Deep Learning for Particle Physicists

Lewis Tunstall | AEC Graduate Seminar | May 6th 2022

🧹 Housekeeping

📅 **Timetable**
14:15-16:00 every Friday in Room 119 from April 29 to **June 10**

🤗 **Hugging Face Hub**

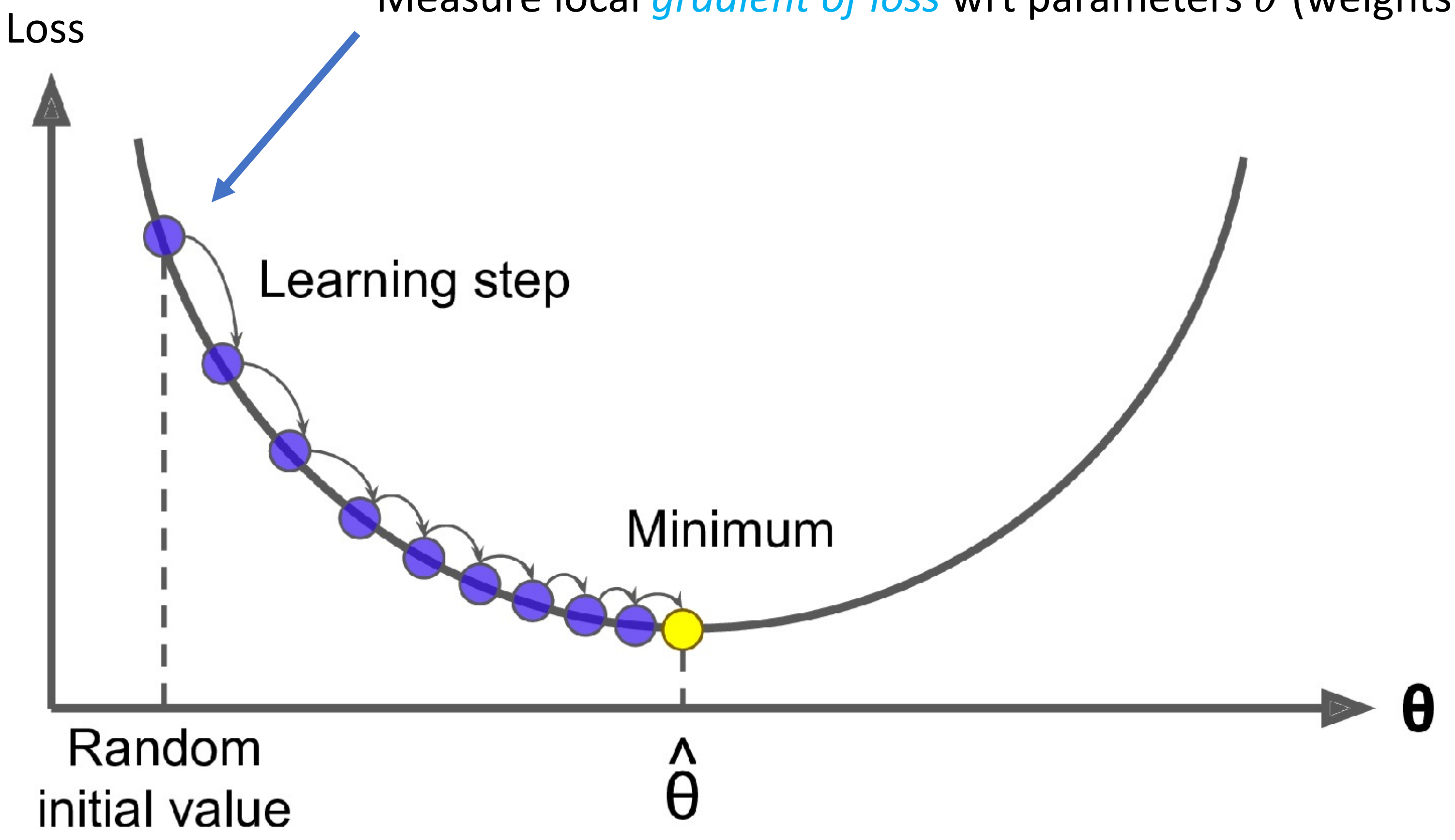
- https://huggingface.co/join
- https://huggingface.co/dl4phys (organisation for this class)

Training models

https://xkcd.com/1838/

Gradient descent

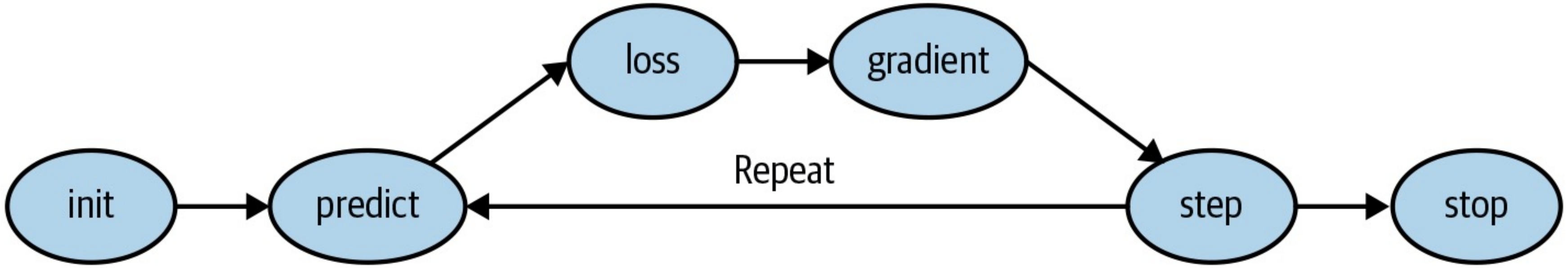
Measure local *gradient of loss* wrt parameters θ (weights & biases)



Hands-On Machine Learning with Scikit-Learn & TensorFlow, A. Geron

Basic idea: tweak parameters *iteratively* to minimise a loss function

Gradient descent



Deep Learning for Coders with fastai and PyTorch, J. Howard & S. Gugger

7 main steps for training ML models and deep NNs

Common loss functions

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

model prediction

linear regression

$$\mathrm{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

mean squared error

Common loss functions

parameter vector
(*weights* $\theta_1 \dots \theta_n$ and *bias* θ_0)

feature vector

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

model
prediction

linear regression

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

mean squared error

Common loss functions

parameter vector
(*weights* $\theta_1 \dots \theta_n$ and *bias* θ_0)

feature vector

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

model prediction

linear regression

$$\text{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

mean squared error

$$\widehat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^{\intercal} \mathbf{x}\right)$$

estimated probability

logistic regression

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} log\left(\widehat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \widehat{p}^{(i)}\right)\right]$$

binary cross entropy

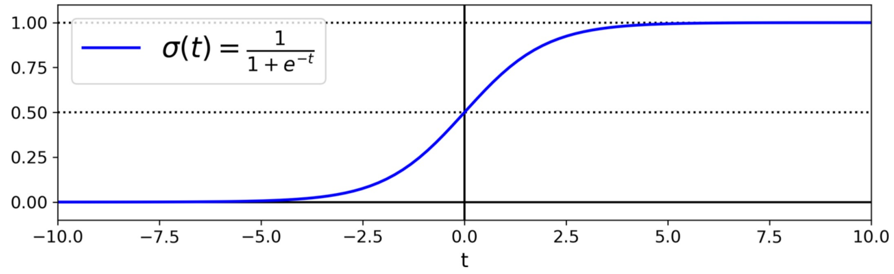


Figure 4-21. Logistic function

Common loss functions

parameter vector
(*weights* $\theta_1 \ldots \theta_n$ and *bias* θ_0)

feature vector

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

model prediction

linear regression

$$\mathrm{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

mean squared error

$$\widehat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}\right)$$

estimated probability

logistic regression

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} log\left(\widehat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \widehat{p}^{(i)}\right)\right]$$

binary cross entropy

generalises to *cross entropy* for $K > 2$ classes

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\widehat{p}_k^{(i)}\right)$$

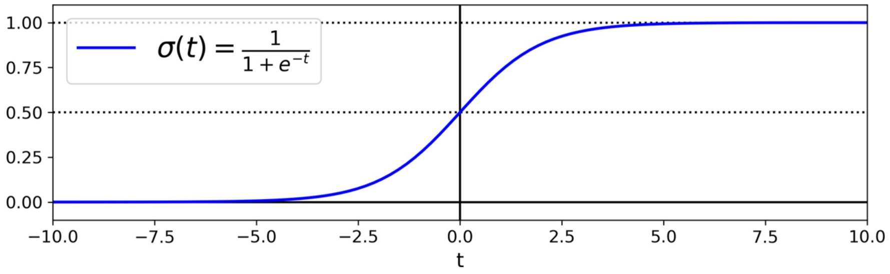
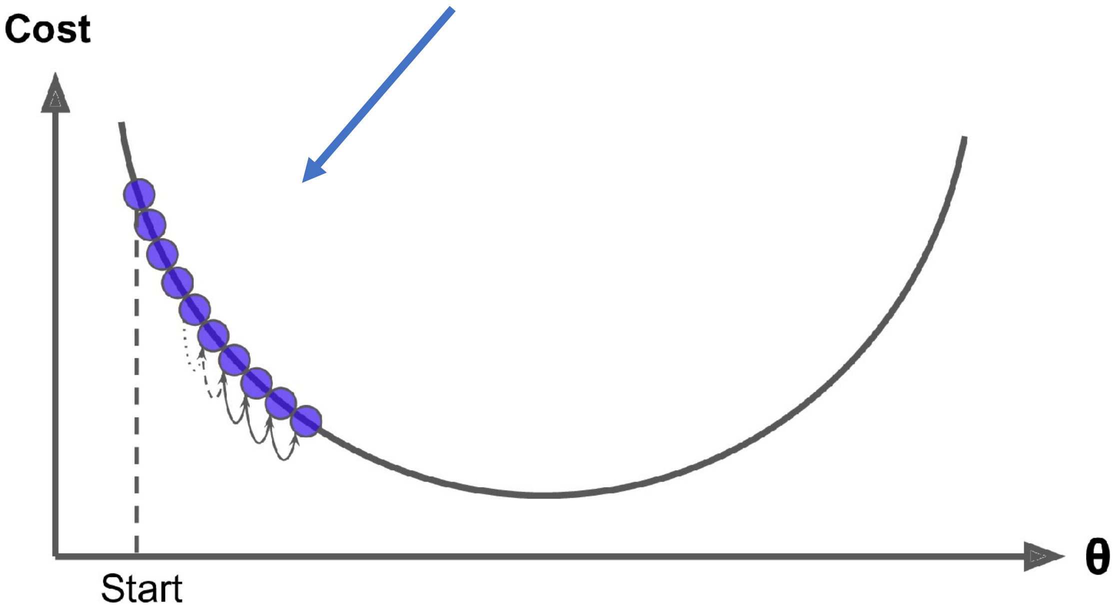


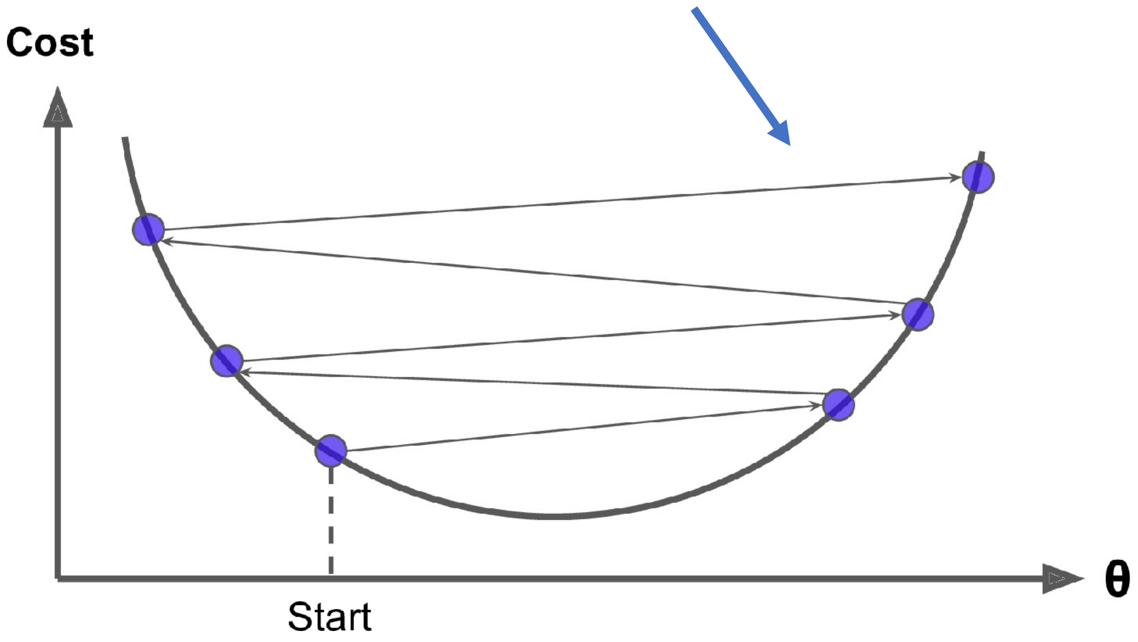
Figure 4-21. Logistic function

Gradient descent

Takes many steps to converge



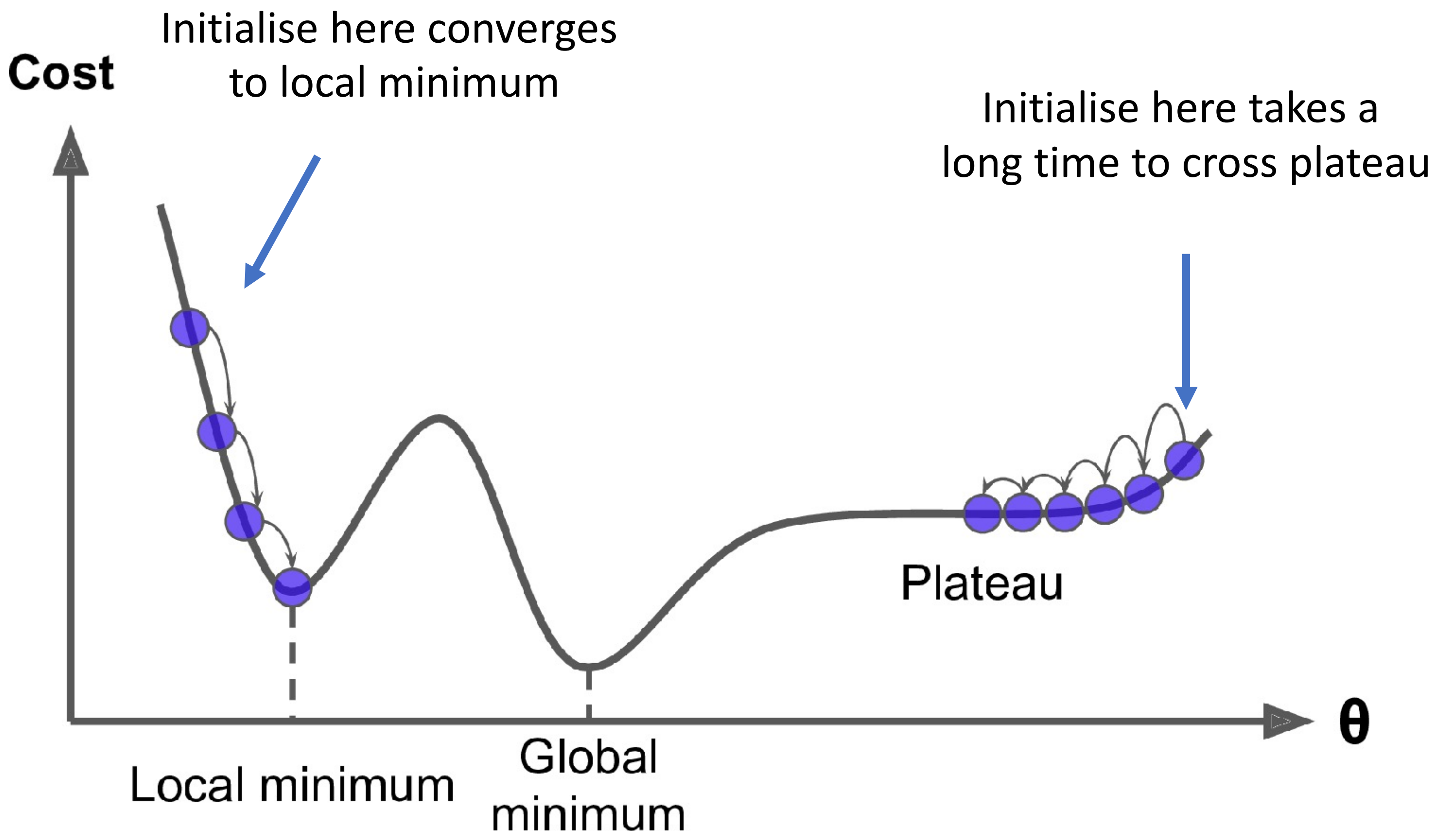
Overshoots the minimum & possibly diverges



Hands-On Machine Learning with Scikit-Learn & TensorFlow, A. Geron

Small vs *large* learning rates

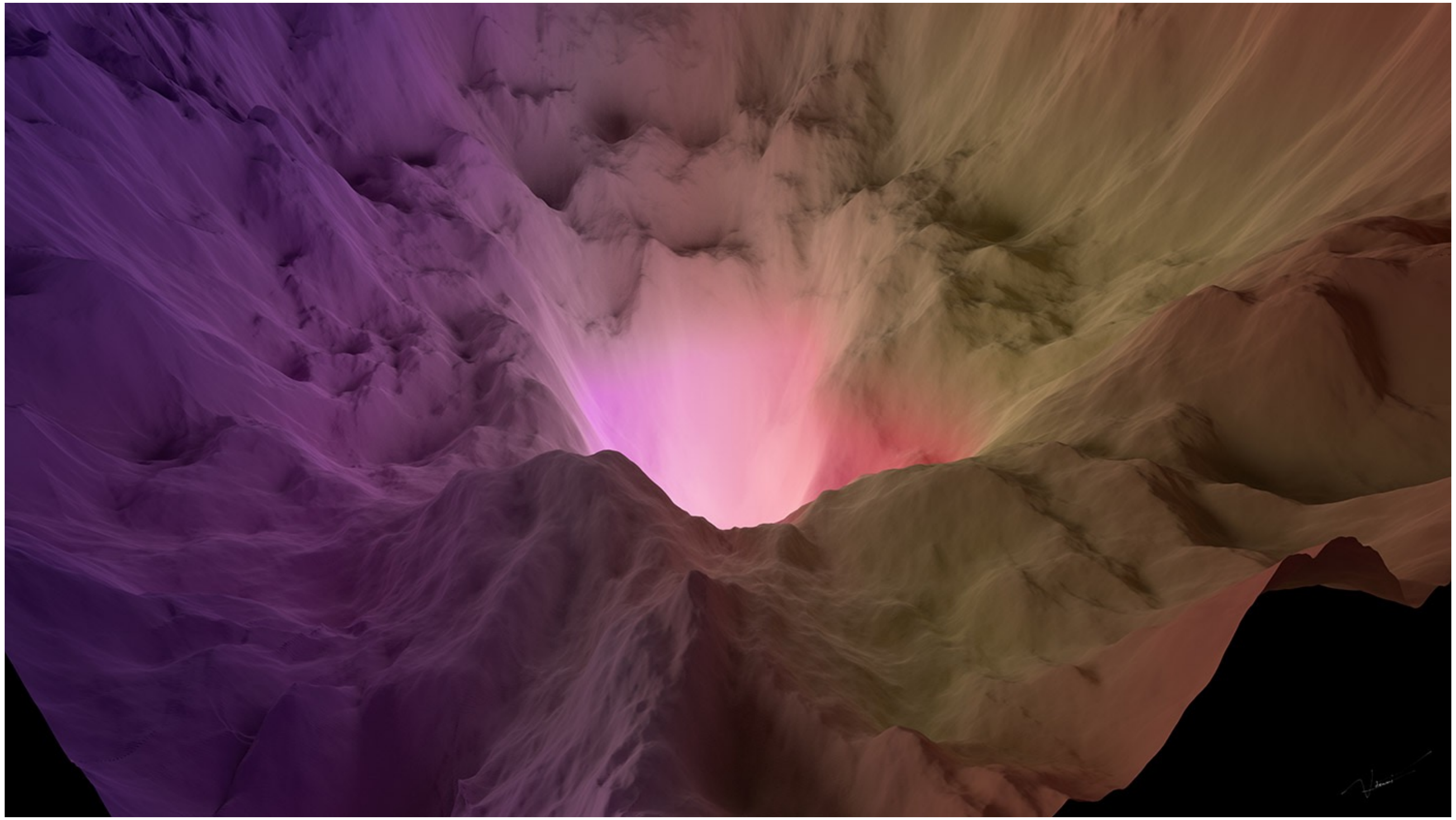
Gradient descent



Hands-On Machine Learning with Scikit-Learn & TensorFlow, A. Geron

Loss functions have multiple minima, plateaus, ridges etc

Gradient descent



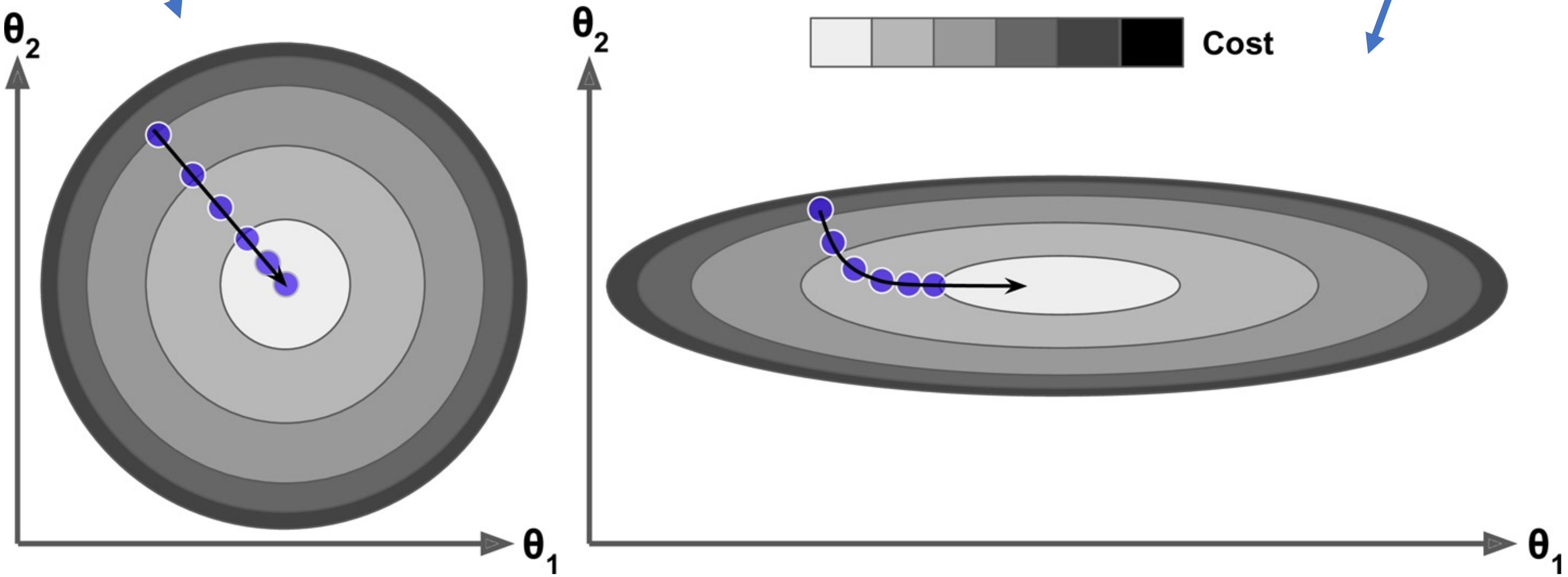
https://losslandscape.com/gallery/

Loss functions have multiple minima, plateaus, ridges etc

Gradient descent

Similar feature scales
⇒ quick to find minimum

Longer time to converge when
features scales are very different



Hands-On Machine Learning with Scikit-Learn & TensorFlow, A. Geron

Feature scaling

Batch gradient descent

$$\mathrm{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}^{(i)} - y^{(i)}\right)^{2}$$

the loss we wish to minimise

$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}\left(\boldsymbol{\theta}\right) = \frac{2}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

the gradients wrt each parameter

$$\nabla_{\boldsymbol{\theta}}\,\mathrm{MSE}\left(\boldsymbol{\theta}\right) = \begin{pmatrix} \frac{\partial}{\partial\theta_0}\mathrm{MSE}\left(\boldsymbol{\theta}\right) \\ \frac{\partial}{\partial\theta_1}\mathrm{MSE}\left(\boldsymbol{\theta}\right) \\ \vdots \\ \frac{\partial}{\partial\theta_n}\mathrm{MSE}\left(\boldsymbol{\theta}\right) \end{pmatrix} = \frac{2}{m}\mathbf{X}^{\intercal}\left(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right)$$

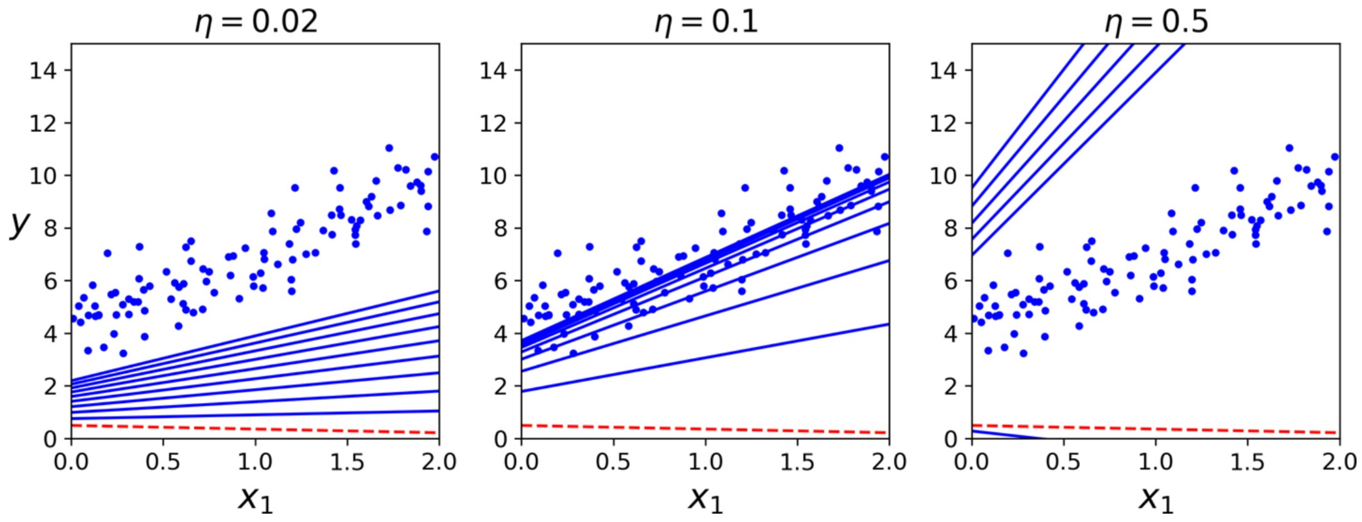
gradients as a *matrix product*
computed over full training set for each step
=> *batch gradient descent*

Calculating gradients for regression tasks

Batch gradient descent

define size of step via *learning rate*

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$$



η too small

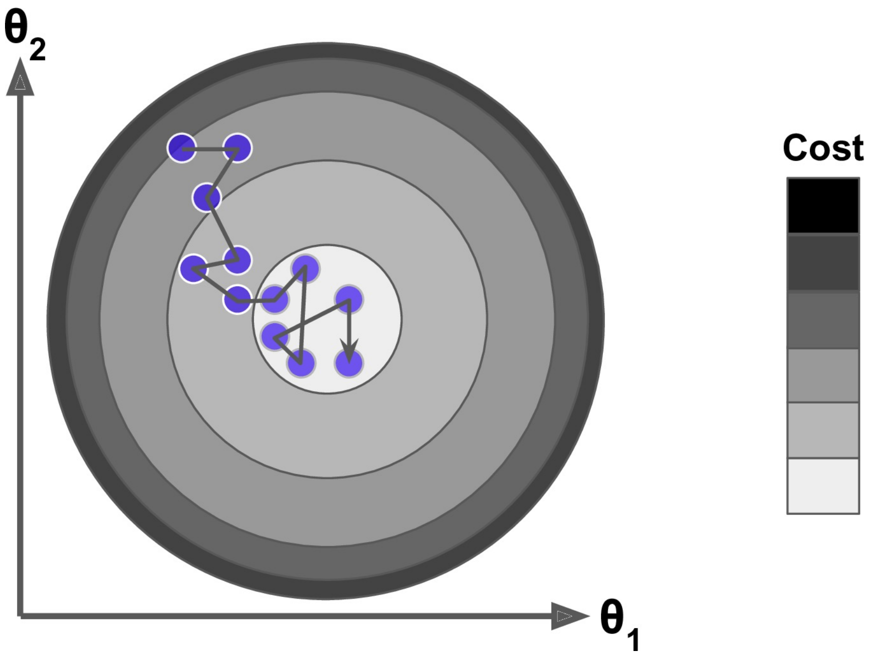
η just right

η too large

Hands-On Machine Learning with Scikit-Learn & TensorFlow, A. Geron

The gradient descent step

Stochastic gradient descent



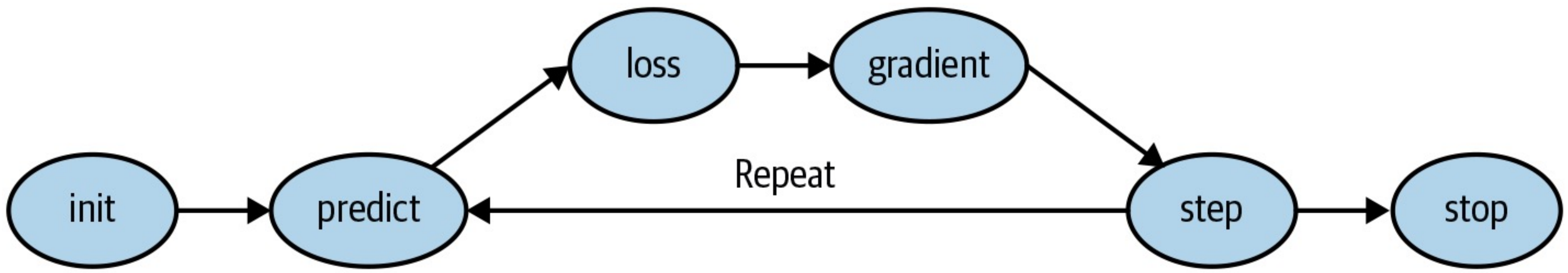
✅ much *faster* than batch gradient descent

✅ randomness allows algorithm to jump out of local minima

❌ randomness implies hard to settle in minimum
(use a *learning rate schedule* to decay learning rate)

Sample & compute gradients on *single instance*

Gradient descent summary



Deep Learning for Coders with fastai and PyTorch, J. Howard & S. Gugger