ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Введение в библиотеку PyTorch. Обучение нейронных сетей

Аннотация к занятию: обучающиеся познакомятся с PyTorch — современной библиотекой глубокого обучения. Разберут три техники регуляризации: dropout, weight decay и batch-нормализацию. Все эти три техники крайне полезны для обучения нейронных сетей. Основные применения этих техник — ускорение и стабилизация обучения.

Цель занятия: познакомить обучающих с наиболее популярными техниками регуляризации в нейронных сетях (dropout, weight decay и batch-нормализацией).

Задачи занятия:

- сформировать представление о библиотеке PyTorch;
- обсудить, зачем нужна регуляризация;
- познакомить обучающихся с техниками регуляризации: dropout, weight decay и batch-нормализацией.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	5 мин.	Дорогие друзья, здравствуйте. Продолжаем изучение нейронных сетей.	Приветствие. Создание в классе психологического комфорта
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	7 мин.	Все алгоритмы машинного обучения в большей или меньшей степени страдают от проблемы переобучения. Если мы сравним, скажем, градиентный бустинг и логистическую регрессию, то увидим, что бустинг в большей степени склонен к переобучению. Почему это происходит? **Ответы обучающихся** Дело в том, что у градиентного бустинга в среднем гораздо больше параметров: каждое дерево средней глубины содержит в себе значительное количество параметров, а в бустинге этих деревьев множество. В этом смысле совершенно ясно, что нейронная сеть как модель с огромным количеством параметров должна быть крайне склонна к переобучению. В целом, ровно так и оказывается. Нейронная сеть способна быстро выучить правильные ответы на	Способствовать обсуждению мотивационных вопросов. Для справки: https://habr.com/ru/post/334380/

		обучающей выборке, а вот как она будет обобщаться на выборку тестовую — всегда большой вопрос. Итак, большое количество параметров нейросети и сложная многослойная архитектура влекут переобучение и другие нестабильности модели, когда она сталкивается с данными, которые не видела в ходе обучения. Решением этой проблемы в машинном обучении является регуляризация. Давайте вспомним, что такое регуляризация. Возможный ответ обучающихся Регуляризация в широком смысле — это совокупность разнообразных техник и приёмов, направленных на стабилизацию модели и борьбу с переобучением. На этом занятии мы познакомимся с наиболее популярными техниками регуляризации в нейронных сетях. Обсудим, как вычислять вероятность принадлежности тому или иному классу, введём понятие логистической регрессии и запишем её в матричном виде	
Изучение нового материала	55 мин.	Начнём с техники weight decay, что в переводе с английского — сокращение весов. Мы проходили её, когда изучали линейные алгоритмы анализа данных. Для нейросетей всё работает аналогично. Как заметить переобучение? Вопрос сложный, точного ответа на него дать не получится. Но в машинном обучении есть некоторые сигналы, которые указывают на	Для справки: https://habr.com/ru/company/wunderfund/blog/315476/

		переобучение. Один из них — слишком большие по модулю веса модели: в хороших моделях большие веса встречаться не должны. Мы решаем эту проблему точно так же, как и для линейных алгоритмов. Пусть у нас есть некоторая функция потерь L(x, y, \theta), на которой мы тренируем модель. Здесь theta — это вектор всех параметров алгоритма. Тогда модифицированная функция потерь будет равна L(x, y, \theta) плюс регуляризационное слагаемое, равное некой константе C, домноженной на норму вектора весов theta в квадрате. Иными словами, C на сумму квадратов всех весов нашей модели. Кстати, можно регуляризовывать только один конкретный слой нейронной сети, тогда theta будет вектором параметров только этого слоя. Отмечу, что константа C — это гиперпараметр нашего алгоритма. Следовательно, её нужно выбирать перед началом обучения. Следующая техника называется dropout. Хорошего перевода с английского нет, но пусть будет «выбрасывание». Ещё один фактор, который может сделать нашу модель нестабильной, — это смысловая «перегруженность» некоторых нейронов, то есть излишняя значимость нейронов для нашей модели. Представьте, что в слое есть единственный нейрон, который реагирует на определённый узор на картинке. Тогда, если этот узор немного изменить, нейрон может перестать на него реагировать. Если же на этот узор будет реагировать несколько нейронов, то, возможно,	Для справки: Дропаут — это метод, который применяется к конкретному слою нейросети.

		хотя бы один из них отреагирует даже на немного изменённый узор. Необходимо попросить нейросеть «не складывать все яйца в одну корзину», избежать ситуации, когда один конкретный нейрон имеет слишком большое значение для всей сетки. Именно это позволяет сделать dropout. Дропаут — это метод, который применяется к конкретному слою нейросети. Давайте зафиксируем некоторую вероятность дропаута p. На стадии обучения рассмотрим все нейроны определённого слоя. К примеру, возьмём второй слой нейросети на картинке. Вероятность дропаута в данном случае равна, допустим, 0,6. Возьмём какой-нибудь батч изображений. Для этого батча «отключим» случайные нейроны слоя, который мы рассматриваем. На слайде мы отключили первый, третий и четвёртый нейроны в выделенном слое. Каждый нейрон мы будем отключать с вероятностью p и оставлять включённым с вероятностью 1−p. Теперь отключенные нейроны для данного батча не участвуют в работе нейросети: они не получают сигнал с предыдущего слоя и не отдают сигнал дальше. Чего мы добиваемся таким отключением? **Ответы обучающихся** У нейронной сети при обучении теперь нет возможности сгрузить всю важную информацию в один нейрон, потому что с вероятностью p этот нейрон	

		будет вообще отключён на стадии обучения! Таким образом, нейронная сеть будет вынуждена дублировать любую информацию сразу в нескольких нейронах, чего мы и добивались. За счёт этого нейросеть становится более стабильной. Казалось бы, всё в порядке и проблема решена. Но пока радоваться рано. Мы разобрались, как обучать нейросеть с дропаутом, но не очень понятно, как её применять. На стадии инференса мы хотим включить все нейроны одновременно. Но если мы включим все нейроны, то в каждый нейрон следующего слоя будет приходить, как в примере на картинке, все 5 рёбер. На стадии же обучения в каждый нейрон приходили только 2 ребра, как видно из второй картинки. Оказывается, на стадии применения модели можно умножить сигнал из каждого нейрона на число (1−p) — в данном случае это 0,4. Тогда в каждый нейрон будет приходить в среднем столько же сигнала, сколько приходило при обучении модели. Понятно, что это объяснение строго не доказывает, что слой дропаута на стадии инференса будет работать так, как мы этого хотим. Но это предположение, которое на практике оказывается справедливым. Нам с вами вообще повезло, так как дропаут в библиотеке PyTorch реализован в виде отдельного слоя и нет нужды самостоятельно его реализовывать. Последняя техника, которую мы изучим, называется batch normalization. На русский язык это переводится	Для справки: https://habr.com/ru/post/309302/

		как нормализация по батчу. Разберёмся, что имеется в виду. Когда мы говорили про линейные алгоритмы, мы обсуждали, что данные необходимо нормировать перед подачей на вход, то есть преобразовывать каждый признак так, чтобы среднее значение признака по всей выборке было равно 0, а стандартное отклонение — единице. Эту же идею мы хотим реализовать для нейронных сетей. Отличие от линейных алгоритмов в том, что мы бы хотели иметь возможность вставлять нормализацию данных между слоями нейронной сети. Здесь в роли признака выступает выход одного конкретного нейрона: мы бы хотели, чтобы числа, выходящие с каждого нейрона, были одного и того же масштаба. Здесь нас неизбежно настигает проблема. Чтобы вычислить мю и сигма (то есть среднее и дисперсию) для одного признака на каком-то конкретном слое, необходимо прогнать через этот слой всю обучающую выборку. Ничего страшного, если бы нам нужно было прогнать всю выборку через слой один раз, это мы бы могли себе позволить. Проблема в том, что на каждой итерации обучения нейронной сети среднее и дисперсия выхода конкретного нейрона могут очень сильно меняться, потому что меняются веса самой нейросети. Итак, вычислять среднее и сигму по всей выборке мы не можем: это очень дорого. Появляется решение: давайте в процессе обучения делать нормировку не по всей обучающей выборке, а по одному батчу из	

		обучающей выборки! Тогда проблемы лишних вычислений не возникает: вычислить среднее и дисперсию по батчу — это быстро. В библиотеках глубокого обучения эта идея реализована в виде отдельного слоя, который так и называется — слой батч-нормализации. Давайте опишем этот алгоритм. Итак, вход нашего слоя — это значения выхода одного нейрона x на одном батче. Батч мы обозначим буквой B красивое. У нашего алгоритма будут два обучаемых параметра — гамма и бета. Чуть позже мы поймём, зачем они нужны, но сразу скажу, что это не среднее и дисперсия. Выходы нашего слоя обозначим через игреки. Слой действует следующим образом: вычисляем среднее значение признака по батчу и дисперсию. Затем нормализуем выходы: вычитаем среднее и делим на корень из дисперсии с небольшой фиксированной добавкой эпсилон, которая нужна для того, чтобы знаменатель дроби никогда не обращался в ноль. Казалось бы, всё: мы нормировали выходы, это то, что мы хотели. Но здесь в игру вступают как раз те параметры гамма и бета, которые мы анонсировали. К полученным нормированным значениям мы должны применить ещё одно линейное преобразование, где	

		гамма и бета — это обучаемые параметры нашей модели. Зачем мы это делаем? **Ответы обучающихся** Мотивировка может быть такой: мы хотим, чтобы модель имела возможность ничего не делать с выходами, если это по какой-то причине не является целесообразным. Понятно, что гамму и бету можно подобрать так, чтобы игреки, то есть выходы, были равны входам, то есть иксам. Это даёт модели дополнительную свободу, не сильно увеличивая количество параметров нейросети. Отдельно отмечу, как именно вычисляются значения мю и сигма во время инференса. На стадии обучения алгоритм запоминает среднее значение и дисперсию по всей выборке. На стадии инференса мы можем не вычислять среднее по батчу (так как батча на стадии применения может и не быть — должна же быть возможность применить модель к одному объекту). Вместо этого для нормировки мы используем агрегированные значения мю и сигма. Итак, давайте перечислим преимущества батч-нормализации. Этот алгоритм был предложен в 2015 году Сергеем Иоффе и Кристианом Жегеди. Алгоритм улучшал обучение нейронной сети и делал его более стабильным. Но за несколько лет роль батч-нормализации была переосмыслена сообществом ресёрчеров. Сейчас считается, что батч-нормализация в основном ускоряет обучение в нейронных сетях, так как решает проблему слишком маленьких весов и	

		слишком маленьких градиентов. Мы в эту проблематику вдаваться не будем, а лишь отметим, что батч-нормализация — это хороший способ борьбы с проблемами, которые возникают при обучении нейросетей с большим количеством слоёв. Хорошая новость в том, что батч-нормализация реализована в библиотеках глубокого обучения, поэтому я призываю её использовать	
Закрепление изученного материала	10 мин.	**Вопросы для обсуждения:** ● В чём смысл техники weight decay? ● В чём смысл техники dropout? ● В чём смысл техники batch normalization?	Педагог организует беседу по вопросам
Этап подведения итогов занятия (рефлексия)	8 мин.	**Вопросы для обсуждения:** ● Чему я научился? ● С какими трудностями я столкнулся? ● Каких знаний мне не хватает для более глубокого понимания изученного материала? ● Достиг ли я поставленных целей и задач?	Педагог способствует размышлению обучающихся над вопросами
Информация о домашнем задании, инструктаж по его применению	5 мин.	В этом занятии вам предстоит потренироваться в обучении моделей с помощью библиотеки Pytorch. Мы будем работать с игрушечным датасетом moons. Вам предстоит самостоятельно реализовать в Pytorch логистическую регрессию.	

		Вам необходимо написать модуль на PyTorch, реализующий функцию f(X)=Xw, где w — параметр (nn.Parameter) модели. Иначе говоря, здесь мы реализуем своими руками модуль nn.Linear (в этом пункте его использование запрещено). Инициализируйте веса нормальным распределением (torch.randn). В самом модуле LogisticRegression мы не применяем к выходу сигмоиду, поскольку после этого неудобно будет вычислять функцию потерь. Сигмоиду нужно будет применять к выходу самостоятельно. Поэтому модуль LogisticRegression не будет отличаться от модуля LinearRegression, который мы реализовывали на семинаре Введение в Pytorch. Далее, потренируемся в вычислении функции потерь. Даны матрица объекты-признаки X, вектор весов w и вектор правильных ответов y. Вычислите функцию потерь по алгоритму выше.	

Рекомендуемые ресурсы для дополнительного изучения:

- PyTorch. [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/334380/.
- Глубокое обучение. тонкая настройка нейронной сети. [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/wunderfund/blog/315476/.
- Регуляризация. [Электронный ресурс] – Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Регуляризация.
- Batch Normalization для ускорения обучения нейронных сетей. [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/309302/.