ADVANCE IN GRADIENT BASED OPTIMIZATION METHOD FOR DEEP NEURAL NETWORK

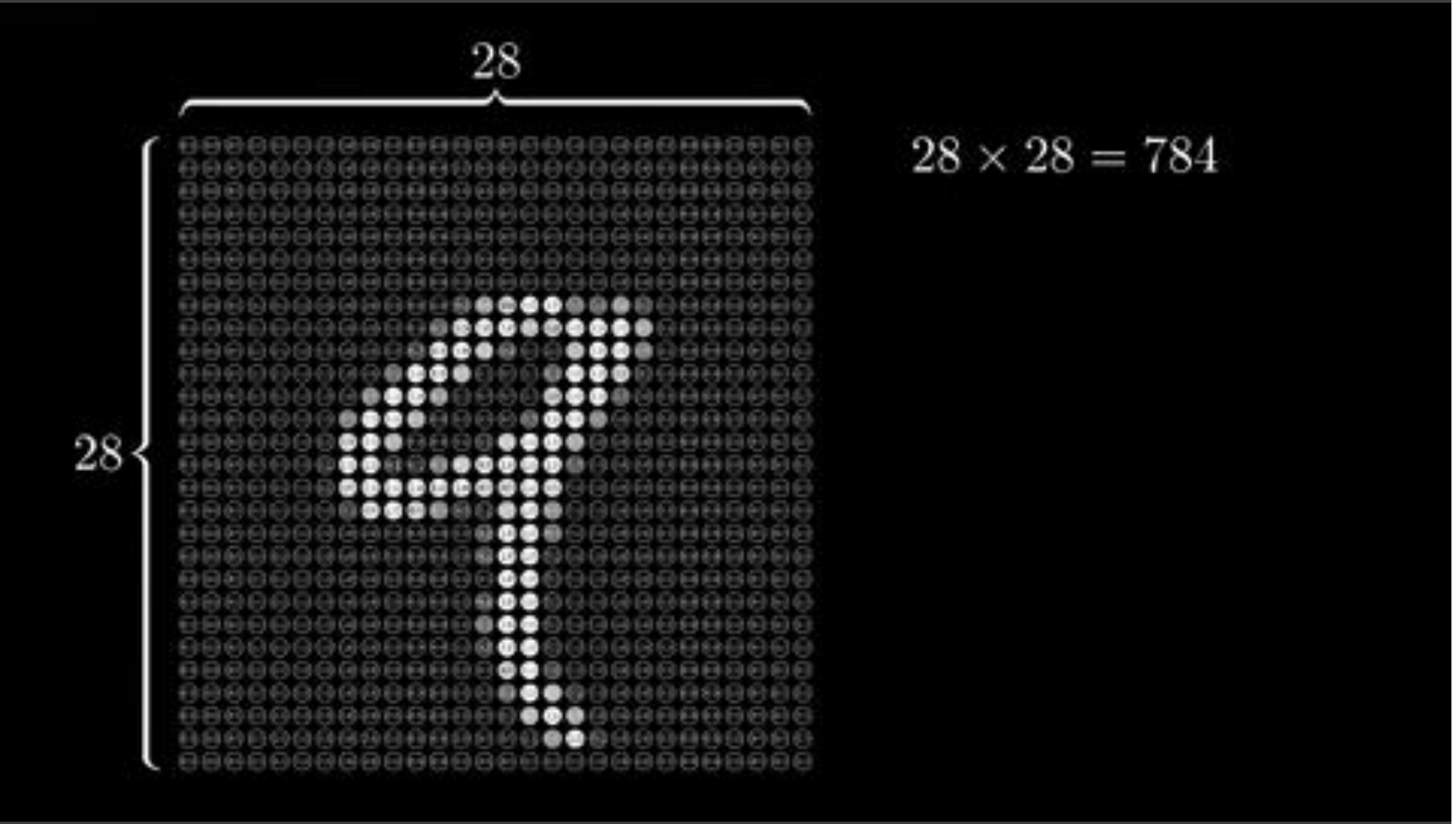
AN EXPLORATION GUIDE FOR THE MOVING LAND.

By Msc. Andrinandrasana David Rasamoelina

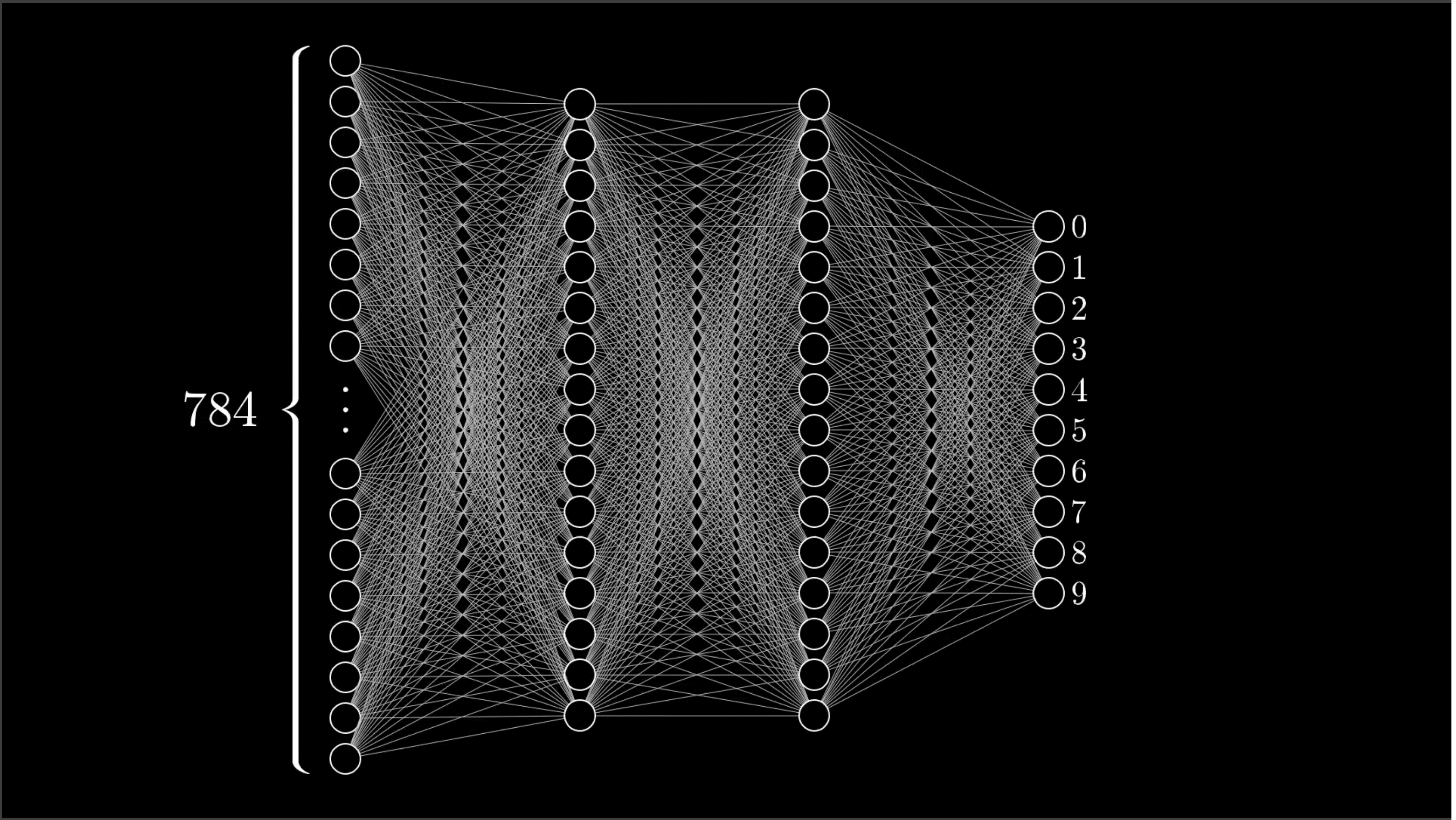
CENTER FOR INTELLIGENT TECHNOLOGIES

INTRODUCTION

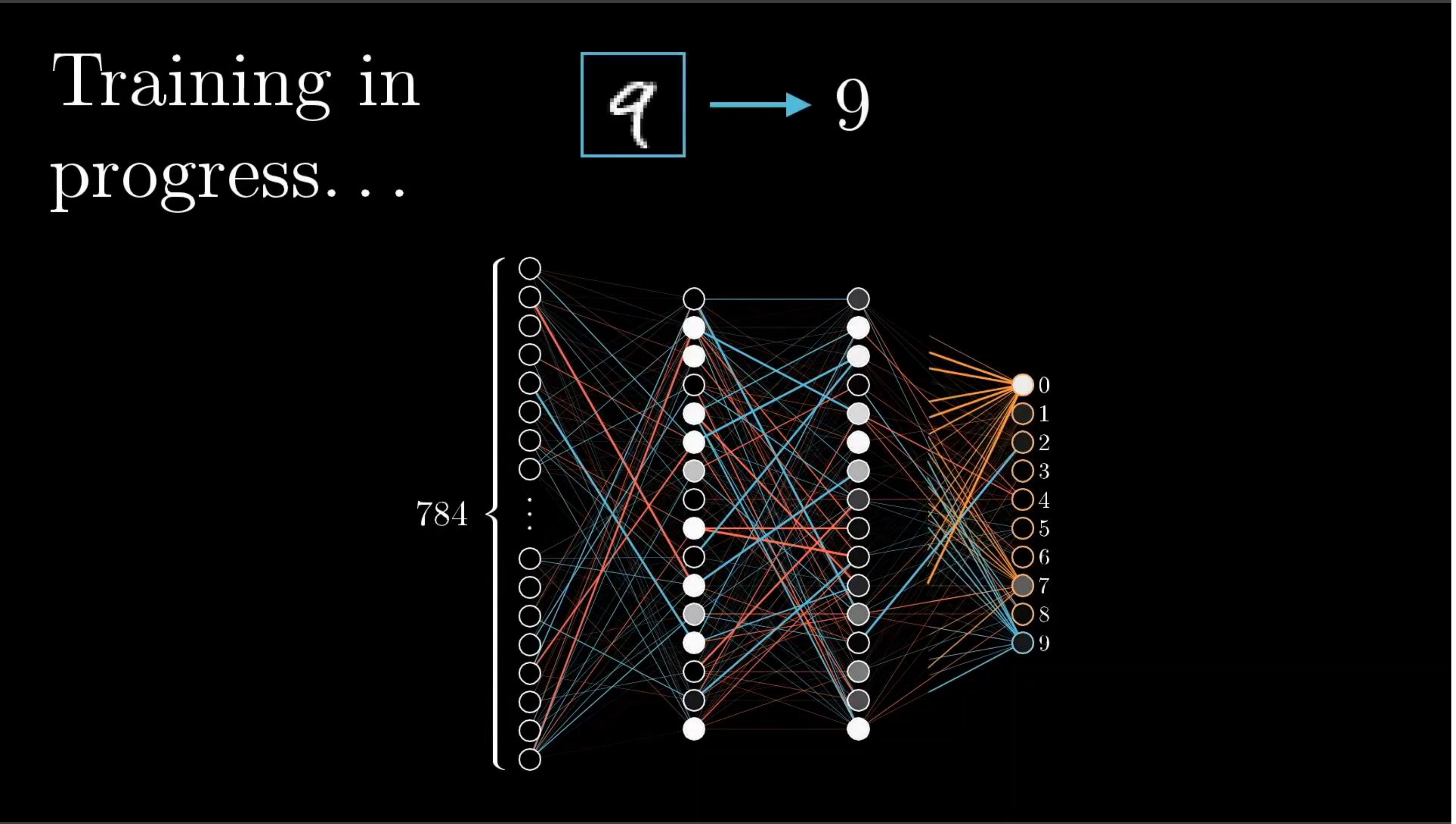
- A brief overview of steps for training Neural Network
- Explanation of different variant of gradient descent
- Types and current state of the art optimizer for Neural Network



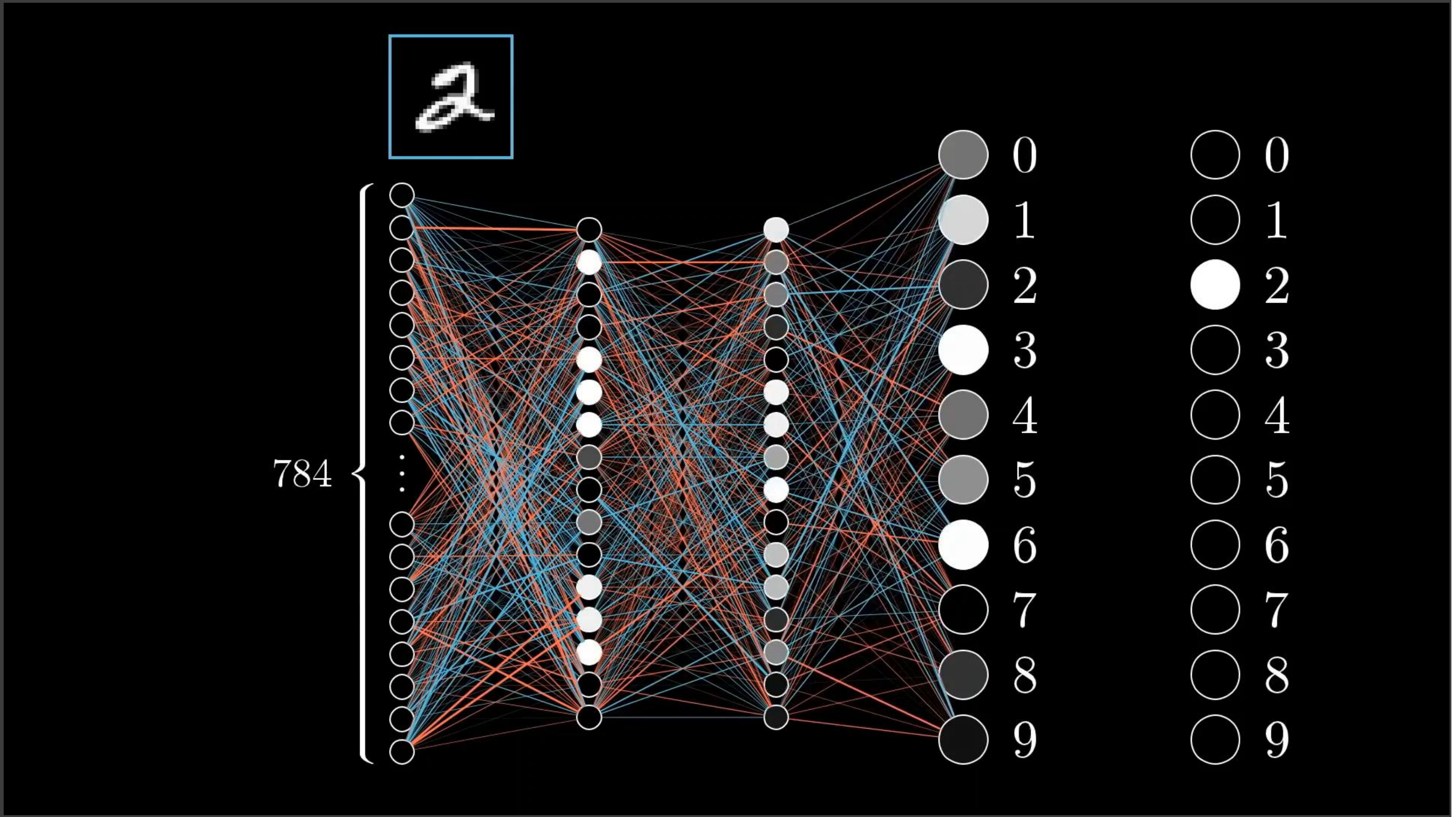
NEURAL NETWORK



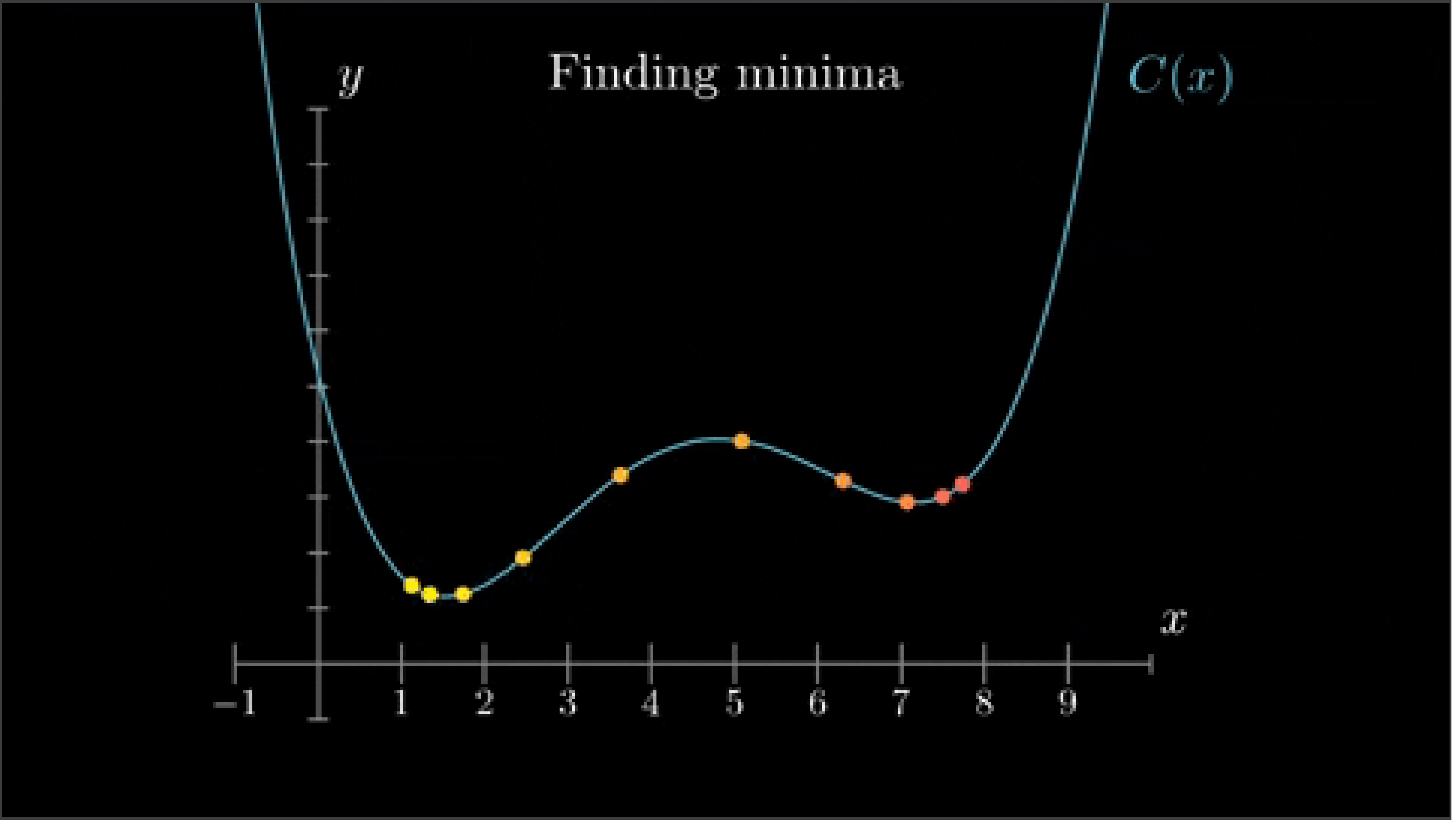
NEURAL NETWORK



BACKPROPAGATION

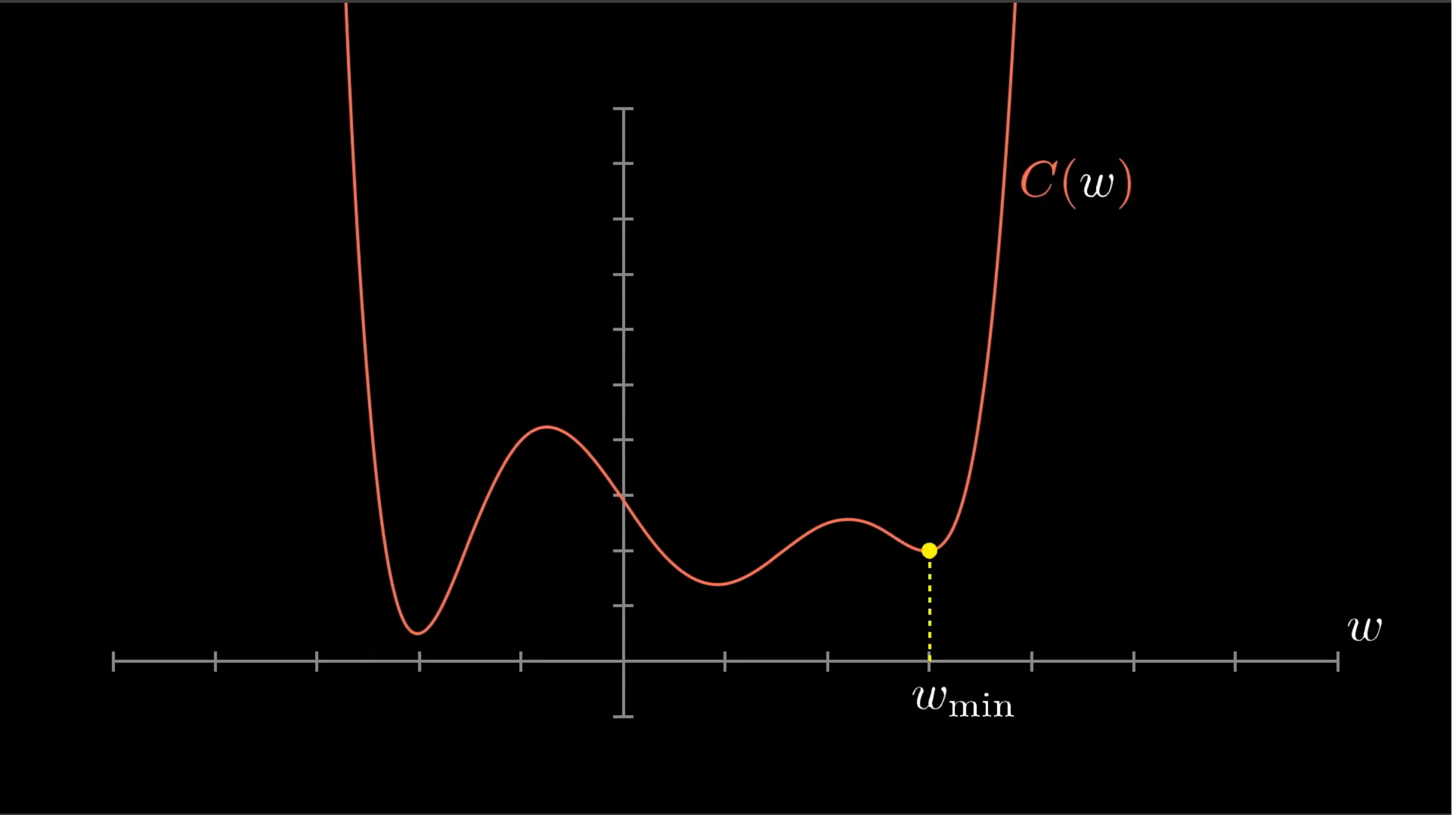


BACKPROPAGATION



GRADIENT DESCENT

GRADIENT DESCENT IS A WAY TO MINIMIZE AN OBJECTIVE FUNCTION.



GRADIENT DESCENT

GRADIENT DESCENT IS A WAY TO MINIMIZE AN OBJECTIVE FUNCTION.

GRADIENT DESCENT VARIANT

Vanilla / Batch Gradient Descent

Stochastic Gradient Descent

Mini-Batch Gradient Descent

BATCH GRADIENT DESCENT

- Computes the gradient of the cost function for the entire datasets.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

η is the learning rate

θ is the model parameter

BATCH GRADIENT DESCENT

PROS

- Guaranty to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

CONS

- Intractable

STOCHASTIC GRADIENT DESCENT

- Computes the gradient of the cost function for the entire datasets.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

STOCHASTIC GRADIENT DESCENT

PROS

- Faster
- Can be used to learn online.

CONS

- Performs frequent updates with a high variance that cause the objective function to fluctuate heavily
- Complicates convergence to the exact minimum

MINI-BATCH GRADIENT DESCENT

- Computes the gradient of the cost function for the entire datasets.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

MINI-BATCH GRADIENT DESCENT

PROS

- Reduces the variance of the parameter updates, which can lead to more stable convergence
- Can make use of highly optimized matrix optimizations

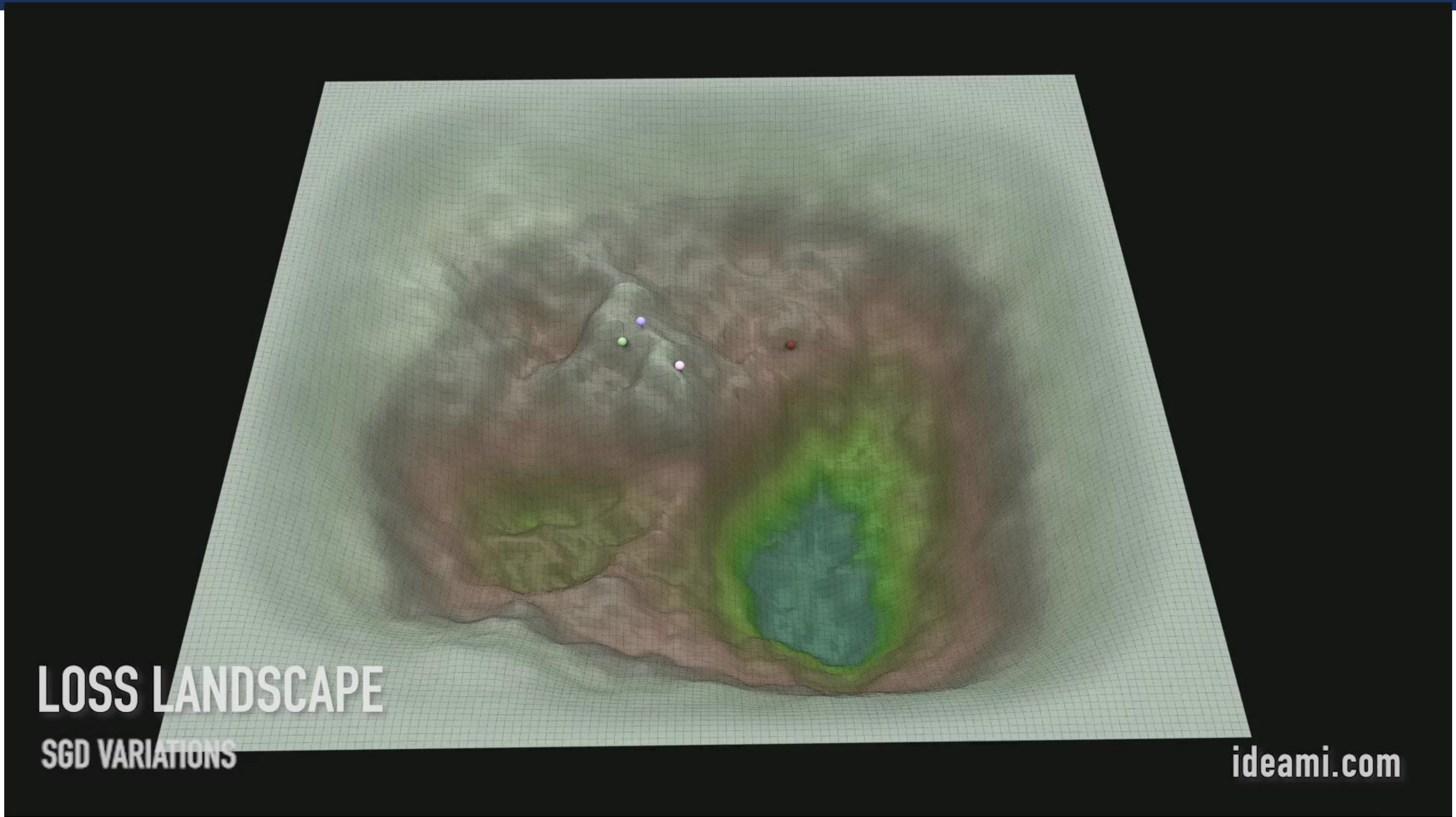
CONS

- Does not guarantee good convergence
- Choosing a proper learning rate can be difficult

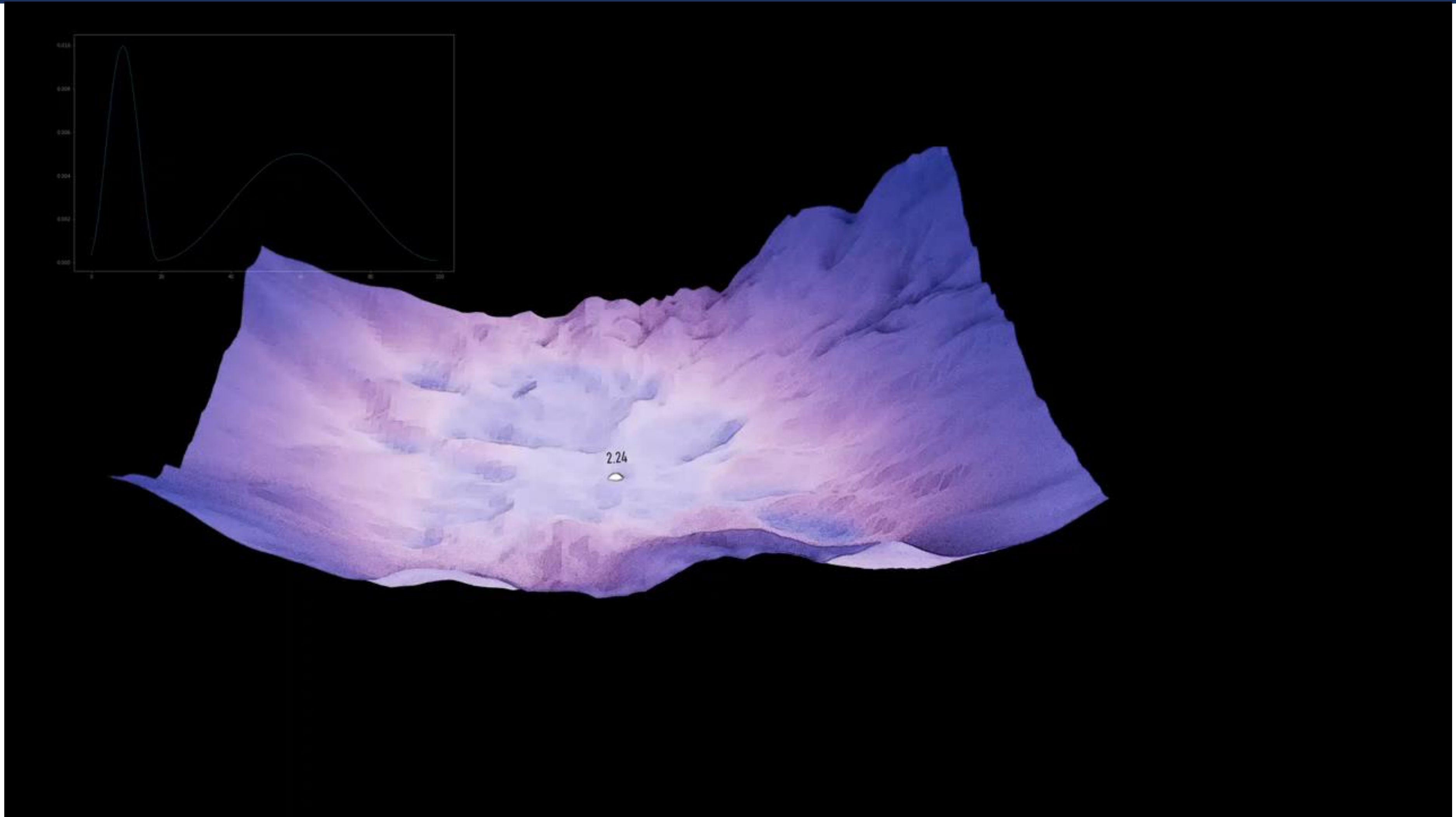
GRADIENT DESCENT OPTIMIZATION ALGORITHMS

- SGD with Momentum
- Nesterov Accelerated Gradient
- Adagrad
- Adadelta
- Adam
- Rectified Adam
- Lookahead

WHY IT MATTER ?



WHY IT MATTER ?



SGD + MOMENTUM

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

- We can think of momentum as **velocity** which is dampened at each step and perturbed by an external force field

NESTEROV ACCELERATION

$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

- Update our parameters w.r.t. the approximate future position of our parameters.

ADAGRAD

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

ADAGRAD

PROS

- No need to manually tune the learning rate
- Able to train on sparse data.
- Learning rate changes for each training parameter.

CONS

- Accumulation of the squared gradients in the denominator
- Computationally expensive as a need to calculate the second order derivative.
- The learning rate is always decreasing results in slow training.

ADADELTA

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

- The running average of the gradient
- The running average of the squared gradient
- The root mean squared error of parameter update
- The root mean squared error of the gradient
- The update rule

ADADELTA

PROS

- No need to set a default learning rate

CONS

- Computationally expensive

ADAM (ADAPTIVE MOMENT ESTIMATION)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

- Exponential moving average of the gradient
- Exponential moving average of the squared gradient

ADAM

PROS

- Converges rapidly.
- Rectifies vanishing learning rate, high variance\
- Prefers flat minima in the error surface

CONS

- Computationally costly.
- Initial training of Adam is unstable

RECTIFIED ADAM

Algorithm 2: Rectified Adam. All operations are element-wise.

Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\beta_1, \beta_2\}$: decay rate to calculate moving average and moving 2nd moment, θ_0: initial parameter, $f_t(\theta)$: stochastic objective function.

Output: θ_t: resulting parameters

$m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)

$\rho_\infty \leftarrow 2/(1-\beta_2) - 1$ (Compute the maximum length of the approximated SMA)

while $t = \{1, \cdots, T\}$ **do**

- $g_t \leftarrow \Delta_\theta f_t(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)
- $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ (Update exponential moving 2nd moment)
- $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ (Update exponential moving 1st moment)
- $\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$ (Compute bias-corrected moving average)
- $\rho_t \leftarrow \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$(Compute the length of the approximated SMA)
- **if** *the variance is tractable, i.e.,* $\rho_t > 4$ **then**
  - $\widehat{v_t} \leftarrow \sqrt{v_t/(1-\beta_2^t)}$ (Compute bias-corrected moving 2nd moment)
  - $r_t \leftarrow \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$ (Compute the variance rectification term)
  - $\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \widehat{m_t}/\widehat{v_t}$ (Update parameters with adaptive momentum)
- **else**
  - $\theta_t \leftarrow \theta_{t-1} - \alpha_t \widehat{m_t}$ (Update parameters with un-adapted momentum)

return θ_T

RECTIFIED ADAM

PROS

- Automated variance reduction
- Robust to learning rate variations

CONS

- Computationally expensive

LOOKAHEAD

Algorithm 1 Lookahead Optimizer:

Require: Initial parameters ϕ_0, objective function L
Require: Synchronization period k, slow weights step size α, optimizer A
for $t = 1, 2, \dots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \dots, k$ **do**
    sample minibatch of data $d \sim \mathcal{D}$
    $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
  end for
  Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

LOOKAHEAD

PROS

- Can be combined with any standard optimization method
- Improves convergence by reducing variance
- Lessens the need for extensive hyperparameter tuning

CONS

- Computationally expensive

CONCLUSION

Thank you for your attention