Trénování modelů

Pokyny k online schůzce

1. Používejte nejlépe desktopovou aplikaci Teams.

2. Pokud nekladete dotaz nebo se neúčastníte diskuze, mějte prosím vypnutý mikrofon.

3. Pokud jen trochu můžete, mějte puštěnou kameru. Vás výraz tváře pomůže vyučujícímu :)

4. Jak můžete položit dotaz:
   a. Zapněte si mikrofon a rovnou se zeptejte.
      nebo:
   b. Napište dotaz do chatu. nebo:
   c. Použijte tlačítko zvednout ruku.
      (Po vyvolání ruku sundejte)

Agenda

- Gradient Descent
- Regularizace
- Logistická regrese
- Softmax regrese

Machine Learning jako blackbox

- ukázali jsme si funkční lineární regresi, rozhodovací stromy nebo SGD klasifikátor aniž bychom pořádně věděli, co se děje uvnitř
  - v řadě případů je to dostačující
- hlubší pochopení, jak modely fungují, však může být klíčové pro dosažení lepších výsledků
  - např. volbou správných hyperparametrů
- vytváření a úspěšné používání neuronových sítí bude vyžadovat pochopení vybraných témat
  - Gradient Descent, regulariazace apod., aktivační funkce, viz dále

Zpátky k lineární regresi

- dva způsoby jak najít řešení:
  - uzavřený tvar (“closed form”) řešení
  - iterativní optimalizační metoda

- uzavřený tvar řešení
  - rovnou dokážeme spočítat nejlepší možné parametry modelu
    - tj. takové parametry, které vedou k nejmenší hodnotě nákladové funkce, typicky RMSE, na trénovacích datech

- iterativní optimalizační metoda Gradient Descent
  - postupně upravuje hodnoty parametrů tak, aby se minimalizovala nákladová funkce
  - po čase bude konvergovat ke stejnému řešení jako v případě uzavřeného tvaru
  - klíčový přístup pro neuronové sítě i další algoritmy

Obecný tvar lineární regrese

- obecný tvar lineární regrese:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- predikce je tedy vytvořena jako vážená suma vstupních příznaků plus konstanta
- můžeme zapsat také ve vektorové formě takto:

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

tedy jako skalární součin vektoru parametrů θ a vektoru příznaků x, nebo v maticovém zápisu:

$$\hat{y} = \boldsymbol{\theta}^{\mathbf{T}} \mathbf{x}$$

kde θ^T je transponovaná matice (resp. řádkový vektor na místo sloupcového vektoru)

Uzavřená forma řešení lineární regrese

- vektor parametrů můžeme získat přímým výpočtem dle tohoto vzorce

$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \quad \mathbf{X}^T \quad \mathbf{y}$$

- v Pythonu/Numpy bychom mohli spočítat takto:

```
theta_best = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```

- výpočetní náročnost $O(n^{2.4})$ až $O(n^3)$ podle implementace
  - pokud zdvojnásobíme počet příznaků, zvýší se výpočetní čas až osmkrát
  - pokud máme hodně příznaků (např. 100k), může být toto řešení už nepoužitelné
  - toto se týká příznaků - z hlediska počtu instancí (příkladů) je náročnost lineární, takže si poradí i s velkým datasetem (musí se však celý vejít najednou to paměti)

Gradient Descent

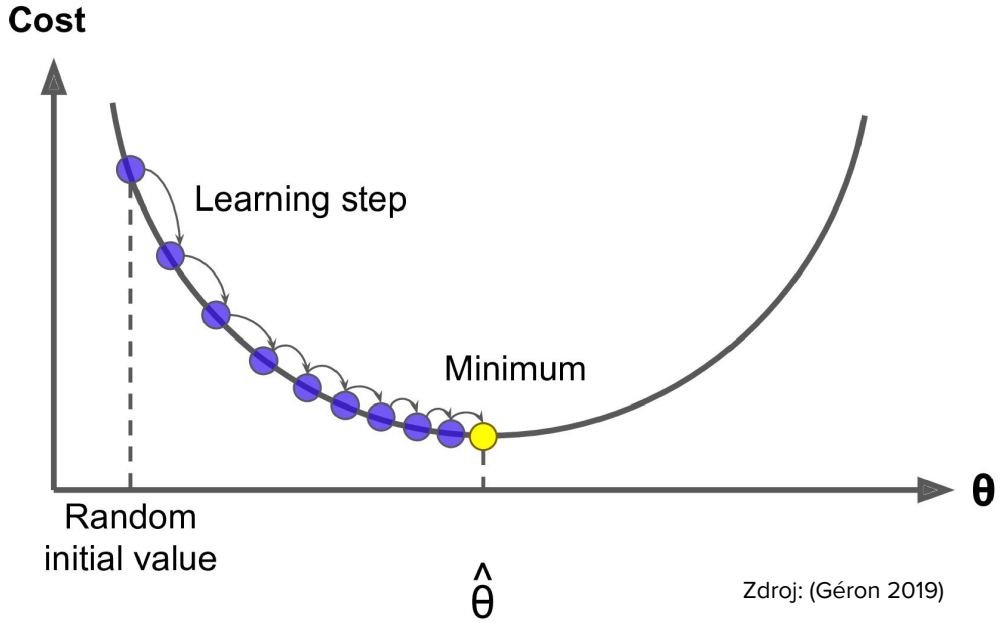
- zcela jiný přístup k řešení
- vhodný i pro lineární regresi, pokud máme hodně příznaků nebo pokud je dataset příliš velký na to, aby se vešel najednou do paměti

- základní myšlenka: iterativně upravovat hodnoty parametrů modelu tak, aby se postupně minimalizovala nákladová funkce

- pro představu: Jste na horách a chcete se dostat do údolí. Je ale absolutní mlha, takže jen cítíte sklon terénu přímo pod nohama. Dobrá strategie bude jít tím směrem, kde je zrovna sklon dolů největší.
    - přesně toto dělá metoda Gradient Descent

Gradient Descent

- princip: Metoda zjišťuje gradient nákladové funkce podle parametrů θ a upravuje jejich hodnoty ve směru klesajícího gradientu. Jakmile je gradient nulový, dosáhli jsme minima.

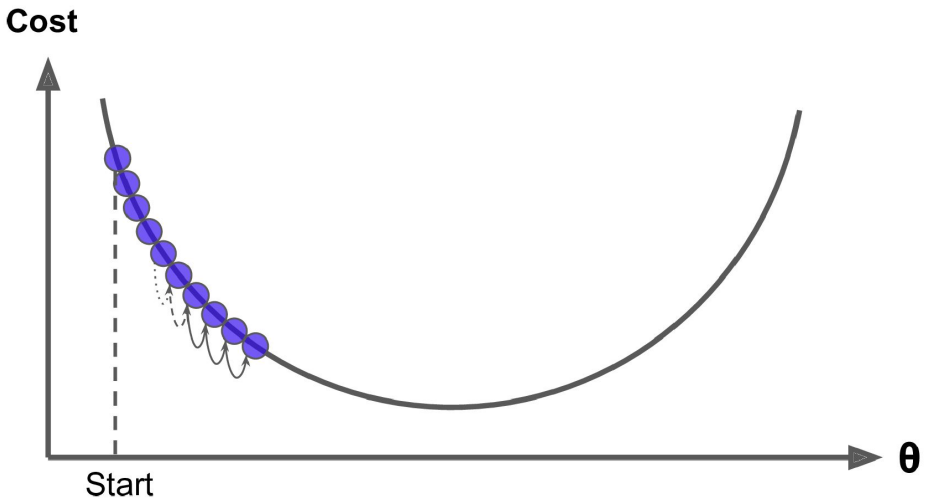


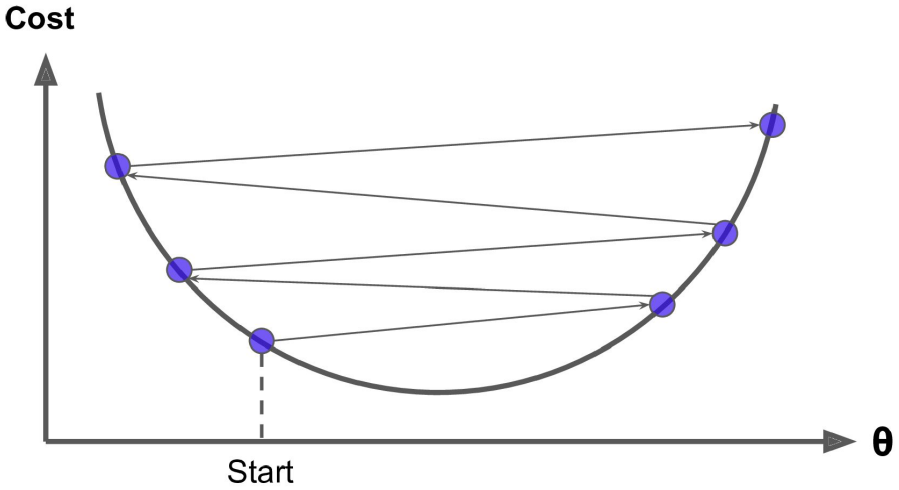
Zdroj: (Géron 2019)

Gradient Descent

- postup: vektor parametrů θ se naplní náhodnými hodnotami a postupně ho po malých krocích upravujeme tak, abychom snížili hodnotu nákladové funkce, dokud nedosáhneme minima

- důležitý hyperparametr této metody: velikost jednoho kroku
  - označováno jako rychlost učení (learning rate)
  - pokud je hodnota příliš malá, může trvat mnoho iterací (a mnoho času) než metoda dosáhne optima
  - pokud je hodnota příliš velká, může být krok natolik velký, že “překročí údolí” a metoda bude divergovat



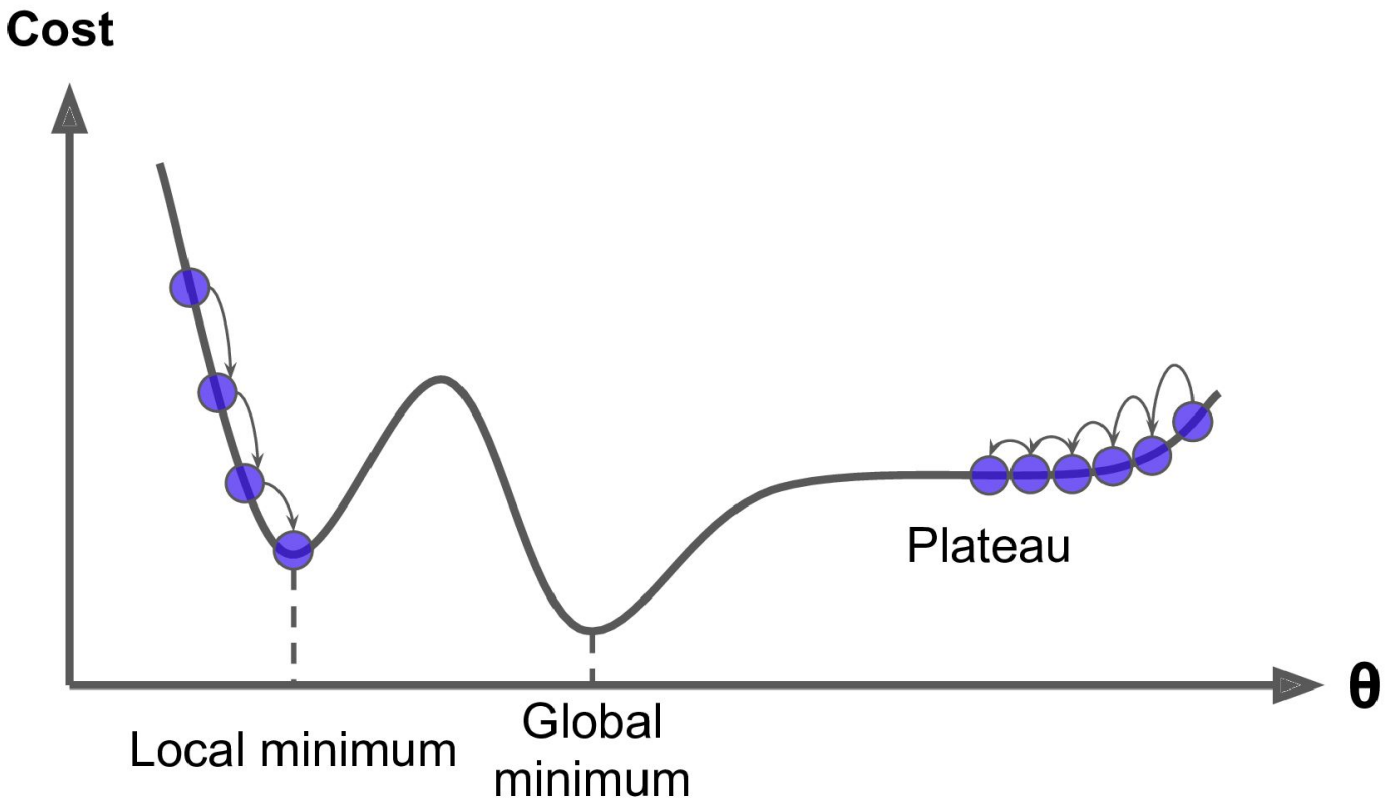


Zdroj: (Géron 2019)

Nepravidelné tvary nákladové funkce

- ne všechny nákladové funkce mají konvexní tvar (tvar mísy nebo údolí)
- mohou se v nich vyskytovat díry, hřebeny, roviny, obecně jakákoliv nepravidelnost
  - díky tomu může být konvergence k minimu velmi obtížná

- dva nejčastější problémy
  - uvíznutí v lokálním minimu (které je horší než globální minimum)
  - uvíznutí na rovině (trvalo by příliš dlouho rovinu překročit, takže algoritmus ukončí optimalizaci)

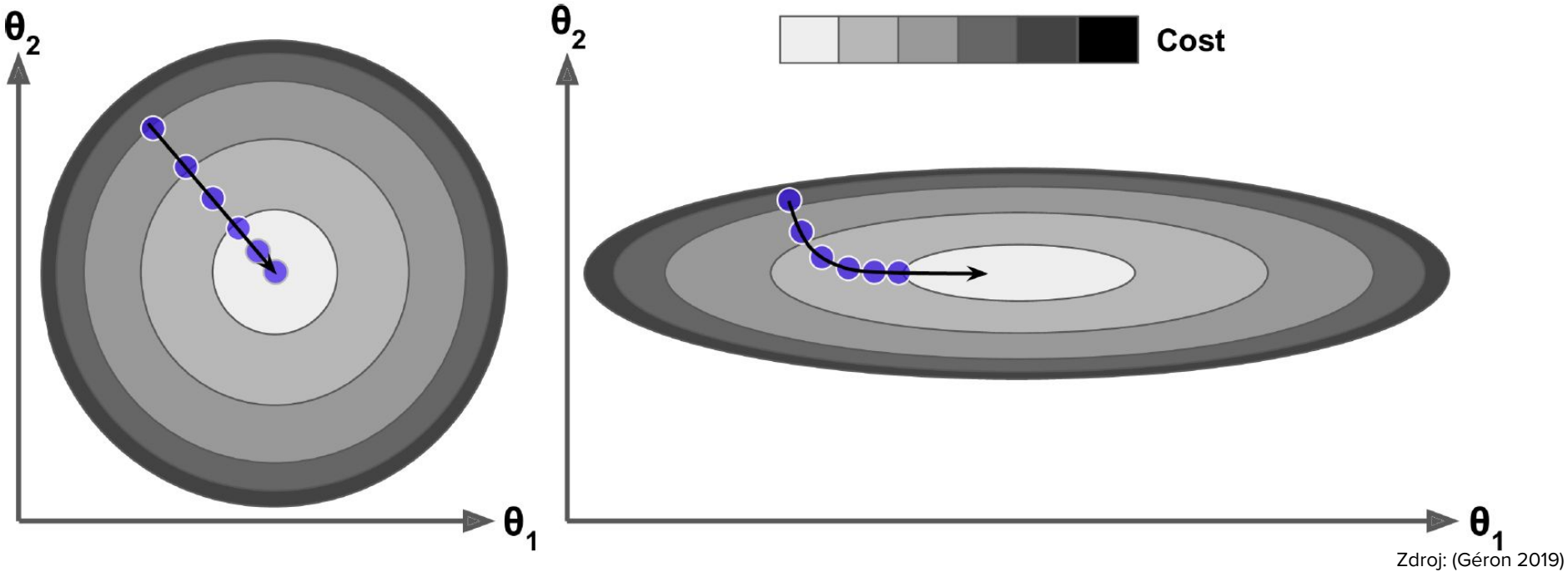


Zdroj: (Géron 2019)

Gradient Descent u lineární regrese

- RMSE (resp. MSE) nákladová funkce je konvexní (spojnice dvou bodů na křivce nikdy neprotne samotnou křivku)
  - nejsou žádná lokální minima, pouze jedno globální minimum
  - sklon této funkce se nikdy nemění náhle
- metoda Gradient Descent tak garantuje nalezení optimálního řešení v případě lin.reg.

- pro lineární regresi má nákladová funkce tvar mísy, nicméně záleží na tom, v jakých škálách se pohybují jednotlivé příznaky
  - proto je nutné hodnoty příznaků normalizovat/standardizovat, abychom dosáhli dobrého řešení



Zdroj: (Géron 2019)

- pokud jsou hodnoty škálované, jde Gradient Descent rovnou směrem optimálního řešení
- pokud škálované nejsou, může se metoda nejdříve vydat směrem, který má k optimálnímu řešení daleko

- trénování modelu tedy znamená hledání hodnot parametrů minimalizujících nákladovou funkci
  - čím více parametrů model má, tím více dimenzí má prostor, který prohledáváme

Gradient Descent

- pro implementaci Gradient Descent potřebujeme znát gradienty nákladové funkce pro každý parametr modelu θ_j
  - tedy jak moc se změní nákladová funkce při drobné změně parametru θ_j (= parciální derivace)
  - “Jaký je sklon svahu pod nohama, když se díváme na sever? Jaký je sklon svahu pod nohama, když se díváme na západ?”

Parciální derivace nákladové funkce MSE:

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Vektor gradientu $\nabla_\theta \mathrm{MSE}(\theta)$:

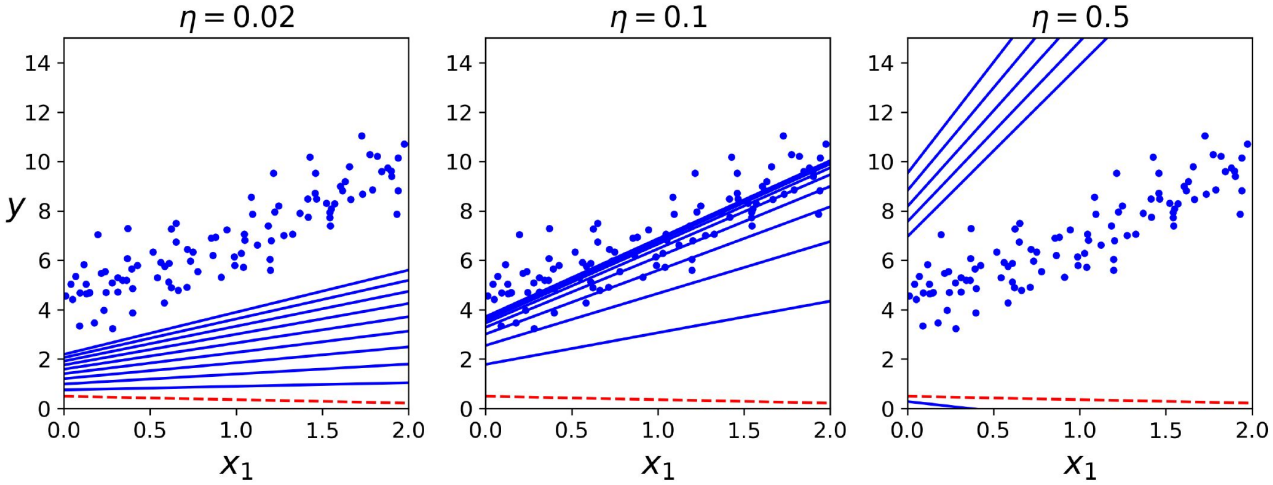
$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Gradient Descent

- vektor gradientu směřuje nahoru, my ale chceme hodnotu minimalizovat, proto nám stačí odečíst $\nabla_\theta MSE(\theta)$ od θ

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$$

η (éta) představuje rychlost učení (learning rate)



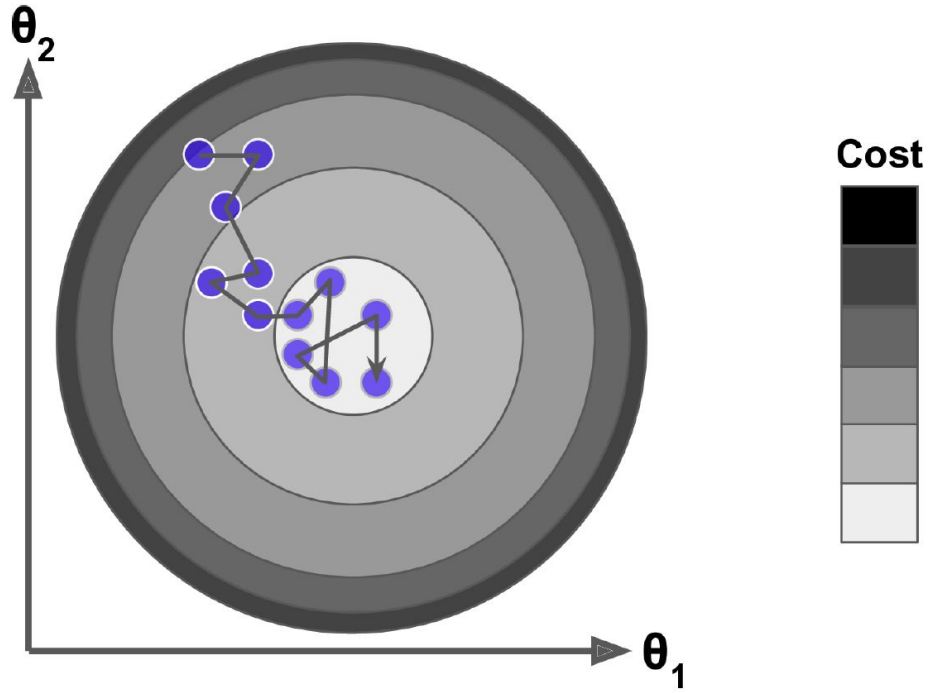
Zdroj: (Géron 2019)

Batch Gradient Descent

- batch = v tomto významu celá dávka dat najednou (X), při každém kroku
- pokud máme velký dataset, bude tato metoda velmi pomalá
- výhodou ale je, že dokáže dobře pracovat s velkým množstvím příznaků (i stovky tisíc)

Stochastic Gradient Descent

- opačný extrém než Batch Gradient Descent - v každém kroku se vybere právě jedna (náhodná) instance z datasetu
- algoritmus je tak pochopitelně mnohem rychlejší
- zároveň je možné ho použít i na obrovské datasety (v paměti stačí aby byla jedna instance)
- nevýhodou je, že vzhledem k prvku náhody se nákladová funkce mění velmi nepravidelně oběma směry a klesá pouze v průměru, nikoliv v každém kroku
- jakmile algoritmus skončí (po určitém počtu iterací), tak finální hodnoty parametrů jsou v průměru dobré, ale nikoliv optimální



Zdroj: (Géron 2019)

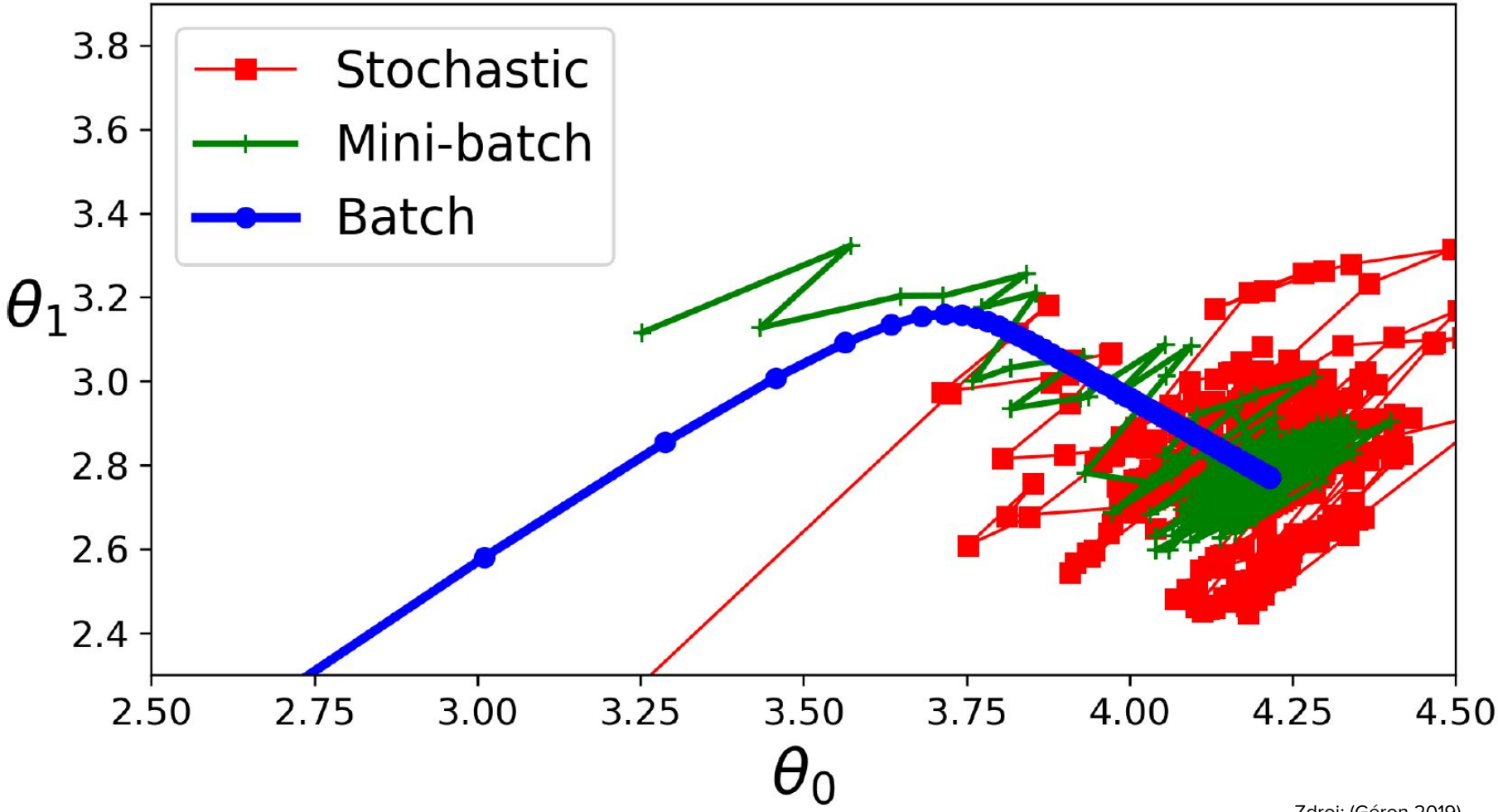
Stochastic Gradient Descent

- pokud má nákladová funkce velmi nepravidelný tvar, může tento přístup naopak pomoci “vyskočit” z lokálního minima a směřovat ke globálnímu minimu
- aby se mohl algoritmus ustálit u optimální cílové hodnoty parametrů, je možné postupně snižovat rychlost učení
  - začne se vysokými hodnotami (rychlý postup vpřed, opuštění lokálních minim)
  - postupně se snižuje
  - nakonec se algoritmus ustálí u globálního minima
- při trénování je třeba, aby dataset byl náhodně zamíchán
  - pokud by byl seřazen podle labelů, tak by implementace SGD brala jednu instanci za druhou a nejdříve optimalizovala pro jednu třídu, pak pro druhou atd, což by nevedlo k úspěchu

Mini-batch Gradient Descent

- na rozdíl od Batch a Stochastic GD se v tomto případě v každém kroku bere mini-dávka trénovacích dat (náhodně vybraná), pro kterou se provádí optimalizace
- hlavní výhoda oproti SGD spočívá v tom, že lze dosáhnout vyšší rychlosti díky využití maticových operací (zejména pokud je k dispozici GPU)

- postup tohoto algoritmu je méně rozptýlený než v případě SGD, ale zase je menší šance na opuštění lokálního minima



Zdroj: (Géron 2019)

Regularizace

- na příkladu lineární a polynomiální regrese (1. cvičení) jsme si ukazovali problém přeučení
  - (pokud má model velkou kapacitu, dokáže se naučit komplexní vztahy v trénovacích datech, ale ve validačních/testovacích datech tyto vztahy reálně nejsou)
- regularizace je jeden z hlavních způsobů, jak omezit problém přeučení
- pro lineární modely se využívají tyto způsoby regularizace:
  - Ridge regrese
  - Lasso regrese
  - Elastic Net
- jejich společným cílem je omezit váhy (parametry modelu), aby nedošlo k přeučení

Ridge regrese

- k nákladové funkci (MSE) je přidán regularizační výraz $\alpha\Sigma\theta^2$
  - tedy druhá mocnina jednotlivých vah (parametrů)
  - (L2-norma vektoru parametrů)
- díky tomu se model trénuje jednak na to, aby vystihl vstupní data, ale zároveň se snaží držet hodnoty parametrů nízké
  - hyperparametr α udává míru regularizace
- regularizační výraz se přidává jen při trénování, při validaci a testování se vynechává
  - nákladová funkce pro trénování může být jiná než pro testování (pro trénování potřebujeme funkci, která jde derivovat)
- nutné data škálovat před regularizací

Příklad Ridge regrese

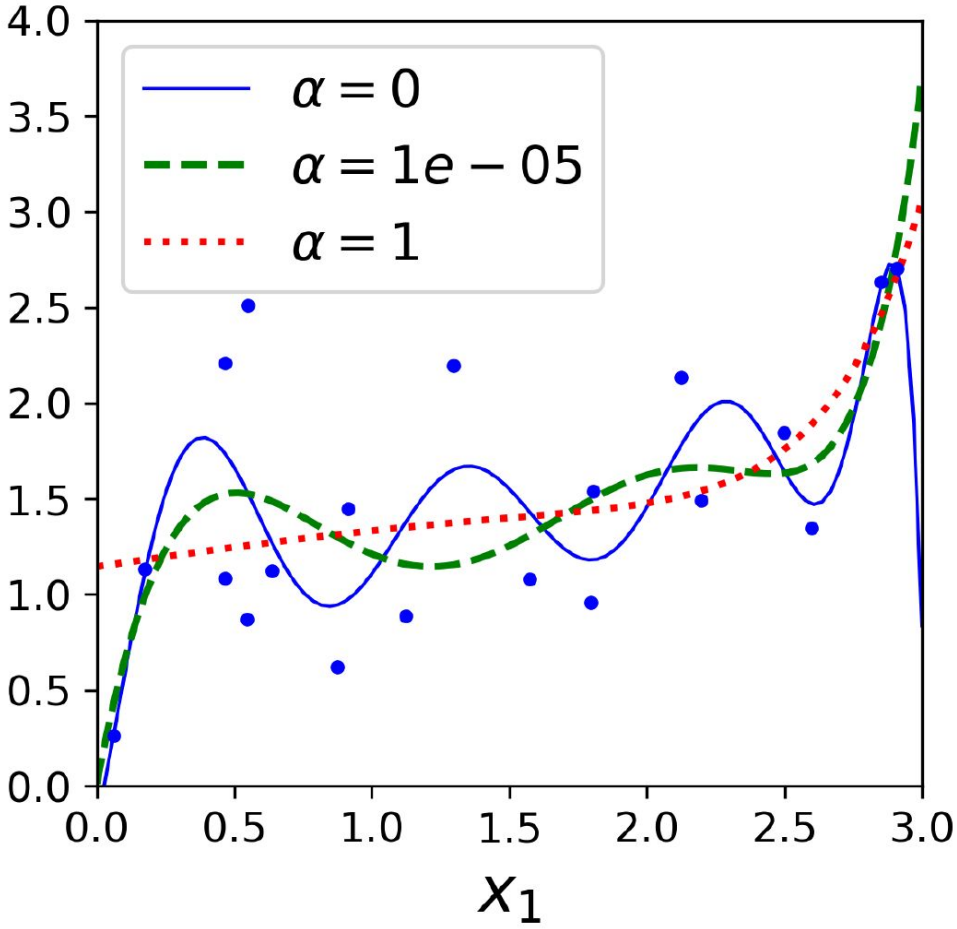
- polynomická regrese
- degree=10
- různé hodnoty α

V Pythonu/sklearn:

```
>>> sgd_reg = SGDRegressor(penalty="l2")

>>> sgd_reg.fit(X, y)

>>> sgd_reg.predict([[1.5]])
```



Zdroj: (Géron 2019)

Lasso regrese

- obdobně jako v případě Ridge regrese i Lasso regrese přidává k nákladové funkci regularizační výraz, ale v tomto případě vycházející z L1-normy vektoru parametrů: $\alpha\Sigma|\theta|$
- důležitá vlastnost Lasso regrese: méně důležité příznaky jsou zcela vynechány (mají nulové váhy)
    - Lasso regrese tedy vede k selekci nejdůležitějších příznaků

Příklad Lasso regrese

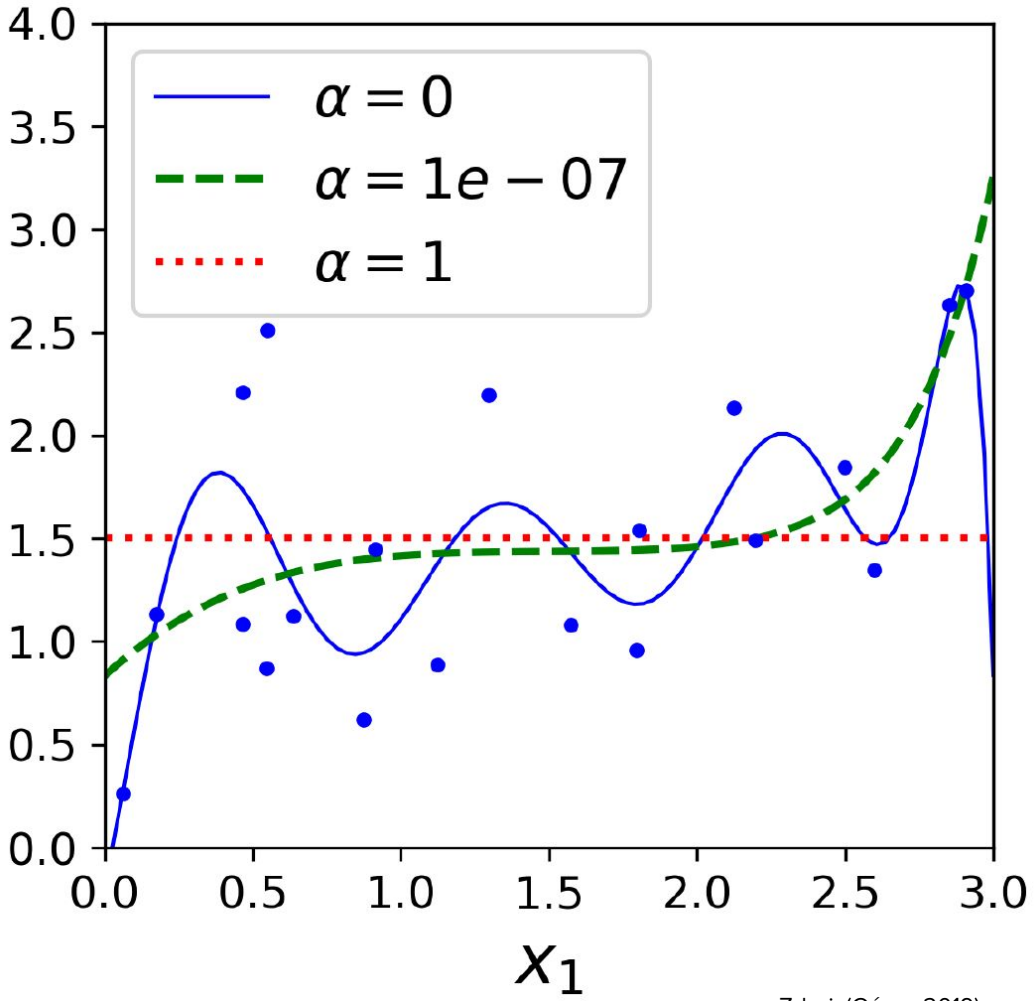
- polynomická regrese
- degree=10
- různé hodnoty α

V Pythonu/sklearn:

```
>>> sgd_reg = SGDRegressor(penalty="l1")

>>> sgd_reg.fit(X, y)

>>> sgd_reg.predict([[1.5]])
```



Zdroj: (Géron 2019)

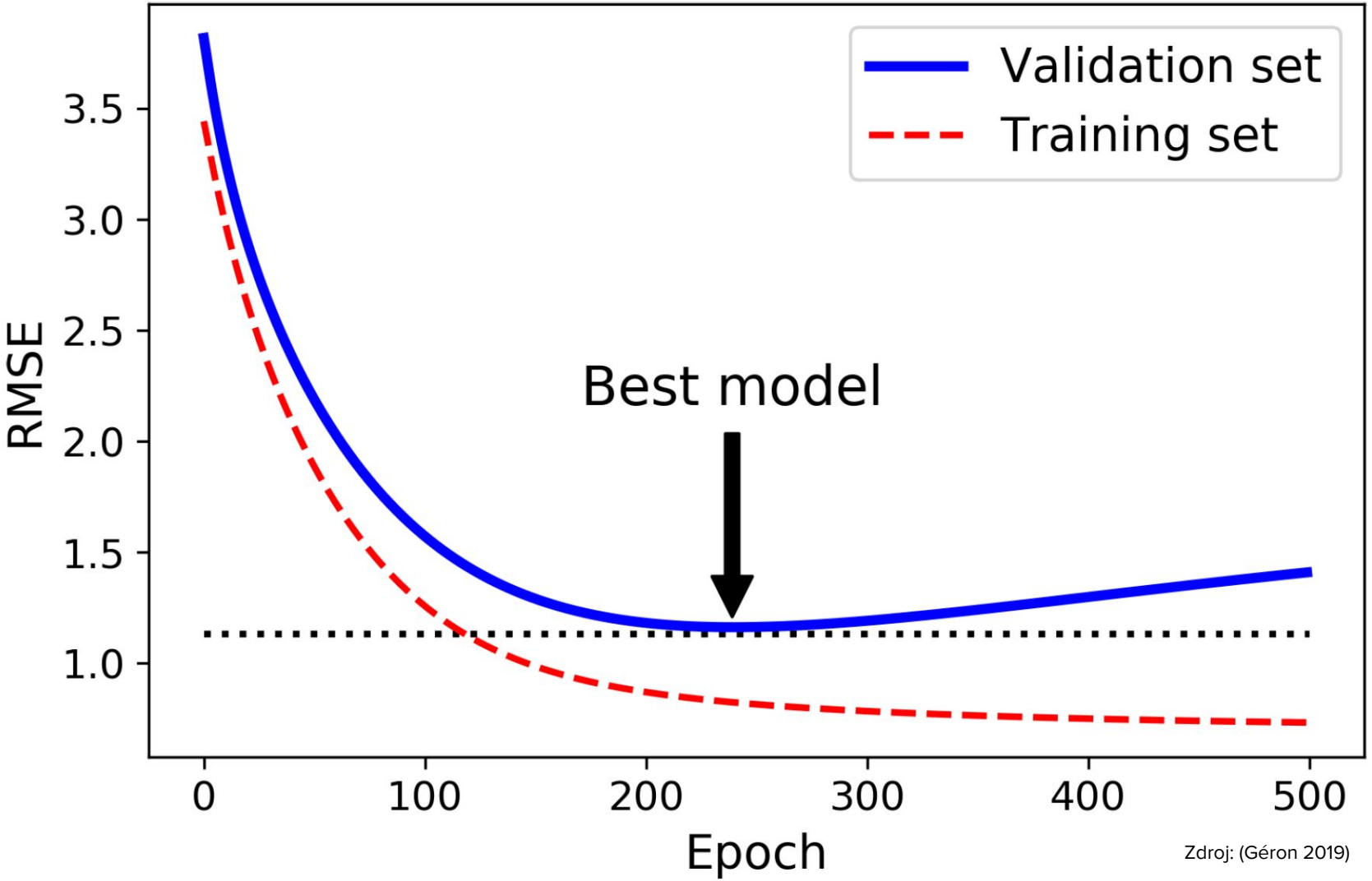
Elastic Net

- kombinace Ridge a Lasso regularizace
- regularizační výraz je tedy mix obou regularizačních výrazů v nastavitelném poměru

- kdy co použít:
  - určitá míra regularizace je doporučována vždy s tím, že se dá začít s Ridge regularizací
  - pokud předpokládáme, že jen některé příznaky jsou důležité, pak Lasso neno Elastic Net je vhodnější volba, protože nám dokážou vybrat jen užitečné příznaky
  - Elastic Net je vhodnější než Lasso, pokud je počet příznaků větší než počet příkladů (pak se Lasso může chovat chybně), nebo pokud jsou některé příznaky silně korelované

Early Stopping

- metoda včasného zastavení je zcela jiný přístup k regularizaci
  - použitelná jen u iterativních algoritmů (což GD je)
- po každé epoše (epocha = balíček několika iterací) se kromě testovací chyby spočítá i validační chyba
- testovací chyba s každou další iterací klesá, zatímco validační chyba do určité doby v průměru klesá a pak začne zase růst jako důsledek přeučení
- metoda early stopping dělá to, že jakmile dosáhne validační chyba minima, trénování se ukončí
- velmi účinná a přitom jednoduchá metoda regularizace
- prakticky si ji vyzkoušíme u trénování neuronových sítí



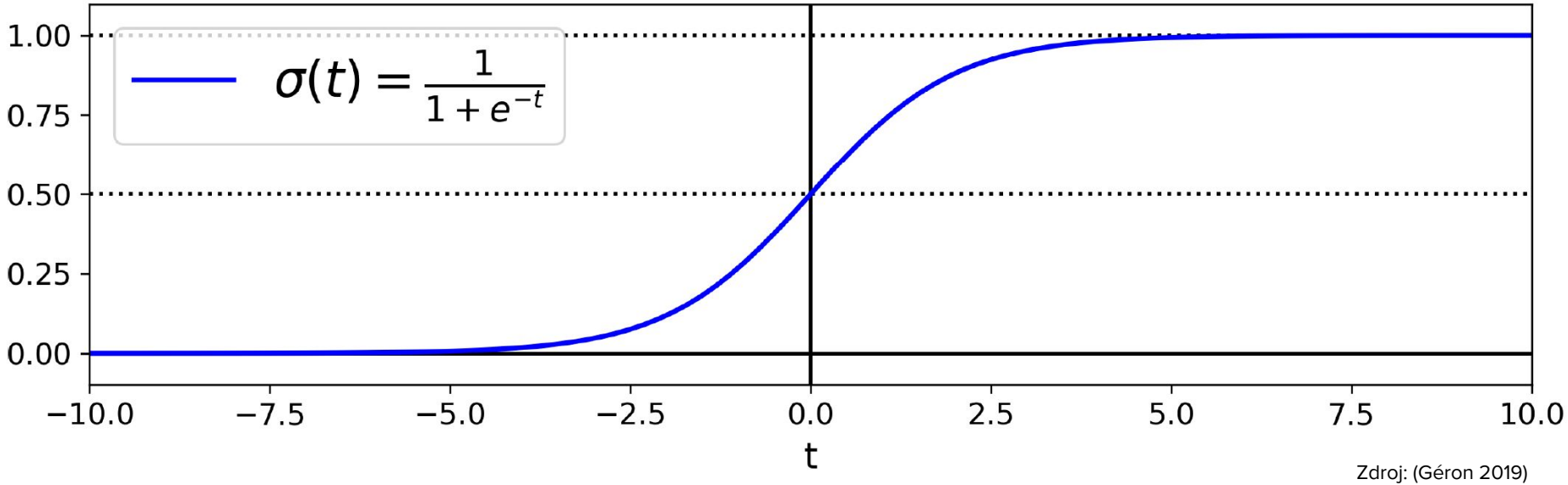
Zdroj: (Géron 2019)

Logistická regrese

- ačkoliv je v názvu slovo regrese, používá se tento přístup pro klasifikaci
- používá se k odhadu pravděpodobnosti, že instance patří k určité třídě
  - jaká je pravděpodobnost, že tato zpráva je spam?
  - pokud je pravděpodobnost vyšší než 50 %, tak se instance přiřadí k cílové třídě a naopak
    - binární klasifikace
- stejně jako lineární regrese i logistická regrese počítá váženou sumu příznaků (plus konstanta), ale výstup je navíc zpracován tzv. logistickou funkcí
  - sigmoidní funkce (tvar S), která vrací hodnotu 0 až 1

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Logistická funkce



Zdroj: (Géron 2019)

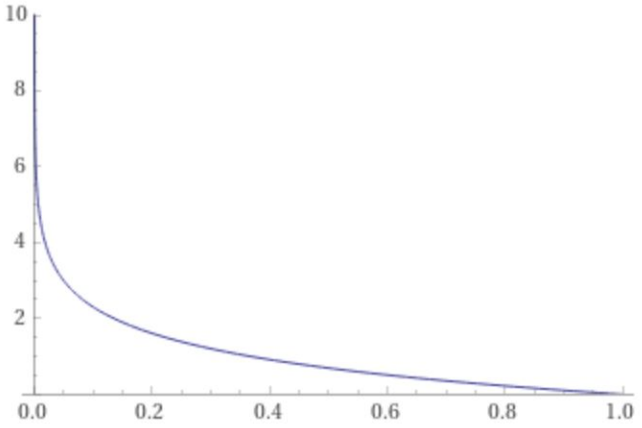
Pokud t je větší než 0, tak σ(t) > 0,5 a model predikuje příslušnost k pozitivní třídě.
Výsledkem výrazu log(p/1-p) je hodnota t pro dané p (tj. inverzní funkce k logistické funkci)

Trénování logistické regrese

- potřebujeme nákladovou funkci, která zajistí vektor parametrů θ tak, že výsledné predikované pravděpodobnosti budou vysoké pro pozitivní instance (y=1) a nízké pro negativní instance (y=0)
- nákladová funkce pro jednu instanci:

$$c(\boldsymbol{\theta}) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1 \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0 \end{cases}$$

Graf funkce -log(p̂):



Trénování logistické regrese

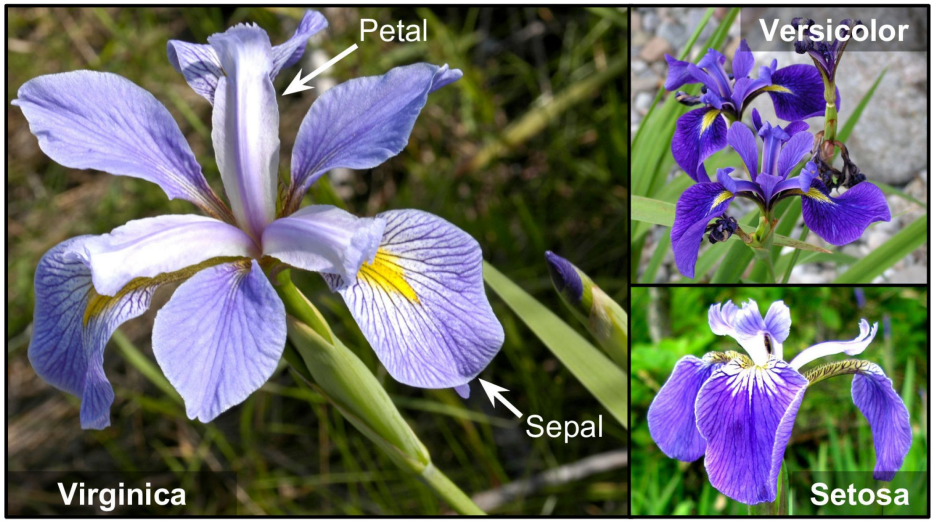
- nákladová funkce pro celý trénovací set:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

- vlastnosti:
  - není známa žádná uzavřená forma řešení, které by určilo optimální hodnoty parametrů
  - nákladová funkce je ovšem konvexní, proto metoda Gradient Descent dokáže s jistotou najít optimální řešení (globální minimum nákladové funkce)

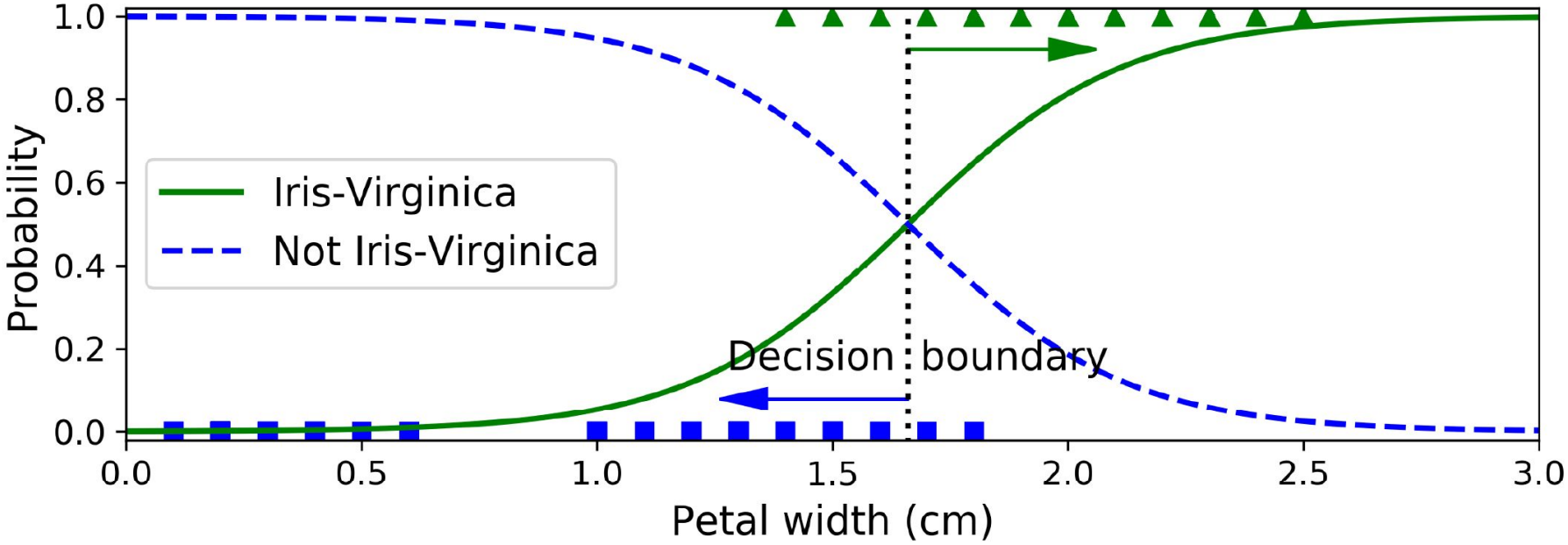
Ukázka logistické regrese

- dataset Iris
  - rostliny rodu kosatec
  - tři různé třídy (druhy kosatců)
  - obsahuje data o délce a šířce okvětních a kališních lístků 150 rostlin a jejich příslušnost ke konkrétnímu druhu
- viz Jupyter



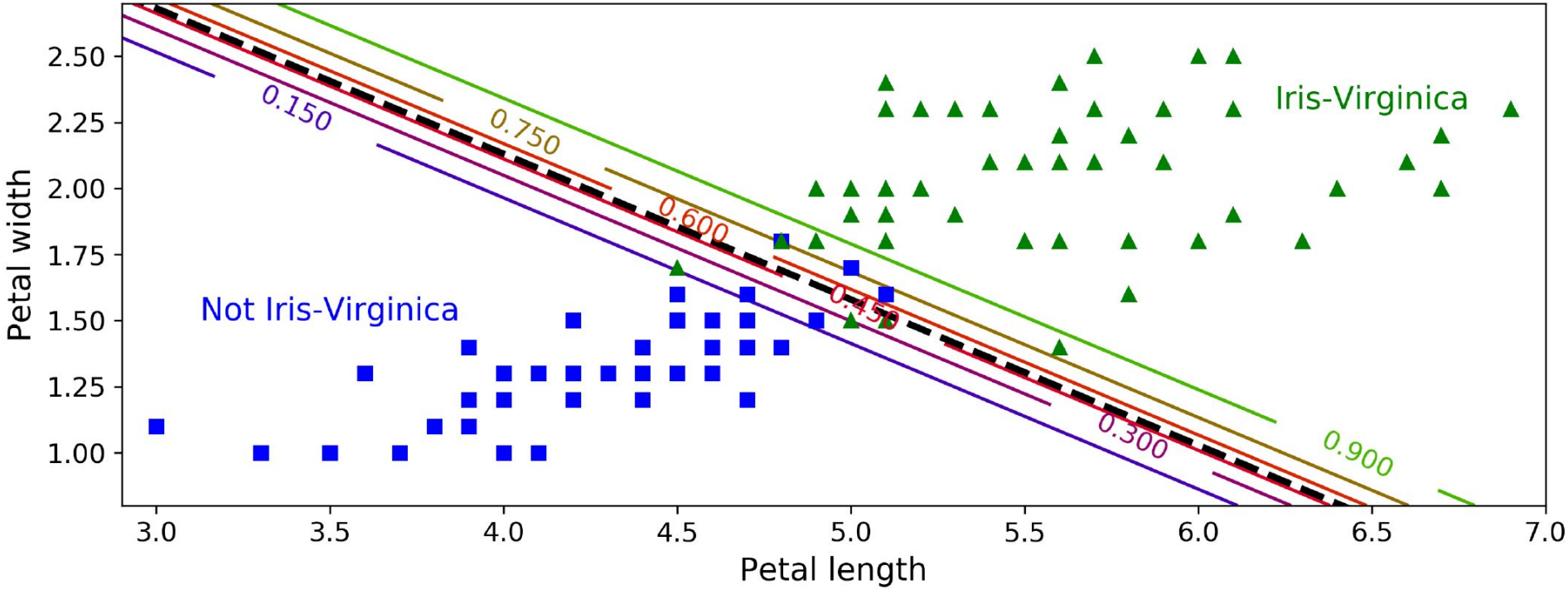
Zdroj: (Géron 2019)

Ukázka logistické regrese



Zdroj: (Géron 2019)

Ukázka logistické regrese - nyní dva příznaky



Zdroj: (Géron 2019)

Softmax regrese

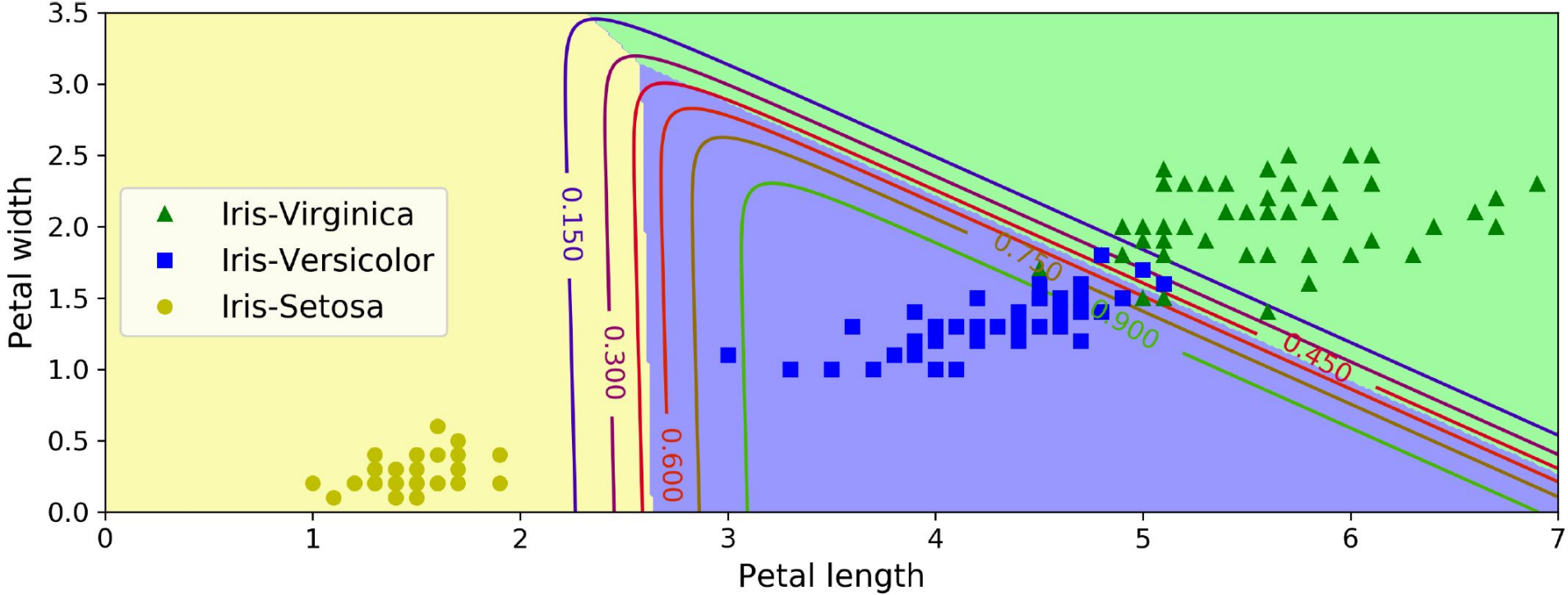
- logistická regrese může být zobecněna do podoby, která přímo umožňuje klasifikaci do více tříd
  - není tedy třeba trénovat více binárních klasifikátorů
- model softmax regrese nejdříve spočítá skóre (vážený součet příznaků, stejně jako v případě lineární regrese nebo logistické regrese) pro každou třídu, na toto skóre následně aplikuje softmax funkci
  - každá třída má tedy vlastní vektor parametrů
- softmax funkce
  - pro každou třídu spočte hodnotu exponenciální funkce jejího skóre a znormalizuje ji (tj. vydělí součtem exp. hodnot všech tříd)
    - součet pravděpodobností jednotlivých tříd je tak pochopitelně roven 1
  - používá se často také jako poslední vrstva v neuronových sítích

Softmax regrese - trénování

- potřebujeme opět vhodnou nákladovou funkci
- používá se tzv. křížová entropie (cross entropy)
  - říká, jak dobře odhadované pravděpodobnosti pro jednotlivé třídy odpovídají skutečným třídám
  - y_k je cílová pravděpodobnost příslušnosti k dané třídě (tedy typicky jedna hodnota 1 a ostatní 0) a -log(p̂) jsme si už ukazovali

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

- ukázka softmax regrese viz Jupyter



Zdroj: (Géron 2019)

Zdroje

- Coelho, L. P.; Richert, W. (2013) Building machine learning systems with Python. Birmingham: Packt Publishing. ISBN 978-1-78216-140-0.

- Géron, A. (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc. ISBN 9781492032649.

- Chollet, F. (2019) Deep Learning v jazyku Python. Knihovny Keras, TensorFlow. Grada Publishing, a.s. ISBN 978-80-247-3100-1.

- Segaran, T. (2007) Programming collective intelligence: building smart web 2.0 applications. Beijing: O'Reilly Media. ISBN 0-596-52932-5.