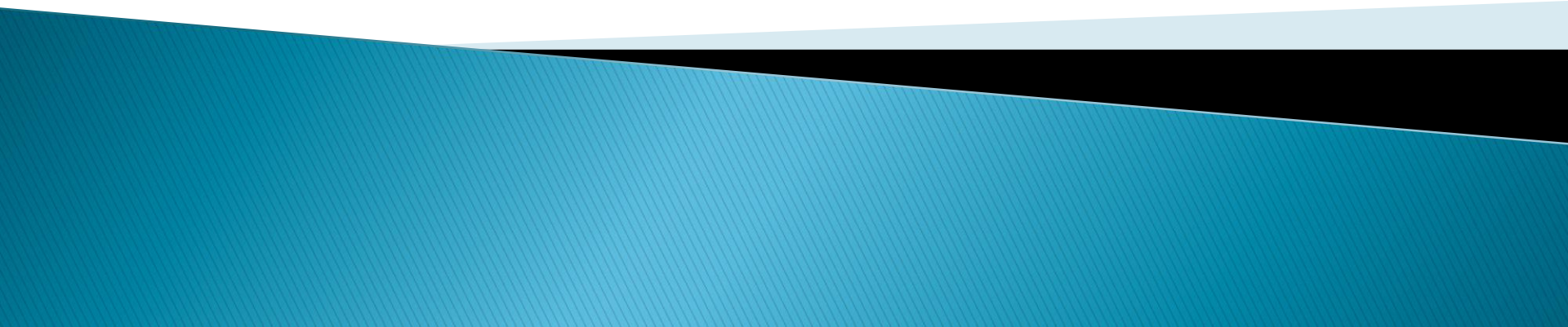
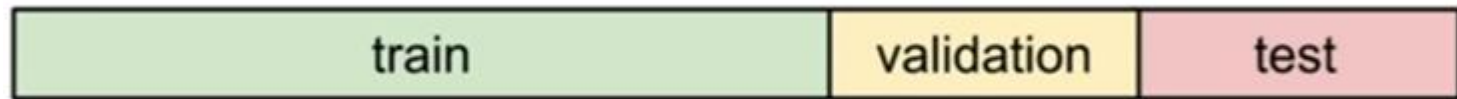
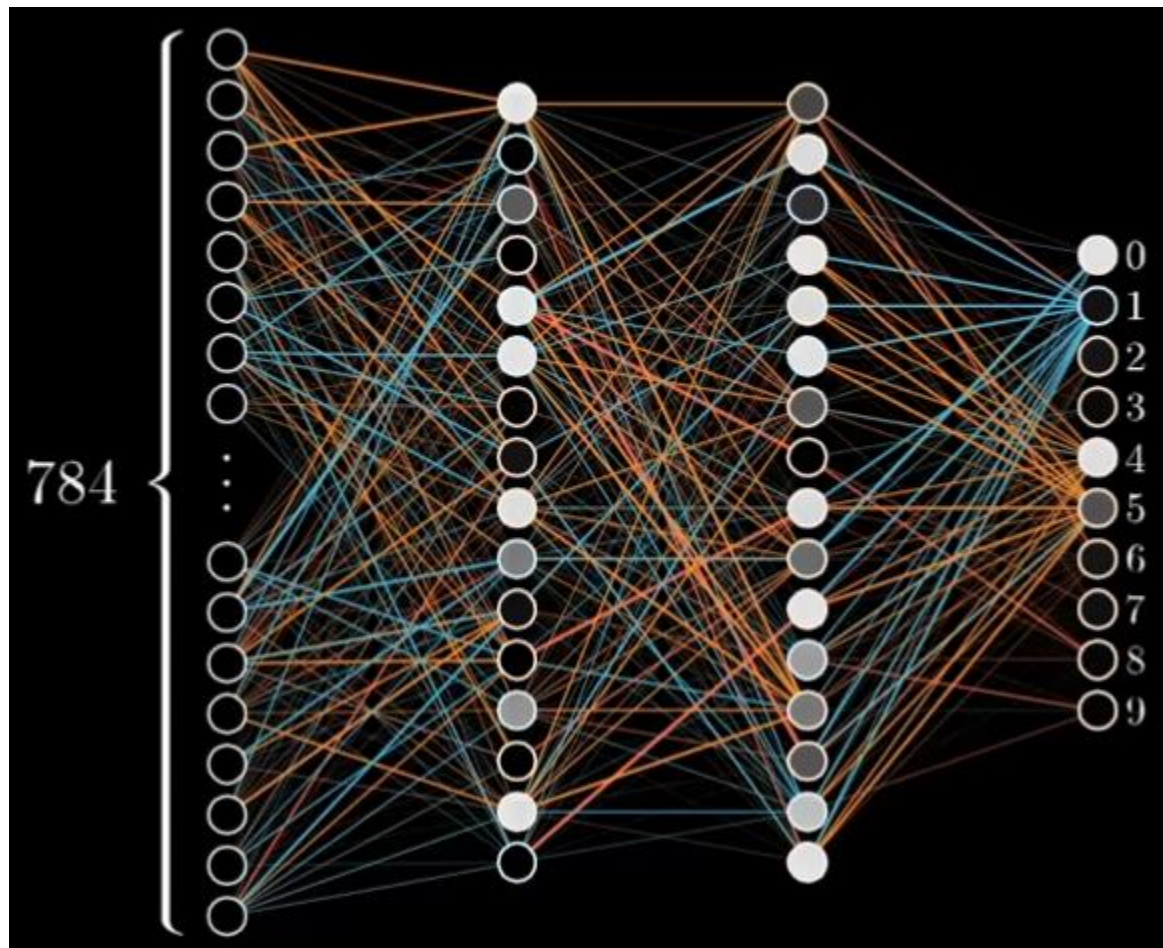


Навчання нейронних мереж



$(X_i, Y_i), i = 1, \dots, N,$



X_i  Y_i

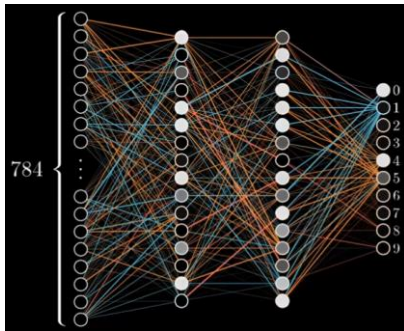
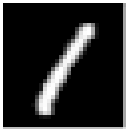
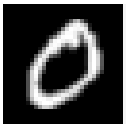
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

Функція помилки (Loss Function)

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Loss function for classification

X



$$p(c = 0|x_1) \quad p(c = 1|x_1) \quad \dots \quad p(c = 9|x_1)$$



...



$$p(c = 0|x_2) \quad p(c = 1|x_2) \quad \dots \quad p(c = 9|x_2)$$



...



.....

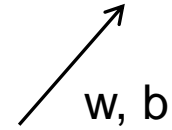


...



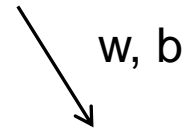
Maximum likelihood

$$p(data) = \prod_i p(c = y_i | x_i)$$



Negative log-likelihood

$$-\ln p(data) = -\sum_i \ln p(c = y_i | x_i)$$



Cross-Entropy Loss Function

$$L = -\frac{1}{N} \sum_i \ln p(c = y_i | x_i)$$

Loss function for classification

Cross-Entropy Loss Function

$$L = -\frac{1}{N} \sum_i \ln p(c = y_i | x_i)$$

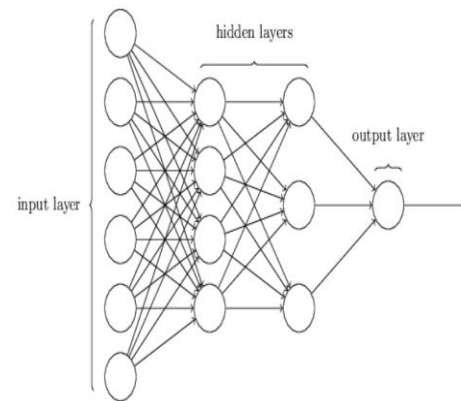
Activation
function  Softmax

Loss function for binary classification

Binary cross-entropy loss

$$L = -\frac{1}{N} \sum_{i=1}^N y_i \ln p(y_i | x_i) + (1 - y_i) \ln(1 - p(y_i | x_i))$$

Activation function \longrightarrow Sigmoid

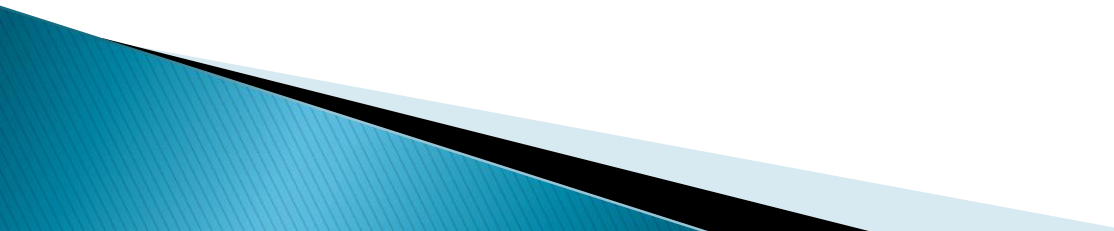


Loss function for regression

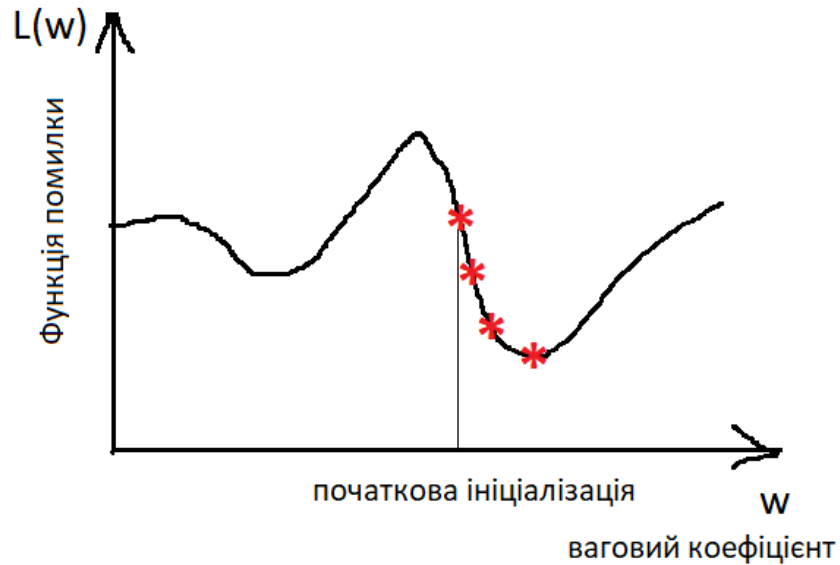
Mean Absolute Error (MAE), L₁ Loss

$$L = \frac{1}{N} \sum_{i=1}^N |f_i - y_i|$$

Mean Square Error (MSE), L₂ Loss

$$L = \frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$


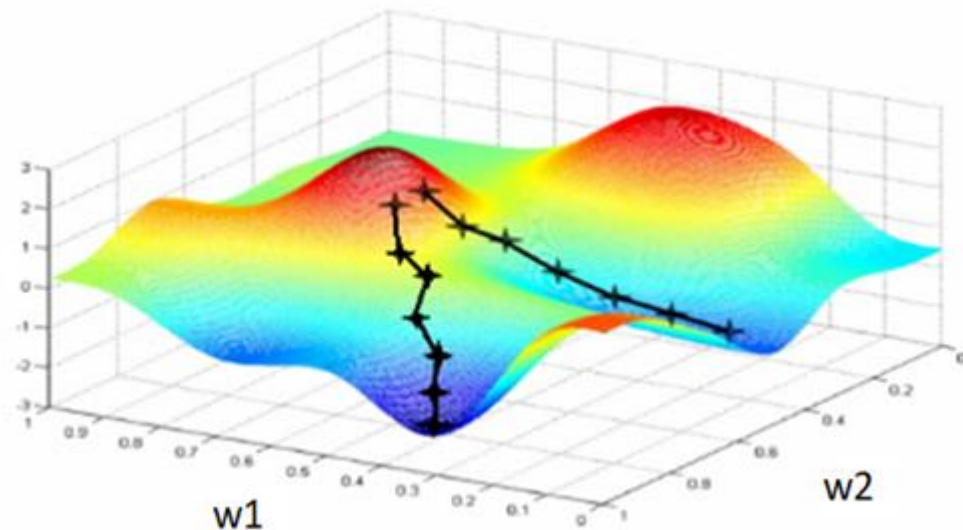
Градiєнтний спуск (Gradient descent)



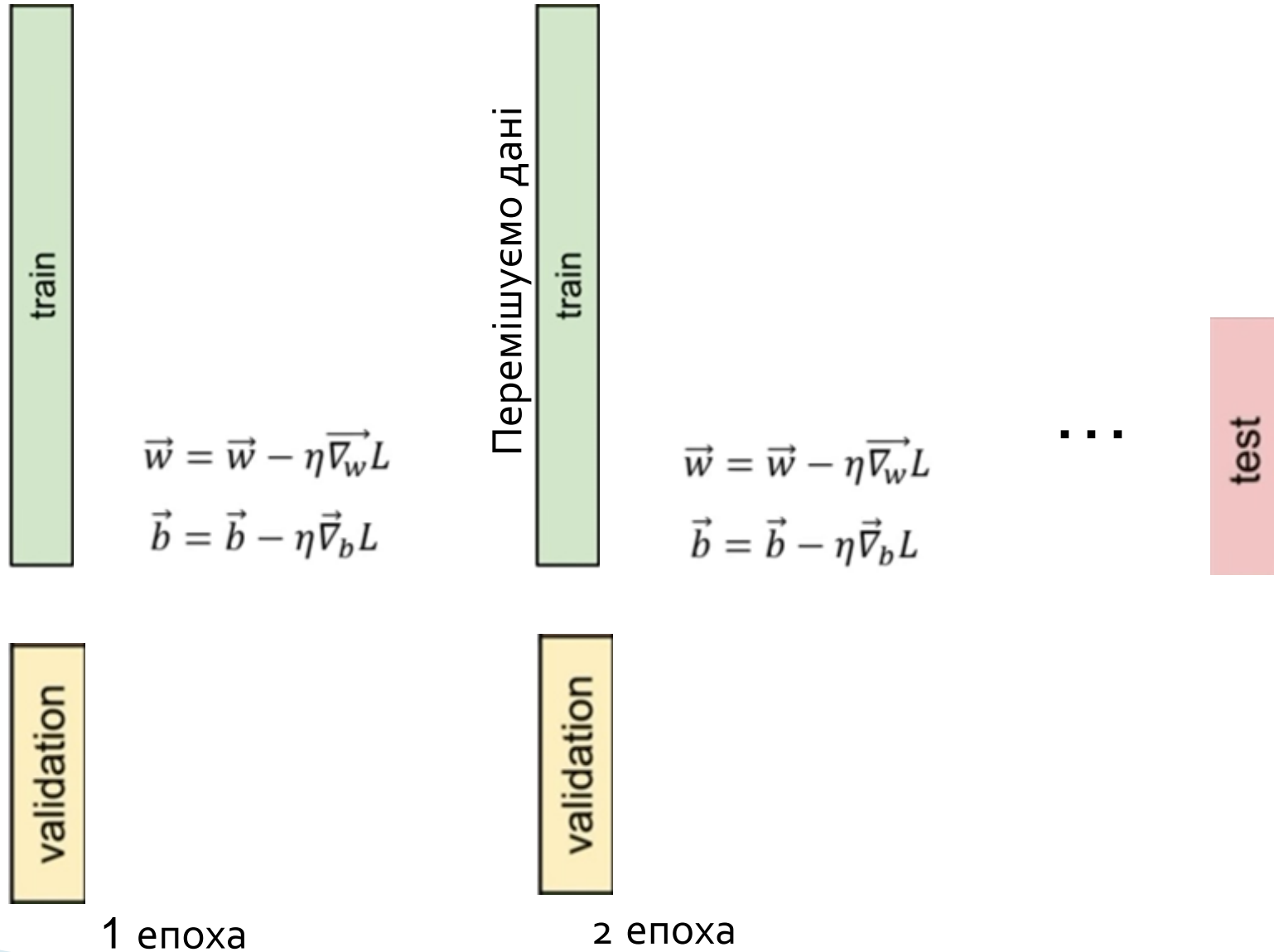
$$\vec{w} = \vec{w} - \eta \vec{\nabla}_w L$$

$$\vec{b} = \vec{b} - \eta \vec{\nabla}_b L$$

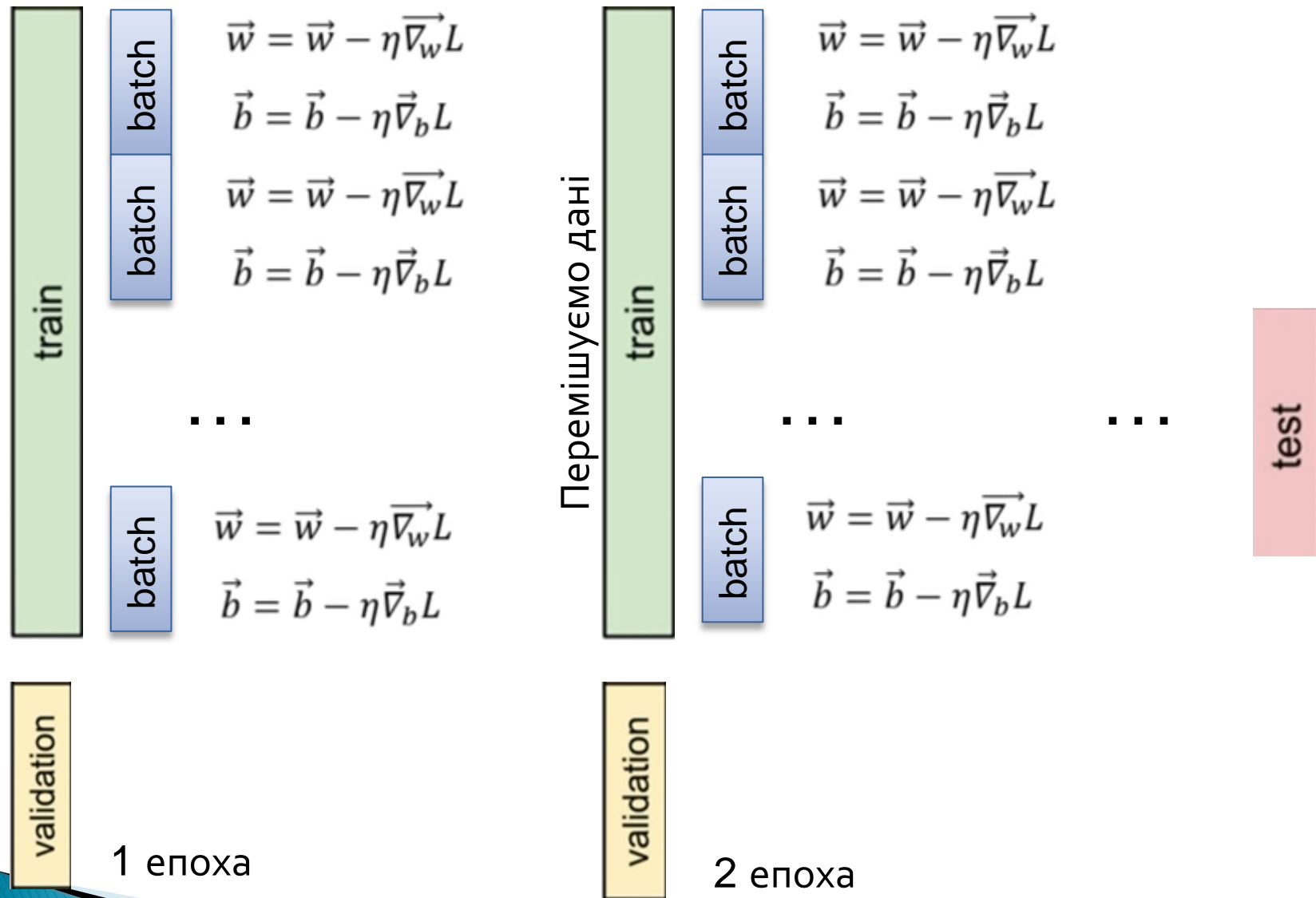
$L(w1, w2)$

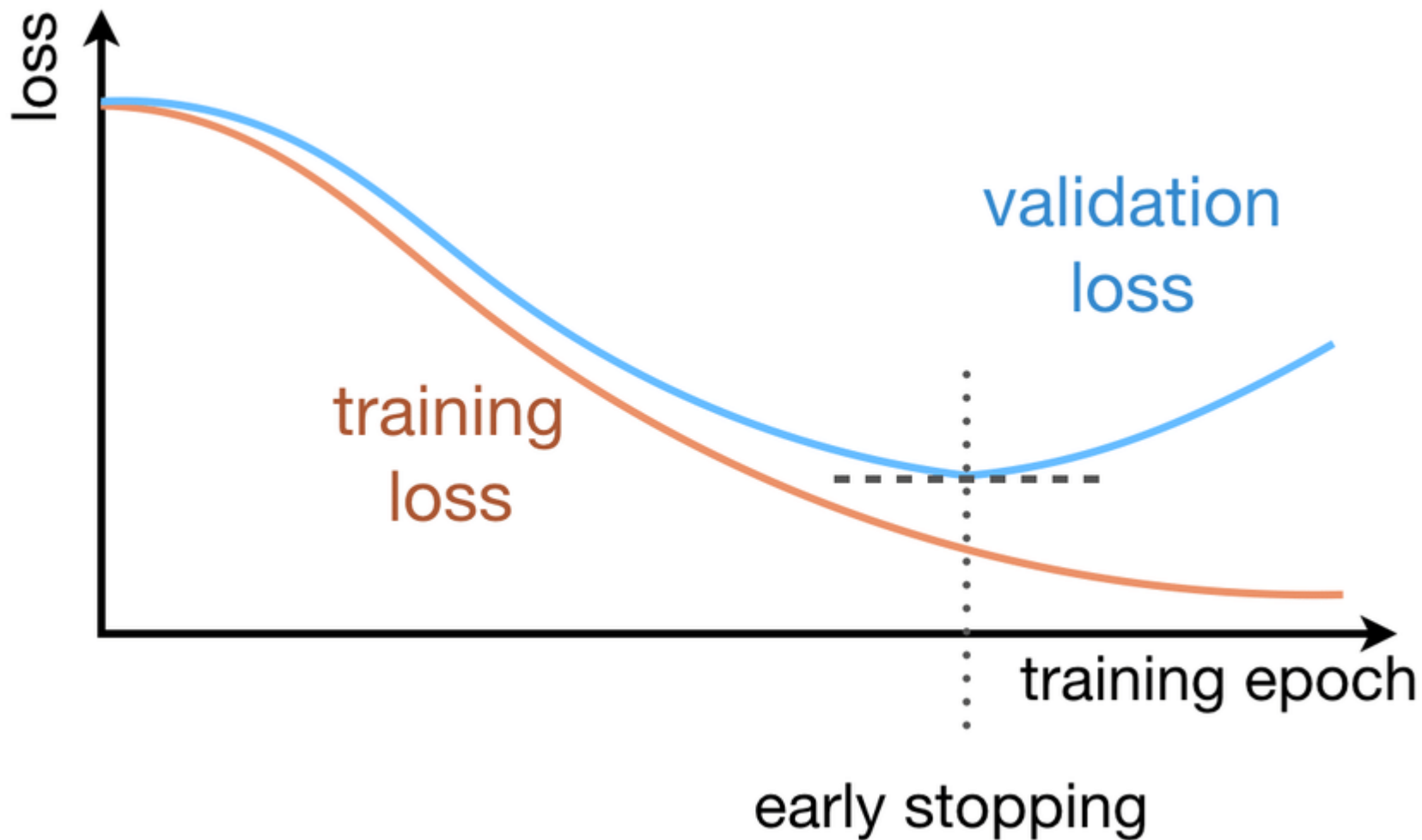


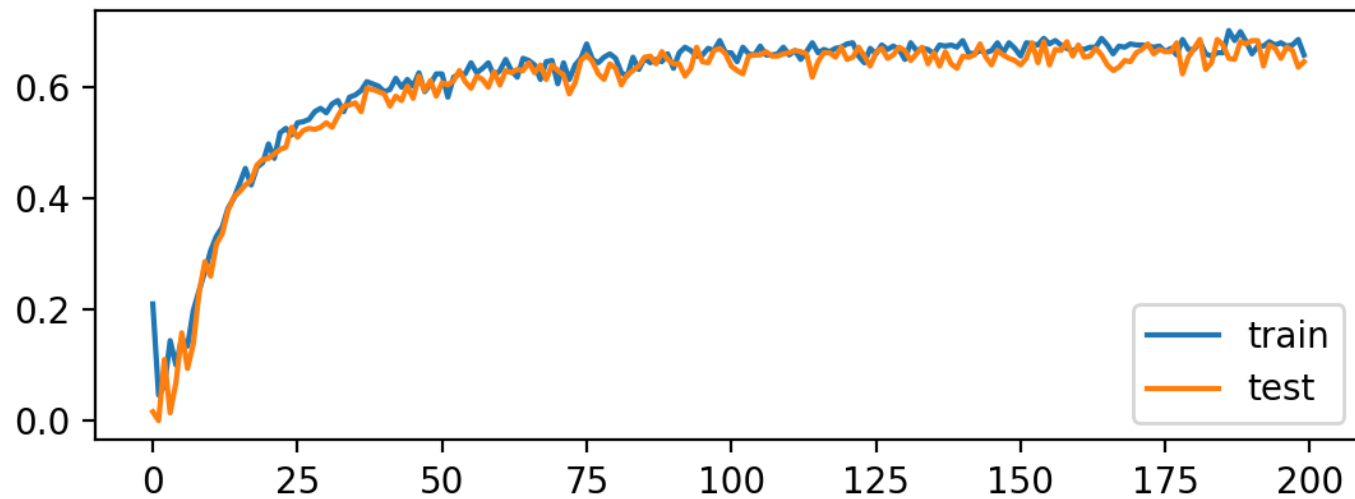
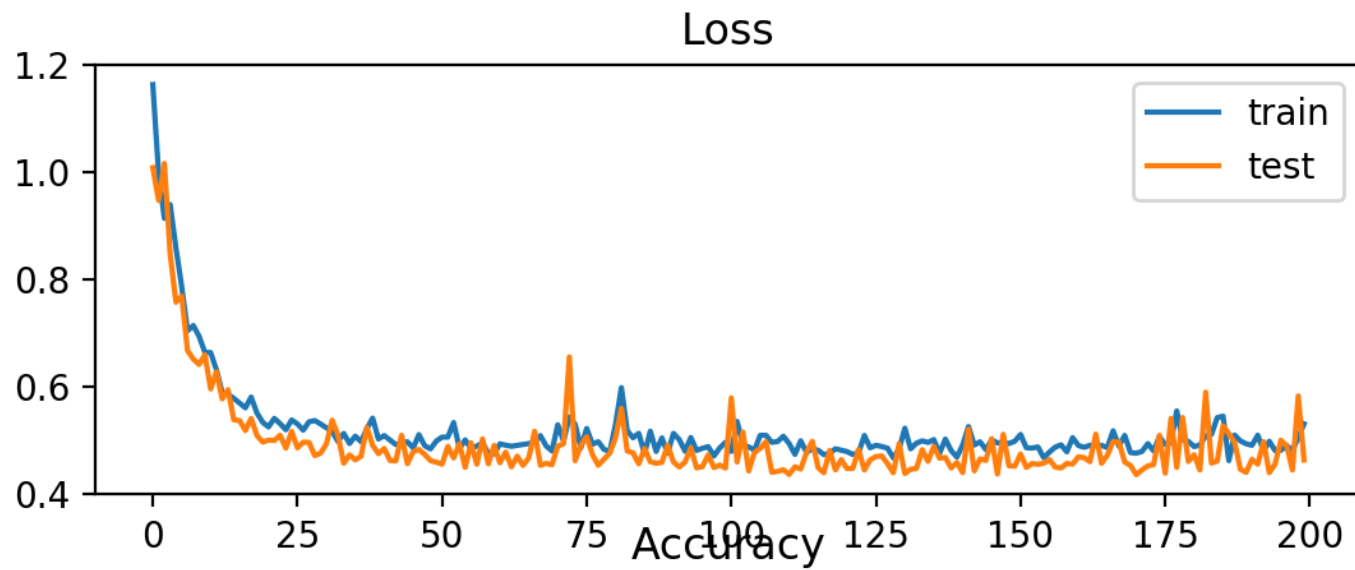
Gradient descent

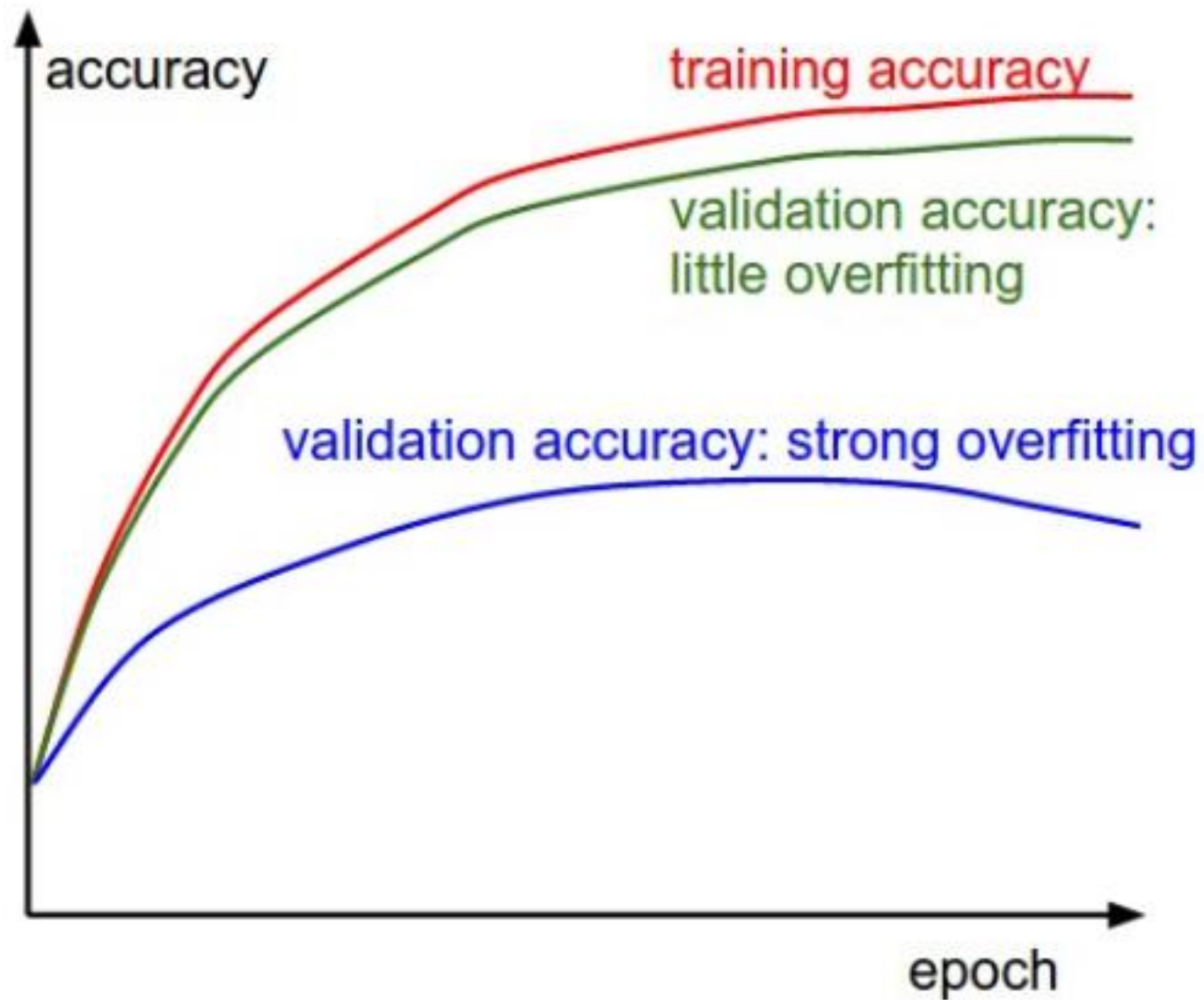


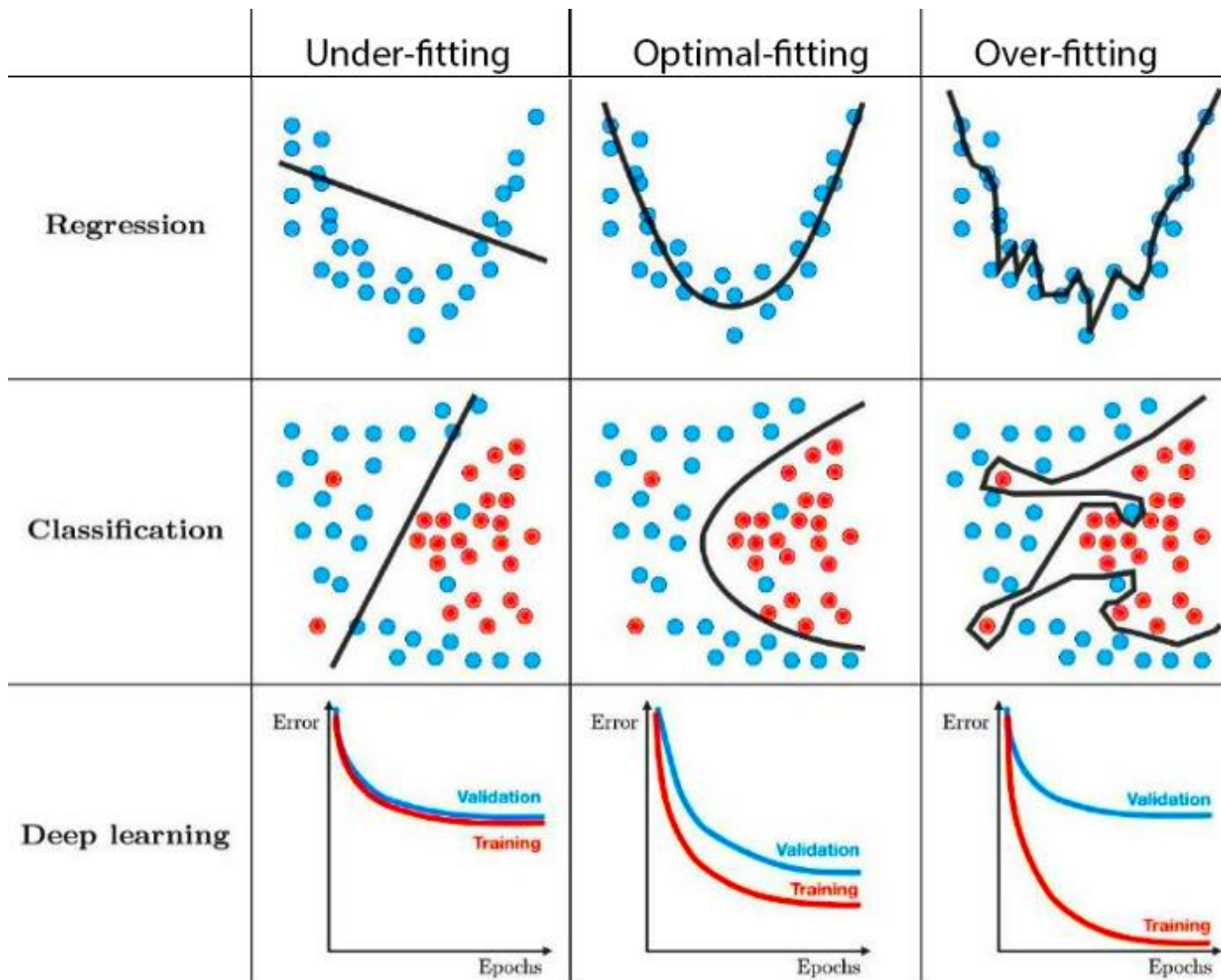
Stochastic Gradient Descent (SGD)

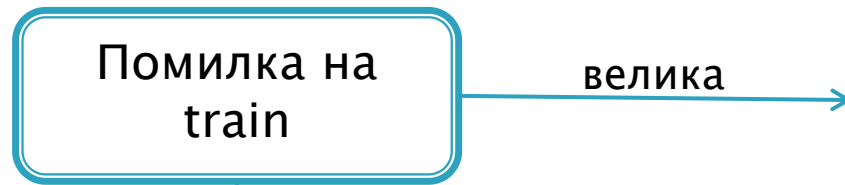












велика

Underfitting

Більше ресурсів для тренування
Більш складна модель
Інший підхід

мала

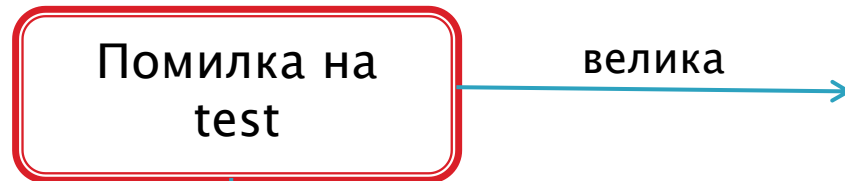


велика

Overfitting

Більше даних
Регуляризація, dropout
Інший підхід

мала



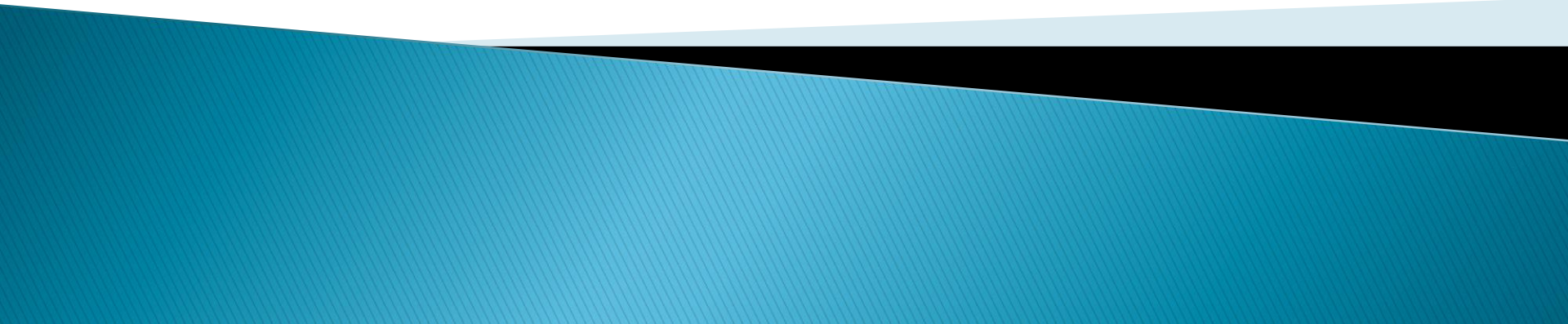
велика

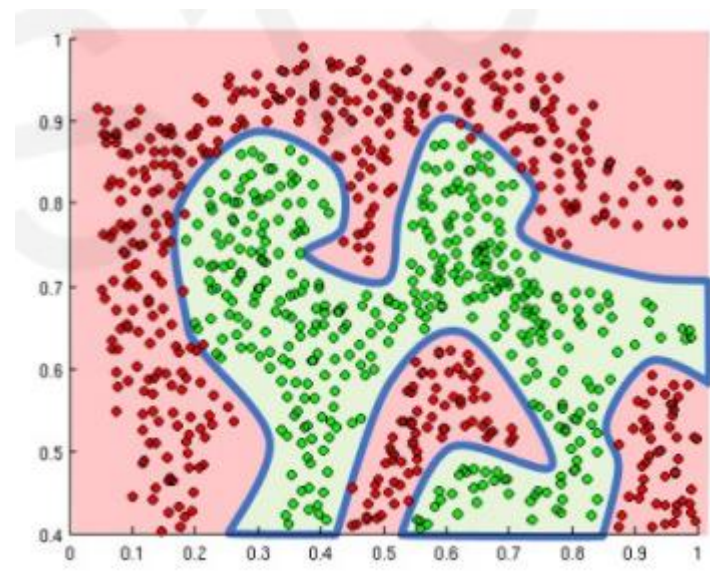
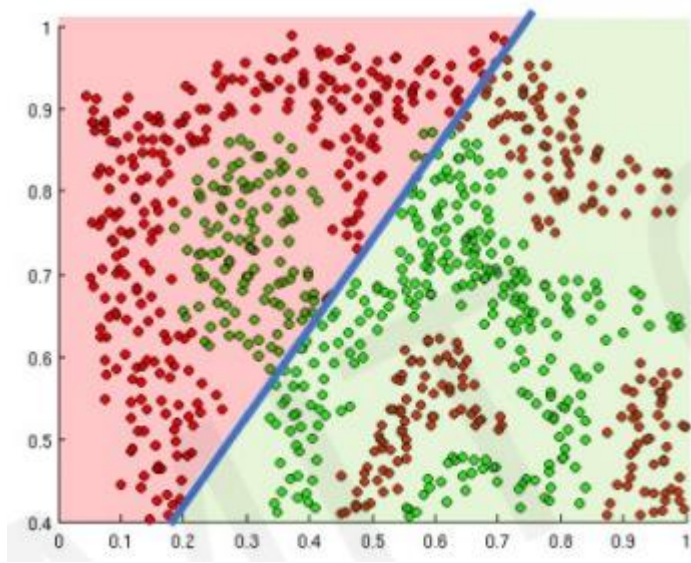
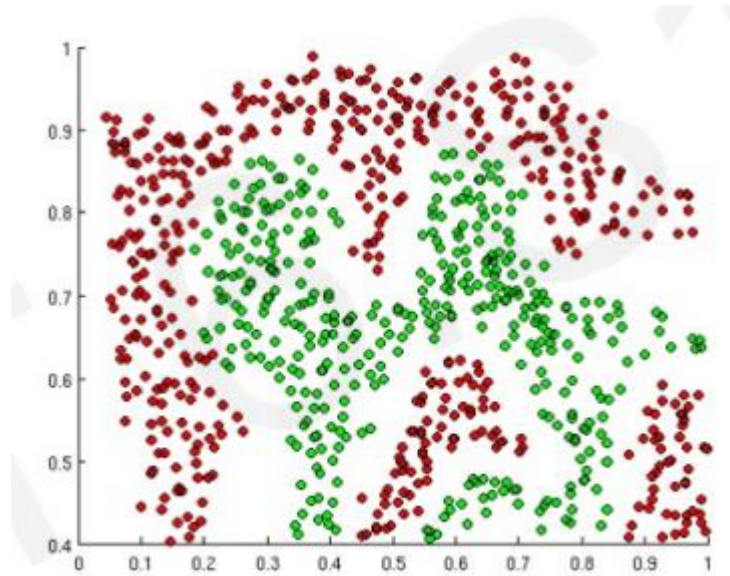
Відрізняються train і test

мала

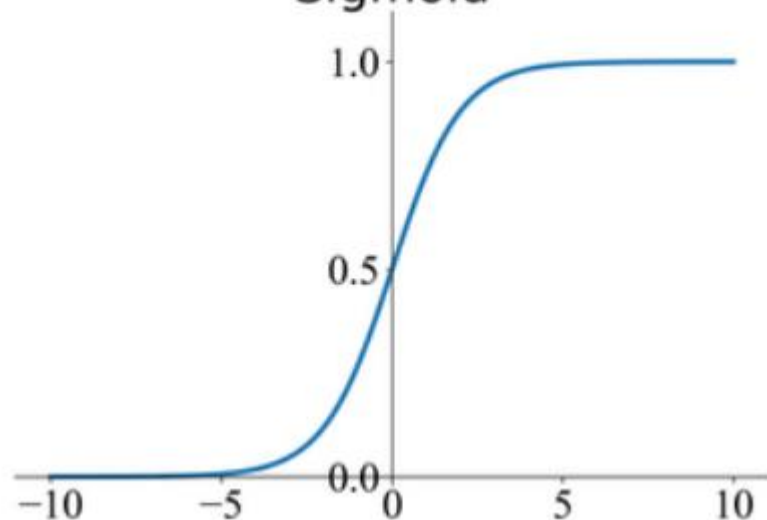
Застосовуємо на практиці

Activation functions



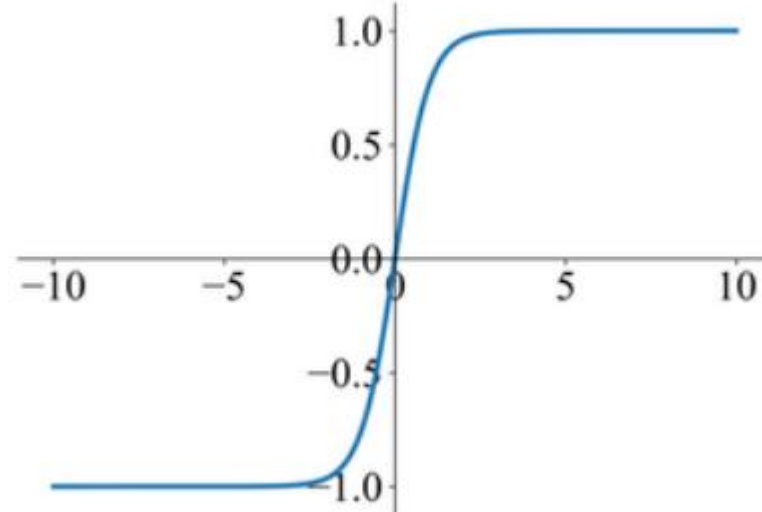


Sigmoid



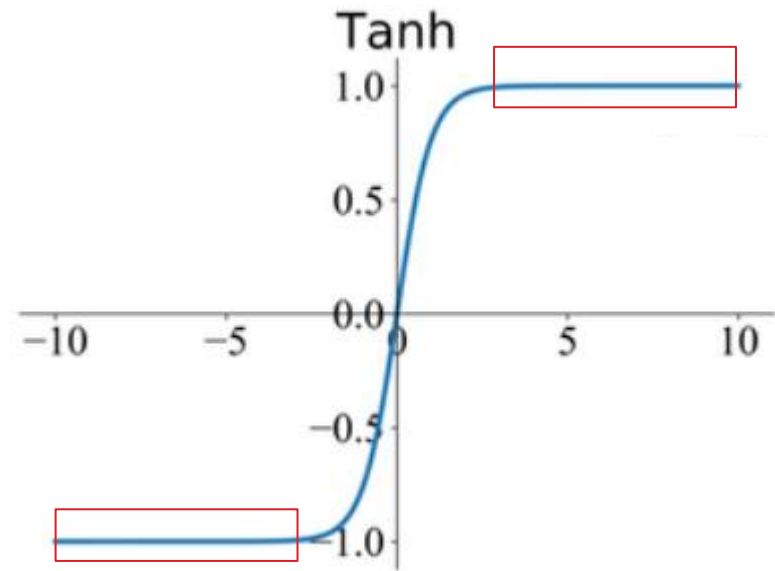
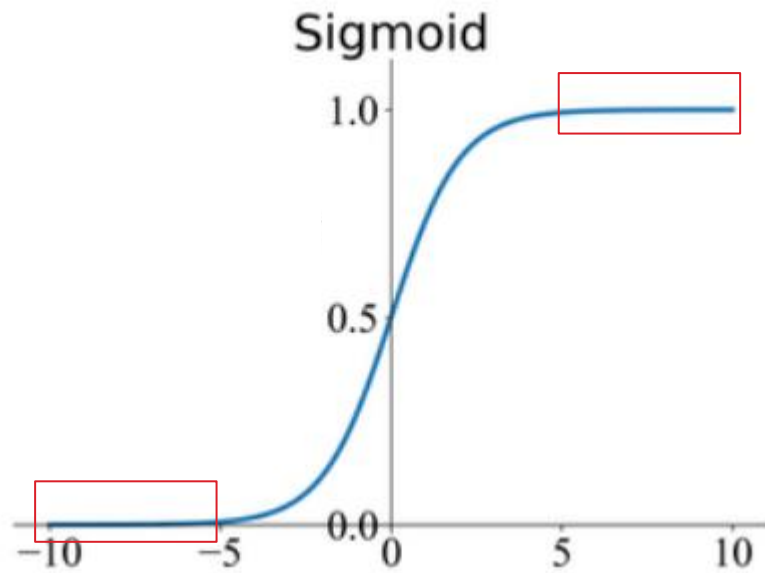
$$g(z) = \frac{1}{1 + e^{-z}}$$
$$g'(z) = g(z)(1 - g(z))$$

Tanh

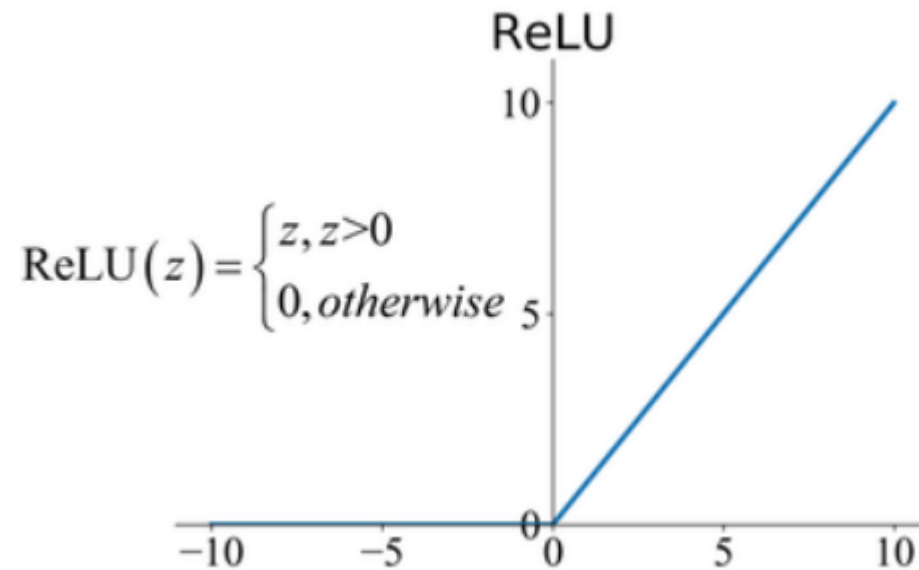


$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
$$g'(z) = 1 - g(z)^2$$

Vanishing gradient problem

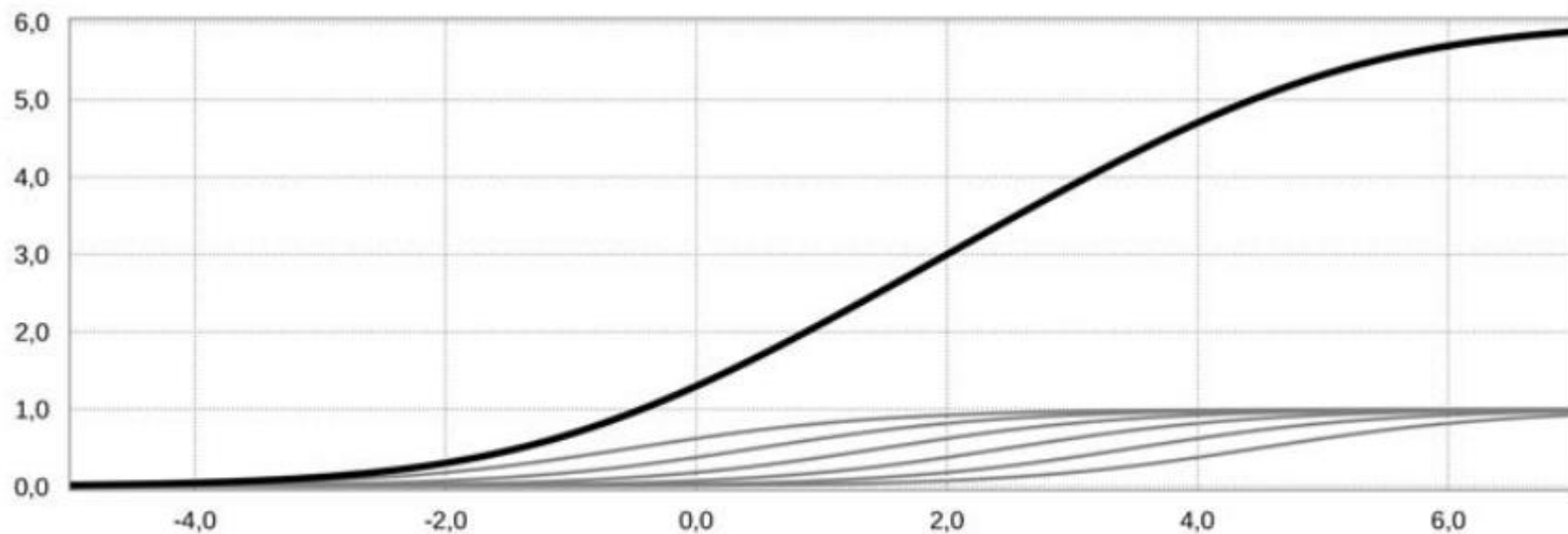


Rectified Linear Unit



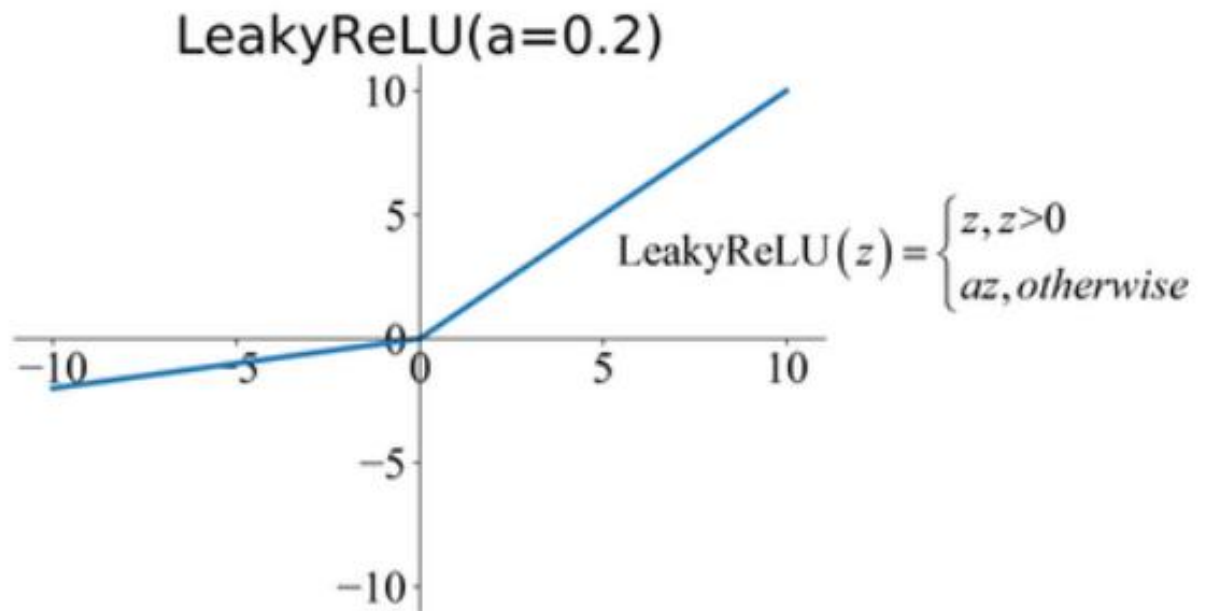
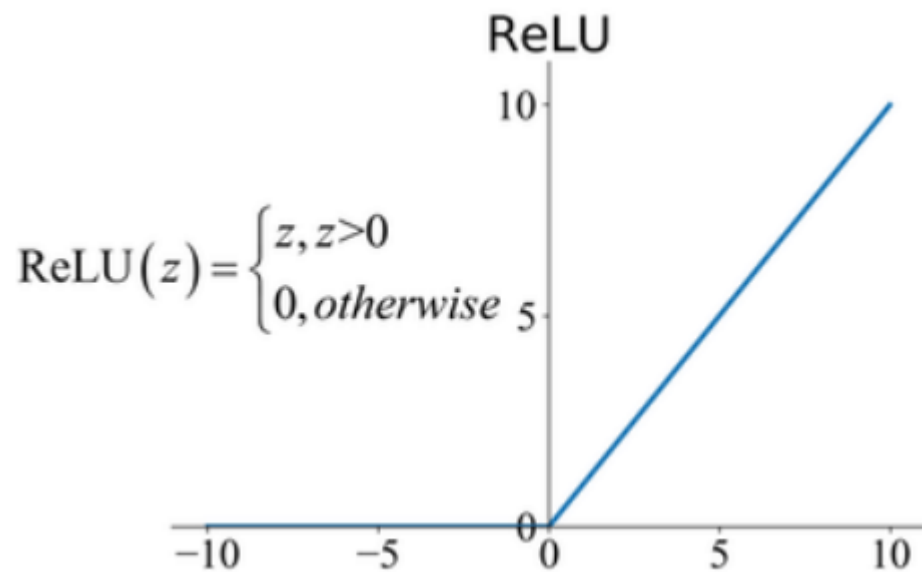
$$g(z) = \max(0, z)$$









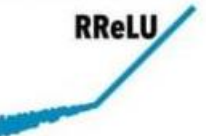
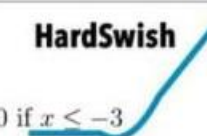
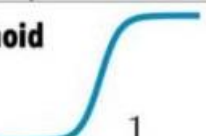
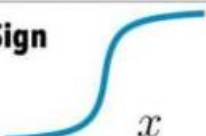

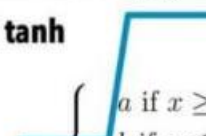
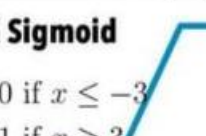
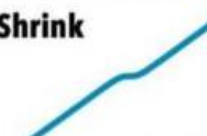
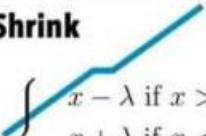
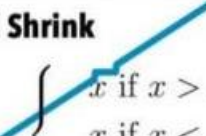
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$



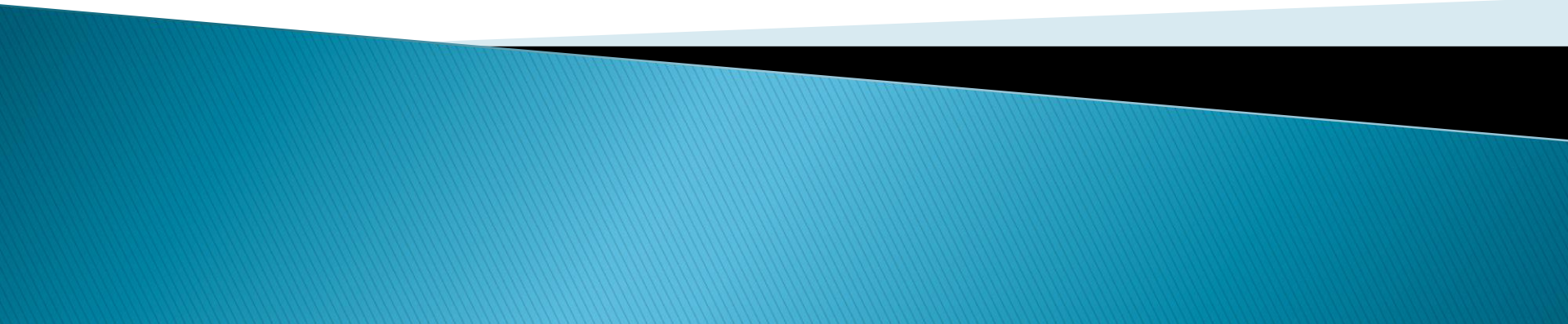
Сумма шести сигмоидов

$$f(x) = \sigma\left(x + \frac{1}{2}\right) + \sigma\left(x - \frac{1}{2}\right) + \sigma\left(x - \frac{3}{2}\right) + \sigma\left(x - \frac{5}{2}\right) + \dots$$



ReLU  $\max(0, x)$	GELU  $\frac{x}{2} \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + ax^3) \right) \right)$	PReLU  $\max(0, y_i) + a_i \min(0, y_i)$
ELU  $\begin{cases} x & \text{if } x > 0 \\ \alpha(x \exp x - 1) & \text{if } x < 0 \end{cases}$	Swish  $\frac{x}{1 + \exp -x}$	SELU  $\alpha(\max(0, x) + \min(0, \beta(\exp x - 1)))$
SoftPlus  $\frac{1}{\beta} \log(1 + \exp(\beta x))$	Mish  $x \tanh \left(\frac{1}{\beta} \log(1 + \exp(\beta x)) \right)$	RReLU  $\begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \text{ with } a \sim \mathcal{R}(l, u) \end{cases}$
HardSwish  $\begin{cases} 0 & \text{if } x < -3 \\ x & \text{if } x \geq 3 \\ x(x+3)/6 & \text{otherwise} \end{cases}$	Sigmoid  $\frac{1}{1 + \exp(-x)}$	SoftSign  $\frac{x}{1 + x }$
Tanh  $\tanh(x)$	Hard tanh  $\begin{cases} a & \text{if } x \geq a \\ b & \text{if } x \leq b \\ x & \text{otherwise} \end{cases}$	Hard Sigmoid  $\begin{cases} 0 & \text{if } x \leq -3 \\ 1 & \text{if } x \geq 3 \\ x/6 + 1/2 & \text{otherwise} \end{cases}$
Tanh Shrink  $x - \tanh(x)$	Soft Shrink  $\begin{cases} x - \lambda & \text{if } x > \lambda \\ x + \lambda & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$	Hard Shrink  $\begin{cases} x & \text{if } x > \lambda \\ x & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases}$

Model training with TensorFlow




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

The screenshot shows the TensorFlow API documentation for the `compile` method. The left sidebar contains a navigation menu with categories like `tf.feature_column`, `tf.graph_util`, `tf.image`, `tf.io`, and `tf.keras`. The `tf.keras` section is expanded, showing sub-entries like `Overview`, `Input`, `Model` (which is selected), `Sequential`, `activations`, `applications`, `backend`, `callbacks`, `constraints`, `datasets`, `dtensor`, `estimator`, `experimental`, `initializers`, `layers`, `losses`, `metrics`, `mixed_precision`, `models`, `optimizers`, `preprocessing`, `regularizers`, `saving`, and `utils`.

The main content area is titled `compile` and includes a `View source` link. It displays the following code snippet:

```
compile(
    optimizer='rmsprop',
    loss=None,
    metrics=None,
    loss_weights=None,
    weighted_metrics=None,
    run_eagerly=None,
    steps_per_execution=None,
    jit_compile=None,
    **kwargs
)
```

Below the code, it states: "Configures the model for training."

An **Example:** section shows the following code:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(),
                      tf.keras.metrics.FalseNegatives()])
```

At the bottom, there is a table with the following structure:

Args	
optimizer	String (name of optimizer) or optimizer instance. See tf.keras.optimizers .
loss	Loss function. May be a string (name of loss function), or a tf.keras.losses.Loss instance. See tf.keras.losses . A loss function is any callable with the signature <code>loss = fn(y_true, y_pred)</code> , where <code>y_true</code> are the ground truth values, and <code>y_pred</code> are the

On the right side of the page, there is a sidebar titled "On this page" with a list of methods and attributes, including `Args`, `Attributes`, `Methods`, `call`, `compile`, `compute_loss`, `compute_metrics`, `evaluate`, `fit`, `get_layer`, `get_metrics_result`, `get_weight_paths`, `load_weights`, `make_predict_function`, `make_test_function`, `make_train_function`, `predict`, `predict_on_batch`, `predict_step`, `reset_metrics`, `reset_states`, `save`, `save_spec`, and `save_weights`.



Public API for tf.keras.losses namespace.

Classes

`class BinaryCrossentropy`: Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between labels and predictions.

`class Hinge`: Computes the hinge loss between `y_true` and `y_pred`.

`class Huber`: Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback-Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss`: Loss base class.

`class MeanAbsoluteError`: Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError`: Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError`: Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError`: Computes the mean squared logarithmic error between `y_true` and `y_pred`.

`class Poisson`: Computes the Poisson loss between `y_true` and `y_pred`.

`class Reduction`: Types of loss reduction.

`class SparseCategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class SquaredHinge`: Computes the squared hinge loss between `y_true` and `y_pred`.

Реалізація власної Loss функції

```
def my_loss_function(y_true, y_pred):  
    return losses
```

```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])  
model.compile(optimizer='sgd', loss='my_huber_loss')
```

```
def my_huber_loss(y_true, y_pred):  
    threshold = 1  
    error = y_true - y_pred  
    is_small_error = tf.abs(error) <= threshold  
    small_error_loss = tf.square(error) / 2  
    big_error_loss = threshold * (tf.abs(error) - (0.5 * threshold))  
    return tf.where(is_small_error, small_error_loss, big_error_loss)
```

$$L_{\delta}(a) \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta \times (|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

```
history = model.fit(x_train, y_train, validation_split = 0.2, epochs=10, batch_size = 32)
```

The screenshot shows the TensorFlow API documentation for the `fit` method. The left sidebar contains a navigation menu with categories like `tf.feature_column`, `tf.graph_util`, `tf.image`, `tf.io`, and `tf.keras`. The `tf.keras` section is expanded, showing sub-items like `Overview`, `Input`, `Model` (which is selected), `Sequential`, `activations`, `applications`, `backend`, `callbacks`, `constraints`, `datasets`, `dtensor`, `estimator`, `experimental`, `initializers`, `layers`, `losses`, `metrics`, `mixed_precision`, `models`, `optimizers`, `preprocessing`, `regularizers`, `saving`, and `utils`.

The main content area displays the `fit` method signature and its source code. The signature is `fit(x=None, y=None, batch_size=None, epochs=1, verbose='auto', callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None, validation_batch_size=None, validation_freq=1, max_queue_size=10, workers=1, use_multiprocessing=False)`. The source code is a Python function that trains a model for a fixed number of epochs.

Below the source code, there is a description: "Trains the model for a fixed number of epochs (iterations on a dataset).".

The `Args` section lists the arguments and their descriptions:

- `x`: Input data. It could be:
 - A Numpy array (or array-like), or a list of arrays (in case the model has multiple inputs).
 - A TensorFlow tensor, or a list of tensors (in case the model has multiple inputs).

The right sidebar contains a list of attributes and methods for the `fit` method, including `call`, `compile`, `compute_loss`, `compute_metrics`, `evaluate`, `fit` (highlighted), `get_layer`, `get_metrics_result`, `get_weight_paths`, `load_weights`, `make_predict_function`, `make_test_function`, `make_train_function`, `predict`, `predict_on_batch`, `predict_step`, `reset_metrics`, `reset_states`, `save`, `save_spec`, `save_weights`, `summary`, `test_on_batch`, and `test_step`.

Реалізація побудови графіків функцій

```
import matplotlib.pyplot as plt
```

```
history = model.fit(train_ds,  
                    validation_data=val_ds,  
                    epochs=10)
```

```
acc      = history.history[ 'accuracy' ]  
val_acc  = history.history[ 'val_accuracy' ]  
loss     = history.history[ 'loss' ]  
val_loss = history.history[ 'val_loss' ]  
  
epochs   = range(len(acc)) # Get number of epochs  
  
plt.plot ( epochs, acc )  
plt.plot ( epochs, val_acc )  
plt.title ('Training and validation accuracy')  
plt.figure()  
  
plt.plot ( epochs, loss )  
plt.plot ( epochs, val_loss )  
plt.title ('Training and validation loss' )
```