Конспект лекции

А.Н. Шевляков

Введение в нейронные сети

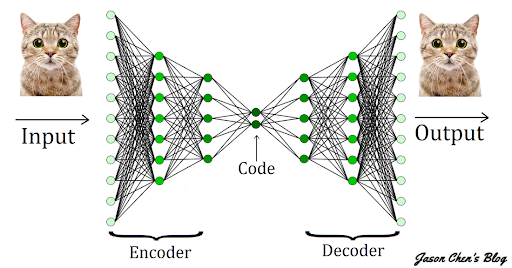
Автокодировщики

# Постановка задачи

Давайте сделаем НС, которая должна выдать ответ, в точности равный входу (вход и выход не обязательно могут быть картинками).

Неужели так сложно подобрать веса НС, чтобы она осуществляла бы тождественное преобразование. То есть ее функция равна FNN(x)=x.

Конечно, сделать так, чтобы внутренние нейроны перенесли всю информацию от начала к концу НС, не сложно. А мы возьмём и усложним жизнь нашей НС!

А давайте сделаем специально слой в центре НС очень узким, то есть число нейронов на этом слое меньше размерности входа (выхода) НС.

Что это даст? Мы **заставляем** НС находить в объекте самую важную информацию и только ее передавать через бутылочное горлышко.

Как подобрать оптимальные веса автокодировщика?

Для этого нужно составить **функцию потерь автокодировщика**!

## Функция потерь автокодировщика

Если на вход АК подается вектор чисел x=(x1,...,xn), то функция сети FNN(x) — это тоже вектор FNN(x)=(F1(x),...,Fn(x)).

То есть

число x1 превращается в число F1(x),

число x2 превращается в число F2(x),

...

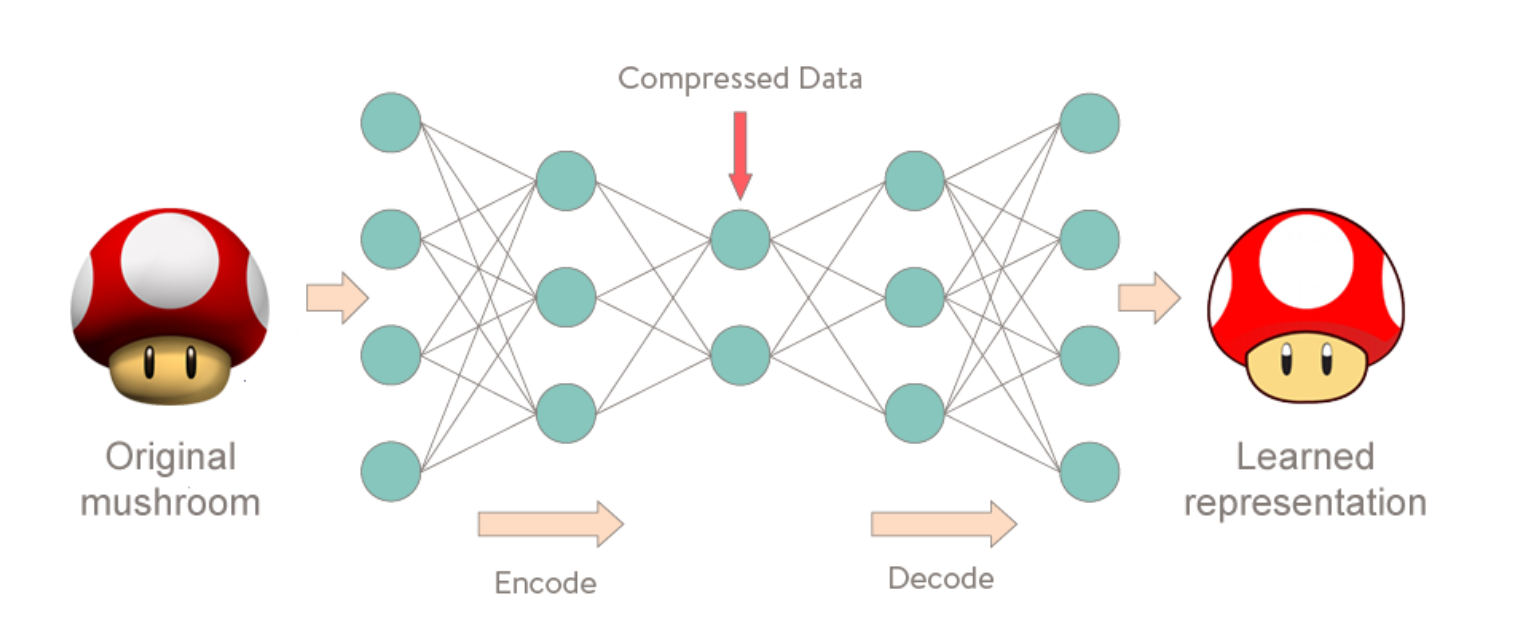
число xn превращается в число Fn(x).

Составим квадрат разности чисел xi  и того, во что они превратились. Просуммируем: L(x)=(F1(x)-x1)2+(F2(x)-x2)2+...+(Fn(x)-xn)2 — **потери на кодировке объекта х**.

А чтобы получить функцию потерь для всей ТВ, необходимо **просуммировать потери всех объектов ТВ**. Осталось найти минимум

этой функции с помощью ГС.

Название частей АК:

* кодирующая часть;
* бутылочное горлышко (слой сжатия);
* декодирующая часть.

**Важная идея:** АК превращает объект Х на входе НС в числовой вектор е(Х) — вектор значений, выходящих из нейронов бутылочного горлышка.

То есть, если бутылочное горлышко состоит из N нейронов, то объект Х будет представлен в виде **вектора e(X) длины N**. Идея превращать сложные объекты в числовые вектора очень важна в ML. Это называется **embedding** (или числовым представлением).

Embedding с помощью АК обладает одним **важным свойством:** похожие объекты X1, X2 будут закодированы близкими друг к другу векторами e(X1), e(X2).

Первое применение АК **- архиватор.** Объект Х превращаем в вектор e(X) и храним его вместо объекта X.

Второе применение **- подавление шумов.** Изображение нужно очистить от шума. Но как натренировать АК? **Тренировка АК**: берём изображение Х, генерируем случайный шум, получаем зашумлённое изображение Х’. Тренируем АК на парах (X’,X). Итак, АК натренирован. Теперь ему на вход поступает зашумлённое изображение с неизвестным ответом. Мы его просто пропускаем через АК, и на выходе будет очищенное от шума изображение.

Почему это работает?АК понимает, что протащить всю информацию об изображении через бутылочное горлышко не получится, и поэтому через горлышко проходят только самые существенные части изображения, а всё случайное (шум) — отсеивается.

Третье применение **- помощь в классификации.** В задачах классификации НС необходимо предсказать метку класса у объекта. Грубо говоря, НС пытается угадать, что изображено на картинке. Такая НС тренируется на размеченной (тренировочной) выборке.

Размеченная выборка — это выборка объектов, для которых известен их точный класс. К сожалению, эта выборка может быть недостаточно хорошей по следующим причинам:

* Размер выборки — объектов в выборке слишком мало для качественного обучения НС.
* Дорогостоящая разметка — разметить выборку крайне тяжело, так как для этого необходимо разметить достаточно много качественных исходных данных (к примеру, рентгеновских снимков) и найти специалистов, которые корректно сделают разметку (например, высококвалифицированных рентгенологов).

Как тут помогут АК? Пусть у нас небольшая размеченная выборка, но много неразмеченных картинок. Пусть Р1,...,Рn — множество неразмеченных картинок. Мы по парам (Р1,Р1),...,(Рn,Рn) тренируем АК. После этого оставляем от АК кодирующие слои и добавляем новые слои для распознавания изображений. Получается такая сеть-монстр. Теперь по размеченной выборке тренируем сеть-монстра.

**Важно**: веса между нейронами АК «заморожены», то есть они уже **не меняются.**

АК реально помогают решать задачу классификации. Можно иметь порядка тысяч неразмеченных изображений и несколько десятков размеченных. И тем не менее данный подход позволит правильно всё классифицировать.

Четвёртое применение **- генерация новых изображений.** Мы знаем, что бутылочное горлышко АК изображение Х превращает в вектор е(Х). А потом вектор е(Х) проходит через слой декодирования и получается похожее на Х изображение.

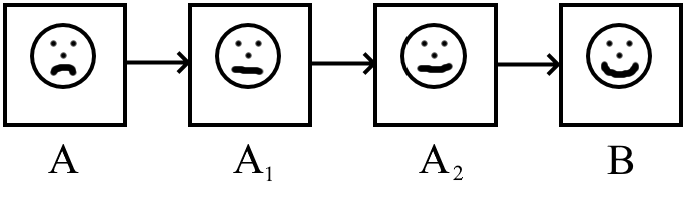
Мы можем слегка менять вектор e(X) перед отправкой его в декодирующую часть (подаём вектор e(X)+e’). В этом случае и итоговое изображение будет отлично от Х. Степень похожести сгенерированного изображения на оригинал Х определяется величиной сдвига e’.

А вот как менять вектор e(X) осмысленно, чтобы получалось новое изображение с требуемыми свойствами?

Если бутылочное горлышко АК превращает изображения А,В в вектора e(A), e(B), то для генерации промежуточного изображения можно взять вектор «между ними».

**Промежуточный вектор задаётся формулой:**

**αe(A)+(1-α)e(B),** где α — число из интервала (0,1).



Допустим, что мы хотим добавлять улыбку (очки, бороду...) на фото людей.

Для этого мы прогоним через АК фото всех улыбающихся людей и получим соответствующие им вектора е(Х). **Вот это облако точек**

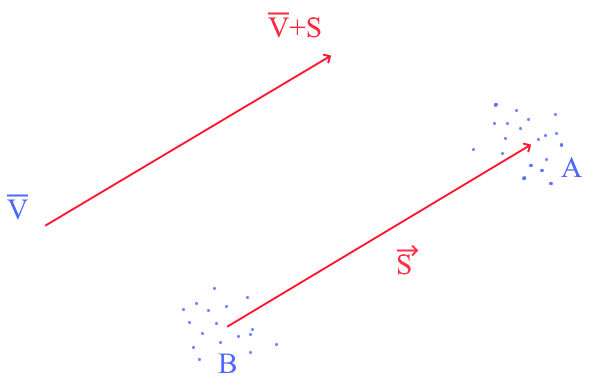
Прогоним через АК фото грустных людей.

Соответствующее им **облако векторов е(Х)** находится здесь:

Найдём центр у каждого облака. Соединим центры вектором S.

**Ну вот и все**: вектор S — вектор, отвечающий за улыбку.

Как им пользоваться?



Пусть нам надо наложить улыбку на грустное фото. Бутылочное горлышко АК превращает это фото в вектор e(X).

Добавляем к нему вектор S. Оказывается, что вектор e(X)+S при пропуске через декодирующую часть АК даст фото того же самого человека, но с улыбкой.

