Борьба с переобучением в нейронных сетях. Часть 1

Охотин Андрей

30.05.2021

Содержание

Аннотация

Нейронные сети являются сложными моделями последовательных пространственных преобразований с множеством оптимизируемых параметров. Поэтому проблема переобучения в таких моделях занимает особое место. Нейронные сети могут легко подстраиваться под тренировочную выборку из-за своей функциональной выразительности, но при этом показывать несопоставимо низкое качество на тесте. В этой работе будут описаны возможные причины возникновения таких эффектов и методы борьбы с ними.

Проблема переобучения в нейронных сетях

Переобучение (на англ. "overfitting") – одна из наиболее известных проблем глубоких нейронных сетей (Deep Neural Networks). Простыми словами, эту проблему можно описать так: сеть показывает хорошее качество на тренировочной выборке и плохое на тестовой. Эту проблему можно также описать, как низкую «обобщающую способность» модели, т.е. способность сети выделять в данных определенные паттерны, соответствующие всему множеству данных такого типа (а не только тем данным, которые мы имеем на данный момент). Обучение модели таким паттернам является нетривиальной задачей, для решения которой используются различные подходы.

Классификация методов

В способах борьбы с переобучением можно выделить следующие типы:

1. Модификация данных вне модели. К этому типу можно отнести различные виды нормализации входных данных, а также методы аугментации. Все они направлены на изменение тренировочной выборки так, чтобы обучающаяся на ней модель повышала свою обобщающую способность.

2. Функциональная модификация. Этот тип подразумевает изменение нейронной сети как функции последовательных преобразований. К нему можно отнести различные виды специальных архитектур, слои Batch Noramlization, Layer Normalization и тд. Все они влияют на функциональную выразительность сети, а также могут учитывать определенные априорные предположения о данных.

3. Модификация процесса оптимизации. Этот тип представлен наибольшим числом методов: различные способы инициализации, как определения стартовой точки процесса оптимизации; Dropout, как способ противодействия кооперативной адаптации нейронов в процессе обучения; оптимизаторы и гиперпараметры, например, темп обучения и размер батча, позволяющие сети эффективнее находить локальные минимумы; регуляризаторы, изменяющие поверхность функции ошибки, чтобы в процессе оптимизации модель сходилась в определенные области пространства весов.

Представленная классификация позволяет систематизировать различные подходы к решению проблемы переобучения в зависимости от их зоны влияния.

Обработка входных данных

Нормировка

Пусть мы имеем признак:

$$X = (X_0, X_1, ..., X_m)$$

среднее и дисперсия данного признака считаются по формулам соответственно:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} X_i, \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (X_i - \mu)\,.$$

Тогда формула для нормировки будет следующая:

$$X = \frac{X - \mu}{\sqrt{\sigma^2}}.$$

Мотивация введения такого преобразования входных данных в том, что математическое ожидание не несет в себе информации. При рассмотрении того или иного признака у группы объектов, информацию об объекте для нас несет именно то, как он отличается от других объектов. Мотивацию деления на стандартное отклонение можно описать по похожему принципу. Информация об отличии одного объекта от остальных заключается в том, насколько оно велико относительно отклонений остальных объектов в этой группе. Дисперсия содержит в себе информацию о среднем (среднеквадратичном) отклонении в группе, поэтому при делении на неё, мы убираем лишнюю информацию об абсолютном значении отклонения.

Более строгое обоснование того, почему нейросетевые модели инварианты к таким преобразованиям, заключается в следующем. Для линейного преобразования признака $aX + b$ будущая дисперсия признака квадратично зависит от коэффициента a, а матожидание – линейно от смещения b. В нейронных сетях параметры линейных преобразований (a и b) являются оптимизируемыми, а значит нормировка преобразует поверхность функционала качества, заданного на пространстве оптимизируемых параметров (если в функционал качества не добавлены регуляризаторы, зависящие от весов, все локальные минимумы просто смещаются). Другая причина использования состоит в повсеместном использовании разных регуляризаторов. Нормировка в свою очередь, значительно упрощает подбор их гиперпараметров. Иначе говоря, производя нормировку, мы, на основании описанных предположений о том, что именно несет для нас информацию об объекте, упрощаем поиск оптимумов нейронной сети.

Инициализация весов нейронной сети

Инициализация нейронной сети фактически является выбором стартовой точки, из которой модель начнет свой пошаговый поиск минимума. При инициализации весов учитываются следующие пункты:

1. Симметричность нейронов приводит к выделению схожих признаков в сети, что сокращает многообразие найденных комбинаций. Мы хотим, чтобы сеть изначально «по-разному» рассматривала наши данные, из-за чего увеличивается вероятность найти информативный признак. В том числе это позволяет эффективнее использовать функциональную выразительность («сложность» с точки зрения числа параметров) нейронной сети.

2. Насыщенность нейрона. Большие изначальные значения весов приводят к тому, что сеть стартует обучение из областей с крайне малыми градиентами, что значительно замедляет процесс оптимизации.

3. Равенство дисперсий. Так как линейные преобразования квадратично преобразуют дисперсию данных, сеть может легко увеличивать дисперсию, что в свою очередь, может привести к насыщению нейронов.

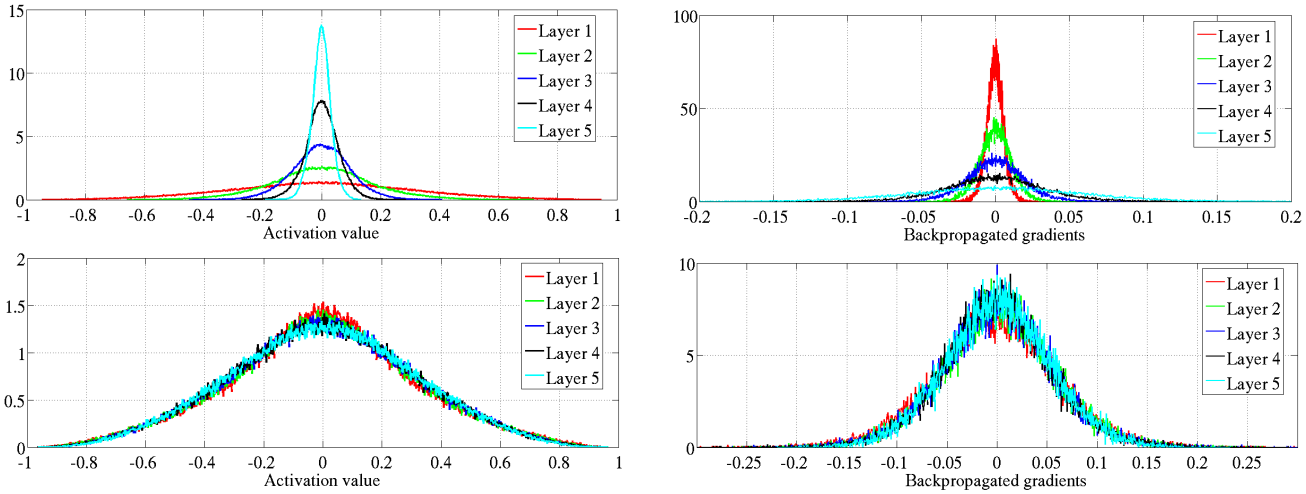
Замечание: смещение, роль которого состоит в сдвиге активаций нейронной сети в нужный промежуток, традиционно полагается 0. Это происходит из соображений того, что не имеет смысла стартовать из точки, в которой смещения уже задают определенные сдвиги активаций.

Xavier (Glorot)

Этот метод нацелен на сохранение дисперсии при последовательных линейных преобразованиях. Инициализируя сеть из определенного распределения (нормального или равномерного), с дисперсией, зависящей от размерностей входа $n\_in^{(k)}$ и выхода $n\_out^{(k)}$ (k – номер слоя):

$$Dw_{ij}^{(k)} = \frac{2}{n_{in}^{(k)} + n_{out}^{(k)}}$$

Усреднение между размерностями входа и выхода берется из соображений сохранения дисперсии не только при прямом, но и при обратном проходе. То, как меняется распределение активаций при смене обычной инициализации на «нормализованную» (предлагаемое в оригинальной статье [1] название) изображено на графиках ниже:



Распределение активаций при прямом проходе и градиентов при обратном, для обычной инициализации и Xavier [1] p.254

На графиках видно, как при обычной инициализации дисперсия увеличивается при прохождении следующего слоя. Итоговая формула инициализации весов выглядит следующим образом:

$$w_{ij}^{(k)} \sim norm\left[0, \frac{2}{n_{in}^{(k)} + n_{out}^{(k)}}\right]$$

$$w_{ij}^{(k)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}^{(k)} + n_{out}^{(k)}}}, +\frac{\sqrt{6}}{\sqrt{n_{in}^{(k)} + n_{out}^{(k)}}}\right]$$

Kaiming (He)

Инициализация Xavier лучше подходит для сетей с активациями Sigmoid. Специально для функции активации ReLU была разработана немного измененная инициализация He [3], основные цели которой

1. бороться с затухающими градиентами, т.е. с остановкой параметров в областях с нулевыми градиентами

2. предотвращать «взрыв» градиентов – накопление градиента, приводящее к очень большим значениям, что делает процесс обучения нестабильным

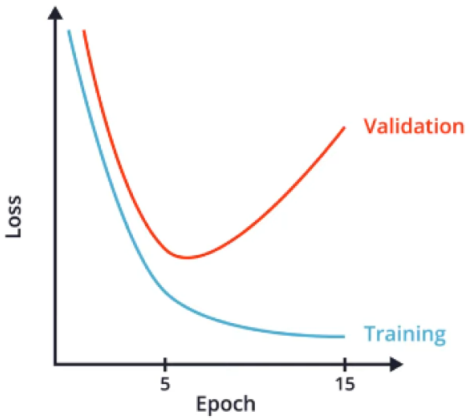
Обе проблемы возникают из-за экспоненциального характера накопления градиента. Если на многих последовательных слоях норма градиента близка к 0, то происходит затухание, а если больше 1, то взрыв. Для борьбы с этими эффектами предлагается инициализация следующего вида:

$$w_{ij}^{(k)} \sim U\left[-\frac{\sqrt{3}}{\sqrt{n_{in}^{(k)} + n_{out}^{(k)}}}, +\frac{\sqrt{3}}{\sqrt{n_{in}^{(k)} + n_{out}^{(k)}}}\right]$$

В оригинальной статье приводится пример того, как модель с 30 слоями не может сойтись при инициализации Xavier, но при это сходится при использовании He.

Верификация

При обучении нейронных сетей часто возникает ситуация, что ошибка на валидационной выборке начинает расти, но при этом на тренировочной – ошибка продолжает уменьшаться. Такое происходит, когда модель, чтобы уменьшить ошибку, начинает находить локальные закономерности, которых не существует в тестовой выборке. Из-за этого падает обобщающая способность сети, что для нас является крайне нежелательным. Поэтому для нейронных сетей особенно важно в нужный момент остановить обучение. Чтобы этого добиться и избежать слишком ранней остановки обучения можно сохранять состояния модели и оптимизировать веса до сходимости.



Траектория функционала потерь на тренировочной и валидационной выборках

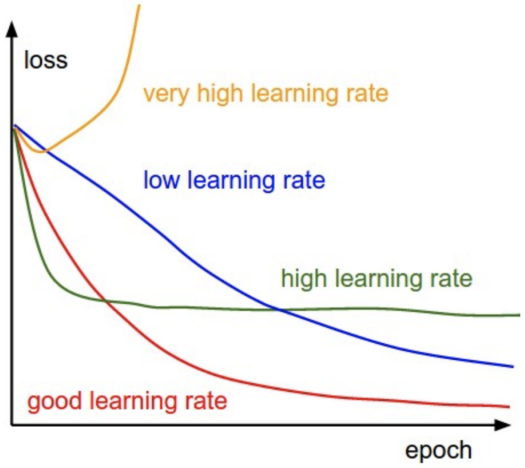
https://habr.com/ru/post/458170/

Далее по траектории функционала потерь на валидационной выборке можно определить наилучшее с точки зрения качества состояние модели.

Темп обучения (Learning rate)

Темп обучения является одним из важнейших гиперпараметров при обучении нейронных сетей. От темпа обучения напрямую зависит то, какого размера минимумы мы не хотим пропускать. Размер шагов оптимизации во многом зависит от оптимизатора, поэтому настройку темпа обучения лучше всего понимать в следующем виде:

1. Уменьшая темп обучения, мы хотим чтобы модель чаще учитывала более узкие минимумы.
2. Увеличивая темп обучения, мы хотим чтобы модель чаще пропускала более узкие минимумы



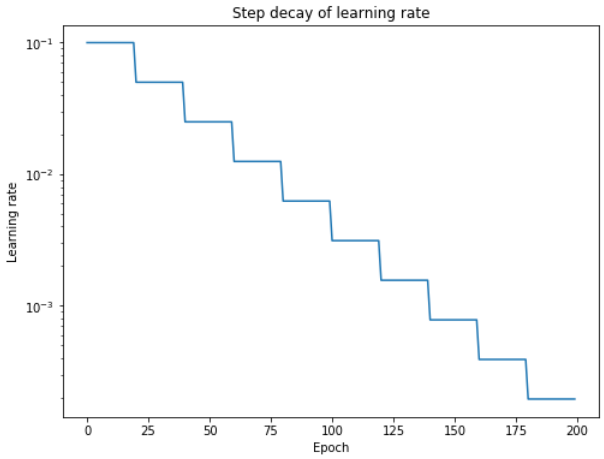
Пример влияния выбора learning rate на функционал ошибки модели

https://www.machinelearningmastery.ru/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594/

Как правило, темп обучения подбирается эвристически. В больших моделях, показавших в

свое время рекорды по качеству на классических для глубокого обучения задачах, зачастую не использовались хитрые методы автоматического подбора темпа обучения. Логику изменения шагов обновления весов сети по большей части на себя берет оптимизатор, о чем будет подробнее описано в соответствующей секции.

Существующие методы изменения темпа обучения основаны на выполнении определенных условий на номер эпохи и/или значения функционала потерь. Существует метод, основанный на снижении темпа обучения после каждого определенного числа эпох (Learning Rate Annealing). Метод понижения темпа обучения, когда функционал ошибки определенное время почти не менялся, имеет название Learning Rate Schedule.



Learning Rate Annealing: step decay

https://www.jeremyjordan.me/nn-learning-rate/

Интуицию таких алгоритмов автоподбора темпа обучения можно описать так: на первых этапах мы хотим большими шагами двигаться в сторону минимума, пропуская все локальные экстремумы, а на последних этапах – маленькими шагами спуститься в конкретный узкий минимум. Важно также помнить, что существует возможность использовать различные темпы для разных групп оптимизируемых параметров.

Основная дилемма состоит в следующем. Если темп обучения будет слишком маленький, модель быстро застрянет в локальном минимуме, который скорее всего далек от оптимального. А при большом темпе, модель, при попытке спуститься в определенный локальный минимум, будет из него «выскакивать». Так или иначе, подбор темпа обучения пока является эвристической задачей, решение которой для каждого конкретного кейса может отличаться.

Разбиение на батчи

Мини-батч оптимизация является одним из частых методов обучения нейронных сетей. Обычный градиентный спуск при современных объемах данных является технически неподъемной задачей. При этом обучение по одному объекту может быть чересчур неэффективным. Предлагается разделить тренировочную выборку на отдельные подвыборки (батчи), и уже по ним вычислять усредненный градиент.

$$w^{(t+1)} = w^{(t)} - \frac{\eta}{|I|} \sum_{i \in I} \nabla \left[l\left(a(x_i \mid w^{(t)}), y_i\right) + \lambda R\left(w^{(t)}\right) \right]$$

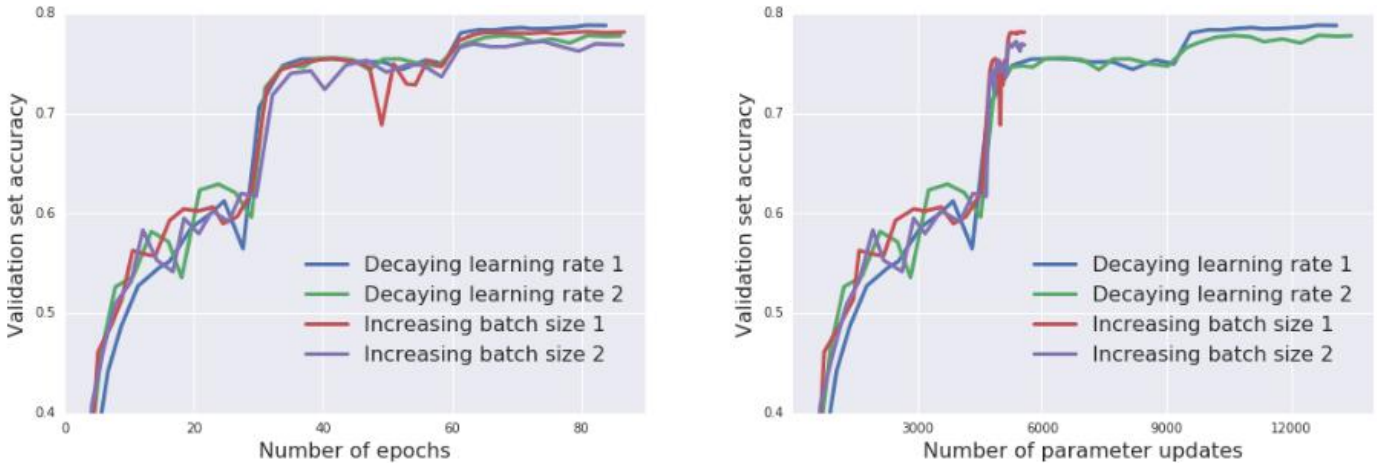
При организации обучения нейронной сети важным моментом является выбор структуры батча.

Размер батча

Размер батча выбирается не только из соображений того, чтобы оптимизация весов была менее случайной, но и с учетом ресурсоемкости процесса. Здесь появляется дилемма, схожая с описанной в разделе про темп обучения.

1. Увеличение размера батча делает процесс обучения более стабильным, и, по идее, должно положительно сказываться.

2. Уменьшение размера батча добавляет стохастичности процессу обучения, что позволяет модели эффективнее избегать застревания в локальных минимумах.



Сопоставление увеличение размера батча с уменьшением темпа обучения.

https://arxiv.org/pdf/1711.00489.pdf, p.7

Существуют различные исследования о том, как влияет размер батча на процесс сходимости нейронной сети. В одной из статей был обнаружен эффект, что увеличение размера батча обладает влиянием, схожим с уменьшением темпа обучения. В другой статье [3], предлагается идея того, что брать батчи размером более 32 объектов не имеет смысла. Тем не менее большие размеры батча используются на практике для эффективной загрузки графических ускорителей. Описанные примеры влияния на процесс сходимости показывают, что размер батча является одним из важных гиперпараметров при обучении нейронной сети. Так как он напрямую воздействует на скорость сходимости, как и темп обучения, их тестирование на конкретной задаче должно происходить совместно.

Организация батча

Одним из важных моментов является распределение исходной выборки по батчам. Например, стратегия заполнения батча объектами разных классов является одной из самых распространенных. Можно попробовать использовать и более хитрые методы организации. Например, если при многоклассовой классификации мы имеем два схожих класса, можно задавать отдельные батчи, состоящие из объектов этих двух классов. Тем самым модель будет учиться правильно классифицировать эти два класса одновременно и, возможно, эффективнее их разделит. В

любом случае наполнение батча объектами должно выполняться не случайным образом (например просто в лексикографическом порядке названий), а по определенной стратегии. Например, особенно важно перемешивать объекты после каждой эпохи (эпоха – проход по всей обучающей выборке).

Оптимизаторы

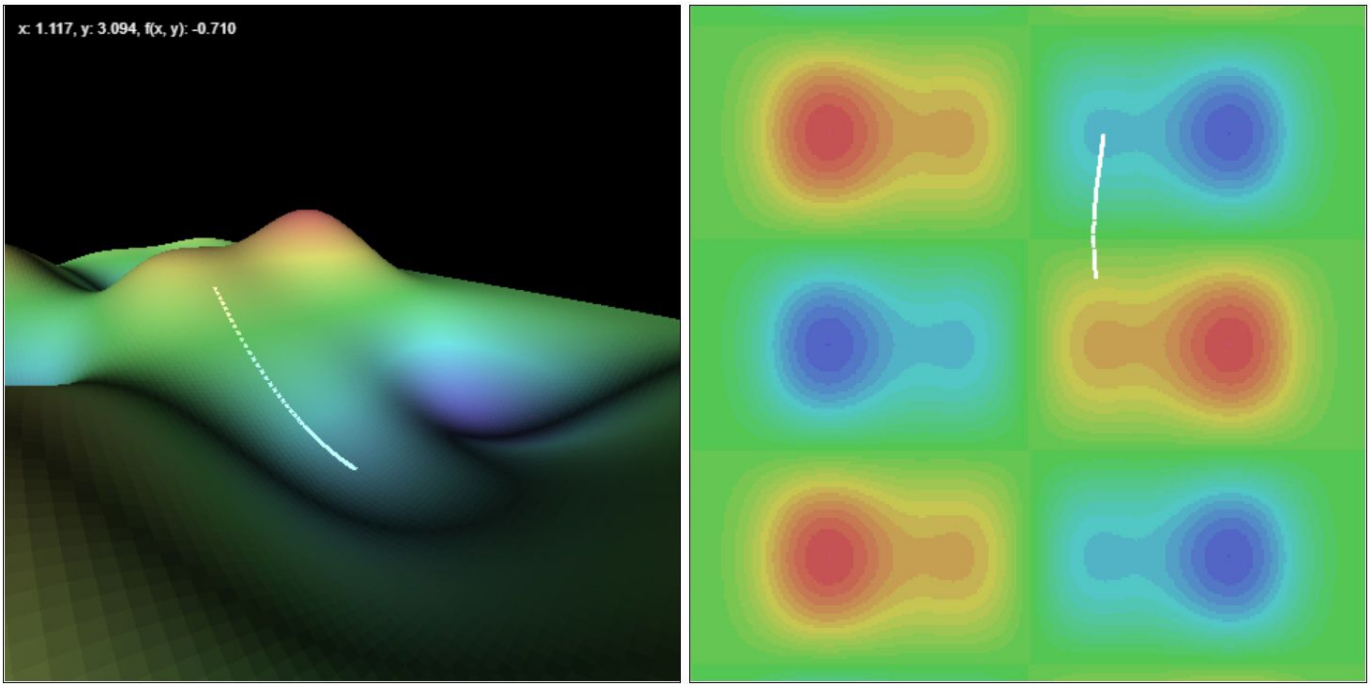
Оптимизаторы, в наиболее общем смысле, являются алгоритмами, определяющими шаг обновления весов на основе градиентов со всех итераций по текущую включительно. Сохранение такого массива данных является невозможным, поэтому вместо буквального использования градиентов со всех итераций, по ним вычисляются определенные величины, использующиеся в последующих шагах. Таким образом сохраняется «память» с предыдущих шагов. Так можно наиболее обще описать то, чем являются оптимизаторы в современном глубоком обучении. Оптимизаторы существуют для того, чтобы эффективнее искать минимумы при разном виде поверхности функционала ошибки. Например, функционал может иметь седловые точки или сильно вытянутые линии уровня. Для решения этих проблем, вместе с разгоревшимся интересом к глубокому обучению, стали разрабатываться новые алгоритмы оптимизации. Рассмотрим подробнее то, какие существовали оптимизаторы, и как трансформировались их основные идеи.

Для методов построена визуализация с помощью интернет-ресурса [5]. Оптимизировалась функция:

$$f(x, y) = \left(\sin\frac{\pi x}{10} + \frac{1}{5}\sin\frac{4\pi x}{5}\right)\cos y, \quad S_0 = (1, 1),$$

, где S_0 – стартовая точка. Все методы запускались с одинаковым темпом обучения $\eta = 0.1$ на 60 итераций.

Стохастический градиентный спуск, SGD

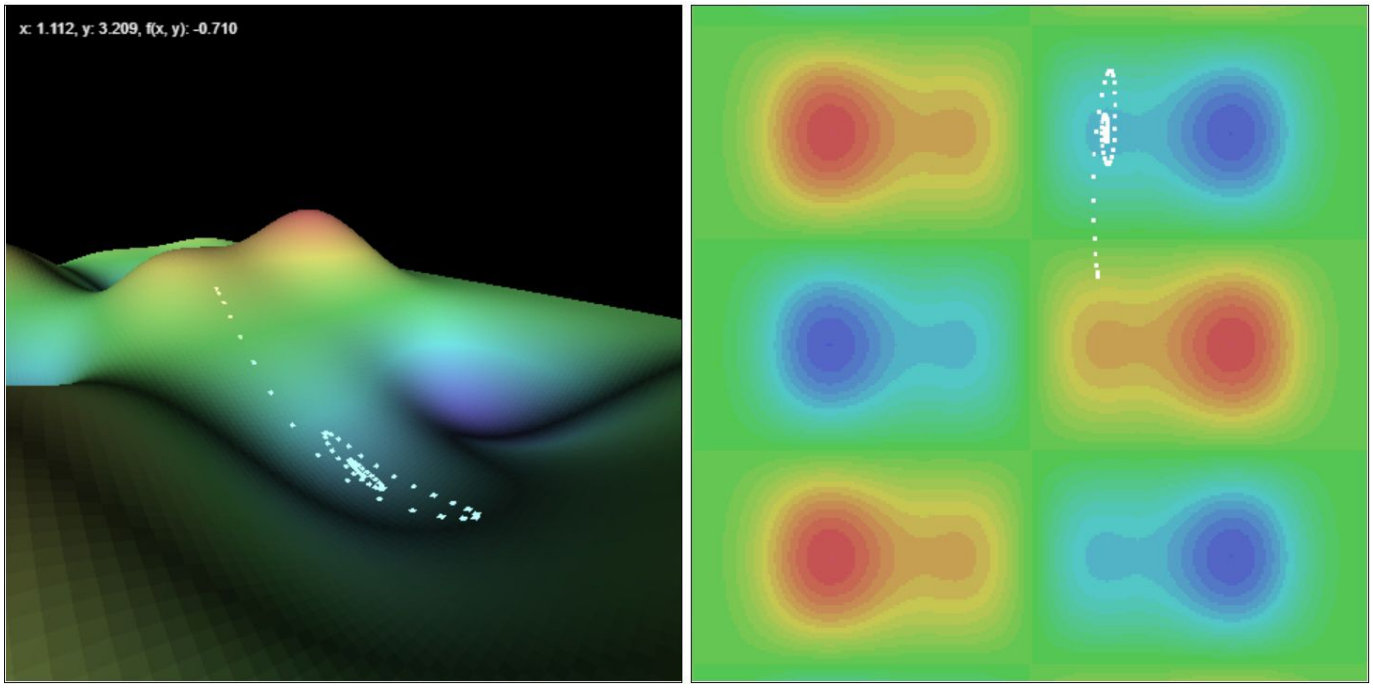


Траектория метода оптимизации SGD

$$w^{(t+1)} = w^{(t)} - \eta \nabla L^{(t)} \left(w^{(t)} \right)$$

Первый и наиболее известный оптимизатор, отличающийся от GD использованием градиента только по одному входному объекту. Существует теоретическое обоснование того, что этот метод лучше GD находит плоские минимумы (которые лучше с точки зрения обобщающей способности сети). Одной из проблем стохастического градиента является его непостоянность от итерации к итерации. Простыми словами: «на каждой новой итерации градиент может смотреть в новую сторону». Для борьбы с такими эффектами был разработан следующий метод.

Стохастический градиент с моментом, momentum

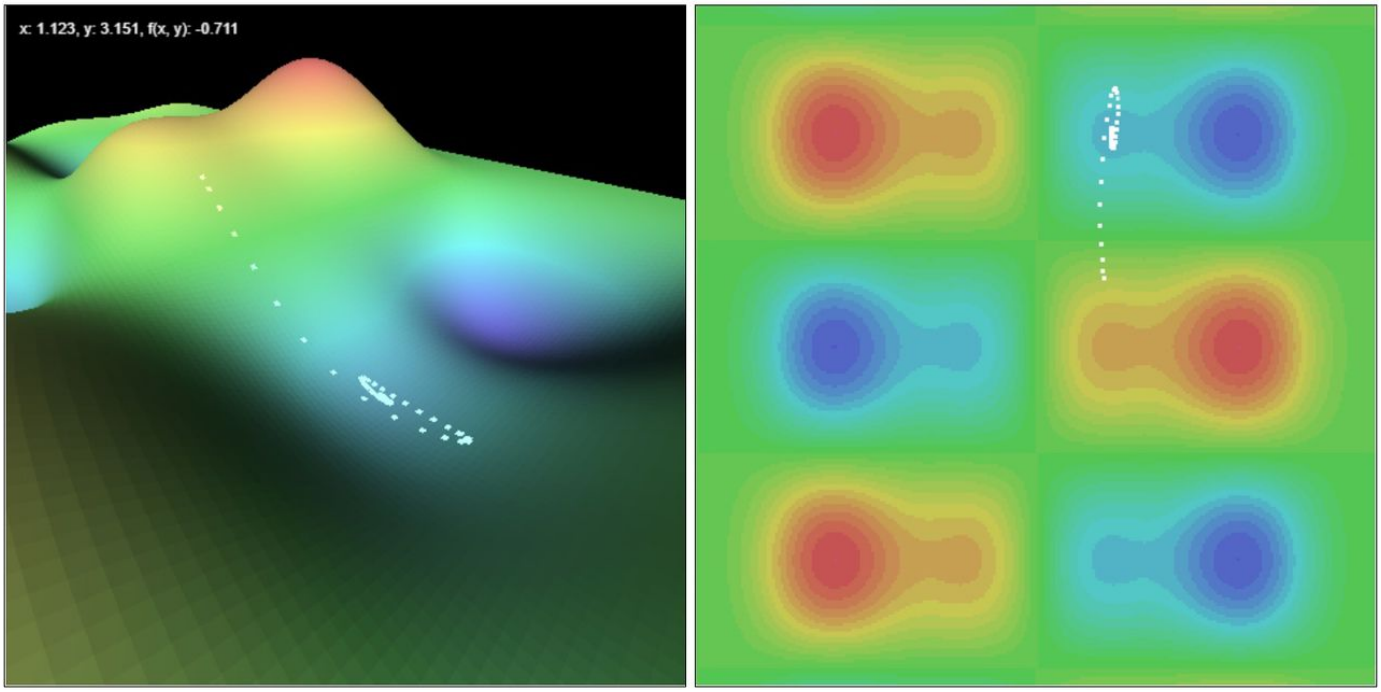


Траектория метода оптимизации momentum

$$m^{(t+1)} = \rho m^{(t)} + \nabla L^{(t)} \left(w^{(t)}\right)$$
$$w^{(t+1)} = w^{(t)} - \eta m^{(t+1)}$$

Этот метод использует накопление градиента $m^{(t)}$, можно сказать для сохранения инерции при движении модели по функционалу ошибки. Предполагается, что таким образом модель будет избегать застревания в неглубоких минимумах. Коэффициент накопления становится еще одним гиперпараметров оптимизатора, помимо темпа обучения. Однако такой вид накопления в итоге может только ухудшить сходимость. После области с большими градиентами модель может слишком большими шагами проскочить все локальные минимумы. Другими словами, в этом алгоритме не хватает механизма погашения инерции.

Метод Нестерова

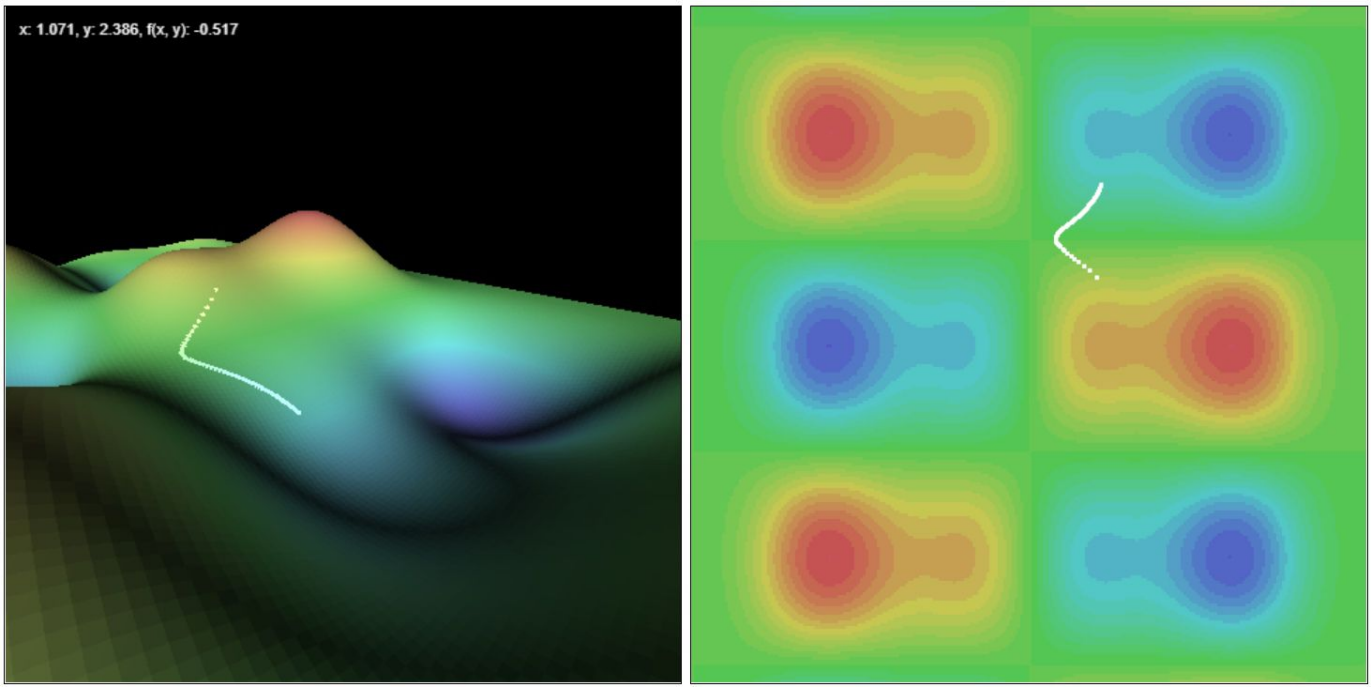


Траектория метода оптимизации Nesterov

$$m^{(t+1)} = \rho m^{(t)} + \nabla L^{(t)} \left(w^{(t)} - \eta m^{(t)} \right)$$
$$w^{(t+1)} = w^{(t)} - \eta m^{(t+1)}$$

Отличие от предыдущего метода здесь заключается в том, что накапливаемый градиент считается не в текущей точке, а в той, куда мы попадем, сделав шаг. Можно попробовать интуитивно объяснить, чем отличается этот метод. Такой накопленный вектор как бы «тянет» нашу модель. Если далее по направлению будет резкий спуск, то такая «будущая» инерция может направить изменение модели в сторону этого спуска. Этот метод, как и momentum помогает проскакивать седловые точки. Здесь также не решена проблема погашения инерции после областей с большим градиентом.

Adagrad

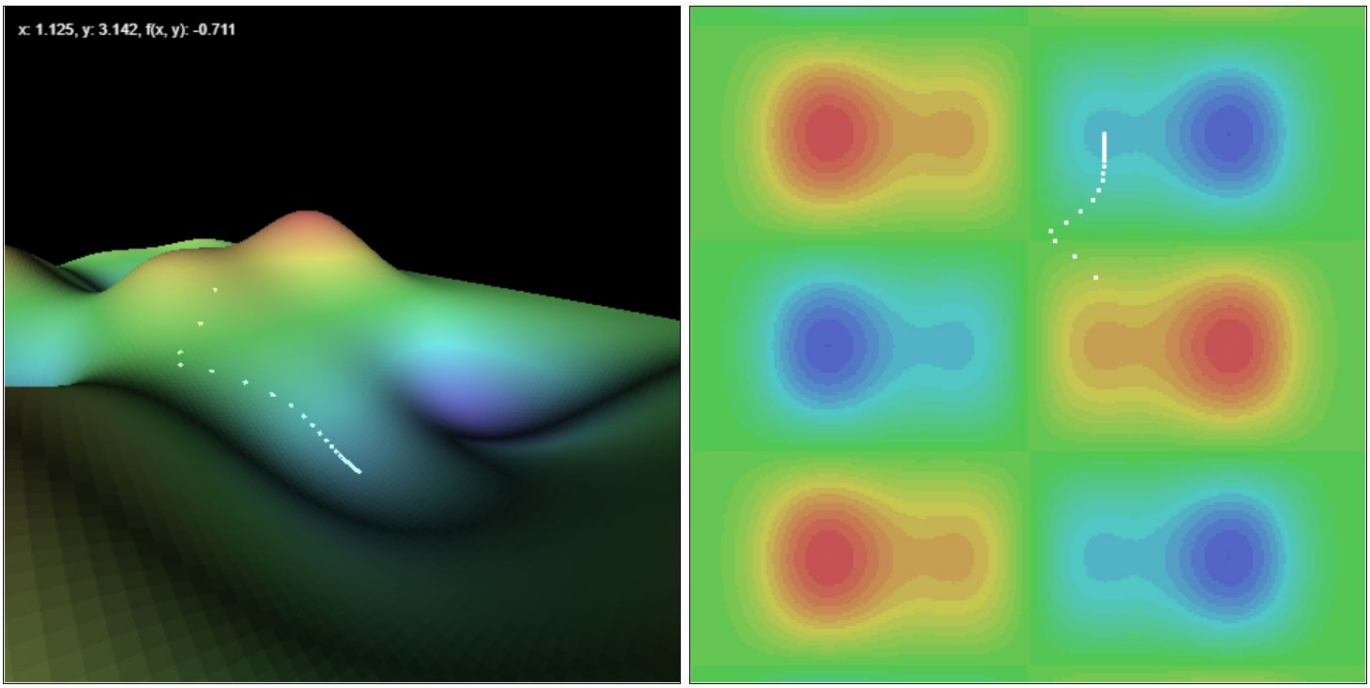


Траектория метода оптимизации Adagrad

$$\nu_i^{(t+1)} = \rho\nu_i^{(t)} + \left(\nabla_i L^{(t)}\left(w^{(t)}\right)\right)^2$$
$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{\nu^{(t+1)} + \epsilon}}\nabla_i L^{(t)}\left(w^{(t)}\right)$$

Смысл метода в том, чтобы накапливать квадрат градиента и далее нормировать градиент в точке на это значение. Таким образом, шаги градиента будут более равномерны. ϵ - нужен для численной подстраховки. При больших значениях градиента модель не слишком сильно разгонится, а при малых, не будет застревать на одном месте. Тем не менее это немного противоречит логике, так как модель будет дольше сходиться. Поэтому этот метод был преобразован к другому виду.

RMSprop



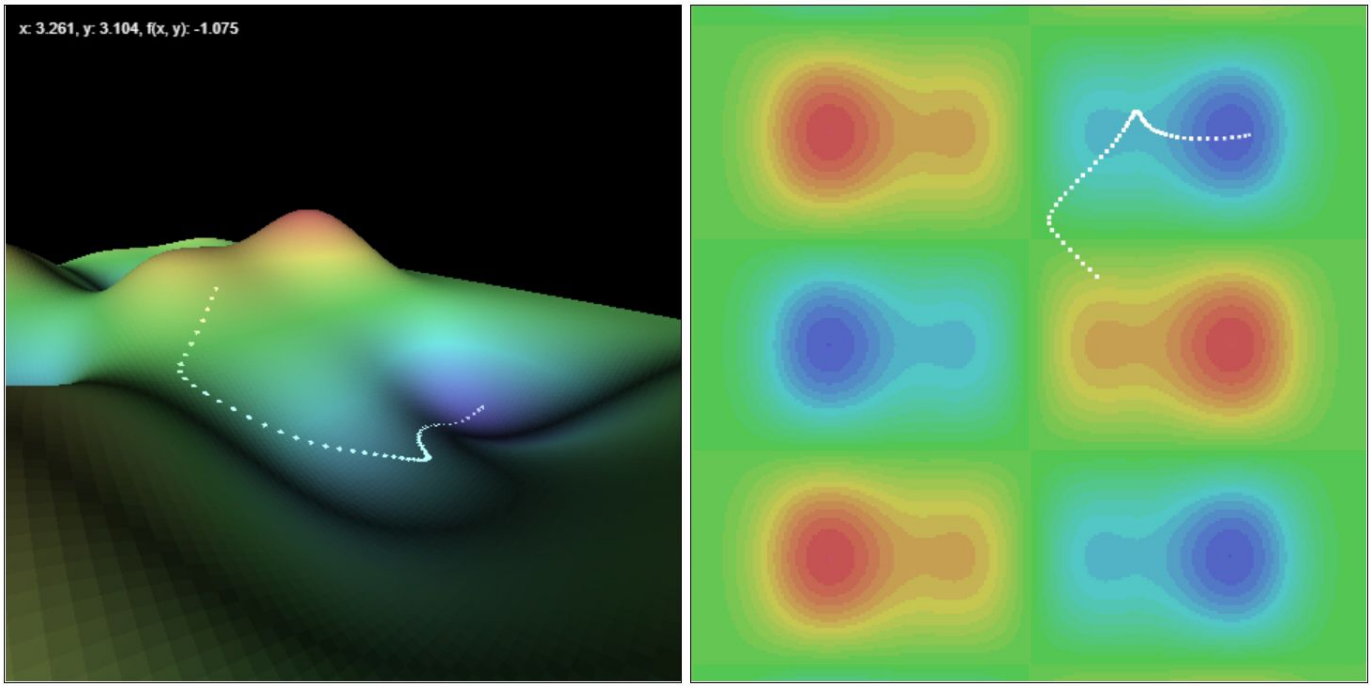
Траектория метода оптимизации RMSprop

$$\nu_i^{(t+1)} = \beta \nu_i^{(t)} + (1-\beta)\left(\nabla_i L^{(t)}\left(w^{(t)}\right)\right)^2$$
$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{\nu^{(t+1)} + \epsilon}} \nabla_i L^{(t)}\left(w^{(t)}\right)$$

Является видоизмененным предыдущим методом. Здесь из-за экспоненциального среднего в первой формуле происходит эффект «забывания», что предотвращает сильное продолжительное замедление изменения параметров после прохождения участка с большими градиентами. Этот эффект можно пронаблюдать на графиках. RMSprop изначально делает шаги большей длины, чем Adagrad. И только при приближении к более плоской области замедляется.

Соответственно, может возникнуть предположение, что если бы этот метод накопил инерцию после крутого спуска, то смог бы добраться до окрестности минимума. Эти предположения приводят нас к следующему методу.

Adam

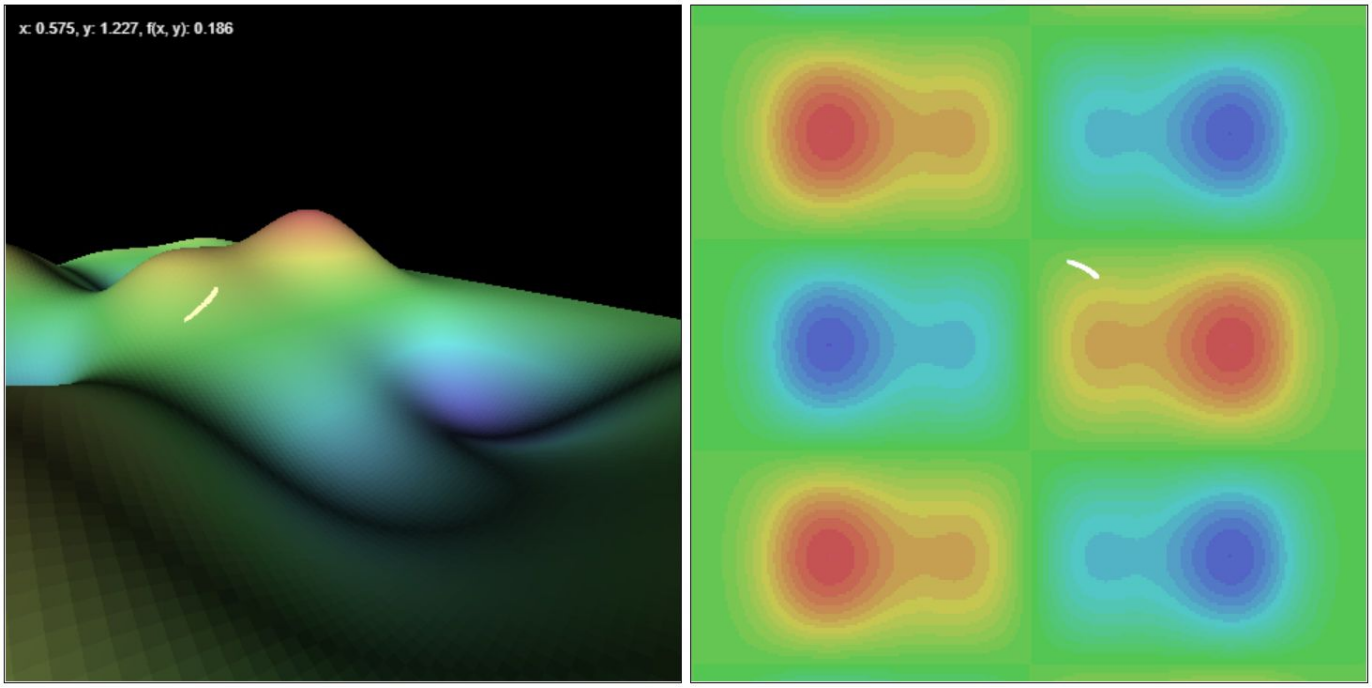


Траектория метода оптимизации Adam

$$m_i^{(t+1)} = \alpha m_i^{(t)} + (1-\alpha)\nabla_i L^{(t)}\left(w^{(t)}\right)$$
$$\nu_i^{(t+1)} = \beta \nu_i^{(t)} + (1-\beta)\left(\nabla_i L^{(t)}\left(w^{(t)}\right)\right)^2$$
$$\hat{m}^{(t+1)} = \frac{m^{(t+1)}}{1-\alpha'},\ \hat{\nu}^{(t+1)} = \frac{\nu^{(t+1)}}{1-\beta'}$$
$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{\hat{\nu}^{(t+1)}+\epsilon}}\hat{m}_i^{(t)}$$

Этот метод является наиболее популярным и используется наиболее активно. Здесь совмещены идеи сохранения инерции и нормировки на накопленную норму, т.е. $Adam = RMSprop + momentum$. Таким образом, этот метод выбирает размер и направление шага под влиянием предыдущих квадратов градиентов, не давая модели на крутых спусках сильно разгоняться, а в областях с малым наклоном замедляться. При этом сохраняется адаптивное поведение momentum при смене областей с разным направлением и градусом наклона. Среди всех протестированных методов является наиболее быстрым в плане сходимости.

Adadelta



Траектория метода оптимизации Adadelta

$$\nu_i^{(t+1)} = \nu_i^{(t)} + \left(\nabla_i L^{(t)}\left(w^{(t)}\right)\right)^2$$

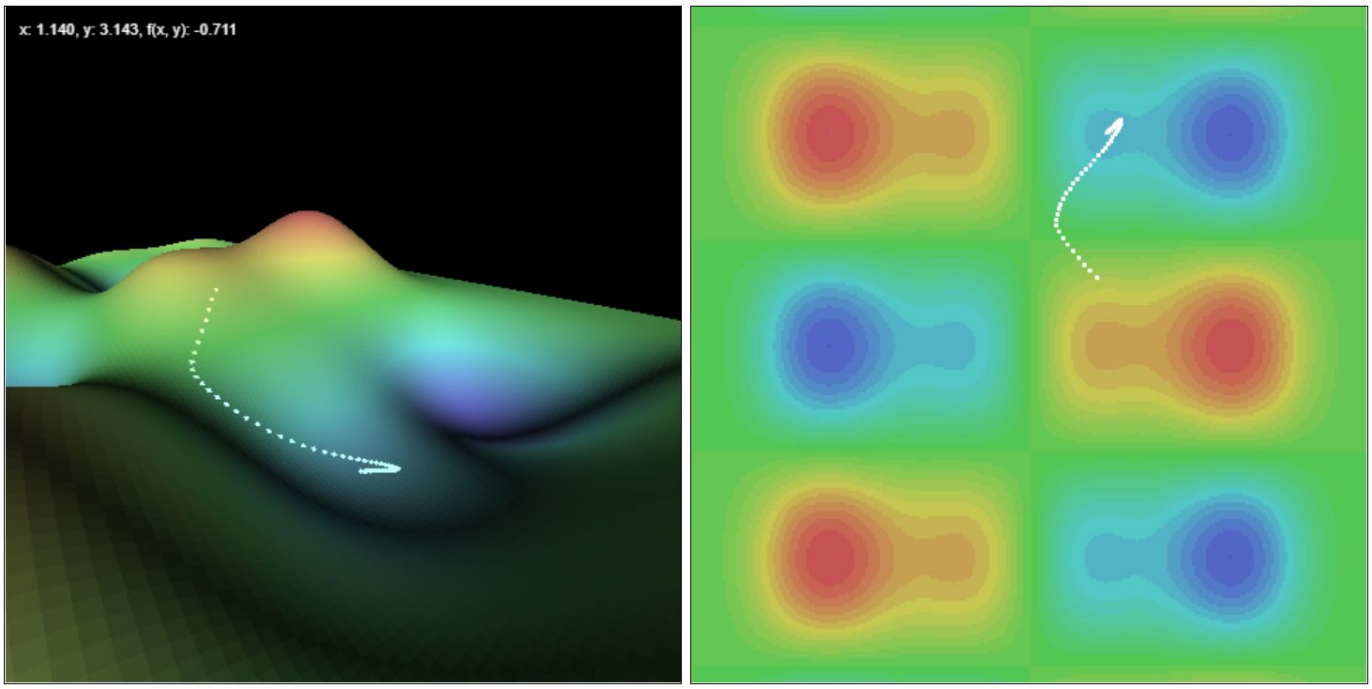
$$\Delta w_i^{(t+1)} = -\frac{\sqrt{\eta_i^{(t)}+\epsilon}}{\sqrt{\nu_i^{(t+1)}+\epsilon}}\nabla_i L^{(t)}\left(w^{(t)}\right)$$

$$w_i^{(t+1)} = w_i^{(t)} + \Delta w_i^{(t+1)}$$

$$\eta_i^{(t+1)} = \gamma\eta_i^{(t)} + (1-\gamma)\left(\Delta w_i^{(t+1)}\right)^2$$

Является модификацией Adagrad. В этом методе момент введен для уже нормированного градиента и вычисляется после обновления весов. При этом накапливается не сам момент, а его квадрат. Экспоненциальное среднее используется только для накопления момента. Накопление нормировки вычисляется как в методе Adagrad. Этот подход также балансирует между нормировкой и инерцией, но немного в другом виде. В этом методе речь идет скорее не о балансе, а о накоплении инерции внутри Adagrad. На визуализации сходимости кажется, что метод работает даже хуже SGD, но в данном случае только по скорости. Для сложных методов оптимизации, таких как Adadelta, скорость сходимости может сильно меняться в зависимости от конкретного функционала, так как внутри них используются различного рода «накопления» инерции. В данном случае ему нужен более высокий темп обучения и больше итераций, чтобы он показал удовлетворительный результат.

Adamax



Траектория метода оптимизации Adamax

Формулы для вычисления нормировки:

$$\nu_i^{(t+1)} = \beta\nu_i^{(t)} + (1-\beta)\left(\nabla_i L^{(t)}\left(w^{(t)}\right)\right)^2$$
$$\hat{\nu}^{(t+1)} = \frac{\nu^{(t+1)}}{1-\beta'}$$

заменяются на:

$$\nu_i^{(t+1)} = max\left[\beta\nu_i^{(t)}, |\nabla_i L^{(t)}\left(w^{(t)}\right)|\right]$$

Адаптация метода Adam, но с использованием бесконечной нормы. В итоге получается достаточно простой и стабильный алгоритм. В оригинальной статье он приведен как метод, схожий с Adam. Особенностью данного алгоритма является то, что значение p нормы устремляется в бесконечность. Использование норм с конечными большими p в описанном алгоритме является численно неустойчивым. Поэтому бесконечная норма выступает, как взгляд на Adam со стороны возможной эффективности больших значений p. Так или иначе, преимущества Adamax перед обычным Adam в статье не описаны.

Meta-ML

Отдельно можно рассмотреть тему мета-обучения. Описанные в этой работе гиперпараметры модели, вроде темпов обучения, коэффициентов накопления момента и нормировки в оптимизаторе Adam подбираются вручную, чтобы обеспечить модели попадание в такой минимум, в котором обобщающая способность будет наибольшей среди возможных. Эти параметры в значительной степени влияют на качество минимумов, в которые в итоге попадает модель. В качестве обобщения методов обучения нейронной сети может выступать Meta-ML.

Для начала определим, что такое мета-обучение. Самым простым и наиболее известным выражением, обозначающим, чем занимается мета-обучение, является фраза: «learn how to learn», т.е. «учимся тому, как учить». По факту, целью мета-обучения является не конкретная модель с хорошим качеством, а «учитель» способный эффективно обучать модель определенной архитектуры на новой задаче. Это наиболее интуитивное описание того, чем является мета-обучение.

Эта область, на мой взгляд, имеет прямое отношение к теме, рассматриваемой в этой работе. До этого были описаны методы ручного выбора статических алгоритмов изменения (или не изменения) темпа обучения и шагов градиентной оптимизации, направленные на борьбу с переобучением. Мета-обучение изучает методы решения таких проблем динамически, что принципиально отличается от способов, описанных в этой работе.

Оптимизация гиперпараметров

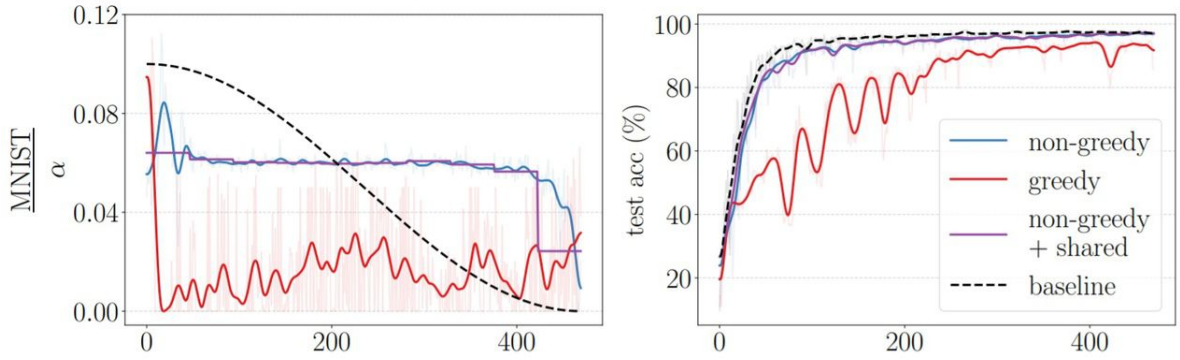
В статье [6] представлен градиентный алгоритм оптимизации гиперпараметров, на примере темпа обучения. Задача оптимизации гиперпараметров, таких темп обучения или коэффициент накопления инерции ставится следующим образом:

$$\lambda^* = argmin_\lambda \; L_{val}\left(\theta_T^*(\lambda), D_{val}\right)$$

$$\theta_T^* = argmin_\theta \; L_{train}\left(\theta, D_{train}\right)$$

По сути мы сначала оптимизируем параметры модели для конкретного значения гиперпараметра, а далее считаем градиент по этому гиперпараметру. Такая задача является ресурсоемкой, так как необходимо хранить все состояния модели в процессе оптимизации, чтобы далее для всех состояний вычислить градиент по гиперпараметру. Градиенты со всех состояний усредняются и происходит шаг оптимизации. Это так же помогает уменьшить дисперсию градиента для рассматриваемого гиперпараметра. В такой задаче присутствует проблема деградации градиента для большого числа итераций оптимизации модели.

На представленном ниже графике изображены различные существующие решения описанной задачи минимизации. Baseline – ручное изменение темпа обучения. В статье представлен non-greedy («не жадный») алгоритм оптимизации.



Сравнение различных алгоритмов оптимизации темпа обучения α

Получившаяся в работе траектория изменения является достаточно логичной. На первых итерациях идет поиск оптимального шага. Далее идет незначительное снижение, а на последних итерациях сильное уменьшение. Это подпадает под логику того, что в конце обучения необходимо уменьшать темп обучения, чтобы спуститься в локальные минимумы. При этом модели, в которых использовался non-greedy алгоритм оптимизации темпа обучения, достигают очень близкого качества.

Такие алгоритмы оптимизации гиперпараметров потенциально избавят нас от жадного перебора разных значений гиперпараметров для каждого отдельного этапа обучения модели (как это делается для достижения максимального качества моделей, которые устанавливали рекорды). Траектории изменения скорости обучения могут быть зафиксированы для последующего использования в других задачах. Таким образом мета-алгоритмы смогут продвинуть статические алгоритмы оптимизации (такие как выбор константного или равномерно убывающего темпа обучения).

Методы в области обучения с подкреплением

Особенно эффективны методы из этой области в обучении с подкреплением (reinforcment learning – модель агента, действующего в среде). Мета-обучение занимается оптимизацией определенным образом введенной модели «учителя», который в свою очередь влияет на обновление весов конкретного агента. Рассмотрим в общем виде два алгоритма, предлагающие способы такой модификации процесса обучения. Обучение мета-алгоритма проводится на распределении задач (выборок для обычного обучения или сред для обучения с подкреплением). Таким образом, из всех сред выделяются общие черты. Поэтому такой мета-алгоритм потенциально будет обучать конкретную модель с большей обобщающей способностью, так как будет содержать в себе информацию о большем числе схожих задач. Далее будут описаны алгоритмы, являющиеся по сути следующим шагом в области разработки алгоритмов оптимизации.

Обучение внутренней награды

В задаче обучения с подкреплением существует агент с оптимизируемыми параметрами и функционал среды, отвечающий за качество текущего алгоритма взаимодействия агента со средой (политика). В статье [7] предлагается метод введения функционала внутренней награды (сразу для нескольких задач, чтобы получить модель учителя), которая суммируется с внешней. Общая схема всей модели выглядит так:

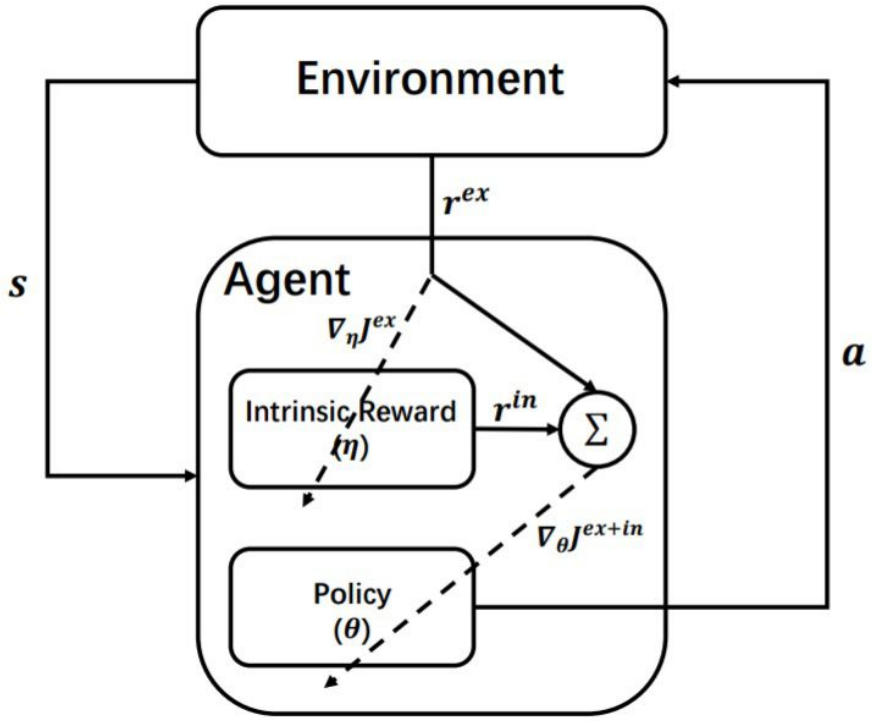
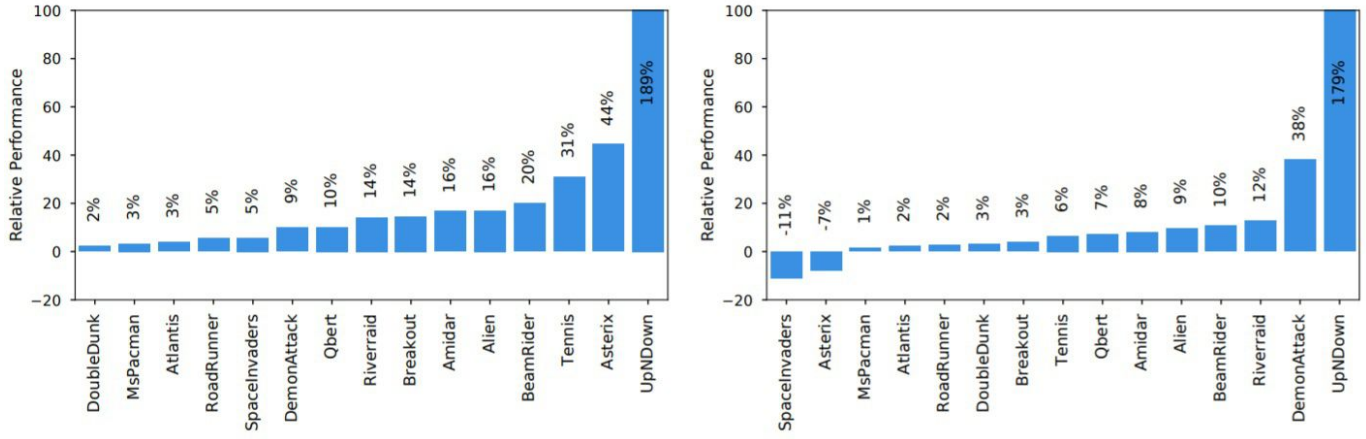


Схема работы модели агента в среде с обучаемым аддитивным оптимизатором – Intrinsic reward; η – обучаемые параметры внутренней награды (аддитивного оптимизатора), стрелка r^{in} - суммируемая прибавка; Policy с параметрами θ – текущая политика агента с обучаемыми параметрами.

В итоге получаем следующий алгоритм оптимизации:

1. прямой проход с вычислением внешней и внутренней награды
2. веса агента оптимизируются в сторону увеличения суммарной награды
3. параметры внутренней награды (в статье предлагается метод посчитать для них градиент) оптимизируются в сторону увеличения внешней награды

Через введенный функционал внутренней награды, влияющий на оптимизацию весов агента, мы учимся наилучшим образом учить агентов. Этот подход по факту является обучением аддитивного оптимизатора. В описанных алгоритмах оптимизации, мы сами задавали, каким образом будут обновляться веса. Такой подход может привести к нетривиальной стратегии оптимизации агентов, характерной для конкретной среды. Такой подход показывает результаты, превосходящие обычное обучение по функционалу внешней награды с привычными оптимизаторами.



Насколько подход с обучаемой внутренней наградой превосходит baseline решения для соответствующих задач. По оси X перечислены классические A2C задачи.

Обучение правила обновления

Этот метод отличается от предыдущего тем, что здесь обучается полноценный единственный оптимизатор, а не дополнительный. В статье [8] предлагается использовать модель учителя, получающего на вход последовательность градиентов и выдающего обновление для весов агента. Такой учитель оптимизируется параллельно на нескольких задачах. Этот мета-алгоритм учится обновлять веса у множества агентов, функционирующих во множестве схожих сред. Можно сказать, что таким образом обучается универсальный оптимизатор для такого класса задач.

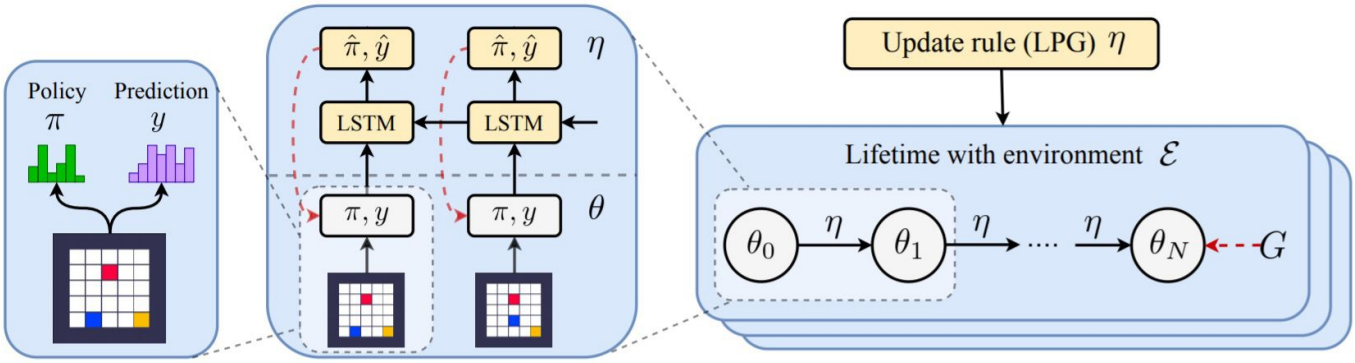


Схема оптимизации множества агентов θ в среде с помощью оптимизируемого учителя LPG η

На схеме выше схематично представлено как обучается модель учителя для агентов (LPG). Этот учитель представляет собой LSTM обратного распространения, получающую на вход последовательность текущих политик и предсказаний агента о среде и выдающую правило обновления (Update rule), для множества агентов. Далее учитель выдает обновление весов агентов θ. Затем

обновляются параметры учителя η по описанному в статье правилу взятия градиента. Этот универсальный учитель становится специальным оптимизатором для агентов определенной архитектуры, функционирующих в определенном виде сред.

Заключение

Существует множество способов борьбы с переобучением в нейронных сетях. Методы предобработки данных являются достаточно типичными для классического машинного обучения. Наиболее характерными для глубокого обучения являются техники оптимизации, не говоря уже о конкретных техниках инициализации специально для нейронных сетей. При борьбе с переобучением мы можем влиять на следующие моменты:

1. на каких данных учится модель
2. как функционально устроена наша модель
3. каким образом оптимизировать ее параметры

Глубокое обучение развивается дальше и теперь также изучаются возможности мета-обучения – класса методов «обучения тому, как учить». Получаемые алгоритмы, в отличие от описанных статических методов, вроде алгоритмов оптимизации, будут заточены под конкретный тип задач.

Список литературы

[1] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks., in: Aistats, Vol. 9, 2010, pp. 249–256.

[2] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[3] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, Quoc V. Le, Don't decay the learning rate, increase the batch size, 2017

[4] Dominic Masters, Carlo Luschi, Revisiting small batch training for deep neural networks, 2018

[5] Визуализация оптимизаторов https://programforyou.ru/projects/gradient-descent-optimization

[6] Paul Micaelli, Amos Storkey, Non-greedy Gradient-based Hyperparameter Optimization Over Long Horizons, 2020

[7] Zeyu Zheng, Junhyuk Oh , Satinder Singh, On Learning Intrinsic Rewards for Policy Gradient Methods, 2018

[8] Junhyuk Oh, Matteo Hessel, Wojciech M. Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, David Silver, Discovering Reinforcement Learning Algorithms, 2021