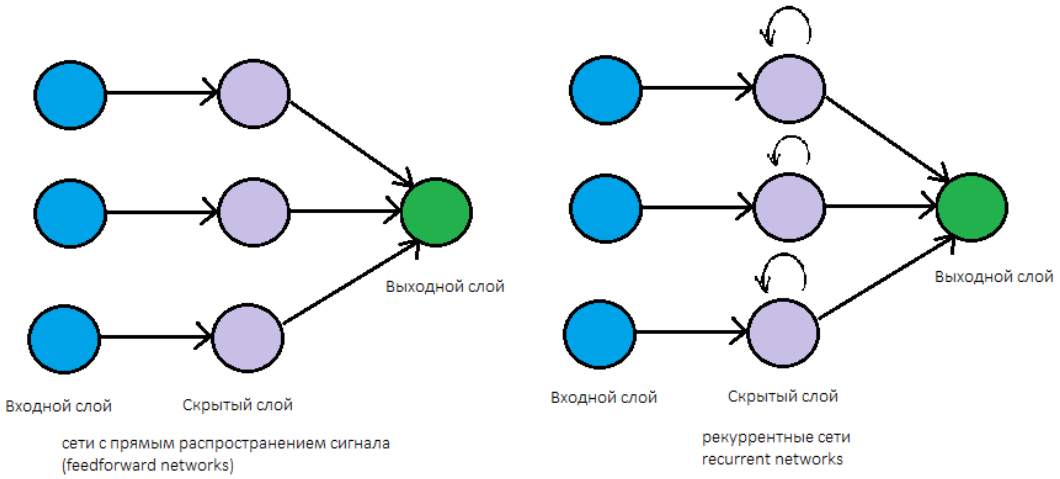
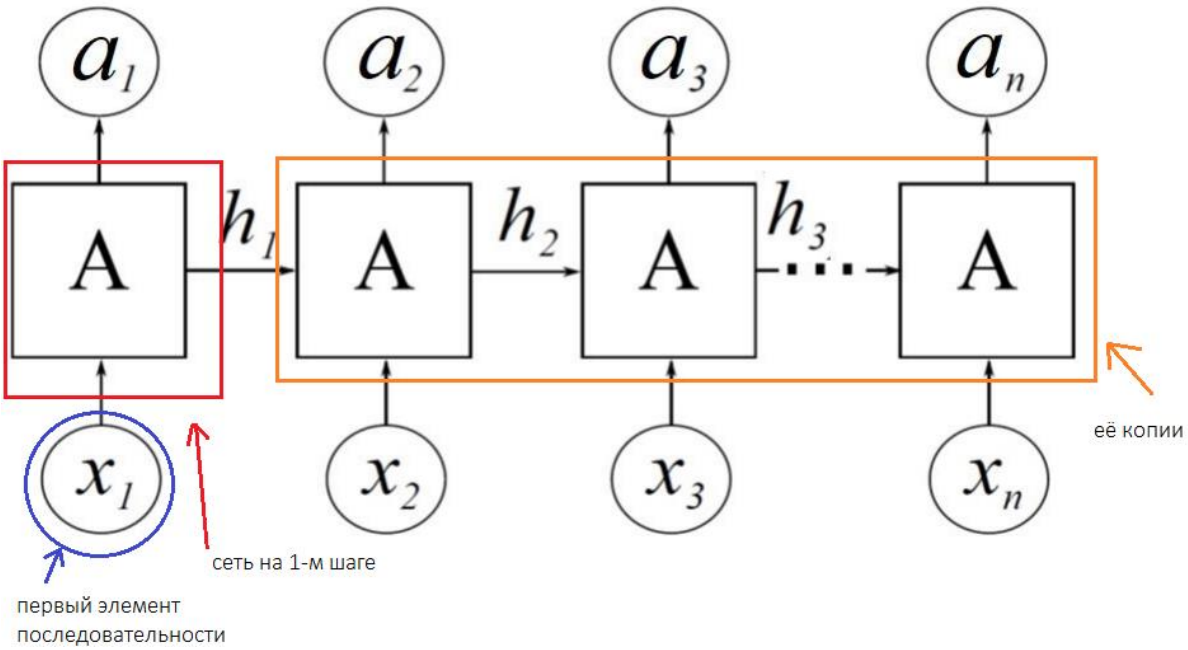
1. Что такое рекуррентная сеть?

Нейросеть, в которой, в отличие от многослойного перцептрона и сверточной нейросети, нейроны дополнительно хранят информацию о предыдущем состоянии сети, у нее будто есть «память». РНС хорошо подходят для задачи анализа последовательностей. В сетях с прямым распространением сигнала запрещены циклы, В то время как в рекуррентных разрешены. В этом случае выход нейрона может быть подключен к его входу, ко входу всех нейронов в текущем слое (такой подход используется чаще всего), а также теоретически можно подключить выход нейрона к любому другому нейрону в любому слое. Через эти циклические соединения и поступает информация о том, что было на предыдущем шаге работы сети или даже на нескольких предыдущих шагах. Так, она анализирует текст не как набор изолированных токенов, а как последовательность.



Рекуррентную нейросеть можно представить в виде сети с прямым распространением сигнала, если использовать прием разворачивания во времени: создается несколько копий РНС, на вход которых поступают элементы последовательности:



РНС выдает два значения: первое значение a — выходное, поступает на выход нейронной сети, а второе — значение h — поступает на вход копии сети в следующий момент времени. Это можно назвать скрытым состоянием, учитывающим то, что было на предыдущих этапах анализа последовательности. Копия нейронной сети в следующий момент времени на вход получает второй элемент последовательности, а также скрытое состояние с предыдущего этапа. Для последнего элемента последовательности РНС выдает только одно выходное значение без скрытого состояния.

Пример создания (задача: анализ текста):

```
model = Sequential() # создаем последовательную модель
model.add(Embedding(input_dim=max_words, output_dim=50, input_length=maxlen)) # для представления текста в виде чисел используем плотное векторное представление (слой Embedding)
model.add(SimpleRNN(8)) # добавляем в модель рекуррентный слой с 8 нейронами, выход каждого нейрона в этом слое подключен ко входам всех нейронов
model.add(Dense(1, activation='sigmoid')) # полносвязный слой с одним нейроном, служащий для классификации текста
```

2. Как обычная свертка раскладывается на поканальную свертку?

Как говорилось на лекции, сначала входящий тензор сворачивается ядром (матрицей) свертки размерностью 1x1x(кол-во каналов входящего тензора), а затем в получившемся результате сначала делается свертка 3x3 по каждому каналу, затем 1x1, однако порядок не влияет на конечный результат, что было доказано практически.

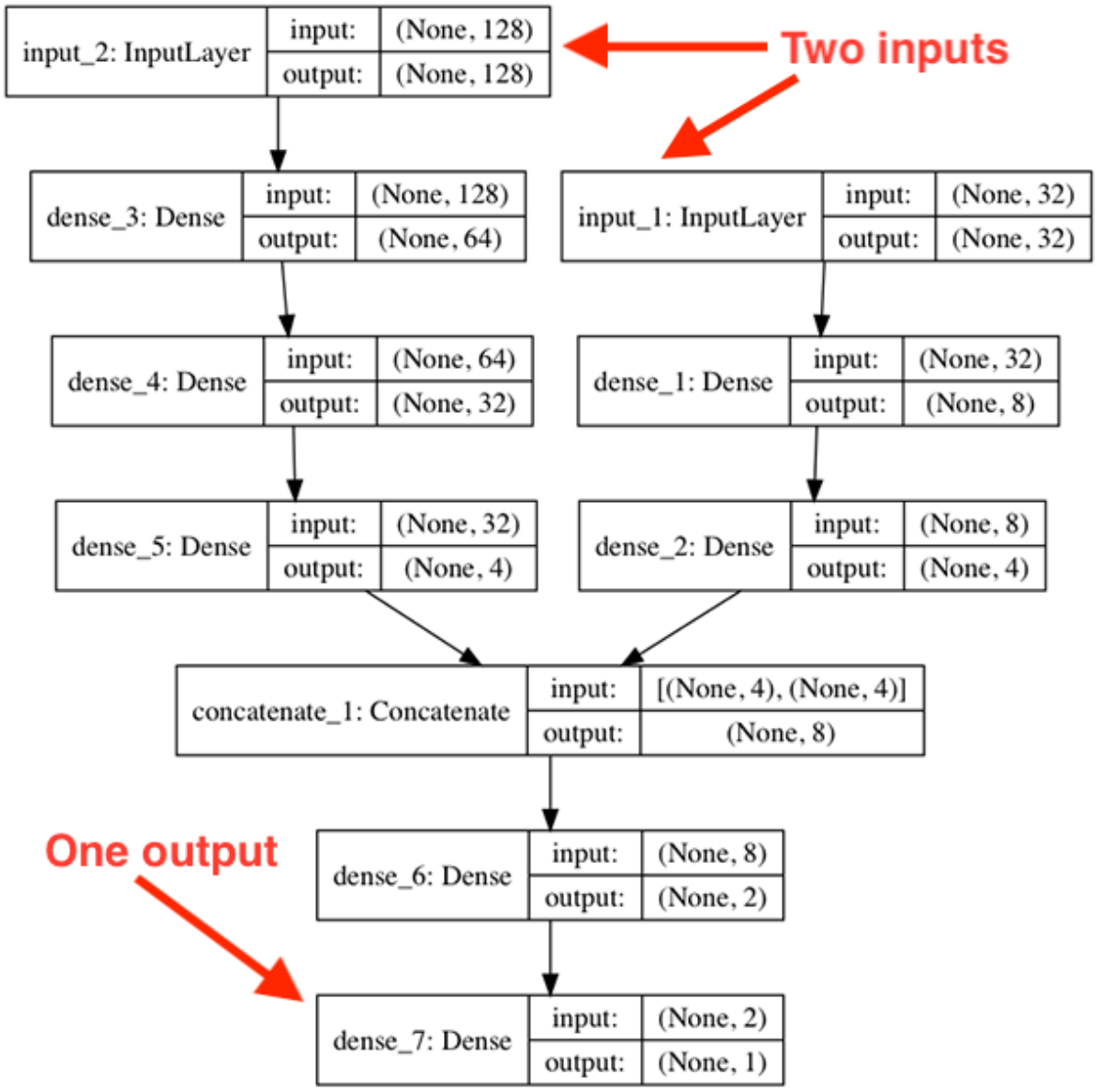
3. В каких случаях может быть необходимо использовать 2 входных слоя в ИНС?

Есть такое понятие, как mixed data. В машинном обучении оно относится к концепции наличия нескольких типов независимых данных. Например, мы работаем в госпитале и нам надо разработать систему, способную классифицировать здоровье пациента. У нас будет несколько типов входных данных для каждого пациента, в том числе: числовые значения (возраст, давление, пульс, вес, т.п.), категориальные данные (пол, этническая принадлежность, т.п.), изображения (рентген, МРТ, ЭКГ, т.п). Все эти данные разных видов, но наша модель должна быть способной усвоить эти «смешанные данные» и сделать точные прогнозы на них. Тут нам понадобится даже аж 3 слоя. Но, допустим, нам для прогнозов не нужны данные из категориального раздела, такие как пол и этническая принадлежность, тогда входных слоев будет 2.

Выглядеть код будет примерно так:

```
# определим два сета входных данных
inputA = Input(shape=(32,))
inputB = Input(shape=(128,))
# первая «ветвь» (branch) работать будет с первым сетом данных
x = Dense(8, activation="relu")(inputA)
x = Dense(4, activation="relu")(x)
# получилась 32-8-4 сеть
x = Model(inputs=inputA, outputs=x)
# вторая со вторым
y = Dense(64, activation="relu")(inputB)
y = Dense(32, activation="relu")(y)
y = Dense(4, activation="relu")(y)
# получилась 128-64-32-4 сеть
```

```
y = Model(inputs=inputB, outputs=y)
# объединяем вывод двух ветвей
combined = concatenate([x.output, y.output])
# применим FC-слой (fully-connected layer) и затем регрессионное
прогнозирование (regression prediction) на объединенных выводах
(outputs)
z = Dense(2, activation="relu")(combined)
z = Dense(1, activation="linear")(z) # linear activation это и
есть regression prediction
# модель примет входные данные обеих ветвей (двух inputs), и
выведет один вывод (output)
model = Model(inputs=[x.input, y.input], outputs=z)
```



4. Как изменится тензор [1 0.5 -1], если был применен слой Dropout с шансом отклонения 0.75, и был отброшен 1 и 3 элемент?

Как можно прочесть в документе, ссылка на который представлена на https://keras.io/layers/core/, выбрасывание нейронов происходит с помощью присоединения переменных Бернулли к нейронным выходам. Эти переменные принимают значение 1 с вероятностью p и 0 с вероятностью $1-p$. Но Keras перевернул эту логику и p в качестве параметра `rate` означает шансы отбросить нейрон, а не сохранить. То же правило действует в Tensorflow, начиная с версий 2.x. У нас $p=0.75$, получается, 25% нейронов должны сохраниться, то есть 1/4, у нас сохраняется 1/3>1/4, значит, отбрасывание 2-х из 3-х нейронов корректно, а отброшенные нейроны примут значение 0. Выходит: с вероятностью 0.75 элементы заданного тензора устанавливаются в 0, а оставшиеся элементы увеличиваются на 1.0 / (1 – 0.75), т.е. в 4 раза. (1.0 / (1 – `rate`) согласно документации, дабы сохранить ожидаемое значение). Таким образом, тензор преобразился в [0., 2., 0.]

Ответ: `[0., 2., 0.]`.

5. Для чего в Вашей модели нужен слой Flatten?

Коротко говоря: для классификации. Последняя стадия почти любой CNN, и заданной в том числе — это классификатор. Для классификации используется многослойный персептрон — полносвязные Dense слои (функция Dense() используется для полного соединения слоев друг с другом), a dense layer, как я прочитала, в принципе является ANN (artificial neural network) classifier-ом. ANN классификатору нужны индивидуальные черты, как и любому другому классификатору, потому он использует вектор из таких черт. Так, нам надо конвертировать вывод сверточной части CNN в 1D вектор из черт для последующей передачи классификатору. Эта операция и называется flattening. Если еще более прямо сказать, то dense не работает с многомерным input-ом как таковым. Ему можно скормить 3D данные, но они сначала flatten-ируются

(конвертируются в одномерные). Это значит, что если ввод имеет форму `(batch_size, sequence_length, dim)`, то dense layer сначала преобразует его в `(batch_size * sequence_length, dim)`, и потом уже наложит dense слой как обычно.