ELBRUS / BOOT CAMP

Глубокое обучение • Deep Learning

Структура фазы

Эльбрус буткемп

1️⃣ Неделя 🤔

- Введение, 🔥 PyTorch, полносвязные сети, сверточные сети, архитектуры классификации изображений
- Transfer learning, finetuning
- **Проект**: streamlit-приложение для классификации картинок

2️⃣ Неделя 😢

- Локализация, детекция и сегментация объектов
- **Проект**: streamlit-сервис обнаружения объектов на изображениях

Структура фазы

Эльбрус буткемп

3️⃣ Неделя 🥲

- Нейросетевой подход к обработке текста: рекуррентные сети, трансформеры, BERT и GPT
- **Проект**: классификация пользовательских отзывов, генерация текста

4️⃣ Неделя 🤯

- Классический и нейросетевой подход в рекомендательных системах (вспомните `pandas` и `matplotlib`)
- **Проект**: семантический поиск

Сегодня

Эльбрус буткемп

- что такое глубокое обучение?
- интуиция нейронных сетей
- базовая архитектура
- функции активации
- функции потерь

Глубокое обучение · Deep learning

Эльбрус буткемп

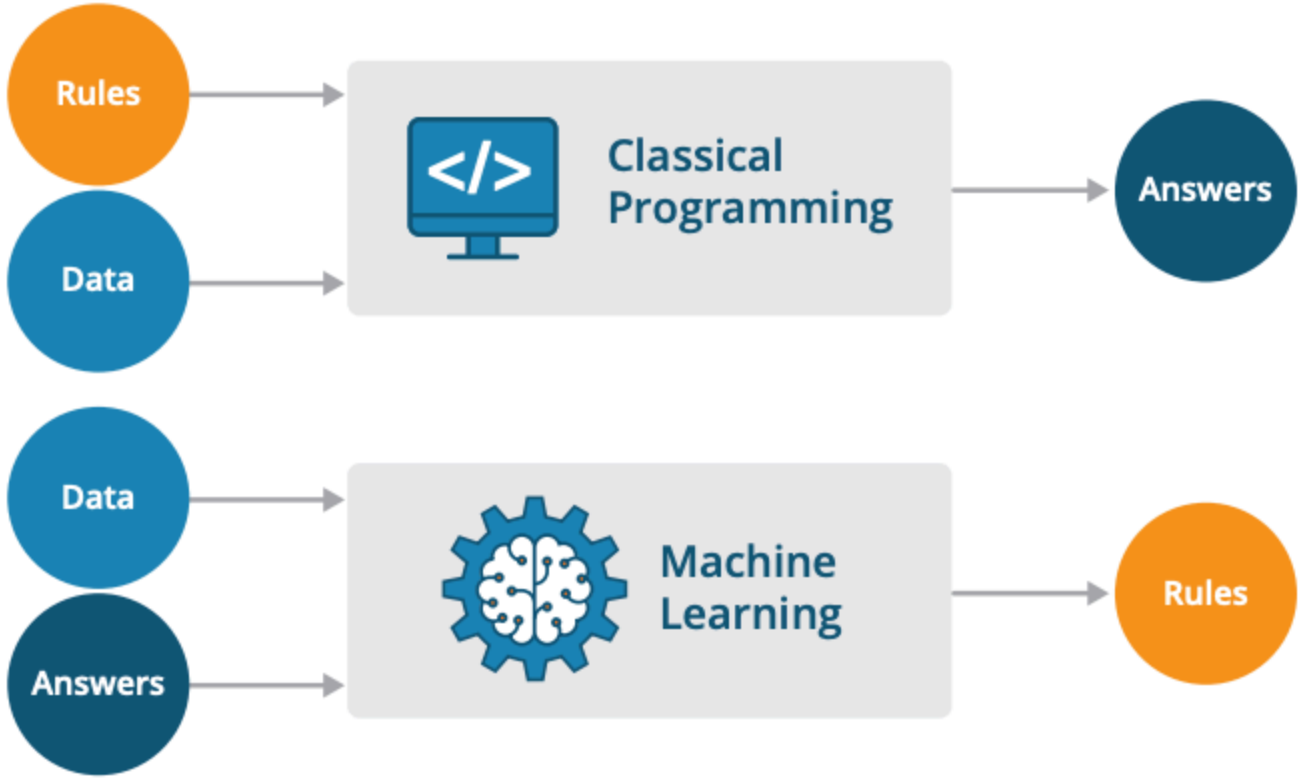


Figure 1. Classical programming vs machine learning

📝Machine Learning is Forcing Software Development to Evolve

Глубокое обучение • Deep learning

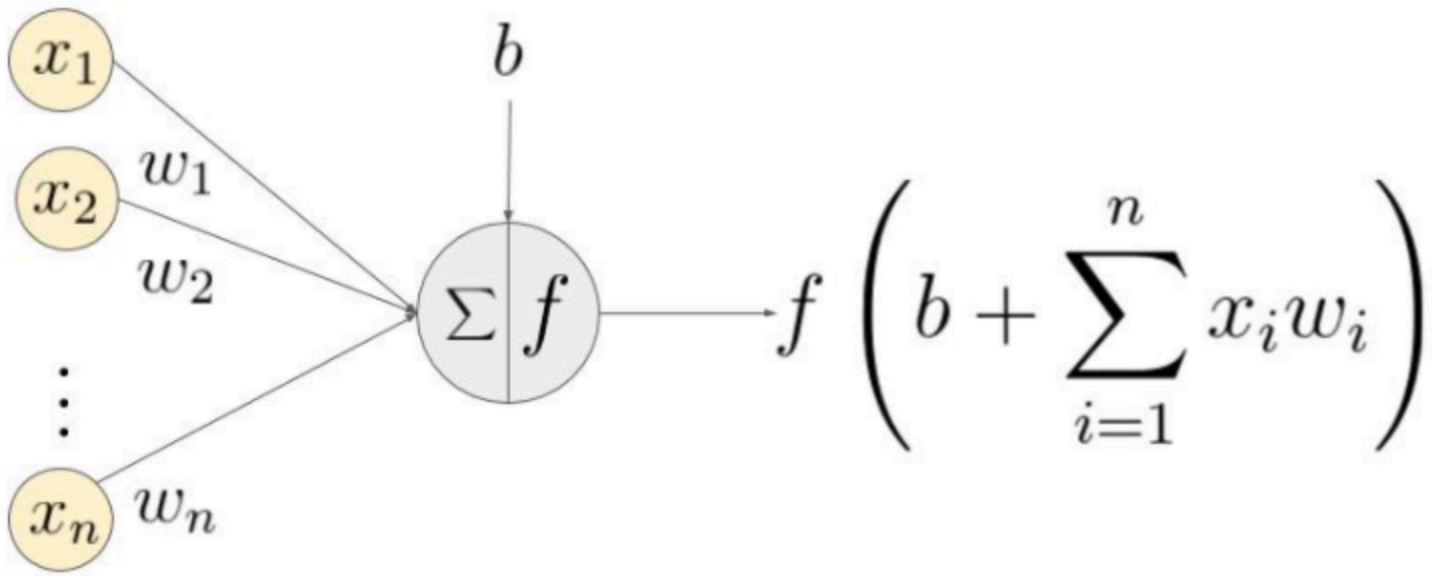
Эльбрус буткемп

Теперь параметров намного больше

- Архитектура сети • Network Architecture
  - число нейронов, функции активации, регуляризация ...
- Оптимизатор • Optimizer
- Функция потерь • Loss Function
- Инициализация весов • Initial weights

Структура нейрона

Эльбрус буткемп



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Структура нейрона

Эльбрус буткемп

$$y = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

b - bias - свободный член

x_i - элемент выборки

w_i - обучаемые параметры

f – *некоторая функция*

> Если в качестве функции f выбрать сигмоиду, во что превратится один нейрон?

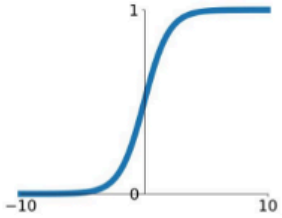
Функции активации

Эльбрус буткемп

Activation Functions

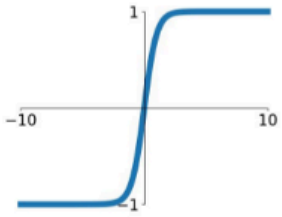
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



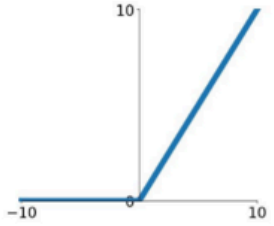
tanh

$$\tanh(x)$$



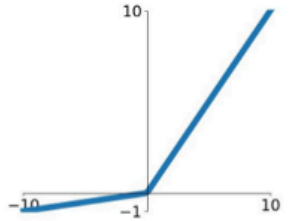
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

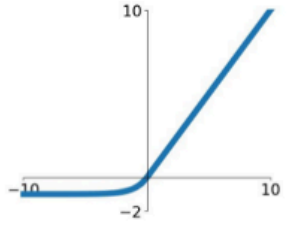


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

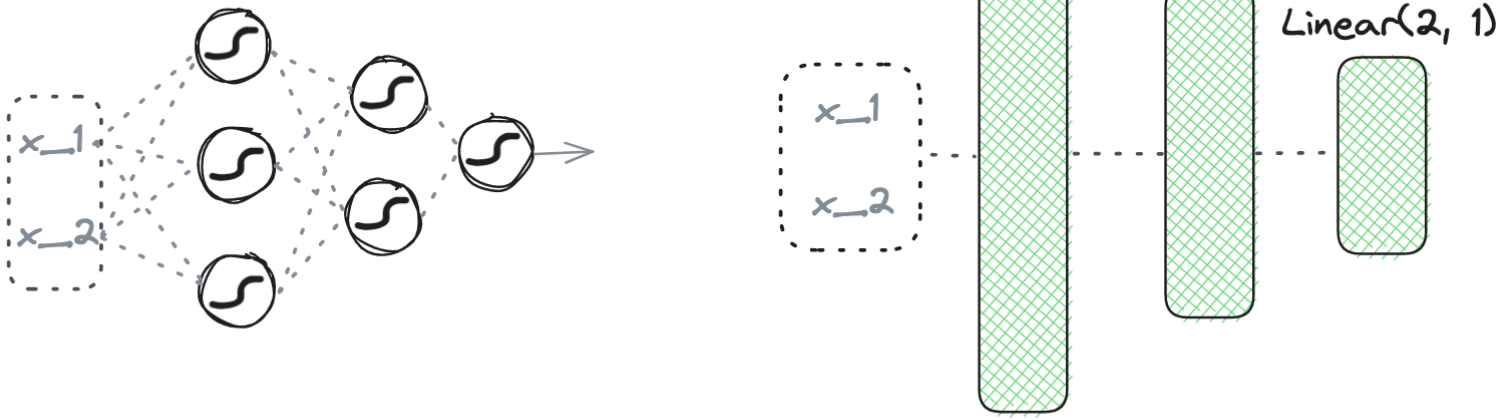
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



🔥 `nn.Sigmoid`, `nn.Tanh`, `nn.ReLU`

Полносвязный слой • Fully Connected • Linear • Dense



🔥 `nn.Linear(in_features, out_features)`

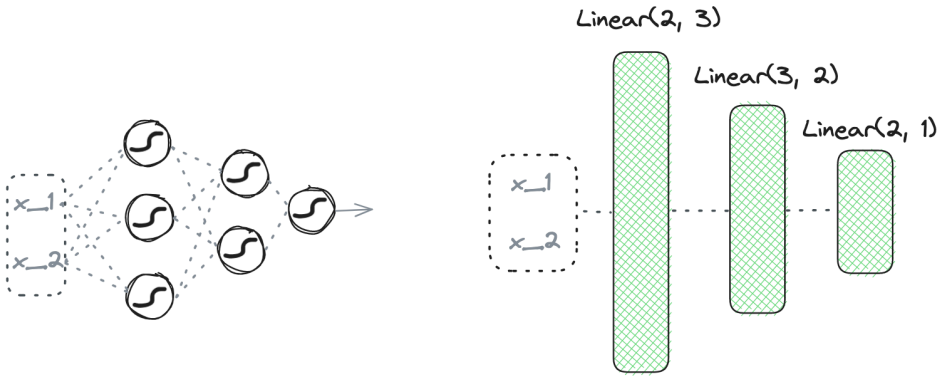
📝Fully Connected vs Convolutional Neural Networks 📊NN-SVG

Полносвязный слой • Fully Connected

Эльбрус буткемп

Если предположить, что активация везде сигмоида, то такая модель задавалась бы очень просто:

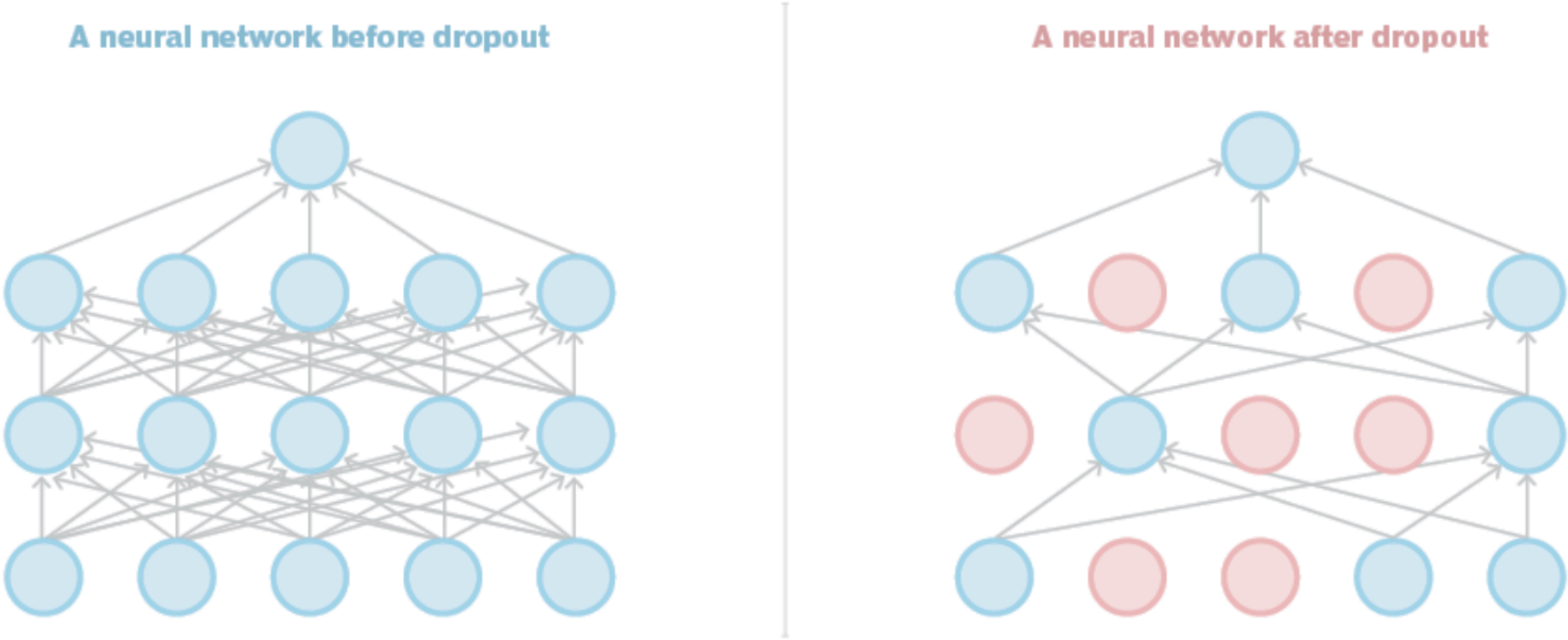
```
model = nn.Sequential(
  nn.Linear(2, 3),
  nn.Sigmoid(),
  nn.Linear(3, 2),
  nn.Sigmoid(),
  nn.Linear(2, 1),
  nn.Sigmoid()
)
```



📝Fully Connected vs Convolutional Neural Networks 📊NN-SVG

Дропаут · Dropout

Эльбрус буткемп



🔥 `nn.Dropout(p)`

📝What is dropout in deep neural networks? 📝5.4. Тонкости обучения

Дропаут · Dropout

Эта операция тоже добавляется просто:

```
model = nn.Sequential(
  nn.Linear(2, 3),
  nn.Dropout(),
  nn.Sigmoid(),
  nn.Linear(3, 2),
  nn.Dropout(),
  nn.Sigmoid(),
  nn.Linear(2, 1),
  nn.Sigmoid()
)
```

А положение относительно активации неважно.

Функция потерь • Loss

Эльбрус буткемп

	Задача	Функция потерь	N выходных нейронов
1	Бинарная классификация	Бинарная кросс-энтропия `BCELoss()`	1

Схема

Функция потерь • Loss

Эльбрус буткемп

	Задача	Функция потерь	N выходных нейронов
1	Бинарная классификация	Бинарная кросс-энтропия `BCELoss()`	1
2	Бинарная классификация	Бинарная кросс-энтропия **без активации последнего нейрона** -> `BCEWithLogitsLoss()`	1

📊 Схема

Бинарная классификация

Эльбрус буткемп

Функция потерь - **бинарная кросс-энтропия**

$\boldsymbol{y}$ – истинные классы	$\hat{\boldsymbol{p}}$ – предсказанные классы	N – объем выборки

$$BCELoss = -\frac{1}{N}\sum_{i}^{N}[y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)]$$

Функция активации - **сигмоида**

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$

🔥 `nn.BCELoss(predictions, targets)`

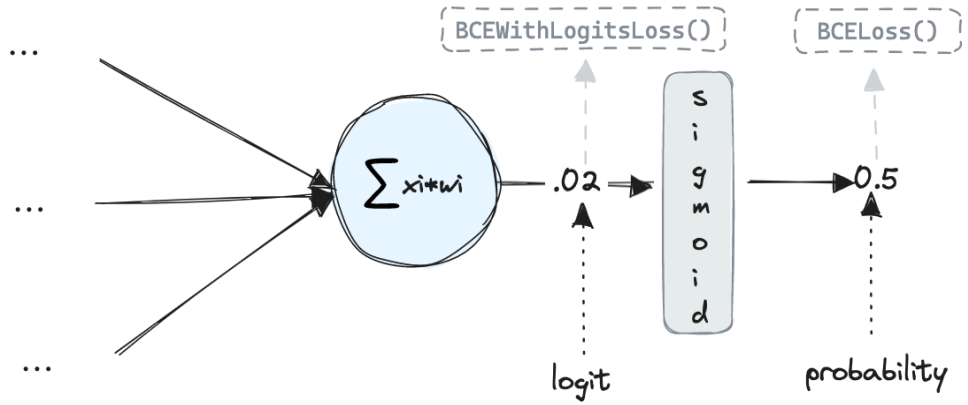
Функция потерь • Loss

Эльбрус буткемп

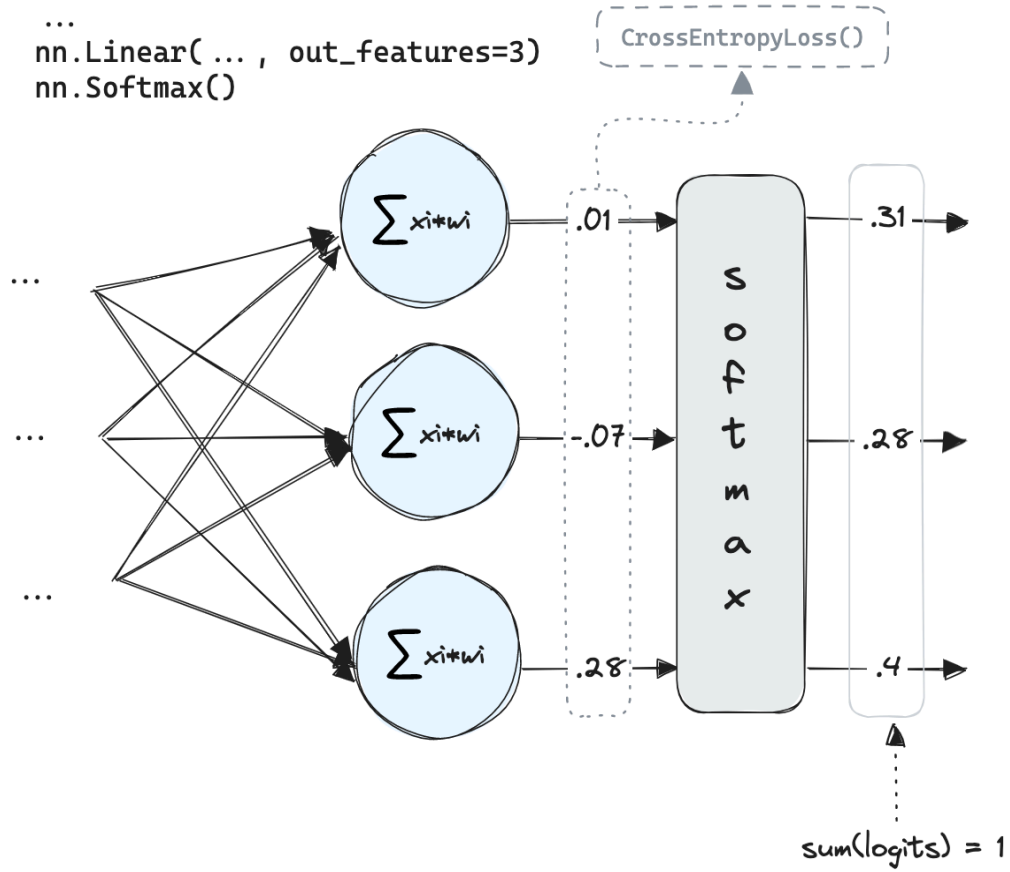
	Задача	Функция потерь	N выходных нейронов
1	Бинарная классификация	Бинарная кросс-энтропия `BCELoss()`	1
2	Бинарная классификация	Бинарная кросс-энтропия **без активации последнего нейрона** -> `BCEWithLogitsLoss()`	1
3	Многоклассовая классификация	Категориальная кросс-энтропия `CrossEntropyLoss()`	N классов

📊 Схема

```
...
...
nn.Linear( ... , out_features=1)
nn.Sigmoid()
```




```
...
...
nn.Linear( ... , out_features=3)
nn.Softmax()
```



Мультиклассовая классификация

Эльбрус буткемп

Функция потерь - **категориальная кросс-энтропия**

$$CELoss = -\sum_{i}^{C} y_i \log(\hat{p}_i)$$

Функция для получения вероятностей - **софтмакс** ℹ️

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

❗ The input is expected to contain the unnormalized logits for each class (which do not need to be positive or sum to 1, in general) soucre.

Иначе говоря, чтобы использовать этот лосс, нам не нужно использовать софтмакс. Мы можем его применить, чтобы посмотрть распределение вероятностей **для себя**

🔥 `nn.CrossEntropyLoss(predictions, targets)`

Мультиклассовая классификация

Эльбрус буткемп

$$CELoss = -\sum_{i}^{C} y_i \log(\hat{p}_i)$$

Три класса: 0, 1, 2

Объект	`y`	`is0?`	`is1?`	`is2?`
obj_1	2	0	0	1
obj_2	2	0	0	1
obj_3	0	1	0	0
obj_4	1	0	1	0

У сети 3 выхода: для оценка шансов для каждого класса

Функция потерь • Loss

Эльбрус буткемп

	Задача	Функция потерь	N выходных нейронов
1	Бинарная классификация	Бинарная кросс-энтропия `BCELoss()`	1
2	Бинарная классификация	Бинарная кросс-энтропия **без активации последнего нейрона** -> `BCEWithLogitsLoss()`	1
3	Многоклассовая классификация	Категориальная кросс-энтропия `CrossEntropyLoss()`	N классов
4	Регрессия	Любая регрессионная `MSELoss(), L1Loss()`, etc	1

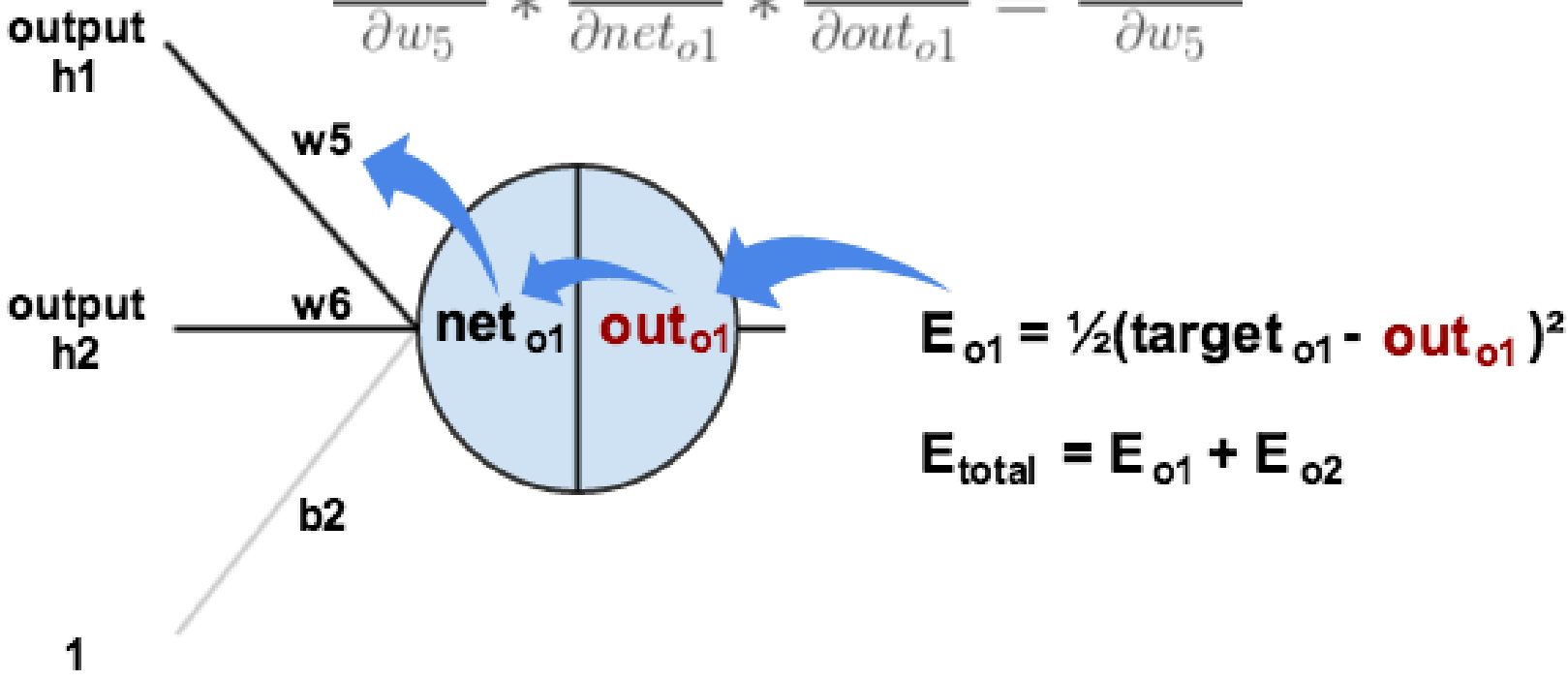
❗ Функции потерь, активации и число нейронов выходного слоя 🔥 pytorch losses

📊 Схема

Обратное распространение ошибки

Эльбрус буткемп

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



Инициализация весов

Эльбрус буткемп

1. Xavier initialization (для гиперболического тангенса)

$$X \sim \mathcal{N}(0, \mathrm{Var}(w_i)) \quad \forall i, \mathrm{Var}(w_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

2. Normalized Xavier

$$X \sim \mathcal{U}(0, \mathrm{Var}(w_i)) \quad \forall i, \mathrm{Var}(w_i) \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right].$$

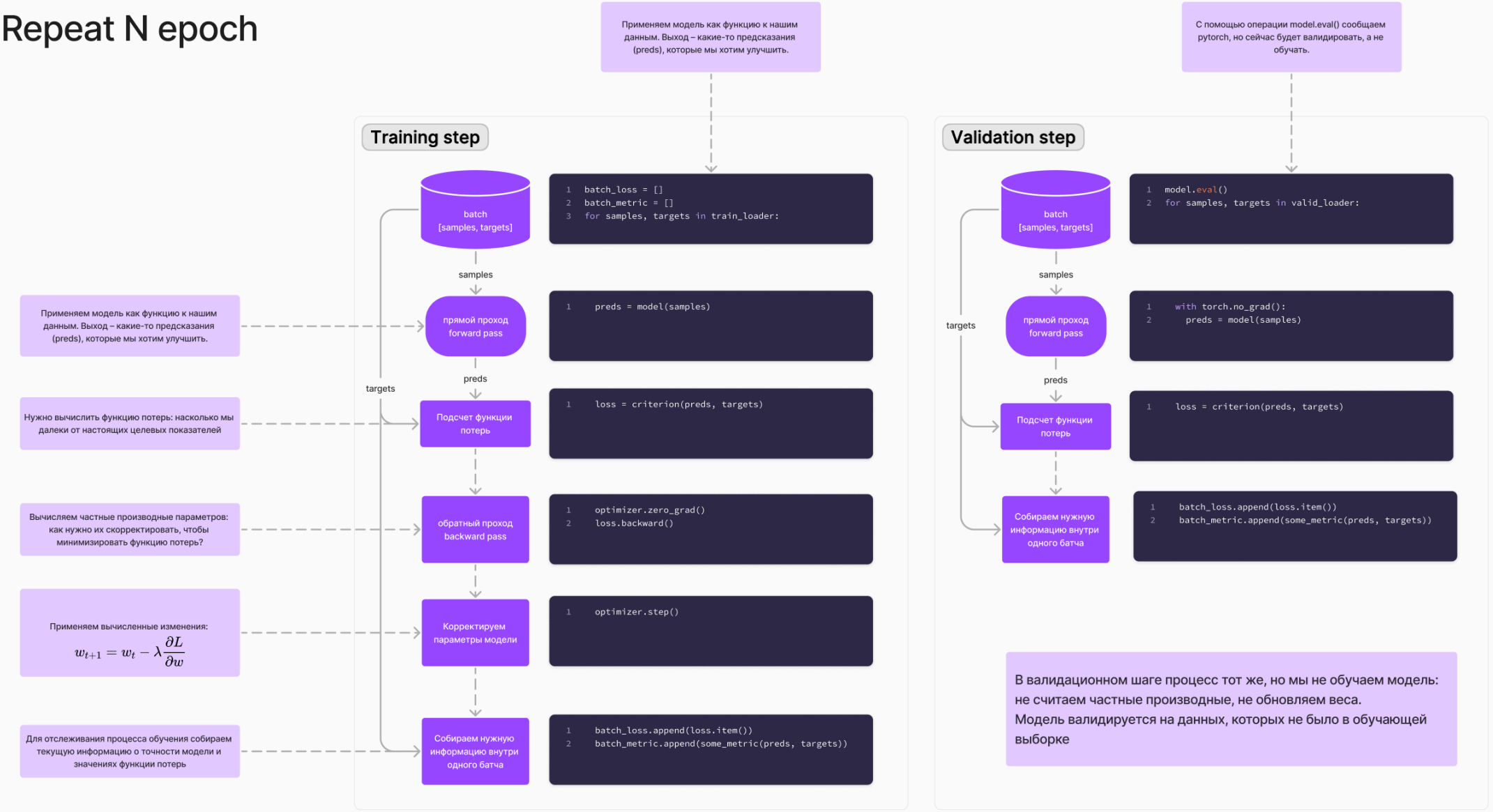
3. He (or Kaiming) initialization (ReLU)

$$\mathcal{N}(0, \frac{2}{n_{\text{in}}})$$

> Чаще всего инициализация производится **автоматически**!

Training loop

Repeat N epoch



→ Figma link

Шаги обучающего цикла • 1 шаг – обучение

Эльбрус буткемп

```
for epoch in range(n_epochs):
  # цикл по данным в рамках одной эпохи
  model.train() # переходим в режим обучения
  for data, target in train_loader:

    preds = model(data)             # считаем выходы модели – предсказания
    loss = criterion(preds, target) # вычисляем значение функции потерь

    optimizer.zero_grad()           # обнуляем градиенты предыдущего шага
    loss.backward()                 # вычисляем градиенты текущего шага
    optimizer.step()                # изменяем значения параметров

    batch_loss.append(loss.item())             # добавляем значение лосса в список
    batch_metric.append(some_metric(preds, target)) # добавляем значение метрики в список

  epoch_train_loss.append(np.mean(batch_loss))
  epoch_train_metric.append(np.mean(batch_metric))
```

Шаги обучающего цикла • 2 шаг – валидация

Эльбрус буткемп

```
batch_loss   = []       # обнуляем список значений функции потерь
batch_metric = []       # обнуляем список значений метрики
# цикл по валидационной части данных
model.eval() # переходим в режим валидации
for data, target in valid_loader:
  with torch.no_grad():             # градиенты модели не трогаем
    preds = model(data)             # считаем выходы модели
  loss = criterion(preds, target)   # вычисляем значение функции потерь
  batch_loss.append(loss.item())
  batch_metric.append(some_metric(preds, target))

epoch_valid_loss.append(np.mean(batch_loss))
epoch_valid_metric.append(np.mean(batch_metric))
```

Итоги

Эльбрус буткемп

- нейронные сети - гибкий инструмент
- много эвристик и гиперпараметров, архитектура определяется задачей
- полносвязный слой - много параметров
- дропаут - инструмент регуляризации
- функция потерь зависит от задачи, как и всегда