Chapter 2:

# Mathematical building blocks of neural networks

# Outlines

1. Data representation of neural networks
2. Tensor operations
3. Gradient-based optimization

# Gradient-based optimization

- Review the example:
  - The neural layer transforms its input data as follows:

```
output = relu(dot(input, W) + b)
```

    - *W* and *b* are tensors that are attributes of the layer.
    - *W* and *b* are tensors that are attributes of the layer. They're called the weights or trainable parameters of the layer
    - These weights contain the information learned by the model from exposure to training data.
    - These weight matrices are filled with small random values in the first start (random).

# Gradient-based optimization

- Review the example:
  - The neural layer transforms its input data as follows (cont):

```
output = relu(dot(input, W) + b)
```

    - ✓ The next is to gradually adjust these weights based on a feedback signal.
    - ✓ This gradual adjustment, also called training, is the learning that machine learning is all about.
    - ✓ The gradual adjustment is continuous and forms a training loop until the loss seems *sufficiently low*.

# Gradient-based optimization

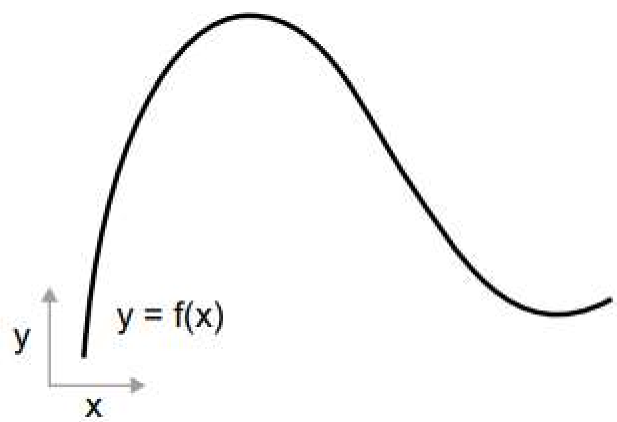
- The training loop:
  1. Draw a batch of training samples, ***x***, and corresponding targets, y_true.
  2. Run the model on ***x*** to obtain predictions, y_pred.
  3. Compute the loss of the model on the batch, a measure of the mismatch between y_pred and y_true.
  4. Update all weights of the model in a way that slightly reduces the loss on this batch -> *this is the significant problem*.
- The last model that has a very low loss on its training data: a low mismatch between predictions, y_pred, and expected targets, y_true.

# Gradient-based optimization

- Update all weights of the model in a manual way to find out the value that minimum the loss needs to maximum cost -> not able to.
- Automatic method for this problem: *gradient descent*
- Gradient descent is an optimization technique used in neural networks.
- Gradient descent bases on the fact that small changes in inputs lead to predictable changes in outputs -> loss function.
- By calculating the gradient, which describes how the loss changes with respect to the coefficients, we can update all coefficients simultaneously in a direction that **reduces the loss**.

# Gradient-based optimization

1. What is derivative?
   - Consider a continuous, $y = f(x)$, mapping a number, $x$, to a new number, $y$.
   - A small change in $x$ can only result in a small change in $y$ — that's the intuition behind continuity

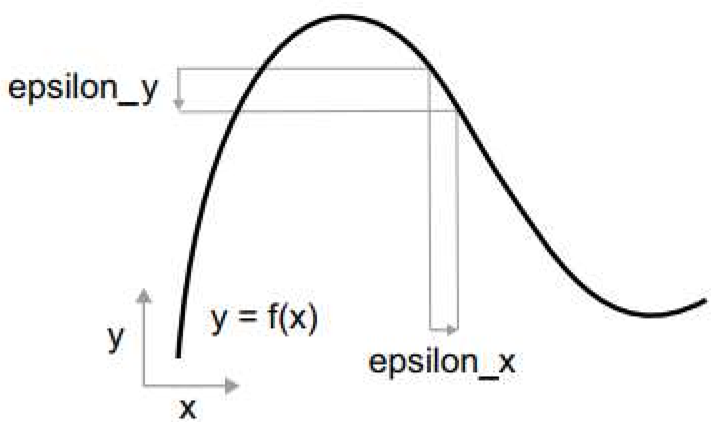



A continuous, smooth function

# Gradient-based optimization

1. What is derivative? (cont)
   - Let's say you increase *x* by a small factor, *epsilon_x*: this results in a small *epsilon_y* change to *y*.
   - When *epsilon_x* is small enough, around a certain point ***p***, it's possible to approximate ***f*** as a linear function of slope ***a***, so that *epsilon_y* becomes ***a*** * *epsilon_x*.

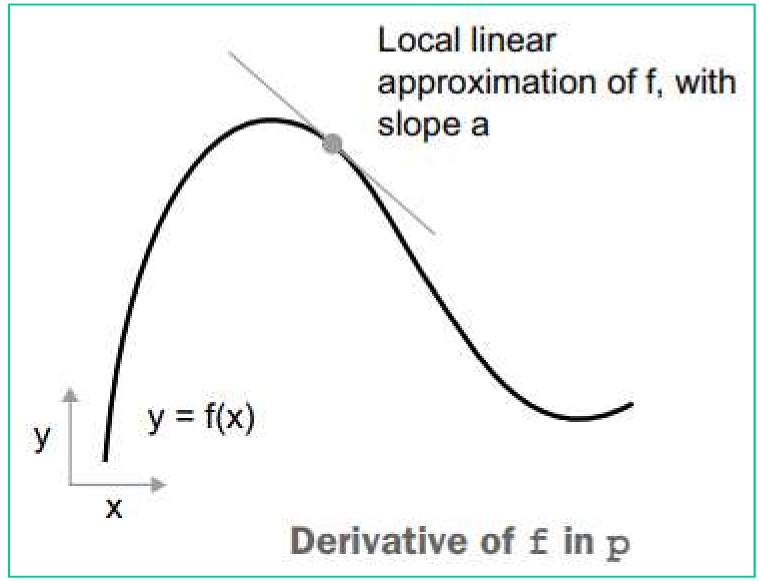
```
f(x + epsilon_x) = y + a * epsilon_x
```




**With a continuous function, a small change in x results in a small change in y.**

# Gradient-based optimization

1. What is derivative? (cont)
   - The slope a is called the derivative of ***f*** in ***p***.
   - If a is negative, it means a small increase in *x* around ***p*** will result in a decrease of *f(x)*, and if a is positive, a small increase in *x* will result in an increase of *f(x)*.
   - The absolute value of a (the magnitude of the derivative) tells us how quickly this increase or decrease will happen.



# Gradient-based optimization

1. What is derivative? (cont)
   - For every differentiable function *f(x)* there exists a derivative function *f'(x)*, that maps values of *x* to the slope of the local linear approximation of *f* in those points.
   - For instance, the derivative of cos(x) is -sin(x), the derivative of f(x) = a * x is f'(x) = a, and so on.
   - Being able to derive functions is a very powerful tool when it comes to optimization, the task of finding values of *x* that minimize the value of *f(x)*.
   - If we want to reduce the value of *f(x)*, we just need to move *x* a little in the opposite direction from the derivative.

# Gradient-based optimization

1. What is derivative? (cont)
   - Derivative of a tensor operation: The gradient
     - The derivative of a tensor operation (or tensor function) is called a gradient.
     - The gradient of a tensor function represents the curvature of the multidimensional surface described by the function.
     - The gradient of a tensor function characterizes how the output of the function varies when its input parameters vary.
     - Give a function f(x), we can reduce the value of f(x) by moving x a little in the opposite direction from the derivative.

# Gradient-based optimization

1. What is derivative? (cont)
   - Derivative of a tensor operation: The gradient (cont)
     - With a function f(W) of a tensor, we can reduce loss_value = f(W) by moving W in the opposite direction from the gradient.
     - For example, $W_1 = W_0$ - step * grad(f($W_0$), $W_0$) (where step is a small scaling factor).
     - That means going against the direction of steepest ascent of f, which intuitively should put we lower on the curve.
     - The scaling factor step is needed because grad(loss_value, $W_0$) only approximates the curvature when we're close to $W_0$, so we don't want to get too far from $W_0$.

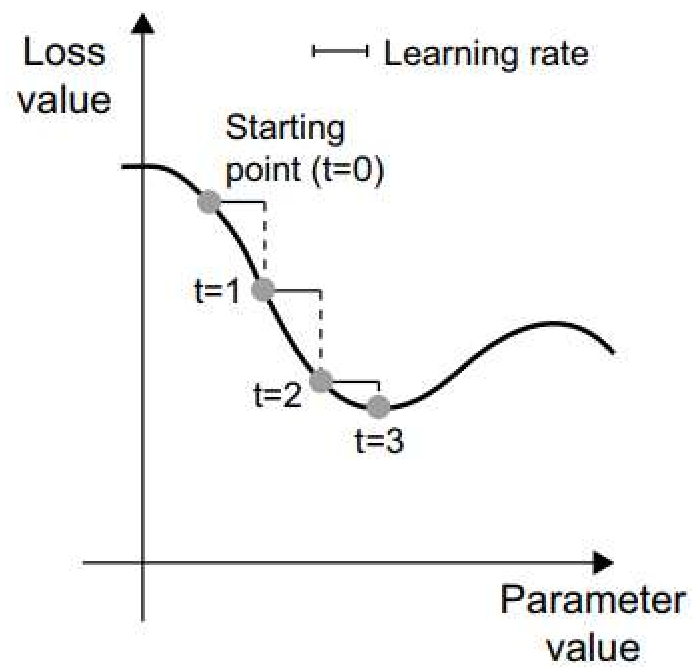
# Gradient-based optimization

2. Stochastic gradient descent
   - Given a differentiable function, the minimum is a point where the derivative is 0.
   - What we have to do is find all the points where the derivative goes to 0 and check for which of these points the function has the lowest value.
   - With a neural network, the smallest possible loss function is found by finding analytically the combination of weight values.
   - This can be done by solving the equation *grad(f(W), W) = 0* for ***W***. This is a polynomial equation of *N* variables, where *N* is the number of coefficients in the model.
   - Solving the equation *grad(f(W), W) = 0* for ***W*** will be difficult when *N* is a large number.

# Gradient-based optimization

2. Stochastic gradient descent (cont)

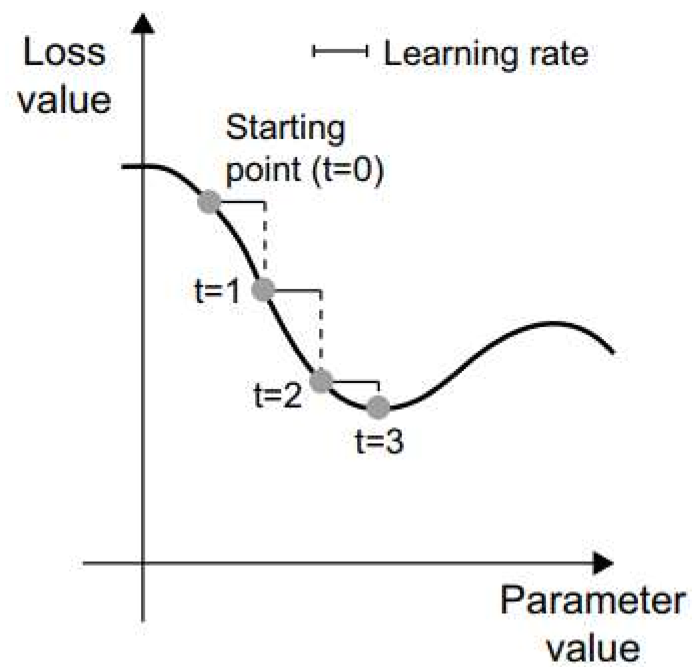
- The term stochastic refers to the fact that each batch of data is drawn at random (stochastic is a scientific synonym of random).
- The image illustrates what happens in 1D, when the model has only one parameter and you have only one training sample.




SGD down a 1D loss curve (one learnable parameter)

# Gradient-based optimization

2. Stochastic gradient descent (cont)
   - It's important to pick a reasonable value for the learning_rate factor.
   - If it's too small, the descent down the curve will take many iterations, and it could get stuck in a local minimum.
   - If learning_rate is too large, the value updates may end up taking we to completely random locations on the curve.



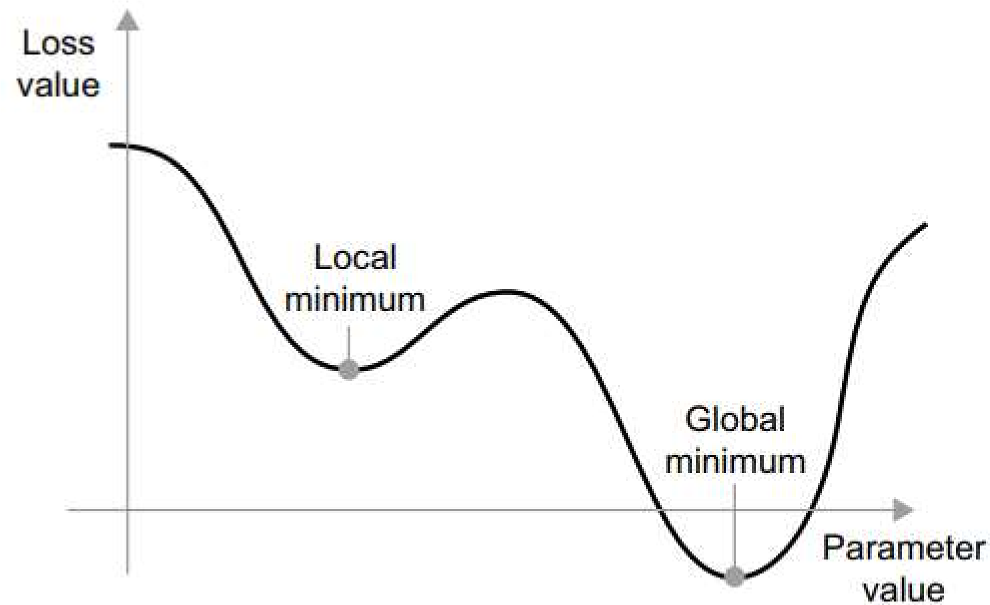

SGD down a 1D loss curve (one learnable parameter)

# Gradient-based optimization

2. Stochastic gradient descent (cont)
   - We can use gradient descent in highly dimensional spaces: every weight coefficient in a neural network is a free dimension in the space, and there may be tens of thousands or even millions of them.

# Gradient-based optimization

2. Stochastic gradient descent (cont)
   - The optimization method or optimizers: taking into account previous weight updates when computing the next weight update, rather than just looking at the current value of the gradients.




A local minimum and a global minimum

   - Momentum: addresses two issues as convergence speed and local minima.

# Gradient-based optimization

2. Stochastic gradient descent (cont)
   - In practice, this means updating the parameter w based not only on the current gradient value but also on the previous parameter update.

```
past_velocity = 0.
momentum = 0.1                  <-- Constant momentum factor
while loss > 0.01:              <-- Optimization loop
    w, loss, gradient = get_current_parameters()
    velocity = past_velocity * momentum - learning_rate * gradient
    w = w + momentum * velocity - learning_rate * gradient
    past_velocity = velocity
    update_parameter(w)
```

# Gradient-based optimization

3. Back-propagation algorithm
   - The Backpropagation algorithm comes in to:
     - ✓ Compute the gradient of complex expressions in practice.
     - ✓ Compute the gradient of the loss with regard to the weights.

# Gradient-based optimization

3. Back-propagation algorithm (cont)
   - The chain rule:
     - ✓ Backpropagation is a method that uses the derivatives of basic operations (e.g., addition, ReLU, tensor product) to compute the gradient of complex combinations of these operations efficiently.
     - ✓ Neural networks, composed of chained tensor operations with simple derivatives, leverage this.
     - ✓ A model can be parameterized by weights and biases across layers, involving differentiable operations like dot products, ReLU, softmax, and loss functions.

# Gradient-based optimization

3. Back-propagation algorithm (cont)
   - The chain rule (cont):
     - ✓ For example, give a model:

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

     - ✓ It can be expressed as a function parameterized by the variables W1, b1, W2, and b2 (belonging to the first and second Dense layers respectively), involving the atomic operations dot, relu, softmax, and +, as well as loss function => it called the **chain rule.**

```
loss_value = loss(y_true, softmax(dot(relu(dot(inputs, W1) + b1), W2) + b2))
```

# Gradient-based optimization

3. Back-propagation algorithm (cont)
   - The chain rule (cont):
     - ✓ Backpropagation is a method that uses the derivatives of basic operations (e.g., addition, ReLU, tensor product) to compute the gradient of complex combinations of these operations efficiently.
     - ✓ Neural networks, composed of chained tensor operations with simple derivatives, leverage this.
     - ✓ A model can be parameterized by weights and biases across layers, involving differentiable operations like dot products, ReLU, softmax, and loss functions.