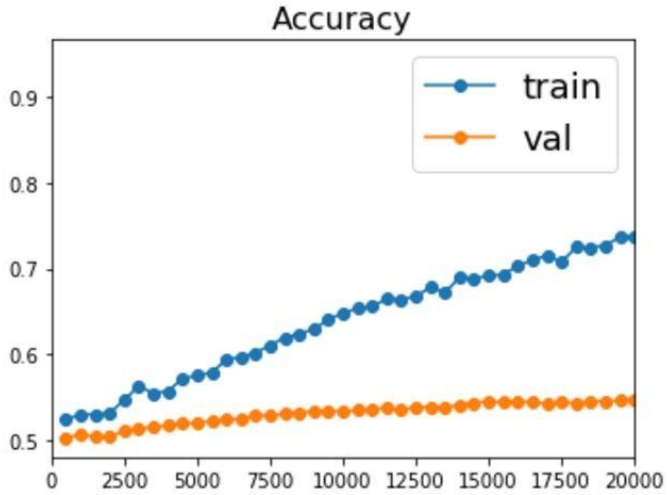
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

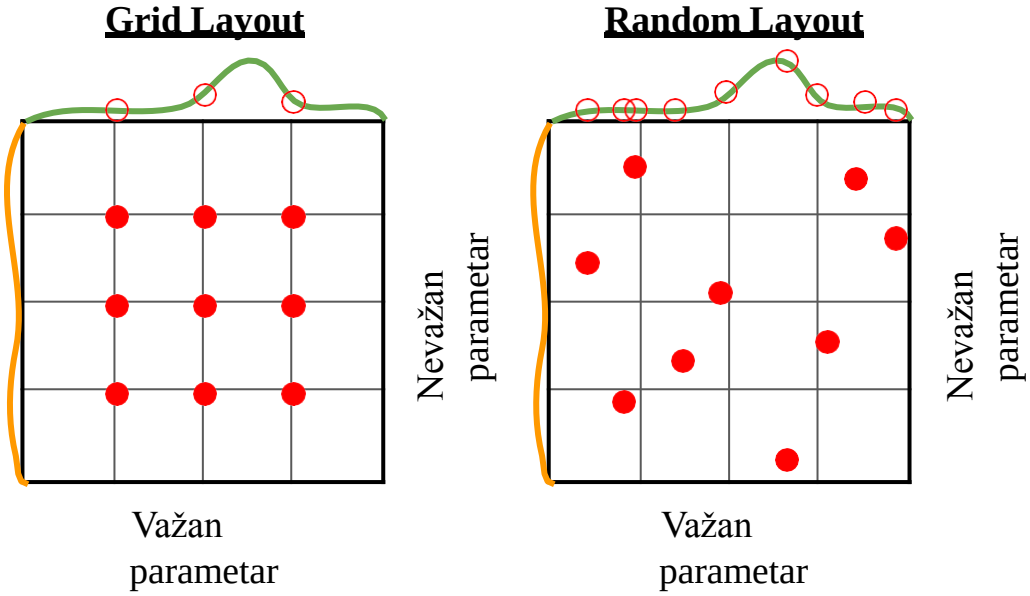
Prošli put: Praćenje Procesa Učenja







Prošli put: Optimizacija Hiperparametara



Od grublje ka finijoj pretrazi

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

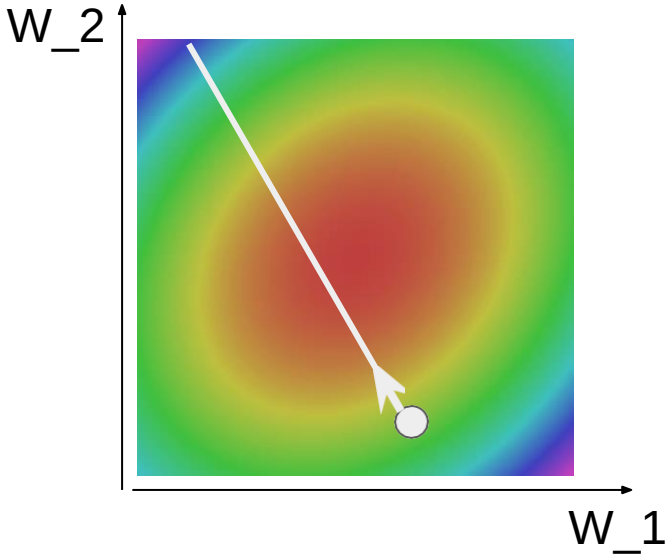
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

Danas

- Poboljšanja opitmizacije
- Regularziacija
- Učenje Transferom (*Transfer Learning*)

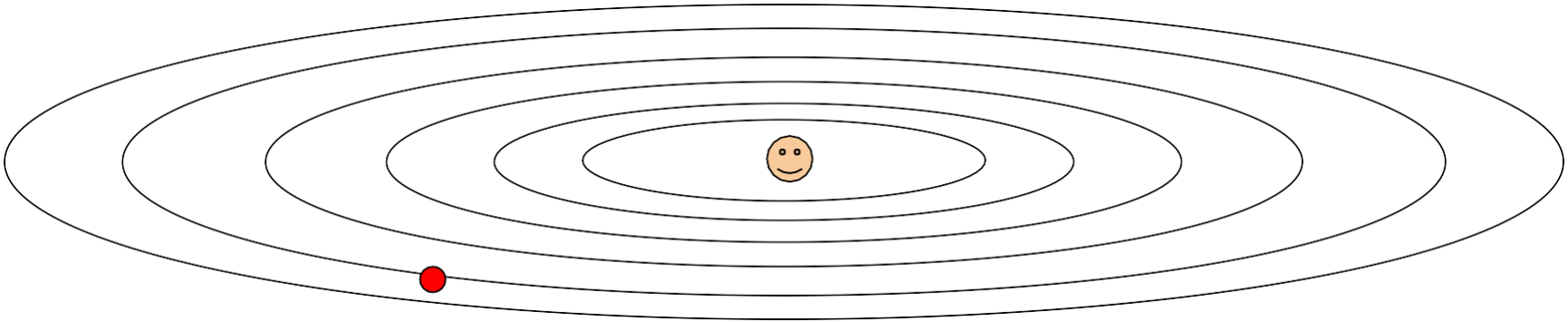
Optimizacija

```
# Vanilla Gradient Descent

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



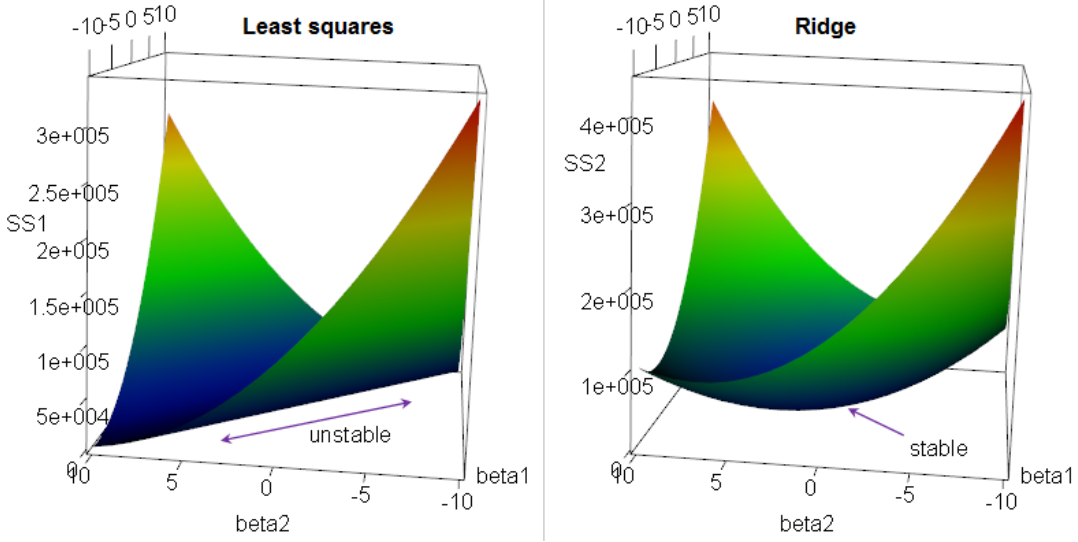
Optimizacija: Problemi sa SGD

Šta ako se greška brzo menja po jednoj koordinatnoj osi, a sporo po drugoj?
Kako onda izgleda gradijentni spust?



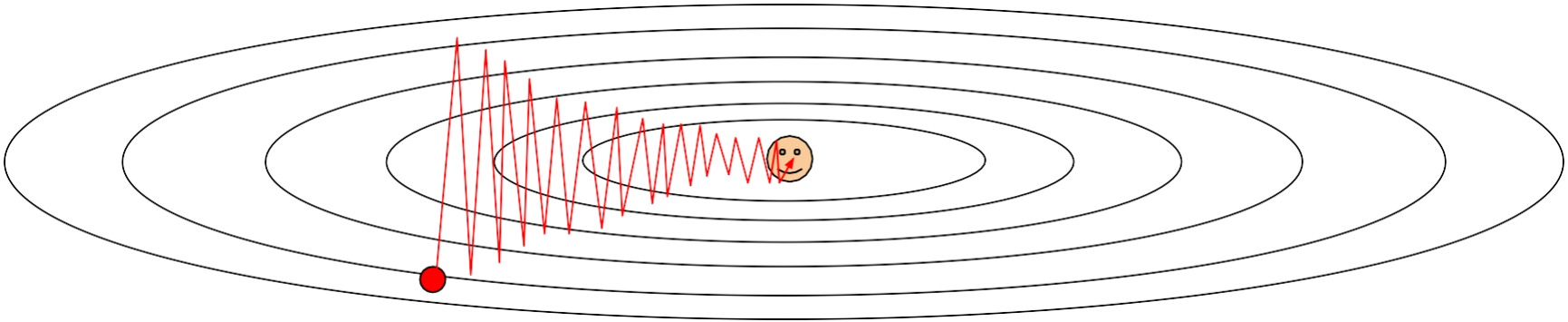
Optimizacija: Problemi sa SGD

Šta ako se greška brzo menja po jednoj koordinatnoj osi, a sporo po drugoj?
Kako onda izgleda gradijentni spust?



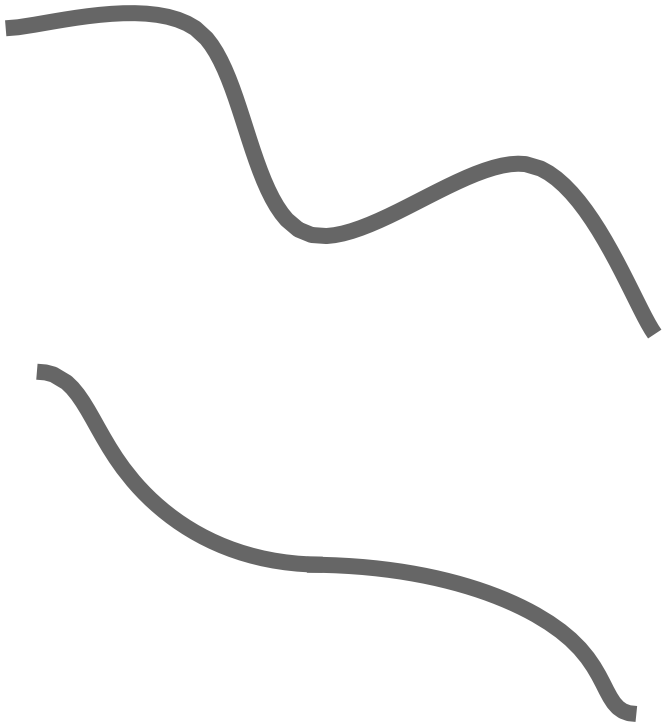
Optimizacija: Problemi sa SGD

Šta ako se greška brzo menja po jednoj koordinatnoj osi, a sporo po drugoj?
Kako onda izgleda gradijentni spust?
Jako spor napredak u plitkom smeru, „cimanje“ u strmom smeru



Optimizacija: Problemi sa SGD

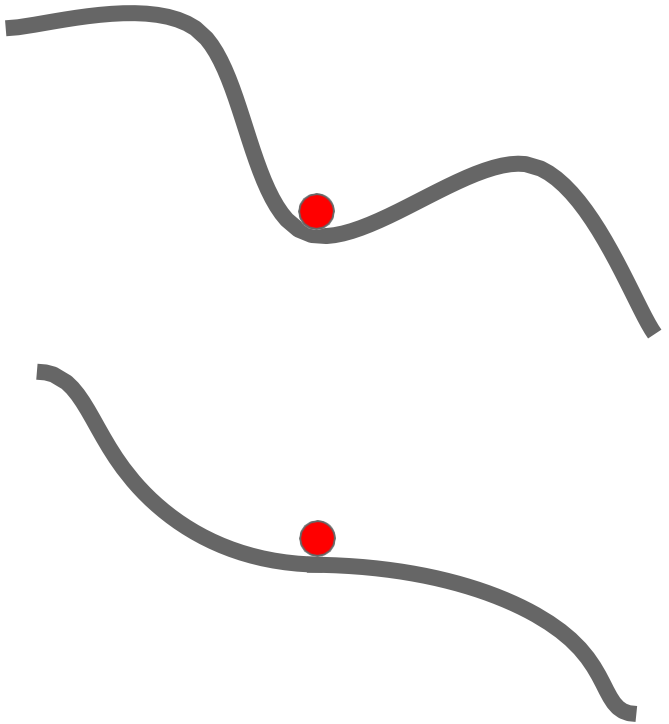
Šta ako funkcija greške ima **lokalne minimume** ili **tačke prevoja**?



Optimizacija: Problemi sa SGD

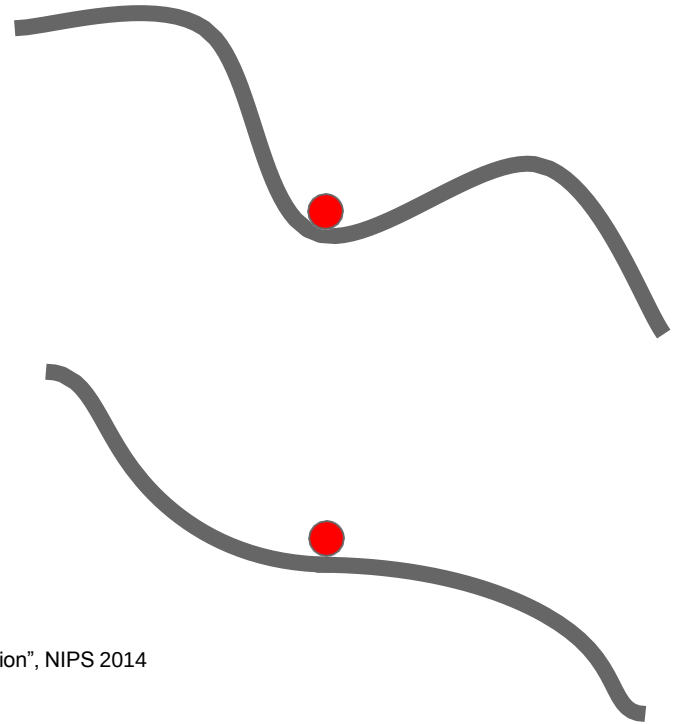
Šta ako funkcija greške ima **lokalne minimume** ili **tačke prevoja**?

Gradijent je 0 ili jako mali i SGD se zaglavljuje



Optimizacija: Problemi sa SGD

Šta ako funkcija greške ima **lokalne minimume** ili **tačke prevoja**?



Tačke prevoja su česte u visko-dimenzionim prostorima

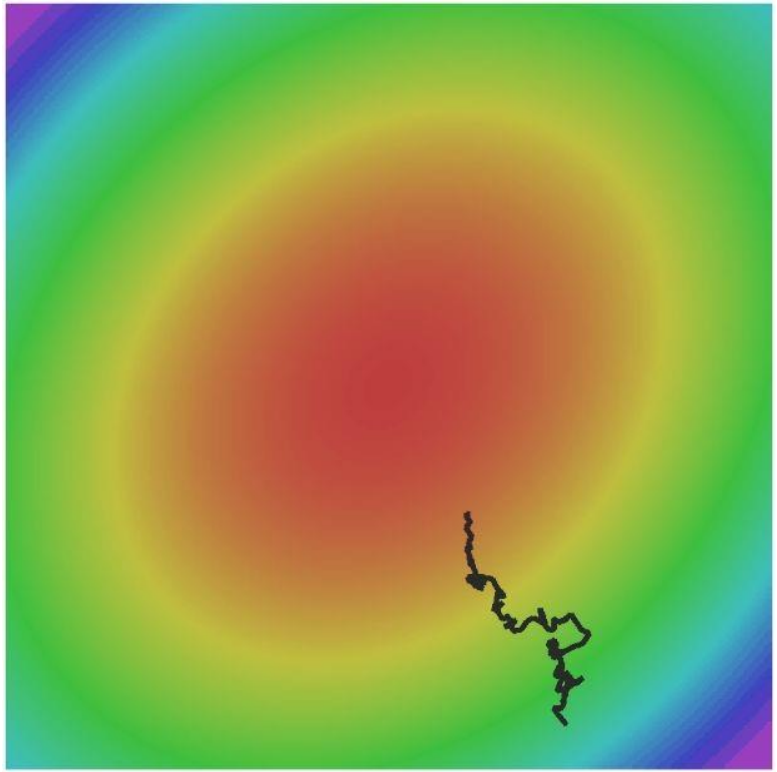
Dauphin et al, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”, NIPS 2014

Optimizacija: Problemi sa SGD

Gradijente računamo pomoću mini podskupova, pa nisu uvek dosledni više su kao na slici! Termin je *noisy gradients*.

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

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

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

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

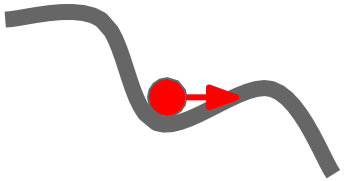
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

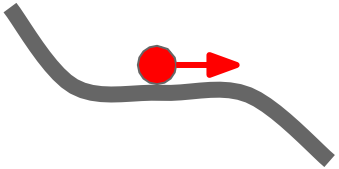
- Nakupljamo “ubrzanje” kao sumu prethodnih gradijenata
- Ro se tumači kao “trenje”; tipično je rho=0.9 ili 0.99

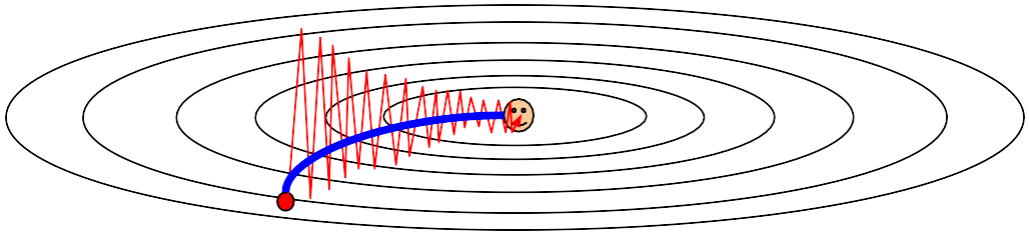
SGD + Momentum

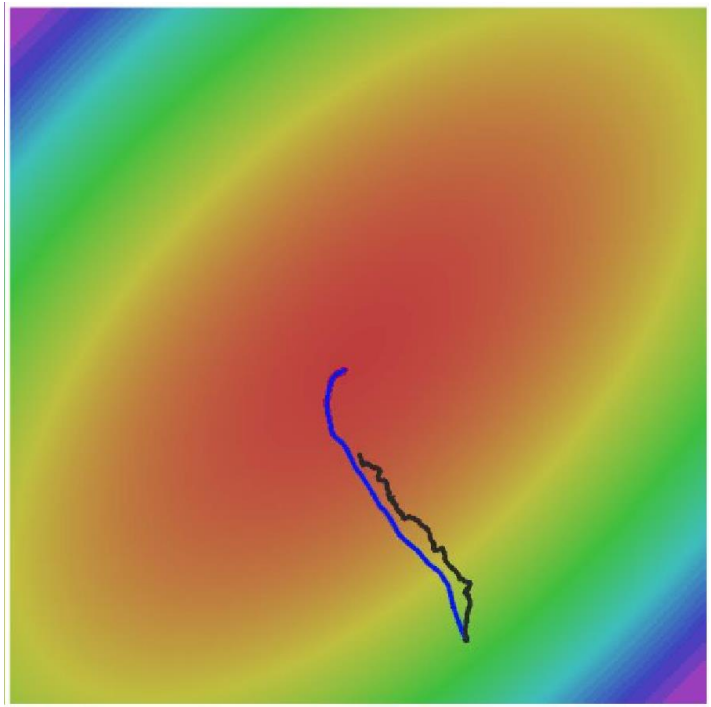
Lolalni min.



Tačka prevoja







AdaGrad

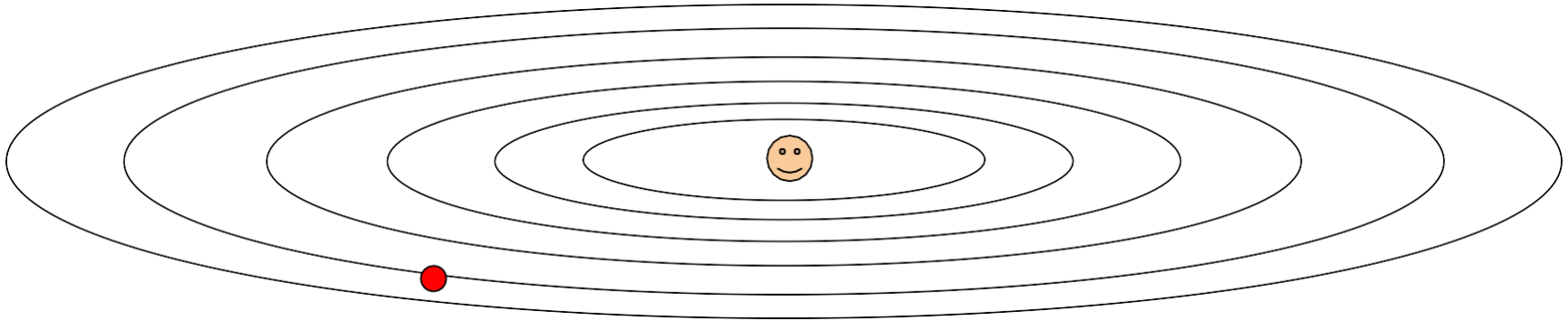
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Skaliramo gradijent pomoću sume kvadrata dosadašnjih gradijenata. Skaliranje je po svakoj dimenziji posebono. Kod je vektorski.

Duchi et al, “Adaptive subgradient methods for online learning and stochastic optimization”, JMLR 2011

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Kako izgleda AdaGrad?

RMSProp

AdaGrad

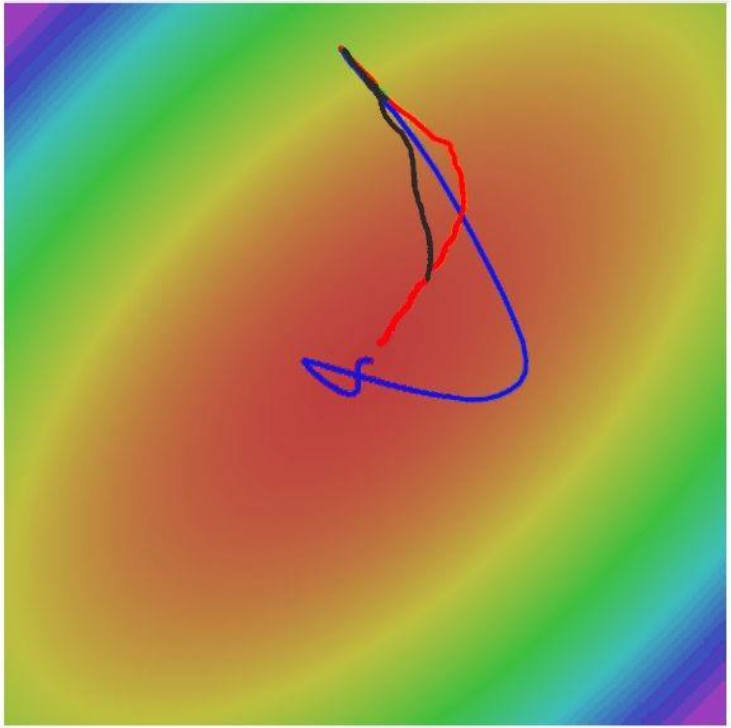
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

RMSProp



SGD

SGD+Momentum

RMSProp

Adam (skoro)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (skoro)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Kao RMSProp sa momentumom

Šta se dešava u prvom koraku?

Adam sa beta1 = 0.9, beta2 = 0.999, i learning_rate = 1e-3 ili 5e-4 je odličan izbor na početku obučavanja.

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Adam (kompletan)

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Promena Bijasa

AdaGrad / RMSProp

Teorijske razloge za korekcije oivičene zelnom bojom nećemo objašnjavati, za detalje pogledati rad dole.

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (kompletan)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

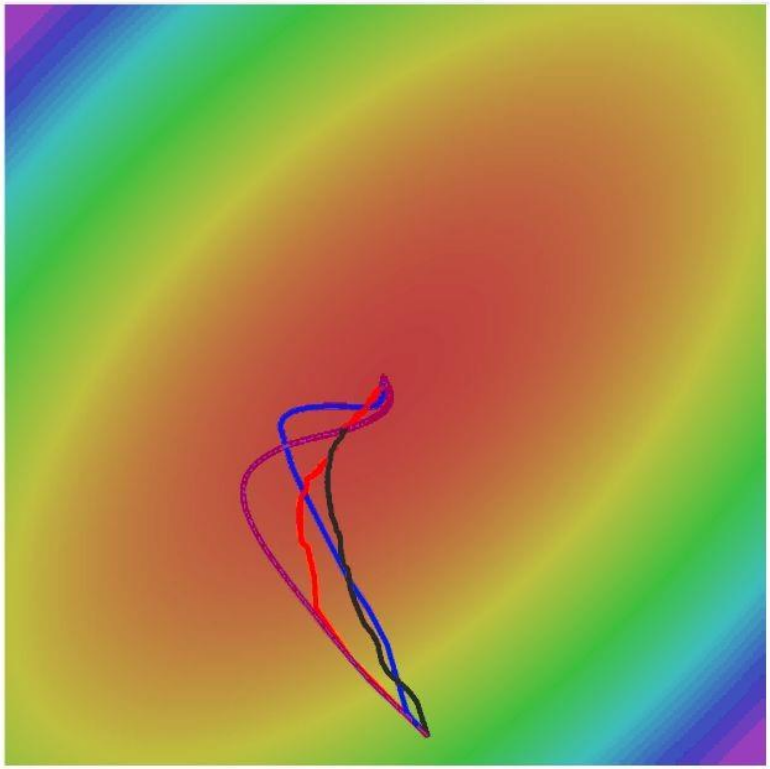
AdaGrad / RMSProp

Teorijske razloge za korekcije ovičene zelnom bojom nećemo objašnjavati, za detalje pogledati rad dole.

Adam sa beta1 = 0.9,
beta2 = 0.999, i learning_rate = 1e-3 ili 5e-4 je odličan izbor na početku obučavanja.

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Adam



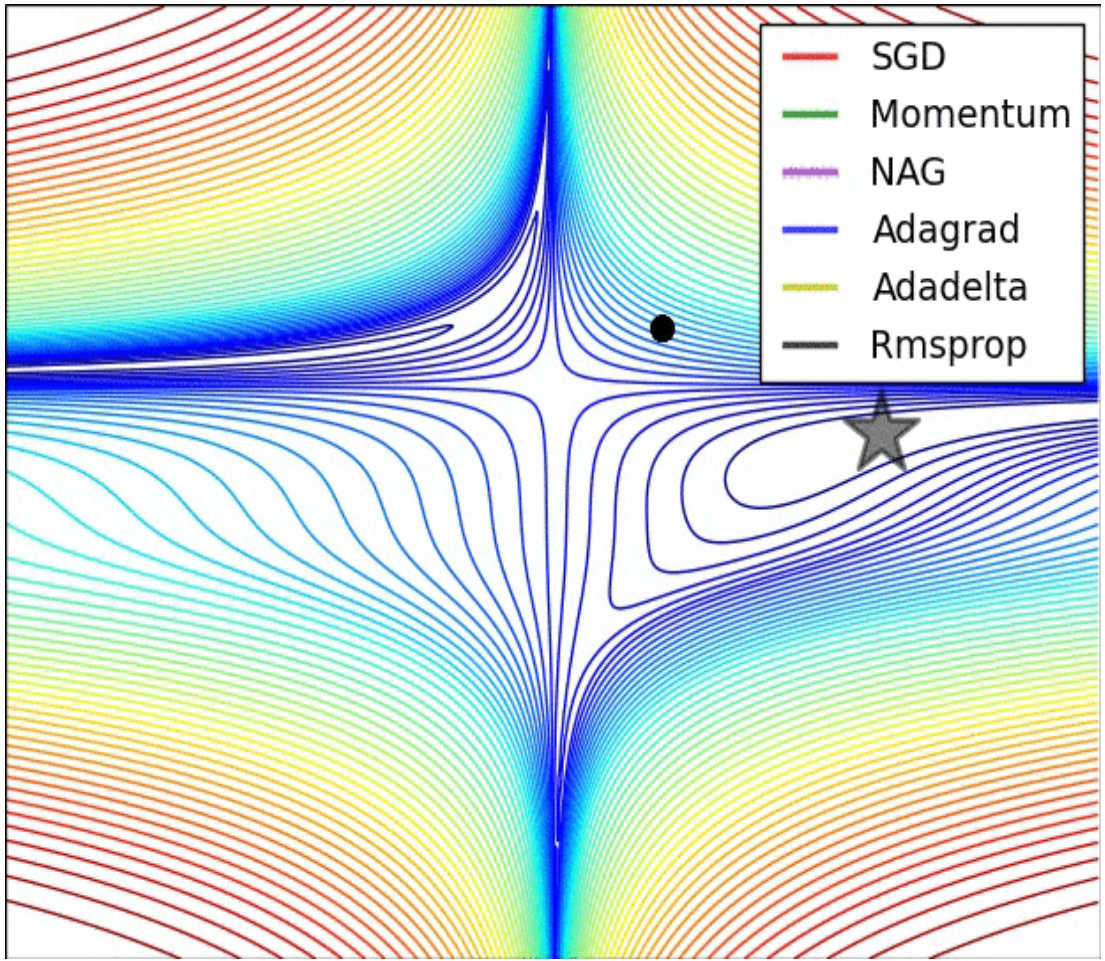
SGD

SGD+Momentum

RMSProp

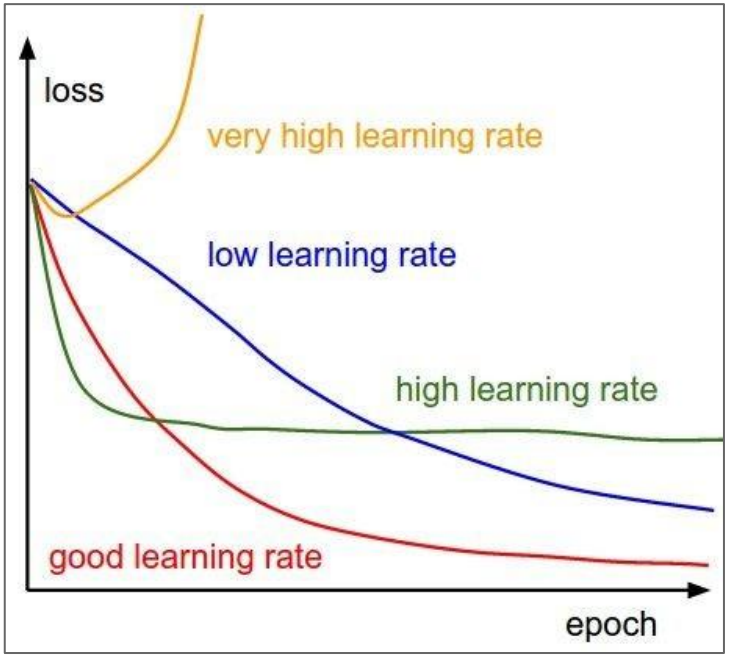
Adam

Ponašanje različitih formula za promene težina



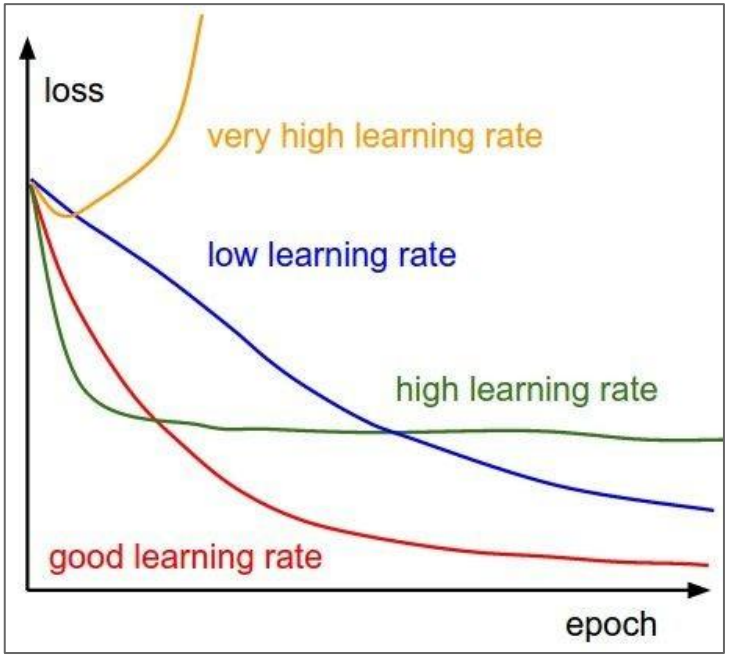
(image credits to Alec Radford)

SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.



Koji je od koraka učenja sa slike najbolji?

SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.



Očigledno je koja je kriva na slici najbolja, ali se ona neće dobiti sa fiksnom vrednošću koraka učenja

=> Korak učenja treba da opada kroz iteracije!

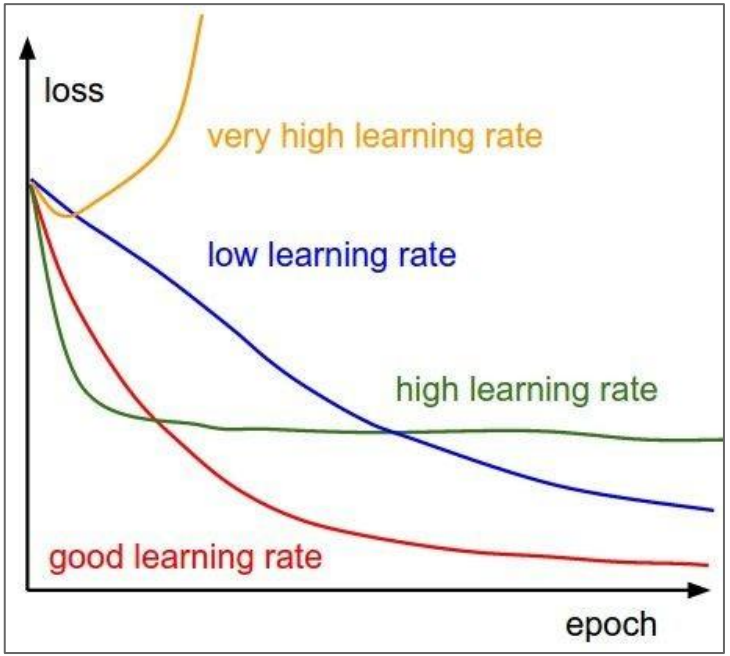
***opadanje po iteracijama (step decay)*:**
npr. delimo korak učenja sa 2 uvek nakon određenog broja epoha.

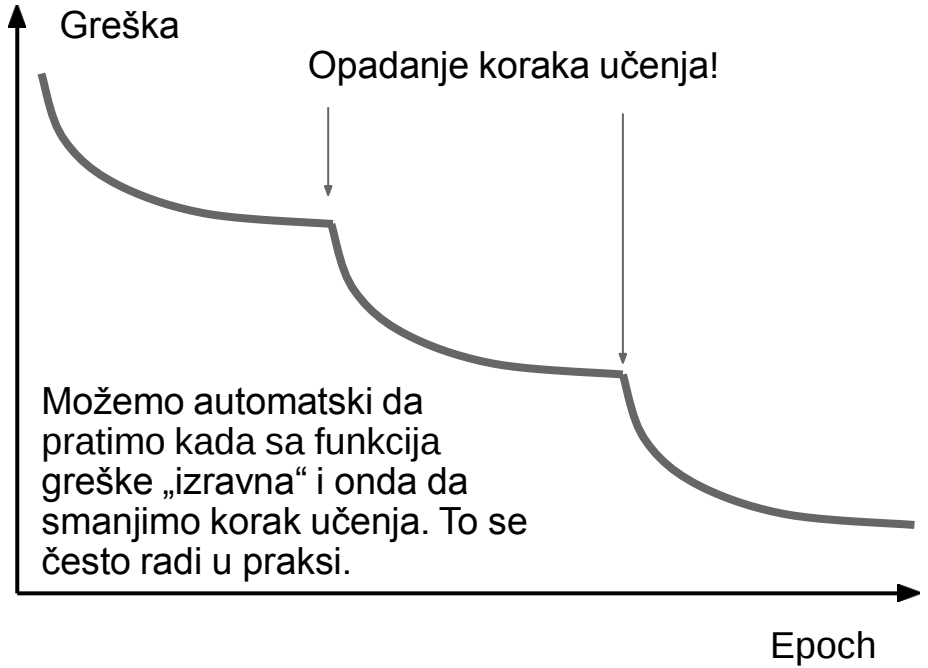
eksponencijalno opdanje: $\alpha = \alpha_0 e^{-kt}$

1/t opadanje:

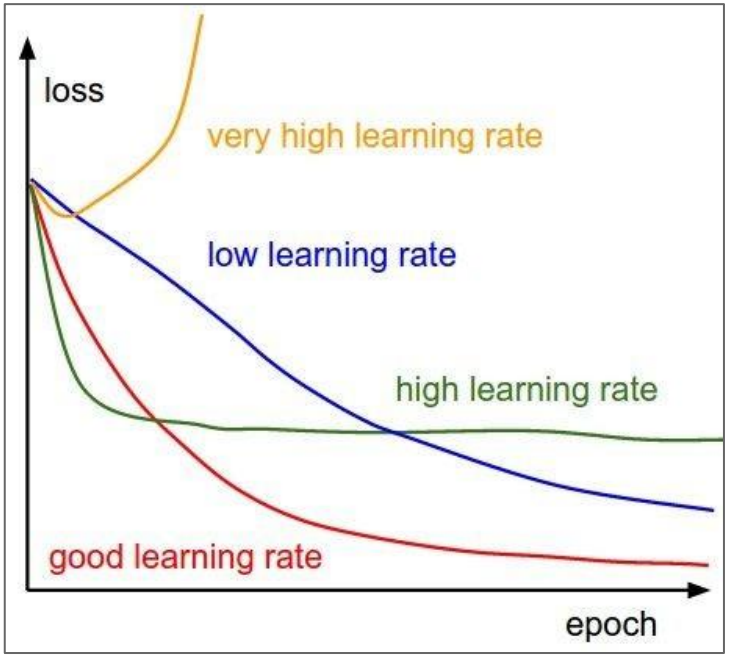
$$\alpha = \alpha_0 / (1 + kt)$$

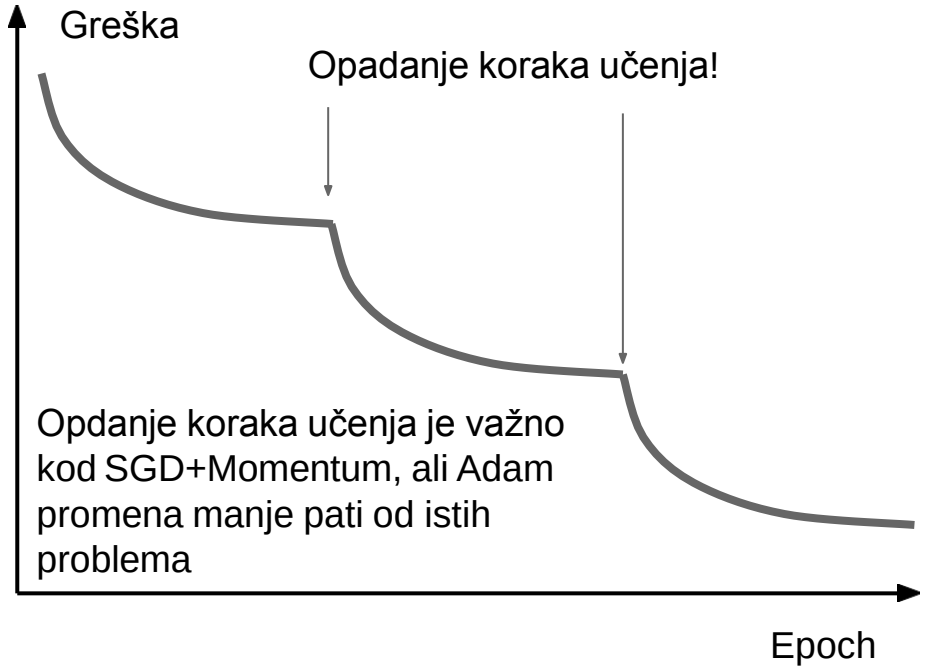
SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.





SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.



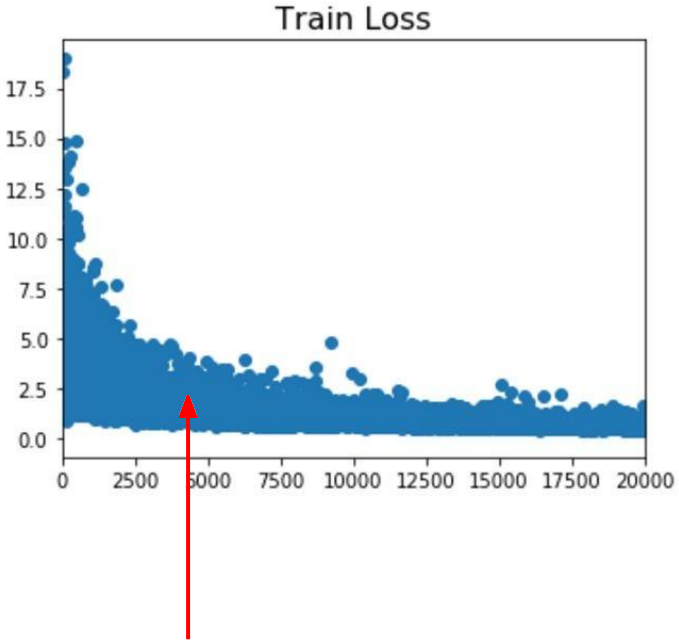


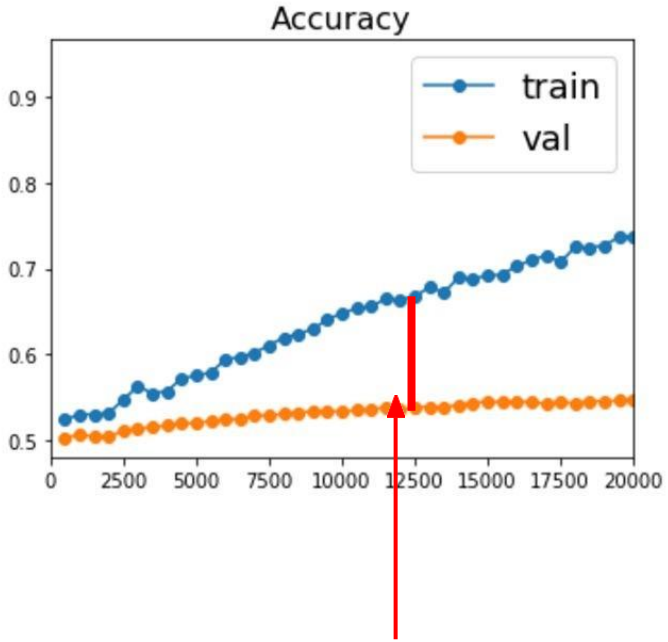
U praski:

- **Adam** je dobar izbor u većini slučajeva.

- Ako vam hardver dozvoli da promenu parametara raditi posle celog ob. skupa, a ne mini podskupova onda probajte **L-BFGS** (ali imajte u vidu da je osetljiv na šum u podacima)
- **L-BFGS** je optmizacioni metod koji koristi drugi izvod, pomenut je samo da znate da postoji, nećemo se baviti njime na ovom kursu. Realno se retko koristi u Deep Learning.

Rešili smo training grešku, ali šta ako greška na validacionom skupu ipak ne opada?





Dobri optimizacioni algoritmi će smanjiti training rešku

Ali, mi želimo da na model dobro radi na nepoznatim podacima. Kako da smanjimo raziku sa slike?

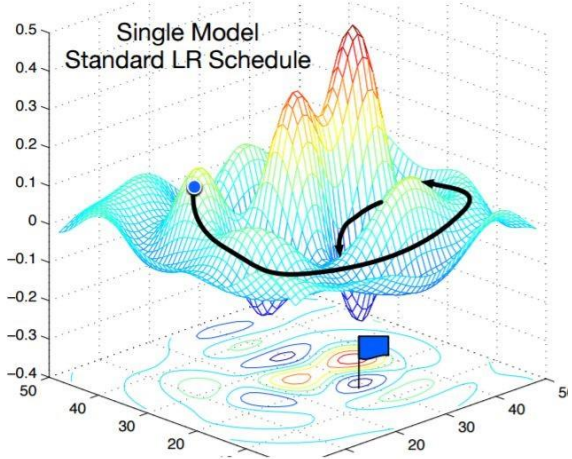
Ansambli Modela

1. Obučite više različitih nezavisnih modela na istim podacima
2. Kada se primenjuju uzmite prosek njihovih predikcija (ili neki bolji metod glasanja)

Uživaje u dodatnih 2% tačnosti ☺

Ansambli Modela: Saveti

Umesto da obučavate više nezavisnih modela, čuvamo neke od modela kroz iteracije (*snapshots*) i koristimo njih.



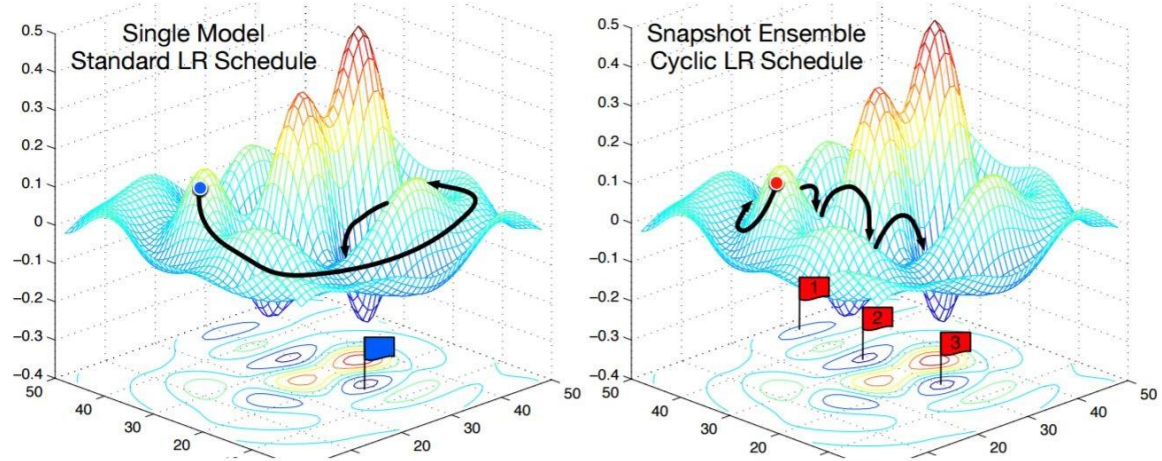
Na primer uzmemo model iz iteracije 1000, 2000, 3000,...., 10000

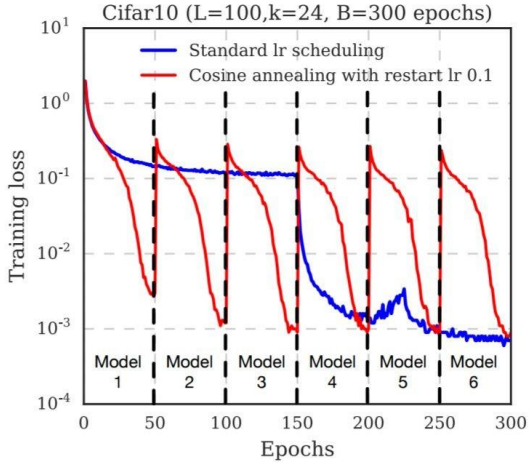
i onda imamo ansambl od 10 modela

Loshchilov and Hutter, “SGDR: Stochastic gradient descent with restarts”, arXiv 2016
Huang et al, “Snapshot ensembles: train 1, get M for free”, ICLR 2017

Ansambli Modela: Saveti

Umesto da obučavate više nezavisnih modela, čuvamo neke od modela kroz iteracije (*snapshots*) i koristimo njih.

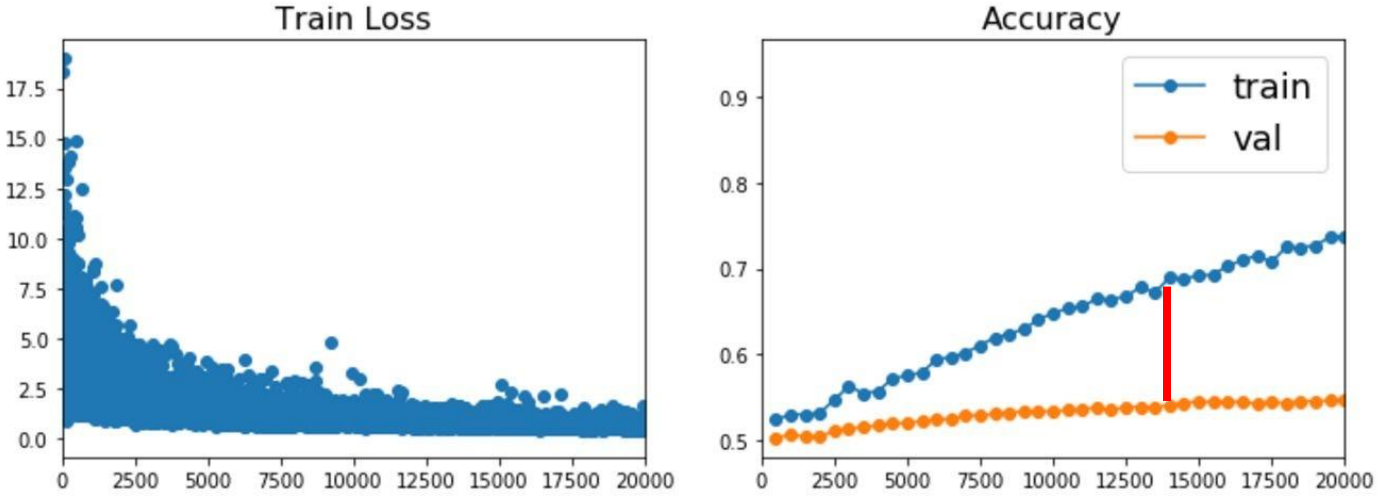




Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

ciklične promene koraka učenja mogu da budu dobar izvor ansambla!

Kako dodatno poboljšati performanse?



Regularizacija

Regularizacija: Dodajemo još jedan izraz u funckiju greške

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

Obično se koristi:

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

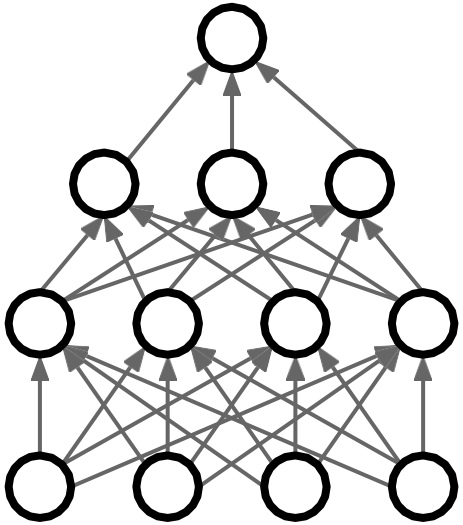
L1 regularizacija $R(W) = \sum_k \sum_l |W_{k,l}|$

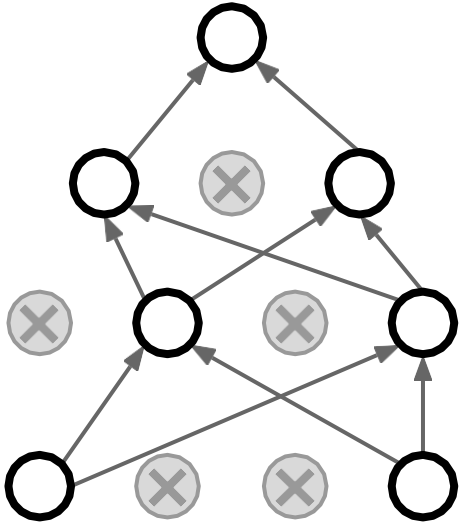
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularizacija: Dropout

Pri svakoj propagaciji unapred, na slučajan način isključiti neke od neurona;

Verovatnoća da će neuron biti isključen je hiper parametar; 0.5 je uobičajena vrednost



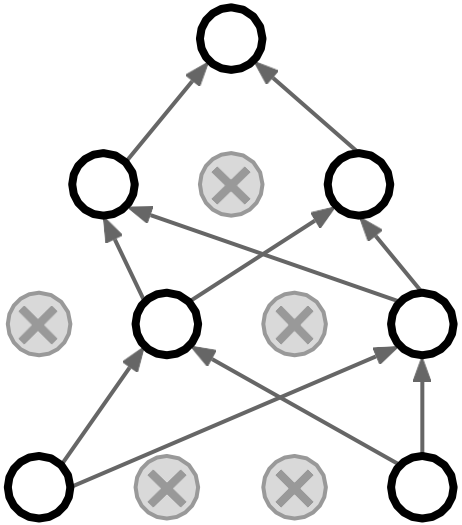


Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Regularizacija: Dropout

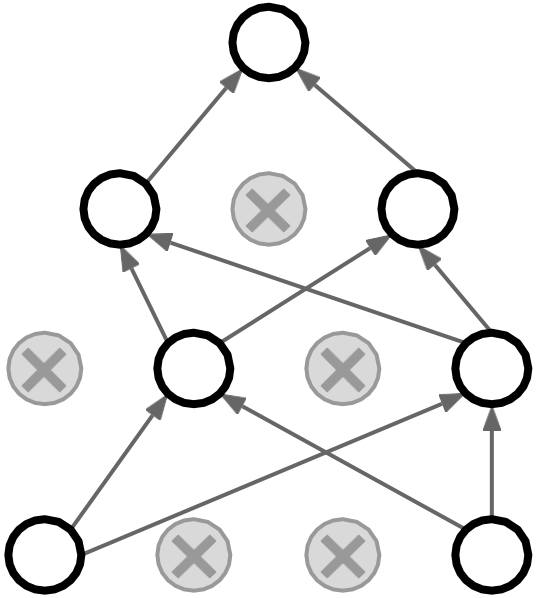
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Primer: propagacija unapred za 3-slojnu mrežu uz dropout

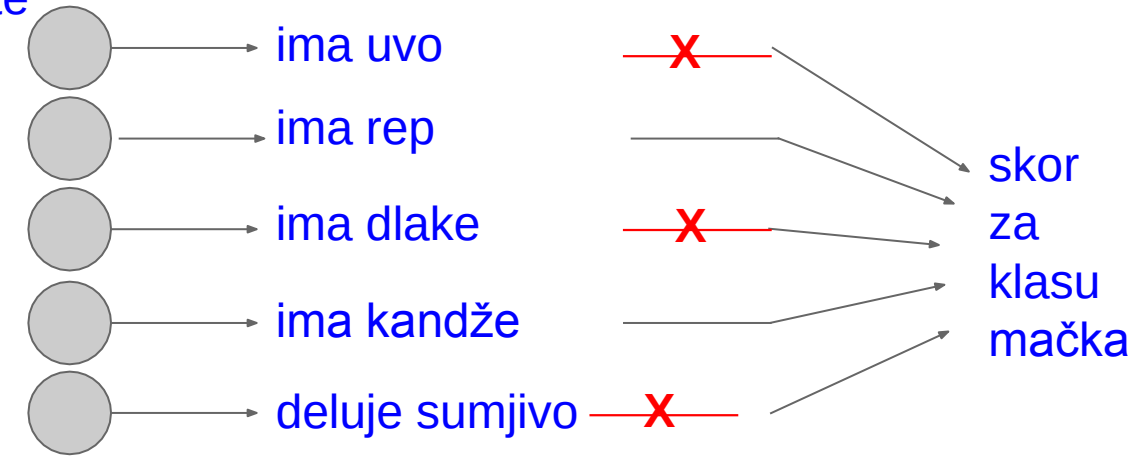


Regularizacija: Dropout

Kako ovo može biti dobra ideja?

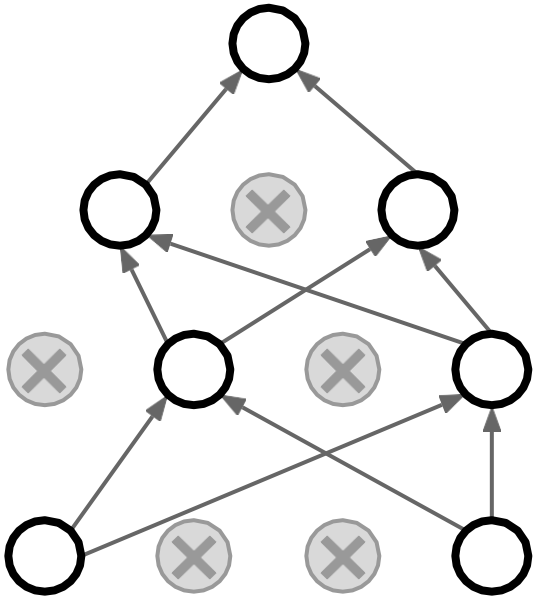


Nateraće mrežu da ima redudantne načine za predikciju iste stvari; Sprečava previše veliko oslanjanje na određene atribute



Regularizacija: Dropout

Kako ovo može biti dobra ideja?



Još jedna interpretacija:

Dropout je obučavanje **ansambla modela** (koji dele parametre).

Svaka binarna maska je jedan model

Potpuno povezan sloj sa 4096 neurona ima $2^{4096} \sim 10^{1233}$ mogućih maski! ~ 10^{82} atoma u svemiru…

Dropout: Primena na nepoznate primere

Izlaz (klasa)

ulaz (slika)

Dropout će učiniti izlaz nepredvidivm jer kad primenjujemo mrežu na nepoznate primere koristimo sve neurone.

$$y = f_W(x, z)$$

Random maska

To ćemo rešiti tako što ćemo da „uprosečimo“ slučajnost

prosečna aktivacija koju je neuron imao tokom obučavanja

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

aktivacija

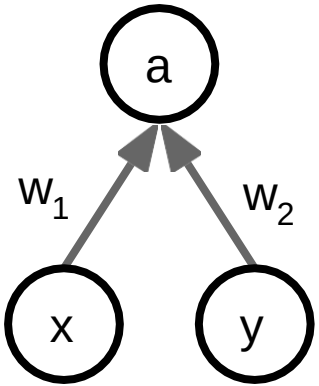
verotnoća da je neuron aktivan

Naravno da nećemo da rešavamo integral …

Dropout: Primena na nepoznate primere

Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Posmatrajmo jedan neuron.

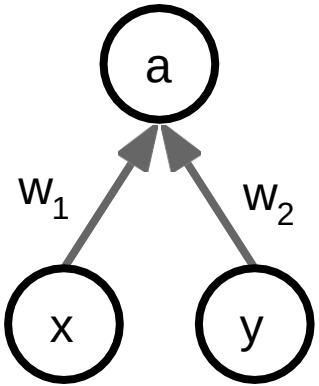
Dropout: Primena na nepoznate primere

Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

Posmatrajmo jedan neuron.

Prilikom primene na nepoznati primer imamo: $E\big[a\big] = w_1 x + w_2 y$

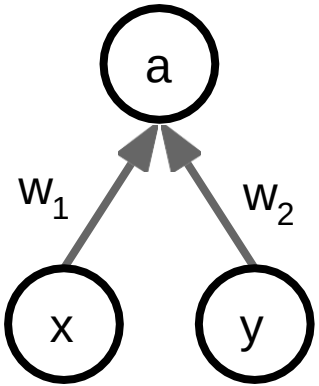


Dropout: Primena na nepoznate primere

Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

Posmatrajmo jedan neuron.



Prilikom primene na nepoznati primer imamo:

$$E\big[a\big] = w_1 x + w_2 y$$

Prilikom obučavanja imamo:

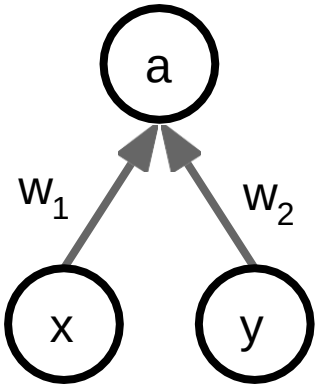
$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Dropout: Primena na nepoznate primere

Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

Posmatrajmo jedan neuron.



Prilikom primene na nepoznati primer imamo: $E\big[a\big] = w_1 x + w_2 y$

Prilikom obučavanja imamo:

$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

To znači da prilikom primene na nepoz. množimo aktivaciju sa verovatnoćom za dropout. Ovde je to 0.5

Dropout: Primena na nepoznate primere

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Prilikom primene na nepoz. prim. svi neuroni su uključeni
=> Moramo da skaliramo aktivacije svih neurona:
izlaz prilikom primene na nepoz. prim. = očekivani izlaz prilikom obučavanja

Dropout Rezime

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

dropout u propagaciji unapred

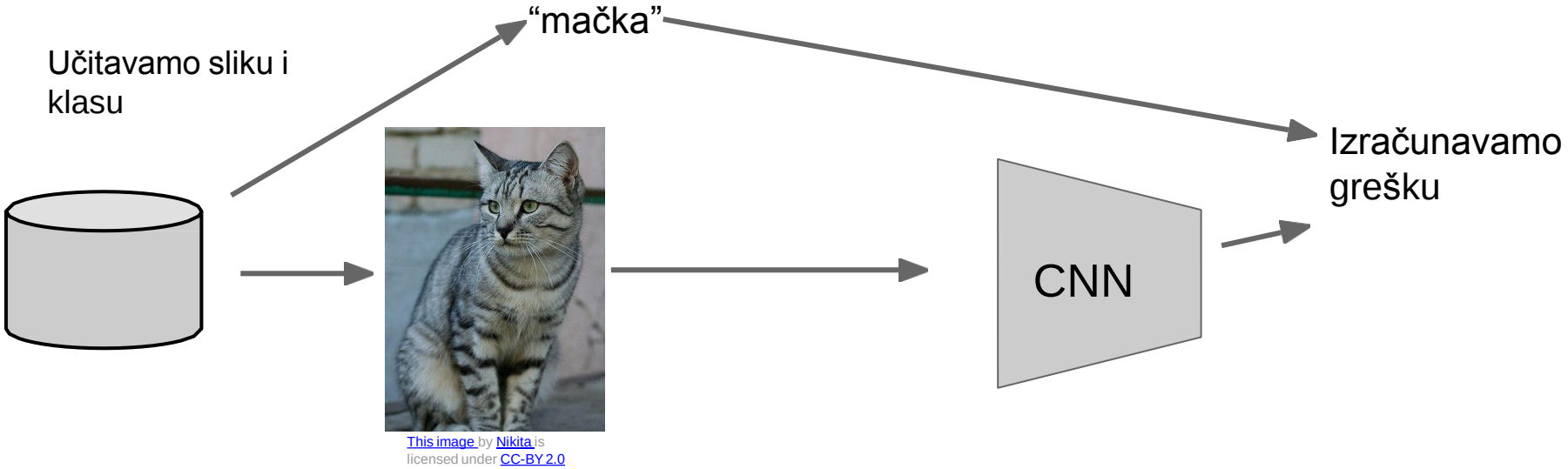
skaliramo kad radimo predikciju

Često se koristi: “Invertovani dropout”

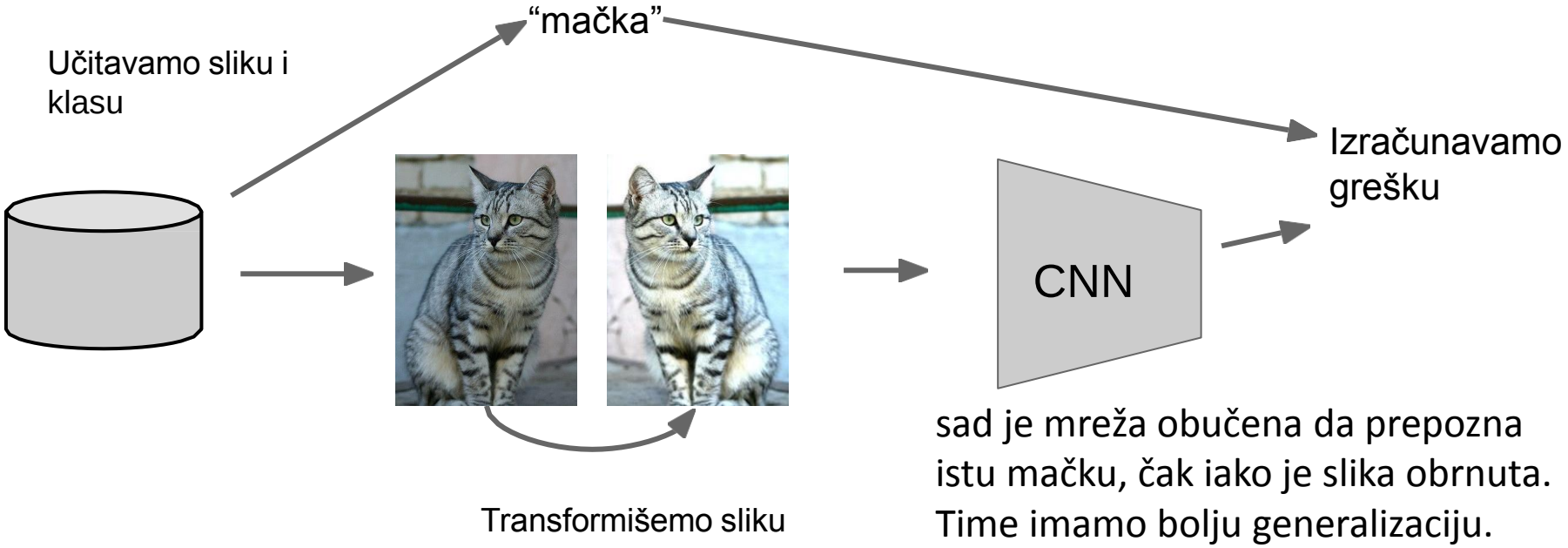
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

ovde je aktivacija nepromenjena!

Regularizacija: Obogaćivanje Skupa Podataka (*Data Augmentation*)

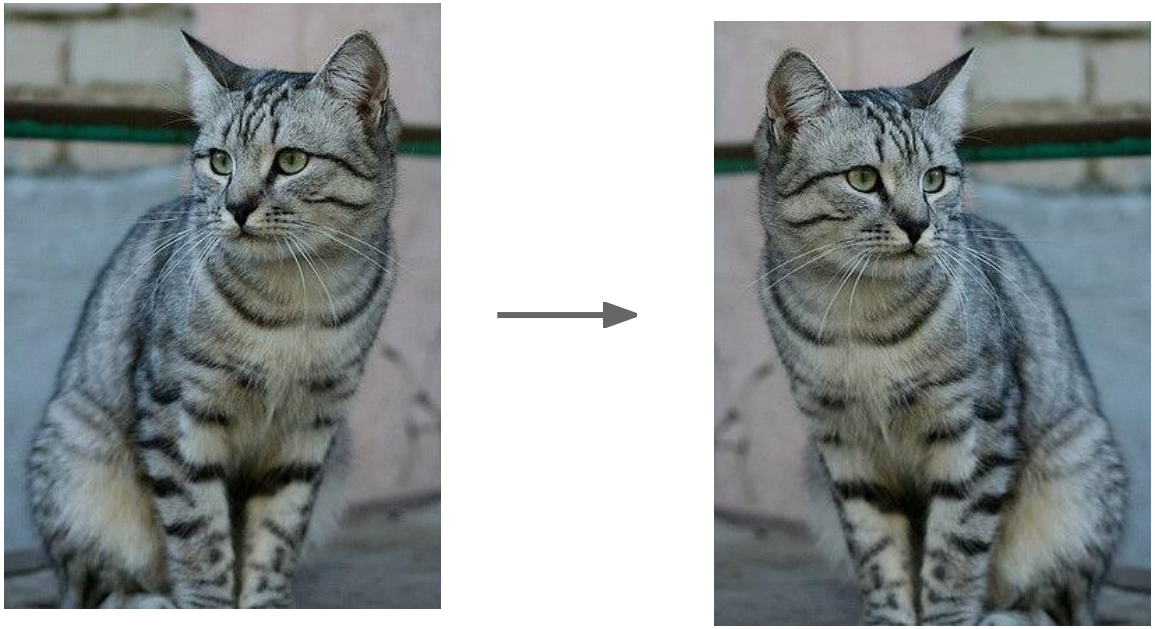


Regularizacija: Obogaćivanje Skupa Podataka (*Data Augmentation*)



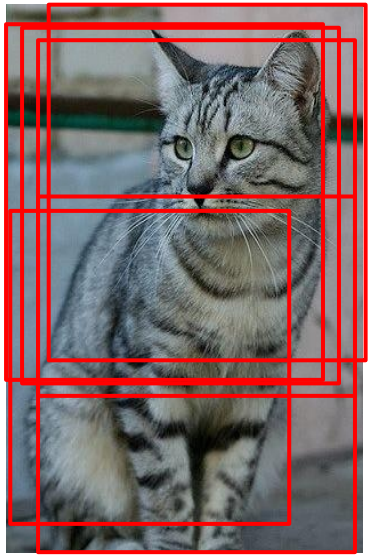
Obogaćivanje Skupa Podataka

Horizontalno Obrtanje



Obogaćivanje Skupa Podataka

Slučajno kropovanje i skaliranje



Obučavanje: Slučajno kropovanje i skaliranje

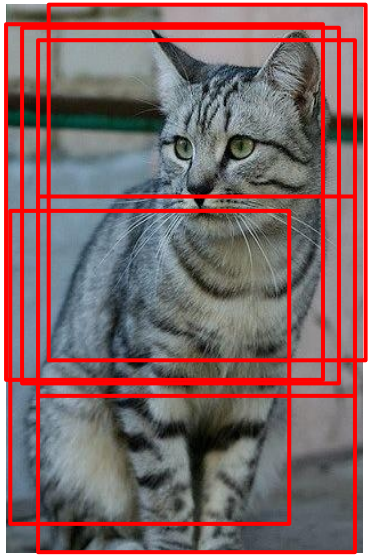
ResNet:

1. Odaberi slučajno L u rasponu [256, 480]
2. Skaliraj sliku tako da manja strana bude L
3. Sempluj slučajno deo dimenzija 224 x 224 iz originalne slike

2. i 3. su dve odvojene stvari, tako u obučavajući skup dodajemo malo reskaliranih primera jedne slike, a takođe i malo nekih njenih delova.

Obogaćivanje Skupa Podataka

Slučajno kropovanje i skaliranje



Obučavanje: Slučajno kropovanje i skaliranje

ResNet:

1. Odaberi slučajno L u rasponu [256, 480]
2. Skaliraj sliku tako da manja strana bude L
3. Sempluj slučajno deo dimenzija 224 x 224 iz slike

Primena: klasa je prosek predikcija koje mreža da za 1. i 2. dole

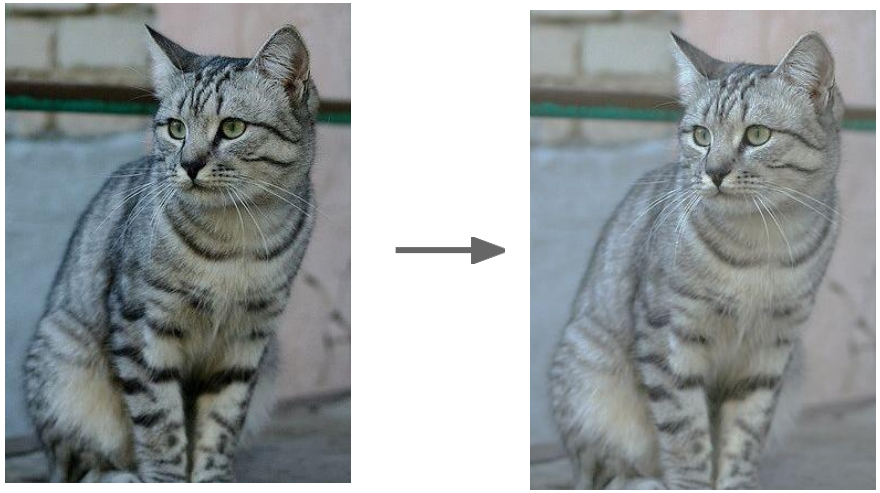
ResNet:

1. Skaliramo sliku u ovih 5 veličina: {224, 256, 384, 480, 640}
2. Za svaku veličinu uzimamo 10 kropova dim. 224 x 224: 4 ugla + center, + obrtanja

Obogaćivanje Skupa Podataka

Male promene boje

Jenostavno: Slučajno promenimo osvetljenje i kontrast



Postoje i kompleksnije metode, ali ih nećemo obrađivati na ovom kursu.

Obogaćivanje Skupa Pdataka

Nema pravila, budite kreativni sa svojim podacima i problemom na kome radite!

Neki slučajan miks:

- translaacije
- rotacije
- istezanja,
- drugih distorzija, … (zavisi šta očekujete od primera na koje ćete da primenjujete mrežu)

Učenje Transferom
Transfer Learning

"Treba nam puno podataka da obučimo Konvolutivnu Neuronsku Mrežu"

Učenje Transferom

Transfer Learning

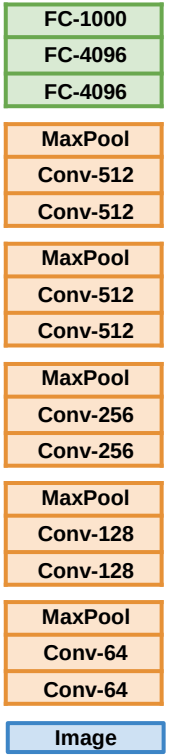
"Treba nam puno podataka da obučimo Konvolutivnu Neuronsku Mrežu"

NETAČNO

Učenje Transferom - CNN

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

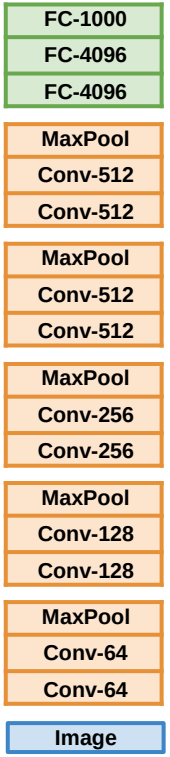
1. Obučiti na Imagenet



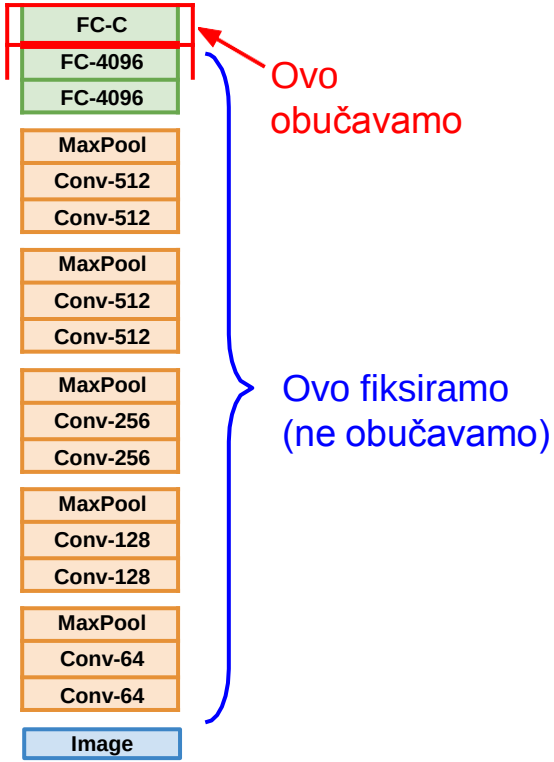
Učenje Transferom - CNN

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Obučiti na Imagenet



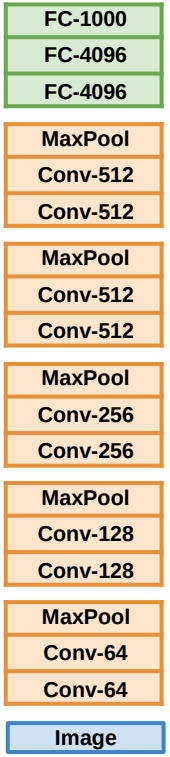
2. Mali skup podataka (C klasa)



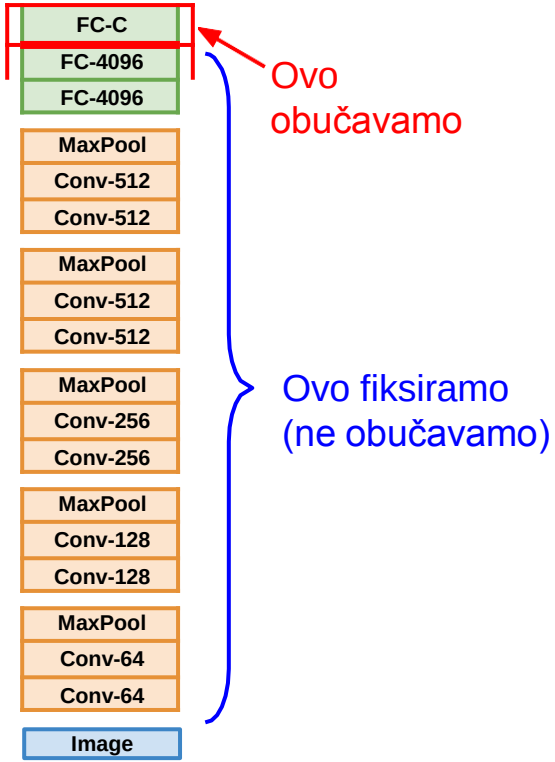
Učenje Transferom - CNN

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

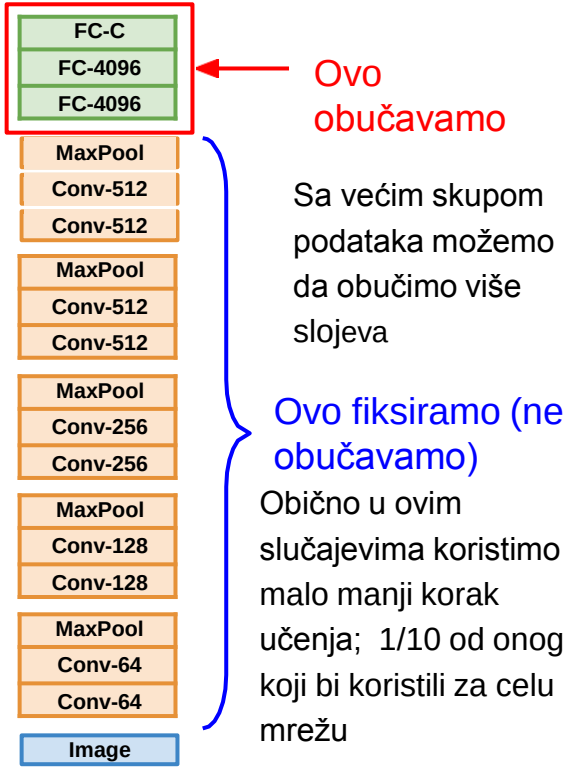
1. Obučiti na Imagenet

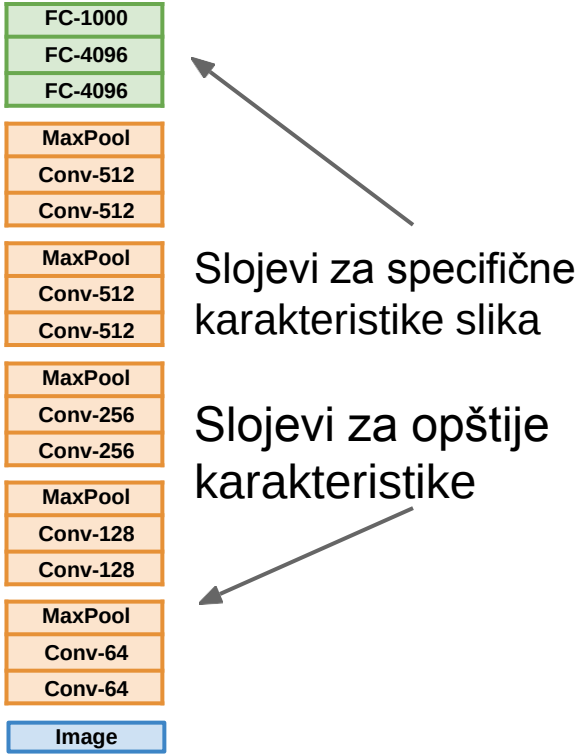


2. Mali skup podataka (C klasa)

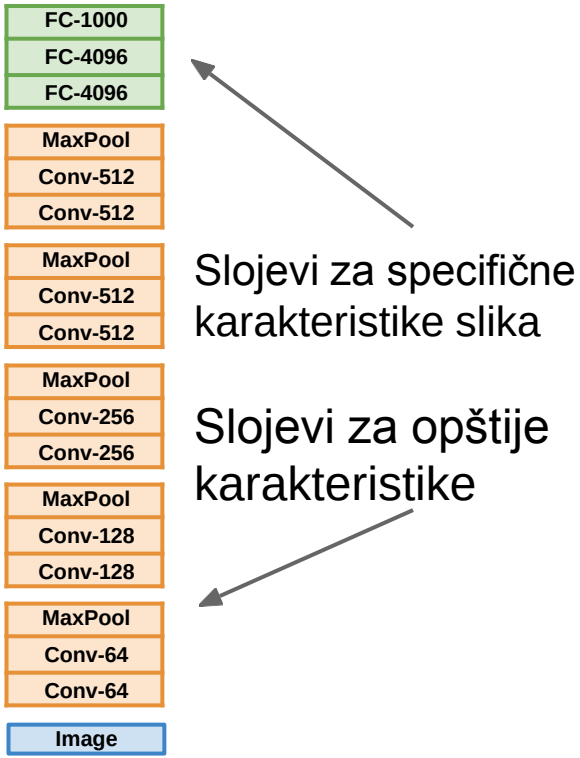


3. Malo veći skup podataka

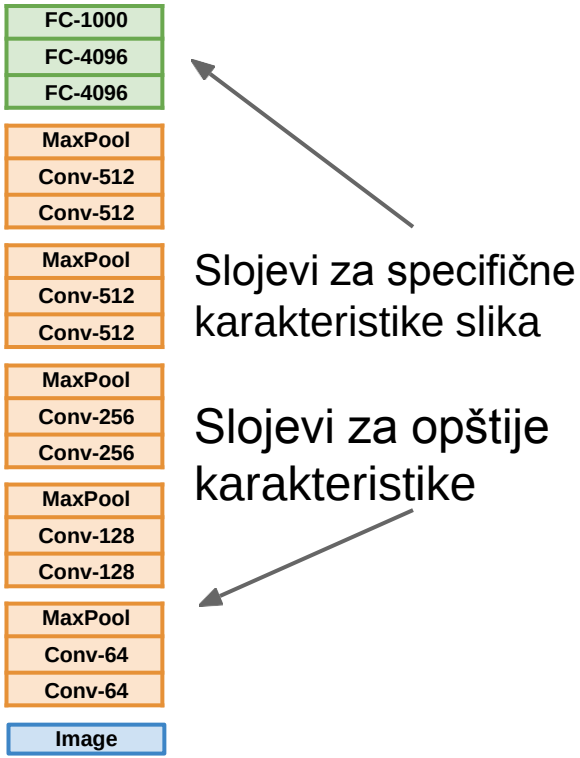




	vrlo sličan obučavajući skup	**vrlo drugačiji obučavajući skup**
mali obučavajući skup	?	?
veliki obučavajući skup	?	?



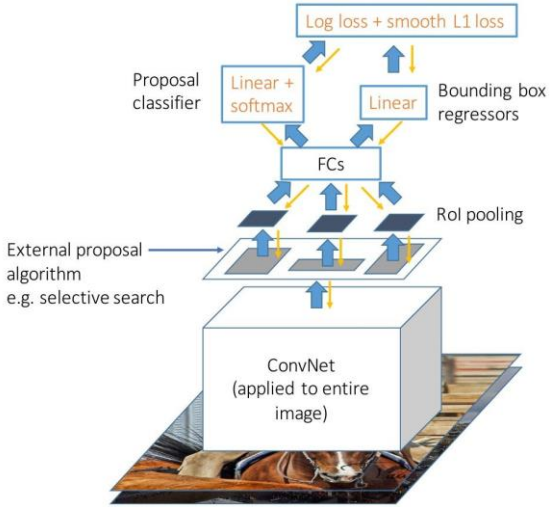
	vrlo sličan obučavajući skup	**vrlo drugačiji obučavajući skup**
mali obučavajući skup	upotrebiti linearni klasifikator umesto gornjeg sloja	?
veliki obučavajući skup	obučavati samo par slojeva na vrhu	?



	vrlo sličan obučavajući skup	**vrlo drugačiji obučavajući skup**
mali obučavajući skup	upotrebiti linearni klasifikator umesto gornjeg sloja	Ovde nema mnogo pomoći... možda ansambli lin. klas. obučenih u različitim delovima mreže
veliki obučavajući skup	obučavati samo par slojeva na vrhu	obučavati samo više slojeva na vrhu

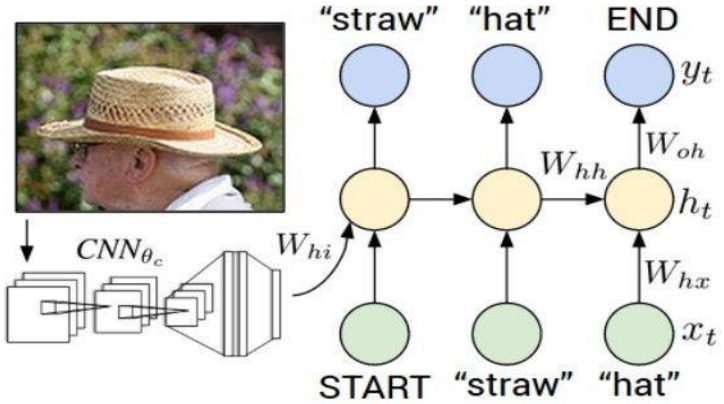
Učenje transferom kod CNN je vrlo često…
(to je više standardno, nego izuzetak)

Detekcija objekata (Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Dodeljivanje objašenja slikama:
CNN + RNN

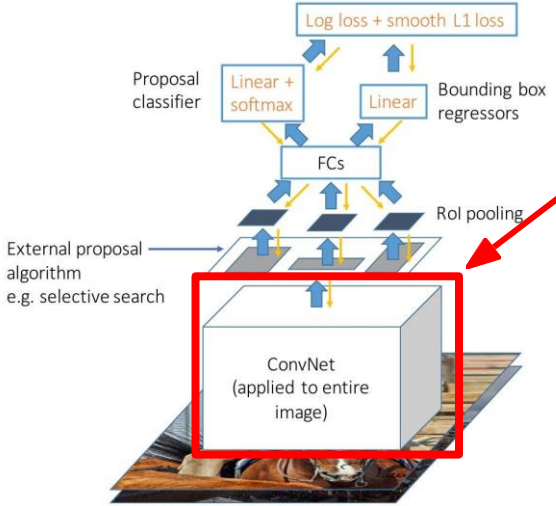


Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Učenje transferom kod CNN je vrlo često…

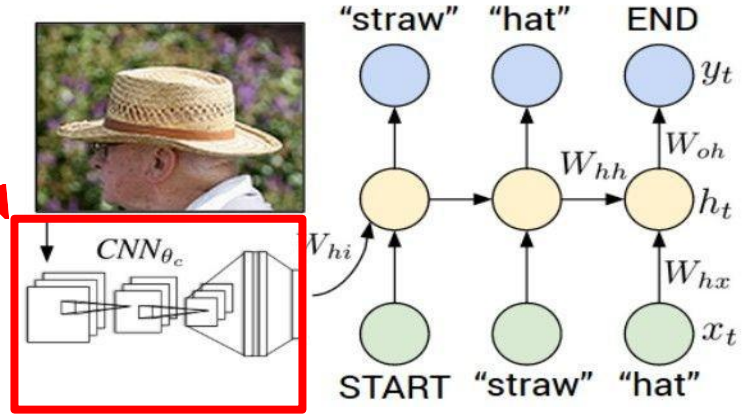
(to je više standardno, nego izuzetak)

Detekcija objekata (Fast R-CNN)



CNN ranije obučena na Imagenet

Dodeljivanje objašenja slikama: CNN + RNN



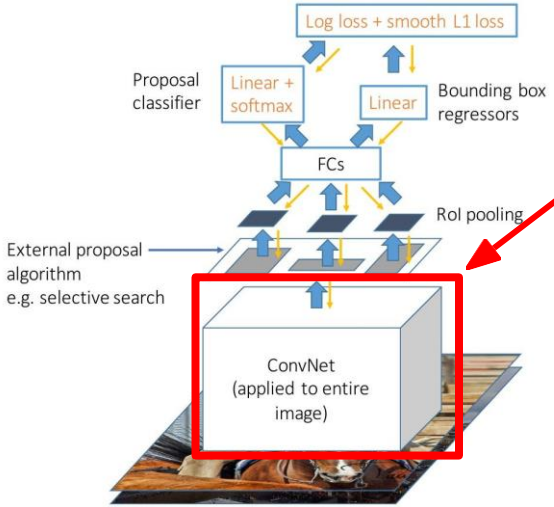
Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Učenje transferom kod CNN je vrlo često…

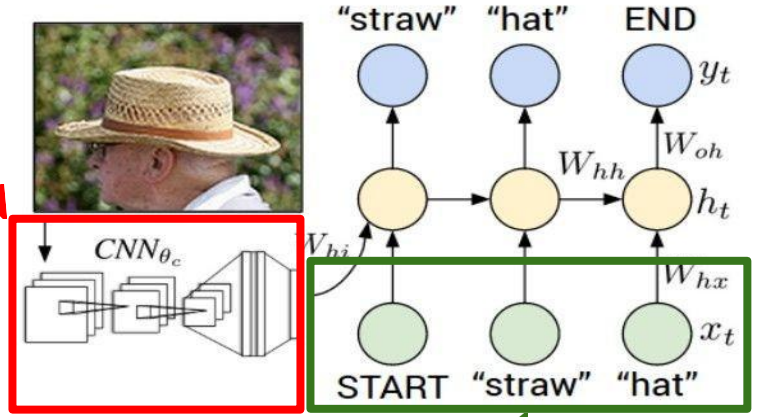
(to je više standardno, nego izuzetak)

Detekcija objekata (Fast R-CNN)



CNN ranije obučena na Imagenet

Dodeljivanje objašenja slikama: CNN + RNN



Vektor za svaku reč obučeni pomoću word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Savet:

Imate interesantan ob. skup slika, ali ima < ~1M primera?

1. Naći veliki skup podataka koji ima slične slike, obučiti veliku CNN na tom skupu
2. Uraditi učenje transferom na vašem manjem skupu

Deep learning okruženja pružaju tzv. “Model Zoo” raznih modela obučenih na velikom broju ob. skupova (*pretrained models*)

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo

TensorFlow: https://github.com/tensorflow/models

PyTorch: https://github.com/pytorch/vision

Rezime

- Optimizacija
  - Momentum, RMSProp, Adam, ...
- Regularizacija
  - Dropout, ...
- Učenje tranferom
  - Treba koristiti ako je moguće!