

Методы машинного обучения

Шорохов С.Г.

кафедра математического моделирования и искусственного интеллекта

Лекция 5. Сверточные нейронные сети





Под **регуляризацией** модели машинного обучения (в широком смысле) понимают набор методов для уменьшения переобучения в моделях машинного обучения. Регуляризация приводит к некоторому снижению качества (увеличению ошибки) модели на обучающих данных в обмен на увеличение ее обобщаемости (то есть ее способности давать точные прогнозы для новых данных), поэтому применение регуляризации обычно приводит к менее точным прогнозам на обучающих данных, но к более точным прогнозам на новых (тестовых) данных.

Вообще говоря, регуляризация приводит к более простым моделям. И, как утверждает принцип бритвы Оккама: самые простые модели, скорее всего, будут работать лучше. Для того, чтобы ограничить модель меньшим набором возможных вариантов, используются различные методы.



Ранняя остановка (early stopping) — один из самых простых методов регуляризации, предполагающий остановку обучения нейронной сети на более ранней эпохе.

По мере обучения нейронной сети на протяжении многих эпох ошибка на обучающей выборке уменьшается и может достичь значения, сколь угодно близкого к нулю. Такая нейронная сеть будет представлять собой модель с высокой дисперсией, которая плохо работает с новыми данными, которых она никогда раньше не видела, несмотря на почти идеальную производительность на обучающей выборке.

Поэтому если предотвратить сколь угодно низкое снижение потерь при обучении модели, то модель с меньшей вероятностью переобучится на обучающих данных и будет лучше обобщать (лучше работать с новыми данными).

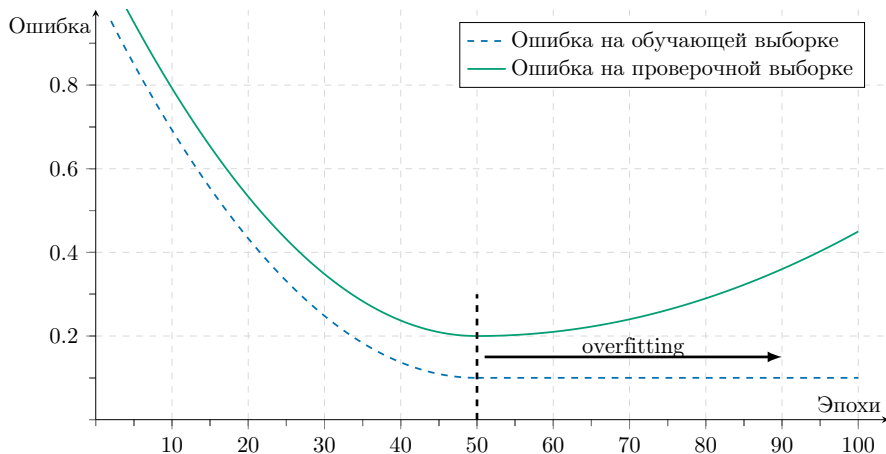
Для ранней остановки можно отслеживать изменение следующих величин:

- Изменение ошибки (потерь) на проверочном наборе данных
- Изменение матрицы весов и вектора смещений

Остановка обучения по проверочной выборке



Если ошибка на проверочной выборке существенно не уменьшается или увеличивается в течение нескольких эпох, то можно прекратить обучение.





Другой способ определения момента остановки – отслеживание изменения параметров модели. Обозначим через θ_t и θ_{t-k} векторы параметров модели (весов и смещений) в эпохи t и $t - k$ соответственно.

Если использовать евклидову норму L_2 для вектора разности $\theta_t - \theta_{t-k}$, то обучение прекращается, если эта величина достаточно мала, скажем, меньше ε :

$$\|\theta_t - \theta_{t-k}\|_2 = \sum_j \left(\theta_t^{(j)} - \theta_{t-k}^{(j)} \right)^2 < \varepsilon.$$

Можно использовать норму L_1 (манхэттенское расстояние), тогда обучение прекращается, если:

$$\|\theta_t - \theta_{t-k}\|_1 = \sum_j \left| \theta_t^{(j)} - \theta_{t-k}^{(j)} \right| < \varepsilon.$$

Можно использовать норму L_∞ (чебышевское расстояние), тогда обучение прекращается, если:

$$\|\theta_t - \theta_{t-k}\|_\infty = \max_j \left| \theta_t^{(j)} - \theta_{t-k}^{(j)} \right| < \varepsilon.$$



Регуляризация (в узком смысле или регуляризация Тихонова) – это метод добавления к функции потерь штрафа за сложность модели с целью предотвратить переобучение модели. Наиболее часто используемые виды регуляризации – L_1 и L_2 , а также их линейная комбинация.

Пусть \mathcal{L} – функция потерь, а $\theta = (\mathbf{W}, \mathbf{b})$ – вектор параметров (весов \mathbf{W} и смещений \mathbf{b}) обучаемой модели $M(\mathbf{x}, \theta)$, а λ – неотрицательный гиперпараметр (коэффициент регуляризации). Будем минимизировать **эмпирический риск** $Q(\theta)$, являющийся оценкой математического ожидания функции потерь:

$$Q(\theta) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, M(\mathbf{x}_i, \theta))$$

Здесь \mathbf{x}_i – точки обучающего набора данных $\mathbf{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, y_i – соответствующие отклики. При регуляризации к потерям на точках обучающего набора данных добавляется штраф $R(\mathbf{W})$ за величину весов \mathbf{W} :

$$Q'(\theta) = Q(\theta) + \lambda R(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, M(\mathbf{x}_i, \theta)) + \lambda R(\mathbf{W})$$



Минимизация регуляризованного соответствующим образом эмпирического риска $Q'(\boldsymbol{\theta})$ приводит к выбору такого вектора весов \mathbf{W} , который не слишком сильно отклоняется от нуля. Поэтому применение регуляризации путем модификации функции потерь также называют **сокращением весов** (weight decay).

При L_2 -регуляризации (ridge regularization) в качестве штрафа используется норма вектора весов \mathbf{W} в L_2 :

$$Q'(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, M(\mathbf{x}_i, \boldsymbol{\theta})) + \lambda \sum_{j=1}^m w_j^2$$

L_2 -регуляризация используется в т.н. гребневой регрессии (ridge regression).



Рассмотрим сначала случай многослойного персептрона с одним скрытым слоем. Параметры модели равны

$$\theta = \{\mathbf{W}_h, \mathbf{b}_h, \mathbf{W}_o, \mathbf{b}_o\}$$

Функция потерь при регуляризации L_2 задается как

$$Q'(\theta) = Q(\theta) + \lambda R_{L_2}(\mathbf{W}_h, \mathbf{W}_o) = Q(\theta) + \lambda \left(\|\mathbf{W}_h\|^2 + \|\mathbf{W}_o\|^2 \right)$$

Регуляризованная функция потерь пытается минимизировать индивидуальные веса для пар нейронов. Это добавляет модели некоторое смещение, но снижает дисперсию, поскольку малые веса более устойчивы к изменениям входных данных с точки зрения прогнозируемых выходных значений. Правило обновления весов \mathbf{W}_o задается как

$$\mathbf{W}_o \leftarrow \mathbf{W}_o - \eta \nabla_{\mathbf{W}_o} = \mathbf{W}_o - \eta \left(\mathbf{z} \delta_o^T + \alpha \mathbf{W}_o \right) = (1 - \eta \alpha) \mathbf{W}_o - \eta \mathbf{z} \delta_o^T$$

Обновленная матрица весов \mathbf{W}_o использует уменьшенные веса из предыдущего шага с использованием коэффициента затухания $1 - \eta \alpha$.



Если дана функция потерь (ошибок) $Q(\boldsymbol{\theta})$, то функция потерь с регуляризацией L_2 равна

$$Q'(\boldsymbol{\theta}) = Q(\boldsymbol{\theta}) + \lambda R_{L_2}(\mathbf{W}_o, \mathbf{W}_1, \dots, \mathbf{W}_h) = Q(\boldsymbol{\theta}) + \lambda \sum_{l=0}^h \|\mathbf{W}_l\|^2,$$

где параметры модели равны $\boldsymbol{\theta} = \{\mathbf{W}_o, \mathbf{b}_o, \mathbf{W}_1, \mathbf{b}_1, \dots, \mathbf{W}_h, \mathbf{b}_h\}$.

На основе вывода для одного скрытого слоя регуляризованный градиент равен:

$$\nabla_{\mathbf{W}_l} = \mathbf{z}^l \left(\boldsymbol{\delta}^{l+1} \right)^T + \alpha \mathbf{W}_l$$

и правило обновления для весовых матриц имеет вид

$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \nabla_{\mathbf{W}_l} = (1 - \eta \alpha) \mathbf{W}_l - \eta \mathbf{z}^l \left(\boldsymbol{\delta}^{l+1} \right)^T$$

для $l = 0, 1, \dots, h$, где $\boldsymbol{\delta}^l$ – чистый вектор градиента для скрытого слоя l .

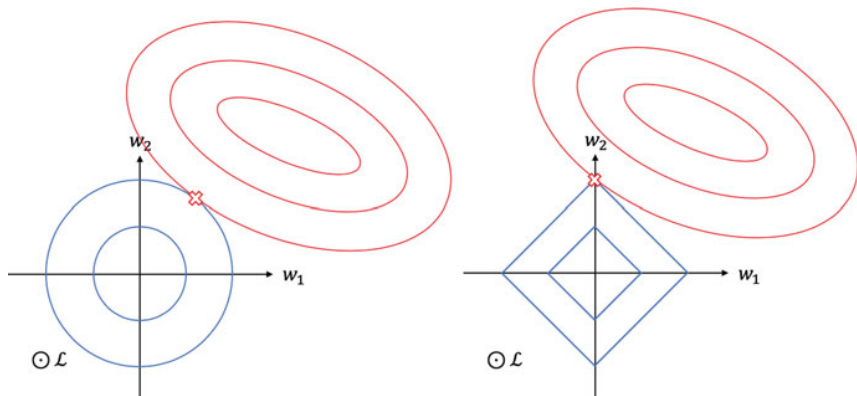


L_1 -регуляризация (lasso regularization) или регуляризация через манхэттенское расстояние:

$$Q'(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, M(\mathbf{x}_i, \boldsymbol{\theta})) + \lambda \sum_{j=1}^m |w_j|$$

Данный вид регуляризации также позволяет ограничить значения вектора \mathbf{w} . Однако, к тому же он обладает интересным и полезным на практике свойством — обнуляет значения некоторых параметров, что приводит к отбору признаков.

L_1 -регуляризация используется в регрессии лассо (lasso regression).



Изображение контурных линий исходной функции потерь (красный) и функций штрафа (синий). Точки пересечения контурных линий указывают на то, что штрафная функция \mathcal{L}_1 может приводить к параметрам с нулевым значением.



Использование функции потерь с штрафом \mathcal{L}_1 позволяет нейронной сети неявно выполнять отбор признаков, то есть отбрасывать некоторые входные признаки, устанавливая соответствующие параметры нейронной сети равными нулю или некоторым небольшим значениям. Как было показано на предыдущем слайде, при заданных параметрах w_1, w_2 в системе координат множество точек $w_1^2 + w_2^2 = r^2$ представляет собой окружность с радиусом r , а множество точек $|w_1| + |w_2| = r$ – квадрат с длиной диагонали $2r$, оба множества которых показаны синими контурными линиями. Красные контурные линии соответствуют исходной функции потерь $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$. Точки пересечения штрафов за норму параметров и первоначальных убытков, обозначенные красными крестиками, указывают на то, что функция штрафа \mathcal{L}_1 с большей вероятностью приведет к параметрам с нулевым значением, чем функция штрафа \mathcal{L}_2 , в то время как функция штрафа \mathcal{L}_2 может привести к параметрам с близкими абсолютными значениями, но отличными от нулевых.



Эластичная сеть (elastic net regularization):

$$Q'(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, M(\mathbf{x}_i, \boldsymbol{\theta})) + \lambda_1 \sum_{j=1}^m |w_j| + \lambda_2 \sum_{j=1}^m w_j^2$$

Эта регуляризация использует как L_1 , так и L_2 регуляризации, учитывая эффективность обоих методов. Ее полезной особенностью является то, что она создает условия для группового эффекта при высокой корреляции переменных, а не обнуляет некоторые из них, как в случае с L_1 -регуляризацией.



В машинном обучении иногда можно столкнуться с ситуацией, когда набор данных имеет ограниченный размер. Но чтобы получить лучшие результаты обобщения модели, необходимо иметь больше данных, в том числе и различные их вариации.

Аугментация данных (data augmentation) — это методика создания дополнительных данных из имеющихся данных.

Часто проблема ограниченного набора данных возникает при решении задач, связанных с обработкой изображений. Следующие способы аугментации изображений являются самыми популярными:

- Отображение по вертикали или горизонтали (flipping)
- Поворот изображения на определенный угол (rotation)
- Создание отступа (padding)
- Вырезание части изображения (cropping)
- Добавление шума (adding noise)
- Манипуляции с цветом (color jittering).

Также, можно применять различные комбинации, к примеру, вырезать часть изображения, повернуть его и изменить цвет фона.



Смешивание (Mixup) – это метод регуляризации, который использует выпуклую комбинацию существующих входных данных для увеличения набора данных.

Пусть \mathbf{x}_i и \mathbf{x}_j – входные изображения, принадлежащие классам i и j соответственно, \mathbf{y}_i и \mathbf{y}_j – это однократные (one-hot) векторы, соответствующие меткам классов i и j соответственно. Новое изображение $\tilde{\mathbf{x}}$ (и его однократная метка $\tilde{\mathbf{y}}$) формируется путем вычисления выпуклой комбинации \mathbf{x}_i и \mathbf{x}_j (\mathbf{y}_i и \mathbf{y}_j соответственно):

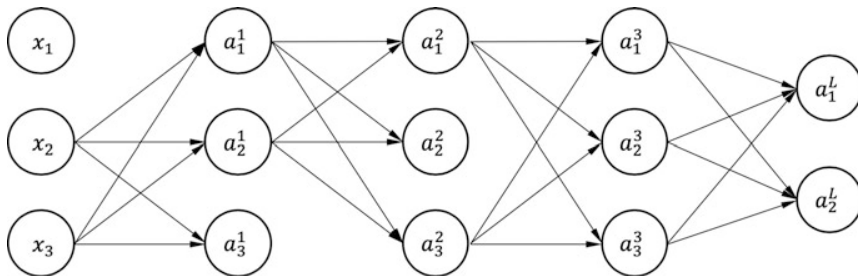
$$\tilde{\mathbf{x}} = \lambda \mathbf{x}_i + (1 - \lambda) \mathbf{x}_j, \tilde{\mathbf{y}} = \lambda \mathbf{y}_i + (1 - \lambda) \mathbf{y}_j, \lambda \in [0, 1].$$

Метод Cutout предполагает случайное удаление частей входного изображения во время обучения. Метод CutMix заменяет удаленные участки частями другого изображения. Метод AugMix делает нейронную сеть устойчивой к изменению распределения. В отличие от метода Mixup, в котором используются изображения из двух разных классов, метод AugMix выполняет серию преобразований одного и того же изображения, а затем использует композицию этих преобразованных изображений для получения результирующего изображения.



Глубокие нейронные сети с большим количеством нейронов могут страдать от взаимной адаптации нейронов, что может привести к переобучению. Взаимная адаптация нейронов означает, что нейроны зависят друг от друга.

Дропаут (dropout) представляет собой метод предотвращения совместной адаптации нейронов, когда при обучении нейронной сети выходы скрытых слоев случайным образом обнуляются, что напоминает случайное отсоединение нейронов.





Термин «dropout» (отсев, выбивание, отбрасывание) характеризует исключение определённого процента (например 30%) случайных нейронов (находящихся как в скрытых, так и видимых слоях) на разных итерациях (эпохах) во время обучения нейронной сети.

Во время обратного распространения ошибки при нулевом выходе соответствующая частная производная функции потерь по отношению к выходу слоя будет равна нулю. Поэтому веса «отключенных» нейронов обновляться не будут, а будут обновляться только оставшиеся подключенными нейроны.

Поэтому, метод дропаута может обучать разные подсети нейронной сети, позволяя всем им использовать одни и те же параметры. Это очень эффективный способ усреднения моделей внутри нейронной сети. В результате более обученные нейроны получают в сети больший вес. Во время прогнозирования (тестирования) дропаут отключается, и никакие выходы слоев не обнуляются. Это означает, что все подсети работают вместе, чтобы предсказать конечный результат.

Дропаут значительно увеличивает скорость обучения, качество обучения на тренировочных данных, а также повышает качество предсказаний модели на новых тестовых данных.



Идея регуляризации **dropout** (отсев) состоит в том, чтобы случайным образом обнулить определенную часть значений нейронов в слое во время обучения. Цель состоит в том, чтобы сделать нейронную сеть более надежной и в то же время избежать переобучения. Отбрасывая случайным образом нейроны для каждой точки обучения, нейронная сеть вынуждена не полагаться на какой-либо конкретный набор связей между нейронами (ребер).

С точки зрения конкретного нейрона, поскольку он не может полагаться на наличие всех входящих связей, нейрон не концентрирует веса на определенных входных ребрах, а скорее распределяет веса среди входящих ребер.

Чистый эффект дропаута аналогичен регуляризации L_2 , поскольку распределение весов приводит к уменьшению весов на входящих ребрах. Таким образом, полученная модель с дропаутом более устойчива к небольшим возмущениям входных данных, что может уменьшить переобучение за счет небольшого увеличения смещения.

Однако, в то время как регуляризация L_2 напрямую изменяет целевую функцию, регуляризация дропаут является формой структурной регуляризации, которая не меняет целевую функцию, а вместо этого изменяет топологию сети с точки зрения того, какие соединения между нейронами в настоящее время активны или неактивны.



На этапе обучения для каждой точки входных данных \mathbf{x} создается случайный **вектор маски**, чтобы отбросить часть скрытых нейронов. Пусть $r \in [0, 1]$ – вероятность сохранения нейрона, тогда вероятность отсева (дропаута) равна $1 - r$.

Пусть скрытый слой состоит из m нейронов, тогда создадим m -мерный многомерный случайный вектор с распределением Бернулли $\mathbf{u} \in \{0, 1\}^m$ (вектор маски), каждый из элементов которого равен 0 с вероятностью дропаута $1 - r$ и 1 с вероятностью r . Пусть $\mathbf{u} = (u_1, u_2, \dots, u_m)^T$, где

$$u_i = \begin{cases} 0 & \text{с вероятностью } 1 - r \\ 1 & \text{с вероятностью } r \end{cases}$$

Тогда фаза прямого прохода задается формулами:

$$\mathbf{z} = f^h (\mathbf{b}_h + \mathbf{W}_h^T \mathbf{x}),$$

$$\tilde{\mathbf{z}} = \mathbf{u} \odot \mathbf{z},$$

$$\mathbf{o} = f^o (\mathbf{b}_o + \mathbf{W}_o^T \tilde{\mathbf{z}}),$$

где знак \odot означает поэлементное умножение.



В приведенном выше базовом подходе дропаута есть одна сложность, а именно, ожидаемый выход нейронов скрытого слоя во время обучения и тестирования отличается, поскольку дропаут не применяется на этапе тестирования (например, не нужно, чтобы прогнозы менялись случайным образом на заданном тестовом входе). При вероятности сохранения скрытого нейрона, равной r , его ожидаемое выходное значение во время обучения равно

$$\mathbb{E}[z_i] = r z_i + (1 - r) 0 = r z_i.$$

Поскольку во время тестирования дропаут не применяется, выходные данные скрытых нейронов во время тестирования будут выше, чем во время обучения. Один из подходов состоит в том, чтобы масштабировать значения скрытых нейронов по r во время тестирования. С другой стороны, существует более простой подход, называемый **инвертированным дропаутом**, который не требует изменений во время тестирования. Идея состоит в том, чтобы изменить масштаб скрытых нейронов после шага дропаута на этапе обучения следующим образом:

$$\mathbf{z} = f^h(\mathbf{b}_h + \mathbf{W}_h^T \mathbf{x}), \tilde{\mathbf{z}} = \frac{1}{r} \mathbf{u} \odot \mathbf{z}, \mathbf{o} = f^o(\mathbf{b}_o + \mathbf{W}_o^T \tilde{\mathbf{z}})$$



Аналогичным образом выполняется регуляризация дропаутом для глубоких сетей MLP. Пусть $r_l \in [0, 1]$ для $l = 1, 2, \dots, h$ обозначает вероятность сохранения скрытого нейрона для слоя l , так что $1 - r_l$ – вероятность дропаута. Можно также использовать единую вероятность r для всех слоев, установив $r_l = r$. Определим вектор маски $\mathbf{u}^l \in \{0, 1\}^{n_l}$ для скрытого слоя l следующим образом:

$$u_i^l = \begin{cases} 0 & \text{с вероятностью } 1 - r_l \\ 1 & \text{с вероятностью } r_l \end{cases}$$

Тогда шаг прямого прохода между слоями l и $l + 1$ задается так:

$$\mathbf{z}^l = f^h \left(\mathbf{b}_l + \mathbf{W}_l^T \tilde{\mathbf{z}} \right),$$

$$\tilde{\mathbf{z}} = \frac{1}{r} \mathbf{u}^l \odot \mathbf{z}^l,$$

используя инвертированный дропаут.



Пакетная нормализация (batch-normalization) – это метод, который позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей. Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

Во время обучения слой пакетной нормализации оценивает среднее значение и дисперсию входных данных в пакете, используя скользящее среднее. Далее скользящие среднее значение и дисперсия обновляются, чтобы нормализовать входные данные пакета. Во время тестирования (прогнозирования) среднее значение и дисперсия фиксированы и применяются для нормализации входных данных.

Помимо повышения производительности и стабильности, пакетная нормализация обеспечивает регуляризацию. Подобно процессу исключения, который добавляет случайный фактор к скрытым значениям, скользящее среднее значение и дисперсия нормализации пакета вносят случайность, поскольку они обновляются на каждой итерации в соответствии со случайным мини-пакетом.



Кроме того, использование пакетной нормализации обладает еще несколькими дополнительными полезными свойствами:

- достигается более быстрая сходимость моделей, несмотря на выполнение дополнительных вычислений;
- пакетная нормализация позволяет каждому слою сети обучаться более независимо от других слоев;
- становится возможным использование более высокого темпа обучения, так как пакетная нормализация гарантирует, что выходы узлов нейронной сети не будут иметь слишком больших или малых значений;
- пакетная нормализация в каком-то смысле также является механизмом регуляризации: данный метод привносит в выходы узлов скрытых слоев некоторый шум, аналогично методу dropout;
- модели становятся менее чувствительны к начальной инициализации весов.



Пусть $f, g : \mathbb{R}^n \rightarrow \mathbb{R}$ – две интегрируемые на пространстве \mathbb{R}^n функции. Тогда их **свёрткой** называется функция $f * g : \mathbb{R}^n \rightarrow \mathbb{R}$, определённая формулой

$$(f * g)(x) = \int_{\mathbb{R}^n} f(y) g(x - y) dy = \int_{\mathbb{R}^n} f(x - y) g(y) dy$$

Свертка функций — это важнейшее математическое понятие, которое используется почти во всех областях науки и техники, в том числе, оно широко применяется для оценки систем изображения и для процессинга цифровых изображений.

В дискретном случае при вычислении свертки операция интегрирования заменяется на суммирование.

Свертка 1D (одномерная свертка)



Для вектора $\mathbf{a} \in \mathbb{R}^k$ определим оператор суммирования sum , складывающий все элементы вектора, т.е.

$$\text{sum}(\mathbf{a}) = \sum_{i=1}^k a_i, \mathbf{a} = (a_1, \dots, a_k)^T$$

Свертка 1D между вектором (изображением) $\mathbf{x} \in \mathbb{R}^n$ и фильтром $\mathbf{w} \in \mathbb{R}^k$ ($n \geq k$), обозначаемая символом $*$, равна

$$\mathbf{x} * \mathbf{w} = (\text{sum}(\mathbf{x}_k(1) \odot \mathbf{w}), \dots, \text{sum}(\mathbf{x}_k(n-k+1) \odot \mathbf{w}))^T,$$

где символ \odot обозначает поэлементное произведение и для $i = \overline{1, n-k+1}$

$$\text{sum}(\mathbf{x}_k(i) \odot \mathbf{w}) = \sum_{j=1}^k x_{i+j-1} w_j = x_i w_1 + x_{i+1} w_2 + \dots + x_{i+k-1} w_k$$

В результате свертки векторов $\mathbf{x} \in \mathbb{R}^n$ и $\mathbf{w} \in \mathbb{R}^k$ при $n \geq k$ получается вектор длины $n - k + 1$. При $n = k$ значением свертки является скалярное произведение векторов \mathbf{x} и \mathbf{w} .

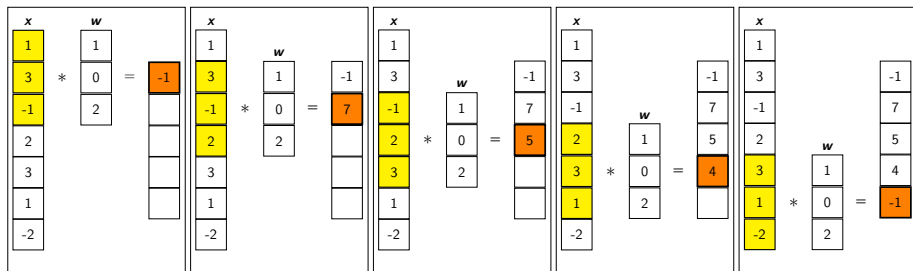


Пусть даны вектор \mathbf{x} для $n = 7$ и фильтр $\mathbf{w} = (1, 0, 2)^T$ для $k = 3$. Первое окно для \mathbf{x} равно $\mathbf{x}_3(1) = (1, 3, -1)^T$. Тогда

$$\text{sum}(\mathbf{x}_3(1) \odot \mathbf{w}) = \text{sum}\left((1, 3, -1)^T \odot (1, 0, 2)^T\right) = \text{sum}\left((1, 0, -2)^T\right) = -1$$

Свертка $\mathbf{x} * \mathbf{w}$ имеет длину $n - k + 1 = 7 - 3 + 1 = 5$ и равна

$$\mathbf{x} * \mathbf{w} = (-1, 7, 5, 4, -1)^T$$



Свертка 2D (двумерная свертка)



Для $k \times k$ -матрицы $\mathbf{A} \in \mathbb{R}^{k \times k}$ определим оператор суммирования sum , складывающий все элементы матрицы, т.е.

$$\text{sum}(\mathbf{A}) = \sum_{i=1}^k \sum_{j=1}^k a_{i,j}, \quad \mathbf{A} = (a_{i,j})_{i,j=1,\overline{k}}$$

Свертка 2D между матрицей (изображением) $\mathbf{X} \in \mathbb{R}^{n \times n}$ и матрицей (фильтром) $\mathbf{W} \in \mathbb{R}^{k \times k}$ ($n \geq k$) равна

$$\mathbf{X} * \mathbf{W} = \begin{pmatrix} \text{sum}(\mathbf{X}_k(1,1) \odot \mathbf{W}) & \dots & \text{sum}(\mathbf{X}_k(1,n-k+1) \odot \mathbf{W}) \\ \dots & \dots & \dots \\ \text{sum}(\mathbf{X}_k(n-k+1,1) \odot \mathbf{W}) & \dots & \text{sum}(\mathbf{X}_k(n-k+1,n-k+1) \odot \mathbf{W}) \end{pmatrix}$$

где \odot – это поэлементное произведение $\mathbf{X}_k(i,j)$ и \mathbf{W} и при $i,j = 1, \dots, n-k+1$

$$\text{sum}(\mathbf{X}_k(i,j) \odot \mathbf{W}) = \sum_{a=1}^k \sum_{b=1}^k x_{i+a-1,j+b-1} w_{a,b} = x_{i,j} w_{1,1} + \dots + x_{i+k-1,j+k-1} w_{k,k}$$

В результате свертки матриц $\mathbf{X} \in \mathbb{R}^{n \times n}$ и $\mathbf{W} \in \mathbb{R}^{k \times k}$ при $n \geq k$ получается $(n-k+1) \times (n-k+1)$ -матрица.



Пусть дана матрица \mathbf{X} для $n = 4$ и и фильтр \mathbf{W} с размером окна $k = 2$.
Свертка первого окна \mathbf{X} , а именно, $\mathbf{X}_2(1, 1)$ с \mathbf{W} равна

$$\text{sum}(\mathbf{X}_2(1, 1) \odot \mathbf{W}) = \text{sum} \left(\left(\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} \odot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) \right) = \text{sum} \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) = 2$$

Шаги свертки для различных сдвигающихся окон размером 2×2 матрицы \mathbf{X} с фильтром \mathbf{W} показаны далее.

Свертка $\mathbf{X} * \mathbf{W}$ имеет размеры 3×3 , так как $n - k + 1 = 4 - 2 + 1 = 3$, и равна

$$\mathbf{X} * \mathbf{W} = \begin{pmatrix} 2 & 6 & 4 \\ 4 & 4 & 8 \\ 4 & 4 & 4 \end{pmatrix}$$

$$\begin{array}{|c|c|c|c|} \hline & X & & \\ \hline 1 & 2 & 2 & 1 \\ \hline 3 & 1 & 4 & 2 \\ \hline 2 & 1 & 3 & 4 \\ \hline 1 & 2 & 3 & 1 \\ \hline \end{array}
 *
 \begin{array}{|c|c|} \hline & W \\ \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}
 =
 \begin{array}{|c|c|c|} \hline 2 & 6 & 4 \\ \hline 4 & 4 & 8 \\ \hline 4 & 4 & 4 \\ \hline \end{array}$$



Пусть \mathbf{W} – $k \times k \times r$ -тензор весов, называемый 3D фильтром, где $k \leq n$ и $r \leq m$. Пусть $\mathbf{X}_k(i, j, q)$ обозначает $k \times k \times r$ -подтензор \mathbf{X} , начиная со строки j , столбца j и канала q , где $1 \leq i, j \leq n - k + 1$ и $1 \leq q \leq m - r + 1$.

Для $k \times k \times r$ -тензора $\mathbf{A} \in \mathbb{R}^{k \times k \times r}$ определим оператор суммирования sum , складывающий все элементы тензоров, т.е.

$$\text{sum}(\mathbf{A}) = \sum_{i=1}^k \sum_{j=1}^k \sum_{q=1}^r a_{i,j,q},$$

где $a_{i,j,q}$ – элемент \mathbf{A} со строкой i , столбцом j и каналом q . Пусть

$$\text{sum}(\mathbf{X}_k(i, j, q) \odot \mathbf{W}) = \sum_{a=1}^k \sum_{b=1}^k \sum_{c=1}^r x_{i+a-1, j+b-1, q+c-1} w_{a,b,c}$$

для $i, j = 1, \dots, n - k + 1$ и $q = 1, \dots, m - r + 1$.

В результате свертки тензоров $\mathbf{X} \in \mathbb{R}^{n \times n \times m}$ и $\mathbf{W} \in \mathbb{R}^{k \times k \times r}$ при $n \geq k$ и $m \geq r$ получается тензор размерами $(n - k + 1) \times (n - k + 1) \times (m - r + 1)$.



Свертка 3D тензоров \mathbf{X} и \mathbf{W} , обозначаемая как $\mathbf{X} * \mathbf{W}$, определяется как

$$\mathbf{X} * \mathbf{W} = \begin{pmatrix} \text{sum}(\mathbf{X}_k(1, 1, 1) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(1, n - k + 1, 1) \odot \mathbf{W}) \\ \text{sum}(\mathbf{X}_k(2, 1, 1) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(2, n - k + 1, 1) \odot \mathbf{W}) \\ \vdots & \vdots & \vdots \\ \text{sum}(\mathbf{X}_k(n - k + 1, 1, 1) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(n - k + 1, n - k + 1, 1) \odot \mathbf{W}) \end{pmatrix}$$

$$\begin{pmatrix} \text{sum}(\mathbf{X}_k(1, 1, 2) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(1, n - k + 1, 2) \odot \mathbf{W}) \\ \text{sum}(\mathbf{X}_k(2, 1, 2) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(2, n - k + 1, 2) \odot \mathbf{W}) \\ \vdots & \vdots & \vdots \\ \text{sum}(\mathbf{X}_k(n - k + 1, 1, 2) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(n - k + 1, n - k + 1, 2) \odot \mathbf{W}) \end{pmatrix}$$

$$\begin{pmatrix} \vdots & \vdots & \vdots \\ \text{sum}(\mathbf{X}_k(1, 1, m - r + 1) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(1, n - k + 1, m - r + 1) \odot \mathbf{W}) \\ \text{sum}(\mathbf{X}_k(2, 1, m - r + 1) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(2, n - k + 1, m - r + 1) \odot \mathbf{W}) \\ \vdots & \vdots & \vdots \\ \text{sum}(\mathbf{X}_k(n - k + 1, 1, m - r + 1) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{X}_k(n - k + 1, n - k + 1, m - r + 1) \odot \mathbf{W}) \end{pmatrix}$$

где \odot – поэлементное произведение $\mathbf{X}_k(i, j, q)$ и \mathbf{W} .

3D подтензор $\mathbf{X}_k(i, j, q)$



Подтензор $\mathbf{X}_k(i, j, q)$ размерами $k \times k \times r$ содержит элементы тензора \mathbf{X} , начиная со строки i , столбца j и канала q :

				$x_{i,j,q+r-1}$	$x_{i,j+1,q+r-1}$	\dots	$x_{i,j+k-1,q+r-1}$
				$x_{i+1,j,q+r-1}$	$x_{i+1,j+1,q+r-1}$	\dots	$x_{i+1,j+k-1,q+r-1}$
				\vdots	\vdots	\dots	\vdots
				$x_{i,j,q+1}$	$x_{i,j+1,q+1}$	\dots	$x_{i,j+k-1,q+1}$
				$x_{i+1,j,q+1}$	$x_{i+1,j+1,q+1}$	\dots	$x_{i+1,j+k-1,q+1}$
				\vdots	\vdots	\dots	\vdots
				$x_{i,j,q}$	$x_{i,j+1,q}$	\dots	$x_{i,j+k-1,q}$
				$x_{i+1,j,q}$	$x_{i+1,j+1,q}$	\dots	$x_{i+1,j+k-1,q}$
				\vdots	\vdots	\dots	\vdots
				$x_{i+k-1,j,q}$	$x_{i+k-1,j+1,q}$	\dots	$x_{i+k-1,j+k-1,q}$



Пусть даны $3 \times 3 \times 3$ -тензор \mathbf{X} ($n = 3$, $m = 3$)

$$\mathbf{X} = \left(\begin{array}{ccc|ccc|ccc} 1 & -1 & 3 & 2 & 1 & 3 & 1 & -2 & 4 \\ 2 & 1 & 4 & 3 & -1 & 1 & 2 & 1 & -2 \\ 3 & 1 & 2 & 1 & 1 & -2 & 1 & 3 & -1 \end{array} \right)$$

и $2 \times 2 \times 3$ -фильтр \mathbf{W} с размером окна $k = 2$ и $r = 3$:

$$\mathbf{W} = \left(\begin{array}{cc|cc|cc} 1 & 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 & 1 & 0 \end{array} \right)$$

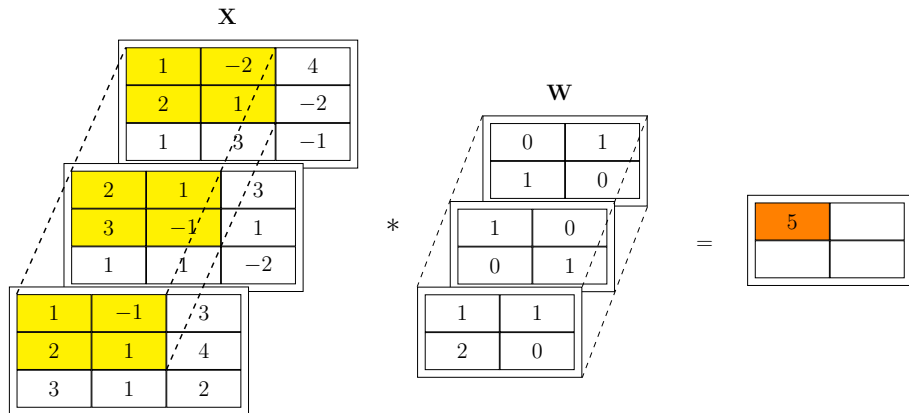
Свертка первого окна \mathbf{X} , а именно, $\mathbf{X}_2(1, 1, 1)$, с фильтром \mathbf{W} равна

$$\begin{aligned} \text{sum} \left(\left(\begin{array}{cc|cc|cc} 1 & -1 & 2 & 1 & 1 & -2 \\ 2 & 1 & 3 & -1 & 2 & 1 \end{array} \right) \odot \left(\begin{array}{cc|cc|cc} 1 & 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 & 1 & 0 \end{array} \right) \right) = \\ = \text{sum} \left(\left(\begin{array}{cc|cc|cc} 1 & -1 & 2 & 0 & 0 & -2 \\ 4 & 0 & 0 & -1 & 2 & 0 \end{array} \right) \right) = 5 \end{aligned}$$

Пример свертки 3D – верхний левый элемент



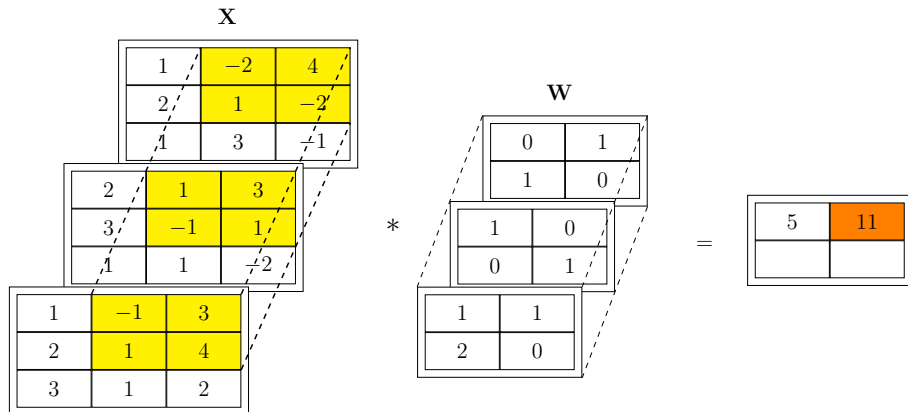
Для $3 \times 3 \times 3$ -тензора \mathbf{X} ($n = 3$, $m = 3$) и $2 \times 2 \times 3$ -фильтра \mathbf{W} с размером окна $k = 2$ и $r = 3$ верхний левый элемент свертки равен:



Пример свертки 3D – верхний правый элемент



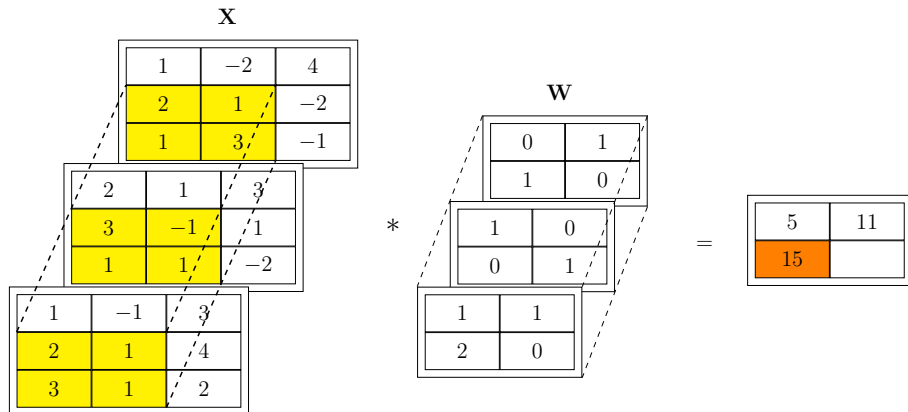
Для $3 \times 3 \times 3$ -тензора \mathbf{X} ($n = 3$, $m = 3$) и $2 \times 2 \times 3$ -фильтра \mathbf{W} с размером окна $k = 2$ и $r = 3$ верхний правый элемент свертки равен:



Пример свертки 3D – нижний левый элемент



Для $3 \times 3 \times 3$ -тензора \mathbf{X} ($n = 3, m = 3$) и $2 \times 2 \times 3$ -фильтра \mathbf{W} с размером окна $k = 2$ и $r = 3$ нижний левый элемент свертки равен:



Пример свертки 3D – нижний правый элемент



Для $3 \times 3 \times 3$ -тензора \mathbf{X} ($n = 3$, $m = 3$) и $2 \times 2 \times 3$ -фильтра \mathbf{W} с размером окна $k = 2$ и $r = 3$ нижний правый элемент свертки равен:

\mathbf{X}

			1	-2	4
			2	1	-2
			1	3	-1
		2	1	3	
		3	-1	1	
		1	1	-2	
	1	-1	3		
	2	1	4		
	3	1	2		

\mathbf{W}

			0	1
			1	0
		1	0	
		0	1	
	1	1		
	2	0		

*

		5	11
		15	5

=

Свертка $\mathbf{X} * \mathbf{W}$ имеет размеры 2×2 , так как $n - k + 1 = 3 - 2 + 1 = 2$ и $r = m = 3$, и равна $\mathbf{X} * \mathbf{W} = \begin{pmatrix} 5 & 11 \\ 15 & 5 \end{pmatrix}$.



Сверточные нейронные сети (Convolutional Neural Networks, CNN) представляют собой глубокие сети MLP, использующие пространственные или временные отношения между различными элементами каждого слоя для построения иерархии признаков. В отличие от обычных сетей MLP, слои которых полностью связаны, в сетях CNN слои локализованы и разрежены. В частности, сети CNN очень эффективны для обработки изображений в качестве входных данных.

Сверточная нейронная сеть — это, по сути, локализованная и разреженная сеть MLP с прямой связью, предназначенная для использования пространственной и/или временной структуры входных данных. В обычной сети MLP все нейроны слоя l связаны со всеми нейронами слоя $l + 1$. Напротив, сеть CNN соединяет некоторое подмножество нейронов в слое l с одним нейроном в следующем слое $l + 1$. Различные скользящие окна, содержащие смежные подмножества нейронов в слое l , соединяются с разными нейронами в слое $l + 1$. Кроме того, все эти скользящие окна предусматривают совместное использование параметров, то есть один и тот же набор весов, называемый **фильтром**, используется для всех скользящих окон. Наконец, различные фильтры используются для автоматического извлечения признаков из слоя l для использования слоем $l + 1$.



В сверточной нейронной сети выходы промежуточных слоев образуют матрицу (изображение) или набор матриц (несколько слоёв изображения). Так, например, на вход сверточной нейронной сети можно подавать три слоя изображения (R-, G-, B-каналы изображения).

Основными видами слоев в сверточной нейронной сети являются

- сверточные слои (convolutional layer)
- пулинговые слои (pooling layer)
- полносвязные (плотные) слои (fully-connected (dense) layer).

Сверточный слой нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса фильтра (ядра) свертки являются обучаемыми параметрами. Еще одним обучаемым параметром является смещение (bias).

В одном сверточном слое может быть несколько сверток. В этом случае для каждой свертки на выходе получится своё изображение.



Рассмотрим роль нейронов смещения и функций активации для тензора нейронов в слое l . Пусть \mathbf{Z}^l – тензор размера $n_l \times n_l \times m_l$ из нейронов в слое l , где $z_{i,j,q}^l$ обозначает значение нейрона в строке i , столбце j и канале q для слоя l для $1 \leq i, j \leq n_l$ и $1 \leq q \leq m_l$.

Пусть \mathbf{W} – 3D фильтр размера $k \times k \times m_l$. В результате свертки тензоров \mathbf{Z}^l и \mathbf{W} получается матрица размерами $(n_l - k + 1) \times (n_l - k + 1)$ в слое $l + 1$. Пусть $b \in \mathbb{R}$ – скалярное смещение фильтра \mathbf{W} и пусть $\mathbf{Z}_k^l(i, j)$ обозначает $k \times k \times m_l$ -подтензор \mathbf{Z}^l в позиции (i, j) . Тогда чистый выход на нейроне $z_{i,j}^{l+1}$ в слое $l + 1$ задается как

$$\text{net}_{i,j}^{l+1} = \text{sum}(\mathbf{Z}_k^l(i, j) \odot \mathbf{W}) + b$$

и значение нейрона $z_{i,j}^{l+1}$ получается применением некоторой функции активации f к чистому выходу

$$z_{i,j}^{l+1} = f(\text{net}_{i,j}^{l+1}) = f(\text{sum}(\mathbf{Z}_k^l(i, j) \odot \mathbf{W}) + b)$$

Функция активации может быть любой из применяемых в нейронных сетях, например, тождественная функция, сигмоида, гиперболический тангенс, ReLU и т.п.



В результате применения 3D фильтра \mathbf{W} с соответствующим смещением b получается $(n_l - k + 1) \times (n_l - k + 1)$ матрица в слое $l + 1$.

Если необходимы m_{l+1} каналов в слое $l + 1$, то требуются m_{l+1} различных $k \times k \times m_l$ фильтров \mathbf{W}_q с соответствующим смещением b_q , чтобы получить $(n_l - k + 1) \times (n_l - k + 1) \times m_{l+1}$ тензор в слое $l + 1$:

$$\mathbf{Z}^{l+1} = \{z_{i,j,q}^{l+1} = f(\text{sum}(\mathbf{Z}_k^l(i,j) \odot \mathbf{W}_q) + b_q)\}_{i,j=1,\dots,n_l-k+1, q=1,\dots,m_{l+1}}$$

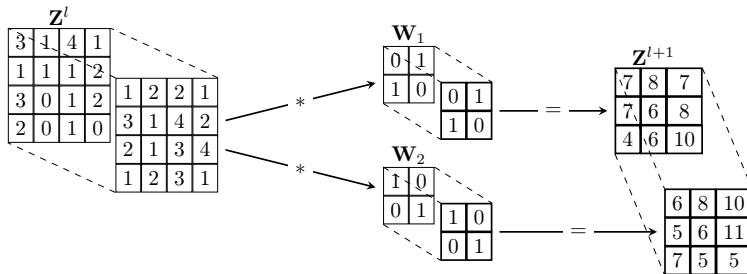
Таким образом, сверточный слой принимает в качестве входных данных $n_l \times n_l \times m_l$ тензор \mathbf{Z}^l нейронов из слоя l , а затем вычисляет $n_{l+1} \times n_{l+1} \times m_{l+1}$ тензор \mathbf{Z}^{l+1} нейронов для следующего слоя $l + 1$ при помощи свертки \mathbf{Z}^l с набором из m_{l+1} различных 3D фильтров размером $k \times k \times m_l$ с последующим добавлением смещения и применением некоторой нелинейной функции активации f .

Каждый 3D фильтр, примененный к \mathbf{Z}^l , приводит к созданию нового канала в слое $l + 1$. Таким образом, m_{l+1} фильтров используются для получения m_{l+1} каналов в слое $l + 1$.



В примере используется $4 \times 4 \times 2$ -тензор \mathbf{Z}^l ($n = 4$, $m = 2$) и два различных $2 \times 2 \times 2$ -фильтра \mathbf{W}_1 и \mathbf{W}_2 с размером окна $k = 2$ и $r = 2$. Так как $r = m = 2$, свертка \mathbf{Z}^l и \mathbf{W}_i ($i = 1, 2$) представляет собой 3×3 -матрицу ($n - k + 1 = 4 - 2 + 1 = 3$).

Фильтр \mathbf{W}_1 создает один канал и фильтр \mathbf{W}_2 создает другой канал, поэтому тензор для следующего слоя \mathbf{Z}^{l+1} имеет размеры $3 \times 3 \times 2$ с двумя каналами (по одному на фильтр):





Применение операции свертки уменьшает изображение, поэтому размер тензоров будет обязательно уменьшаться в каждом последующем сверточном слое сети CNN. Также пиксели, которые находятся на границе изображения участвуют в меньшем количестве сверток, чем внутренние.

Если слой l имеет размеры $n_l \times n_l \times m_l$ и используются фильтры размера $k \times k \times m_l$, то для каждого канала тензор в слое $l + 1$ будет иметь размеры $(n_l - k + 1) \times (n_l - k + 1)$. То есть количество строк и столбцов для каждого последующего тензора будет уменьшаться на $k - 1$, и это ограничит количество сверточных слоев, которые сможет иметь сеть CNN.

В связи с этим в сверточных слоях используется техника **дополнения** изображения (padding). Выходы с предыдущего слоя дополняются пикселями так, чтобы после свертки сохранился размер изображения. Такие свертки называют одинаковыми (same convolution), а свертки без дополнения изображения называются правильными (valid convolution).



Простое решение в технике **padding (дополнение)** состоит в том, чтобы заполнить каждый тензор как по строкам, так и по столбцам в каждом канале некоторым значением по умолчанию, обычно нулем.

Для единообразия всегда добавляется одинаковое количество строк сверху и внизу, а также одинаковое количество столбцов слева и справа.

После добавления p строк/столбцов размер тензора слоя l становится равным $(n_l + 2p) \times (n_l + 2p) \times m_l$. Предположим, что каждый фильтр имеет размеры $k \times k \times m_l$ и что всего имеется m_{l+1} фильтров, тогда размер тензора слоя $l + 1$ будет равен $(n_l + 2p - k + 1) \times (n_l + 2p - k + 1) \times m_l$. Желательно сохранить размер результирующего тензора, чтобы, по возможности

$$n_l + 2p - k + 1 \geq n_l,$$

поэтому используют значение $p = \left\lceil \frac{k-1}{2} \right\rceil$, где $\lceil * \rceil$ – целая часть числа.

Применяя технику дополнения (padding), можно строить сколь угодно глубокие нейронные сети архитектуры CNN.



На рисунках показана двумерная свертка без дополнения ($p = 0$) и с дополнением ($p = 1$). Производится 2D свертка 5×5 -матрицы Z^l ($n = 5$) с фильтром 3×3 W ($k = 3$), что приводит к матрице 3×3 , поскольку $n - k + 1 = 5 - 3 + 1 = 3$. Поэтому размер следующего слоя Z^{l+1} уменьшился.

При $p = 1$ размер слоя Z^{l+1} сохраняется. Таким образом, при помощи дополнения можно использовать столько слоев свертки, сколько необходимо, без уменьшения размера слоев.

$$\begin{array}{|c|c|c|c|c|} \hline & \mathbf{Z}^l & & & \\ \hline 1 & 2 & 2 & 1 & 1 \\ \hline 3 & 1 & 4 & 2 & 1 \\ \hline 2 & 1 & 3 & 4 & 3 \\ \hline 1 & 2 & 3 & 1 & 1 \\ \hline 4 & 1 & 3 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline & \mathbf{W} & \\ \hline 1 & 0 & 0 \\ \hline 0 & 1 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & \mathbf{Z}^{l+1} & \\ \hline 7 & 11 & 9 \\ \hline 9 & 11 & 12 \\ \hline 8 & 8 & 7 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|} \hline & \mathbf{Z}^l & & & & & \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 1 & 0 \\ \hline 0 & 3 & 1 & 4 & 2 & 1 & 0 \\ \hline 0 & 2 & 1 & 3 & 4 & 3 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 1 & 0 \\ \hline 0 & 4 & 1 & 3 & 2 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline & \mathbf{W} & \\ \hline 1 & 0 & 0 \\ \hline 0 & 1 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|} \hline & \mathbf{Z}^{l+1} & & & \\ \hline 6 & 5 & 7 & 4 & 2 \\ \hline 6 & 7 & 11 & 9 & 5 \\ \hline 4 & 9 & 11 & 12 & 6 \\ \hline 7 & 8 & 8 & 7 & 6 \\ \hline 5 & 5 & 7 & 6 & 2 \\ \hline \end{array}$$



Обычно свертка применяется подряд для каждого положения фильтра, но иногда используются только положения, кратные некоторому сдвигу. Техника **сдвига (striding)** часто используется для уменьшения количества скользящих окон, используемых в свертках. Вместо рассмотрения всех возможных окон индекс как по строкам, так и по столбцам увеличивается на целочисленное значение $s \geq 1$, называемое **шагом**. Свертка 3D тензора \mathbf{Z}^l размерами $n_l \times n_l \times m_l$ и фильтра \mathbf{W} размерами $k \times k \times m_l$ с использованием шага s дает следующее:

$$\mathbf{Z}^l * \mathbf{W} = \begin{pmatrix} \text{sum}(\mathbf{Z}_k^l(1, 1) \odot \mathbf{W}) & \text{sum}(\mathbf{Z}_k^l(1, 1 + s) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{Z}_k^l(1, 1 + t \cdot s) \odot \mathbf{W}) \\ \text{sum}(\mathbf{Z}_k^l(1 + s, 1) \odot \mathbf{W}) & \text{sum}(\mathbf{Z}_k^l(1 + s, 1 + s) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{Z}_k^l(1 + s, 1 + t \cdot s) \odot \mathbf{W}) \\ \vdots & \vdots & \cdots & \vdots \\ \text{sum}(\mathbf{Z}_k^l(1 + t \cdot s, 1) \odot \mathbf{W}) & \text{sum}(\mathbf{Z}_k^l(1 + t \cdot s, 1 + s) \odot \mathbf{W}) & \cdots & \text{sum}(\mathbf{Z}_k^l(1 + t \cdot s, 1 + t \cdot s) \odot \mathbf{W}) \end{pmatrix}$$

где $t = \left\lfloor \frac{n_l - k}{s} \right\rfloor$.

При использовании шага s результатом свертки тензора $\mathbf{Z}^l \in \mathbb{R}^{n_l \times n_l \times m_l}$ с фильтром $\mathbf{W} \in \mathbb{R}^{k \times k \times m_l}$ является $(t + 1) \times (t + 1)$ -матрица.



В примере показана двумерная свертка с использованием шага $s = 2$ на матрице \mathbf{Z}^l размерами 5×5 ($n_l = 5$) с фильтром \mathbf{W} размерами 3×3 ($k = 3$). Вместо матрицы \mathbf{Z}^{l+1} размерами 3×3 для шага по умолчанию, равного единице, получаем матрицу \mathbf{Z}^{l+1} размерами $(t + 1) \times (t + 1) = 2 \times 2$, поскольку

$$t = \left\lfloor \frac{n_l - k}{s} \right\rfloor = \left\lfloor \frac{5 - 3}{2} \right\rfloor = 1.$$

При использовании шага индекс следующего окна увеличивается на шаг s по строкам и столбцам.

Например, первое окно – это $\mathbf{Z}_3^l(1, 1)$, второе окно – $\mathbf{Z}_3^l(1, 1 + s) = \mathbf{Z}_3^l(1, 3)$. Затем движемся от первого окна вниз на шаг $s = 2$, так что третье окно имеет вид $\mathbf{Z}_3^l(1 + s, 1) = \mathbf{Z}_3^l(3, 1)$, а последнее окно имеет вид $\mathbf{Z}_3^l(3, 1 + s) = \mathbf{Z}_3^l(3, 3)$.

Пример свертки со сдвигом ($s = 2$)


 \mathbf{Z}^l

1	2	2	1	1
3	1	4	2	1
2	1	3	4	3
1	2	3	1	1
4	1	3	2	1

 $*$
 \mathbf{W}

1	0	0
0	1	1
0	1	0

 $=$
 \mathbf{Z}^{l+1}

7	

 \mathbf{Z}^l

1	2	2	1	1
3	1	4	2	1
2	1	3	4	3
1	2	3	1	1
4	1	3	2	1

 $*$
 \mathbf{W}

1	0	0
0	1	1
0	1	0

 $=$
 \mathbf{Z}^{l+1}

7	9

 \mathbf{Z}^l

1	2	2	1	1
3	1	4	2	1
2	1	3	4	3
1	2	3	1	1
4	1	3	2	1

 $*$
 \mathbf{W}

1	0	0
0	1	1
0	1	0

 $=$
 \mathbf{Z}^{l+1}

7	9
8	

 \mathbf{Z}^l

1	2	2	1	1
3	1	4	2	1
2	1	3	4	3
1	2	3	1	1
4	1	3	2	1

 $*$
 \mathbf{W}

1	0	0
0	1	1
0	1	0

 $=$
 \mathbf{Z}^{l+1}

7	9
8	7



Пулинговый слой в сверточной нейронной сети призван снижать размерность изображения. Исходное изображение делится на блоки одинакового размера и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (max pooling) или среднего (average pooling). Обучаемых параметров у этого слоя нет.

Основными целями пулингового слоя являются:

- уменьшение изображения, чтобы последующие свертки оперировали над большей областью исходного изображения;
- увеличение инвариантности выхода сети по отношению к малому переносу входа;
- ускорение вычислений.

Пулинг представляет собой альтернативный тип агрегирования данных в дополнение к суммированию, используемый в сетях CNN.



Среднее значение (Avg-Pooling) по поэлементному произведению $\mathbf{Z}_k^l(i, j, q)$ и \mathbf{W} равно:

$$\begin{aligned}\text{avg}(\mathbf{Z}_k^l(i, j, q) \odot \mathbf{W}) &= \text{avg}_{\substack{a=1,2,\dots,k \\ b=1,2,\dots,k \\ c=1,2,\dots,r}} \{z_{i+a-1,j+b-1,q+c-1}^l \cdot w_{a,b,c}\} \\ &= \frac{1}{k^2 \cdot r} \cdot \text{sum}(\mathbf{Z}_k^l(i, j, q) \odot \mathbf{W})\end{aligned}$$

Максимум (Max-Pooling) по поэлементному произведению $\mathbf{Z}_k^l(i, j, q)$ и \mathbf{W} равен:

$$\max(\mathbf{Z}_k^l(i, j, q) \odot \mathbf{W}) = \max_{\substack{a=1,2,\dots,k \\ b=1,2,\dots,k \\ c=1,2,\dots,r}} \{z_{i+a-1,j+b-1,q+c-1}^l \cdot w_{a,b,c}\}$$



Обычно **макс-пулинг** (max-pooling) используется чаще, чем avg-pooling. Кроме того, для агрегирования очень часто шаг устанавливается равным размеру фильтра ($s = k$), так что функция агрегирования применяется к непересекающимся окнам размера $k \times k$ в каждом канале в \mathbf{Z}^l .

Что еще более важно, при агрегировании фильтр \mathbf{W} по умолчанию принимается равным тензору размера $k \times k \times 1$, все веса которого фиксированы и равны 1, так что $\mathbf{W} = \mathbf{1}_{k \times k \times 1}$.

Другими словами, веса фильтра фиксируются на 1 и не обновляются во время обратного распространения. Кроме того, фильтр использует фиксированное нулевое смещение (то есть $b = 0$).

Наконец, обратите внимание, что пулинг неявно использует функцию тождественной активации. В этом случае свертка $\mathbf{Z}^l \in \mathbb{R}^{n_l \times n_l \times m_l}$ с фильтром $\mathbf{W} \in \mathbb{R}^{k \times k \times 1}$ с использованием шага $s = k$ создает тензор \mathbf{Z}^{l+1} размером $\left\lfloor \frac{n_l}{s} \right\rfloor \times \left\lfloor \frac{n_l}{s} \right\rfloor \times m_l$.



В примере показано применение Max-Pooling на матрице \mathbf{Z}^l размерами 4×4 ($n_l = 4$) с использованием размера окна $k = 2$ и шага $s = 2$, равного размеру окна.

Таким образом, результирующий слой \mathbf{Z}^{l+1} имеет размеры 2×2 , поскольку $\left\lfloor \frac{n_l}{s} \right\rfloor = \left\lfloor \frac{4}{2} \right\rfloor = 2$. Фильтр \mathbf{W} имеет фиксированные веса, равные 1.

Свертка первого окна \mathbf{Z}^l , а именно $\mathbf{Z}_2^l(1, 1)$ с фильтром \mathbf{W} задается как

$$\max(\mathbf{Z}_2^l(1, 1) \odot \mathbf{W}) = \max\left(\left(\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix} \odot \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\right)\right) = \max\left(\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}\right) = 3$$

Пример Max-Pooling со сдвигом ($s = 2$)



$$\begin{array}{c}
 \mathbf{Z}^l \\
 \begin{array}{|c|c|c|c|}
 \hline
 1 & 2 & 2 & 1 \\
 \hline
 3 & 1 & 4 & 2 \\
 \hline
 2 & 1 & 3 & 4 \\
 \hline
 1 & 2 & 3 & 1 \\
 \hline
 \end{array}
 \end{array}
 *
 \begin{array}{c}
 \mathbf{W} \\
 \begin{array}{|c|c|}
 \hline
 1 & 1 \\
 \hline
 1 & 1 \\
 \hline
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 \mathbf{Z}^{l+1} \\
 \begin{array}{|c|c|}
 \hline
 3 & \\
 \hline
 & \\
 \hline
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 \mathbf{Z}^l \\
 \begin{array}{|c|c|c|c|}
 \hline
 1 & 2 & 2 & 1 \\
 \hline
 3 & 1 & 4 & 2 \\
 \hline
 2 & 1 & 3 & 4 \\
 \hline
 1 & 2 & 3 & 1 \\
 \hline
 \end{array}
 \end{array}
 *
 \begin{array}{c}
 \mathbf{W} \\
 \begin{array}{|c|c|}
 \hline
 1 & 1 \\
 \hline
 1 & 1 \\
 \hline
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 \mathbf{Z}^{l+1} \\
 \begin{array}{|c|c|}
 \hline
 3 & 4 \\
 \hline
 & \\
 \hline
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 \mathbf{Z}^l \\
 \begin{array}{|c|c|c|c|}
 \hline
 1 & 2 & 2 & 1 \\
 \hline
 3 & 1 & 4 & 2 \\
 \hline
 2 & 1 & 3 & 4 \\
 \hline
 1 & 2 & 3 & 1 \\
 \hline
 \end{array}
 \end{array}
 *
 \begin{array}{c}
 \mathbf{W} \\
 \begin{array}{|c|c|}
 \hline
 1 & 1 \\
 \hline
 1 & 1 \\
 \hline
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 \mathbf{Z}^{l+1} \\
 \begin{array}{|c|c|}
 \hline
 3 & 4 \\
 \hline
 2 & \\
 \hline
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 \mathbf{Z}^l \\
 \begin{array}{|c|c|c|c|}
 \hline
 1 & 2 & 2 & 1 \\
 \hline
 3 & 1 & 4 & 2 \\
 \hline
 2 & 1 & 3 & 4 \\
 \hline
 1 & 2 & 3 & 1 \\
 \hline
 \end{array}
 \end{array}
 *
 \begin{array}{c}
 \mathbf{W} \\
 \begin{array}{|c|c|}
 \hline
 1 & 1 \\
 \hline
 1 & 1 \\
 \hline
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 \mathbf{Z}^{l+1} \\
 \begin{array}{|c|c|}
 \hline
 3 & 4 \\
 \hline
 2 & 4 \\
 \hline
 \end{array}
 \end{array}$$



В типичной архитектуре сети CNN чередуются слои свертки (с суммированием в качестве функции агрегации, а также обучаемыми весами фильтра и коэффициентом смещения) и слои пулинга (например, макс-пулинга с фиксированным фильтром из единиц).

Идея состоит в том, что в то время как слой свертки обучает фильтры для извлечения информативных признаков, слой пулинга применяет функцию агрегирования, такую как \max (или avg), для извлечения наиболее важного значения нейрона (или среднего значения нейрона) в каждом скользящем окне для каждого из каналов.

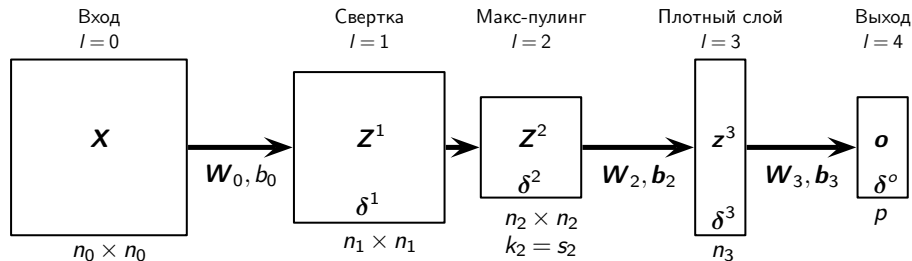
Начиная с входного слоя, глубокая сеть CNN состоит из нескольких, обычно чередующихся, слоев свертки и пулинга, за которыми следуют один или несколько полносвязных слоев, а затем последний выходной слой.

Для каждого слоя свертки и пулинга нужно выбрать размер окна k , а также значение шага s и использовать заполнение (padding) p или нет. Также нужно выбрать нелинейные функции активации для сверточных слоев, а также количество рассматриваемых слоев.



Чтобы увидеть, как обучать сеть CNN, рассмотрим сеть с одним слоем свертки и слоем макс-пулинга, за которым следует полносвязный слой.

Для простоты будем считать, что на входных данных \mathbf{X} имеется только один канал, и далее используем только один фильтр.





Обозначим через $\mathbf{D} = \{\mathbf{X}_i, \mathbf{y}_i\}_{i=1}^n$ обучающие данные, включающие n тензоров $\mathbf{X}_i \in \mathbb{R}^{n_0 \times n_0 \times m_0}$ ($m_0 = 1$ для простоты) и через $\mathbf{y}_i \in \mathbb{R}^p$ соответствующий вектор отклика.

Для обучающей пары $(\mathbf{X}, \mathbf{y}) \in \mathbf{D}$ на этапе прямого прохода прогнозируемый результат \mathbf{o} задается с помощью следующих уравнений:

$$\mathbf{Z}^1 = f^1 ((\mathbf{X} * \mathbf{W}_0) + b_0),$$

$$\mathbf{Z}^2 = \mathbf{Z}^1 *_{s_2, \max} \mathbf{1}_{k_2 \times k_2},$$

$$\mathbf{Z}^3 = f^3 (\mathbf{W}_2^T \mathbf{Z}^2 + \mathbf{b}_2),$$

$$\mathbf{o} = f^o (\mathbf{W}_3^T \mathbf{Z}^3 + \mathbf{b}_3),$$

где $*_{s_2, \max}$ обозначает макс-пулинг с шагом s_2 .



Пусть δ^1 , δ^2 и δ^3 обозначают чистые векторы градиента в слоях $l = 1, 2, 3$ соответственно и пусть δ^o обозначает чистый вектор градиента в выходном слое.

Выходной чистый вектор градиента может быть получен обычным способом путем вычисления частных производных функции потерь ($\partial \mathcal{E}_X$) и функции активации (∂f^o):

$$\delta^o = \partial f^o \odot \partial \mathcal{E}_X$$

при условии, что выходные нейроны независимы.

Так как слой $l = 3$ полностью соединен с выходным слоем и, аналогично, слой макс-пулинга $l = 2$ полностью соединен с \mathbf{Z}^3 , чистые векторы градиента на этих слоях вычисляются так же, как в обычной сети MLP:

$$\delta^3 = \partial f^3 \odot (\mathbf{W}_o \cdot \delta^o),$$

$$\delta^2 = \partial f^2 \odot (\mathbf{W}_2 \cdot \delta^3) = \mathbf{W}_2 \cdot \delta^3$$



Последнее выражение вытекает из равенства $\partial f^2 = 1$, так как макс-пулинг неявно использует тождественную функцию активации. Рассмотрим чистый градиент δ_{ij}^1 в нейроне z_{ij}^1 в слое $l = 1$ при $i, j = 1, \dots, n_1$:

$$\delta_{ij}^1 = \begin{cases} \delta_{ij}^2 \cdot \partial f_{ij}^1 & \text{при } i = i^*, j = j^* \\ 0 & \text{в противном случае} \end{cases}$$

Другими словами, чистый градиент на нейроне z_{ij}^1 в слое свертки равен нулю, если этот нейрон не имеет максимального значения в своем окне.

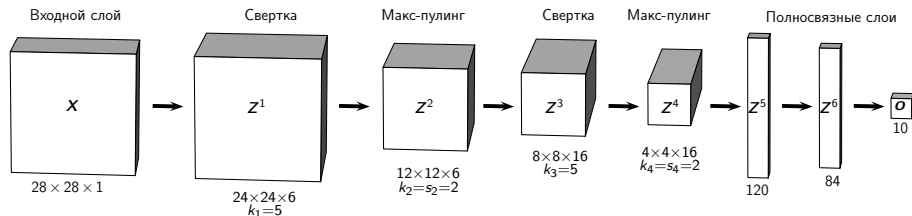
В противном случае, если значение является максимальным на этом нейроне, то чистый градиент распространяется обратно от слоя макс-пулинга к этому нейрону, а затем умножается на частную производную функции активации.

Матрица чистых градиентов δ^1 содержит чистые градиенты δ_{ij}^1 для всех $i, j = 1, \dots, n_1$.



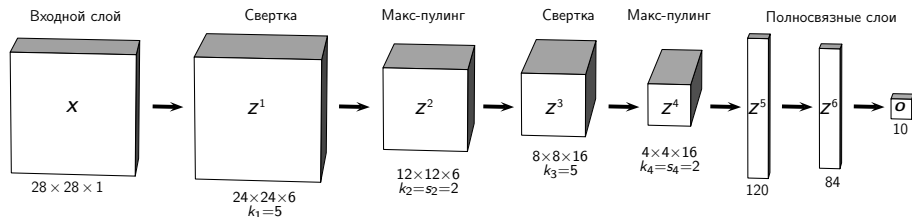
Пример показывает применение сети CNN для распознавания рукописных цифр. Эта CNN обучена и протестирована на наборе данных MNIST, который содержит 60 000 обучающих изображений и 10 000 тестовых изображений.

Каждое входное изображение представляет собой матрицу размерами 28×28 со значениями пикселей от 0 до 255, которые делятся на 255, так что каждый пиксель находится в интервале $[0, 1]$. Соответствующий (истинный) выход \mathbf{y}_i представляет собой двоичный вектор с однократным кодированием, который обозначает цифру от 0 до 9; цифра 0 кодируется как $\mathbf{e}_1 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$, цифра 1 как $\mathbf{e}_2 = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)^T$ и так далее.





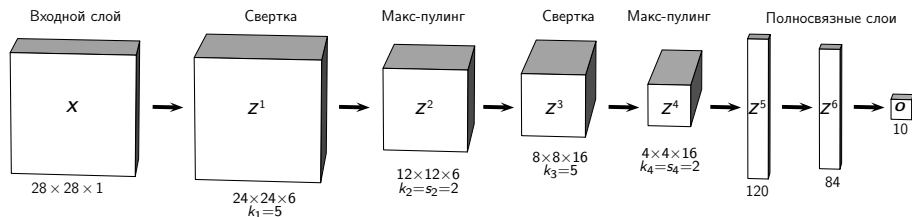
В представленной сети CNN все слои свертки используют шаг, равный единице, и не используют заполнение (padding), тогда как все слои макс-пулинг используют шаг, равный размеру окна. Поскольку каждый вход представляет собой изображение цифры размером 28×28 пикселей с одним каналом (оттенки серого), то $n_0 = 28$ и $m_0 = 1$, и, следовательно, вход $\mathbf{X} = \mathbf{Z}^0$ представляет собой тензор $n_0 \times n_0 \times m_0 = 28 \times 28 \times 1$. Первый слой свертки использует $m_1 = 6$ фильтров с $k_1 = 5$ и шагом $s_1 = 1$ без заполнения (padding).





Каждый фильтр представляет собой тензор весов размерами $5 \times 5 \times 1$, и для шести фильтров результирующий тензор \mathbf{Z}^1 слоя $l = 1$ имеет размеры $24 \times 24 \times 6$, при этом $n_1 = n_0 - k_1 + 1 = 28 - 5 + 1 = 24$ и $m_1 = 6$. Второй скрытый слой представляет собой слой макс-пулинга ($k_2 = 2$ и $s_2 = 2$).

Поскольку макс-пулинг по умолчанию использует фиксированный фильтр $\mathbf{W} = \mathbf{1}_{k_2 \times k_2 \times 1}$, результирующий тензор \mathbf{Z}^2 имеет размеры $12 \times 12 \times 6$, при этом $n_2 = \left\lfloor \frac{n_1}{k_2} \right\rfloor = \left\lfloor \frac{24}{2} \right\rfloor = 12$ и $m_2 = 6$. Третий слой представляет собой сверточный слой с $m_3 = 16$ каналами с размером окна $k_3 = 5$ (и шагом $s_3 = 1$), что дает тензор \mathbf{Z}^3 размерами $8 \times 8 \times 16$, в котором $n_3 = n_2 - k_3 + 1 = 12 - 5 + 1 = 8$.

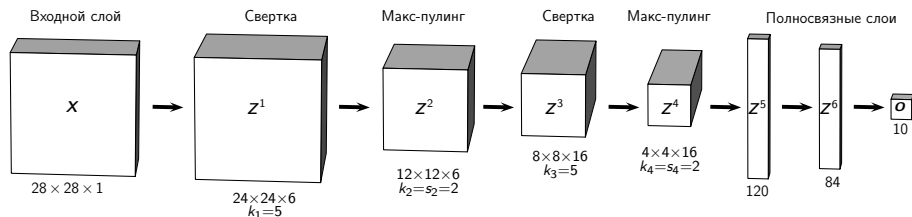




Затем следует еще один слой макс-пулинга, который использует $k_4 = 2$ и $s_4 = 2$, что дает тензор \mathbf{Z}^4 с размерами $4 \times 4 \times 16$, где $n_4 = \left\lfloor \frac{n_3}{k_4} \right\rfloor = \left\lfloor \frac{8}{2} \right\rfloor = 4$, а $m_4 = 16$.

Следующие три слоя полностью связаны, как в обычной сети MLP. Все 256 нейронов ($256 = 4 \times 4 \times 16$) в слое $l = 4$ связаны с нейронами тензора \mathbf{Z}^5 слоя $l = 5$, который представляет собой просто вектор, и его можно рассматривать как вырожденный тензор размерами $120 \times 1 \times 1$.

Слой $l = 5$ также полностью связан со слоем $l = 6$ с 84 нейронами. Выход \mathbf{o} имеет 10 нейронов и использует функцию активации softmax. Нейроны в тензорах \mathbf{Z}^1 , \mathbf{Z}^3 , \mathbf{Z}^5 и \mathbf{Z}^6 используют активацию ReLU.



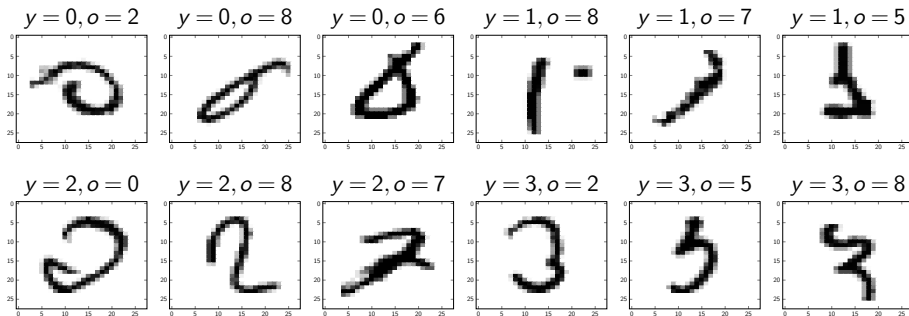


Модель сети CNN обучается на $n = 60000$ изображениях из набора данных MNIST. Обучение проводится в течение 15 эпох с размером шага обучения $\eta = 0.2$ и функцией кросс-энтропийной ошибки (поскольку в данных встречается 10 классов).

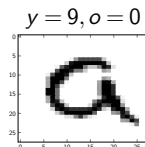
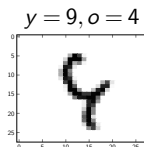
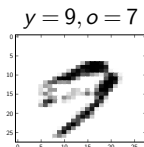
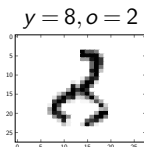
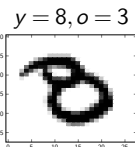
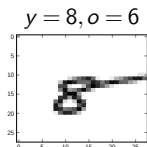
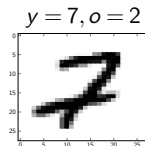
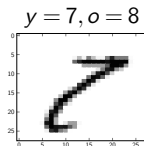
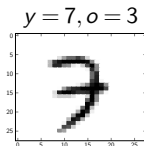
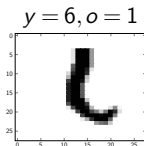
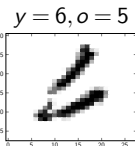
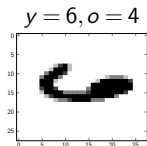
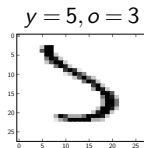
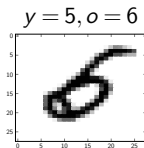
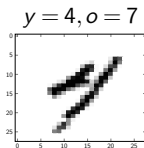
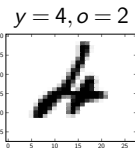
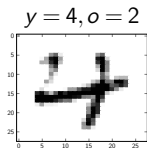
Обучение проводилось с использованием мини-пакетов с размером пакета 1000. После обучения модель CNN оценивается на тестовом наборе данных из 10 000 изображений. Модель CNN допускает 147 ошибок в тестовом наборе, что приводит к частоте ошибок 1.47%.



На рисунке показаны примеры изображений, которые были неправильно классифицированы сетью CNN. Видно, что некоторые из неправильно классифицированных изображений зашумлены, неполны или ошибочны, и их трудно правильно классифицировать даже человеку.



MNIST: некорректные предсказания





Для сравнения также обучается глубокая сеть MLP с двумя полносвязными скрытыми слоями с теми же размерами, что и два полносвязных слоя перед выходным слоем в CNN, показанном на рисунке.

Следовательно, сеть MLP состоит из слоев \mathbf{X} , \mathbf{Z}^5 , \mathbf{Z}^6 и \mathbf{o} , при этом входные изображения размерами 28×28 рассматриваются как вектор размера $d = 784$. Первый скрытый слой имеет размер $n_1 = 120$, второй скрытый слой имеет размеры $n_2 = 84$, а выходной слой имеет размеры $p = 10$. Функция активации ReLU используется для всех слоев, кроме выходного, который использует функцию softmax. Мы обучаем модель MLP для 15 эпох на обучающем наборе данных с $n = 60000$ изображений, используя размер шага $\eta = 0.5$. В тестовом наборе данных сеть MLP допустила 264 ошибки, что соответствует частоте ошибок 2.64%.



На графике показано количество ошибок в тестовом наборе после каждой эпохи обучения как для модели сети CNN, так и для модели сети MLP. Модель сети CNN обеспечивает значительно лучшую точность, чем MLP.

