Neurónové siete v praxi

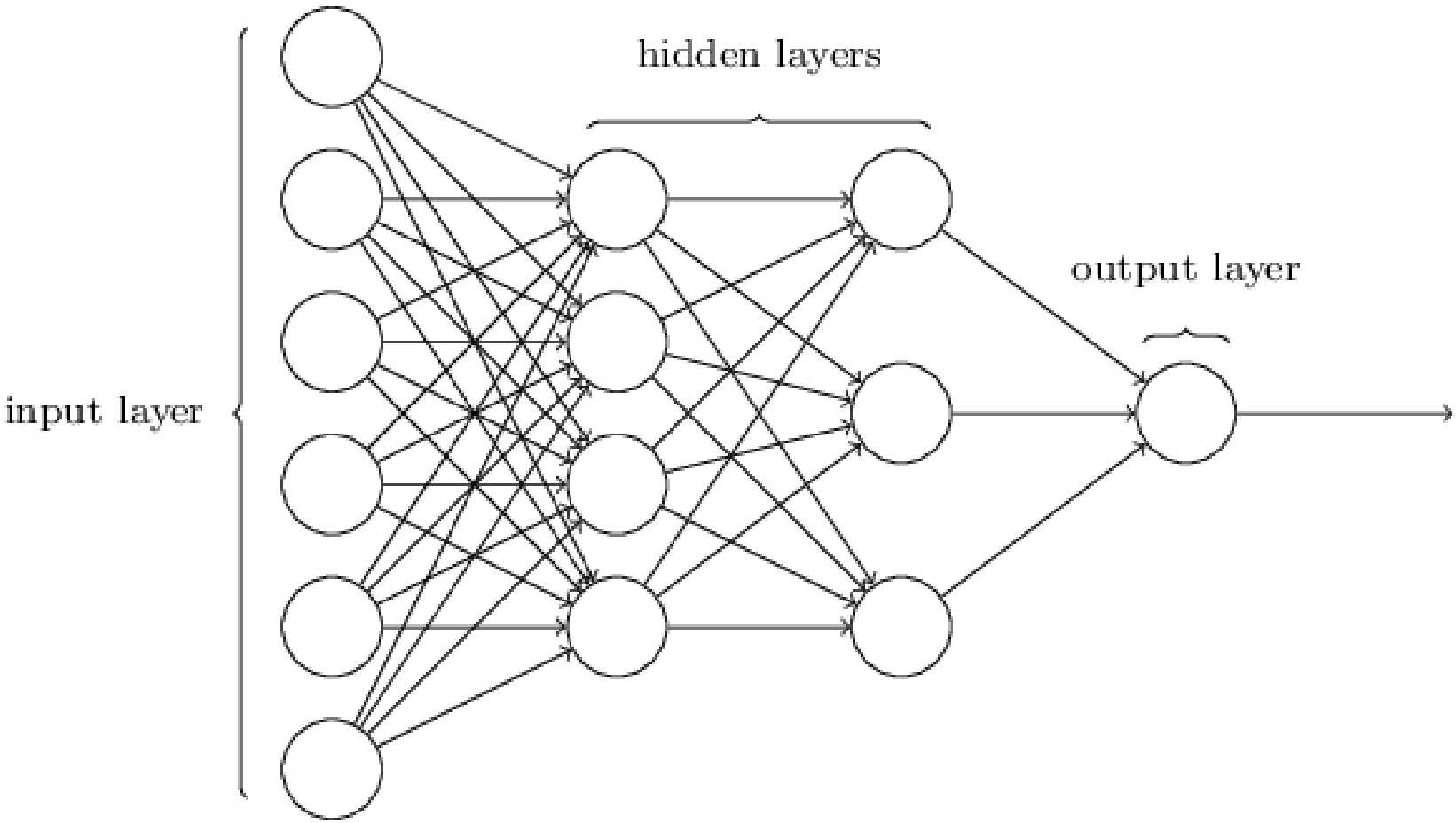
Ing. Viktor Kocur
viktor.kocur@fmph.uniba.sk

DAI FMFI UK

19.3.2018

Obsah

Fully-connected NN



Dopredný výpočet

Obecne sa aktivácia (hodnota) nevstupného vrcholu p počíta pomocou vzorca:

$$a_p = f\left(\sum_{q \in p_{in}} w_{p,q} a_q + b_p\right) = f\left(z_p\right) \tag{1}$$

- a_p je aktivácia daného vrcholu
- $w_{p,q}$ je váha vrcholu q pre vrchol p
- b_p je prah vrcholu p
- f je aktivačná funckia
- z_p je zjednodušený zápis vstupu aktivačnej funkcie

Dopredný výpočet - FCN

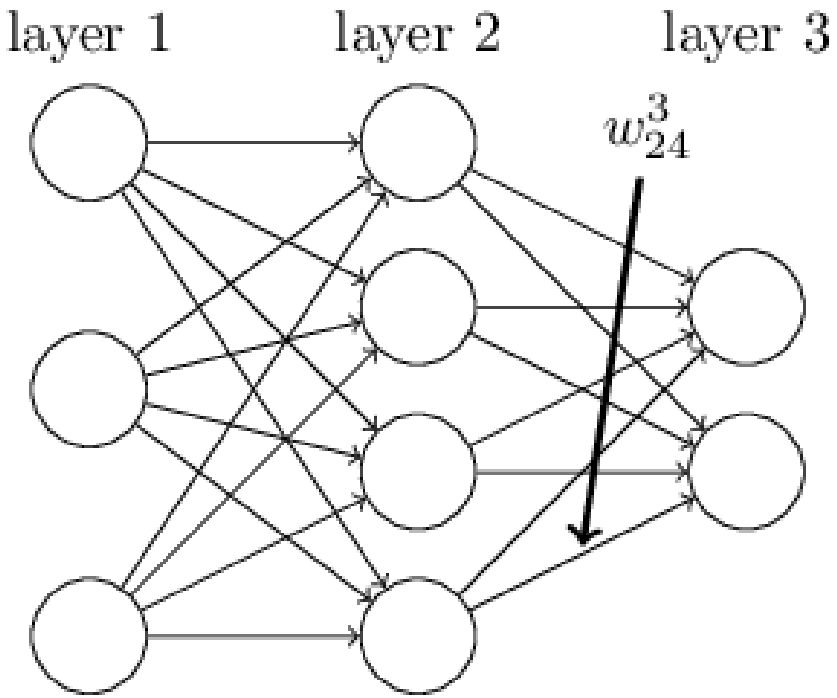
Pre FCN siete môžeme tento vzťah formalizovať krajšie:

$$a_j^l = f\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = f\left(z_j^l\right) \tag{2}$$

$$a^l = f\left(w^l a^{l-1} + b^l\right) = f\left(z^l\right) \tag{3}$$

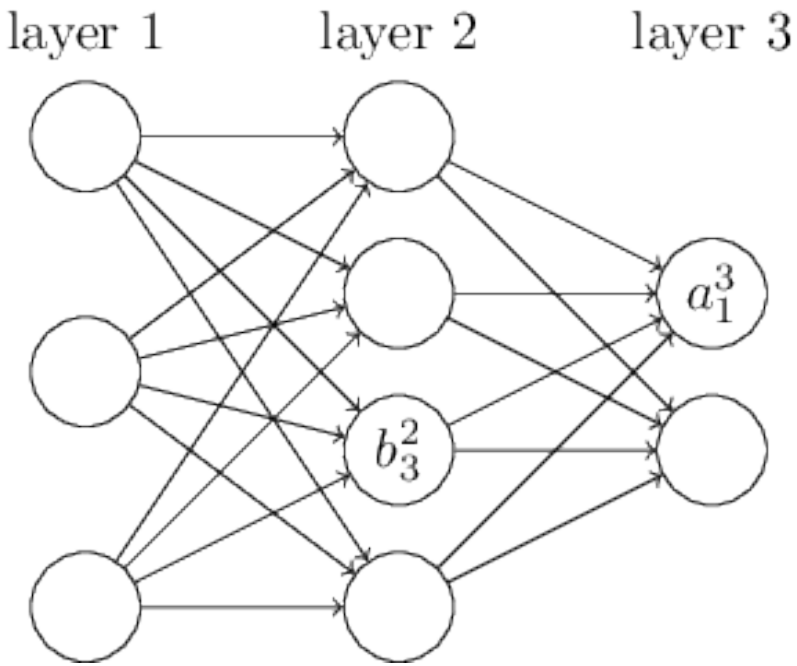
- a^l je vektor aktivácií v l-tej vrstve
- w^l je váhová matica l-tej vrstvy (size(l) $\times$ size$(l-1)$)
- b^l je vektor prahov v l-tej vrstve
- f je aktivačná funckia

FCN - váhy



$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

FCN - aktivácia a bias



$$a_j^l = f\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Aktivačné funkcie

- Sigmoid: $f(z) = \frac{1}{1+e^{-z}}$
- Tanh: $f(z) = tanh(z)$
- ReLU: $f(z) = max(x, 0)$
- SoftPlus: $f(z) = ln(1 + e^z)$
- LeakyReLU: $f(z) = max(x, ax), a \leq 1$

Výsup

Ako interpretovať aktiváciu poslednej vrstvy (L)? Pri klasifikačnej výberu kategórie k z množiny K možných napr:

- Najlepší odhad: $k = \underset{i \in K}{\operatorname{argmax}} \left(a_i^L(x) \right)$
- Softmax: $P(k|x) = \frac{e^{z_k^L}(x)}{\sum_{i \in K} e^{z_i^L}(x)}$

Cenová funkcia

Trénovanie sieťe je vlastne optimalizačná úloha. Chceme nájsť parametre P, tak aby sme na trénovacích dátach dosiahli čo najmenšiu cenu (chybu) C. Na to však musíme definovať cenu na vstupoch $\{x_1, x_2, ..., x_n\}$ pri požadovaných výstupoch $\{y_1, y_2, ..., y_n\}$. Možností je opäť viac.

- MSE: $C = \frac{1}{n} \sum_x \left\| a^L(x) - y \right\|_2^2$
- CE: $C = -\frac{1}{n} \sum_{x,i} y_i ln\left(a_i^L(x)\right) + (1 - y_i)\, ln\left(1 - a_i^L(x)\right)$
- Softmax: $C = -\frac{1}{n} \sum_x ln(z_y^L(x))$

Ako dospejeme k parametrom s optimálnou cenou?

Použijeme gradientný zostup pre všetky parametre!

$$p := p - \eta \frac{\partial C}{\partial p}, \tag{4}$$

kde p je ľubovolný parameter nášho modelu, C je cenová funkcia a η je hyperparameter rýchlosti zostupu.

Budeme rátať gradient na všetkých trenovacích dátach?

Nie!

Vždy použijeme cenu len pre podmnožinu trénovacej množiny, tzv. minibatch.

Budeme rátať gradient na všetkých trenovacích dátach?

Nie!

Vždy použijeme cenu len pre podmnožinu trénovacej množiny, tzv. minibatch.

Stochastic gradient descent

Po jednom kroku zostupu načítame nový minibatch z trenovacích dát. Postupne takto pokryjeme celú trénovaciu množinu. To sa volá jedna epocha. Pred začatím ďalšej epochy trénovaciu množinu premiešame. Premiešanie trénovacej množiny je náhodné, preto názov stochastic gradient descent.

Ako zistíme parciálne derivácie pre všetky parametre?

Pre FCN je možné určiť parciálne derivácie podľa parametrov poslednej vrstvy postupne. Určimé si pomocný výraz δ_j^l.

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{5}$$

Pre MSE cenovú funkciu platí:

$$\delta^L = \left(a^L - y\right) \odot f'\left(z^L\right) \tag{6}$$

Pre CE cenovú funkciu:

$$\delta^L = \left(a^L - y\right) \tag{7}$$

Propagujeme chybu smerom späť!

Chyba vrstvy l závisí od vrstvy $l+1$!

$$\delta^l = \left(\left(w^{l+1} \right)^T \delta^{l+1} \right) \odot f' \left(z^l \right) \tag{8}$$

Dôkaz

$$z_j^{l+1} = w_{jk}^{l+1} f\left(z_k^l\right) + b_j^{l+1} \tag{9}$$

$$\frac{\partial z_j^{l+1}}{\partial z_k^l} = w_{jk}^{l+1} f'\left(z_k^l\right) \tag{10}$$

$$\delta_k^l = \frac{\partial C}{\partial z_k^l} = \sum_j \frac{\partial C}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_k^l} = \sum_j \delta_j^{l+1} w_{jk}^l f'\left(z_k^l\right) \tag{11}$$

Parciálne derivácie

Pre parametre w a b platí:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{12}$$

$$\frac{\partial C}{\partial w_{j,k}^l} = a_k^{l-1} \delta_j^l \tag{13}$$

Dôkazy

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \tag{14}$$

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} = \sum_k \delta_k^l \delta_{jk}^{kr} = \delta_j^l \tag{15}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_p \frac{\partial C}{\partial z_p^l} \frac{\partial z_p^l}{\partial w_{jk}^l} = \sum_p \delta_p^l \delta_{pj}^{kr} a_k^{l-1} = \delta_j^l a_k^{l-1} \tag{16}$$

Backprop algoritmus

Stačí to dať dohromady a máme algoritmus jedného kroku SGD.

- Realizujeme dopredný výpočet - pamätáme si z^l a a^l.
- Vypočitame δ^L podľa (6), alebo (7).
- Spätne propagujeme chybu δ^l pomocou (8).
- Vypočítame parciálne derivácie podľa (12) a (13).
- Opakujeme pre celý minibatch, derivácie spriemerujeme a updatneme parametre podľa gradientného zostupu.

Backprop algoritmus

Stačí to dať dohromady a máme algoritmus jedného kroku SGD.

- Realizujeme dopredný výpočet - pamätáme si z^l a a^l.
- Vypočitame δ^L podľa (6), alebo (7).
- Spätne propagujeme chybu δ^l pomocou (8).
- Vypočítame parciálne derivácie podľa (12) a (13).
- Opakujeme pre celý minibatch, derivácie spriemerujeme a updatneme parametre podľa gradientného zostupu.

Pozor!

Pred trenovaním je nutné inicializovať premenné! Váhy s nejakým rozdelením, prahy stačia na jednu hodnotu.

Validačná množina

Stačí nám trenovacia množina?

Náš model sa snáď úspešne učí na trénovacích dátach. Môžeme pozorovať že cena klesá a presnosť na trénovacej množine stúpa. Máme zaručené, že naš model bude dobre generalizovať svoje znalosti?

Validačná množina

Stačí nám trenovacia množina?

Náš model sa snáď úspešne učí na trénovacích dátach. Môžeme pozorovať že cena klesá a presnosť na trénovacej množine stúpa. Máme zaručené, že naš model bude dobre generalizovať svoje znalosti?

Validačná množina

Časť trenovacích dát si odložíme. Na nich budeme pozorovať ako sa model zlepšuje vždy po niekoľkých krokoch backprop algoritmu. Ak cena na trénovacej množine klesá, ale presnosť na validačnej množine nestúpa došlo k overfittingu, tzv. high variance.

Ako určit hyperparametre

Správne hyperparametre

Ak sa náš model učí príliš pomaly, alebo nedostatočne, prípadne došlo k overfittingu, tak musíme upraviť hyperparametre. Momentálne to sú η, veľkosť minibatch a architektúra siete.

Ako určit hyperparametre

Správne hyperparametre

Ak sa náš model učí príliš pomaly, alebo nedostatočne, prípadne došlo k overfittingu, tak musíme upraviť hyperparametre. Momentálne to sú η, veľkosť minibatch a architektúra siete.

Train/Val

Na validačnej množine (zväčša ručne) určujeme hyperparametre a na trénovacej množine určujeme parametre modelu.

Ako určit hyperparametre

Správne hyperparametre

Ak sa náš model učí príliš pomaly, alebo nedostatočne, prípadne došlo k overfittingu, tak musíme upraviť hyperparametre. Momentálne to sú η, veľkosť minibatch a architektúra siete.

Train/Val

Na validačnej množine (zväčša ručne) určujeme hyperparametre a na trénovacej množine určujeme parametre modelu.

A čo testovacia množina?

Testovacej množiny sa chytáme až úplne na konci!

Čo ak náš model negeneralizuje dostatočne?

Viac dát

Pri nedostatočnej schopnosti modelu generalizovať svoje vedomosti je možné zlepšiť situáciu zväčšením trénovacej množiny.

Čo ak náš model negeneralizuje dostatočne?

Viac dát

Pri nedostatočnej schopnosti modelu generalizovať svoje vedomosti je možné zlepšiť situáciu zväčšením trénovacej množiny.

Máme iné možnosti?

Môžeme si vytvoriť syntetické dáta, alebo môžeme obmedziť kapacitu modelu pomocou regularizácie.

Cenová funkcia s regularizáciou

Jeden spôsob ako zabrániť overfittingu je upraviť cenovú funkciu, tak aby sme preferovali modely s menšiou variance. Vytvoríme tak novú cenovú funkciu.

$$C = C_0 + \lambda R(P), \tag{17}$$

v ktorej vystupuje naša pôvodná cenová funkcia C_0, tzv. regularizačný člen R, ktorý závisí len od hodnôt parametrov a hyperparameter λ, ktorý určuje silu regularizačného člena.

L2 Regularizácia

Najčastejšie sa používa L2 regularizácia. Pre ktorú platí

$$R_{L2} = \frac{1}{2n} \sum_{w,j,k} w_{j,k}^2 \tag{18}$$

Parciálna derivácia ceny podľa w sa tak zmení na:

$$\frac{\partial C}{\partial w_{j,k}^l} = \frac{\partial C_0}{\partial w_{j,k}^l} + \lambda \frac{\partial R}{\partial w_{j,k}^l} = a_k^{l-1} \delta_j^l + \frac{\lambda}{n} w_{j,k}^l \tag{19}$$

L2 Regularizácia

Ak si teda rozpíšeme update pravidlá dostaneme:

$$w^l := w^l - \eta \frac{\partial C_0}{\partial w^l} - \frac{\eta\lambda}{n} w^l \tag{20}$$

$$w^l := \left(1 - \frac{\eta\lambda}{n}\right) w^l - \eta \frac{\partial C_0}{\partial w^l} \tag{21}$$

A pre prahy jednoducho:

$$\frac{\partial C}{\partial b^l} = b^l - \eta \frac{\partial C_0}{\partial b^l} \tag{22}$$

L1 a Dropout

L1

L1 regularizácia je podbná L2. Má iný tvar: $R = \frac{1}{n} \sum_{w,j,k} |w_{j,k}|$

Dropout

Pri dropoute sa niektoré vrcholy siete počas učenia deaktivujú s pravdepodobnosťou p. To zabráni tzv. koadaptácií príznakov. Je nutné si dať pozor a pri normálnom behu sieťe prenásobiť príslušné váhy parametrom p.

Ukážka - MNIST

Matlab

Ukážka - MNIST

Tensorflow

Motivácia

Parametrické peklo

Ak by sme chceli spracovávať obraz pomocou viacerých FC vrstiev, tak by nám čoskoro narástlo množstvo parametrov nad únosnú mieru.

Motivácia

Parametrické peklo

Ak by sme chceli spracovávať obraz pomocou viacerých FC vrstiev, tak by nám čoskoro narástlo množstvo parametrov nad únosnú mieru.

Zdielanie parametrov

Počet parametrov môžeme redukovať tak, že niektoré z nich spojíme do jednej skupiny. Všetky parametre z tejto skupiny teda budú mať rovnakú hodnotu.

Konvolúcia

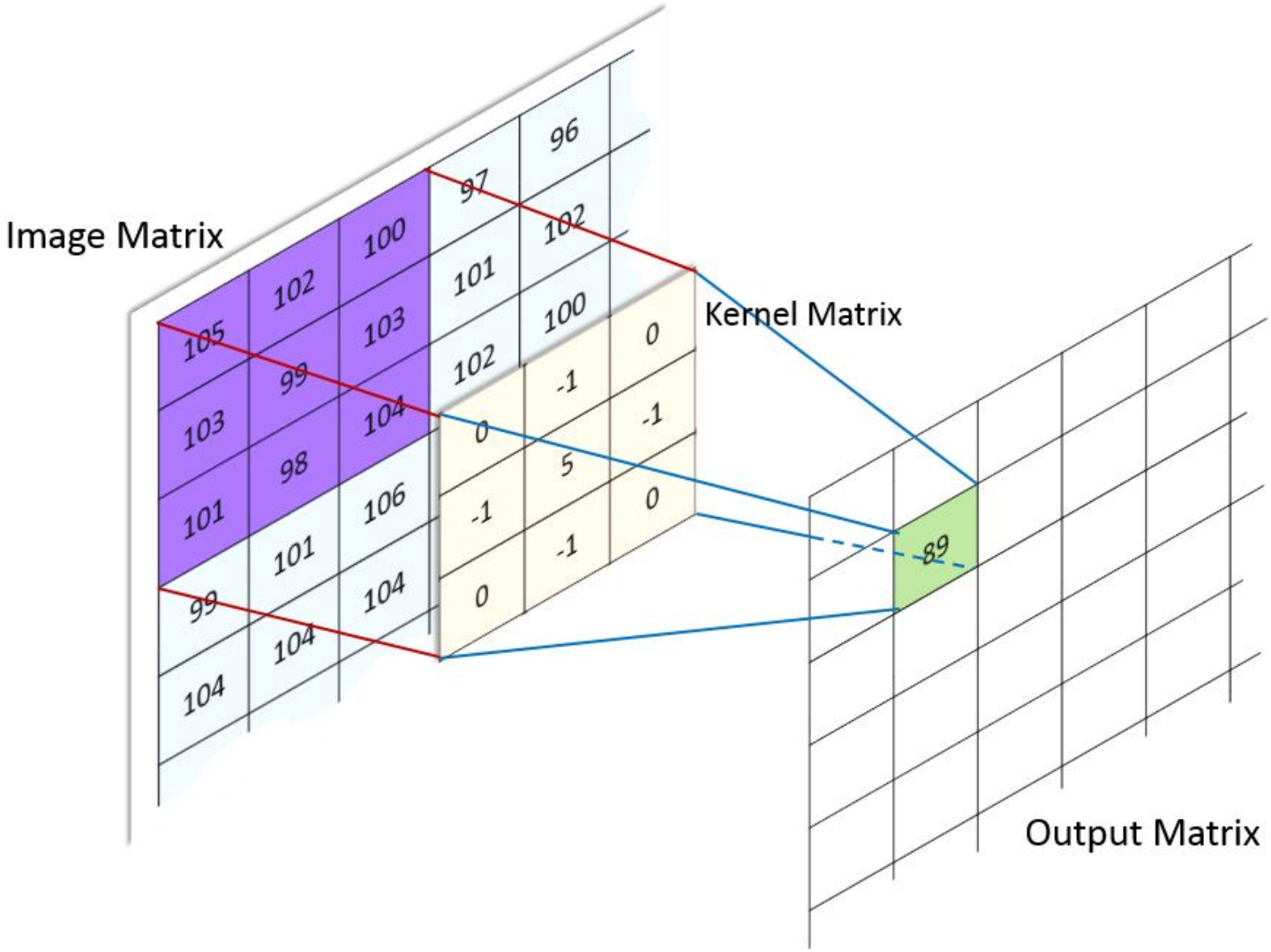


Image Matrix
105 102 100 97 96
103 99 103 101 102
101 98 104 102 100
99 101 106 0 -1 0
104 104 104 -1 5 -1
0 -1 0
Kernel Matrix
89
Output Matrix

Konvolučné vrstvy

Filtre - Kernels

V každej kovolučnej vrstve máme P_{out} konvolučných filtrov veľkosti $F \times F \times P_{in}$.

Aplikácia

Na vstup $N_{in} \times N_{in} \times P_{in}$ aplikujeme filtre $F \times F \times P_{in}$ so 'stride-om' S. Výsledkom bude výstup $N_{out} \times N_{out} \times P_{out}$, kde $N_{out} = \frac{N_{in} - F}{S} + 1$. Každý filter má navyše aj svôj bias, ktorý k jednotlivých konvolúciam pripočíta.

Konvolučné vrstvy

Padding

V prípade že používame padding, tak sa vzorec pre veľkosť nového obrázka zmení na $N_{out} = \frac{N_{in} - F + 2*P}{S} + 1$, kde P označuje, koľko pixelov pridáme na kraj obrázka ako padding.

1×1 Konvolúcia

Pre konvolučné neurónove siete je zmysluplné aplikovať aj filtre s $F = 1$ a $P_{out} \leq P_{in}$. Takýto filter zredukuje počet kanálov.

Pooling vrstvy

Motivácia

Chceme postupne zmenšovať veľkosť rozmeru N pre vrstvy. Mohli by sme používať veľký stride v kovolučných vrstvách, to ale nieje vždy vhodné.

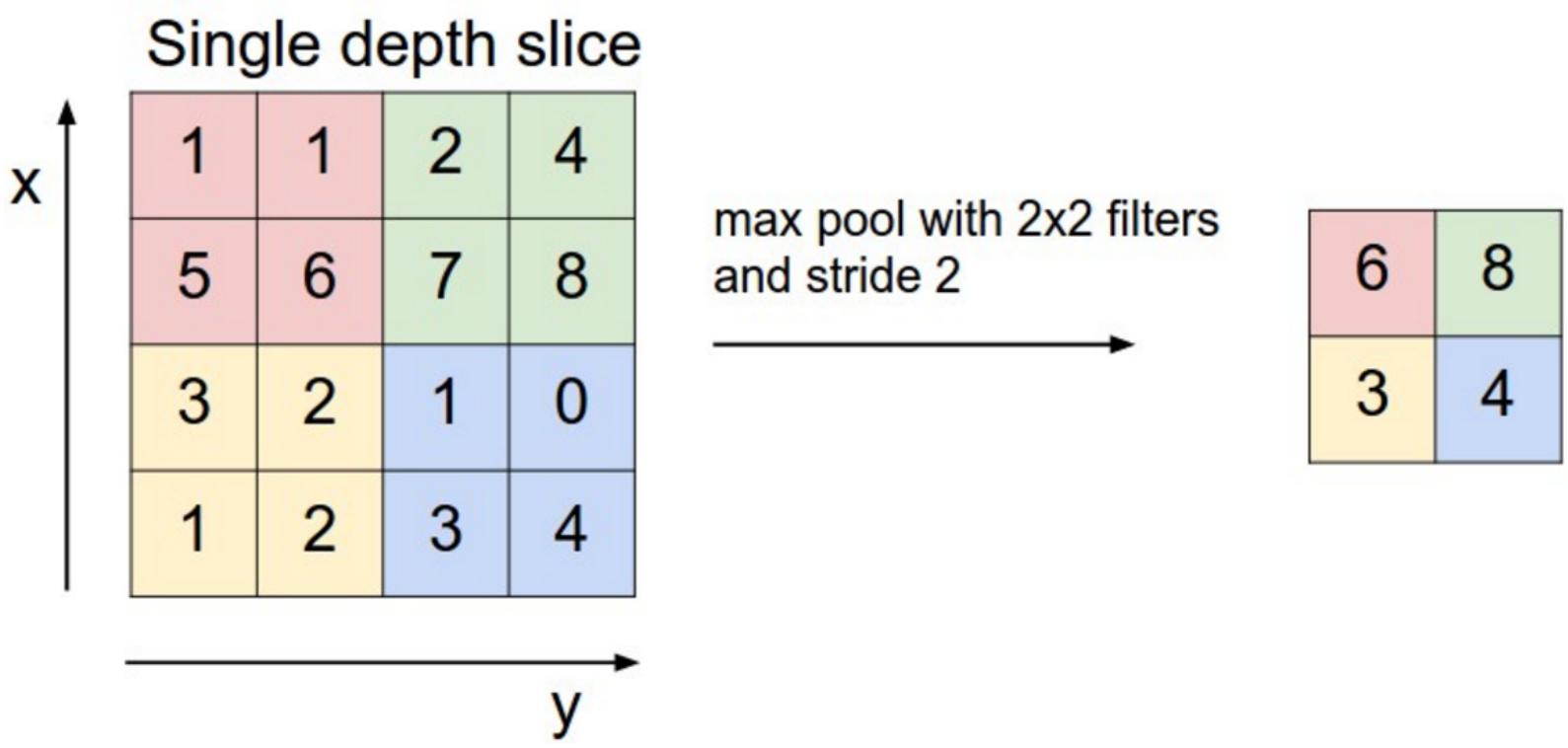
Max pooling

Najčastejšie používame tzv. 2×2 max pooling. so stride-om 2. Rozdelíme obraz na štvorčeky 2×2 a z nich vyberieme najväčšiu hodnotu.

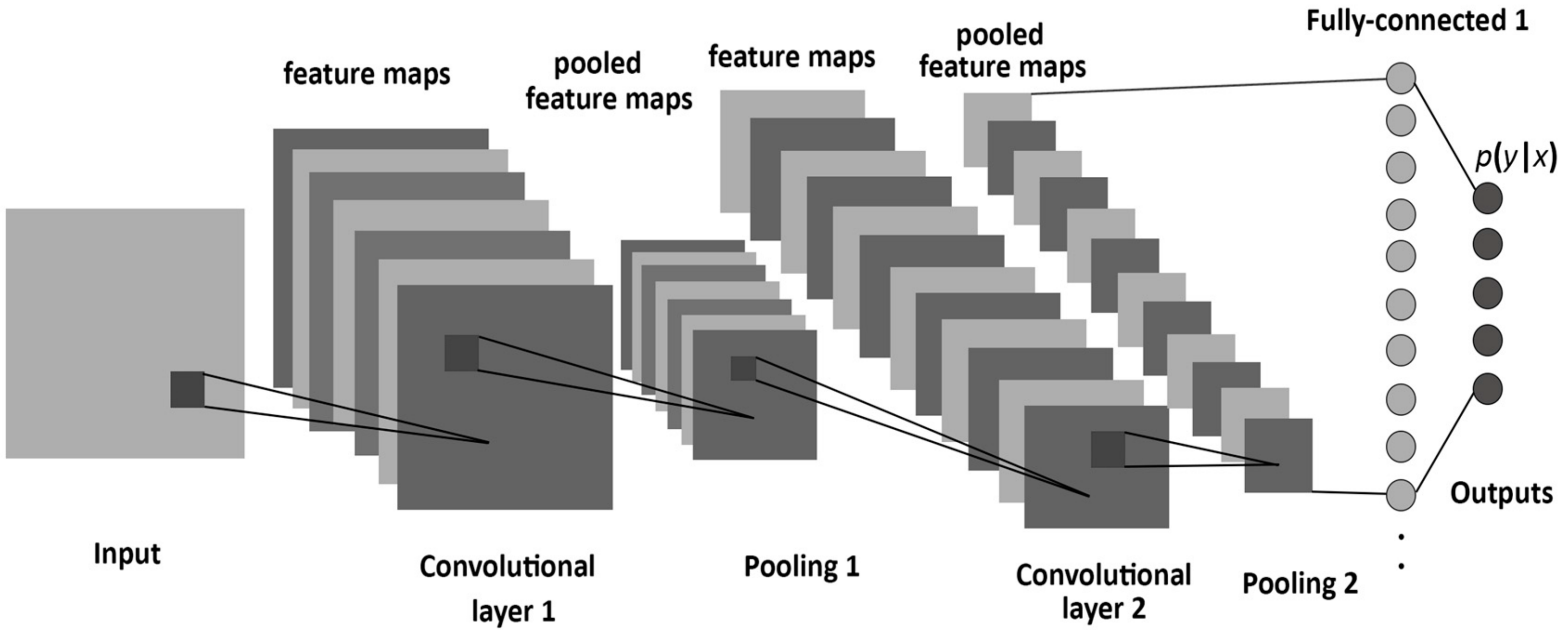
Mean pooling

Aj to je možné, ale nieje to v praxi často využívané.

Max-Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Kompletná architektúra



BatchNormalization - Motivácia

Ako veľmi záleží na inicializácii?

Niekedy je priestor vhodných distribúcií pre inicializáciu siete veľmi malý. Trénovanie, tak vôbec nemusí byť úspešné!

BatchNormalization

Po každej FC, alebo Conv vrstve normalizujeme aktivácie!

BatchNormalization - Motivácia

Ako veľmi záleží na inicializácii?

Niekedy je priestor vhodných distribúcií pre inicializáciu siete veľmi malý. Trénovanie, tak vôbec nemusí byť úspešné!

BatchNormalization

Po každej FC, alebo Conv vrstve normalizujeme aktivácie!

Je to dobré?

Čo keď ďalšia vrstva nechce mať na vstupe normálnu distribúciu?

BatchNormalization - Motivácia

Ako veľmi záleží na inicializácii?

Niekedy je priestor vhodných distribúcií pre inicializáciu siete veľmi malý. Trénovanie, tak vôbec nemusí byť úspešné!

BatchNormalization

Po každej FC, alebo Conv vrstve normalizujeme aktivácie!

Je to dobré?

Čo keď ďalšia vrstva nechce mať na vstupe normálnu distribúciu?

Parametre

Sieť sa naučí ako po normalizácií aktivácie upraviť aby dávali najlepší výsledok!

BatchNormalization - Ako to funguje?

Pre vstup x chceme výstup y.

$$\hat{x} = \frac{x - \bar{x}}{\sqrt{\sigma^2(x) + \epsilon}}, \tag{23}$$

kde priemerujeme cez minibatch.

$$y = \gamma\hat{x} + \beta, \tag{24}$$

kde γ a β sú naučiteľné parametre.

BatchNormalization

Môžeme sa naučiť úplne ignorovať normalizáciu?

Ak $\gamma = \sigma(x)$ a $\beta = \bar{x}$, tak sme vlastne nič nenormalizovali!

Čo keď netrénujeme?

Sieť si musí pamatať nejakú hodnotu pre $\bar{x}$ a $\sigma(x)$ a tú použiť ak máme iba jeden vstup (nemáme minibatch). Tú môžeme naakumulovať počas trénovania, alebo po trénovaní zbehneme celú sieť na všetky prvky trénovacej množiny a z aktivácií vypočítame tieto hodnoty.