11. **Дълбоки невронни мрежи**

Въведение в дълбокото обучение

През последните години подходите за **дълбоко обучение** в машинното обучение оказаха значително влияние върху разпознаването на реч и компютърното зрение. Други дисциплини, като обработката на естествен език, също започват да виждат ползи. Критична съставка е използването на много по-големи количества данни, отколкото беше възможно досега. Скорошните успехи възникнаха в контекста на модели с висока сложност — такива с много параметри. Тук методите за **дълбоко обучение** създават гъвкави модели, които използват информация, скрита в огромни масиви от данни, много по-ефективно, отколкото традиционните техники за машинно обучение с ръчно проектирани характеристики.

Погледи върху машинното обучение

• Един начин да разглеждаме машинното обучение е чрез три основни подхода:

1. Класически техники за машинно обучение, които правят прогнози директно от предварително зададени от потребителя характеристики;
2. Техники за обучение на представяния (**representation learning**), които трансформират характеристиките в някакво междинно представяне преди да ги отнесат към финални прогнози;
3. Техники за **дълбоко обучение**, форма на обучение на представяния, която използва множество стъпки на трансформация за създаване на много сложни характеристики.

Ренесансът на невронните мрежи и революцията в дълбокото обучение

• Терминът „ренесанс“ отразява огромния възобновен интерес към невронните мрежи и техниките за **дълбоко обучение**.

• Много водещи медии (напр. The New York Times) документираха впечатляващите успехи на техниките за **дълбоко обучение** върху ключови референтни проблеми.

• От около 2012 г. бяха постигнати впечатляващи резултати върху дългогодишни проблеми в разпознаването на реч и компютърното зрение, както и в състезателни предизвикателства като ImageNet Large Scale Visual Recognition Challenge и оценката Labeled Faces in the Wild.

GPU-та, графи и тензори

• Лесната достъпност до високоскоростни изчисления под формата на графични процесори (GPUs) беше критична за успеха на техниките за **дълбоко обучение**.

• Когато се формулират във вид на матрично-векторна форма, изчисленията могат да бъдат ускорени с помощта на оптимизирани графични библиотеки и хардуер.

• Ето защо ще изучаваме обратното разпространение (**backpropagation**) в матрично-векторна форма.

– Читателите, които не са запознати с манипулирането на функции с матрични аргументи и техните производни, се съветват да разгледат Приложение А.1 за обобщение на полезна основна информация.

• С увеличаването на сложността на мрежовите модели някои величини могат да бъдат представени само чрез многомерни масиви от числа.

– Такива масиви понякога се наричат тензори, обобщение на матриците, което позволява произволен брой индекси.

• Софтуерът за **дълбоко обучение**, поддържащ изчислителни графи

и тензори, е безценен за ускоряване на създаването на сложни мрежови структури и улесняване на тяхното обучение.

Ключови развития

Следните развития изиграха решаваща роля за възраждането на методите за невронни мрежи:

• Правилната оценка на методите за машинно обучение;
• Значително увеличени количества данни;
• По-дълбоки и по-големи мрежови архитектури;
• Ускорено обучение с помощта на GPU техники.

Mixed National Institute of Standards and Technology (MNIST)

• Това е база данни и настройка за оценка за разпознаване на ръкописни цифри.
• Съдържа 60 000 тренировъчни и 10 000 тестови екземпляра на ръкописни цифри, кодирани като изображения в сиви тонове с размер 28×28 пиксела.
• Данните са преработка на по-ранен набор данни на NIST, в който възрастни са генерирали тренировъчните данни, а ученици от гимназия — тестовите.
• Нека сравним производителността на различни методи.

Загуби и регуляризация

• Логистичната регресия може да се разглежда като проста невронна мрежа без скрити слоеве.
• Основният критерий за оптимизация за предвиждане на етикети yi y_i yi (за i=1,…,N i=1,…,N i=1,…,N) от характеристики xi x_i xi с параметри θ \theta θ, състоящи се от матрица от тегла W W W и вектор от отклонения b b b, може да се разглежда като:

∑i=1N−log⁡p(yi|xi;W,b)+λ∑j=1Mwj2=∑i=1NL(fi(xi;θ),yi)+λR(θ)\sum_{i=1}^N -\log p(y_i | x_i; W, b) + \lambda \sum_{j=1}^M w_j^2 =

\sum_{i=1}^N L(f_i(x_i; \theta), y_i) + \lambda R(\theta)i=1∑N −logp(yi|xi;W,b)+λj=1∑Mwj2=i=1∑NL(fi(xi;θ),yi)+λR(θ)

• Първият член, L(fi(xi;θ),yi) L(f_i(x_i; \theta), y_i) L(fi(xi;θ),yi), е отрицателната условна логарифмична вероятност или загуба, а вторият член, λR(θ) \lambda R(\theta) λR(θ), е претеглен регуларизатор, използван за предотвратяване на прекомерно напасване (**overfitting**).

Емпирична минимизация на риска

• Тази формулировка като целева функция, базирана на загуби и регуларизатор, ни дава свободата да избираме или вероятностни загуби, или други функции за загуба.

• Използването на средната загуба върху тренировъчните данни, наречена емпиричен риск, води до следната формулировка на оптимизационния проблем: минимизиране на емпиричния риск плюс регуларизационен член, т.е.

arg⁡min⁡θ[1N∑i=1NL(fi(xi;θ),yi)+λR(θ)].\arg\min_{\theta} \left[\frac{1}{N} \sum_{i=1}^N L(f_i(x_i; \theta), y_i) + \lambda R(\theta) \right].argθmin[N1i=1∑NL(fi(xi;θ),yi)+λR(θ)].

• Имайте предвид, че факторът N N N трябва да се вземе предвид, ако се свързва теглото на регуларизацията тук с параметъра, получен от формален вероятностен модел за разпределение на параметрите.

На практика

• В **дълбокото обучение** често ни интересува разглеждането на криви на обучение (**learning curves**), които показват загубата или някакъв друг показател за производителност в графика като функция от броя на преминаванията, които алгоритъмът е направил върху данните.

• По-лесно е да се сравни средната загуба върху тренировъчния набор със средната загуба върху валидационния набор на същата графика, защото деленето на N им дава един и същ мащаб.

Често срещани загуби за невронни мрежи

• Финалната изходна функция на невронна мрежа обикновено има формата

• Често използвани функции за загуба на изхода, функции за активиране на изхода и основните разпределения, от които те произлизат, са показани по-долу.

Архитектури на дълбоки невронни мрежи

• Състоят се от изчисления, извършвани от много слоеве.

• Обозначавайки изхода на скритите, изчислението за мрежа с L скрити слоя е:

• Където функциите за предварително обикновено са линейни

• Тази формулировка може да се изрази с помощта на единна параметрична матрица θ \theta

Дълбоки мрежи с права връзка

• За разлика от байесовите мрежи, скритите единици тук са междинни детерминистични изчисления, а не случайни променливи, поради което не се представят като кръгове.

• Въпреки това, изходните променливи y_k се изобразяват като кръгове, защото могат да бъдат формулирани вероятностно.

Функции за активиране

• Функциите за активиране обикновено действат върху векторите за предварително активиране по елементен начин.

• Докато сигмоидните функции бяха популярни, хиперболичният тангенс понякога се предпочита, отчасти защото има стабилно

състояние при 0.
• Напоследък функцията **rectify()** или коригирани линейни единици (**ReLUs**) показаха по-добри резултати в много различни контексти.
– Тъй като ReLUs дават 0 за отрицателни стойности на аргумента, някои единици в модела ще имат активации, равни на 0, което дава свойство на разреденост, полезно в много контексти.
– Градиентът е особено прост — или 0, или 1.
– Това помага за решаването на проблема с избухващия градиент (**exploding gradient**).
• Редица софтуерни пакети улесняват използването на различни функции за активиране, като определят градиентите автоматично чрез символни изчисления.

Библиографски бележки и допълнителна литература
• Алгоритъмът за обратно разпространение (**backpropagation**) е известен в почти сегашната си форма от докторската дисертация на Уърбос (1974).
• В обширния си литературен преглед на **дълбокото обучение**, Шмидхубер (2015) проследява ключови елементи на алгоритъма още по-назад.
– Той също проследява идеята за „дълбоки мрежи“ до работата на Ивахненко и Лапа (1965).
• Популярността на техниките за невронни мрежи премина през няколко цикъла и макар някои фактори да са социални, има важни технически причини за тенденциите.
• Еднослойна невронна мрежа не може да реши проблема XOR, недостатък, осмян от Мински и Папърт (1969), което спря развитието на невронните мрежи през следващите десетилетия.

Обучение и оценка на дълбоки мрежи
Ранно спиране

• **Дълбокото обучение** включва архитектури с висока сложност, които са податливи на прекомерно напасване (**overfitting**), дори когато данните са изобилни.
• Ранното спиране (**early stopping**) е стандартна практика, дори когато се използват други методи за намаляване на прекомерното напасване, напр. регуляризация и **dropout**.
• Идеята е да се наблюдават кривите на обучение, които изобразяват средната загуба за тренировъчния и валидационния набор като функция от епохата.
• Ключът е да се намери точката, в която средната загуба на валидационния набор започва да се влошава.

Валидационни набори и хиперпараметри
• В **дълбокото обучение** хиперпараметрите се настройват чрез идентифициране на настройките, които водят до най-добра производителност върху валидационния набор, използвайки ранно спиране.
• Често срещани хиперпараметри включват силата на регуляризацията на параметрите, но също и сложността на модела по отношение на броя на скритите единици и слоеве, тяхната свързаност, формата на функциите за активиране и параметрите на самия алгоритъм за обучение.
• Поради многото избори, мониторингът на производителността върху валидационни набори играе още по-централна роля, отколкото при традиционните методи за машинно обучение.

Тестови набори
• Трябва да се отделят за истинска финална оценка.
• Повтарящите се експерименти с данни от тестовия набор дават заблуждаващи (напр. оптимистични) оценки за производителността върху нови данни.

• Поради тази причина изследователската общност предпочита публични предизвикателства с скрити етикети на тестовия набор, развитие, което несъмнено помогна за измерване на напредъка в областта.

Валидационни набори срещу кръстосана валидация

• Използването на валидационен набор е различно от използването на k-кратна кръстосана валидация за оценка на техника за обучение или за избор на хиперпараметри.
• Кръстосаната валидация включва създаването на множество тренировъчни и тестови дялове.
• Наборите данни за **дълбоко обучение** обикновено са толкова масивни, че един голям тестов набор адекватно представя производителността на модела, намалявайки нуждата от кръстосана валидация.
– Тъй като обучението често отнема дни или седмици, дори с използване на GPU-та, кръстосаната валидация често е непрактична.

Инициализация на параметри

• Може да бъде изненадващо важна!
• Отклоненията често се инициализират на 0 без проблеми.
• Матриците на теглата са по-проблематични, напр.:
– Ако се инициализират на всички 0, може да се покаже, че функцията за активиране **tanh** ще даде нулеви градиенти.
– Ако теглата са еднакви, скритите единици ще произвеждат еднакви градиенти и ще се държат по един и същ начин (хабейки параметри).
• Едно решение: инициализиране на всички елементи на матрицата на теглата от равномерно разпределение в интервала [-b, b\].
• Предложени са различни методи за избор на стойността на b b b,

често мотивирани от идеята, че единици с повече входове трябва да имат по-малки тегла.

Ненадзирано предварително обучение

• Идея: моделиране на разпределението на немаркирани данни с метод, който позволява параметрите на научения модел да информират или да бъдат прехвърлени към мрежата.
• Може да бъде ефективен начин за инициализиране и регуляризиране на мрежа с права връзка (**feedforward network**).
• Особено полезно, когато обемът на маркираните данни е малък спрямо капацитета на модела.

Увеличаване на данните

• Може да бъде критично за най-добри резултати.
• Както се вижда в таблицата за MNIST, увеличаването дори на голям набор данни с трансформирани данни може да подобри производителността.
• Проста трансформация е леко разместване на изображението.
• Ако обектът, който трябва да се класифицира, може да бъде изрязан от по-голямо изображение, случайни ограничаващи рамки могат да бъдат поставени около него, добавяйки малки транслации във вертикална и хоризонтална посока.

Автокодери

• Използват се за ненадзирано обучение (**unsupervised learning**).
• Това е мрежа, която учи ефективно кодиране на своя вход.
• Целта е просто да се възстанови входът, но чрез междинно представяне с намалена размерност.
• Ако изходът е формулиран с вероятност, целевата функция е да се оптимизира p(x|x~;f(x~)) p(x | \tilde{x}; f(\tilde{x})) p(x|x~;f(x~)), т.е. вероятността моделът да даде на случайна променлива x x x

стойността, дадена от наблюдението x~ \tilde{x} x~, където x~=h=act(a(1)) \tilde{x} = h = \text{act}(a^{(1)}) x~=h=act(a(1)).

Прост автокодер

• Предсказва собствения си вход
• Преминава през кодиране.

Дълбоки автокодери

• Когато се изграждат автокодери от по-гъвкави модели, често се използва тесен участък (**bottleneck**) в мрежата, за да се получи представяне с намалена размерност, предоставяйки механизъм за получаване на кодиране с по-ниска размерност от входа.
• Дълбоките автокодери могат да научат представяния с ниска размерност с по-малка грешка при възстановяване от PCA, използвайки същия брой размери.

Софтуер за дълбоко обучение

Theano

• Библиотека в Python, разработена с конкретната цел да улесни изследванията в **дълбокото обучение** (Бергстра и др., 2010; Екип за разработка на Theano, 2016).
• Също така е мощен инструмент за общо математическо програмиране.
• Theano разширява NumPy (основният пакет на Python за научни изчисления) чрез добавяне на символна диференциация и поддръжка на GPU, наред с други функции.

TensorFlow

• Софтуерна библиотека, базирана на C++ и Python, за видовете числови изчисления, обикновено свързани с **дълбокото обучение** (Абади и др., 2016).
• Силно вдъхновена от Theano и, подобно на нея, използва графи на

потока от данни, за да представи начина, по който многомерните масиви от данни комуникират помежду си.

Torch

• Отворена библиотека за машинно обучение, изградена с помощта на C и скриптов език на високо ниво, известен като Lua (Колобер и др., 2011).
• Поддържа многомерни масиви от данни и различни основни манипулации с линейна алгебра.

Computational Network Toolkit (CNTK)

• C++ библиотека за манипулиране на изчислителни мрежи (Ю и др., 2014).
• Създадена от Microsoft Research, но пусната под разрешителен лиценз.

Caffe

• C++ и Python базирана библиотека за конволюционни невронни мрежи под BSD лиценз (Джия и др., 2014).
• Има чист и разширяем дизайн, което я прави популярен алтернативен вариант на оригиналната имплементация с отворен код на AlexNet на Крижевски и др. (2012), която спечели предизвикателството ImageNet през 2012 г.

Deeplearning4j

• Java-базирана библиотека за **дълбоко обучение** с отворен код, достъпна под лиценз Apache 2.0.
• Използва клас за многомерни масиви и предоставя поддръжка за линейна алгебра и манипулация на матрици, подобна на тази на NumPy.

Lasagne, Keras и cuDNN

• **Lasagne** е лека Python библиотека, изградена върху Theano, която

опростява създаването на слоеве на невронни мрежи.
• **Keras** е Python библиотека, която работи върху Theano или TensorFlow (Чолет, 2015), позволяваща бързо дефиниране на мрежова архитектура чрез слоеве и включваща функционалност за предварителна обработка на изображения и текст.
• **cuDNN** е високо оптимизирана GPU библиотека за NVIDIA устройства, която позволява по-бързо обучение на дълбоки мрежи.

Поддръжка на дълбоко обучение в Weka

Дълбокото обучение може да се имплементира в Weka чрез три метода:
• C wrapper класификатори за пакета на трета страна DeepLearningForJ, налични в пакета deepLearningForJ.
• Използване на MLRClassifier от пакета RPlugin, за да се използват имплементации на **дълбоко обучение** в R.
• Достъп до Python-базирани библиотеки за **дълбоко обучение** чрез пакета PyScript.