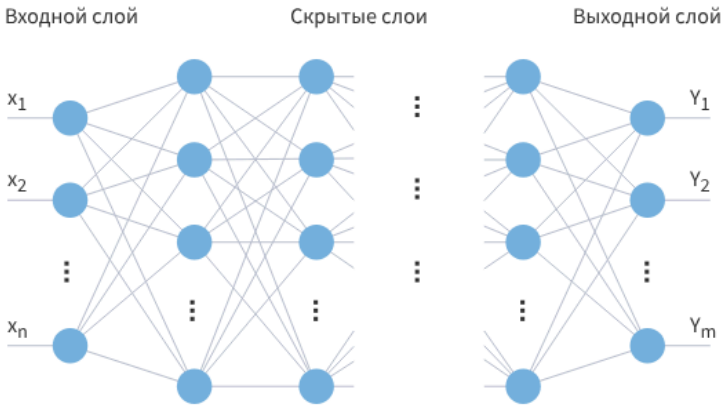
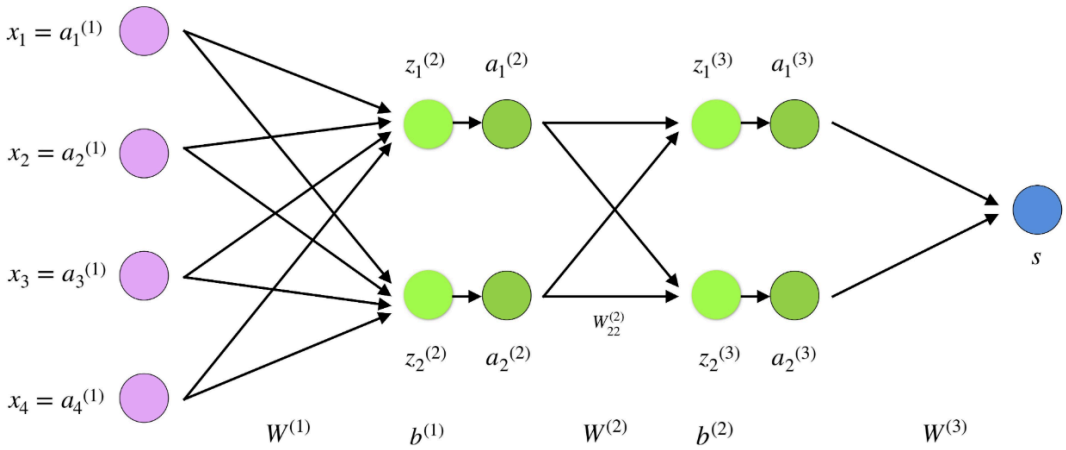
3.7 Нейронные сети

- Сверточные сети (конволюционные) нейронные сети (CNN) - компьютерное зрение, классификация изображений
- Рекуррентные нейронные сети (RNN) -распознавание рукописного текста, обработка естественного языка
- Генеративные состязательные сети (GAN) - создание художественных, музыкальных произведений
- Многослойный перецептрон - простейший тип нейронных сетей



Алгоритм обратного распространения ошибки



Популярные framework

- TensorFlow
- PyTorch

ONNX - Open Neural Network Exchange

Пример

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input,
                                                    ResNet50, decode_predictions
import matplotlib.pyplot as plt
import numpy as np

img_path = "cat.png"
img = image.load_img(img_path, target_size=(224, 224))

img_array = image.img_to_array(img)
img_batch = np.expand_dims(img_array, axis=0)

img_processed = preprocess_input(img_batch)

model = ResNet50()
prediction = model.predict(img_processed)

print(decode_predictions(prediction, top=5)[0])
# [('n02123159', 'tiger_cat', np.float32(0.5864534)),
# ('n02123045', 'tabby', np.float32(0.39245087)),
# ('n02124075', 'Egyptian_cat', np.float32(0.01629017)),
# ('n02127052', 'lynx', np.float32(0.0018590131)),
# ('n02129604', 'tiger', np.float32(0.0013254558))]
```

Перенос обучения

План решения задачи

1. Организуем данные
2. Построим пайплан подготовки
3. Аугментация данных, обогащение
4. Определение модели. Заморозка коэффициентов. Алгоритм оптимизатора, метрика оценки
5. Обучение модели → итераций → метрика не станет приемлемой
6. Сохранение модели

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, ResNet50
from tensorflow.keras.layers import (
    Input, Flatten, Dense, Dropout, GlobalAveragePooling2D, Lambda
)
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import numpy as np
import math

TRAIN_DATA_DIR = "data/train_data"
VALIDATION_DATA_DIR = "data/val_data"

TRAIN_SAMPLES = 500
VALIDATION_SAMPLES = 500
NUM_CLASSES = 2

IMG_WIDTH, IMG_HEIGHT = 224, 224
```

```
BATCH_SIZE = 64

train_datagen = image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

val_datagen = image.ImageDataGenerator(
    rescale=1./255,
)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DATA_DIR,
    target_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=12345,
    class_mode="categorical",
)

val_generator = val_datagen.flow_from_directory(
    VALIDATION_DATA_DIR,
    target_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=BATCH_SIZE,
    shuffle=False,
    class_mode="categorical",
)


def create_resnet_model():
    base_model = ResNet50(
        weights='imagenet',
        include_top=False,
        input_shape=(IMG_WIDTH, IMG_HEIGHT, 3),
    )
    base_model.trainable = False

    input_tensor = Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3))

    x = Lambda(preprocess_input, name='preprocessing')(input_tensor)
```

```
    x = base_model(x, training=False)

    x = GlobalAveragePooling2D()(x)
    x = Dense(64, activation="relu")(x)
    x = Dropout(0.5)(x)
    predictions = Dense(NUM_CLASSES, activation="softmax")(x)

    return Model(inputs=input_tensor, outputs=predictions)


model = create_resnet_model()

model.compile(
    loss="categorical_crossentropy",
    optimizer=Adam(learning_rate=0.001),
    metrics=["accuracy"]
)

num_steps = math.ceil(float(TRAIN_SAMPLES) / BATCH_SIZE)
val_steps = math.ceil(float(VALIDATION_SAMPLES) / BATCH_SIZE)

model.fit(
    train_generator,
    steps_per_epoch=num_steps,
    epochs=10,
    validation_data=val_generator,
    validation_steps=val_steps,
)

print(val_generator.class_indices)
# {'cat': 0, 'dog': 1}
model.save('trained_model.h5')
```