

4. Fully-Connected Deep Networks

4. Fully-Connected Deep Networks

- Choosing the right optimizer will speed up the efficiency of computation.

4.1 << Batch Gradient Descent >> : using the **entire** training dataset

$X = [x^{(1)} \ x^{(2)} \ x^{(3)} \ \dots \ x^{(m)}]$: *feature data*, $\dim = (n, m)$

$Y = [y^{(1)} \ y^{(2)} \ y^{(3)} \ \dots \ y^{(m)}]$: *target (labeled data)*, $\dim = (1, m)$

STEP 1 :

Forward propagation on X

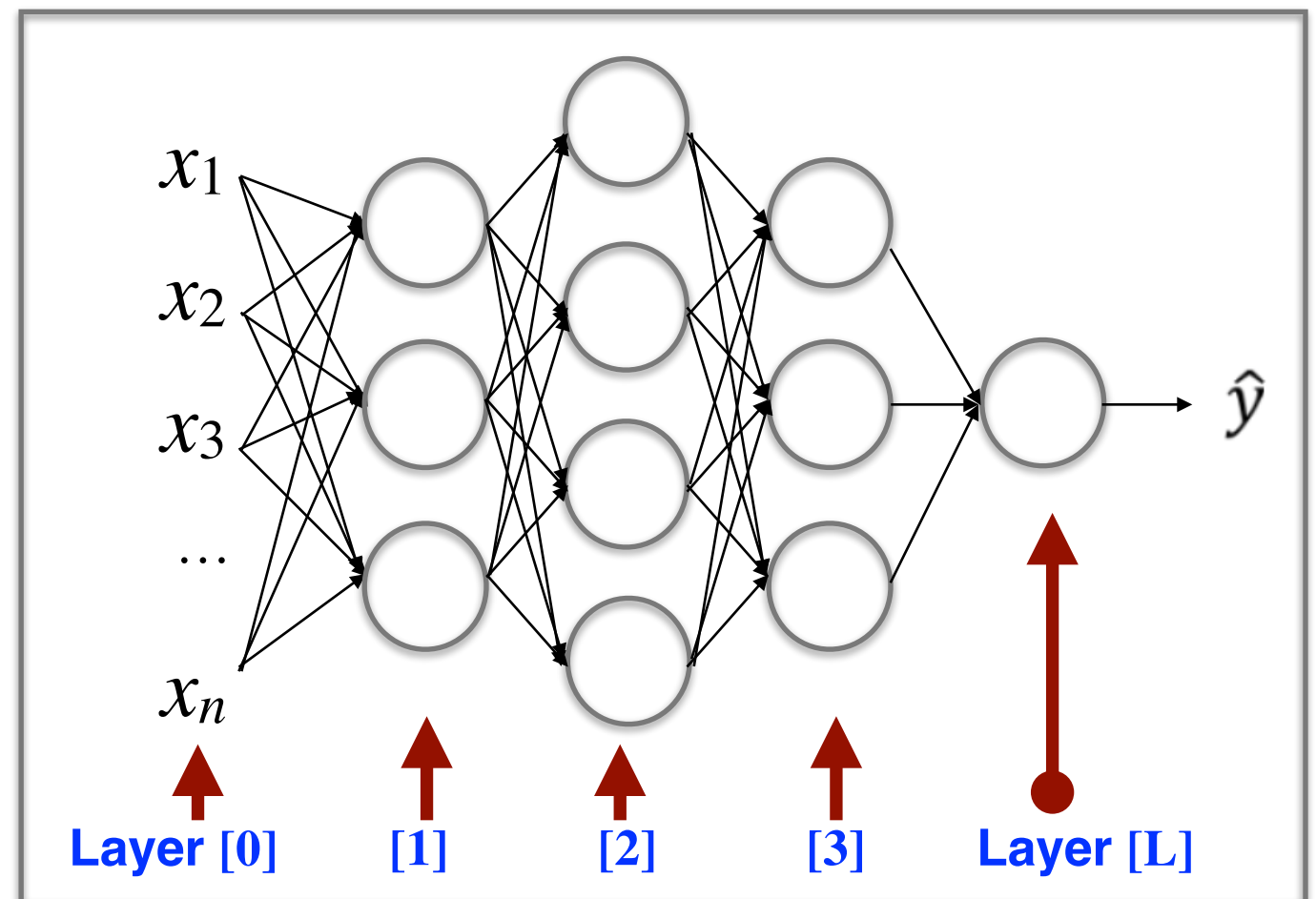
using **vectorization**:

$$\text{Layer [1]} \quad \begin{cases} z^{[1]} = w^{[1]} X + b^{[1]} \\ A^{[1]} = g(z^{[1]}) \end{cases}$$

$$\text{Layer [2]} \quad \begin{cases} z^{[2]} = w^{[2]} A^{[1]} + b^{[2]} \\ A^{[2]} = g(z^{[2]}) \end{cases}$$

... ..

$$\text{Layer [L]} \quad \begin{cases} z^{[L]} = w^{[L]} A^{[L-1]} + b^{[L]} \\ A^{[L]} = g(z^{[L]}) \end{cases}$$



4.1 << Batch Gradient Descent >>

(cont'd)

STEP 2 :

Compute the *cost* function, $J(w, b)$:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(y_{pred}^{(i)} - y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

STEP 3 :

Back propagation to compute **gradients** with respect to $J(w, b)$:

for all Layers $\Rightarrow \{ l = 1, 2, \dots, L$

$$\text{Layer } [l] \left\{ \begin{array}{l} w^{[l]} := w^{[l]} - \text{eta} * (dJ / dw)^{[l]} \\ b^{[l]} := b^{[l]} - \text{eta} * (dJ / db)^{[l]} \end{array} \right. \}$$

[NOTE] :

- “*d*” — partial derivative
- *eta* — learning rate

Ref : Andrew Ng, “15. **Vectorization**” <https://youtu.be/9YHWgxwzwD8>

One epoch computation *passing through the entire training dataset :*

{

STEP 1 :

Forward propagation on X using **vectorization**

STEP 2 :

Compute the *cost* function,
$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(y_{pred}^{(i)} - y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2$$

STEP 3 :

Back propagation to compute **gradients** with respect to $J(w, b)$

}

[NEXT] : Running a number of epochs till the approximation converged.

4.2 << Mini-Batch Gradient Descent >>

Q : But what if $m = 5,000,000$? (A huge training data size...)
Batch Gradient Descent Optimizer will slow down processing.

Q : But what if $m = 5,000,000$? (A huge training data size...)
Batch Gradient Descent Optimizer will slow down processing.

A: Splitting the entire training dataset into the “mini-batch” training datasets:

$$\mathbf{X} = [\underbrace{x^{(1)} \ x^{(2)} \ \dots \ x^{(1000)}}_{\mathbf{x}^{(1)}} \mid \underbrace{x^{(1001)} \ \dots \ x^{(2000)}}_{\mathbf{x}^{(2)}} \mid \dots \mid \dots \mid \underbrace{\dots \ x^{(m)}}_{\mathbf{x}^{(5000)}}] : \text{dim} = (n, 5,000,000)$$

5000 mini-batches

$$Y = [\underbrace{y^{(1)} \ y^{(2)} \ \dots \ y^{(1000)}}_{y^{\{1\}}} \mid y^{(1001)} \ \dots \ y^{(2000)} \mid \dots \mid \dots \mid y^{(m)}] : \dim = (1, 5,000,000)$$

$y^{\{1\}} \qquad y^{\{2\}} \qquad \dots \dots \ y^{\{5000\}} \qquad 5000 \text{ mini-batches}$

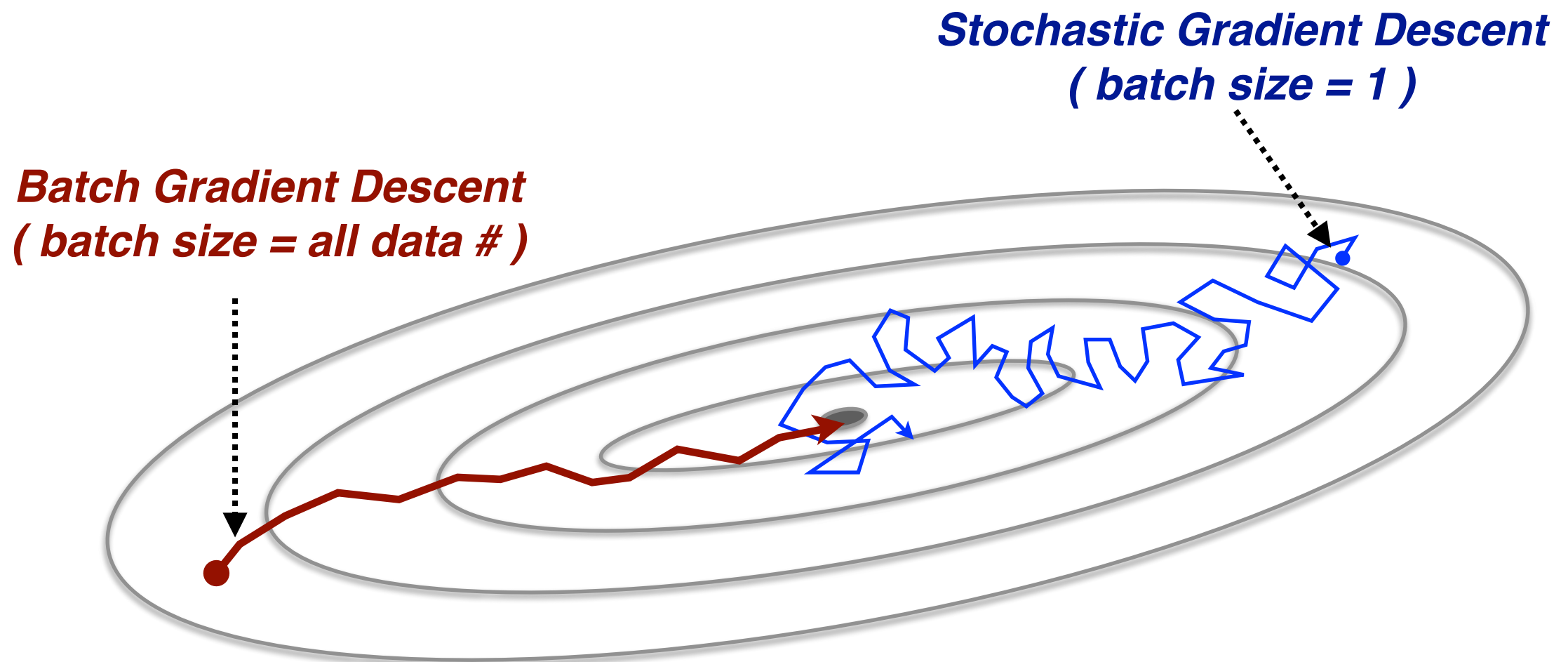
=> 5000 mini-batches with *1000 samples/ mini-batch*

=> “Mini-Batch Gradient Descent” Optimizer

4.2 << Mini-Batch Gradient Descent >>

(cont'd)

Mini-Batch Gradient Descent \Rightarrow *batch size = 64, 128, 256 or 512*



Stochastic vs. Batch Gradient Descent

4.3 Cost Functions for Logistic Regression

(Ref : “Cross entropy” from Wikipedia, https://en.wikipedia.org/wiki/Cross_entropy)

Binary Cross Entropy — for Binary Classifier

Loss Function : $p \in \{y, 1 - y\}$ and $q \in \{\hat{y}, 1 - \hat{y}\}$

$$L(p, q) = - \sum_i p_i \log q_i = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Binary Classifier : $y = 0$ or 1 (*Sigmoid Function*)

$$\begin{cases} y = 0 : & L = -\log(1 - \hat{y}) \\ y = 1 : & L = -\log \hat{y} \end{cases}$$

4.3 Cost Functions for Logistic Regression

(cont'd)

(Ref : “Cross entropy” from Wikipedia, https://en.wikipedia.org/wiki/Cross_entropy)

Categorical Cross Entropy — for Multiple-Output Classifier

Cost Function :

$$J(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N L(p_n, q_n) = -\frac{1}{N} \sum_{n=1}^N \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

Multiple-Output Classifier : *Activation Function — Softmax*

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

(Ref. “Softmax” from Wikipedia, <https://zh.wikipedia.org/wiki/Softmax%E5%87%BD%E6%95%B0>)

4.4 Activation Functions

- Sigmoid Function
- **tanh** Function
- ReLU Function (Rectifier Linear Unit)
- Leaky ReLU
- Softmax

4.5 Faster Gradient Descent Algorithms for Optimization

[Concept] : ***Exponentially Weighted Moving Average***

— <https://youtu.be/lAq96T8FkTw>

- < Optimizer > : **Gradient Descent with Momentum**

https://youtu.be/k8fTYJPd3_I

- < Optimizer > : **RMSProp Optimization Algorithm**

https://youtu.be/_e-LFe_igno

- < Optimizer > : **Adam Optimization Algorithm**

https://youtu.be/JXQT_vxqwls

4.6 Hyperparameter Tuning Process

- learning rate (α)
- momentum term (β)
- numbers of hidden layers
- numbers of hidden units (i.e., neurons)
- learning-rate decay
- mini-batch size
-

[Tuning Rules for Hyperparameters] :

- **Try random values for hyperparameters** : *Don't use a grid.*
- **Coarse to Fine** (<https://youtu.be/AXDBByU3D1hA>)
- **Using an Appropriate Scale** (https://youtu.be/cSoK_6Rkbfg)

Advanced Topics

Batch Normalization & Swish Activation

Batch Normalization

- 在 **Neural Networks** 開始訓練之前，通常會對輸入資料進行 **normalization** (正規化) 的前置處理；原因是這樣一來，將會提升 **Neural Networks** 的訓練效能。
- 因此，如果能夠針對 **Deep Neural Networks** 的各個隱藏層 (**hidden layers**) 的輸入資料進行 **batch normalization** (批次正規化) 處理，除了增加效能之外，甚至於無須擔心 **overfitting** 的問題 (亦即，無須進行 **dropout** 或 **regularization** 修正)。

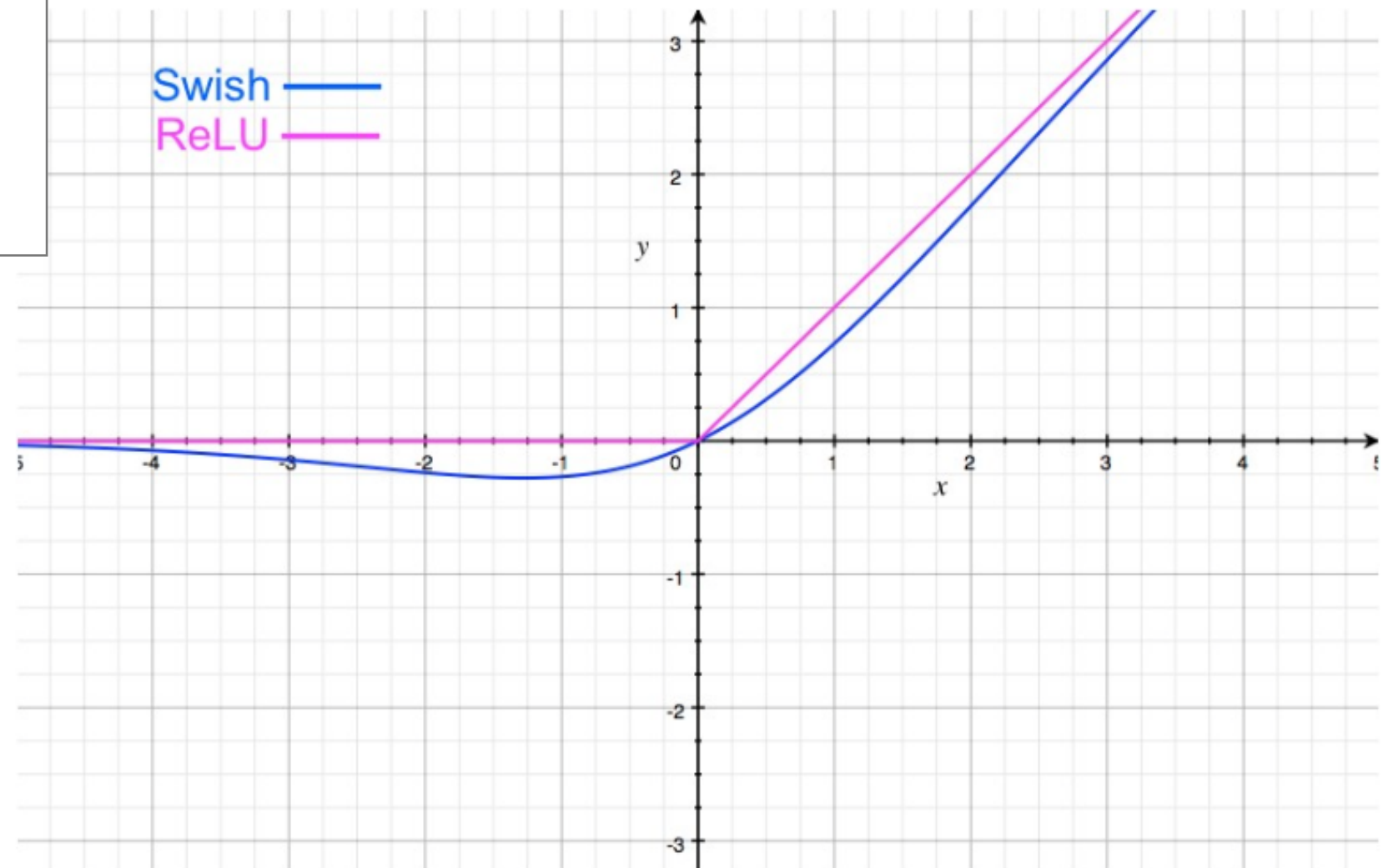
[REFERENCE]

1. “**Fitting Batch Norm Into Neural Networks (C2W3L05)**” <https://youtu.be/nUUqwaxLnWs>
2. “**Why Does Batch Norm Work? (C2W3L06)**” <https://youtu.be/nUUqwaxLnWs>
3. “**Batch Normalization**” : <http://violin-tao.blogspot.com/2018/02/ml-batch-normalization.html>
4. “**Advanced Tips for Deep Learning,**” **Hung-yi Lee** — https://www.csie.ntu.edu.tw/~yvchen/f106-adl/doc/171116+171120_Tip.pdf

Swish Activation

[Activation Functions] :

1. Sigmoid Function
2. Relu Function
3. tanh Function
4. Softmax Function
5. **Swish Function**



Ref : “**Experiments with SWISH activation function on MNIST dataset**”

<https://medium.com/@jaiyamsharma/experiments-with-swish-activation-function-on-mnist-dataset-fc89a8c79ff7>