Forward-forward алгоритм

Петляк Максим, БПМИ213

Приветствие

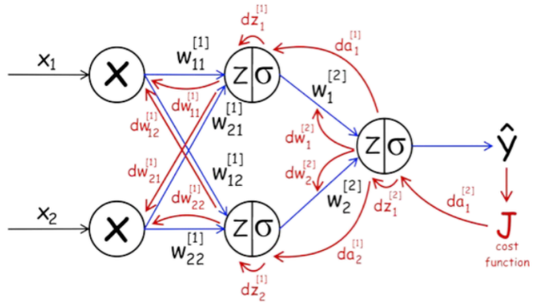


Иллюстрация идеи backpropagation

Все мы с начала года знаем, как происходит все глубинное обучение в нашей жизни: строим архитектуру нейросети, потом обучаем, потом делаем инференс. И при обучении у нас сначала происходит прямой проход, а потом обратный проход.

В этой статье описан другой взгляд на обучение нейросетей -- с модификацией прямого прохода и без использования обратного прохода.

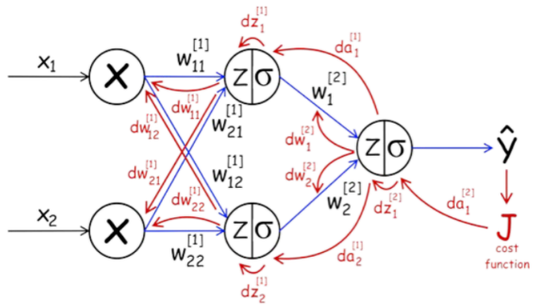


Иллюстрация идеи backpropagation

Проблемы:

Для начала, обсудим некоторые проблемы с обычной конфигурацией с forward pass и backward pass. У неё есть несколько проблем:

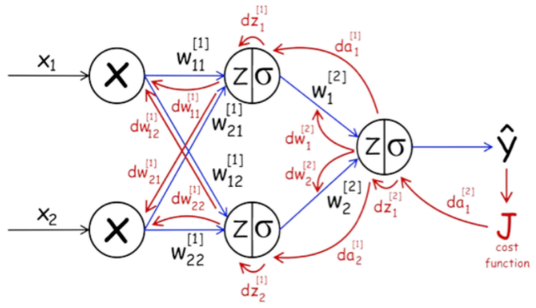


Иллюстрация идеи backpropagation

Проблемы:

1. Необходимость хранить все промежуточные вычисления

1. Первая проблема -- прикладная: необходимость запоминать все промежуточные значения в нейросети для осуществления обратного прохода. Это требует дополнительной памяти.

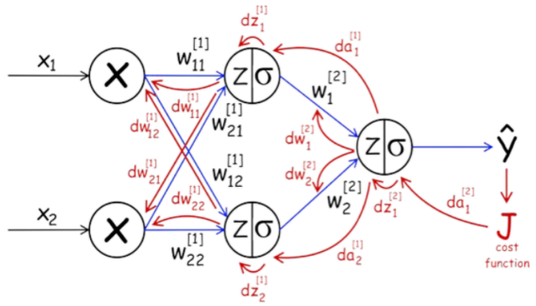


Иллюстрация идеи backpropagation

Проблемы:

1. Необходимость хранить все промежуточные вычисления
2. Идейные расхождения с человеческим мозгом

2. Вторая проблема -- идейная: в каком-то смысле, нейросети стремятся понять, как работает человеческий мозг, и в некотором роде скопировать эту архитектуру. Однако, модель с бекпропом далеко не всегда соответствует нашей голове. Бекпроп требует в какой-то момент обучения остановиться, сделать обратный проход, и только потом продолжить обучение. Легко заметить, что если бы это было так в нашем голове, то нам после минутной лекции по матану нужна была бы ещё одна минута на обратный проход. А это не так, и мы можем учиться налету. Вот это несоответствие в какой-то мере и исправляется в ФФ-алгоритме.

Два прохода:

С настоящими данными

С фейковыми данными

Теперь про сам алгоритм.

Как можно было понять из названия, мы делаем два каких-то прямых прохода: один проход -- с самыми обычными данными. Второй проход -- с "фейковыми" данными (по факту, это всё один и тот же тип прохода, они отличаются только данными, их не обязательно в точности одинаковое количество).

С настоящими данными всё понятно. А что такое фейковые данные? Сразу приведу пример, Пусть, мы хотим обучать классификатор собачки или котика на картинке. Тогда настоящими данными будут фотографии собачек и котиков. А фейковыми данными -- фотографии шкафов, мэра Варшавы или просто рандомный шум. В общем, не фотографии котиков и собачек (тут есть такое замечание -- фейковые фотографии должны быть кое-как похожи на котиков и собачек, чтобы не делать обучение слишком простым).

Зачем нужны эти два вида проходов? Ответ -- мы хотим научиться различать реальные данные от фейковых данных (в статье это явно не прописано, но лично мне интуиция говорит, что таким образом мы можем хорошо понимать структуру данных, что будет нам неплохо помогать в дальнейшем обучении готовой модели).

$$goodness = \sum y_i^2$$

$$\theta - threshold$$

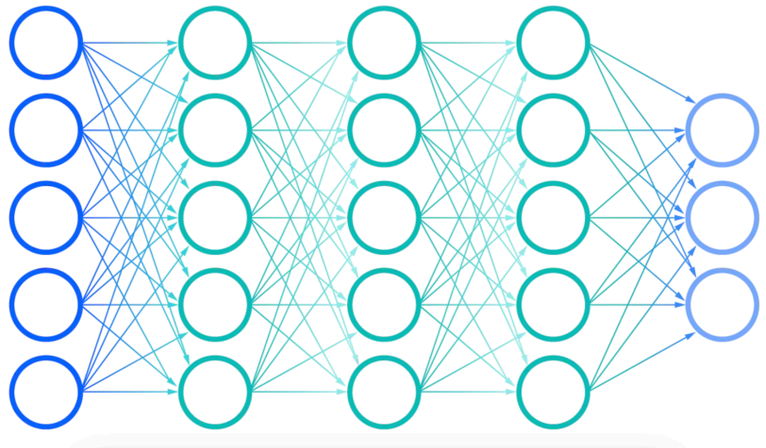
$$P(image\ is\ real) = \sigma(goodness - \theta)$$

Как мы будем классифицировать изображения на фейковые и настоящие? Введём некоторую величину "хорошесть". Чем больше хорошесть -- тем больше мы уверены в том, что картинка настоящая. И мы хотим, что если картинка настоящая, то хорошесть этой картинки больше какого-то порога. И тогда вероятность хорошести картинки -- сигмоида от хорошести минус порог.

Более подробно про хорошесть. Важно -- мы считаем хорошесть на каждом слое нашей нейросети. В статье хорошесть равна квадрату нормы выхода слоя, но можно брать и другие функции. Ну, и как мы оптимизируем хорошесть -- просто градиентными спуском на двух слоях. Нам не надо хранить состояние всей сети как раз из-за этого.

1. Оптимизируем хорошесть на каждом слое
2. Используем на каждом слое нормализацию

Пример архитектуры



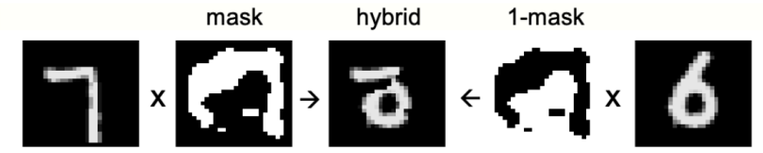
Теперь, что происходит, если у нас больше двух слоев в сети? Можем просто оптимизировать параметры на соседних слоях градиентным спуском. Но появляется проблема -- если мы на одном слое дали сырой выход следующему слою, то это не очень хорошо -- потому что следующему слою очень легко будет понять фейковость данных по абсолютным значениям своего входа. Для этого мы выход каждого слоя подвергаем ещё нормализации. Это можно интерпретировать так: наш выход -- это вектор, который характеризуется направлением и длиной. Ну, и мы можем убрать длину -- это уменьшает размерность данных на единицу, что совсем не критично для нас.

Losses on MNIST

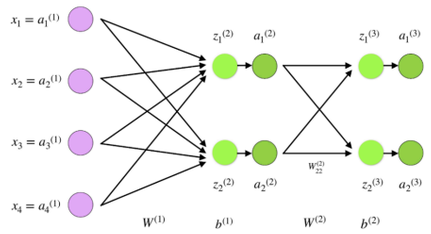
- With default back-propagation: 0.6%
- Back-propagation on permutation-invariant MNIST: 1.4%
- With dropout and label-smoothing: 1.1%

Для начала, давайте обсудим бейзлайны. То есть, какую ошибку дает обучение без ФФ-алгоритма. Обычная нейронка с несколькими свертками обычно дает что-то около 0.6% ошибки на тестовой выборке при ванильном мнисте. На версии мниста с рандомной перестановкой пикселей нейронка с несколькими релу и полносвязными слоями дает где-то 1.4% за 20 эпох. Уменьшается до 1.1% если юзать дропаут и лейблСмузинг. Перейдем теперь к ФФ-алгоритму.

Генерация фейковых данных



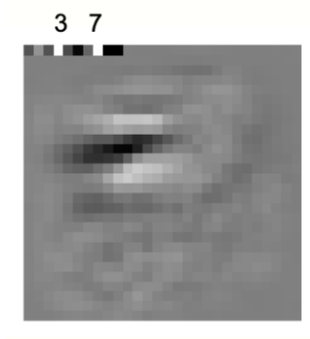
Обучение



Для начала, посмотрим на процесс генерации фейковых данных. По картинке видно, что мы просто берем какую-то маску, накладываем её на два изображения и потом складываем, тем самым получая фейковые данные.

Переходим к самому обучению. Для начала, скажем, как мы будем получать сам результат. Сначала, мы конкатенируем те самые нормализованные выходы всех слоев в один вектор и применяем на них линейный слой (его обучаем как обычно). Естественно, чтобы оно нормально училось, надо, чтобы слои между собой соединялись нелинейными связями. В статье использует 4 скрытых слоя с 2000релу на каждом и обучаеют 100эпох каждый. Таким образом получаем ошибку 1.37%. Это значит, что наша идея работает!!
Архитектура нейросети

Supervised learning of MNIST



Когда мы обучали большие модели, мы использовали в основном только unsupervised методы. Но так как мы сейчас заинтересованы только в одной задаче (в мнисте), мы можем попробовать внедрить некоторые supervised методы (это бывает полезно, когда мы не хотим, чтобы наша маленькая модель учила распределение всех наших данных). Единственный способ достичь этого с ФФ-алгоритмом -- это запихнуть во вход лейбл картинки (если картинка фейковая, давайте пихать неправильный лейбл). Будем пихать лейбл просто бинарной строкой размера 10 в начале картинки (в мнисте это сделать просто, потому что у нас все картинки имеют черную границу). 4 скрытых слоя с 2000релу на 60 эпохах дает 1.36%, а бекпроп дает такое же качество на 20 эпохах.

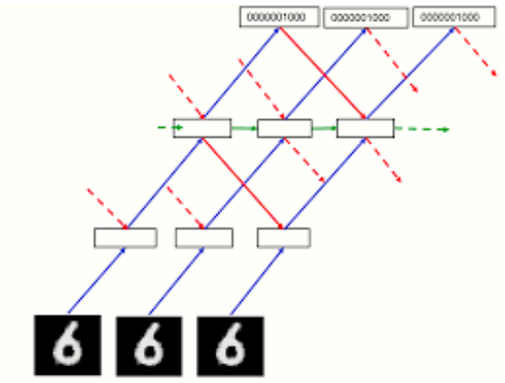
Ну и после обучения такой модельки, можно делать предсказания если выставлять все 10 пикселей в 0.1.

Теперь, когда мы разобрали базовый пример применения, можно разобрать, как происходит применение высокоуровневых фичей.

Для начала -- что это такое? Покажу на примере двух слов и средней буквы. Просто по самой средней букве нереально понять, что это за буква. Однако по тому самому контексту, который можно назвать высокоуровневой фичей, мы можем понять, что это за буква на самом деле.

Ну вот как раз для того, чтобы модель учила эти фичи, нам нужно какое-то соединение между верхними слоями и нижними слоями. Эту проблему можно решить в рамках ФФ-алгоритма.

FF-algorithm to model high-level features



Мы видим на картинке синие линии которые идут вверх к последнему слою, и красные которые идут вниз. И тут получается, что у нас теперь слой принимает на вход не только информацию с прошлого слоя, но и со следующего. А следующий слой -- со слоя ещё выше. Таким образом, если мы применяем это технику достаточное количество раз (на рисунке нарисовано только 3 раза), мы можем передавать высокоуровневую информацию из самого вверха в самый низ, что улучшает качество обучения нашей модели.