→ *03/10/2024*

Optimization

DAT-300

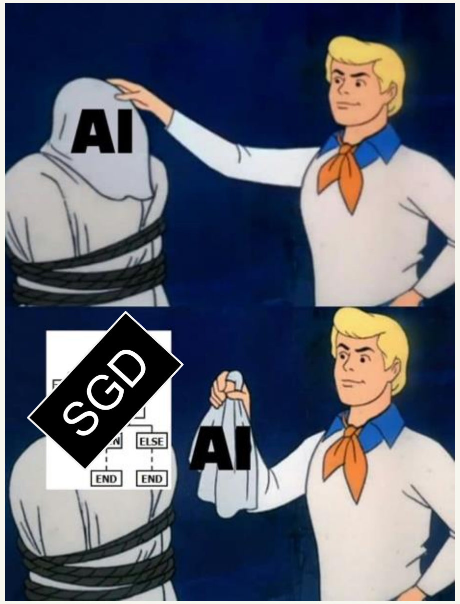
Agenda

- → Motivation
- → Key Concepts
- → Optimization Algorithms
- → Challenges
- → Hands on-application

Motivation (θ refers to $\boldsymbol{w}$ in the jupyter notebook)

$$\theta^* \in \underset{\theta \in \Theta}{\operatorname{argmin}} \mathcal{L}(\theta)$$

- Optimization: finding the minimum/maximum value of a function
  - In ML: by adjusting model parameters to minimize an objective function (cost function, loss function) approximated by the data.
  - Approximation: local minimum that perform “well enough” → better generalization?
- *“… all of deep learning is powered by one very important algorithm: stochastic gradient descent (1869/1952)”* - Ian Goodfellow



Key Concepts

- Backpropagation ≠ optimization
  - Computational graph → backward pass
  - Gradients; partially derivatives (rate of change)
- Object (cost, loss) function
- Convex vs Non-convex
- Loss Landscape
  - Min, max, saddle point

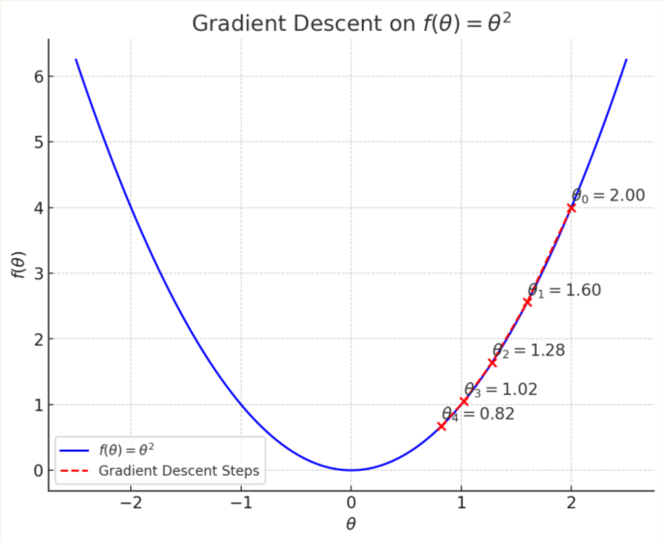
- Smooth vs nonsmooth
  - Continuous differentiable
  - Lipschitz continuity
- Jacobian matrix
  - Generalization to a vector valued function
- Hessian matrix
  - Second order derivative matrix
  - Explains the curvature
  - Newton Method

  → n^2 - elements

Gradient Descent - (θ refers to w in the jupyter notebook)

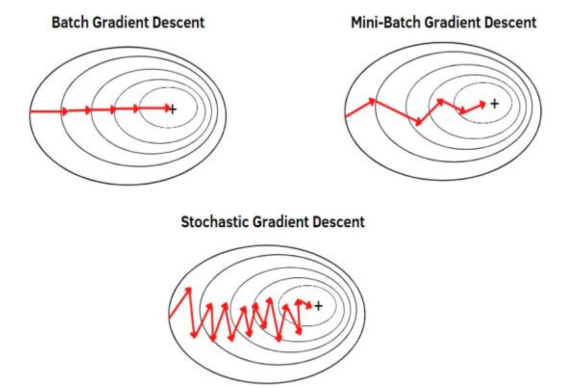
- An iterative optimization algorithm to find the/a minimum of a function
- Minimize discrepancy between prediction and true label/value
- Move in the the direction of the steepest decrease (opposite of the Gradient)

- I.e. using MSE
  - $\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}[(h_\theta(x^i) - y^{(i)}]^2$
  - e.g. linear regression $\rightarrow h_\theta(x^{(i)}) = wx^{(i)} + b$
  - $Repet\ until\ converge\ (treshold\ is\ reach)$
    $$\theta \leftarrow \theta_{old} - \eta\nabla\mathcal{L}(\theta)$$
  - Learning rate: η



Stochastic Gradient Descent (SDG)

- Problem with GD?
  - $\frac{(3x224x224)x4\ x\ 10000\ samples}{1024**3} \approx 5.5GB$
- Batch size $m \ll N$
  - $m = 1$ [Stochastic GD]
  - $m = 2^x$ [Mini-batch GD (aka SGD in ML)]
  - $m = N$ [(Batch) GD]



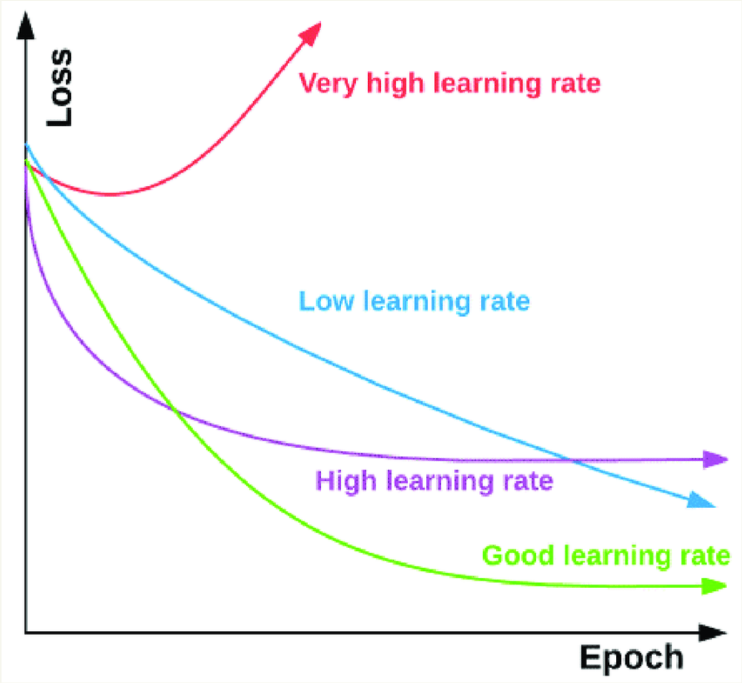
- Error in computing mean $\rightarrow \frac{\sigma}{\sqrt{m}}$
  - **Diminishing returns -** the more samples you use, the smaller the error becomes, but the improvement slows down as m increases

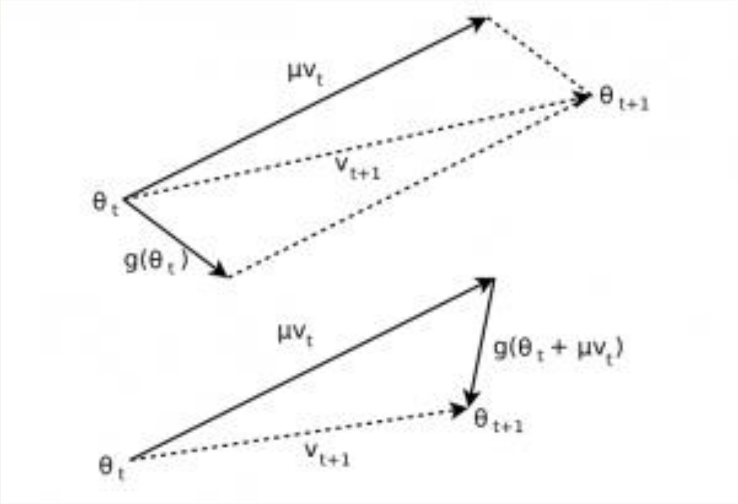
	Small batch size	Large batch size
Gradient estimate	Noisy (high variance)	Smooth (low variance)
Convergence speed	Fast (unstable)	Slow (stable)
Memory consumption	Low	High
Generalization	Overfit (sample importance)	Better
Learning rate	Small	Higher
Updates	Frequent	Fewer
Common batch sizes	32, 64, 128, (256, 512)	> 2048

Momentum - (θ refers to w in the jupyter notebook)

- Learning rate ⇒ $\theta \leftarrow \theta_{old} - \boldsymbol{\eta}\nabla\mathcal{L}(\theta)$
- Problem
  - Oscillation (when steep), slow convergence (when flat), stuck in local minimum
- Momentum ⇒ $mass\ x\ velocity$
- Update rule:
  - $v_t = \gamma v_{t-1} + \boldsymbol{\eta}\nabla\mathcal{L}(\theta_{t-1})$
  - $\theta \leftarrow \theta_{t-1} - \mathrm{v_t}$

  → Moment coefficient → γ $[0.9, 0.99]$
- Moving average of past gradients → additional hyperparameter
- Nesterov Momentum (“look ahead”) → $v_t = \gamma v_{t-1} + \boldsymbol{\eta}\nabla\mathcal{L}(\theta_{t-1}, v_t)$





ML Optimization Algorithm - (θ refers to $\boldsymbol{w}$ in the jupyter notebook)

*"…researchers have long realized that the **learning rate** … the hyperparameters that is the **most difficult to set** … a **significant impact** on model performance." Ian Goodfellow*

Solution ⇒ SGD w/ adaptive Learning rate

- AdaGrad → **Ada**ptive **Grad**ient
- RMSProp → **R**oot **M**ean **S**quare **Prop**agation
- Adam → **Ada**ptive **M**oment (Estimation)

RMSProp: $G_t = \sqrt{\frac{1}{t}\sum_{\tau}^{t} g_\tau g_\tau^T}$

Adam: RMSProp + Momentum

AdaGrad:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon I + diag(G_t)}} \cdot g_t, \quad \Big| \quad G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^\top .$$

$$\begin{bmatrix} \theta_{t+1}^{(1)} \\ \theta_{t+1}^{(2)} \\ \vdots \\ \theta_{t+1}^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_t^{(1)} \\ \theta_t^{(2)} \\ \vdots \\ \theta_t^{(m)} \end{bmatrix} - \eta \left(\begin{bmatrix} \varepsilon & 0 & \cdots & 0 \\ 0 & \varepsilon & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \varepsilon \end{bmatrix} + \begin{bmatrix} G_t^{(1,1)} & 0 & \cdots & 0 \\ 0 & G_t^{(2,2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & G_t^{(m,m)} \end{bmatrix} \right)^{-1/2} \cdot \begin{bmatrix} g_t^{(1)} \\ g_t^{(2)} \\ \vdots \\ g_t^{(m)} \end{bmatrix}, \tag{4}$$

$$\begin{bmatrix} \theta_{t+1}^{(1)} \\ \theta_{t+1}^{(2)} \\ \vdots \\ \theta_{t+1}^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_t^{(1)} \\ \theta_t^{(2)} \\ \vdots \\ \theta_t^{(m)} \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{\varepsilon + G_t^{(1,1)}}} & 0 & \cdots & 0 \\ 0 & \frac{\eta}{\sqrt{\varepsilon + G_t^{(2,2)}}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\eta}{\sqrt{\varepsilon + G_t^{(m,m)}}} \end{bmatrix} \cdot \begin{bmatrix} g_t^{(1)} \\ g_t^{(2)} \\ \vdots \\ g_t^{(m)} \end{bmatrix}. \tag{5}$$

$$\begin{bmatrix} \theta_{t+1}^{(1)} \\ \theta_{t+1}^{(2)} \\ \vdots \\ \theta_{t+1}^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_t^{(1)} \\ \theta_t^{(2)} \\ \vdots \\ \theta_t^{(m)} \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{\varepsilon + G_t^{(1,1)}}} g_t^{(1)} \\ \frac{\eta}{\sqrt{\varepsilon + G_t^{(2,2)}}} g_t^{(2)} \\ \vdots \\ \frac{\eta}{\sqrt{\varepsilon + G_t^{(m,m)}}} g_t^{(m)} \end{bmatrix}.$$

SGD:

$$\begin{bmatrix} \theta_{t+1}^{(1)} \\ \theta_{t+1}^{(2)} \\ \vdots \\ \theta_{t+1}^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_t^{(1)} \\ \theta_t^{(2)} \\ \vdots \\ \theta_t^{(m)} \end{bmatrix} - \begin{bmatrix} \eta g_t^{(1)} \\ \eta g_t^{(2)} \\ \vdots \\ \eta g_t^{(m)} \end{bmatrix}.$$

tf.keras.optimizers

```python
import tensorflow as tf

# Stochastic gradient descent, with or without momentum
op1=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.0,
                            nesterov=False, name='SGD', **kwargs)

op2=tf.keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9,
                                momentum=0.0, epsilon=1e-07,
                                centered=False,name='RMSprop',**kwargs)

op3=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9,
                             beta_2=0.999, epsilon=1e-07,
                             amsgrad=False,name='Adam', **kwargs)
```

```python
# Train step for a single batch
def train_step(image, y):

        # get predictions loss and update model parameters
        with tf.GradientTape() as tape:
                y_pred = model(image, training=True)
                loss = loss_fn(y, y_pred)
        grads = tape.gradient(loss, model.trainable_variables)
        optimizer_fn.apply_gradients(zip(grads, model.
            trainable_variables))
```

```python
model.compile(
    optimizer=optimizer_fn,  # Optimizer
    loss=loss_fn,# Loss function to minimize
    metrics=[metric_fn],# List of metrics to monitor
)

history = model.fit(
    train_dataset,
    batch_size=64,
    epochs=2,
    validation_data=val_dataset,
)
```

Strategies to improve optimization

In the algorithm

1. Parameter initialization
    - *keras.kernel_initializer()*
2. Learning rate scheduling/decay
    - *keras.callbacks.LearningRateScheduler()*
3. Early stopping

In constructing the model

1. Activation function
    - Sigmoid, ReLU, tanh … ⇒ vanishing and exploding gradient
2. Batch Normalization
    - Stable gradient
    - Deeper network ⇒ sensitive to input distribution