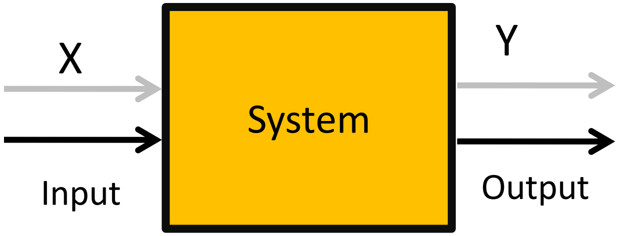
Machine Learning

Than Lwin Aung

Static Systems



Dynamic Systems

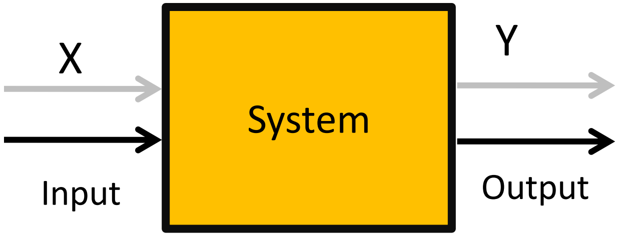
In Static Systems, Output (Y) is only dependent on the Linear or Non-Linear Combinations of Inputs (X).

There is “No State” in Static Systems.

Therefore, in Static Systems, there are only **one** major Equations:

1) Output Equations

Linear Static Systems



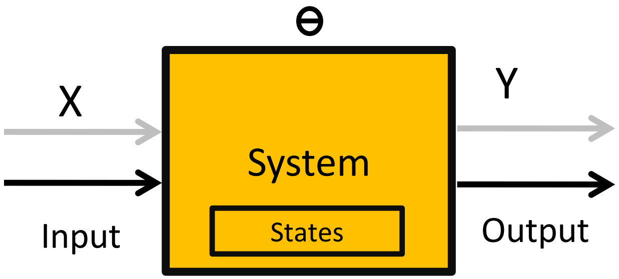
Dynamic Systems

In Linear Static Systems, Output (Y) is only dependent on the Linear Combinations of Inputs (X).

Output Equations :

$\mathbf{Y_n = a_nX_n}$

Dynamic Systems



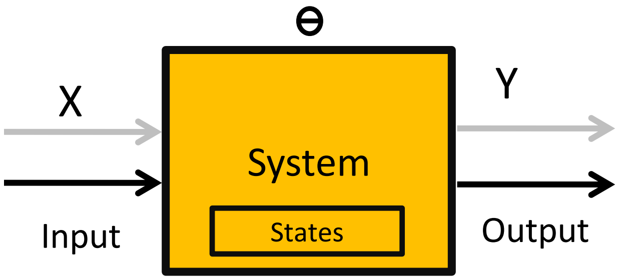
Dynamic Systems

In Dynamic Systems, Output (Y) is not only dependent on the Linear or Non-Linear Combinations of Inputs (X), but also dependent on the Linear nor Non-Linear Combinations of States (ϴ).

In Dynamical Systems, there are **two** major equations:

1) State Equations
2) Output Equations

Linear Dynamic Systems



Dynamic Systems

In Linear Dynamic Systems, Output (Y) is the Linear Combinations of Input (X) and States (ϴ).

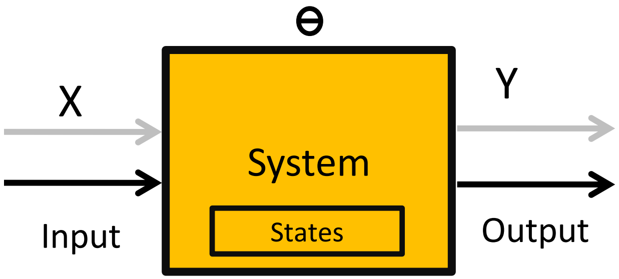
Output Equations :

$\mathbf{Y_n = X_n + \Theta_n + \Theta_{n-1} + \ldots + \Theta_0}$

State Equations :

$\mathbf{\Theta_{n+1} = \Theta_n + \Theta_{n-1} + \ldots + \Theta_0 + X_n}$

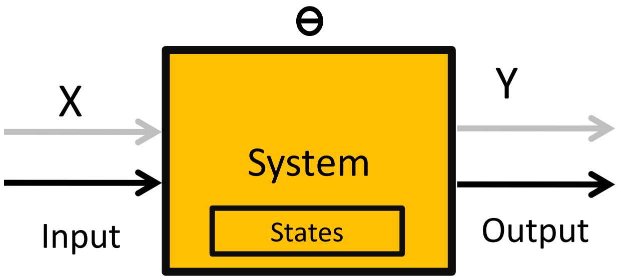
Non Linear Dynamic Systems



Dynamic Systems

In Non-Linear Dynamic Systems, Output (Y) is the Non-Linear Combinations of Input (X) and States (ϴ).

Analytical Approach



Dynamic Systems

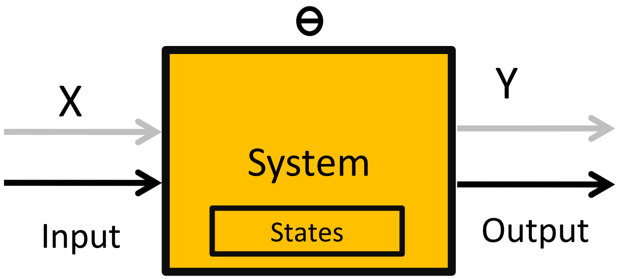
Traditionally, Linear Systems are studied based on Analytical Approaches:

1) Time Domain
2) Frequency Domain

The main purpose of the study is to figure out the Parameters (ϴ) of the Systems, given Input (X).

Generally, Linear Dynamical Systems are modeled with Differential Equations or Difference Equations.

Statistical Approach



Dynamic Systems

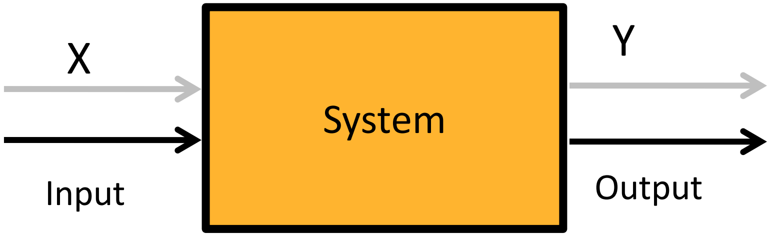
However, Analytical Approaches cannot be used for all Systems, especially for Non-Linear Systems.

Therefore, Statistical Approaches have become more useful if it is not possible to study the systems analytically.

Statistical Approaches are based on:

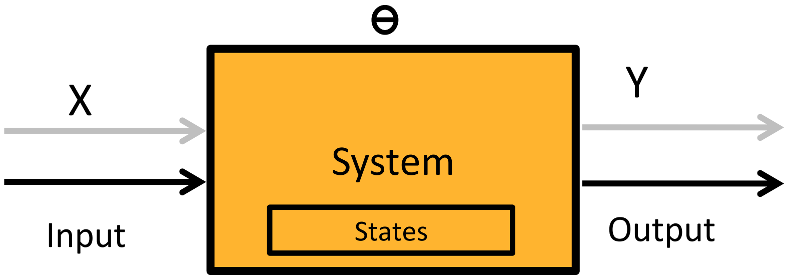
1) Probability Function
2) Likelihood Function

Systems Approach



Static Systems

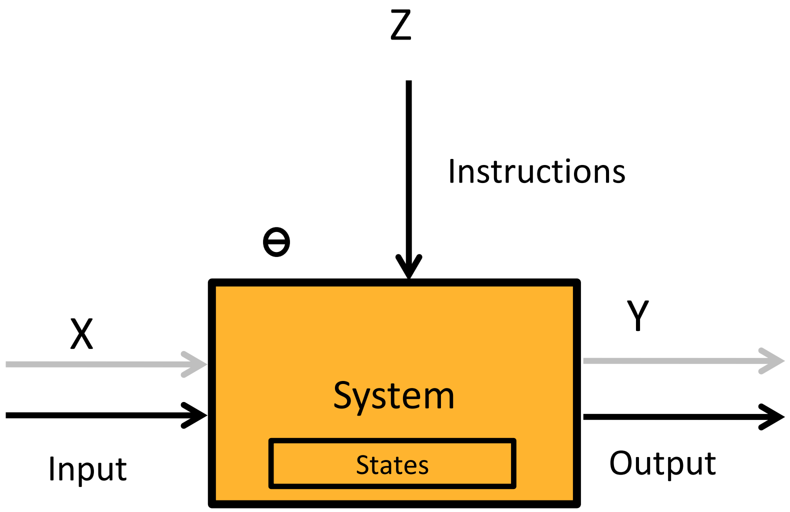
$Y = F(X)$



Dynamic Systems

$Y = F(X, \Theta)$

Programmability

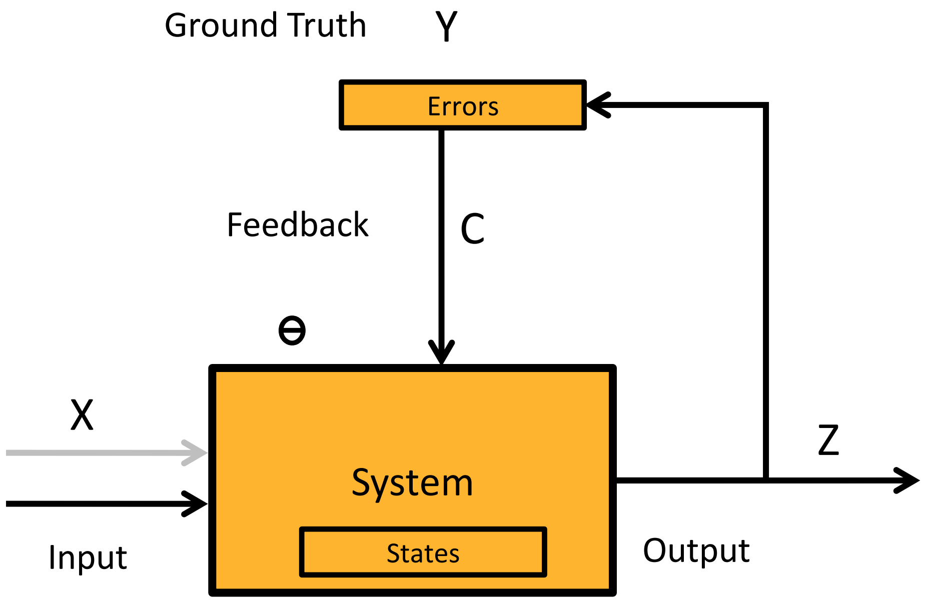


Programmable Dynamic Systems

$$\Theta = H(X, Z)$$

$$Y = F(X, \Theta)$$

Learnability



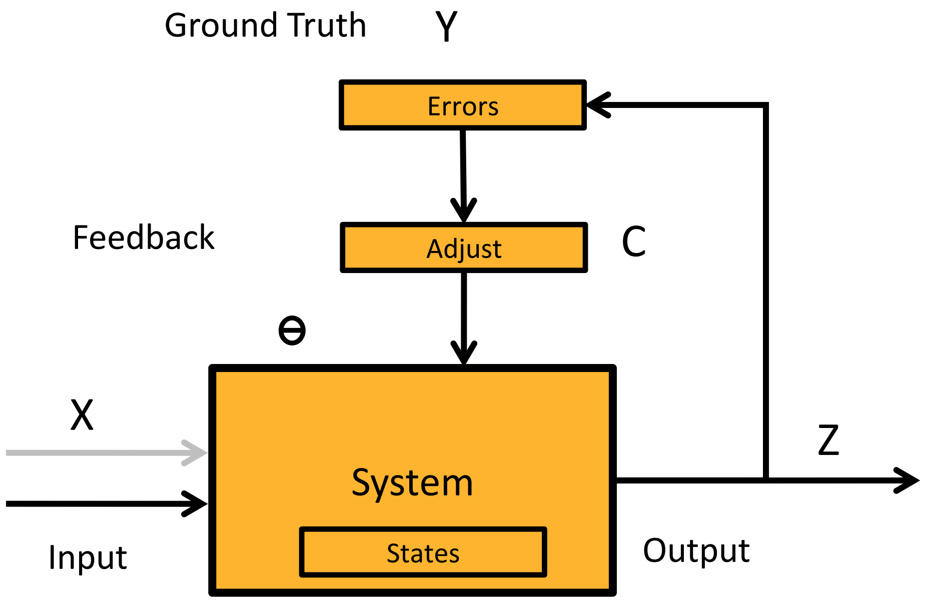
Learnable Dynamic Systems

Z = F(X, ϴ)

C = L(Z, Y)

ϴ = H(X, C)

Evolvability?



Ground Truth Y

Errors

Feedback

Adjust

C

ϴ

X

System

States

Input

Output

Z

Learnable Dynamic Systems

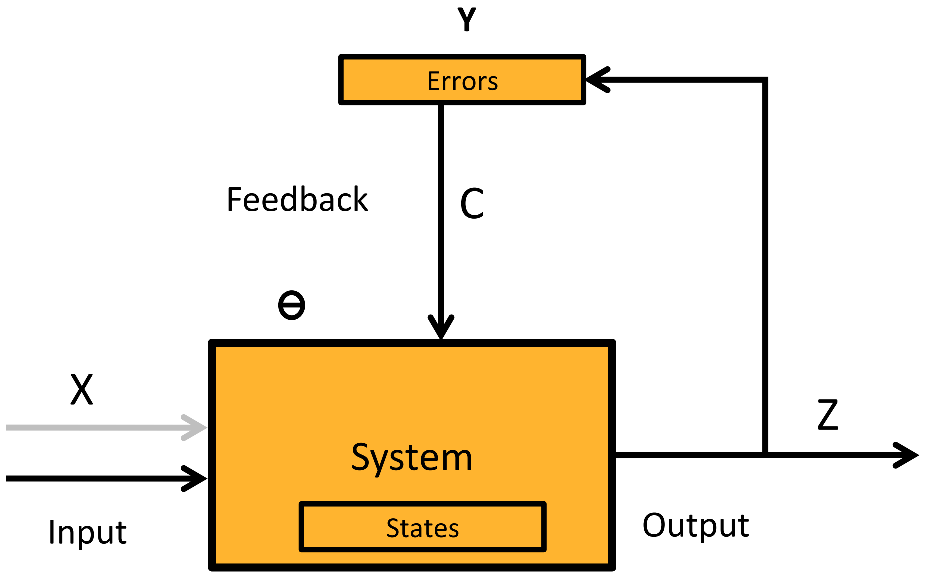
$$Z = F(X, \Theta)$$

$$C = L(Z, Y)$$

$$\Theta = H(X, C)$$

In Learnable Systems, the Cost Function is predetermined; however, if Cost Function itself is adaptable to the Environment or Directives, it could possibly evolve.

Statistical Approach



In Statistical Approach, it is impossible to directly represent the Output (Y) as an Equation of Input (X) and States (**Θ**), as we do not know the exact relationship between X,Y and **Θ**. Instead, Output Probability Distribution is defined by:

Z = P(X, Θ), Joint Probability Distribution of X and Θ

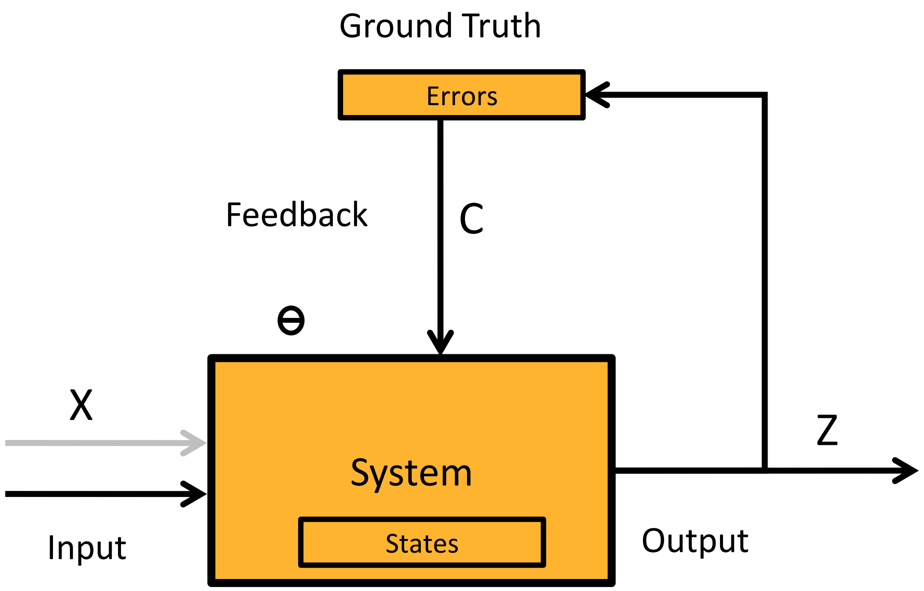
As Y is only the subset of All Possible Output, Conditional Probability Function of Y is also defined by:

***P* (Y | X, Θ)** However, it is difficult to directly find P(Y | X, Θ), therefore, it is calculated from

L((X,Y) | Θ) Log Likelihood Function of (Y,X) given Θ, as

***P*((Y , X)| Θ)) = P(X | Θ)P(Y | X, Θ), normally P(X | Θ) is ignored**

Learning Approach

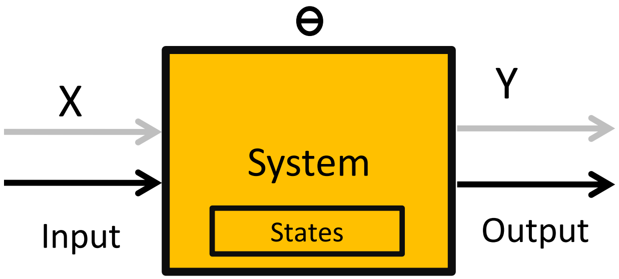


Output Equation: $Z = H(X, \theta)$

Cost Equation: $C = L(Z, Y)$

State Equation: $\theta_{n+1} = \theta_n - r\nabla C$

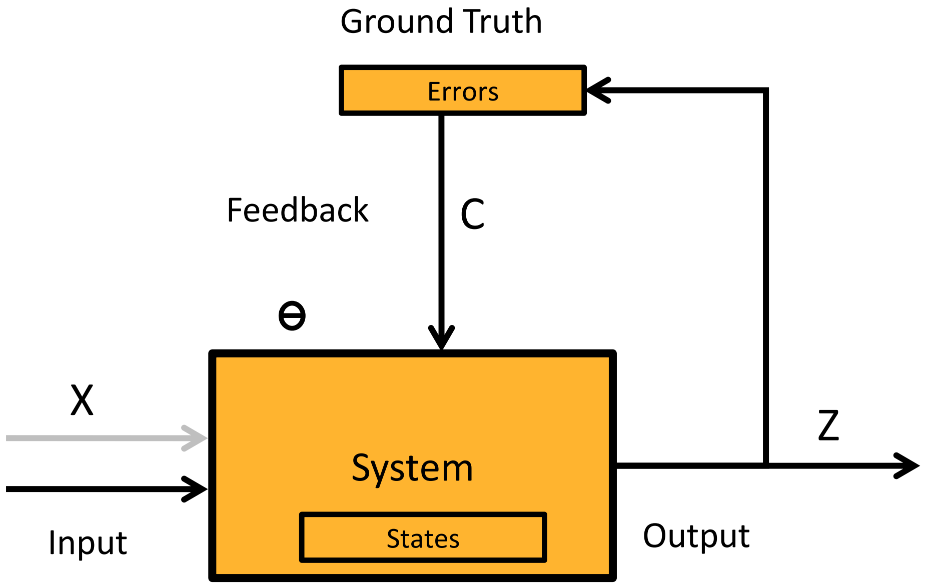
Statistical Definition of Systems



Dynamic Systems

PROBABILITY DISTRIBUTION	SUPERVISED LEARNING	UNSUPERVISED LEARNING
P(X)		FIND
P(Y)	FIND	
P(ϴ)		
P(X,Y)		
P((X,Y) | ϴ)	OPTIMIZE	
P(Y | ϴ)		
P(X | ϴ)		
P(ϴ|(X,Y))		OPTIMIZE

Learning Approach



There are 3 types of Learning:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforced Learning

The main purpose of Learning Approach is to learn Parameters (States) of System with Statistical Analytic Approach.

Supervised Learning

In Supervised Learning, there are 3 major equations:

1) Output Equations
2) Cost Equations
3) State Equations

Output Equations are normally Activation Function :

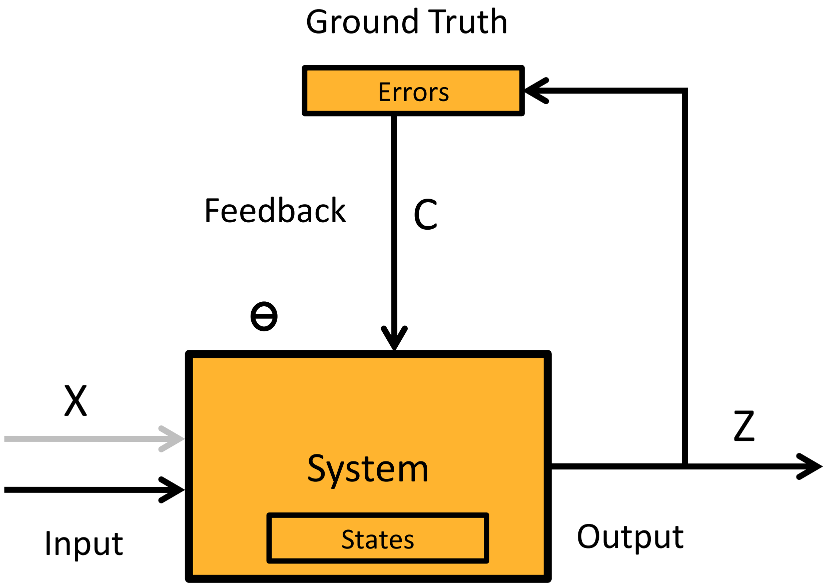
Y = *F*(ϴX + b)

Cost Equation is normally Log Loss or Error Function:

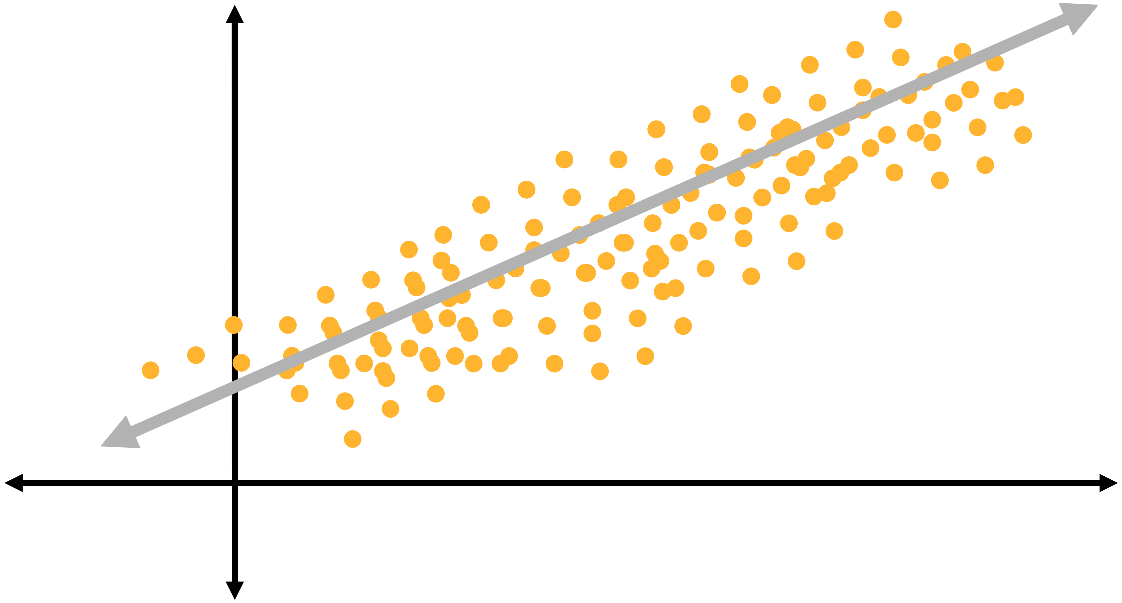
C = -*L*(Z,Y) or *E*(Z-Y)

State Equation is normally Gradient Descent Function:

$$\Theta_{n+1} = \Theta_n - r \nabla C$$



Linear Regression



Output Equation: $Z = H(X, \Theta) = \Theta X + b$ ų σ

Cost Equation: $C = L(Z, Y) = -\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}e^{\left(-1\frac{1}{2\sigma^2}\ (Y\text{-}Z)\right)^2}\right]$

State Equation: $\Theta_{n+1} = \Theta_n - r\nabla C$

Linear Regression – Normal

In Linear Regression Model,

Output Equations are Linear Function and Log Likelihood Function with Normal Distribution:

$P(X, \Theta) = \Theta X + b$

$$\mathbf{L(Y \mid X, \Theta)} \ = \mathbf{log} \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{(-1\frac{1}{2\sigma^2}(y-P)^2} \right]$$

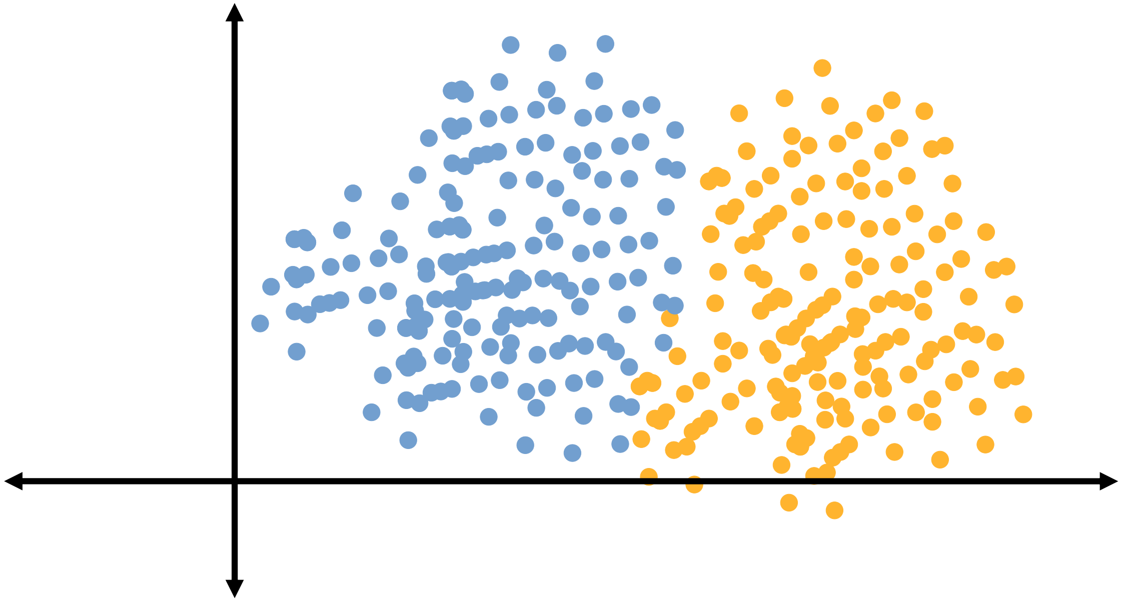
Cost Equation is normally Log Likelihood Function:

$$C = \mathbf{-L(Y \mid X, \Theta)} \ = - \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\ (y-P)^2$$

State Equation is normally Learning (Gradient Descent) Function:

$\Theta_{n+1} = \Theta_n - r \nabla C$

Logistic Regression (Binary Classification)



Output Equation: $Z = H(X, \Theta) = \dfrac{1}{1 + e^{(-\theta x + b)}}$

Cost Equation: $C = L(Z, Y) = -\log\left[Z^{y}\,(1 - Z)^{(1-y)}\right]$

State Equation: $\Theta_{n+1} = \Theta_{n} - r\nabla C$

Logistic (Sigmoid) Regression – Bernoulli

In Binary Classification Model,

Output Equations are Sigmoid Function and Log Likelihood Function with Bernoulli Distribution :

P(X, Ө) = $\frac{1}{1+\ e^{(-\theta x+b)}}$

L(Y | X, Ө) = log $\left[P^{y}\ (1-P)^{(1-y)}\right]$

Cost Equation is normally Log Likelihood Function:

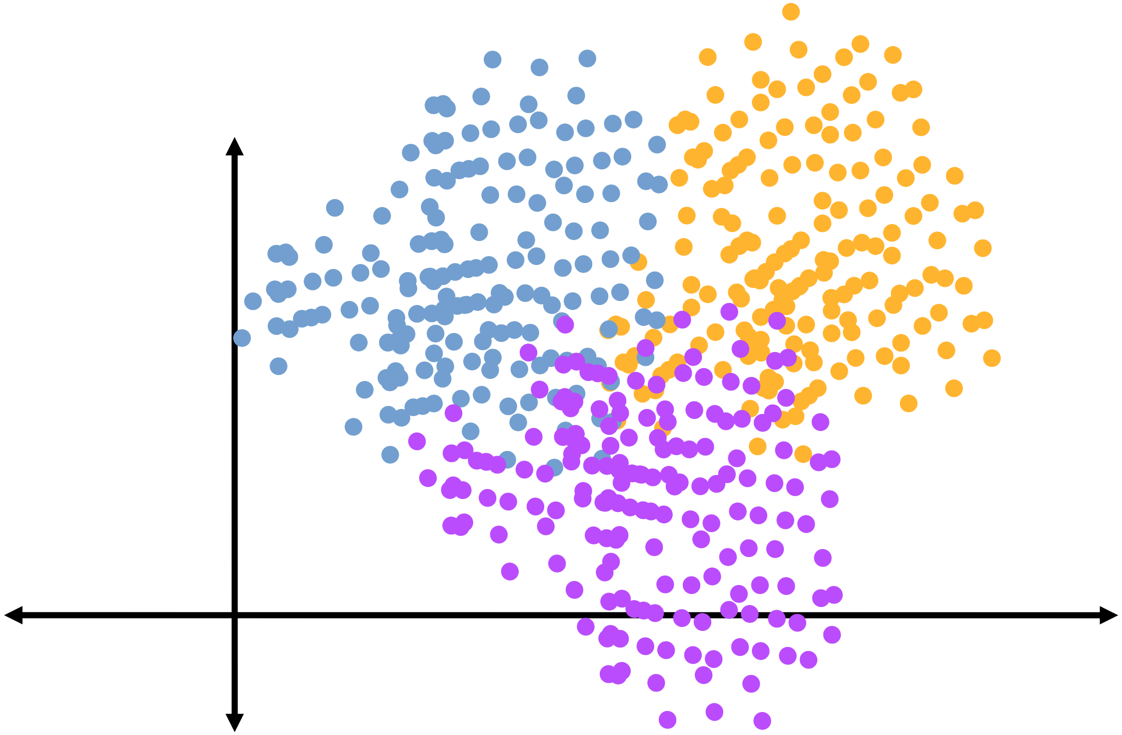
C = **-L(Y | X, Ө) = -** $\log\ [\ P^{y}\ (1-P)^{(1-y)}\ ]$

C = **-L(Y | X, Ө) =** $-ylogP-(1-y)\log\ (1-P)$

State Equation is normally Learning (Gradient Descent) Function:

$Ө_{n+1}$ = $Ө_{n}$ - r ∇C

Multiple Classification



Output Equation: Z = H(X, ϴ) = $\frac{e^{(\theta_i x+b_i)}}{\sum_{j=1}^{k} e^{(\theta_j x+b_j)}}$

Cost Equation: C = L(Z, Y) = $-\sum_{i=1}^{k} y_i$ log(Z)

State Equation: $\Theta_{n+1} = \Theta_n - r\nabla C$

Softmax – Cross-Entropy

In Multiple Classification Model,

Output Equation are Softmax Function and Log Likelihood Function with Cross Entropy:

P (X, ϴ) = $\frac{e^{(\theta_i x + b_i)}}{\sum_{j=1}^{k} e^{(\theta_j x + b_j)}}$

L(Y | X, ϴ) = $\sum_{i=1}^{k} \mathrm{y}_{\mathrm{i}} \log (P)$

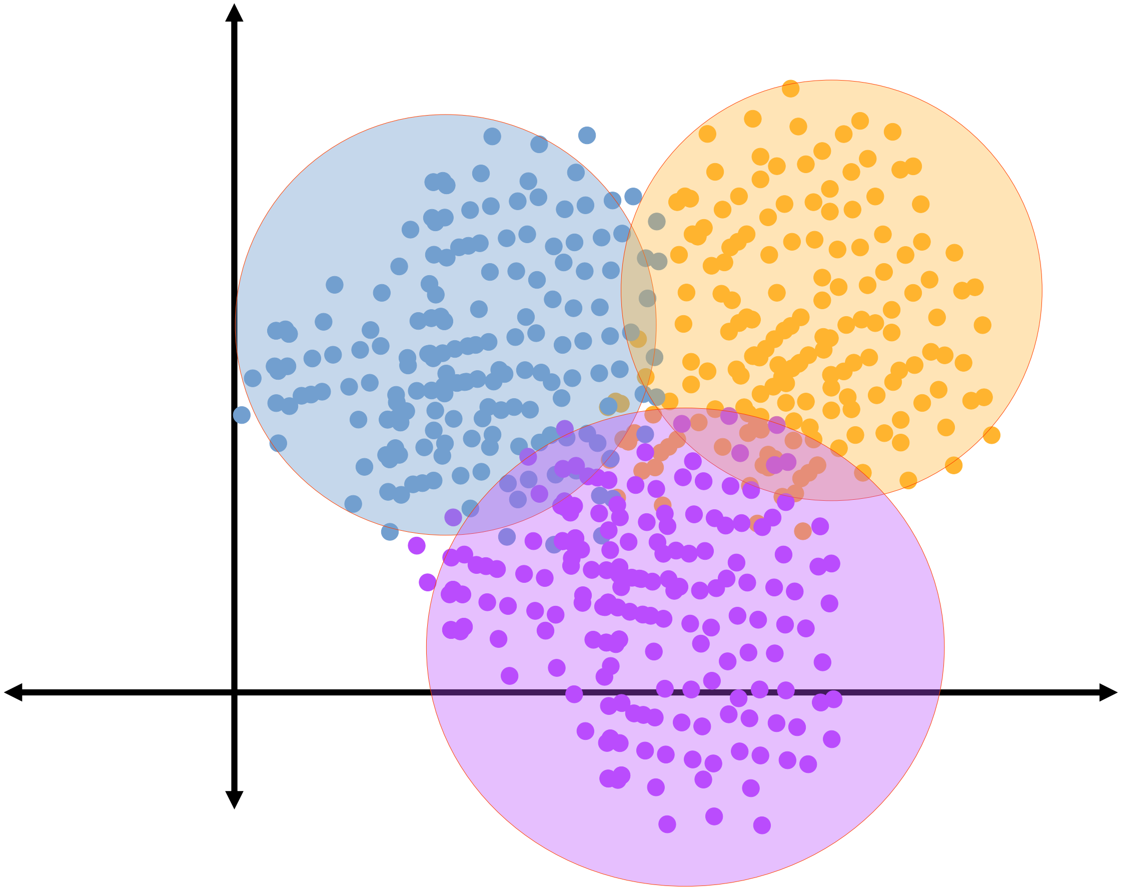
Cost Equation is normally Log Likelihood Function:

C = **-L(Y | X, ϴ) =** $\sum_{i=1}^{k} -y_i \log P$

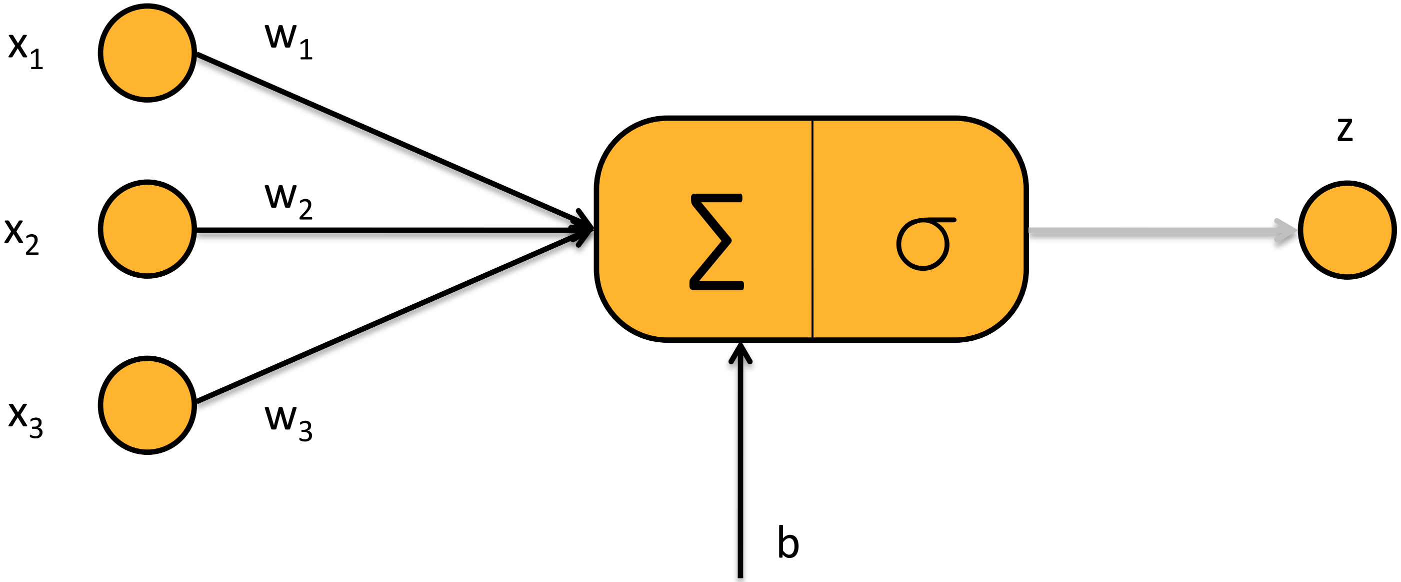
State Equation is normally Learning (Gradient Descent) Function:

$\Theta_{n+1} = \Theta_n - r \nabla C$

Clustering

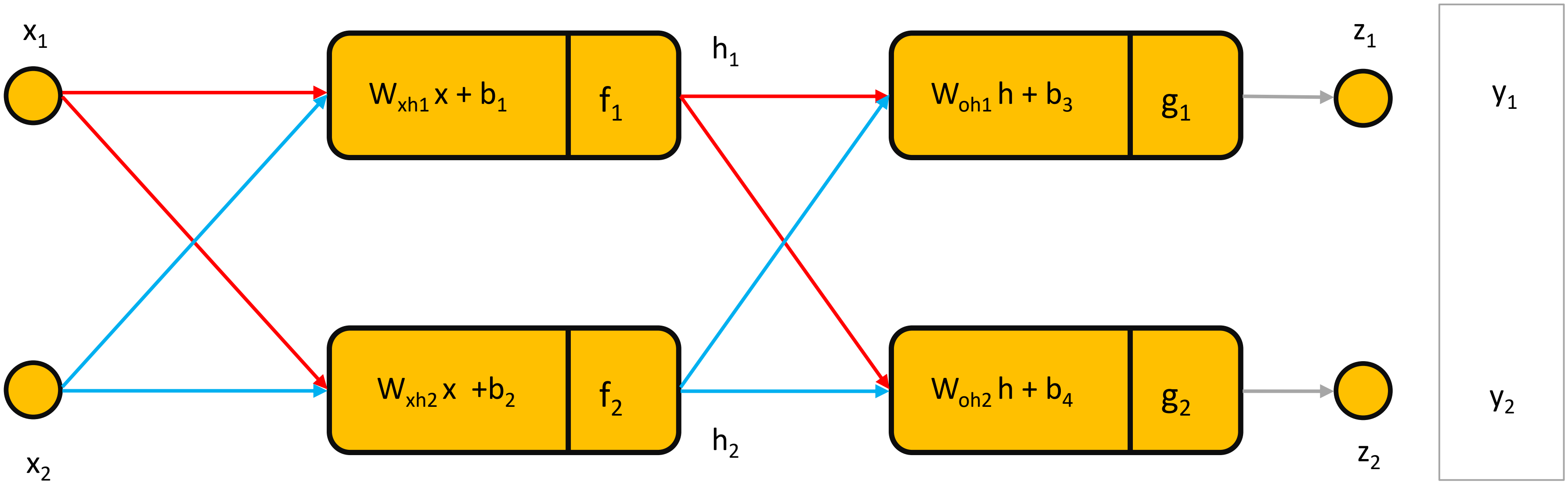


Perceptron

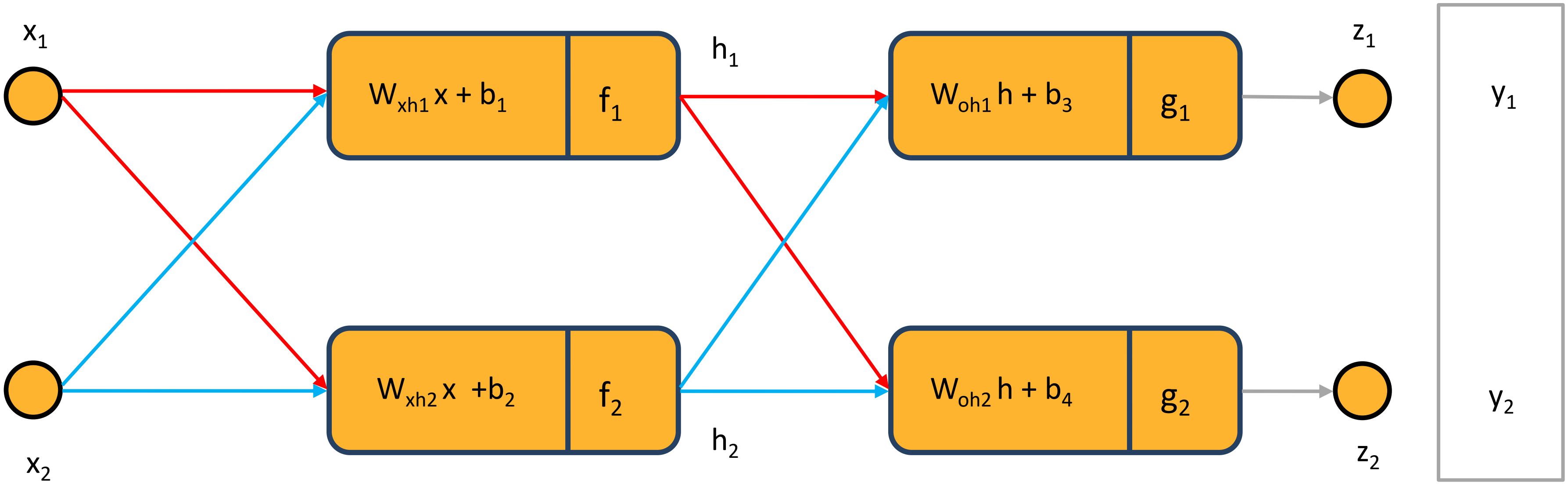


$$Z = \sigma\,[\,\sum w_i x_i + b]$$

Multi-Layer Perceptron



Multi Layer Perceptron

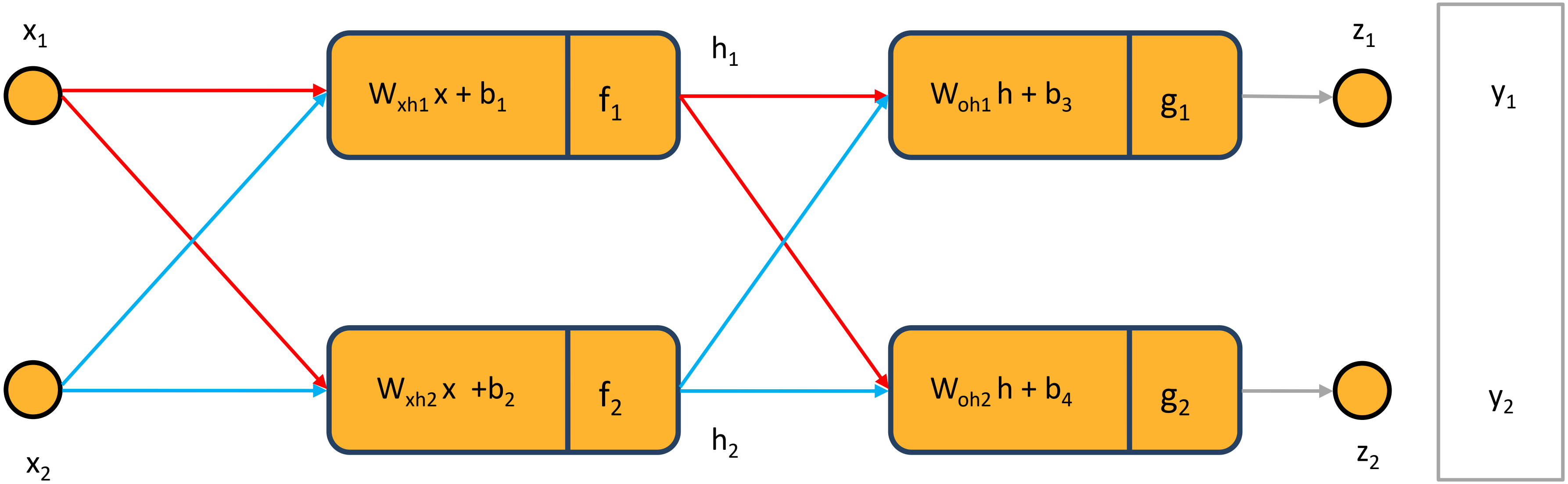


$h_1 = f_1(W_{xh1}\,x + b_1)$ $z_1 = g_1(W_{oh1}\,h + b_3)$

$h_2 = f_2(W_{xh2}\,x + b_2)$ $z_2 = g_2(W_{oh2}\,h + b_4)$

$c = ½\,(y – z)^2$ $c = (y\ )\ln(z)$ $c = (y\ )\ln(z) + (1\text{-}y)\ln(1\text{-}z)$

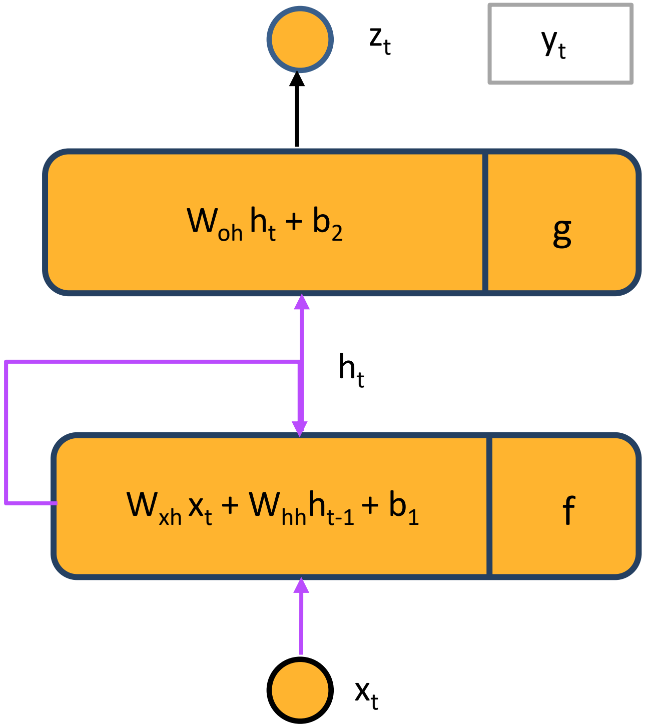
MLP – Backward Propagation



$$\frac{\partial c}{\partial w_{oh1}} = \left[\frac{\partial c}{\partial z_1}\frac{\partial z_1}{\partial h}\right]\frac{\partial h}{\partial w_{oh1}} \qquad \frac{\partial c}{\partial w_{oh2}} = \left[\frac{\partial c}{\partial z_2}\frac{\partial z_2}{\partial h}\right]\frac{\partial h}{\partial w_{oh2}}$$

$$\frac{\partial c}{\partial w_{xh1}} = \left(\sum_{i=1}^{n}\left[\frac{\partial c}{\partial z_i}\frac{\partial z_i}{\partial h}\right]w_{ohi}\right)\frac{\partial h_1}{\partial x}\frac{\partial x}{\partial w_{xh1}} \qquad \frac{\partial c}{\partial w_{xh2}} = \left(\sum_{i=1}^{n}\left[\frac{\partial c}{\partial z_i}\frac{\partial z_i}{\partial h}\right]w_{ohi}\right)\frac{\partial h_2}{\partial x}\frac{\partial x}{\partial w_{xh2}}$$

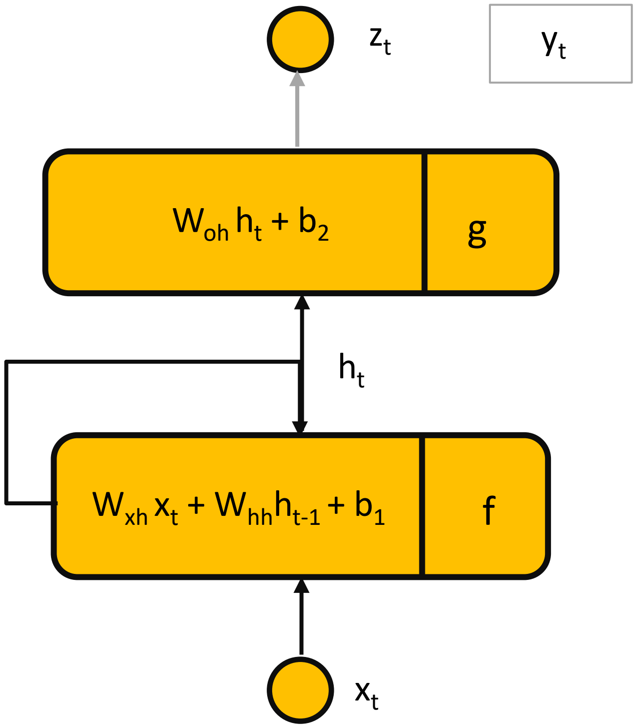
Recurrent Neural Network



$h_t = f(W_{xh}\, x_t \; + W_{hh}\, h_{t-1} + b_1)$ $z_t = g(W_{oh}\, h_t + b_2)$

$c_t = ½(y_t – z_t)^2$ $c_t = (y_t\,)\ln(z_t)$ $c_t = (y_t\,)\ln(z_t) + (1\text{-}y_t)\ln(1\text{-}z_t)$

Recurrent Network – Forward Propagation



z_t

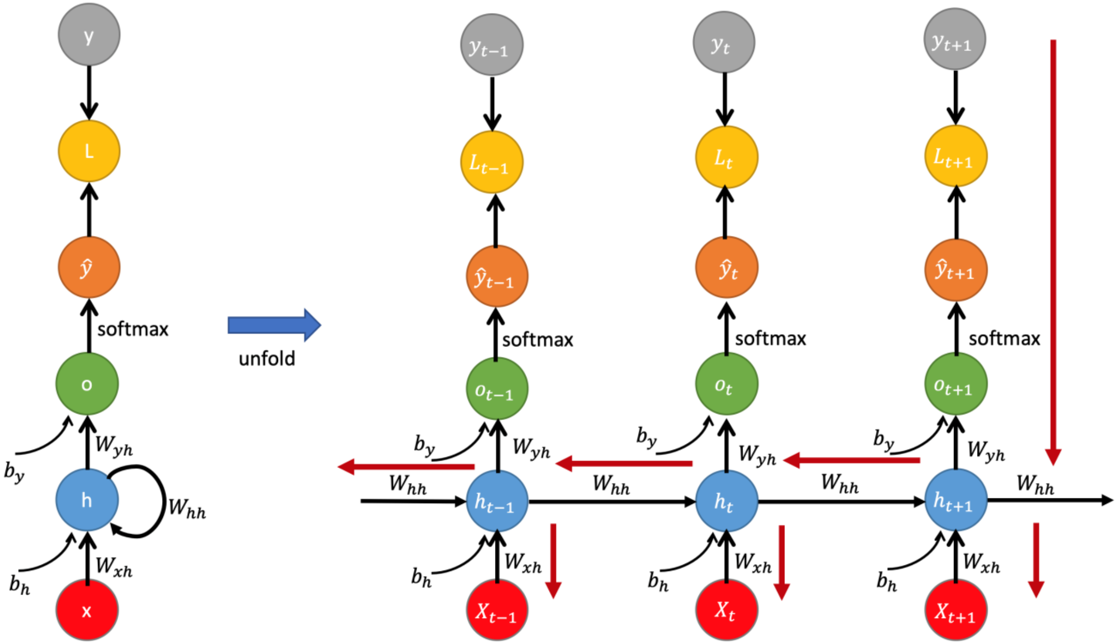
y_t

$W_{oh}\,h_t + b_2$ | g

h_t

$W_{xh}\,x_t + W_{hh}h_{t-1} + b_1$ | f

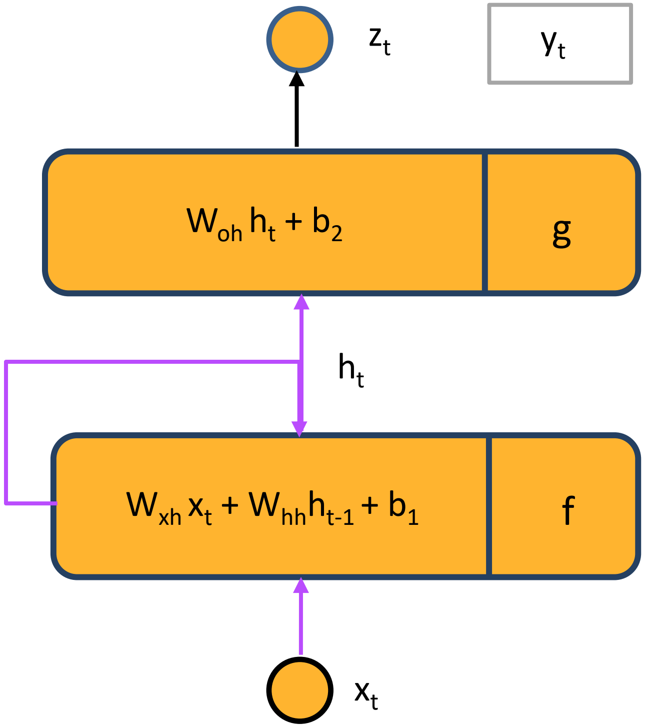
x_t



$$h_t = f(W_{xh}\,x_t \;+ W_{hh}\,h_{t-1} + b_1) \qquad\qquad z_t = g(W_{oh}\,h_t + b_2)$$

$$c_t = ½(y_t - z_t)^2 \qquad c_t = (y_t\,)\ln(z_t) \qquad c_t = (y_t\,)\ln(z_t) + (1\text{-}y_t)\ln(1\text{-}z_t)$$

RNN – Backward Propagation

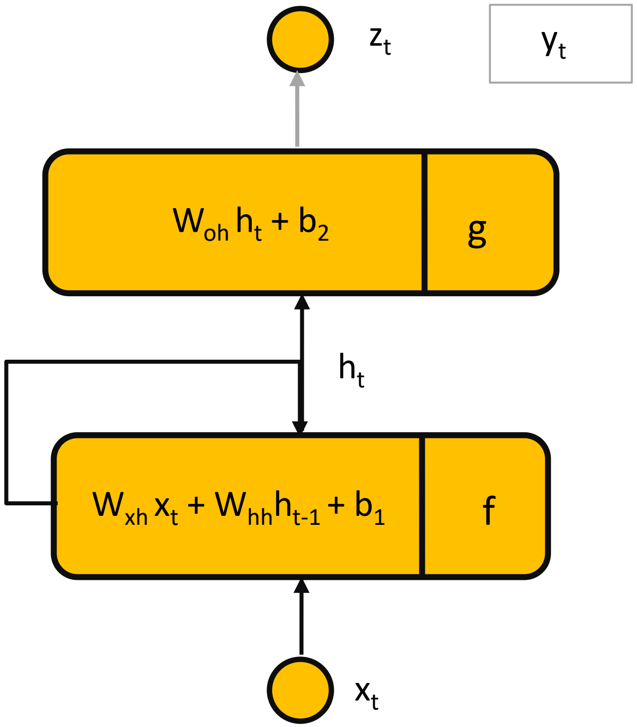


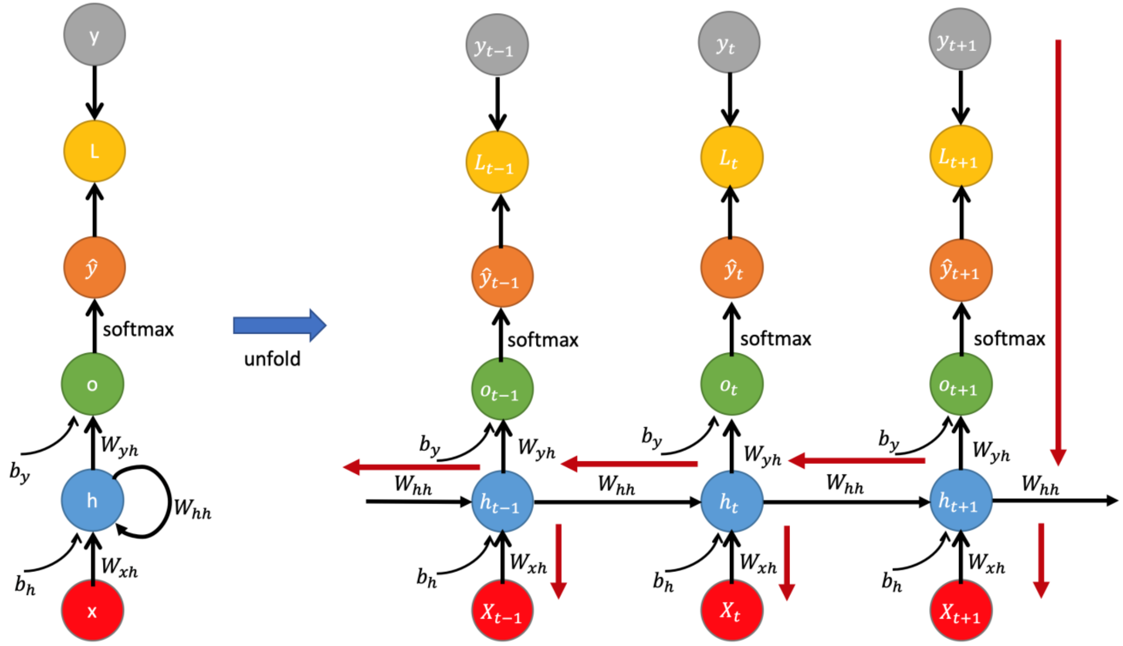
$$\frac{\partial c_t}{\partial w_{oh}} = \left[\frac{\partial c_t}{\partial z_t}\frac{\partial z_t}{\partial h_t}\right]\frac{\partial h_t}{\partial w_{oh}}$$

$$\frac{\partial c_{t+1}}{\partial w_{hh}} = \left[\frac{\partial c_{t+1}}{\partial z_{t+1}}\frac{\partial z_{t+1}}{\partial h_{t+1}}\right]\underline{\left[\prod_{j=k}^{t}\frac{\partial h_{j+1}}{\partial h_j}\right]}\frac{\partial h_k}{\partial w_{hh}}$$

$$\frac{\partial c_{t+1}}{\partial w_{xh}} = \left[\frac{\partial c_{t+1}}{\partial z_{t+1}}\frac{\partial z_{t+1}}{\partial h_{t+1}}\right]\underline{\left[\prod_{j=k}^{t}\frac{\partial h_{j+1}}{\partial h_j}\right]}\frac{\partial h_k}{\partial w_{xh}}$$

Recurrent Network – Backward Propagation

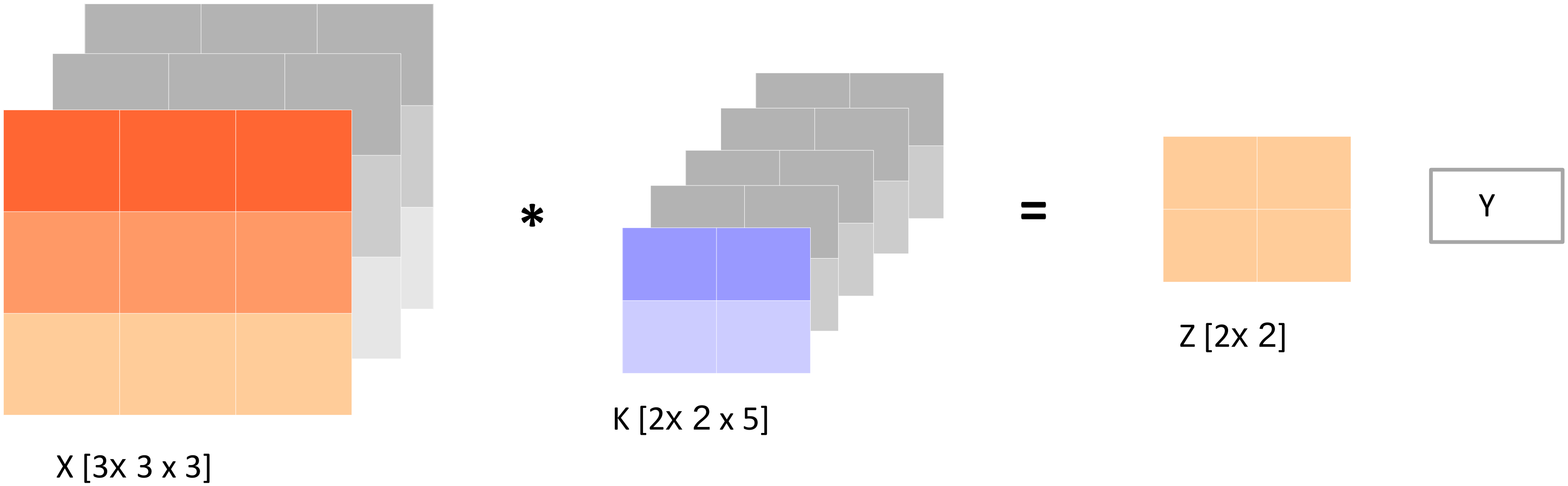




$$\frac{\partial c_t}{\partial w_{oh}} = \left[\frac{\partial c_t}{\partial z_t}\frac{\partial z_t}{\partial h_t}\right]\frac{\partial h_t}{\partial w_{oh}}$$

$$\frac{\partial c_{t+1}}{\partial w_{hh}} = \left[\frac{\partial c_{t+1}}{\partial z_{t+1}}\frac{\partial z_{t+1}}{\partial h_{t+1}}\right]\left[\prod_{j=k}^{t}\frac{\partial h_{j+1}}{\partial h_j}\right]\frac{\partial h_k}{\partial w_{hh}} \qquad \frac{\partial c_{t+1}}{\partial w_{xh}} = \left[\frac{\partial c_{t+1}}{\partial z_{t+1}}\frac{\partial z_{t+1}}{\partial h_{t+1}}\right]\left[\prod_{j=k}^{t}\frac{\partial h_{j+1}}{\partial h_j}\right]\frac{\partial h_k}{\partial w_{xh}}$$

Convolution Neural Network



X [3x 3 x 3]

*

K [2x 2 x 5]

=

Z [2x 2]

Y

$$(f * g)(t) := \int_{-\infty}^{\infty} f(t - \tau) g(\tau) \, d\tau.$$

Weight Sharing By Convolution

W_1	W_2	W_3	W_4	W_5
X_1	X_2	X_3	X_4	X_5
W_1X_1	W_2X_2	W_3X_3	W_4X_4	W_5X_5

PRODUCT OF WEIGHTS AND INPUT = W × X

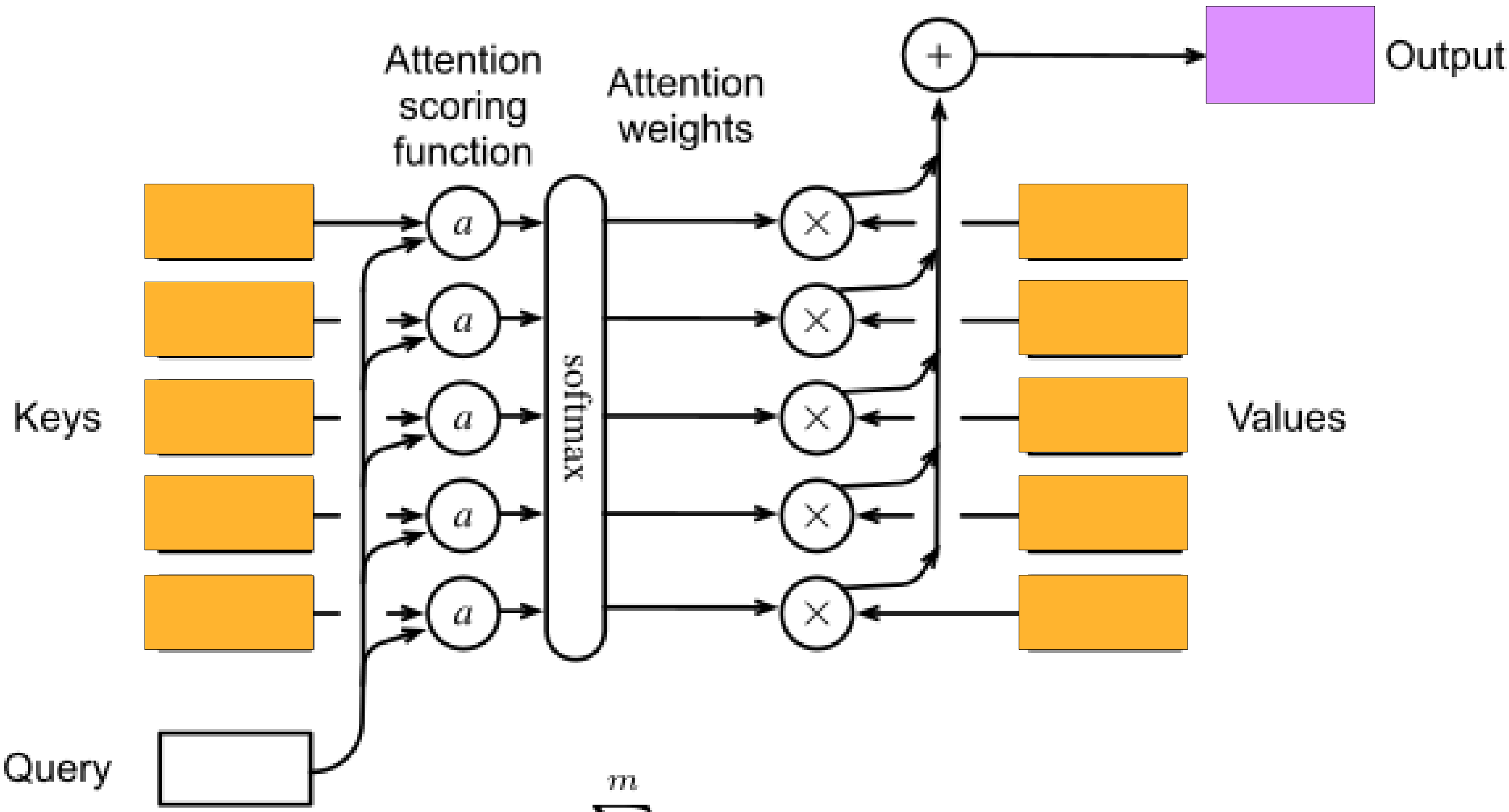
W_1	W_2	W_3

X_1	X_2	X_3	X_4	X_5

W_3X_1	$W_2X_1+W_3X_2$	$W_1X_1+ W_2X_2+W_3X_3$	$W_1X_2+ W_2X_3+W_3X_4$	$W_1X_3+W_2X_4+W_3X_5$	$W_1X_4+ W_2X_5$	W_1X_5

CONVOLUTION OF WEIGHTS AND INPUT = W * X

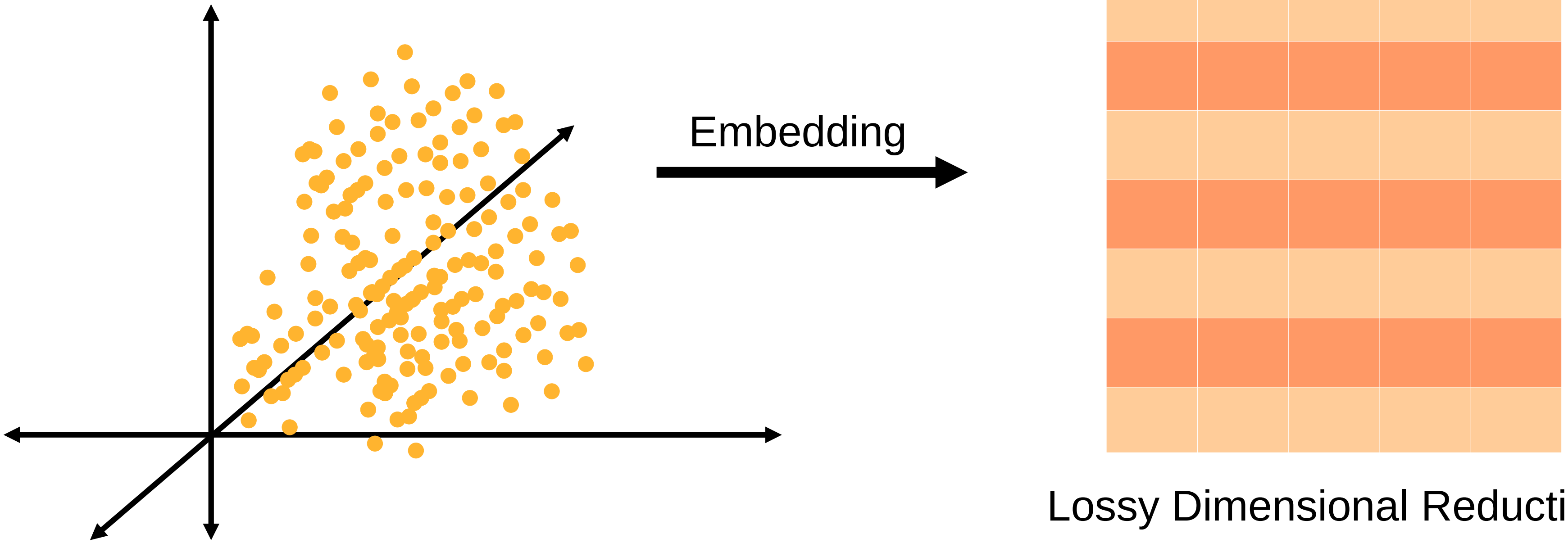
Attention Pooling



$$\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \in \mathbb{R}^v,$$

$$f(x) = \sum_{i=1}^{n} \alpha(x, x_i) y_i,$$

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}.$$

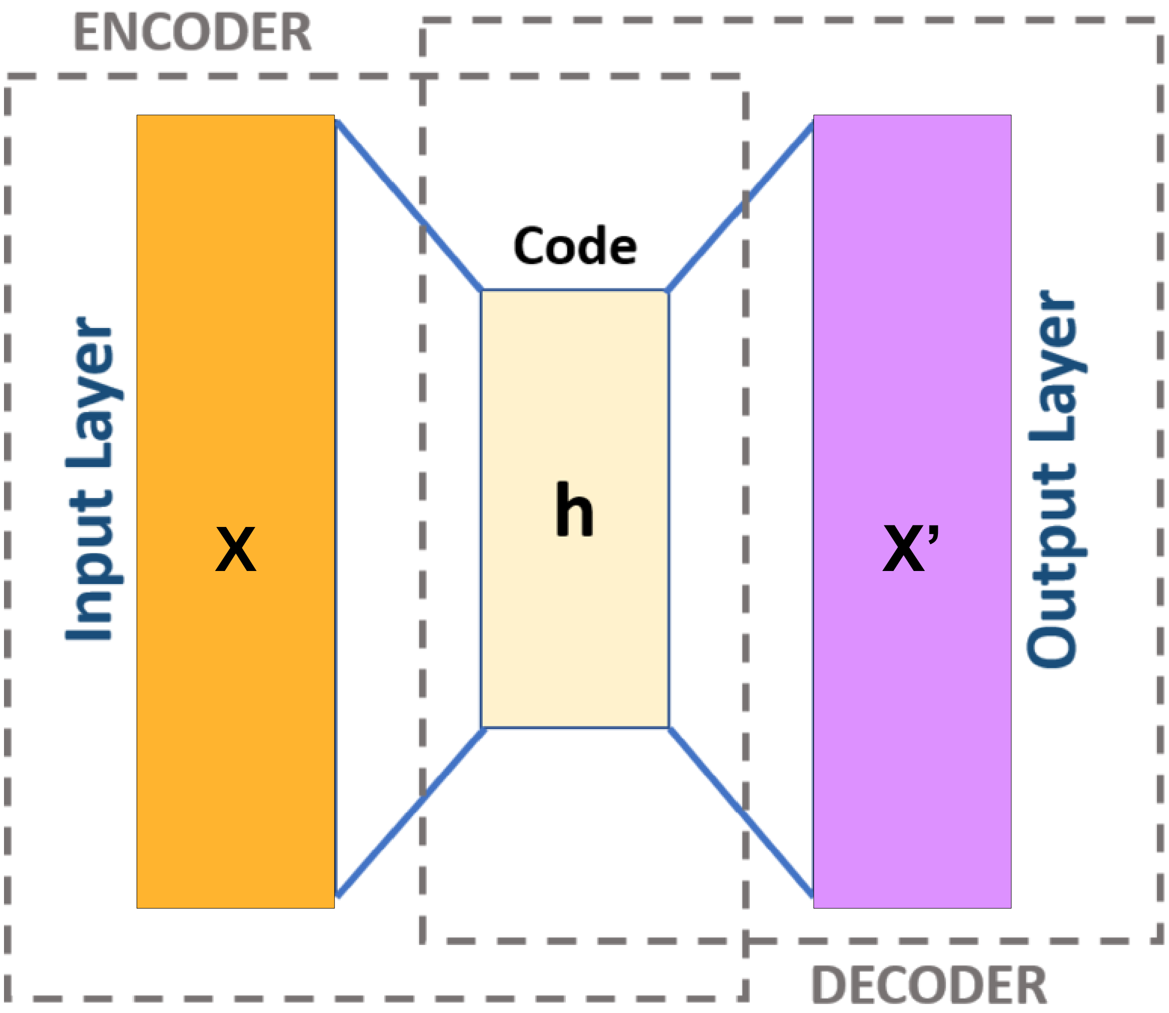


Input Space [X, Y, Z]

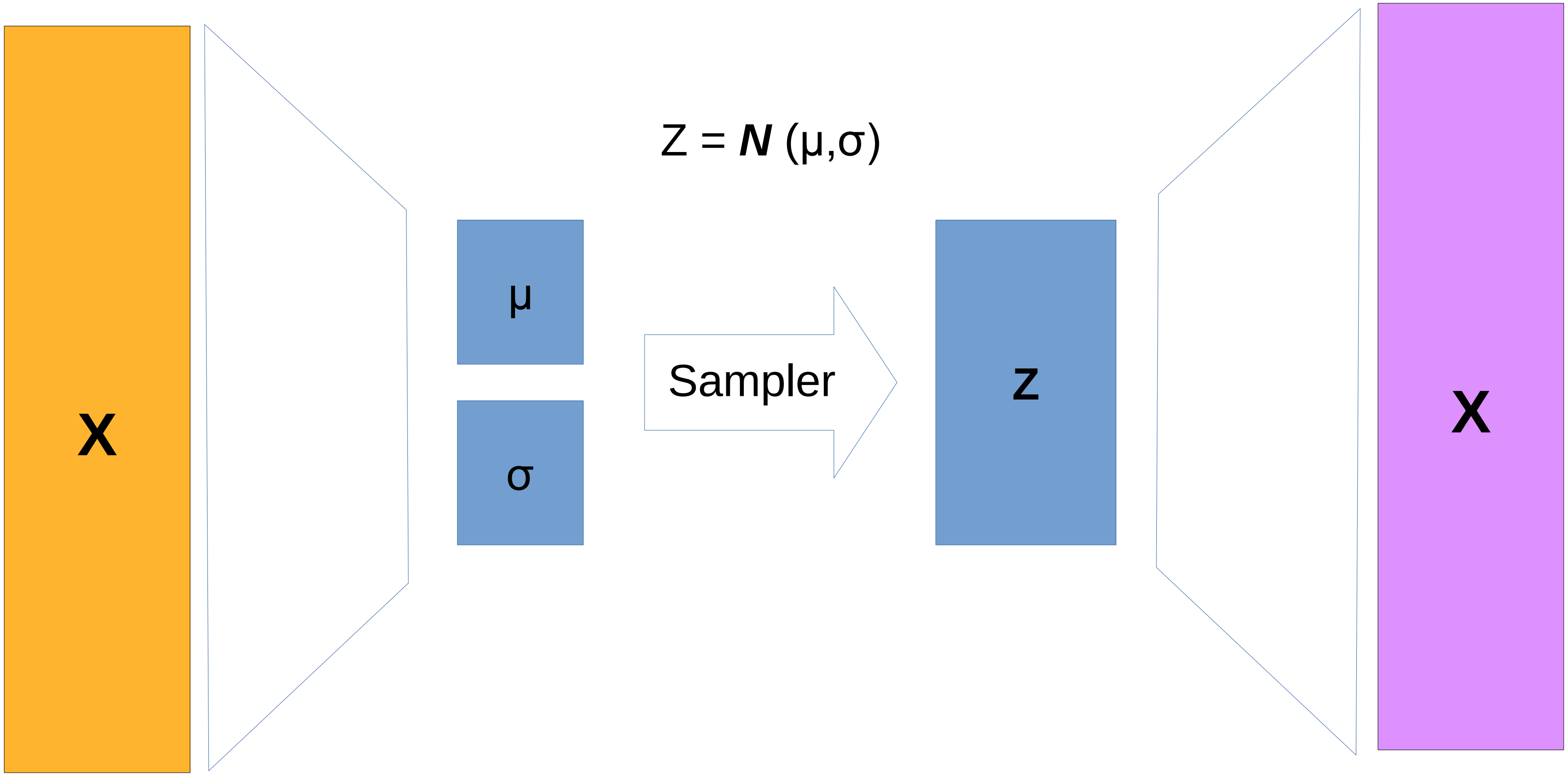
Embedding

Latent Space [U, V]

Lossy Dimensional Reduction



Auto Encoder



$Z = \boldsymbol{N}(\mu,\sigma)$

X

μ

σ

Sampler

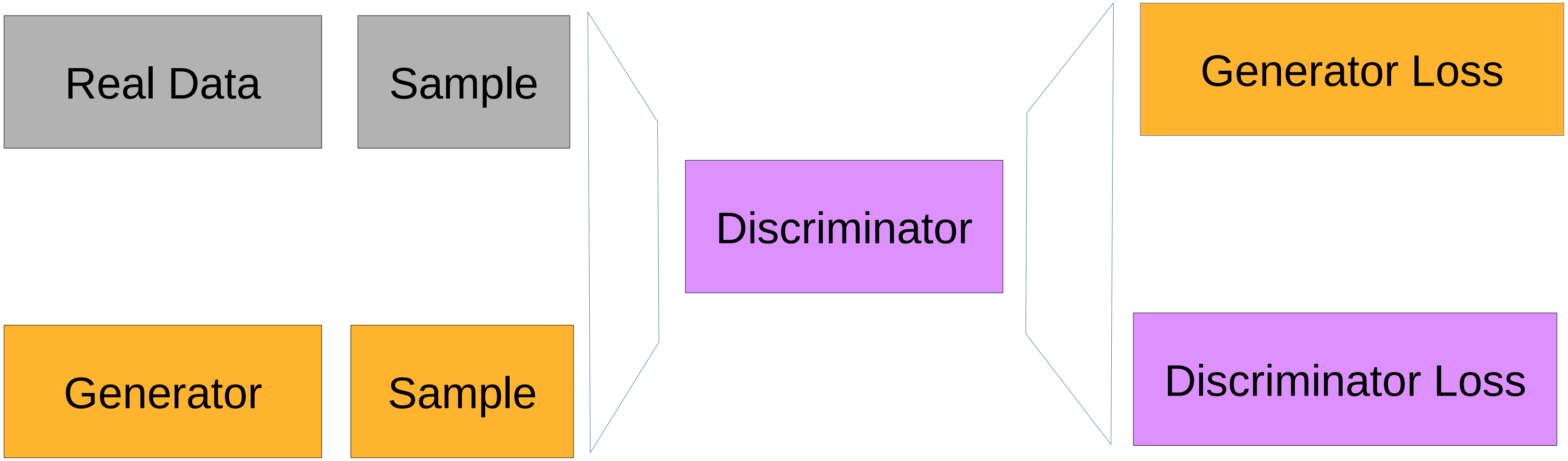
Z

X

Variational Auto Encoder

Zero-Sum Game between Generator and Discriminator

$$U(D,G)=\mathrm{E}_{x\sim P_x(x)}\left[\log D(x)\right]+E_{z\sim P_z(z)}[\log\left(1-D(G(z))\right]$$


Real Data

Sample

Generator

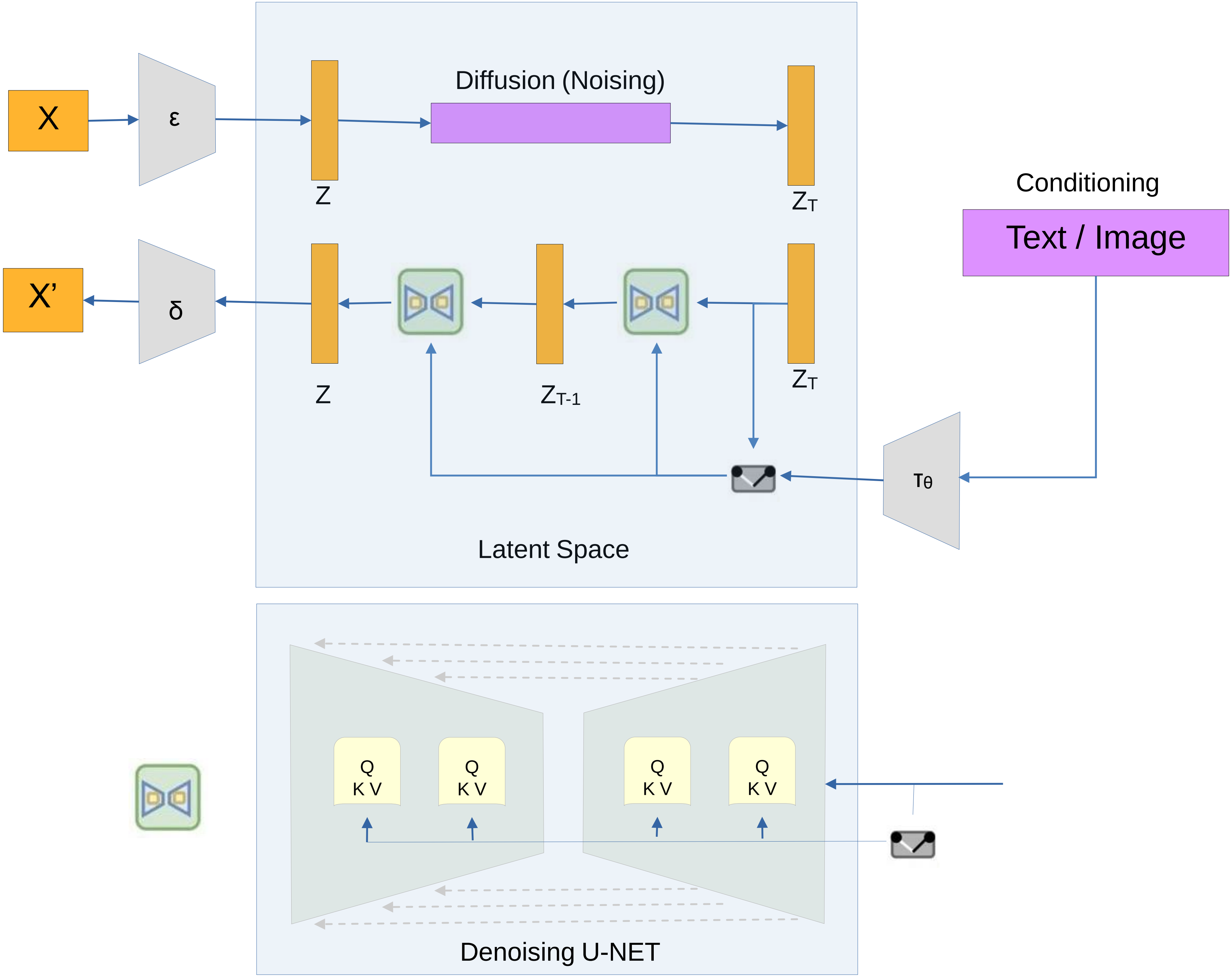
Sample

Discriminator

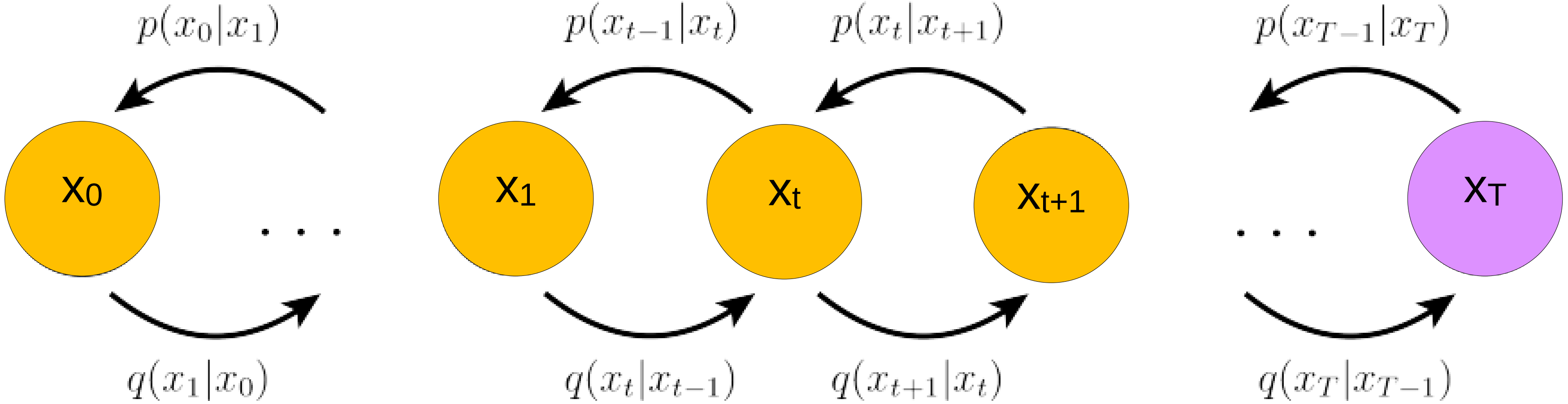
Generator Loss

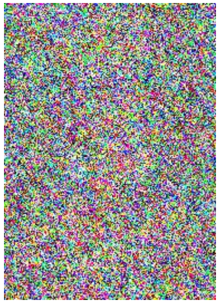
Discriminator Loss

Generative Adversarial Network

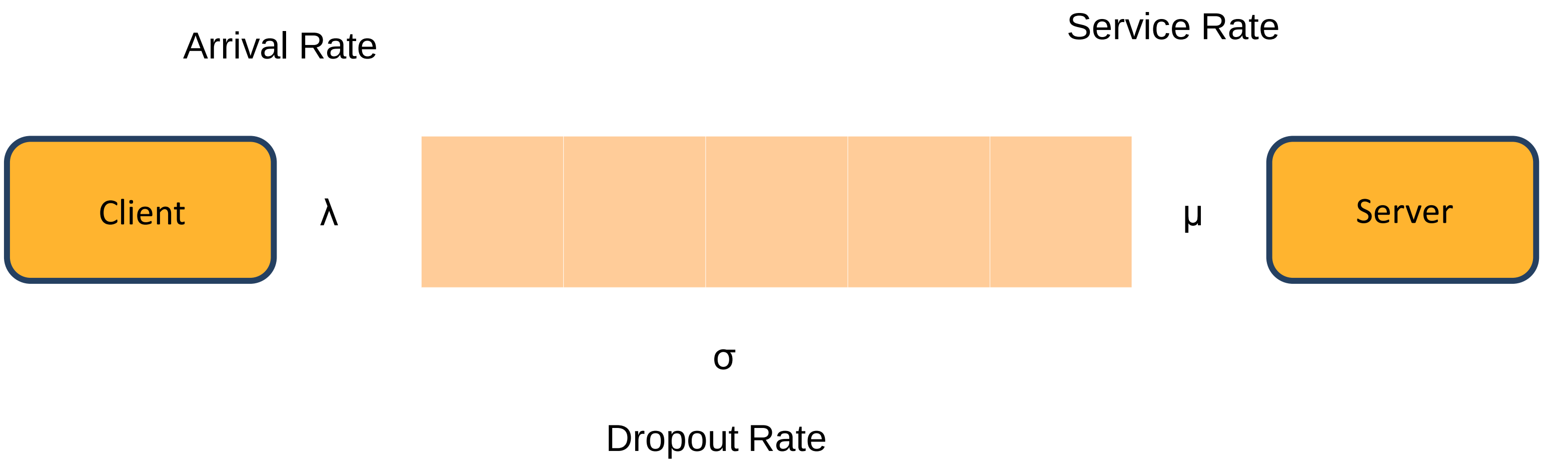

X
ε
Z
Diffusion (Noising)
ZT
Conditioning
Text / Image
X'
δ
Z
ZT-1
ZT
Tθ
Latent Space
Q
K V
Q
K V
Q
K V
Q
K V
Denoising U-NET

$$\frac{1}{2}\left(\mathrm{SNR}(t-1)-\mathrm{SNR}(t)\right)\left[\|\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t,t)-\boldsymbol{x}_0\|_2^2\right]$$





Queuing Theory



Queue Size

$$L = \frac{\lambda - \sigma}{\mu}$$

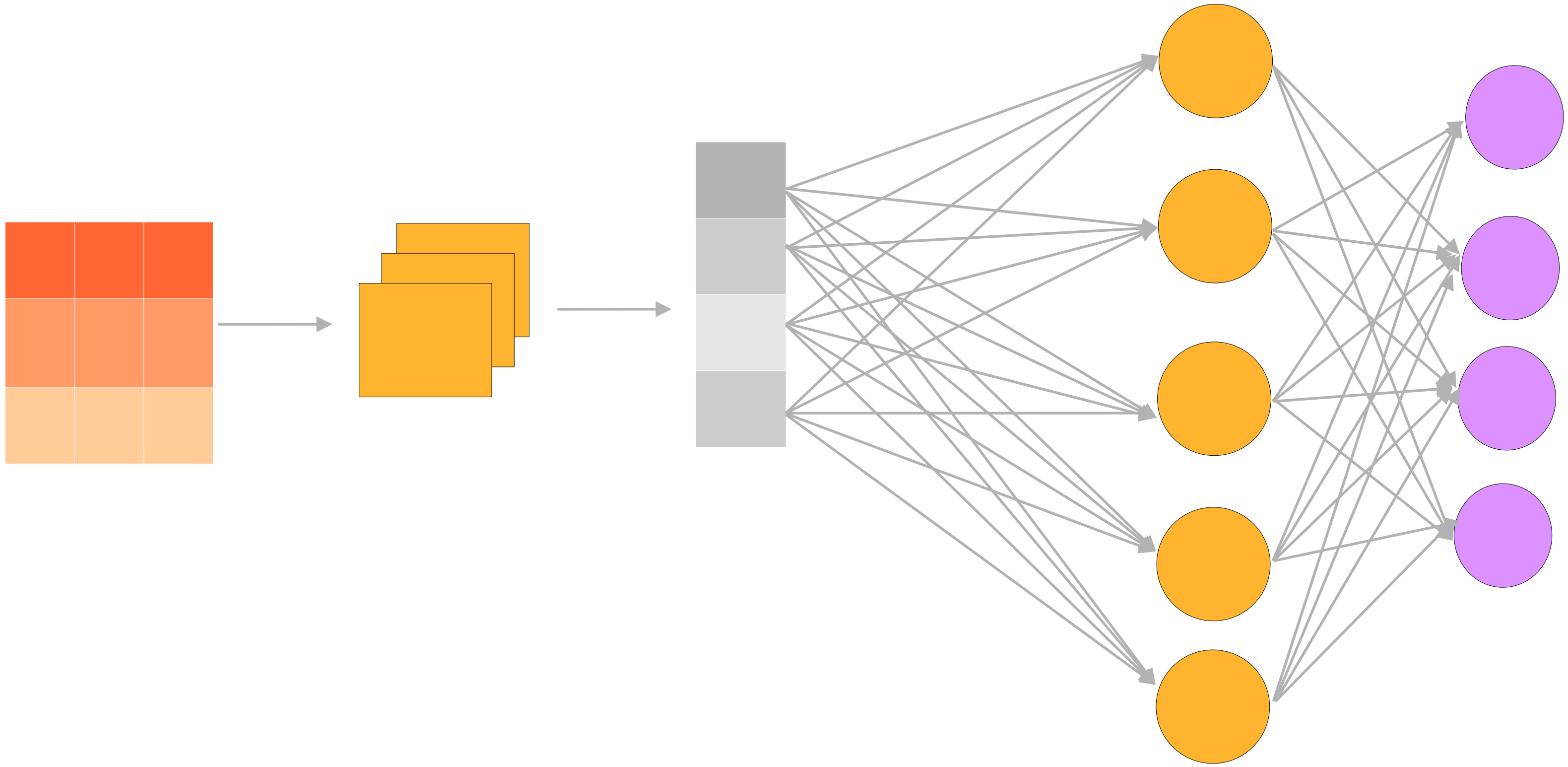
Process Rate

$$\frac{\mu}{\lambda} = e^{-W\mu}$$

Wait Time

$$W = \frac{1}{\mu} \ln \frac{\lambda}{\mu}$$

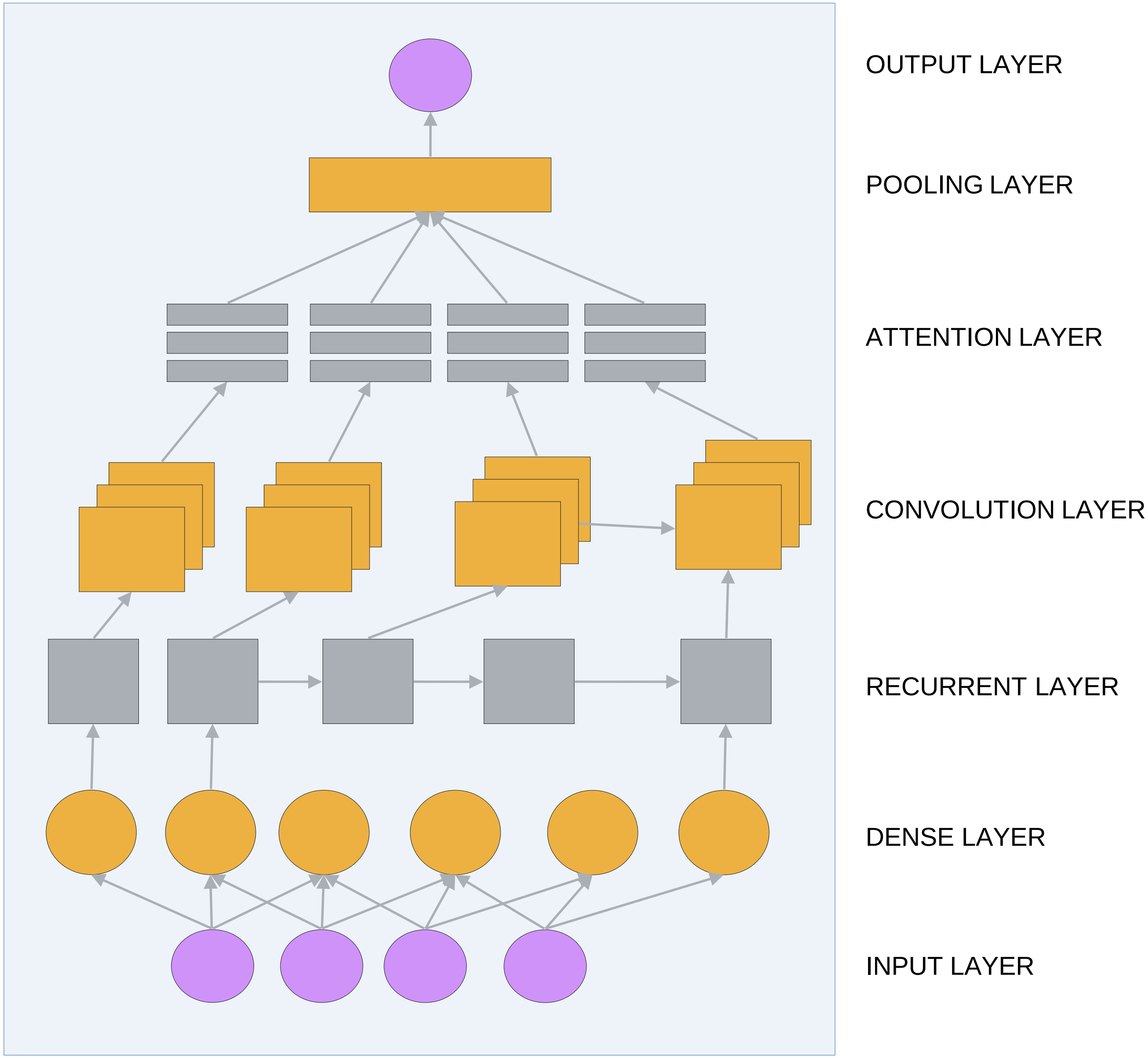
Image Classification



INPUT CONVOLUTION FLATTEN (F) DENSE (M) OUTPUT (Z)

Z = M(F), where F = Features and M = Transformation Function

A Typical Neural Network Architecture



Energy Based Model

Learnable Systems have 3 Functions:

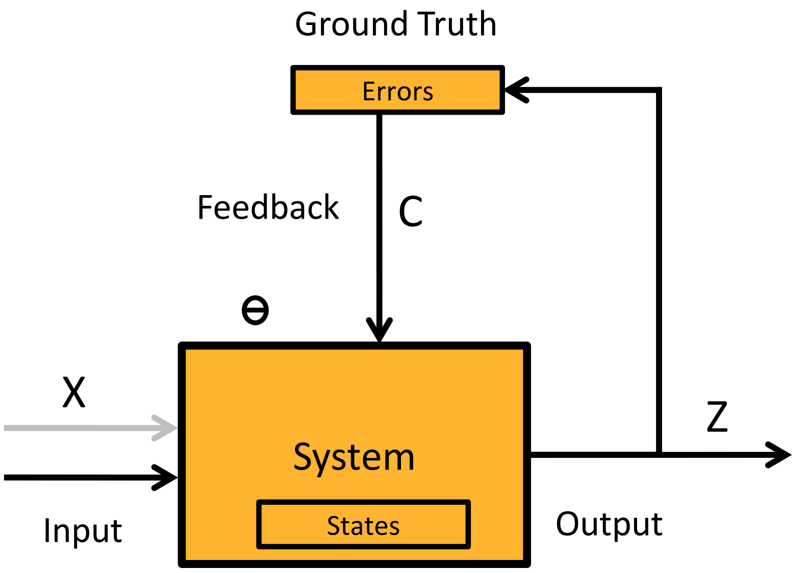
1. Output Function
2. Cost Function
3. State Function

Statistically, we can define:

P(X) = Probability Distribution of Input
P(Θ) = Probability Distribution of States
P(Z) = Probability Distribution of Output

P(Z) = P(X) P(Θ) = P(X, Θ)

EBM can explain well why Machine Learning works.



Energy Landscape

Energy is the ability to change the States of a System.

Whenever energy is applied to a system, the states of the system change.

However, how the states change will always follow a Trajectory (Path), which is defined by Entropy.

In other words, Entropy is the State Function which follows a Trajectory in Energy Landscape.

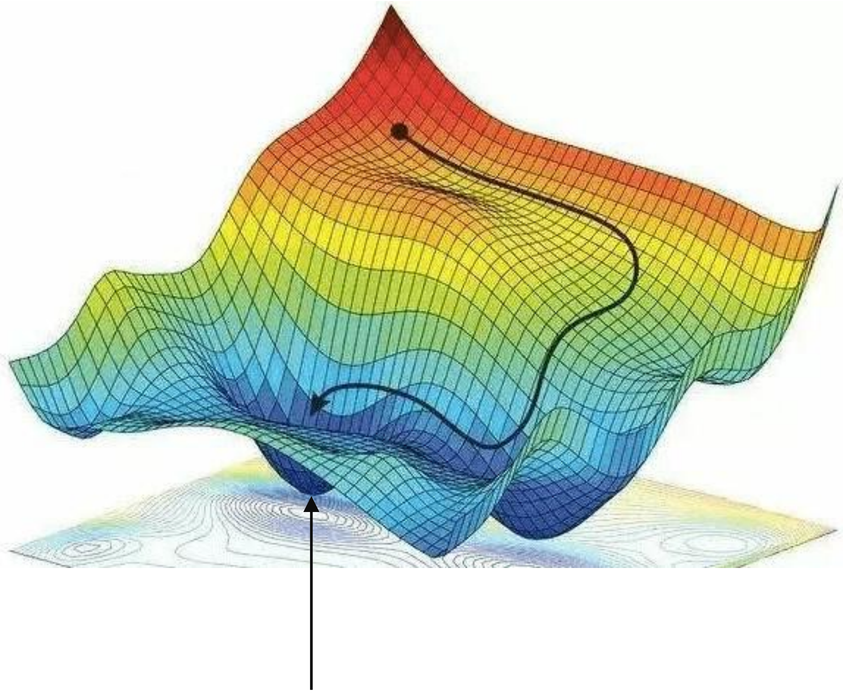
All possible trajectories of state changes will define an Energy Landscape.

Maximum Entropy defines the Local or Global Minimum in Energy Landscape.

In other words , Maximum Entropy is the Equilibrium in which all possible states in the system are randomly uniformly distributed.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

Entropy is inversely proportional to (Free) Energy.



Maximum Entropy is the highest when all possible states are random-uniform, and (Free) Energy is the lowest.

State Space

State Space of a system defines the probability distribution of all possible states in the system: $P(\Theta)$

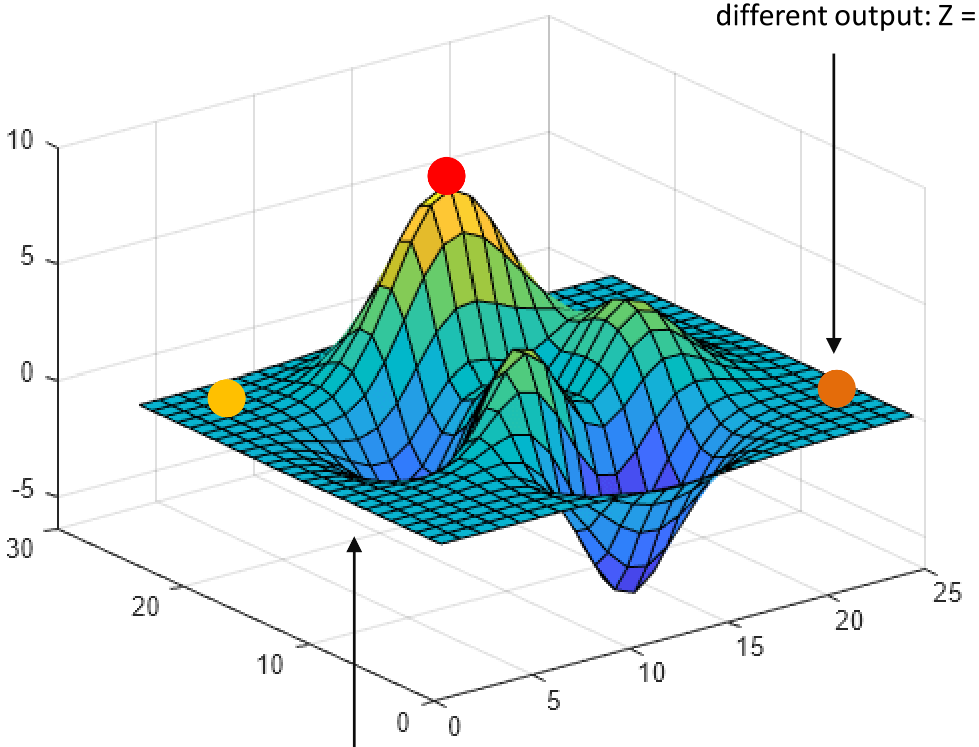
When Input (X) is applied to the system, $P(X)$ will define the all possible inputs for the System.

Then, Output (Z) will be the Joint Probability Distribution of all possible inputs and states of the systems: $P(Z) = P(X)\, P(\Theta) = P(X, \Theta)$.

However, if we marginalize $P(\Theta)$, then this joint probability distribution will be reduced to a point estimate function: $Z = F(X, \Theta)$ where Θ is kept constant.

Actually, $F(X, \Theta)$ also defines a State Space Landscape which determines the trajectory of Inputs when states (Θ) is kept constant.

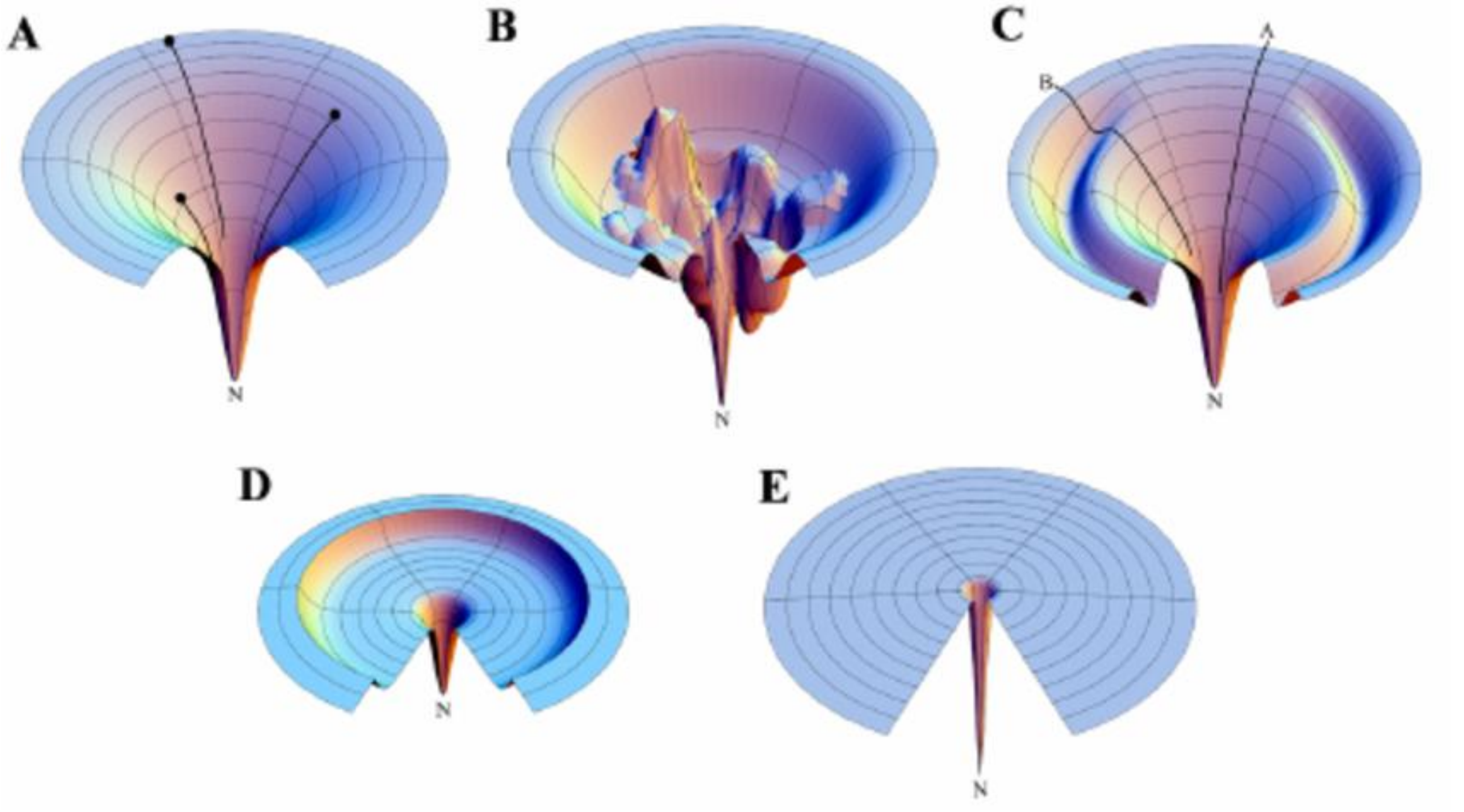
State Space Landscape



X defines the different Input States which will follow different trajectory, which will give different output: Z = F(X, ϴ)

State Space Landscape in which when states (ϴ) is kept constant or frozen.

Different State Space



Changing State Space

As we can see different State Space can define different Output Function $Z = F(X, \Theta)$ for the same input X.

But how do we change the State Space, and what would be the trajectory of changing State Space?

On what basis, do we have to **change the State Space**?

We already know that the probability distribution of all possible states of a system is $P(\Theta)$.

Therefore, if we allow the trajectory of changing states to follow the entropy, it will change the Landscape of State Space.

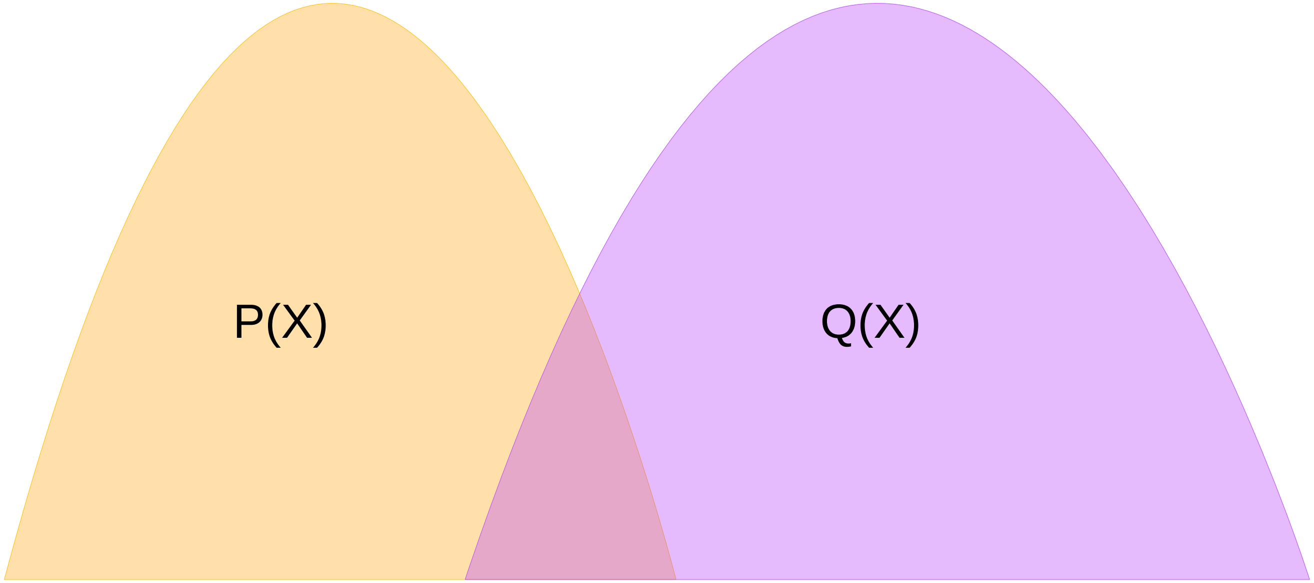
The Entropy of the State Space is defined with **Cost Function Space**: the **Joint Entropy** of **Output** (Z) and **Ground Truth** (Y).

However, since computing **Joint Entropy** is not always available, **Cross-Entropy** and **Relative Entropy** (KL Divergence) is mostly used in Machine Learning.

In short, training the State Space of DNN (Deep Neural Network) always follows **the reverse trajectory** of **Joint Entropy** of Output (Z) and Ground Truth (Y): from **Maximum Entropy** to **Minimum Entropy**.

And that is the **Gradient Descent**.

Joint Entropy



$$H(P(X)) - H(P(X),Q(X)) \sum D_{KL}(P(X), Q(X))$$

$$I(Q;P) = \underbrace{H(P) - H(P \mid Q)}_{\text{DISCRIMINATIVE}} = \underbrace{H(Q) - H(Q \mid P)}_{\text{GENERATIVE}}$$

Similarity

If we try to understand the **Joint Entropy** from the perspective of **Similarity**, it will also make sense.

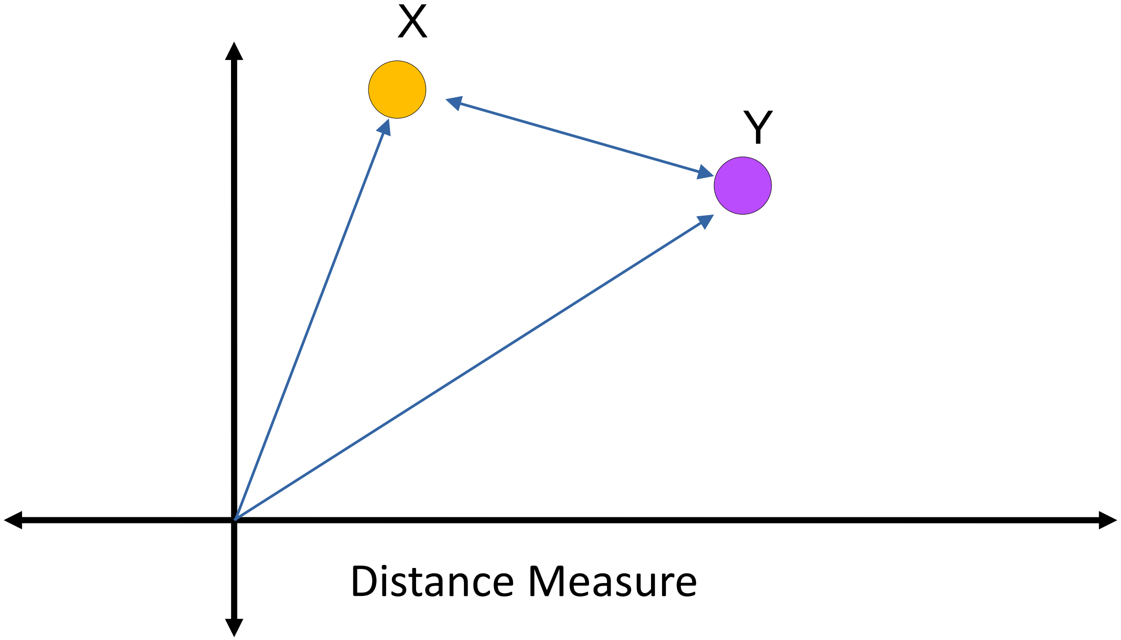
Y and Z will become more similar if their Joint Entropy is minimized, and their **Mutual Information** is maximized.

If we also look at from the perspective of Feature Space, which can be defined by Inner (Dot) Product Space between Feature Map of Y: **Ψ(Y)** and Feature Map of Z: **Ψ(Z)** , we can see more similar features are closer.

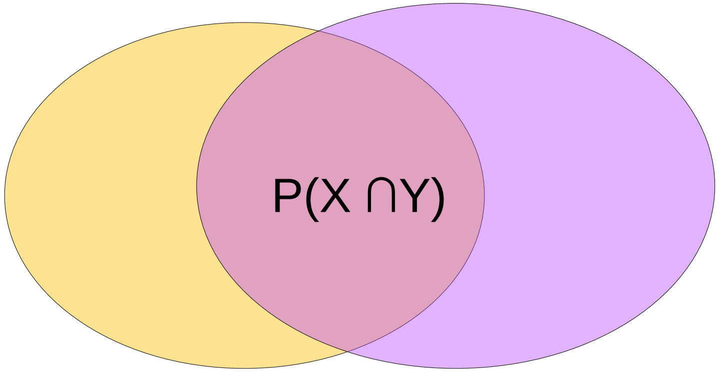
That's why we can also use **Distance Measure** to minimize the **Joint Entropy**.

$$\boldsymbol{K}(Y,Z) = \boldsymbol{\Psi}(Y).\ \boldsymbol{\Psi}(Z)$$

Similarity Measure



Distance Measure



Intersection Measure

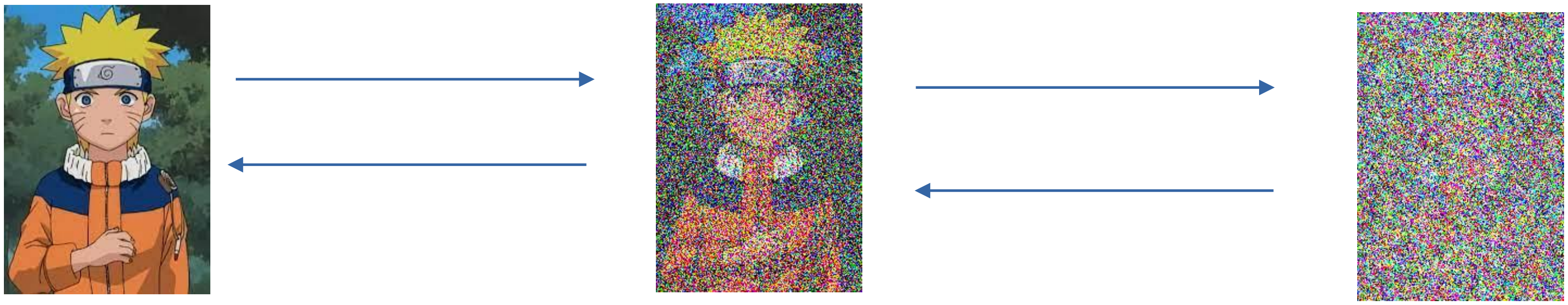
Trajectory of Entropy

We can see the Trajectory of Entropy more clearly in Diffusion Model.

When we are noising an image, we are actually following the **Natural Course of Entropy**: from lower to higher Entropy, which is relatively easy.

However, when we are de-noising an image, we are following **the Reverse Course of Entropy**: from higher to lower Entropy, which is relatively more difficult.

Yet, since the **Natural Trajectory of Entropy** is already established in noising steps, Diffusion Model just needs to **ascend the same path** from: Higher to Lower Entropy, in de-nosing steps.



Arrows of Time

The Second Law of Thermodynamics defines the **Natural Course of Entropy**: from lower to higher Entropy States, which determines “The Arrow of Time”.

However, from Energy Based Model, we can assume that there can also be **the Reverse Course of Entropy**: from higher to lower Entropy States.

It could also pose a more general question as to whether “**Life**” or “**Intelligence**” has something to do with “Second Arrow of Time”.

Lately, there is another popular Theory in Neuroscience, commonly known as “**Free Energy Principle**”, first proposed by **Karl. J. Friston**.

Evolution will also make sense from that perspective. Life evolves by reversing the Entropy.

What if, I really mean what if, the origin of life (or intelligence) has started from following the Second arrow of Time by following the **Reverse Course of Entropy**?

Footnote: Interestingly, the **Natural Course of Entropy** and **the Reverse Course of Entropy** also coincide with Buddhism: the **Truth of Suffering** states that everything will eventually decay, and **Truth of Attachment** states that all life-forms want to reverse it as much as possible.

More Questions

Although EBM (Energy Based Model) explains well about Machine Learning, we still have many questions.

Actually, most Machine Learning has two separate phases: **Learning (Training) Phase** and **Inference Phase**.

During the Learning Phase, we are optimizing for **P(Θ)** with respect to Joint Entropy of **H(Y, Z)**, that is, we are changing **Θ**.

However, during the Inference Phase, **P(Θ)** is usually marginalized, that is to say, **Θ** is kept constant.

Although there are some notions of **Zero-Shot Learning**, **P(Θ)** is usually marginalized or **Θ** is kept **constant**.

However, Output (Z) will be the Joint Probability Distribution of all possible inputs and states of the systems: P(X, Θ), AI also need to learn or change **Θ**, during the Inference Phase.

Yet, we still don’t know how to change **Θ**, during the Inference Phase, as **Θ** is kept **constant**.

In other words, we still don’t know “**Learn on the fly**” Model yet, even at least theoretically, which would require changing **Θ** in real time.

I think that would be the key to Next Level of AI, and I think “**World Model**” would be the solution.