Практическая работа № 6

“Продвинутые архитектуры сверточных нейронных сетей и перенос обучения”

Цель: изучить продвинутые архитектуры сверточных нейронных сетей и изучить задачи которые они решают. Изучить методику переноса обучения.

Skip connections

В настоящее время существует бесконечное количество приложений, которые кто-то может сделать с помощью Deep Learning. Однако, чтобы понять множество вариантов дизайна, таких как пропущенные соединения, которые вы видите во многих работах, очень важно немного понять механизмы обратного распространения.

Если бы вы пытались обучить нейронную сеть еще в 2014 году, вы бы обязательно наблюдали так называемую **проблему исчезающего градиента**. Проще говоря: вы находитесь за экраном, проверяя процесс обучения вашей сети, и все, что вы видите, это то, что потери при обучении перестают уменьшаться, но они все еще далеки от желаемого значения. Вы всю ночь проверяете все строки своего кода, чтобы увидеть, не было ли что-то не так, и не находите никаких зацепок. Не лучший опыт в мире, поверьте!

В основном происходит то, что вы пытаетесь обновить параметры, **изменив их на небольшую величину.** \Дельта w\_{i}Δ *шя*​который был рассчитан на основе градиента, например, предположим, что для раннего слоя средний градиент 1e-15 (ΔL/δw). Учитывая скорость обучения 1e-4 (λ в уравнении), вы в основном изменяете параметры слоя произведением ссылочных величин, что составляет 1e-19 (\Дельта w\_{i}Δ *шя*​). В результате **вы фактически не наблюдаете никаких изменений в модели при обучении сети**. Вот как вы можете наблюдать проблему исчезающего градиента.

Немного заглянув в теорию, можно легко понять проблему исчезающего градиента из алгоритма обратного распространения. Мы кратко рассмотрим алгоритм обратного распространения **через призму цепного правила**, начиная с базового исчисления, чтобы получить представление о пропущенных соединениях.

Короче говоря, обратное распространение — это «магия оптимизации», стоящая за архитектурами глубокого обучения. Учитывая, что глубокая сеть состоит из конечного числа параметров, которые мы хотим изучить, наша цель — итеративно **оптимизировать эти параметры по отношению к функции потерь** L.

Как вы видели, каждая архитектура имеет некоторые входные данные (т. е. изображение) и производит выходные данные (прогноз). Функция потерь сильно зависит от задачи, которую мы хотим решить. На данный момент вам нужно знать, **что функция потерь является количественной мерой расстояния между двумя тензорами**, которые могут представлять метку изображения, ограничивающую рамку на изображении, переведенный текст на другом языке и т. д. Обычно вам нужен какой-то вид контроля для сравнения прогноза сети с желаемым результатом (наземной истиной). Имейте в виду, что обратное распространение относится к категории контролируемого машинного обучения.

Итак, **красивая идея обратного распространения заключается в том, чтобы постепенно минимизировать эту потерю, обновляя параметры сети**. Но **как** можно распространить скалярные измеренные потери внутри сети? Именно здесь в игру вступает обратное распространение.

**Проще говоря, обратное распространение — это понимание того, как изменение весов (параметров) в сети изменяет функцию потерь путем вычисления частных производных.** Для последнего мы используем простую идею цепного правила, чтобы минимизировать расстояние в желаемых предсказаниях. Другими словами, обратное распространение — это **вычисление градиента функции потерь** с учетом различных весов в этой нейронной сети, что является не чем иным, как вычислением частных производных функции потерь по параметрам модели. Повторяя этот шаг много раз, мы будем постоянно минимизировать функцию потерь до тех пор, пока она не перестанет уменьшаться или не будут выполнены некоторые другие предопределенные критерии завершения.

Цепное правило в основном описывает **градиент** (изменение) **функции потерь,** то есть z**, по отношению к некоторому параметру нейронной сети**, скажем, x и y, которые являются функциями параметра предыдущего слоя t. Пусть f, g, h — разные слои сети, выполняющие нелинейную операцию во входном векторе.

Теперь предположим, что вы изучаете исчисление и хотите выразить градиент z по отношению к входным данным. Вот чему вы научитесь в исчислении с несколькими переменными:

Интересно, что знаменитый алгоритм делает точно **такую ​​же операцию, но обратным образом**: он начинает с выхода z и **вычисляет частные производные каждого параметра**, выражая его только на основе градиентов более поздних слоев.

Действительно стоит заметить, что **все эти значения часто меньше 1**, независимо от знака. Чтобы распространить градиент на более ранние слои, обратное распространение использует умножение частных производных (как в цепном правиле). В общем, умножение с абсолютным значением меньше 1 — это хорошо, потому что оно обеспечивает некоторую стабильность обучения, хотя строгой математической теоремы об этом нет. Однако можно заметить, что для каждого слоя, который мы идем назад в сети, **градиент сети становится все меньше и меньше**.

В настоящее время пропуск соединения является стандартным модулем во многих сверточных архитектурах. Используя пропускное соединение, мы предоставляем альтернативный путь для градиента (с обратным распространением). Экспериментально подтверждено, что эти дополнительные пути часто полезны для сходимости модели. **Пропуск соединений** в глубоких архитектурах, как следует из названия, пропускает **какой-то слой в нейронной сети и передает выходные данные одного слоя в качестве входных данных для следующих слоев** (а не только для следующего).

Как объяснялось ранее, используя цепное правило, мы должны продолжать умножать члены на градиент ошибки по мере продвижения назад. Однако в длинной цепочке умножения, если мы умножим много вещей, меньших единицы, то результирующий градиент будет очень маленьким. Таким образом, градиент **становится очень маленьким, когда мы приближаемся к более ранним слоям в глубокой архитектуре**. В некоторых случаях градиент становится нулевым, что означает, что **мы вообще не обновляем ранние слои**.

В общем, есть два основных способа, которыми можно использовать пропущенные соединения через разные непоследовательные слои:

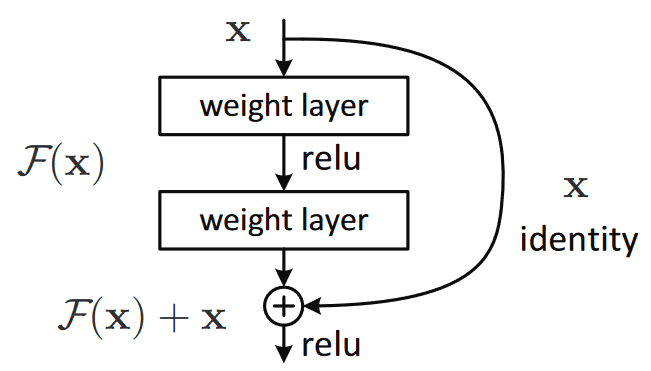
а) **добавление**, как в остаточных архитектурах,

б) **конкатенация**, как в плотно связанных архитектурах.

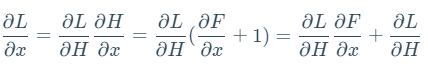
Сначала мы опишем добавление, которое обычно называют соединениями с остаточным пропуском.

ResNet

Основная идея состоит в том, чтобы выполнить обратное распространение через функцию идентификации, просто используя векторное сложение. Тогда градиент будет просто умножен на единицу, и его значение будет сохранено в более ранних слоях. Это основная идея Residual Networks (ResNets): они складывают эти пропущенные остаточные блоки вместе. Мы используем функцию тождества, чтобы **сохранить градиент**.



Математически мы можем представить остаточный блок и вычислить его частную производную (градиент) с учетом функции потерь следующим образом:



Помимо исчезающих градиентов, есть еще одна причина, по которой мы их часто используем. Для множества задач (таких как семантическая сегментация, оценка оптического потока и т. д.) есть некоторая информация, которая была получена на начальных уровнях, и мы хотели бы позволить более поздним уровням также учиться на них. **Было замечено, что на более ранних уровнях изученные признаки соответствуют более низкой семантической информации, которая извлекается из входных данных.** Если бы мы не использовали пропускное соединение, эта информация стала бы слишком абстрактной.

DenseNet

Как уже говорилось, для многих задач плотного прогнозирования существует информация низкого уровня, разделяемая между входом и выходом, и было бы желательно передавать эту информацию напрямую через сеть . Альтернативный способ пропустить соединения — объединить предыдущие карты объектов. Самая известная архитектура глубокого обучения — DenseNet . Ниже вы можете увидеть пример возможности повторного использования функции путем объединения с 5 сверточными слоями:

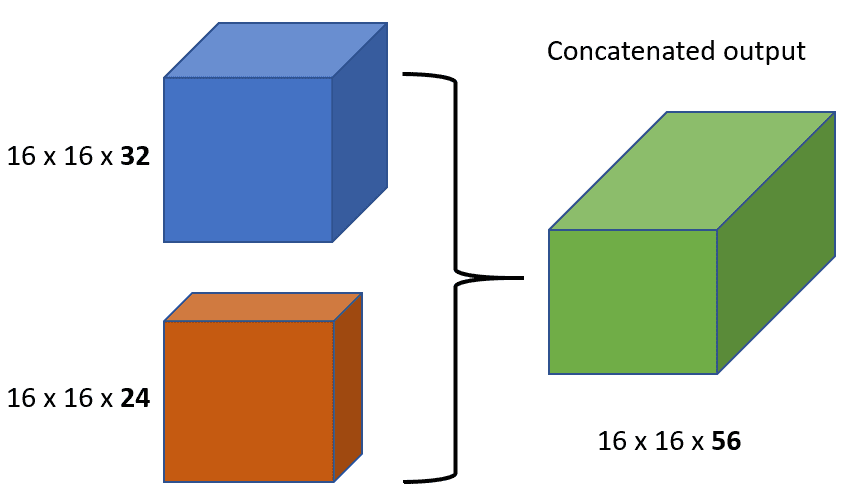


Эта архитектура активно использует конкатенацию функций, чтобы обеспечить максимальный поток информации между уровнями в сети. Это достигается путем **соединения через конкатенацию всех слоев напрямую друг с другом**, в отличие от ResNets. Практически, что вы в основном делаете, так это объединяете измерение функционального канала. Это ведет к

а) **огромное количество функциональных каналов на последних слоях** сети,

б) к более **компактным** моделям,

c) экстремальная **возможность повторного использования функций**.



Перенос обучения

Трансферное обучение является популярным методом в компьютерном зрении, потому что оно позволяет нам **создавать точные модели, экономя время**. С помощью трансферного обучения вместо того, чтобы начинать процесс обучения с нуля, вы начинаете с шаблонов, которые были изучены при решении другой задачи. Таким образом, вы используете предыдущие знания и избегаете начинать с нуля. Примите это как глубокую обучающую версию «стоящий на плече гигантов».

В компьютерном зрении передача обучения обычно выражается с помощью **предварительно обученные модели**, предварительно обученная модель — это модель, которая была обучена на большом наборе эталонных данных для решения проблемы, аналогичной той, которую мы хотим решить. Соответственно, из-за вычислительной стоимости обучения таких моделей обычной практикой является импорт и использование моделей из опубликованной литературы (например,VGG,Inception,MobileNet). Всесторонний обзор эффективности предварительно обученных моделей по проблемам компьютерного зрения с использованием данных из задачи ImageNet (Deng et al. 2009) представлен Canziani et al. (2016).

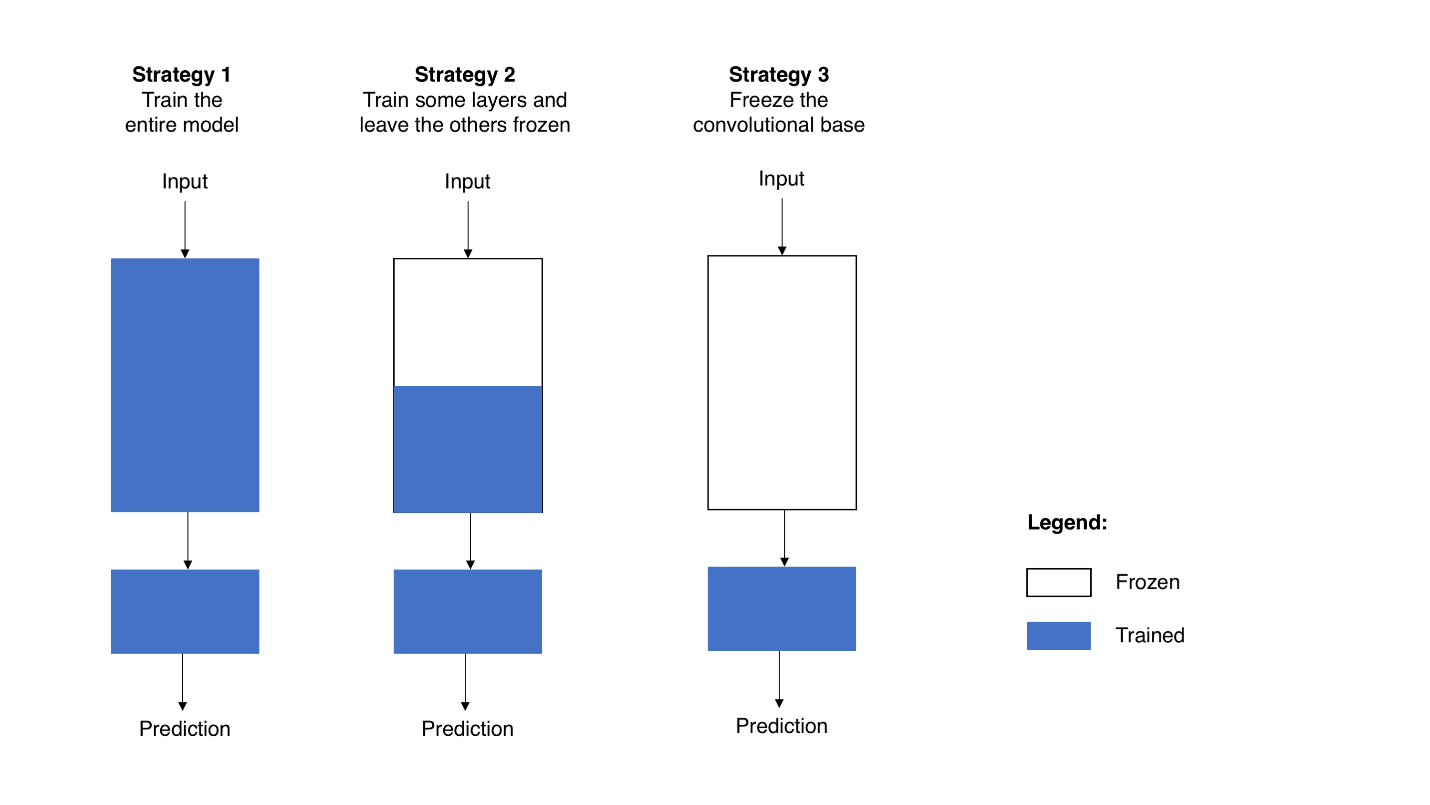
Несколько предварительно обученных моделей, используемых в трансферном обучении, основаны на**сверточные нейронные сети (CNN)** (Вулодимос и др. 2018). В целом было показано, что CNN преуспевает в широком спектре задач компьютерного зрения (Bengio 2009). Его высокая производительность и легкость в обучении являются двумя основными факторами, влияющими на популярность CNN в последние годы.

Типичный CNN состоит из двух частей:

1. **Сверточная база**, который состоит из стека сверточных и пулирующих слоев. Основная цель сверточной базы - генерировать элементы изображения. Интуитивно понятное объяснение сверточных и пулирующих слоев см. В Chollet (2017).
2. **классификатор**, который обычно состоит из полностью связанных слоев. Основная цель классификатора - классифицировать изображение на основе обнаруженных признаков. Полностью связанный слой — это слой, чьи нейроны полностью связаны со всей активацией на предыдущем уровне.

Когда вы настраиваете предварительно обученную модель для своих собственных нужд, вы начинаете с удаления исходного классификатора, затем добавляете новый классификатор, который соответствует вашим целям, и, наконец, вы должны**отрегулируйте свою модель в соответствии с одной из трех стратегий**:

1. **Тренируй всю модель.**В этом случае вы используете архитектуру предварительно обученной модели и обучаете ее в соответствии с вашим набором данных. Вы изучаете модель с нуля, поэтому вам потребуется большой набор данных (и много вычислительных мощностей).
2. **Тренируйте некоторые слои и оставляйте другие замороженными.**Как вы помните, нижние уровни относятся к общим функциям (не зависят от проблемы), в то время как верхние уровни относятся к определенным функциям (зависит от проблемы). Здесь мы играем с этой дихотомией, выбирая, насколько мы хотим отрегулировать вес сети (замороженный слой не изменяется во время обучения). Обычно, если у вас небольшой набор данных и большое количество параметров, вы оставляете больше слоев замороженными, чтобы избежать переобучения. В отличие от этого, если набор данных велик, а количество параметров невелико, вы можете улучшить свою модель, обучив больше слоев новой задаче, поскольку переоснащение не является проблемой.
3. **Заморозить сверточную базу.**Этот случай соответствует экстремальной ситуации компромисса между поездом и заморозкой. Основная идея заключается в том, чтобы сохранить сверточную базу в ее первоначальном виде, а затем использовать ее выходные данные для подачи в классификатор. Вы используете предварительно обученную модель в качестве механизма извлечения фиксированных функций, которая может быть полезна, если у вас недостаточно вычислительных мощностей, ваш набор данных небольшой и / или предварительно обученная модель решает проблему, очень похожую на ту, которую вы используете. хочу решить.



В отличие от **Стратегия 3** чья заявка **простой**, **Стратегия 1,** а также **Стратегия 2** требует от вас **быть осторожным** со скоростью обучения, используемой в сверточной части. Скорость обучения — это гиперпараметр, который контролирует, насколько вы корректируете веса вашей сети. Когда вы используете предварительно обученную модель, основанную на CNN, разумно использовать небольшую скорость обучения, потому что высокая скорость обучения увеличивает риск потери предыдущих знаний. Предполагая, что предварительно обученная модель была хорошо обучена, что является справедливым допущением, сохранение небольшого уровня обучения гарантирует, что вы не будете искажать веса CNN слишком рано и слишком много.