ML на клиенте: практикуемся на белочках

Чтобы пририсовывать лазеры из глаз

Генерировать умные фразы

Зачем фронтам ML ?

Победить всех в starcraft

Детектировать белочек

Чтобы создавать странное или красивое

Neural style transfer

Чтобы понимать аналитику

Убедить менеджеров в чем-нибудь :)

Анализировать поведение пользователей

Получать интересные инсайты

Чтобы разбирать статистику вашей сборки на винтики :)

Никита Сидоров

Яндекс

Почти 5 лет в Яндексе. За это время Никита успел поучиться в ШАДе, почитать и продолжить читать лекции в ШРИ. На Чемпионате Яндекса по программированию он заставил людей писать свой тришейкинг за 30 минут (поверьте, это реально сделать!). В продакшене Яндекса крутится его код на: JS, TS, Python, C++, Java и даже не падает.

Помимо острой любви к Яндексу (накопленные RSU тут не причем), увлекается темой code mining: поиск схожих участков кода через code2seq, minhashing и прочее.

Доклады

Фронтенд: Графы. Сборка. Мертвый код

8 июня | 13:30–14:30 | RU | Технический долг

Чтобы решать любые задачи

С чего начинается клиентский ML?

Попытка расшифровать доки №1

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Попытка расшифровать доки №2

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Чем заканчивается клиентский ML?

какие-то непонятные тензоры

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

злой питон

мат. ожидание?

$\mathbb{E}_x$

ожидаем мн только отпуска!

формулы с закорючками

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Я тут доклад один видел...

ML за 5 минут:

Декларативно: Я хочу чтобы… Не важно как

Императивно: Я делаю вот так… Не важно что я хочу

Императивно!

пишем что сделать

```
const createValueGetter = (a, b) => x => a * x + b
const getValue = createValueGetter(2, 5)
console.log([1, 2].map(getValue)) // [7, 9]
```

получаем результат

$(x_1) = \text{matrix1d}(x_0) + b$

Декларативно!

пишем что сделать

описываем результат

```
const inputs = [1, 2, 3, 4]
const realOutputs = [7, 9, 11, 13] // values.map(getOutput)

// ... some boring implementation of a network

const learnedParams = doTrain()
const result = layer(inputs, ...learnedParams)
```

оно само!

Ну если вы настаиваете…

```
// императивненько
const createValueGetter = (a, b) => x => a * x + b
const getOutput = createValueGetter(2, 5)
console.log([1, 2].map(getOutput)) // [7, 9]

const inputs = [1, 2, 3, 4]

const realOutputs = [7, 9, 11, 13] // values.map(getOutput)
// const getValueWithML =  ??

// готовим декларативность :)

const trainStep = (a, b, inputs, realOutputs, step) => {
  const outputs = layer(inputs, a, b)
  const gradL = outputs.map((y, index) => y - realOutputs[index])

  const gradA = gradL.map((gr, i) => gr * outputs[i]).reduce((a, b) => a + b, 0)
  const gradB = gradL.reduce((a, b) => a + b, 0)

  return [a - gradA * step, b - gradB * step]
}

// задаем начальные параметры
const learningRate = 0.001
const numberOfSteps = 10000
const initialParams = [Math.random(), Math.random()]
```

```
// задаем нашу "архитектуру"
const layer = (inputs: number[], ...params: [number, number]): number[] =>
  inputs.map(x => params[0] * x + params[1])

// задаем как сравнивать результаты
const loss = (outputs: number[], realOutputs: number[]): number =>
  outputs
    .map((y, index) => Math.pow(y - realOutputs[index], 2))
    .reduce((a, b) => a + b, 0)

// 🏋️
const doTrain = (): [number, number] =>
  [...Array(numberOfSteps)].reduce((currentParams: [number, number]) => {
    console.log(
      ...currentParams,
      loss(layer(inputs, ...currentParams), realOutputs)
    )
    return trainStep(...currentParams, inputs, realOutputs, learningRate)
  }, initialParams)

const learnedParams = doTrain()

const result = layer(inputs, ...learnedParams)
console.log(result)
```

… чуть-чуть кода

Модель — черный ящик

```
const inputs = [1, 2, 3, 4]

const realOutputs = [7, 9, 11, 13]
```

Что хорошо, а что плохо?

```
const loss = (outputs: number[], realOutputs: number[]): number =>
  outputs
    .map((y, index) => Math.pow(y - realOutputs[index], 2))
    .reduce((a, b) => a + b, 0)
```

Метрика хорошести

Алгоритм определяет архитектуру ящика

Архитектура ящика: торт «Наполеон»

```
const layer = (inputs: number[], a: number, b: number): number[] =>
  inputs.map(x => a * x + b)
```

```
const networkOutput =
  // ... layer 100
  layer3(layer2(layer1(inputs)))
```

Вот так все и работает

Прогоняем данные и обновляем слои

```
const trainStep = (a, b, inputs, realOutputs, step) => {
  const outputs = layer(inputs, a, b)
  const gradL = outputs.map((y, index) => y - realOutputs[index])

  const gradA = gradL.map((gr, i) => gr * outputs[i]).reduce((a, b) => a + b, 0)
  const gradB = gradL.reduce((a, b) => a + b, 0)

  return [a - gradA * step, b - gradB * step]
}
```

градиенты

Когда это работает?

У нас есть какая-то зависимость между входными и выходными параметрами

Мы как-то можем посчитать потери

Функция потерь достаточно симпатичная (простите меня, товарищи небезразличные к математике)

универсальная теорема аппроксимации

А что там в этих слоях?)

Тензоры и матрицы 👻

У тензора есть ранг и размерность.

Ранг – сколько индексов нам понадобится чтобы достать элемент из тензора.

Размерность – количество элементов по каждой из осей.

Тензоры описывают преобразования между элементами какого-нибудь пространства

Тензор *можно представить* в виде матрицы

Лирическое отступление №1: заблюрим белочку

```
const convStep = (arr1: number[], kernel): number =>
  kernel.flat().reduce((acc, v, i) => acc + v * arr1[i], 0)

const convolve = (
  array: Uint8ClampedArray,
  kernel: number[][],
  w: number,
  h: number,
  stride = 1,
  chInImage = 4
): Uint8ClampedArray => {
  const result = new Uint8ClampedArray(w * h * chInImage).fill(255)
  const kh = kernel.length
  const kw = kernel[0].length

  for (let i = 0; i < w - kw; i += stride) {
    for (let j = 0; j < h - kh; j += stride) {
      for (let c = 0; c < chInImage; c++) {
        const arrToConsolve: number[] = []
        for (let k = 0; k < kw; k++) {
          for (let l = 0; l < kh; l++) {
            arrToConsolve.push(
              array[
                chInImage * w * j +
                  chInImage * i +
                  c +
                  chInImage * k +
                  chInImage * l * kw
              ]
            )
          }
        }

        const convStepResult = convStep(arrToConsolve, kernel)
        result[chInImage * w * j + chInImage * i + c] = convStepResult
      }
    }
  }

  return result
}
```

вчч! 5 вложенных циклов

```
const convolve = (
  array: Uint8ClampedArray,
  kernel: number[][],
  w: number,
  h: number,
  stride = 1,
  chInImage = 4
): Uint8ClampedArray
```

$$\begin{matrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{matrix}$$

„свёртка"

Магия заблюривания (Convolution)

=

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

×	×	×
×	×	×
×	×	×

0	0	0
0	0	0
0	0	255

15

```
[[[0.0625 0.125  0.0625]
  [0.125  0.25   0.125 ]
  [0.0625 0.125  0.0625]]]
```

```
[[  0   0   0   0   0   0 255 150   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0 255   0 232   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0 255   0   0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 255   0   0 101   0   0   0   0   0   0   0   0   0   0   0 254 255 255 255 148   0   0   0]
 [  0   0   0   0   0 255   0   0   0   0   0   0   0   0   0   0 189 255 166   0   0   0   0 255   0   0   0]
 [  0   0   0   0   0   0 255   0 255   0   0   0   0 255 247   0   0   0   0   0   0   0   0   0 255   0   0]
 [  0   0   0   0   0   0  84   0 250   0   0   0 255   0   0   0   0   0   0   0   0   0   0 139 242   0   0]
 [  0   0   0   0   0   0   0 255   0 255   0 104   0   0   0   0   0   0   0   0   0   0  26  25   0 237   0]
 [  0   0   0   0   0   0   0 186   0   0 114   0   0   0   0   0   0   0   0   0 181 253   0   0 255   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255  68   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 197   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  58   0   0]
 [  0   0   0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   5 252 250 252 254 254]
 [  0   0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0 231   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  77 255   0   0   0]
 [  0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  22   0   0   0   0]
 [134  34   0   0   0 217 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255   0   0   0   0   0   0]
 [255   0 195 255 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4 254   0   0   0   0   0   0]
 [ 84   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255 255   0   0   0]
 [255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 252 245   0   0   0]
 [  0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 163   0   0   0   0   0]
 [  0  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255   0   0  13 255 253]
 [  0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  25   0   0   0   0   0]
 [  0   0   0 224   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0 255  12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0 243   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   3 122   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  27]
 [  0   0   0   0   0   0   0   0 255   3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255]
 [  0   0   0   0   0   0   0   0   0   5 252  45   0   0   0   0   0   0   0   0   0   0   0   0   0   0 243]
 [  0   0   0   0   0   0   0   0   0   0   0  23 255   0   0   0   0   0   0   0   0   0   0   0   0 255   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 255 255  60   0   0   0   0   0   0 255   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  74 255 255 255 255 255 255   0   0   0]]
```

```
[[  0  15  31  47  79 118 124  54   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0  31  79  79  47  68 105  52   0   0   0   0   0   0   0   0   0  15  47  63  63  57  34   9   0]
 [  0  15  63  95  63  44  57  28   0   0   0   0   0   0  11  39  54  68 105 127 127 130 100  34   0]
 [  0   0  15  63  95  70  44  38  15   0   0  15  47  46  39  79 108  89  68  63  63 105 130  57   0]
 [  0   0   0  15  69  90  84  95  47   0  15  63 110  93  42  39  54  36  10   0   8  80 134  63   0]
 [  0   0   0   0  26  68 105 126  79  22  44  86  79  46  15   0   0   0   0   1  22  85 126  75  14]
 [  0   0   0   0   5  54 108 106  86  59  49  44  15   0   0   0   0   0  11  41  61  57  87 106  45]
 [  0   0   0   0   0  39  94  87  62  67  59  22   0   0   0   0   0  15  54  94  90  36  64 129  71]
 [  0   0   0   0  15  43  71  89  66  60  77  43   5   0   0  15  43  70  86  70  43  15  60 129  77]
 [  0   0   0  15  63  79  47  74 117 106  91  55  11   0  15  62 102  93  54  15   0   0  44  92  52]
 [  0   0  15  63  95  63  15  51 135 146 101  43  21  41  66  87  74  39  28  48  63  63  82 102  83]
 [  0  15  63  95  63  15   0  28  73  76  47  32  63  98  85  50  29  45  79 112 127 126 129 133 130]
 [ 14  60  94  63  15   0   0   6  15  27  34  48  80  73  35   9  28  89 122 124 109  79  62  63  63]
 [ 44 105  76  15   0   0   0  15  48  82  81  48  32  16   0   0  14  44  74 104 103  70  43  15   0]
 [ 78 108  46   0   0  14  31  51  92 102  59  15   0   0   0   0   0   0  14  53  87 103  87  31   0]
 [ 92  66  15  13  43  74  79  55  49  40  12   0   0   0   0   0   0   0  15  49  65  61  45  15   0]
 [ 85  60  60  75 102 105  63  20   2   0   0   0   0   0   0   0   0   0  48 104  67  11   1   0   0]
 [ 74  82 120 109  75  45  15   0   0   0   0   0   0   0   0   0   0   0  48  96  63  47  47  15   0]
 [ 54  40  60  47  15   0   0   0   0   0   0   0   0   0   0   0   0   0  16  32  63 142 142  47   0]
 [ 69  15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  10  67 151 140  46   0]
 [ 81  32   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  36  88  83  47  32  48]
 [ 51  49  15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  43  87  43   1  35  97]
 [ 33  78  59  14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  19  38  19   0  17  48]
 [ 15  75 104  45   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   3   1   0   0   0]
 [  0  45 109  80  17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0  15  64  96  63  15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0  15  62  92  62  15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0  15  62  87  47   7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1]
 [  0   0   0   0  16  47  62  47  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  19]
 [  0   0   0   0   0   8  47  72  49  35  21   2   0   0   0   0   0   0   0   0   0   0   0   0  48]
 [  0   0   0   0   0   0  15  32  49  71  61  39  15   0   0   0   0   0   0   0   0   0   0  15  78]
 [  0   0   0   0   0   0   0   0  16  37  59  69  31  15  47  51  23   3   0   0   0   0  15  63  95]
 [  0   0   0   0   0   0   0   0   0   1  18  33  15  31  95 107  72  60  63  63  63  63  79  95  63]]
```

Заблюрено :)

Выбирая правильные фильтры можно по-разному преобразовывать картинку:

Детектировать грани разных направлений, сглаживать или добавлять резкости, менять цвета, яркость и контраст.

Нейросетку угат фильтры.

Horizontal Sobel-edges

Vertical Sobel-edges

Лирическое отступление №2. Что такое картинки?

Лирическое отступление №3. Что есть белка?

PCA лучше чем пиксели :)

10000

340

Восстановленная белка

В общем (dev, learning)

1: У нас есть входные данные в виде набора тензоров

2: Есть модель которая преобразует эти данные во внутреннее представление

3: Прогоняя данные через модель мы оцениваем что получилось

4: Меняем параметры модели чтобы получилось лучше чем было

5: Goto 1

В общем (prod, inference)

1: У нас есть входные данные в виде набора тензоров

2: Есть модель которая преобразует эти данные во внутреннее представление

3: Прогоняя данные через модель

Теперь мы все понимаем и можем…

…скачать tensorflow

Отличить белку от кота!

Эту задачу давно решили за нас :)

Tensorflow(js) + MobileNet

Грузим модель

```
const loadModel = async (): Promise<tf.LayersModel> => {
  const model = await tf.loadLayersModel(MODEL_URL)
  return model
```

Что там внутри?

```
block_16_depthwise_relu (R [[null,7,7,960]]        [null,7,7,960]        0        block_16_depthwise_BN[0
__________________________________________________________________________________________________
block_16_project (Conv2D)  [[null,7,7,960]]        [null,7,7,320]        307200   block_16_depthwise_relu
__________________________________________________________________________________________________
block_16_project_BN (Batch [[null,7,7,320]]        [null,7,7,320]        1280     block_16_project[0][0]
__________________________________________________________________________________________________
Conv_1 (Conv2D)            [[null,7,7,320]]        [null,7,7,1280]       409600   block_16_project_BN[0][
__________________________________________________________________________________________________
Conv_1_bn (BatchNormalizat [[null,7,7,1280]]       [null,7,7,1280]       5120     Conv_1[0][0]
__________________________________________________________________________________________________
out_relu (ReLU)            [[null,7,7,1280]]       [null,7,7,1280]       0        Conv_1_bn[0][0]
__________________________________________________________________________________________________
global_average_pooling2d ( [[null,7,7,1280]]       [null,1280]           0        out_relu[0][0]
__________________________________________________________________________________________________
predictions (Dense)        [[null,1280]]           [null,1000]           1281000  global_average_pooling2
==================================================================================================
Total params: 3538984
Trainable params: 3504872
Non-trainable params: 34112
```

input

output

мы так делали!

параметры

1000 классов

model.summary()

Препроцессинг

```
const PREPROCESS_DIVISOR = tf.scalar(255 / 2)
const WIDTH = 224
const HEIGHT = 224

// функция препроцессинга
const processInputImage = (input: tf.Tensor) => {
  const preprocessedInput = tf.div(
    tf.sub(input.asType('float32'), PREPROCESS_DIVISOR),
    PREPROCESS_DIVISOR
  )
  return preprocessedInput.reshape([-1, ...preprocessedInput.shape])
}

const predict = async (input: tf.Tensor, model: tf.LayersModel) =>
  model.predict(processInputImage(input))
```

обычно сетки кушают модели фиксированного размера

Если вы не знаете что ожидаем модель…
Читайте документацию :)

Вам нужно знать как процессился инпут при обучении

```
const getModelInfo = (model: LayersModel) => {
  model.summary()

  // inspect layer
  console.log(model.layers)

  console.log(layer[n].input.shape, layer[n].output.shape)
}
```

Предсказания

```
const getProbs = async (
  img: HTMLImageElement,
  model: tf.LayersModel
): Promise<Predictions> => {
  // превращаем картинку в тензор
  const tensor = tf.browser.fromPixels(img)
  // предсказываем
  const result = await predict(tensor, model)
  const predictedClasses = tf.tidy(() => {
    // получаем самые вероятные предсказания
    const { values, indices } = tf.topk(result as tf.Tensor)
    const valuesData = values.dataSync() as Float32Array
    const indexData = indices.dataSync()
    return valuesData.reduce(
      (acc, val, idx) => [
        ...acc,
        {
          prob: val,
          // и превращаем циферку соответствующую классу в его название
          cl: IMAGENET_CLASSES[indexData[idx]][1],
        },
      ],
      []
    )
  })

  return predictedClasses
}
```

один из 1000 классов

ыть :)

fox_squirrel with probability 43.5%

tiger_cat with probability 37.2%

Кажется кого-то не уволят с работы :)

Но есть вопрос…

Как вытащить слой?

```
const LAYER_NAME = 'block_12_expand'
const layer = model.getLayer(LAYER_NAME)

// или вот так
const layer100 = model.layers[100]
```

Слой это функция. Пруф:

```
const layer = model.layers[0]
const result = layer.apply(processInputImage(input))
```

А модель это граф :)

Используя apply с заглушкой можно строить новые модели

```
// находим нужный слой и его output
const layerOutput = layer.output
let layerIndex = model.layers.findIndex(l => l.name === LAYER_NAME)
const [_, ...outputShape] = (layerOutput as tf.SymbolicTensor).shape

// первая модель
const m1 = tf.model({
  inputs: model.inputs,
  outputs: layerOutput,
})

// вторая модель
const m2Input = tf.input({ shape: outputShape })

let nextTensor = newModelInput
const m2Layers = model.layers.slice(layerIndex + 1)
for (const l of m2Layers) {
  nextTensor = l.apply(nextTensor) as tf.SymbolicTensor
}
const m2 = tf.model({ inputs: newModelInput, outputs: nextTensor })
```

создавать новый граф

Heatmap куда «смотрела» наша модель

```
// функция для которой будем считать градиенты. Возвращает вероятность получить интересующий нас класс для второй модели.
const classProbability = (input: tf.Tensor) =>
  (m2.apply(input, { training: true }) as tf.Tensor).gather([CLASS_INDEX], 1)

// собственно градиент
const gradFn = tf.grad(classProbability)

// прогоняем первую модель
const m1Output = m1.apply(input)

// считаем как output интересующего нас слоя влияет на вероятность получить нужный класс
const gradValues = tf.mean(gradFn(m1Output as tf.Tensor), [0, 1, 2])

// применяем градиенты
const m2ScaledOutput = (m1Output as tf.Tensor).mul(gradValues)

// на основе градиентов строим тепловую карту
heatMap = getHeatMap(scaledConvOutputValues)

// ресайзим heat map
tf.image.resizeBilinear(heatMap as tf.Tensor<tf.Rank.R3>, [width, height])
```

ОРЕХ
ДЕТЕКТЕД!

Есть техники повеселее

Guided backpropagation

Deconvolution

Deep Dream

Axiomatic Attribution

Transfer learning

Поточнее и полегче

squirrel with probability 99.9%
cat with probability 0.1%
dog with probability 0.0%

cat with probability 99.2%
dog with probability 0.3%
squirrel with probability 0.2%

Альтернативы

TensorFlow VS PyTorch

На клиенте работает и то и то

TensorFlow

tensorflowjs

VS

PyTorch

onnx.js

(now ONNX RUNTIME WEB)

Можно конвертировать

TensorFlow

PyTorch

```
!tensorflowjs_converter \
    --input_format=keras_saved_model \
    /content/saved_model.pb \
    /content
```

```
torch.onnx.export(model,
  input,
  name=model_name,
  export_params=True,
  opset_version=10,
  do_constant_folding=True,
  input_names = ['input'],
  output_names = ['output'] )
```

Типичный flow

Учите модель на мощной машине

Конвертируете в tensorflow или ONNX

Загружаете на клиенте

TensorFlow

Производительность

Все зависит от модели. MobileNet из примера выше - 20Мб

Есть огромные модели, которые вы скорее всего не сможете запихнуть в браузер

Вы можете использовать Service Worker или другую магию загрузки

Грузить можно разные модели. Используйте *graphModel* вместо *layersModel*

Вот мы задеплоили… а дальше что?

JS-модели можно гонять и в облаке

Для ноды есть специальные версии (tensoflow-node и ONNX runtime node)

Круто если есть GPU или TPU

Что там за бекенды такие?

GPU / TPU – сила векторизации

TensorFlow VS PyTorch

Wasm или WebGL

Wasm или WebGL

Про WebGL не интересно, давайте про WebGPU

Получаем adapter

```
// getting device
  const adapter = await navigator.gpu?.requestAdapter()
  if (!adapter) return
  const device = await adapter.requestDevice()
```

Пишем шейдер

```
// creating shader
const module = device.createShaderModule({
  code: `

  struct Ball {
    radius: f32,
    position: vec2<f32>,
    velocity: vec2<f32>,
  }

  @group(0) @binding(1)
  var<storage, write> output: array<Ball>;

  @group(0) @binding(0)
  var<storage, read> input: array<Ball>;

  let TIME_STEP: f32 = 0.016;

  @stage(compute) @workgroup_size(64)
  fn main(
    @builtin(global_invocation_id) global_id : vec3<u32>,
    @builtin(local_invocation_id) local_id : vec3<u32>,
  ) {
    let num_balls = arrayLength(&output);
    if(global_id.x >= num_balls) {
      return;
    }

    output[global_id.x].position =
      input[global_id.x].position +
      input[global_id.x].velocity * TIME_STEP;
  }
  `,
})
```


https://surma.dev/things/webgpu/index.html

Подключаем layout и закидываем данные

```
// creating buffers
  const BUFFER_SIZE = 1000

  const output = device.createBuffer({
    size: BUFFER_SIZE,
    usage: GPUBufferUsage.STORAGE | GPUBufferUsage.COPY_SRC,
  })

  const input = device.createBuffer({
    size: BUFFER_SIZE,
    usage: GPUBufferUsage.STORAGE | GPUBufferUsage.COPY_DST,
  })

  const stagingBuffer = device.createBuffer({
    size: BUFFER_SIZE,
    usage: GPUBufferUsage.MAP_READ | GPUBufferUsage.COPY_DST,
  })

  // creating bind group layout
  const bindGroupLayout = device.createBindGroupLayout({
    entries: [
      {
        binding: 0,
        visibility: GPUShaderStage.COMPUTE,
        buffer: {
          type: 'read-only-storage',
        },
      },
      {
        binding: 1,
        visibility: GPUShaderStage.COMPUTE,
        buffer: {
          type: 'storage',
        },
      },
    ],
  })
```


https://surma.dev/things/webgpu/index.html

Запускаем и может заработает :)

Бу!

```
// creating bind group
  const bindGroup = device.createBindGroup({
    layout: bindGroupLayout,
    entries: [
      {
        binding: 0,
        resource: {
          buffer: input,
        },
      },
      {
        binding: 1,
        resource: {
          buffer: output,
        },
      },
    ],
  })

  // creating pipeline
  const pipeline = device.createComputePipeline({
    layout: device.createPipelineLayout({
      bindGroupLayouts: [bindGroupLayout],
    }),
    compute: {
      module,
      entryPoint: 'main',
    },
  })
```

```
const commandEncoder = device.createCommandEncoder()
    const passEncoder = commandEncoder.beginComputePass()
    passEncoder.setBindGroup(0, bindGroup)
    passEncoder.setPipeline(pipeline)
    passEncoder.dispatch(Math.ceil(BUFFER_SIZE / 64))
    passEncoder.end()
```

```
commandEncoder.copyBufferToBuffer(
    output,
    0, // Source offset
    stagingBuffer,
    0, // Destination offset
    BUFFER_SIZE
  )
  const commands = commandEncoder.finish()
  device.queue.submit([commands])

  await stagingBuffer.mapAsync(
    GPUMapMode.READ,
    0, // Offset
    BUFFER_SIZE // Length
  )
  const copyArrayBuffer = stagingBuffer.getMappedRange(0, BUFFER_SIZE)
  const data = copyArrayBuffer.slice()
  const newBalls = new Float32Array(data)
  stagingBuffer.unmap()
```


https://surma.dev/things/webgpu/index.html

Бонус: Браузерные апишки

Возможно, сети вам не нужны

Barcode detection

Speech recognition api (Web Speech)

hellsquirrel.dev

Спасибо!

https://speakerdeck.com/hellsquirrel

@polina_gurtovaya

@pgurtovaya

hellsquirrel@evl.ms