3. Fully-Connected Deep Networks

3. Fully-Connected Deep Networks

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

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

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

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

STEP 1 :

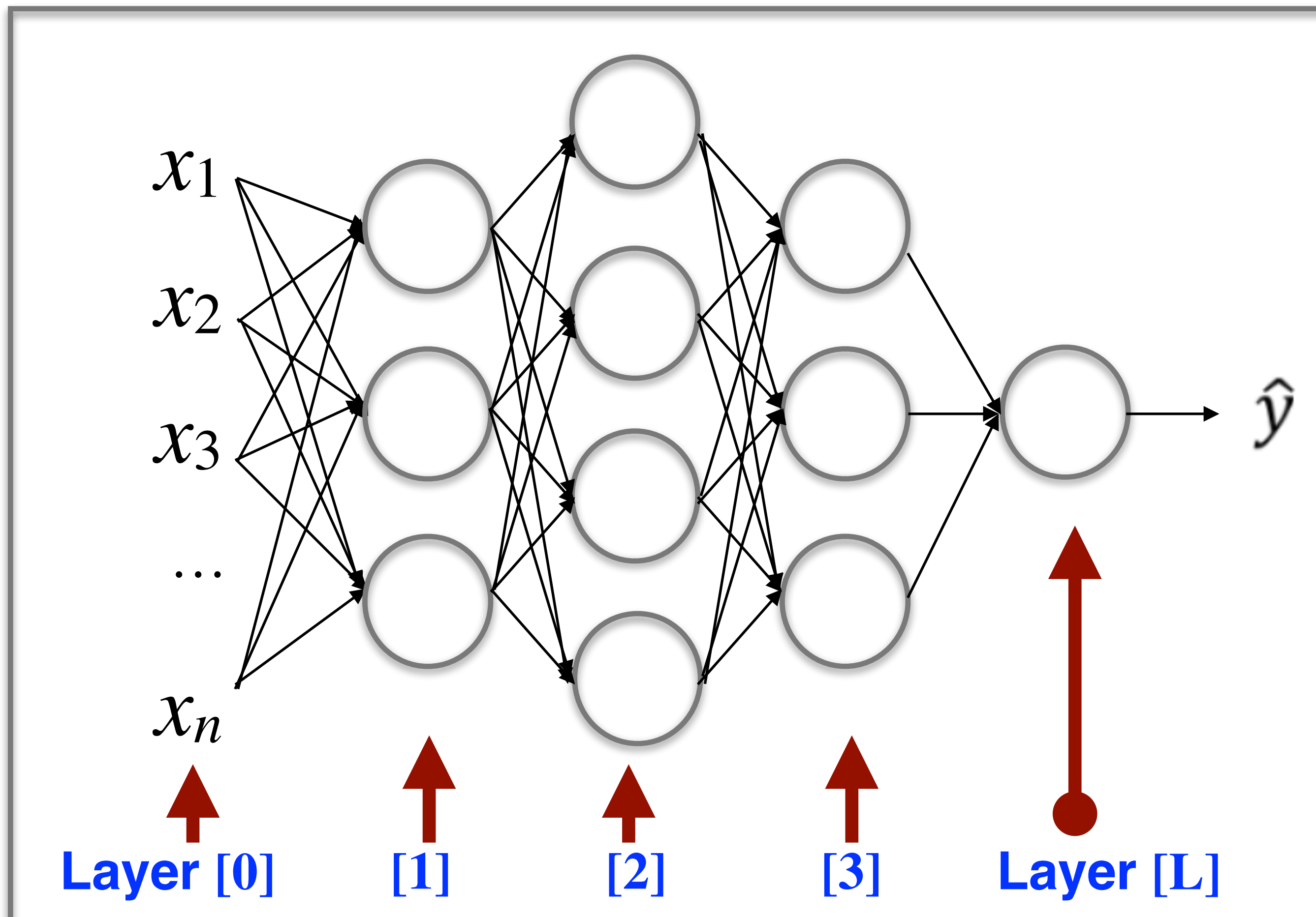
Forward propagation on X using **vectorization**:

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

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

$\cdots \cdots \cdots$

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



3.1 << Batch Gradient Descent >> (cont'd)

STEP 2 :

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

Residual Sum of Squares (RSS)

Frobenius Regularization *for Neural Network*

What for?

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

STEP 3 :

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

for all Layers => { $l = 1, 2, \ldots, L$

Layer [l]

$$\begin{cases} w^{[l]} := w^{[l]} - eta * (dJ / dw)^{[l]} \\ b^{[l]} := b^{[l]} - eta * (dJ / db)^{[l]} \end{cases}$$

}

[NOTE] :

- " ***d*** " — partial derivative
- *eta* — *learning rate*

Ref : Andrew Ng, "15. Vectorization" https://youtu.be/9YHWgxwzwD8

3.1 << Batch Gradient Descent >> (cont'd)

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

{

STEP 1 :

Forward propagation on X using **vectorization**

STEP 2 :

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

STEP 3 :

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

}

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

3.2 << 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.

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

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

5000 mini-batches

$$Y = [\underbrace{\boxed{y^{(1)}\ y^{(2)} \cdots y^{(1000)}}}_{y^{\{1\}}} \mid \underbrace{y^{(1001)} \cdots y^{(2000)}}_{y^{\{2\}}} \mid \cdots \cdots \mid \underbrace{\cdots\ y^{(m)}}_{y^{\{5000\}}}] \ : \ \text{dim} = (1, 5{,}000{,}000)$$

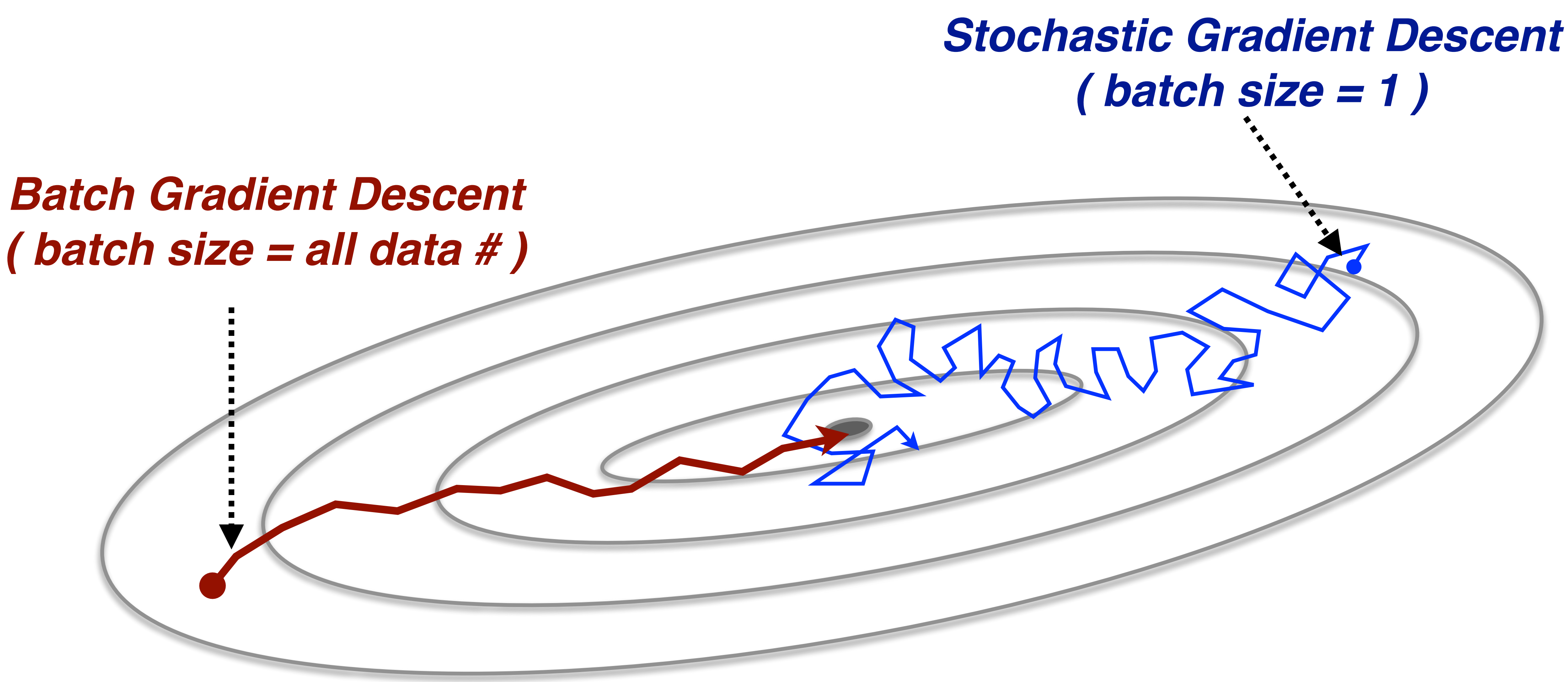
5000 mini-batches

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

=> **"Mini-Batch Gradient Descent" Optimizer**

3.2 << Mini-Batch Gradient Descent >> *(cont'd)*

Mini-Batch Gradient Descent => batch size = 64, 128, 256 or 512



Stochastic vs. Batch Gradient Descent

3.3 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 $q \in \{\hat{y}, 1-\hat{y}\}$

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

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

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

3.3 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***

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

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

3.4 Activation Functions

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

3.5 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_vxqwIs

3.6 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/AXDByU3D1hA)
- **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)。

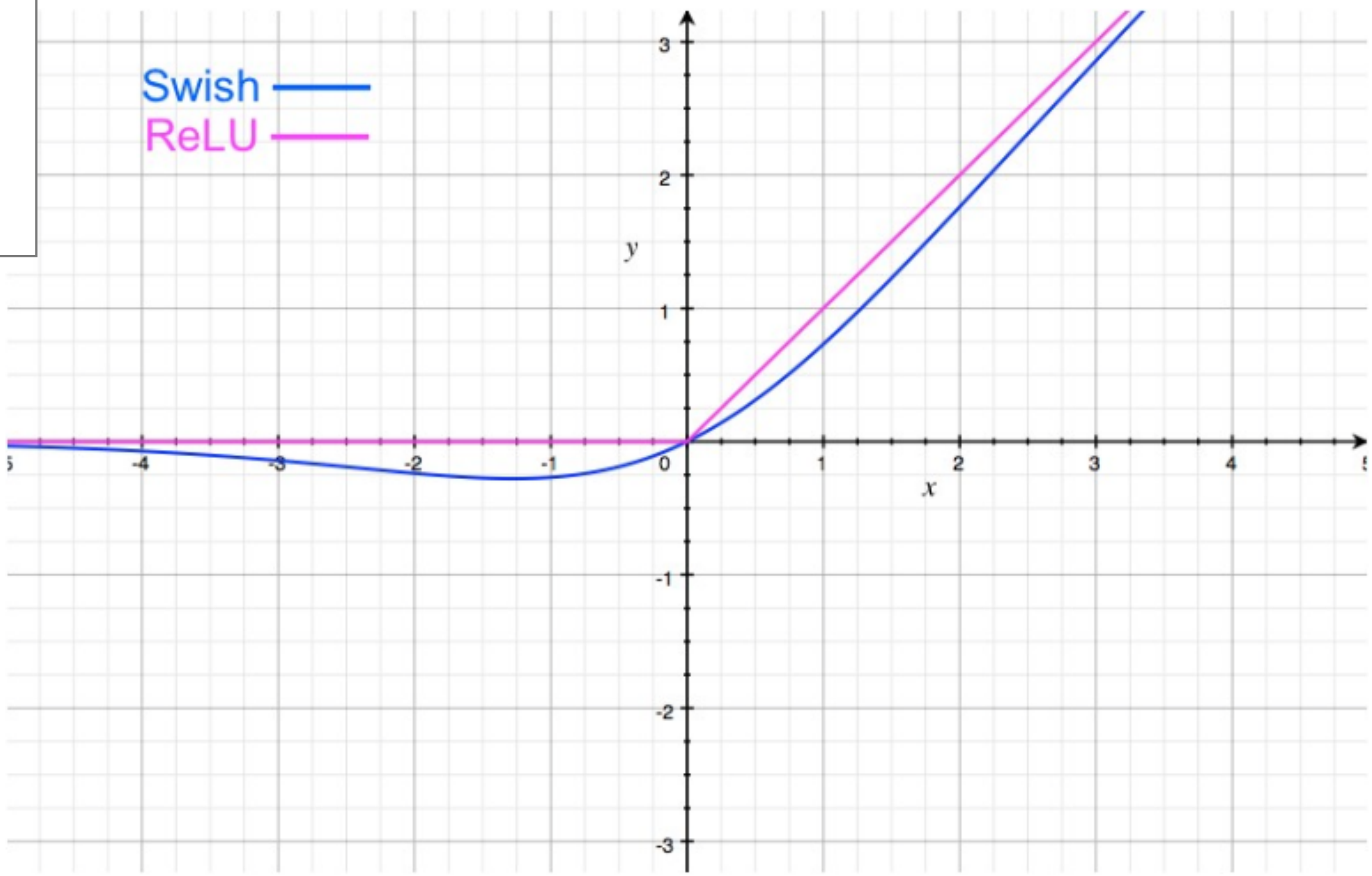
[REFERENCE]

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