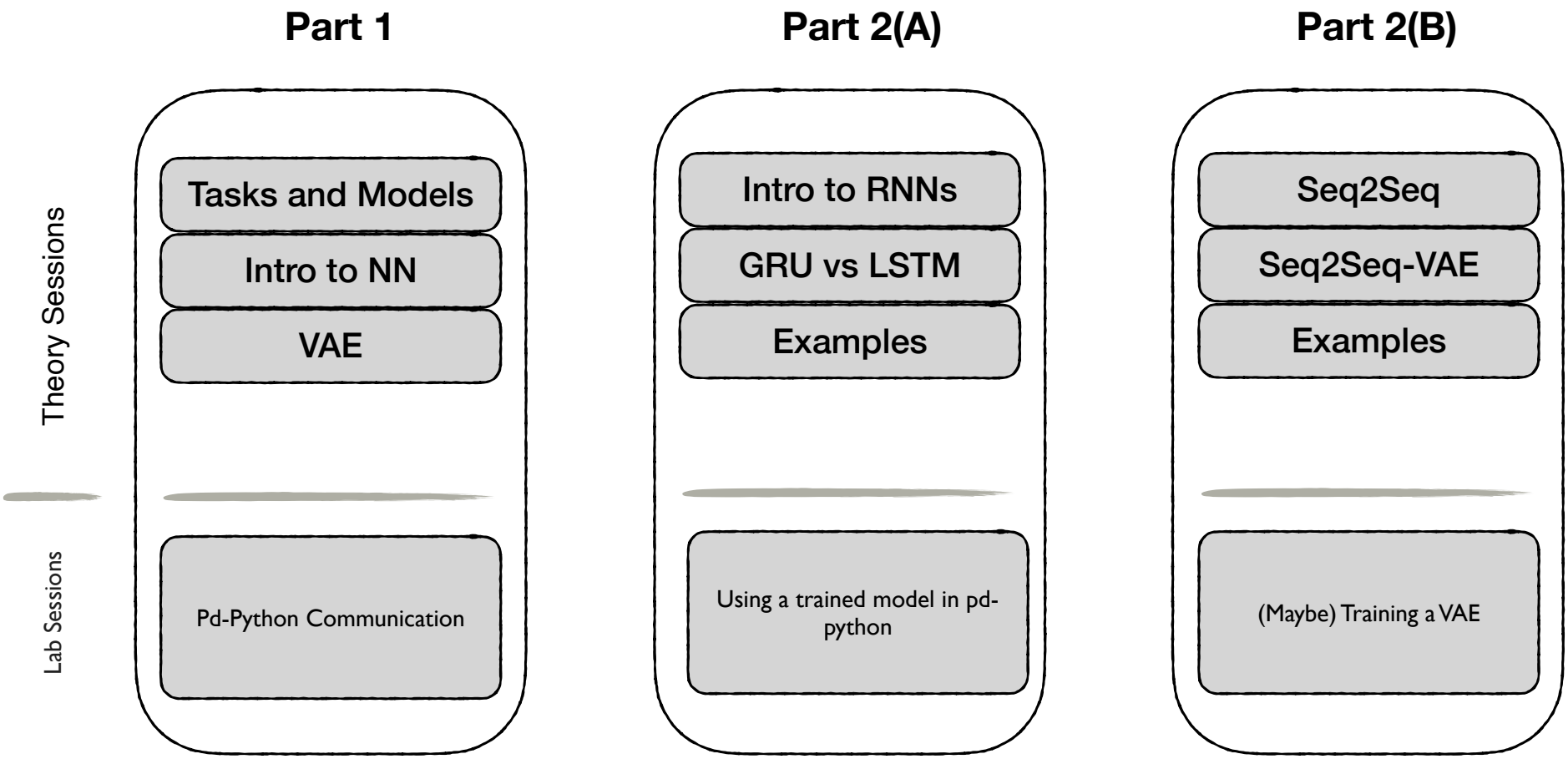
Generative Models of Music

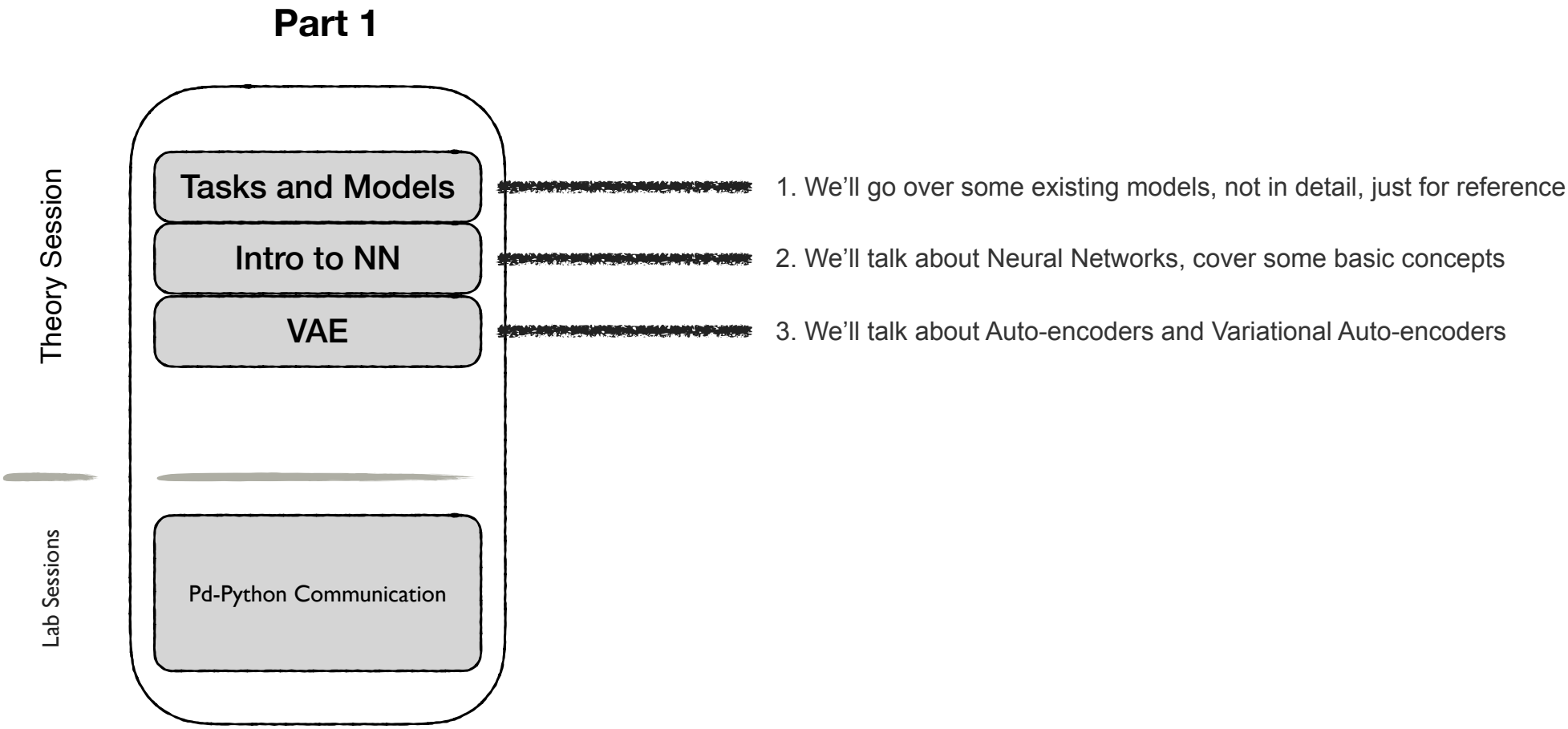
Computational Music Creativity, SMC (2021/22)

Prepared by Behzad Haki, February 2022

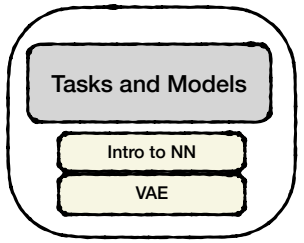
Tentative Plan

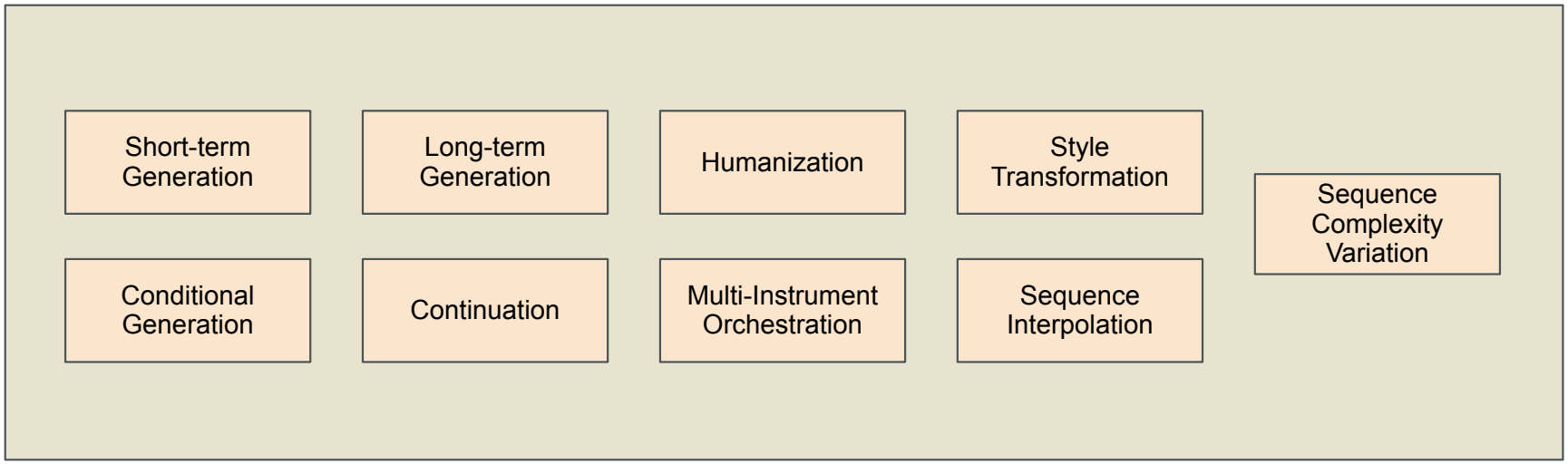


Plan for Today

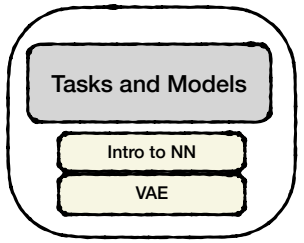


Tasks in Generative AI Music





Models in Generative AI Music



ARCHITECTURE

Seq2Seq

GANs

Transformers

Multi-Scale VQ-VAE

MODEL

MusicVAE

GrooVAE

MuseGAN

Music Transformer

LakhNess

Jukebox

TASKS ACCOMPLISHED

16-bar Generation, Continuation and Interpolation

Short-term Drum Generation, Humanization, Completion, Sequence Interpolation

4-Bar Multi-instrument Generation (Conditional and Unconditional)

Piano Generation, Continuation

Multi-instrument Generation, Continuation

Long-term Multi-instrument Generation

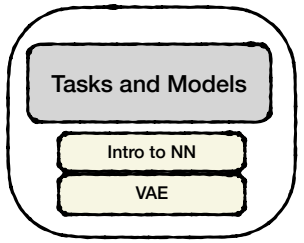
Work well for shorter sequences

More successful for longer sequences

Symbolic

Audio

Models in Generative AI Music



ARCHITECTURE

MODEL

TASKS ACCOMPLISHED

Seq2Seq	GANs	Transformers	Multi-Scale VQ-VAE
MusicVAE, GrooVAE	MuseGAN	Music Transformer, LakhNess	Jukebox
16-bar Generation, Continuation and Interpolation; Short-term Drum Generation, Humanization, Completion, Sequence Interpolation	4-Bar Multi-instrument Generation (Conditional and Unconditional)	Piano Generation, Continuation; Multi-instrument Generation, Continuation	Long-term Multi-instrument Generation

Work well for shorter sequences

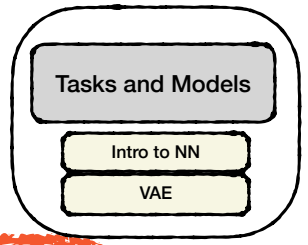
More successful for longer sequences

Symbolic

Audio

We'll be focusing on these in the next two lectures

Models in Generative AI Music



ARCHITECTURE

Seq2Seq

GANs

Transformers

Multi-Scale VQ-VAE

MODEL

MusicVAE

GrooVAE

MuseGAN

Music Transformer

LakhNess

Jukebox

TASKS ACCOMPLISHED

16-bar Generation, Continuation and Interpolation

Short-term Drum Generation, Humanization, Completion, Sequence Interpolation

4-Bar Multi-instrument Generation (Conditional and Unconditional)

Piano Generation, Continuation

Multi-instrument Generation, Continuation

Long-term Multi-instrument Generation

Work well for shorter sequences

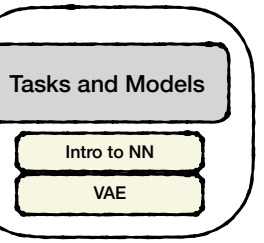
More successful for longer sequences

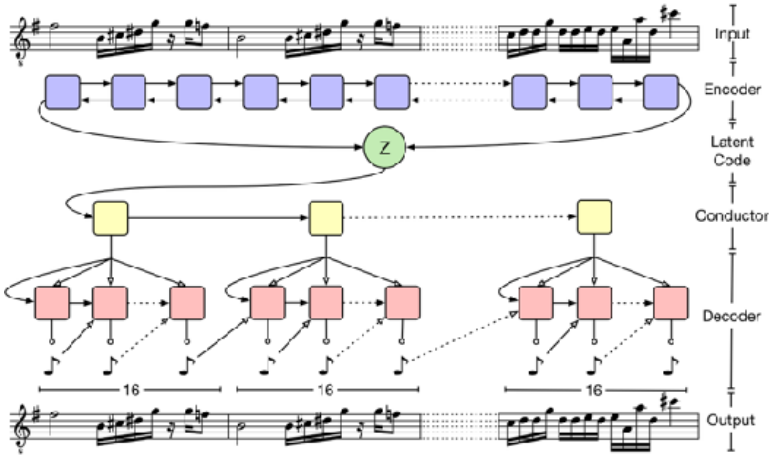
Symbolic

Audio

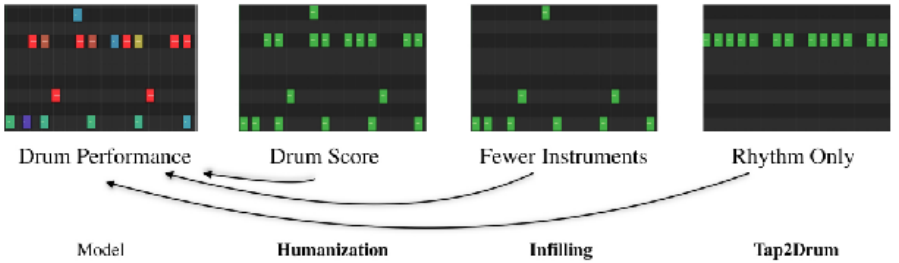
Will not be able to cover these in detail in this course

Sequence-to-Sequence Models

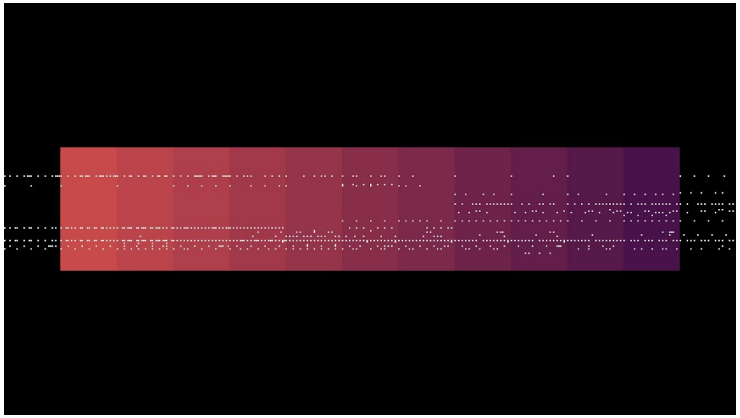




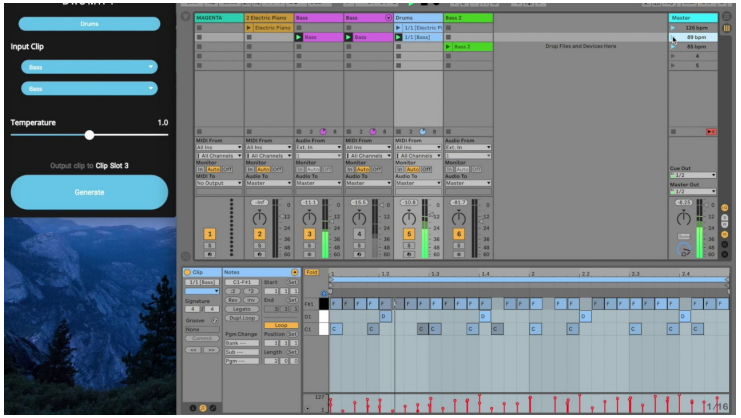
Source: https://magenta.tensorflow.org/music-vae



Gillick, Jon, et al. "Learning to groove with inverse sequence transformations." *International Conference on Machine Learning*. PMLR, 2019.

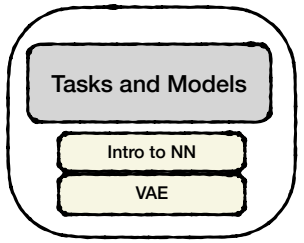


https://youtube.com/watch?v=OsxP4lFp76I



https://youtube.com/watch?v=x2YLmXzovDo

MuseGAN



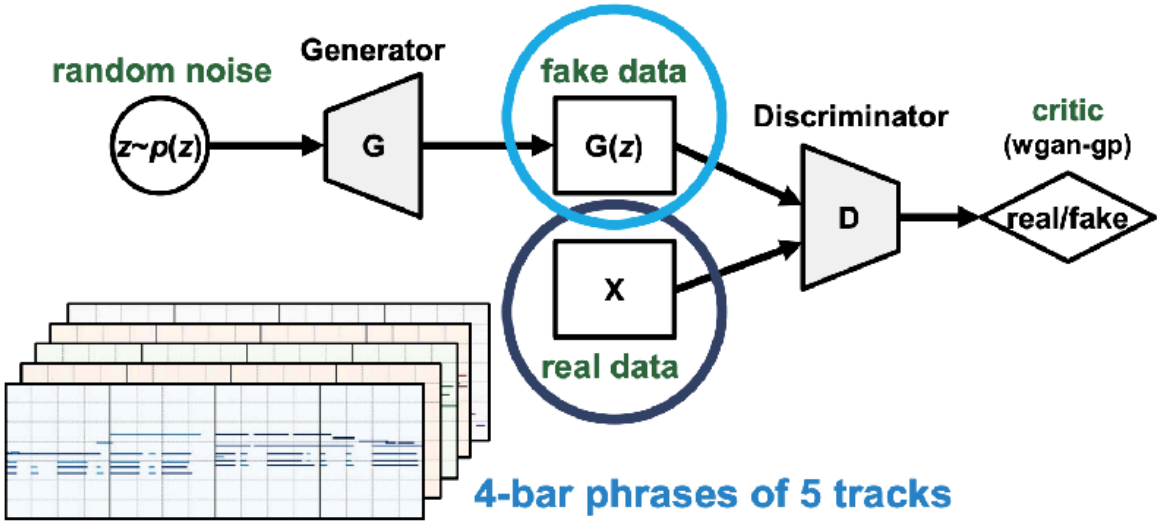
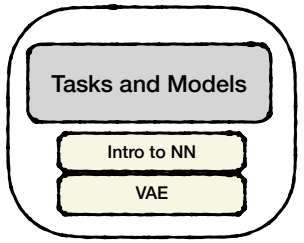
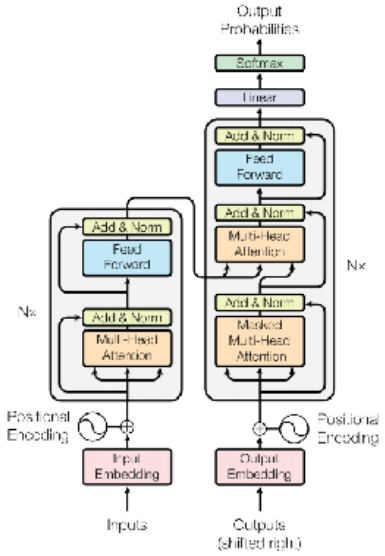


Image from https://salu133445.github.io/musegan/pdf/musegan-aaai2018-slides.pdf

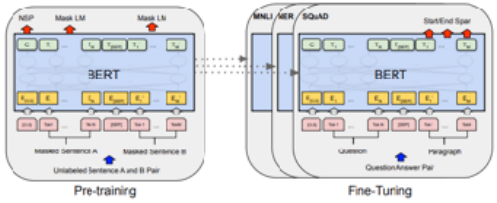
https://youtube.com/watch?v=0aN9tKNCYxM



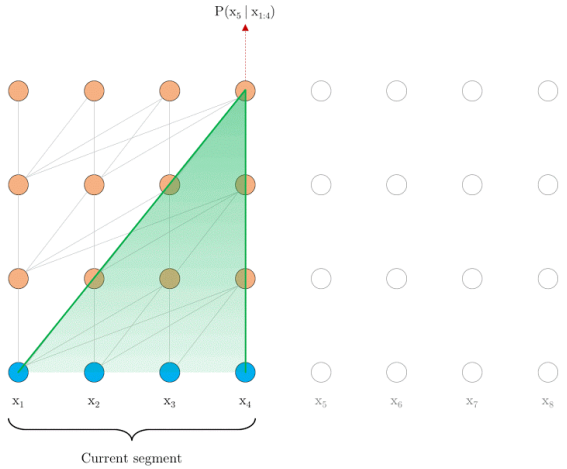
Transformers



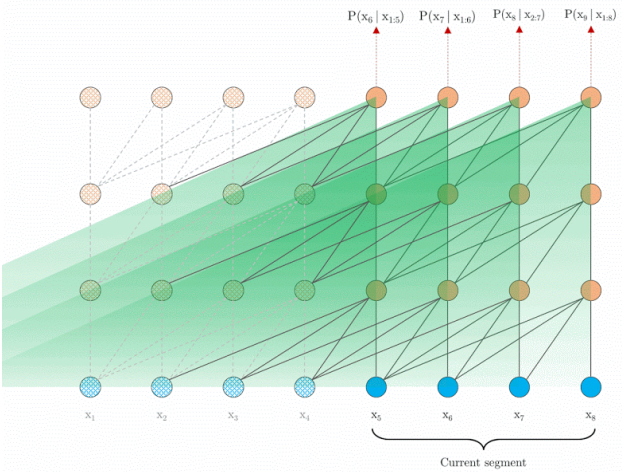
Vanilla Transformer
Source: Vaswani, Ashish, et al. "Attention is all you need." *arXiv preprintarXiv:1706.03762* (2017).



BERT: Bidirectional Encoder Representation for Language Understanding
Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

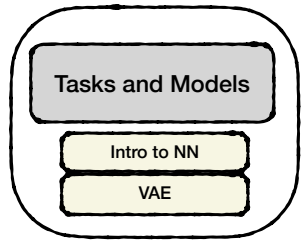


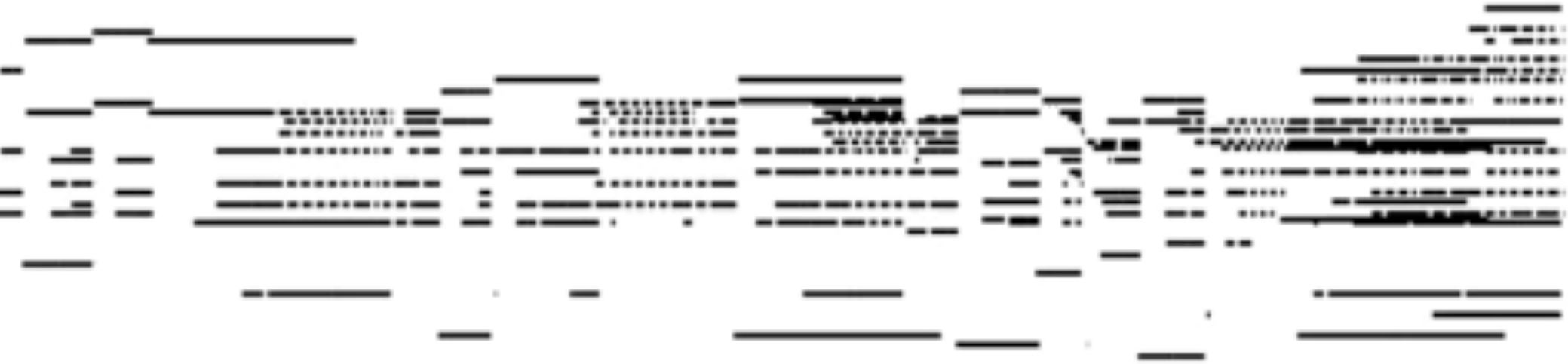
Vanilla Transformer
Source: https://ai.googleblog.com/2019/01/transformer-xl-unleashing-potential-of.html



Transformer-XL
Source: https://ai.googleblog.com/2019/01/transformer-xl-unleashing-potential-of.html

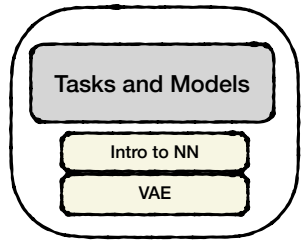
Music Transformer [1]



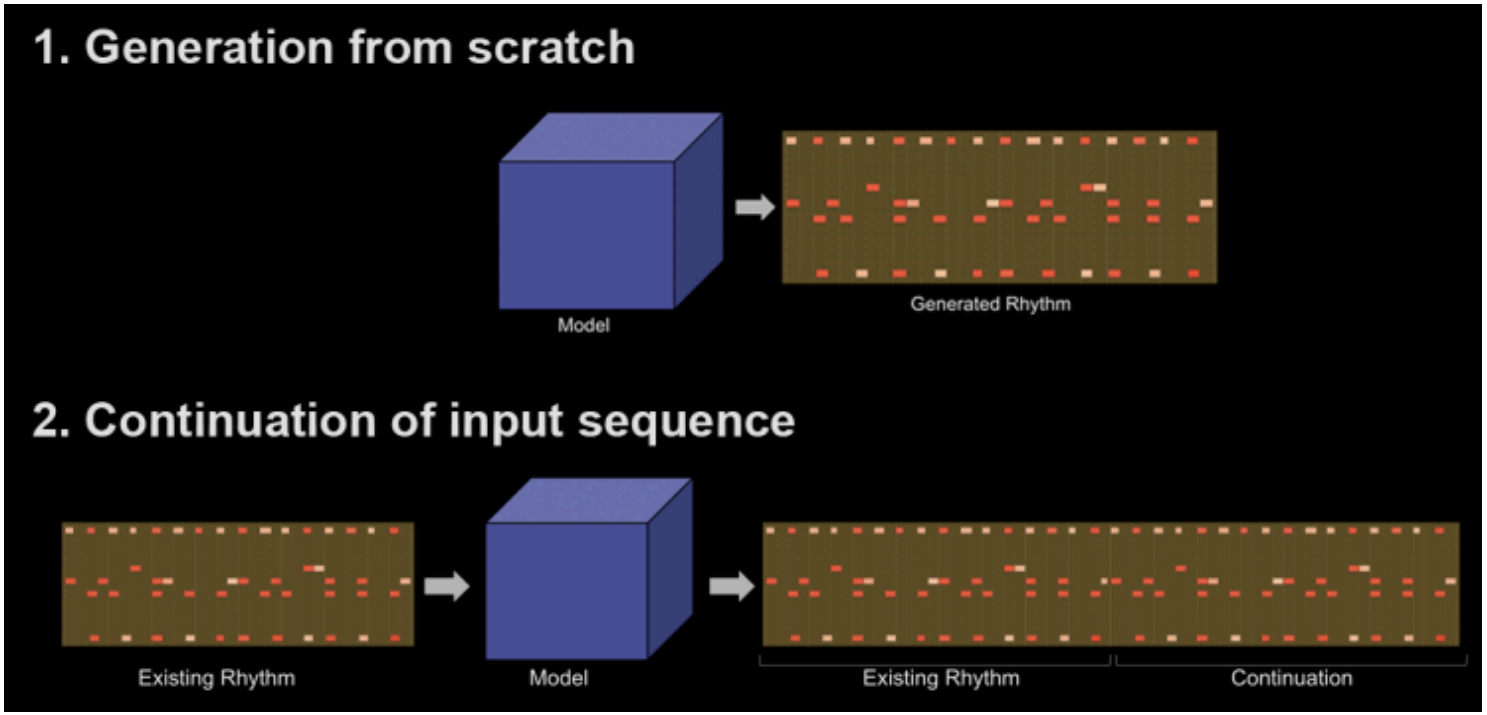


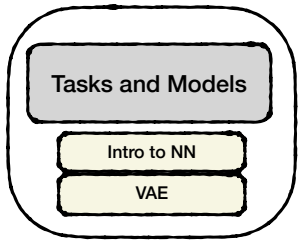
Video from https://magenta.tensorflow.org/assets/music_transformer/motifs_visualization.mp4

[1] Reference: https://magenta.tensorflow.org/music-transformer



Transformer Neural Networks for Automated Rhythm Generation (Master Thesis 2020 by Thomas Nuttall)





Transformer Architecture for Generating Short Drum Loops Given a Performed Monotonic Groove (Master Thesis 2021 by Marina Nieto)

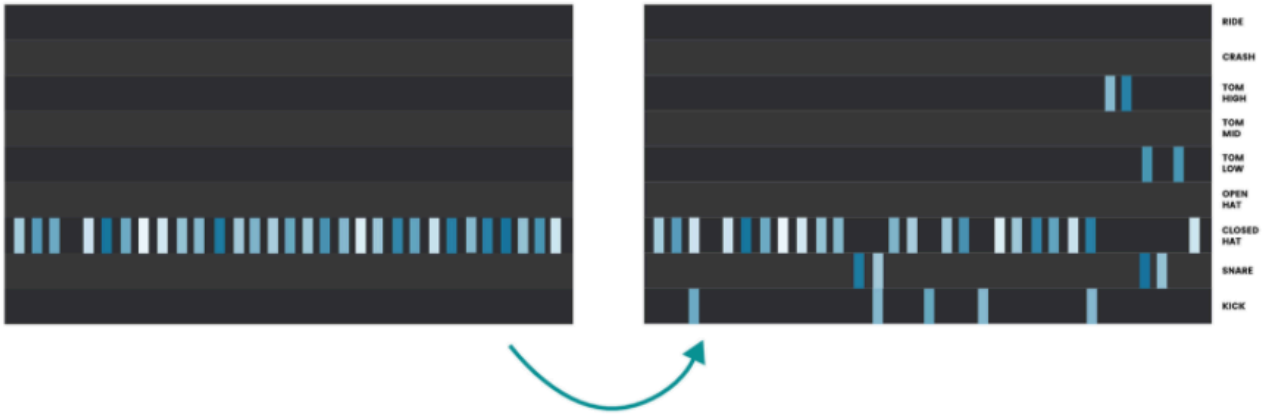
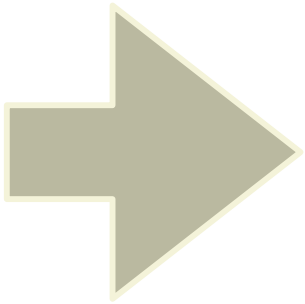


Image 1

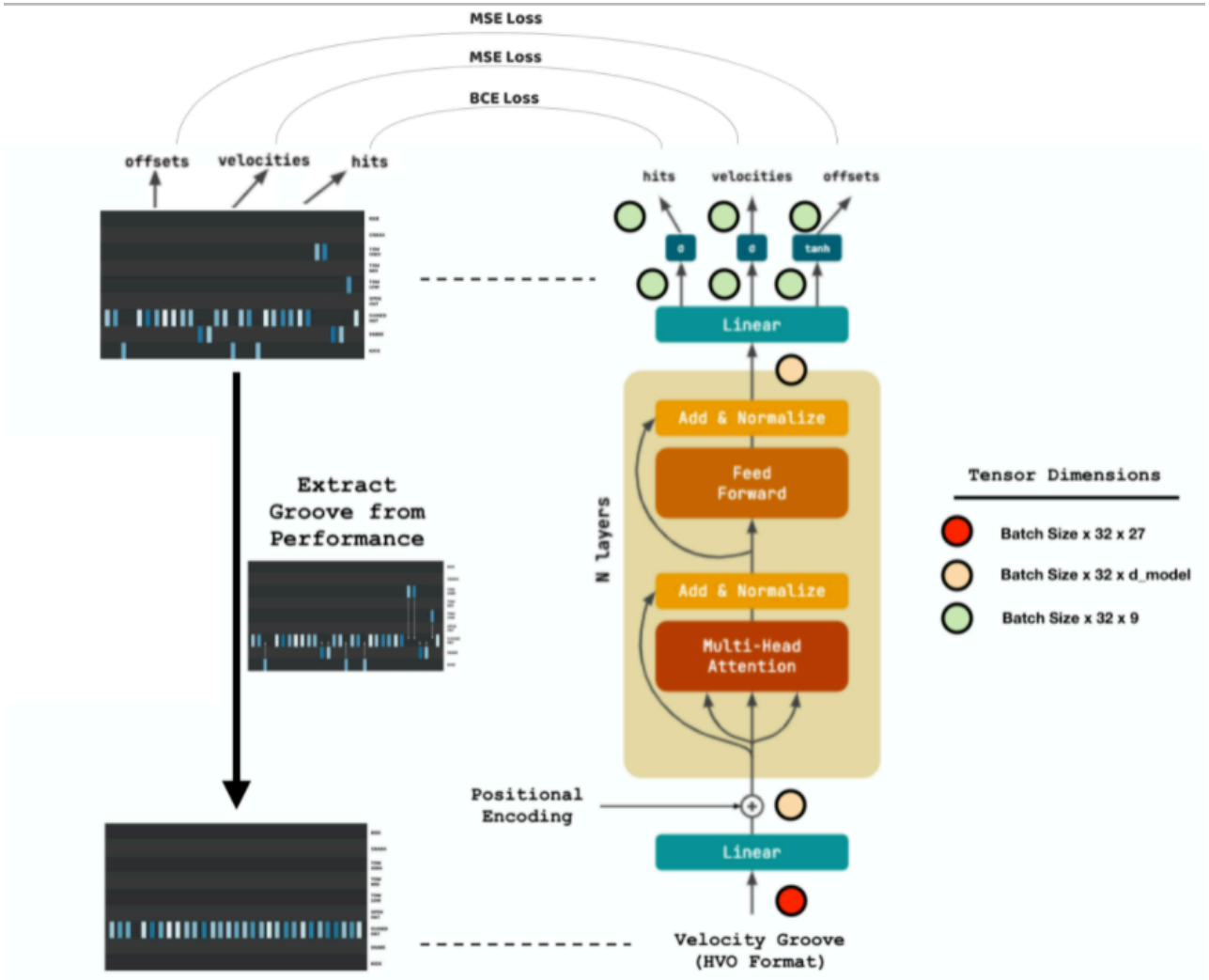
Monotonic groove performance (left piano roll) to full drum beat (right piano roll) conversion

Groove

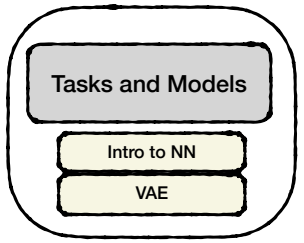
Drum Pattern Generated By the Transformer Model



Transformer Architecture for Generating Short Drum Loops Given a Performed Monotonic Groove (Master Thesis 2021 by Marina Nieto)



JukeBox [2]



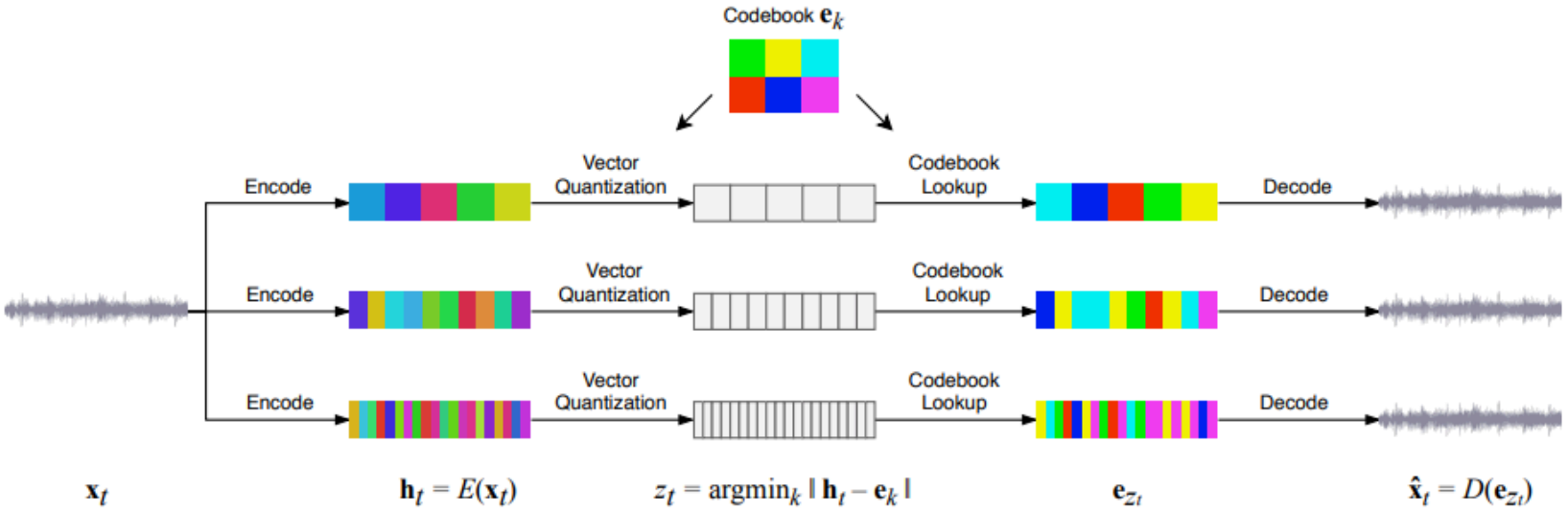
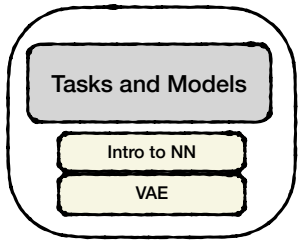


Image from [2]

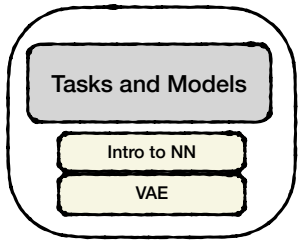
Samples Here: https://jukebox.openai.com/?song=787640146

[2] Dhariwal, Prafulla, et al. "Jukebox: A generative model for music." *arXiv preprint arXiv:2005.00341* (2020).



Introduction to Neural Nets

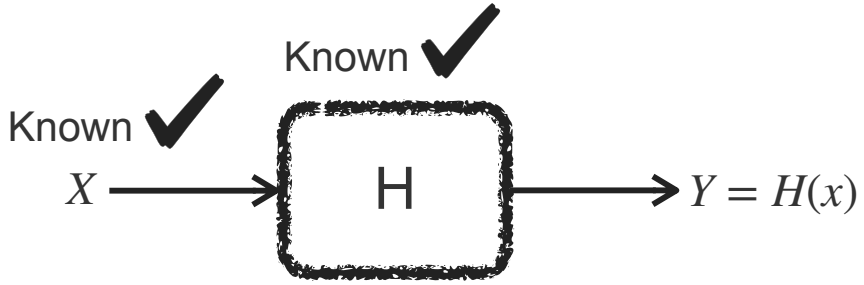
A Music Generation Perspective



What Are Neural Nets?

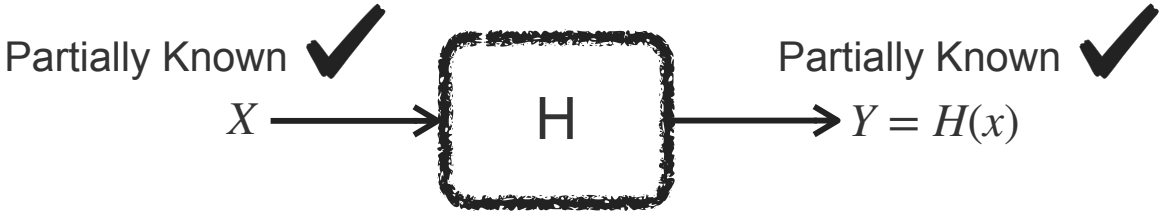
Why do we use them?

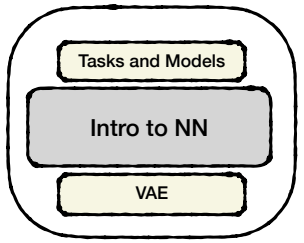
Imagine you have a system which transforms some input data X to some output Y.



If we know H, then for any new data points, we can calculate Y by passing X

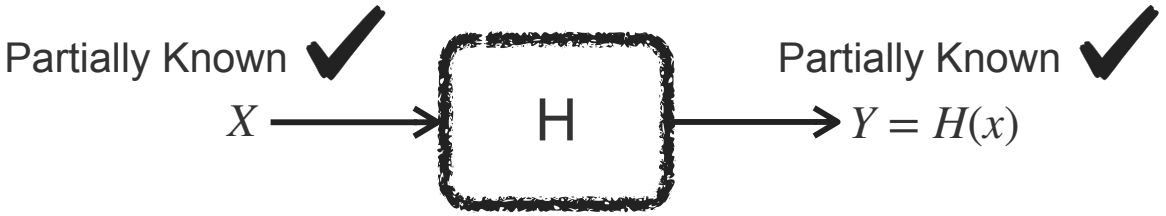
However, in reality, we usually have *observations* of X and Y, using which, we want to actually figure out what the transformation is. This is quite useful, because if we can figure out what H is based on our observations, then we can *predict the value for Y given an input value outside of our observations.*





What Are Neural Nets?

Why do we use them?

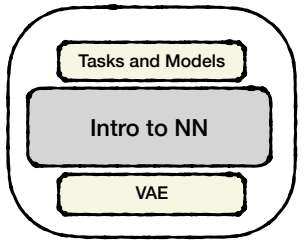


A few notes:

1. **If we only partially know X and Y, the best we can do is to come up with an approximation of H such that it works "well" on the available observations**

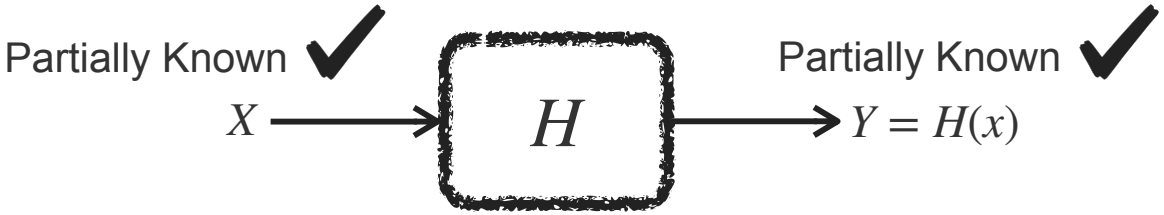
2. **No matter how much data we've observed, we can not be certain that we've observed all possible combinations of X and Y. More data may allow us to be more confident in our approximation of H, but it does not allow us to be fully confident that we know everything about the behaviour of H.**

3. **We can not be certain if Y is only dependant on X**
4. **We can not be sure that H is a deterministic transformation**



What Are Neural Nets?

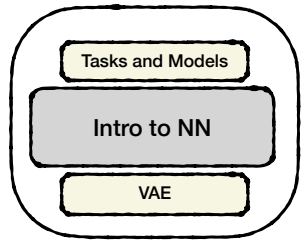
Why do we use them?



In the absence of full knowledge about X, Y and H,
All we can do is to come up with the best approximation of H,
given all available observations

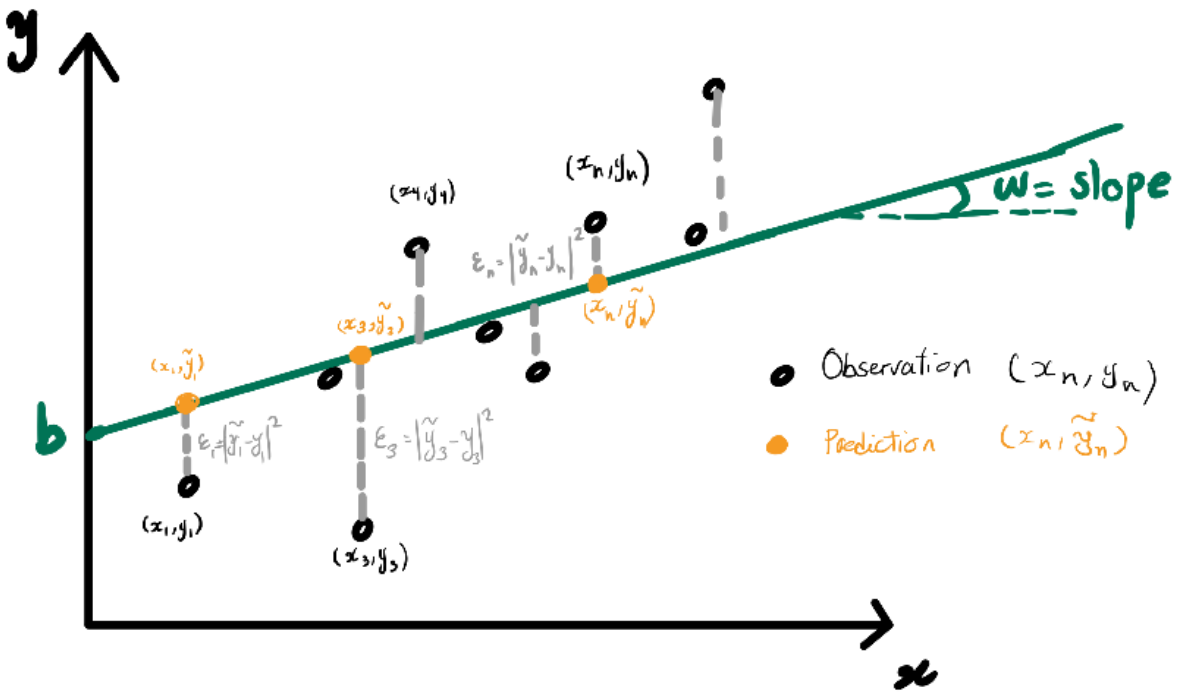
Objective: Based on observations of X and Y, find a reasonable approximation for H

How can we do this?



What Are Neural Nets?

How can we approximate the X to Y transformation?



Step 1 Draw a random line

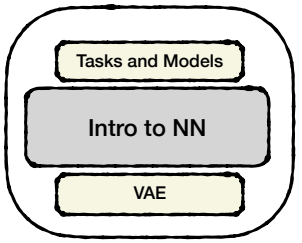
Step 2 Calculate error (ε) between predictions and observations

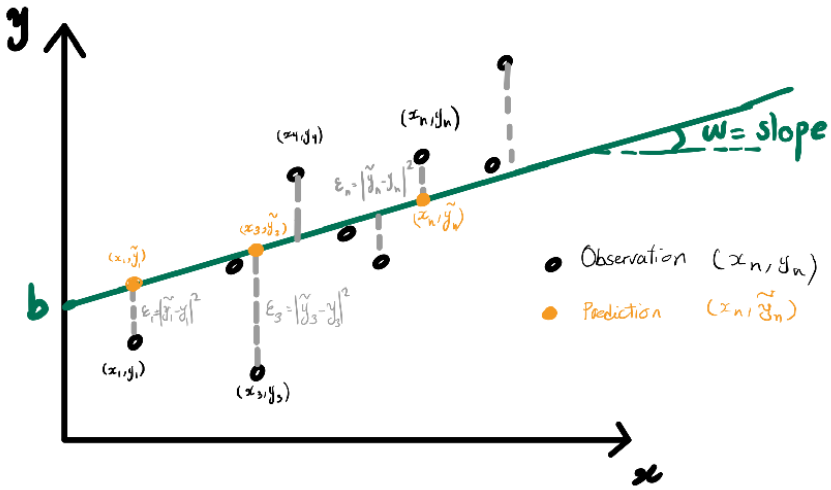
Step 3 Move line up/down slightly and/or increase/decrease slope in a direction that total of errors (ε_n's) decreases slightly

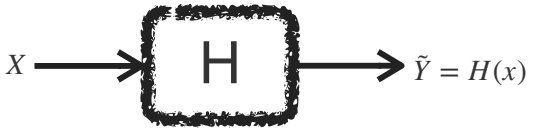
Step 4 Keep repeating steps 2 and 3 until the total error stabalizes

What Are Neural Nets?

How can we approximate the X to Y transformation?







$$J_{w,b}(H) = 1/m \sum_{i=1}^{m} (y_i - \tilde{y})^2 = 1/m \sum_{i=1}^{m} (y_i - H(x_i))^2$$

Initialize/Randomize parameters of H

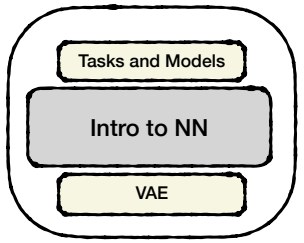
→

Find $J_{w,b}(H)$
(distance/loss between the actual observation (Y) and the prediction ($\tilde{Y}$))

→

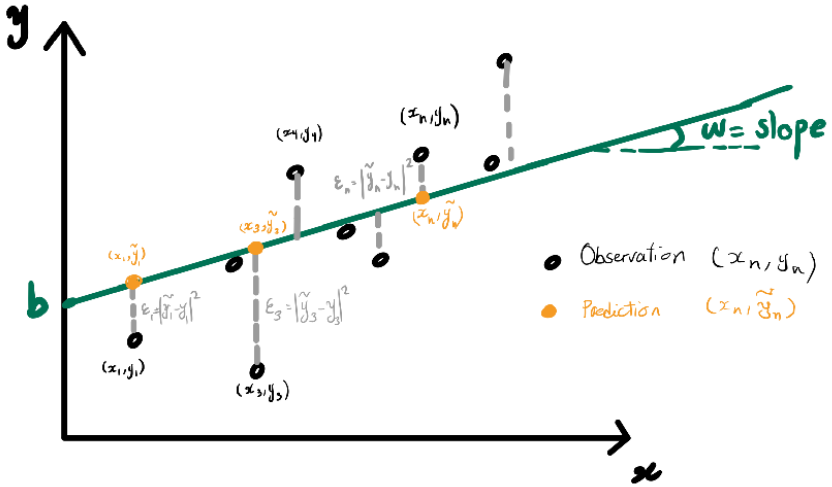
Slightly adjust the parameters of H such that the distance/loss reduces slightly

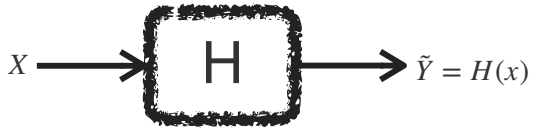
Keep Repeating Until Distance/Loss Stabilizes

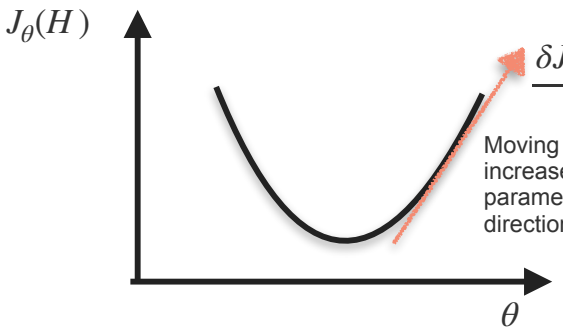


What Are Neural Nets?

How can we approximate the X to Y transformation?







$$\frac{\delta J_\theta(H)}{\delta\theta}$$

Moving along gradient, increases loss, so move the parameters in the opposite direction of the gradient

We can do this by inspecting the gradient of $J_{w,b}(H)$

$$J_{w,b}(H) = 1/m \sum_{i=1}^{m} (y_i - \tilde{y})^2 = 1/m \sum_{i=1}^{m} (y_i - H(x_i))^2$$

Initialize/Randomize parameters of H

Find $J_{w,b}(H)$

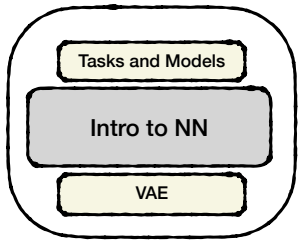
(distance/loss between the actual observation (Y) and the prediction ($\tilde{Y}$))

Slightly adjust the parameters of H such that the distance/loss reduces slightly

Keep Repeating Until Distance/Loss Stabilizes

What Are Neural Nets?

How can we approximate the X to Y transformation?

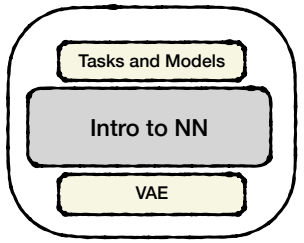


But what if we're dealing with high dimensional data
And much more complex transformations?

Turns out Neural Networks are quite useful for such cases, because
NNs can be used to created highly complex non-linear operations

What Are Neural Nets?

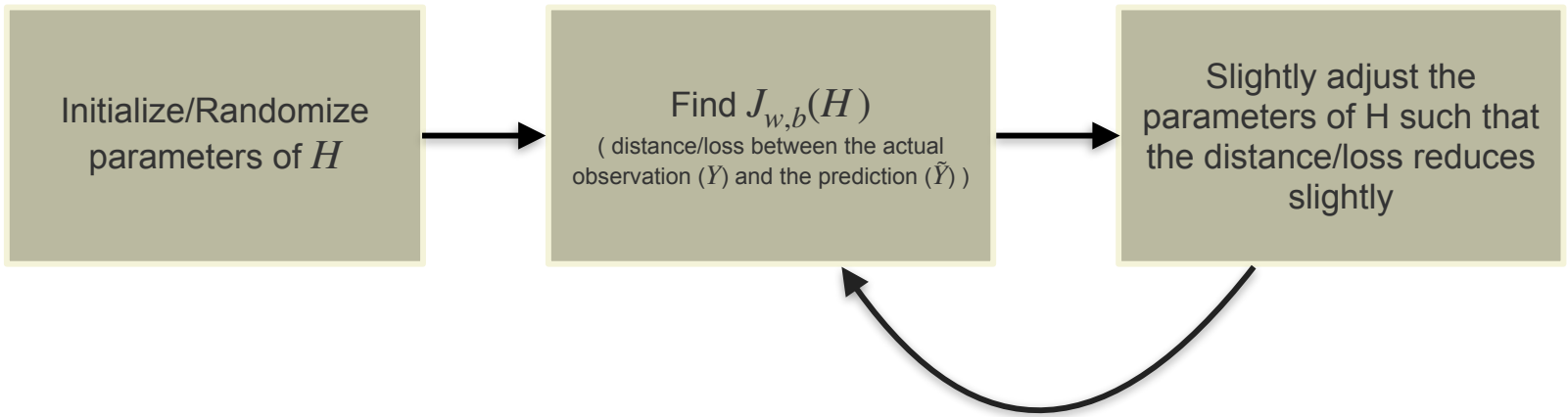
How can we approximate the X to Y transformation?

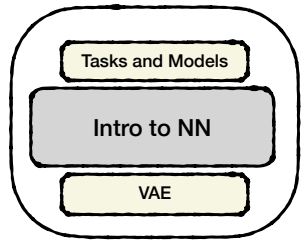


But what if we're dealing with high dimensional data
And much more complex transformations?

Turns out Neural Networks are quite useful for such cases, because NNs can be used to created highly complex non-linear operations.

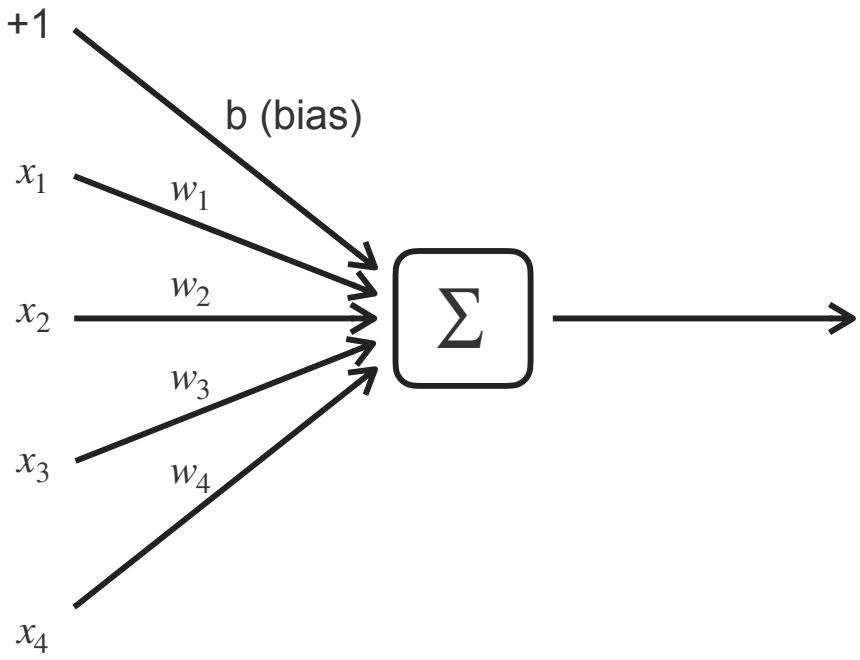
The following still applies



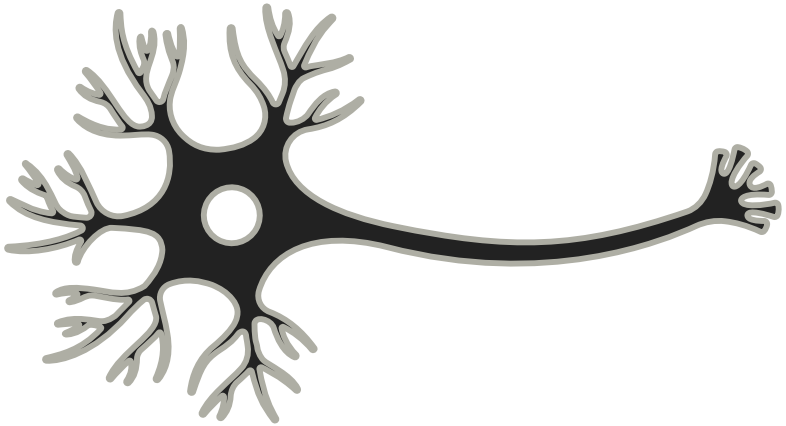


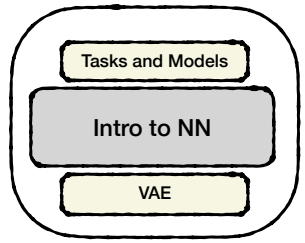
Neural Networks

A Single Neuron

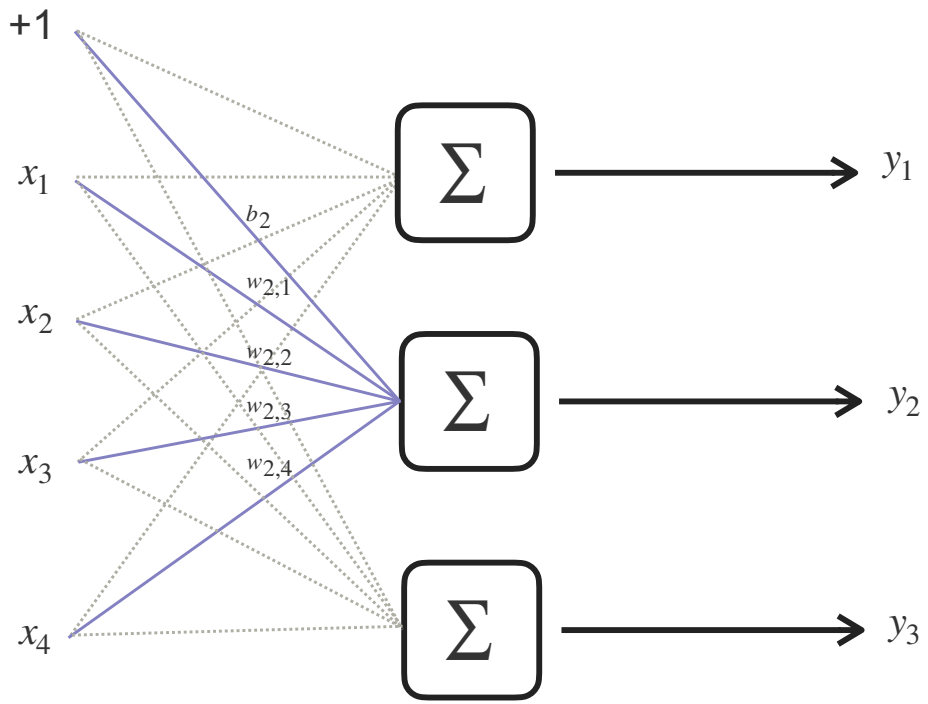


$$y = Wx + b = [w_1 \quad w_2 \quad w_3 \quad w_4] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + b$$

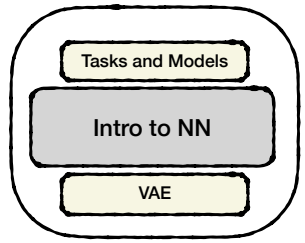




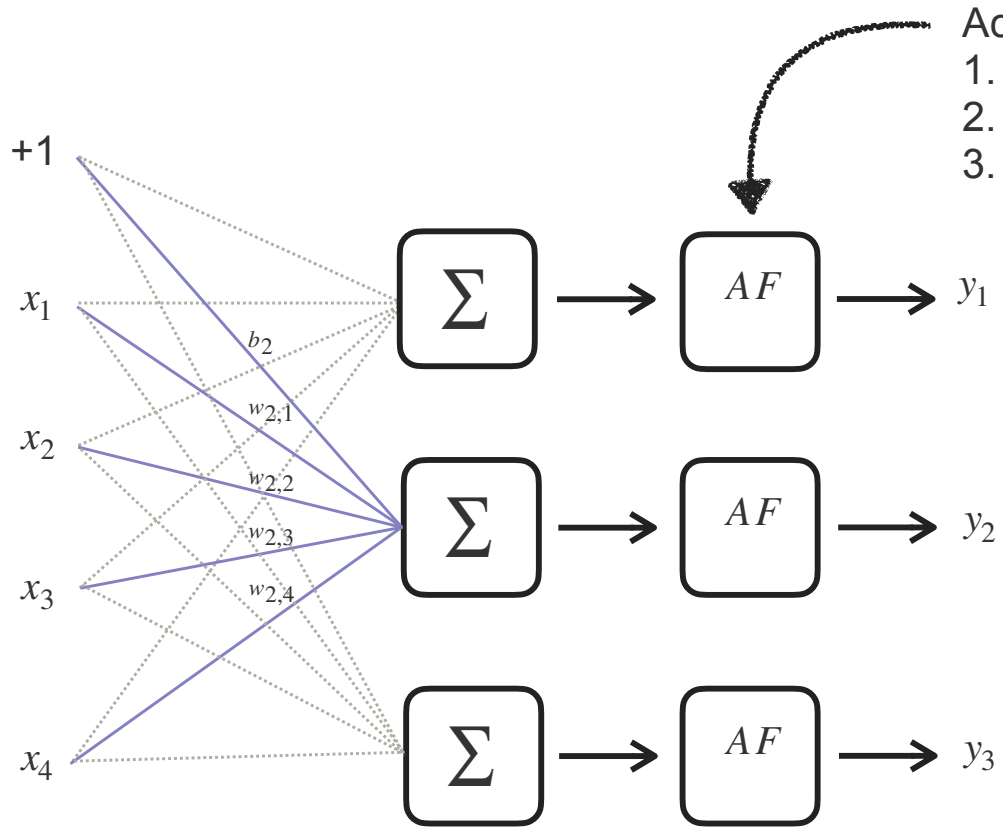
Neural Networks
Multiple Neurons



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = Wx + b = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

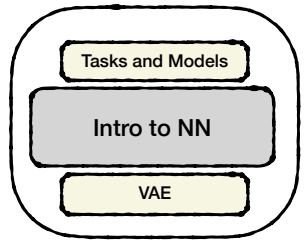


Neural Networks
Multiple Neurons

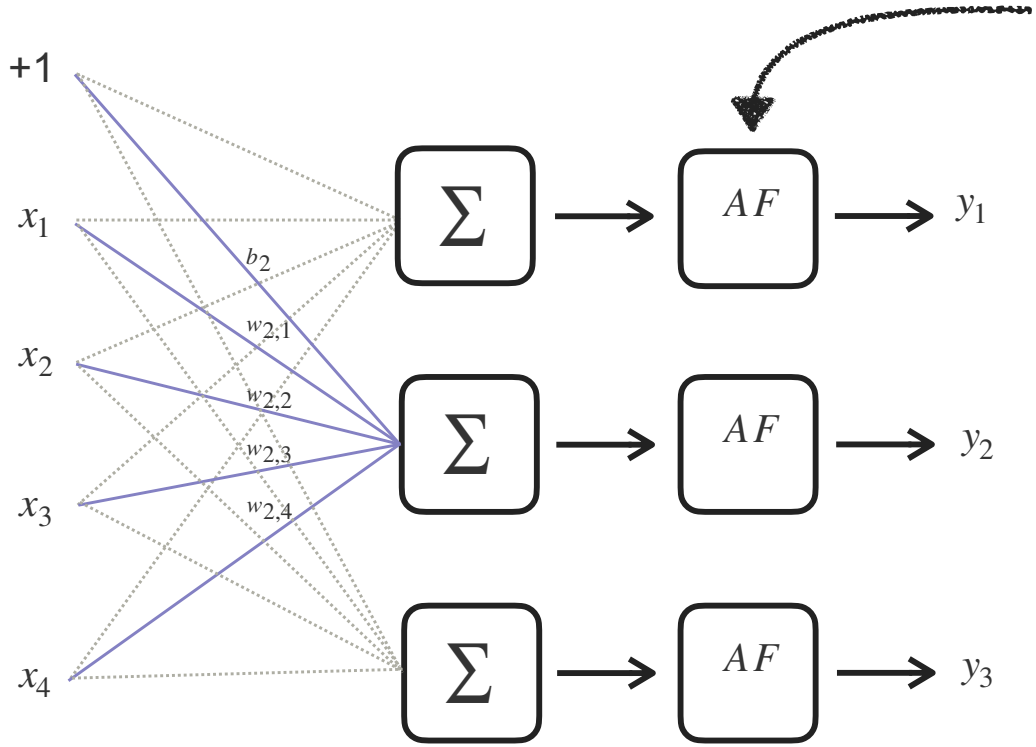


Activation Function, used to:
1. Add Nonlinearity,
2. Activate/Deactivate neuron,
3. Specify the type of output (probability vs. Scalar)

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = AF(Wx + b) = AF\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$



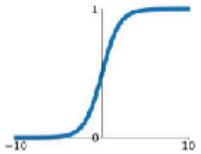
Neural Networks
Multiple Neurons



Activation Functions

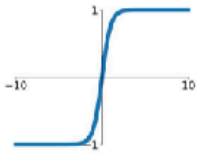
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



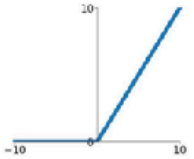
tanh

$$\tanh(x)$$

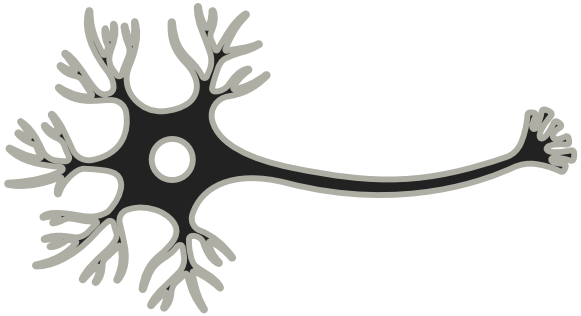


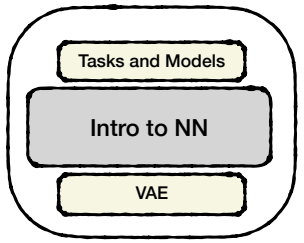
ReLU

$$\max(0, x)$$



Reference: https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

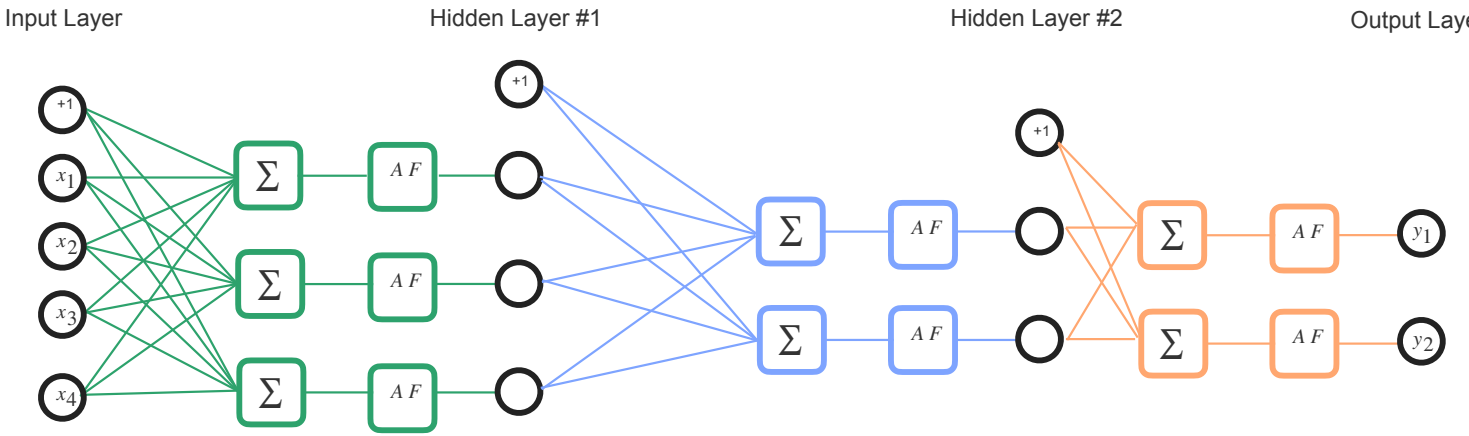




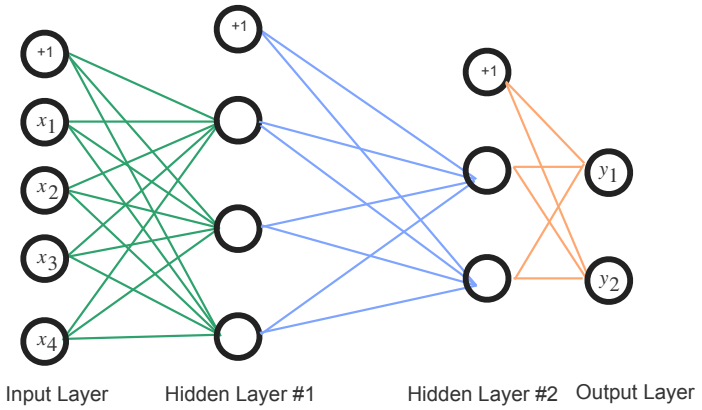
Neural Networks

Multiple Layers (Different Ways of Visualization)

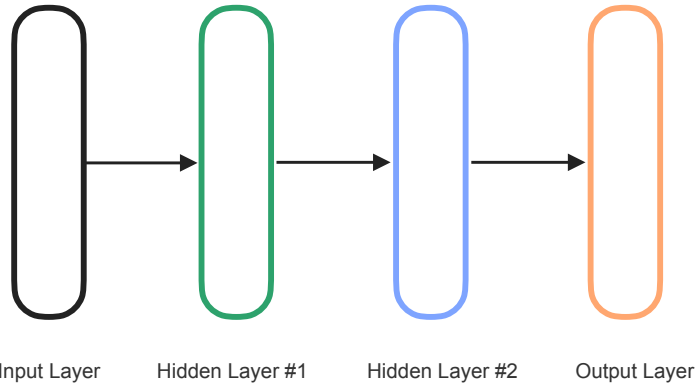
Detailed Representation

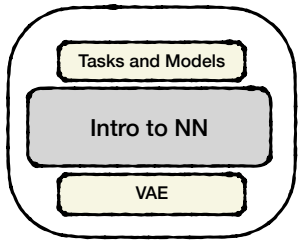


Simplified Representation



Abstract Representation

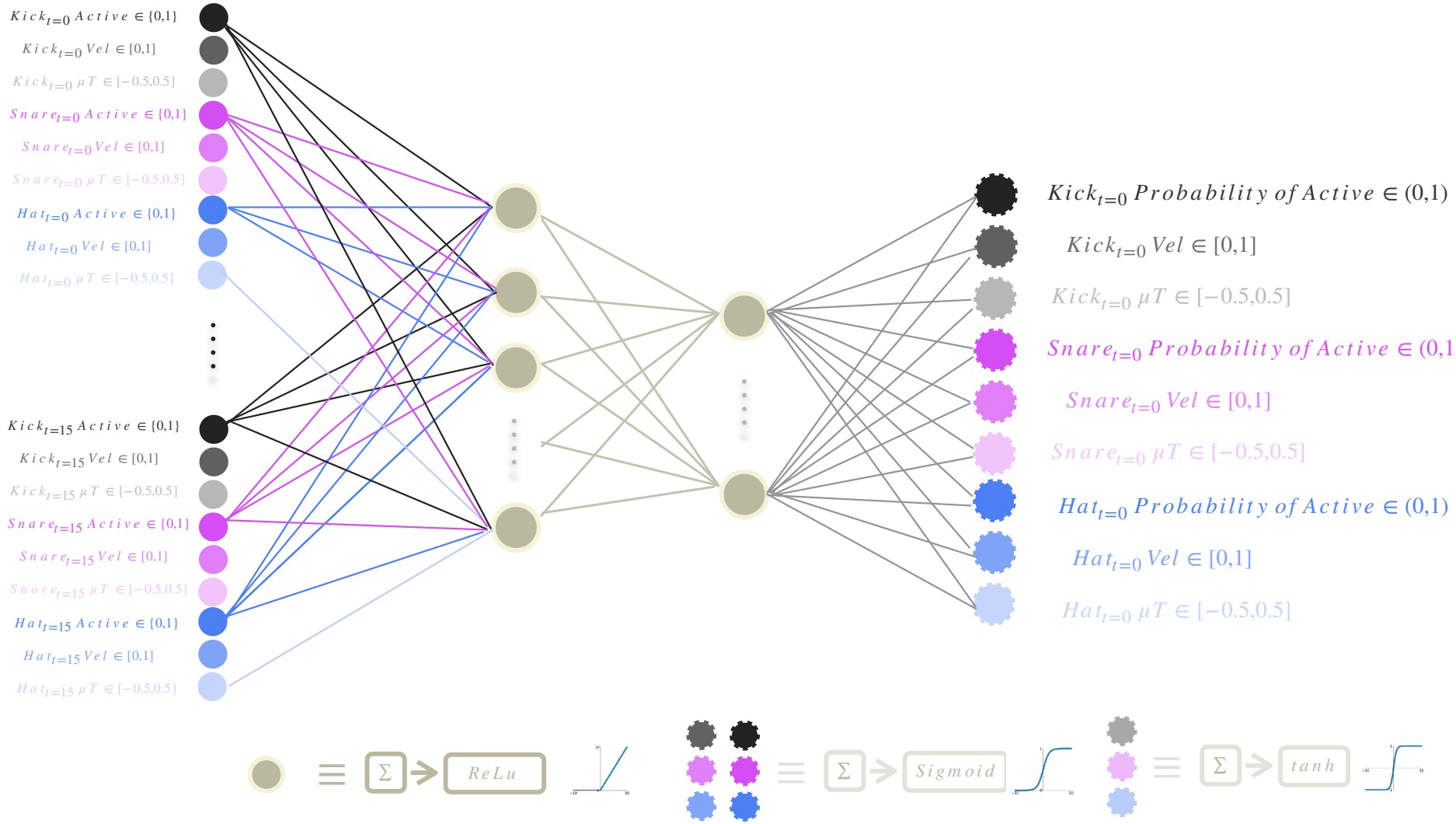


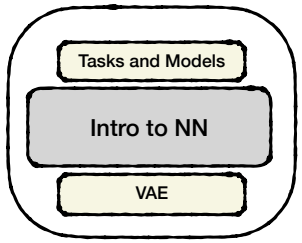


Music Generation Using Feedforward Networks

Example 1

Given a 1 bar drum sequence, predict the events at the next time-step
(16th note quantized, with uTiming, velocity and 4/4 meter)

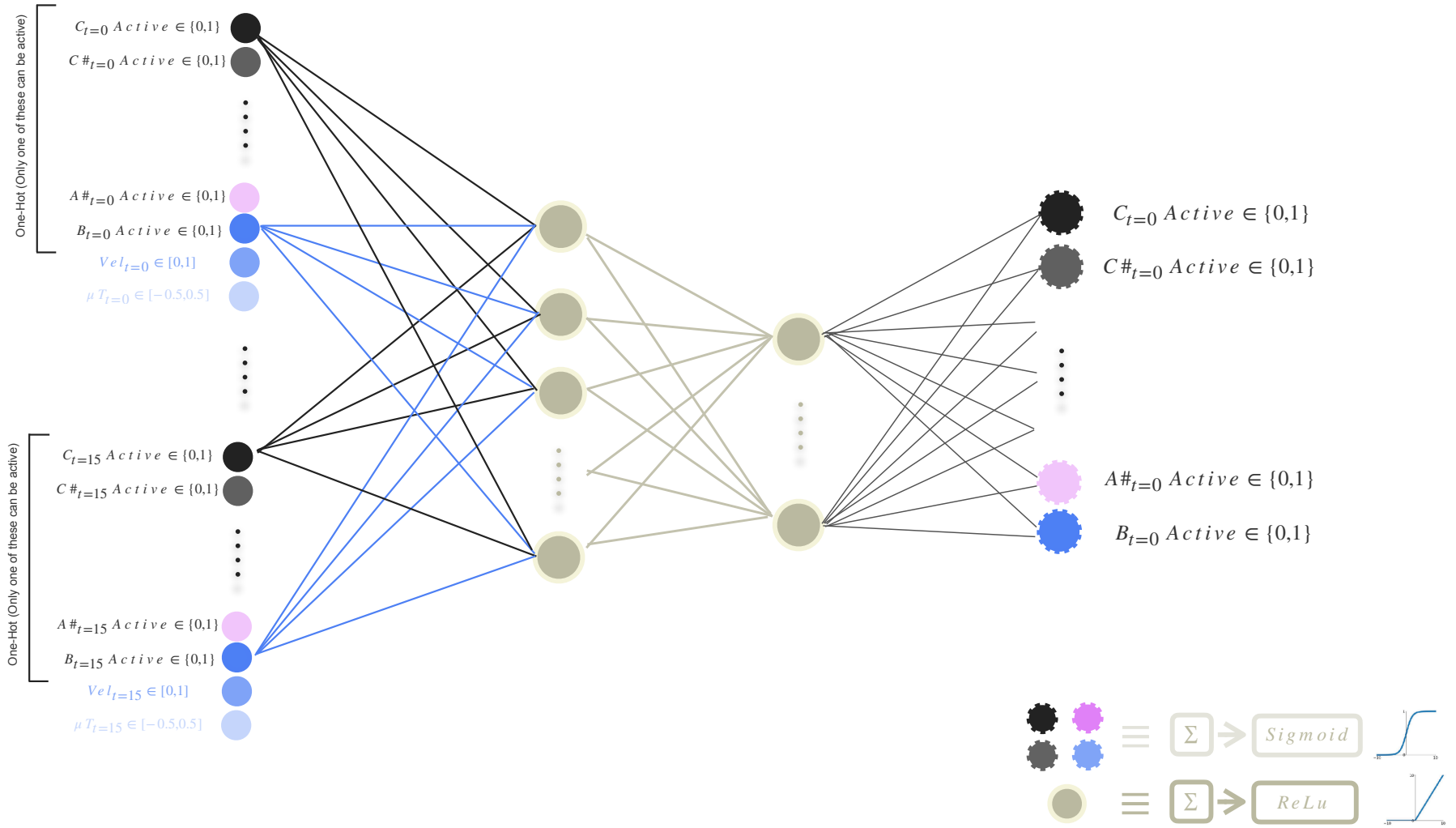


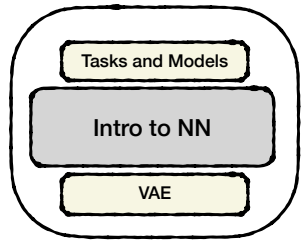


Music Generation Using Feedforward Networks

Example 2

Predict a chord for a given 1 bar Melodic phrase
(16th note quantized, with uTiming, velocity and 4/4 meter)

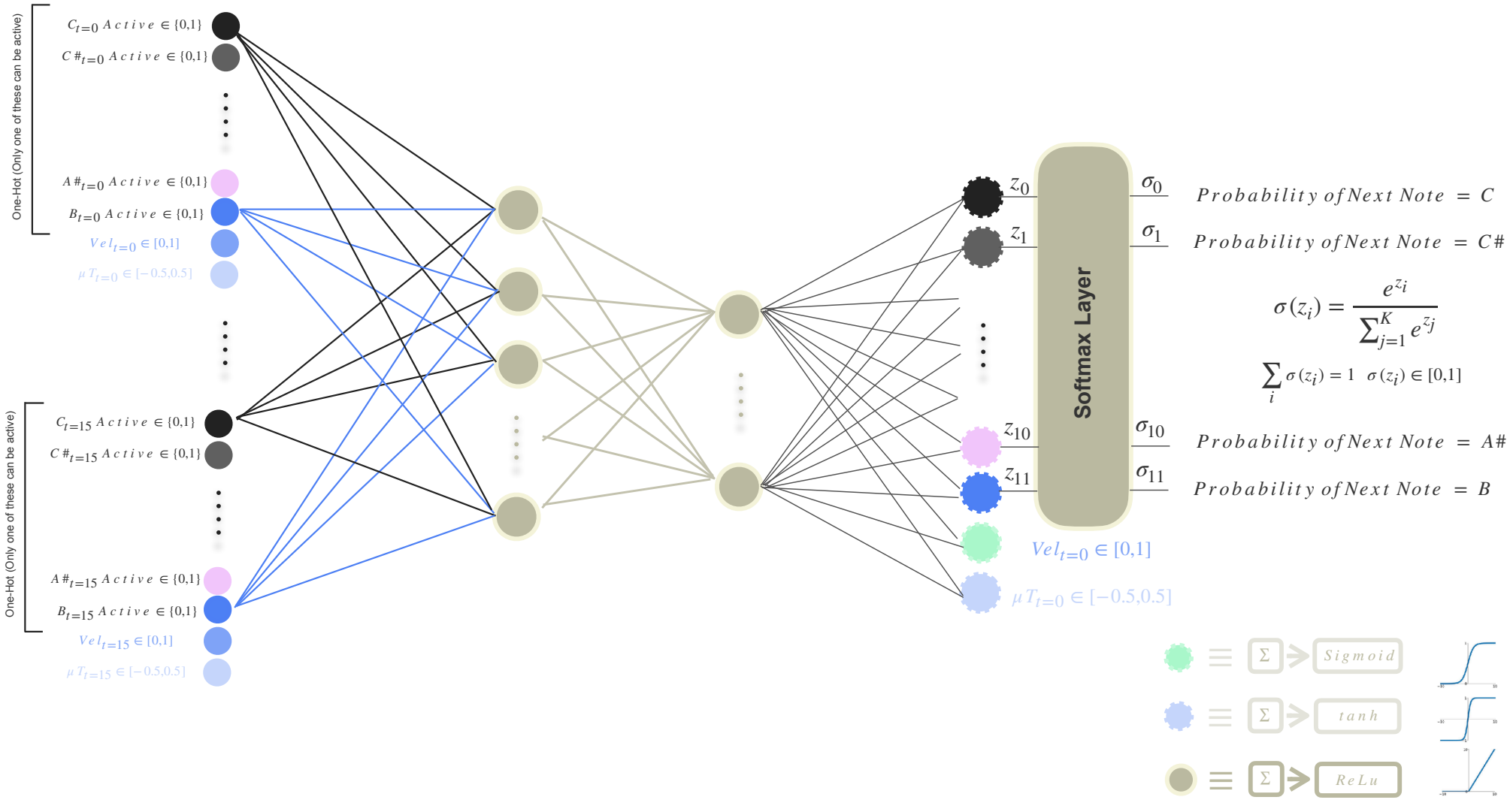


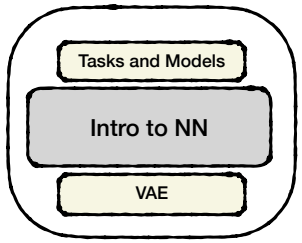


Music Generation Using Feedforward Networks

Example 3

Given a 1 bar Melodic phrase, predict the the next note
(16th note quantized, with uTiming, velocity and 4/4 meter)

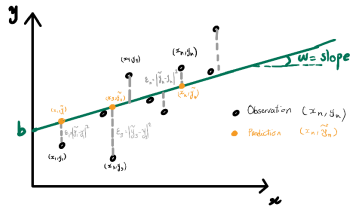




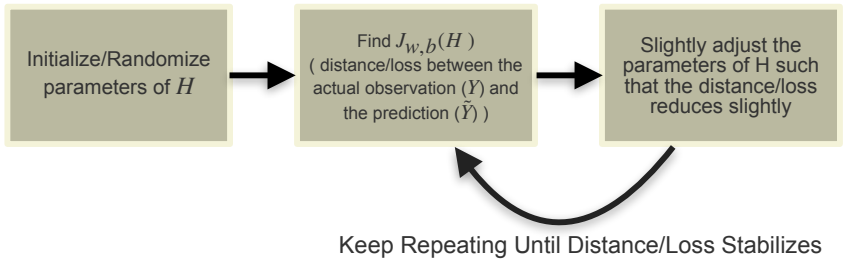
Music Generation Using Feedforward Networks

Cost Functions

Remember the following example?

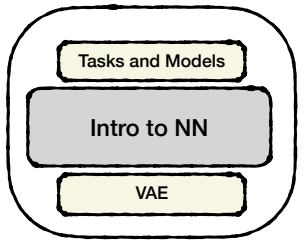


$$J_{w,b}(H) = 1/m \sum_{i=1}^{m} (y_i - \bar{y})^2 = 1/m \sum_{i=1}^{m} (y_i - H(x_i))^2$$



$$J_{\theta(w,b)} = Cost\ Function$$

The criterion used for evaluating the closeness of the predictions to a target → We use different cost (loss) functions for different cases

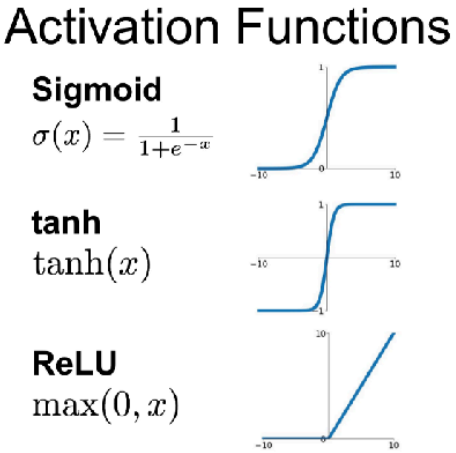


Music Generation Using Feedforward Networks

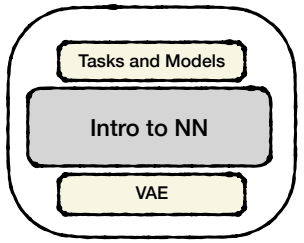
Cost Functions

Task	Output Type	Target Encoding	Output Activations	Cost (loss)
Regression	Real	Real Values, or a subset of real values	Linear (Identity), ReLu, Sigmoid, Tanh	MSE (Mean Squared Error)
Classification	Binary	{0, 1}	Sigmoid	Binary cross-entropy
Classification	Multi-class Single Label	One-hot	Softmax	Categorical cross-entropy
Classification	Multi-class Multi-label	Many-hot	Sigmoid	Binary cross-entropy

Table from [1]



[1] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. *Deep learning techniques for music generation*. Springer, 2020. Page 72



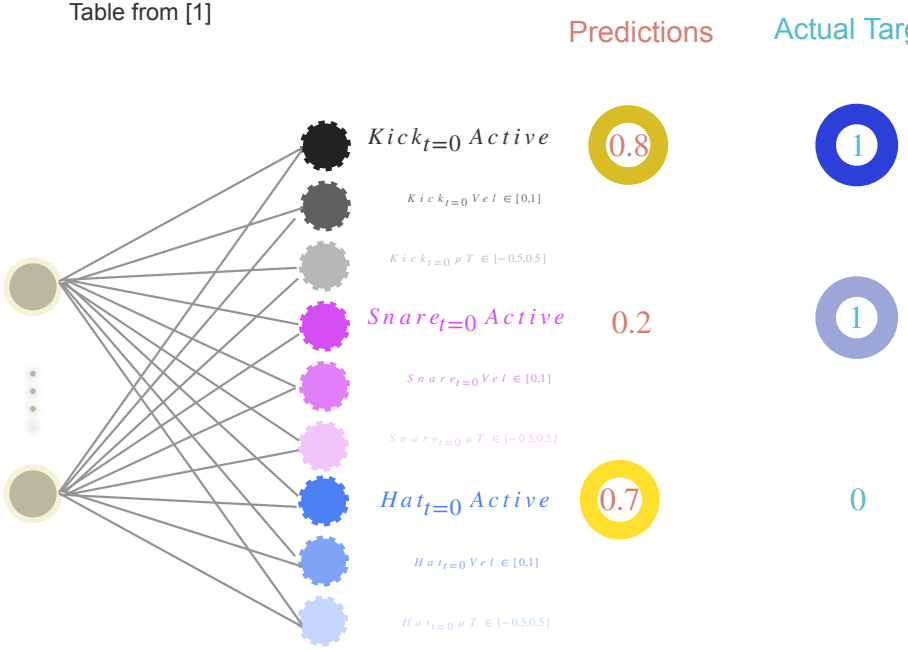
Music Generation Using Feedforward Networks

Cost Functions (Binary Cross Entropy)

Task	Output Type	Target Encoding	Output Activations	Cost (loss)
Regression	Real	Real Values, or a subset of real values	Linear (Identity), ReLu, Sigmoid, Tanh	MSE (Mean Squared Error)
Classification	Binary	{0, 1}	Sigmoid	Binary cross-entropy
Classification	Multi-class Single Label	One-hot	Softmax	Categorical cross-entropy
Classification	Multi-class Multi-label	Many-hot	Sigmoid	Binary cross-entropy

Table from [1]

Monophonic Output (Example 1)
Multiple notes allowed to be played
This means that the probability of a note occurring should be measured individually



For the predictions, we'll assume that probabilities above 0.5 mean an active note. Hence, the model is predicting that we have a **Kick** and **Hat** coming up while the real data (target) shows that in fact we should have a **Kick** and **Snare** coming up.

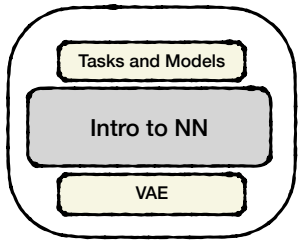
Let's find how far are the predictions from the targets

It doesn't make sense to find the euclidean distance between Actual targets and the predictions. Hence, we use a probabilistic criterion for the distance between predictions and the targets. In this case, we find the losses for kick, snare and hat predictions individually and then add them up. Each each individual loss can be calculated using Binary Cross Entropy Loss (BCE):

$$BCE = -(target_i \log(prediction_i) + (1-target_i) \log((1-prediction_i))$$

$$where\ target_i \in \{0,1\}\ and\ prediction_i \in (0,1)$$

[1] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. *Deep learning techniques for music generation*. Springer, 2020. Page 72



Music Generation Using Feedforward Networks

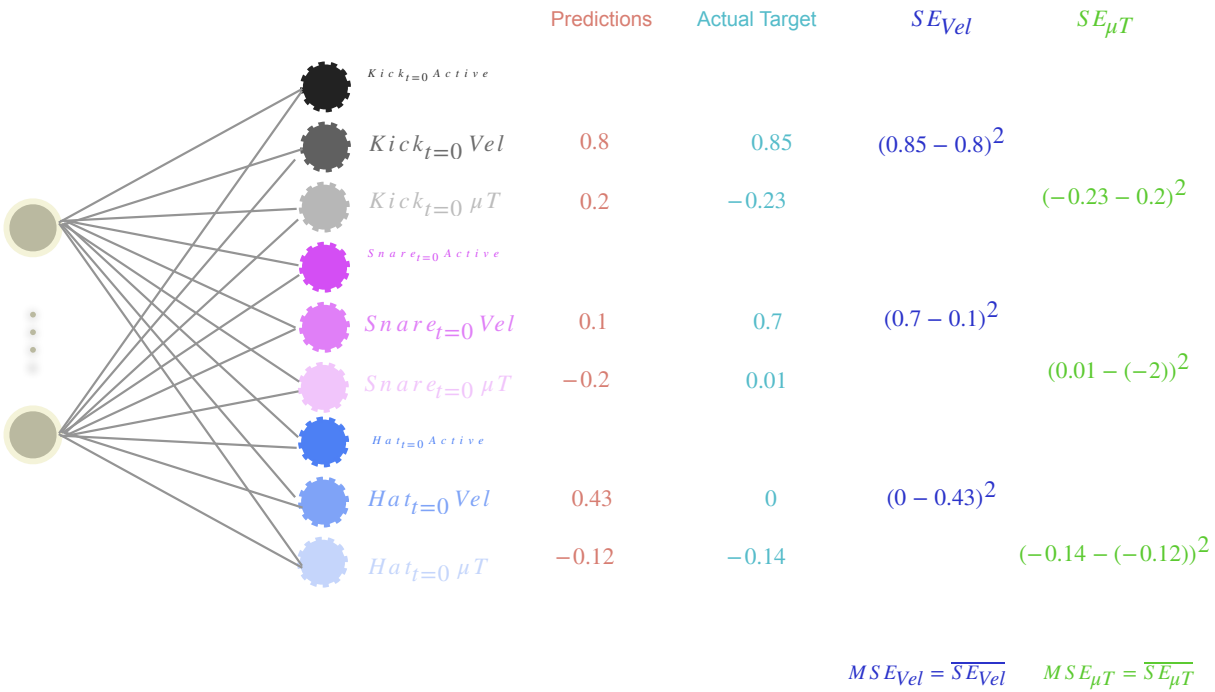
Cost Functions (Mean Squared Error)

Task	Output Type	Target Encoding	Output Activations	Cost (loss)
Regression	Real	Real Values, or a subset of real values	Linear (Identity), ReLu, Sigmoid, Tanh	MSE (Mean Squared Error)
Classification	Binary	{0, 1}	Sigmoid	Binary cross-entropy
Classification	Multi-class Single Label	One-hot	Softmax	Categorical cross-entropy
Classification	Multi-class Multi-label	Many-hot	Sigmoid	Binary cross-entropy

Table from [1]

Real Valued Outputs (Example 1-3)
Velocity and Micro-Timings are real valued predictions

$\mu T \in [-0.5,0.5]$, $Vel \in [0,1]$



This case is similar to the linear regression example we saw earlier

Let's find how far are the predictions from the targets

Here, the euclidean distance between Actual targets and the predictions can be used as the criterion for evaluating the "closeness" of the predictions to the targets. This loss is called Mean Squared Error Loss (MSE):

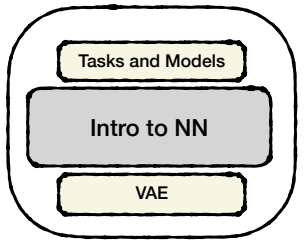
$$MSE_{Vel} = \frac{1}{n}\sum_{i=0}^{\#\,Voices-1}(target\ velocity_i - predicted\ velocity_i)^2$$

$$where\ target\ velocity_i \in [0,1]\ and\ predicted\ velocity_i \in (0,1)$$

$$MSE_{\mu T} = \frac{1}{n}\sum_{i=0}^{\#\,Voices-1}(target\ \mu T_i - predicted\ \mu T_i)^2$$

$$where\ target\ \mu T_i \in [-0.5,0.5]\ and\ predicted\ \mu T_i \in (-0.5,0.5)$$

[1] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. *Deep learning techniques for music generation*. Springer, 2020. Page 72



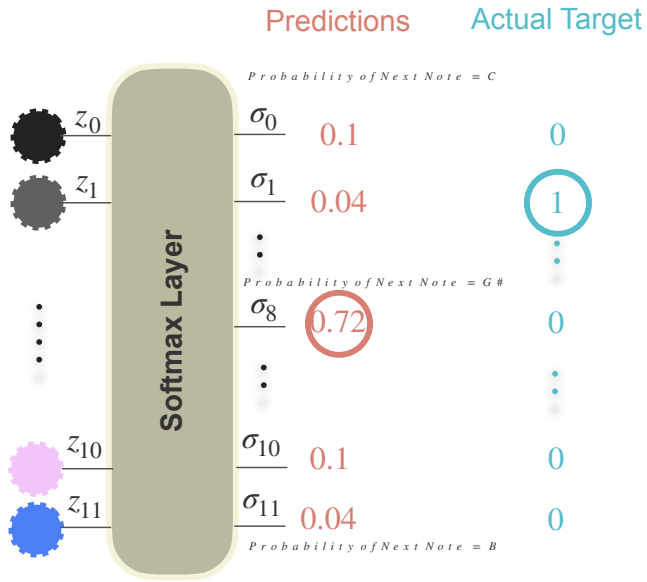
Music Generation Using Feedforward Networks

Cost Functions (Categorical Cross-Entropy)

Task	Output Type	Target Encoding	Output Activations	Cost (loss)
Regression	Real	Real Values, or a subset of real values	Linear (Identity), ReLu, Sigmoid, Tanh	MSE (Mean Squared Error)
Classification	Binary	{0, 1}	Sigmoid	Binary cross-entropy
Classification	Multi-class Single Label	One-hot	Softmax	Categorical cross-entropy
Classification	Multi-class Multi-label	Many-hot	Sigmoid	Binary cross-entropy

Monophonic Output (Example 3)
Only one note can be played from all available pitch classes

Table from [1]



$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad \sum_i \sigma(z_i) = 1 \quad \sigma(z_i) \in [0,1]$$

σ_8 is the largest probability in the predictions. So, the model predicts the next note to be G#. However, the actual target (from real data) shows that the next note should be C#.

Let's find how far are the predictions from the targets

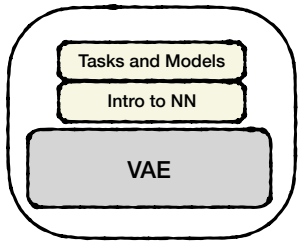
It doesn't make sense to find the euclidean distance between Actual targets and the predictions. Hence, we use a probabilistic criterion for the distance between predictions and the targets. The total loss can be calculated using Categorical Cross-Entropy Loss (CCE):

$$CCE = -\sum_{i=0}^{N-1} target_i \, log(1 - prediction_i)$$

$$where \; target_i \in \{0,1\} \; and \; prediction_i \in (0,1)$$

[1] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. *Deep learning techniques for music generation*. Springer, 2020. Page 72

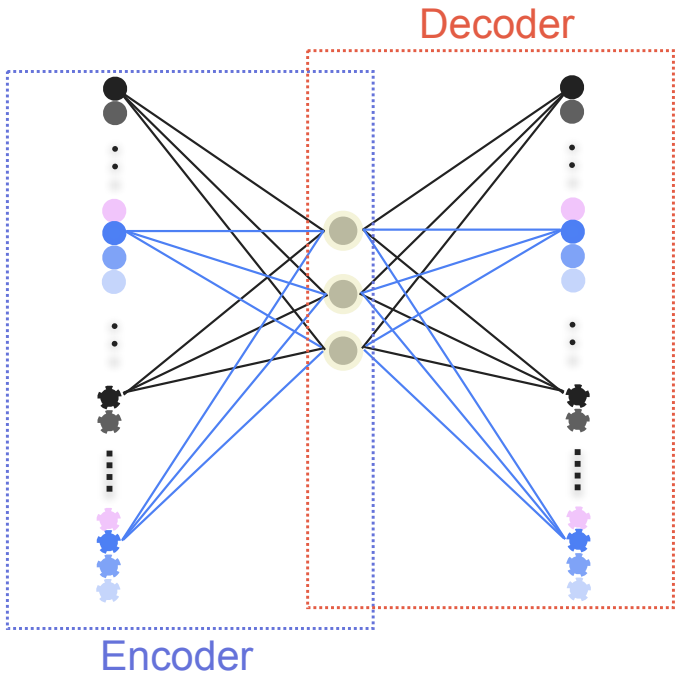
Auto-Encoders



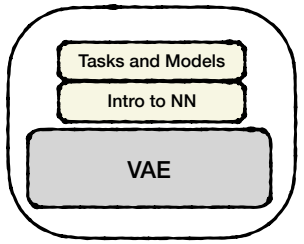
So far, we've been dealing with examples in which we know what the inputs and outputs are (this is called **Supervised Learning**).

However, in many cases, we have some data for which we want to just extract patterns. In other words, we may deal with cases in which we don't aim to map an input to an output, rather we try to find patterns existing in the input (this is called **unsupervised Learning**). Many music generation tasks fall in this category!

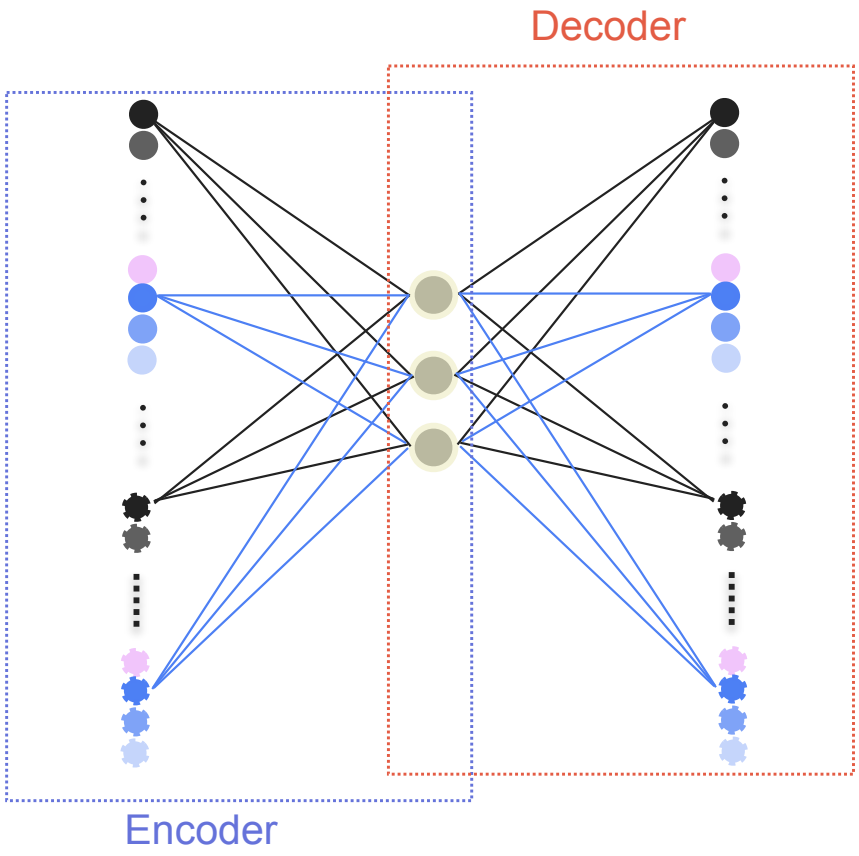
Auto-Encoders are one of the most common architectures for unsupervised learning



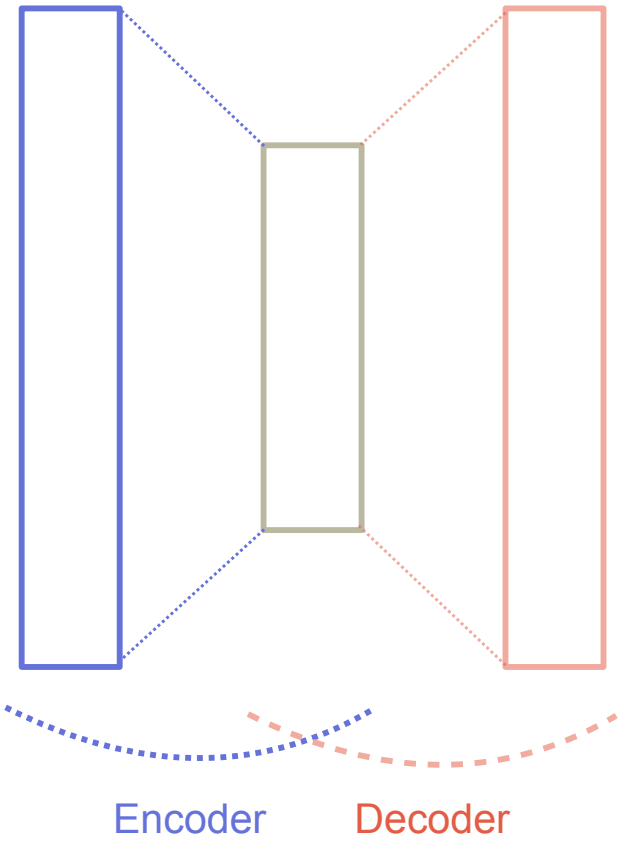
Auto-Encoders



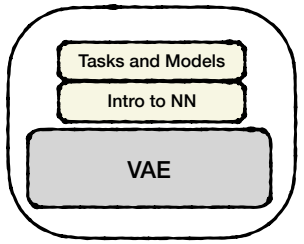
Simplified Representation

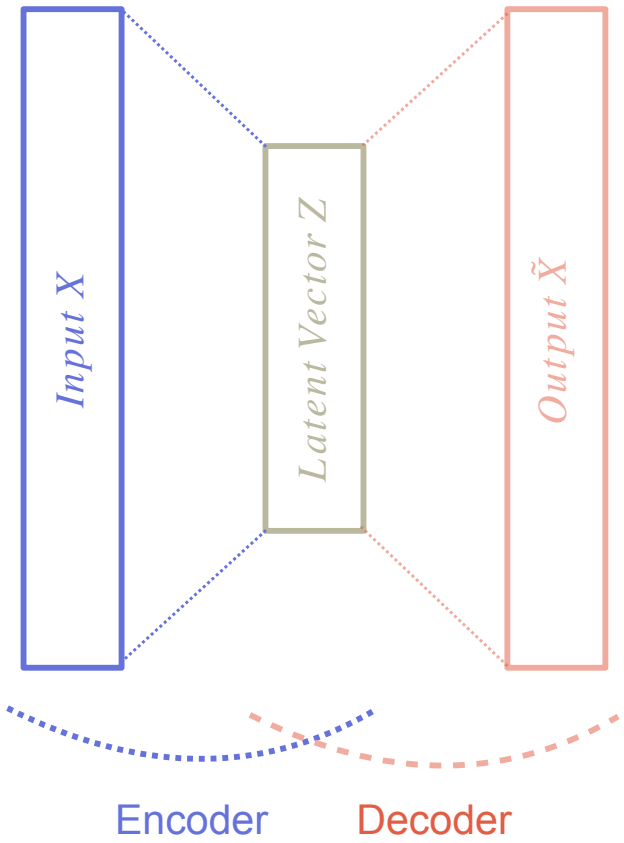


Abstract Representation



Auto-Encoders



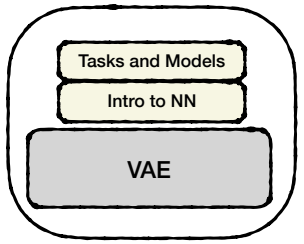


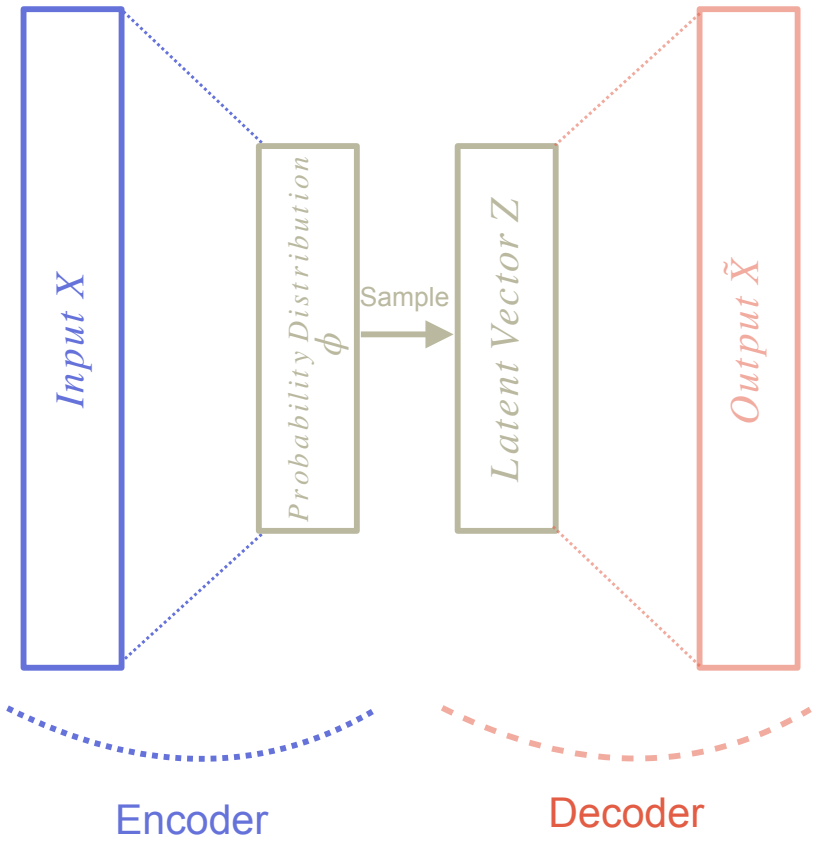
The aim here is to train this model such that $\tilde{X}$ gets as close to X as possible

In other words, we try to replicate the inputs on the outputs

After training, Latent Vector Z can be used as a lower dimensional representation of input X

Variational Auto-Encoders





Variational Auto Encoders are similar to Auto-Encoders except that instead of coming up with a “Compressed Representation” of data, they try to come up with a “Probabilistic Distribution” that describes the data.

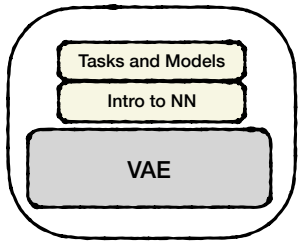
If we can come up with a Probabilistic Distribution of data, then we can generate outputs by “sampling” from the latent vector. This is very useful for generative tasks!

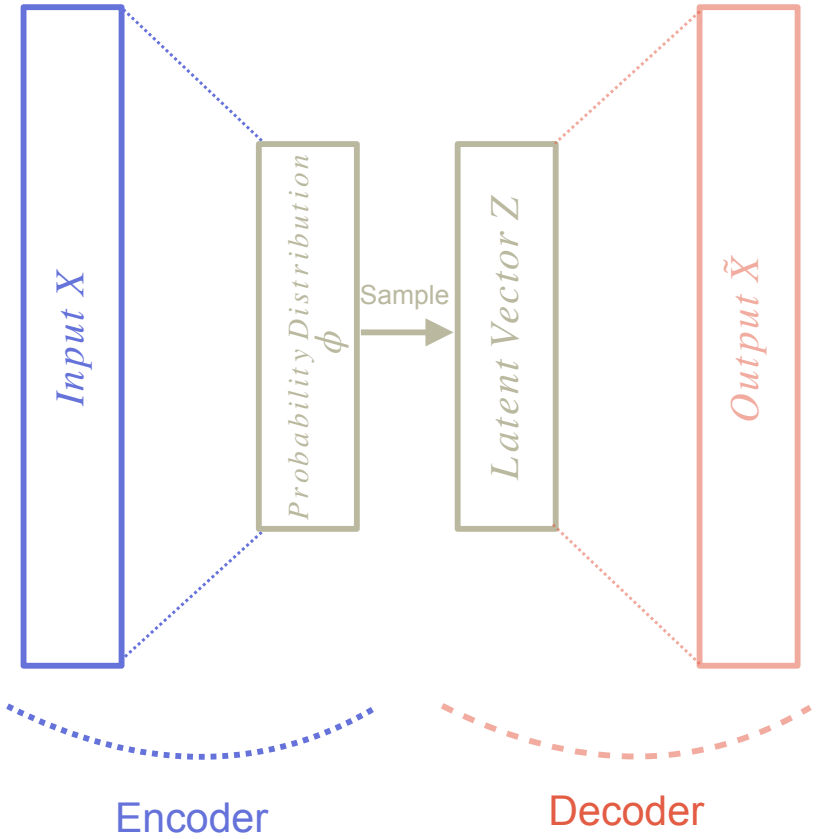
Again, here the aim here is to train this model such that $\tilde{X}$ gets as close to X as possible. However, there is an additional requirement here: Distribution

Latent Vector Z can be sampled from Probability Distribution ϕ. The sampled vector is then fed to the decoder to generate an output that “imitates” the input.

But, how can we come up with Probability Distribution ϕ?

Variational Auto-Encoders



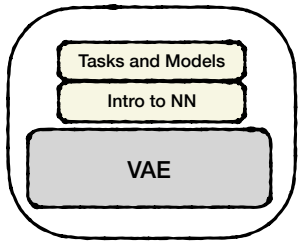


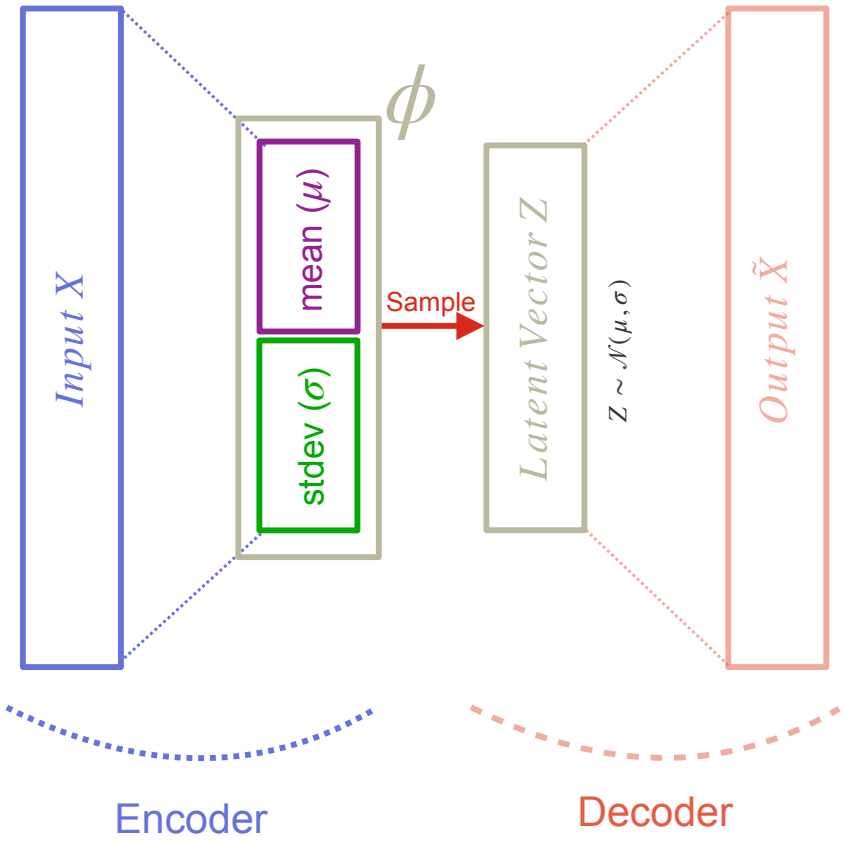
How can we come up with Probability Distribution ϕ?

We start from a prior assumption about the distribution. Let's assume it follows a Gaussian distribution.

Any Gaussian Distribution can be represented using two variables: mean (μ), standard deviation (σ) —> We can easily predict these in the encoder section

Variational Auto-Encoders



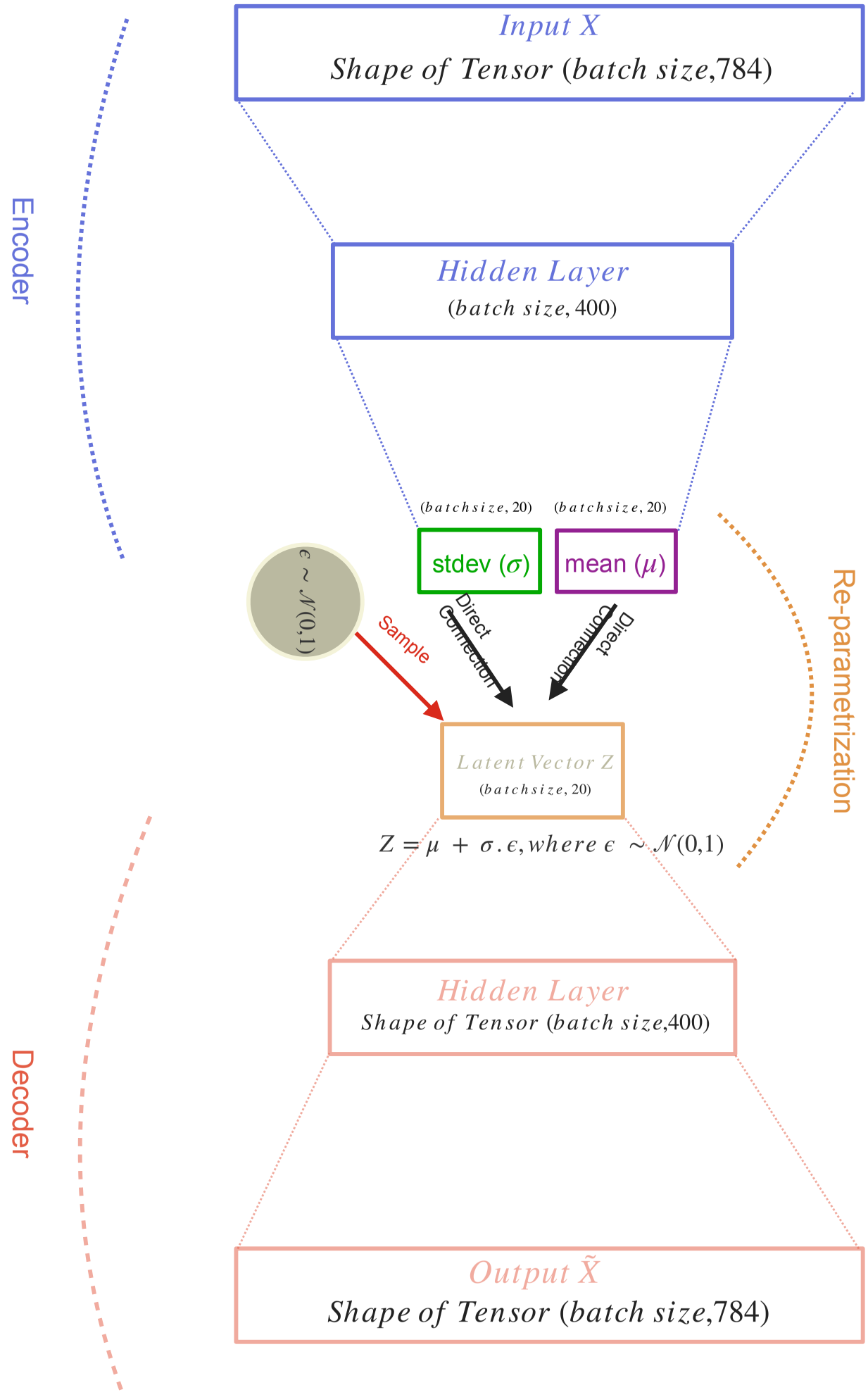


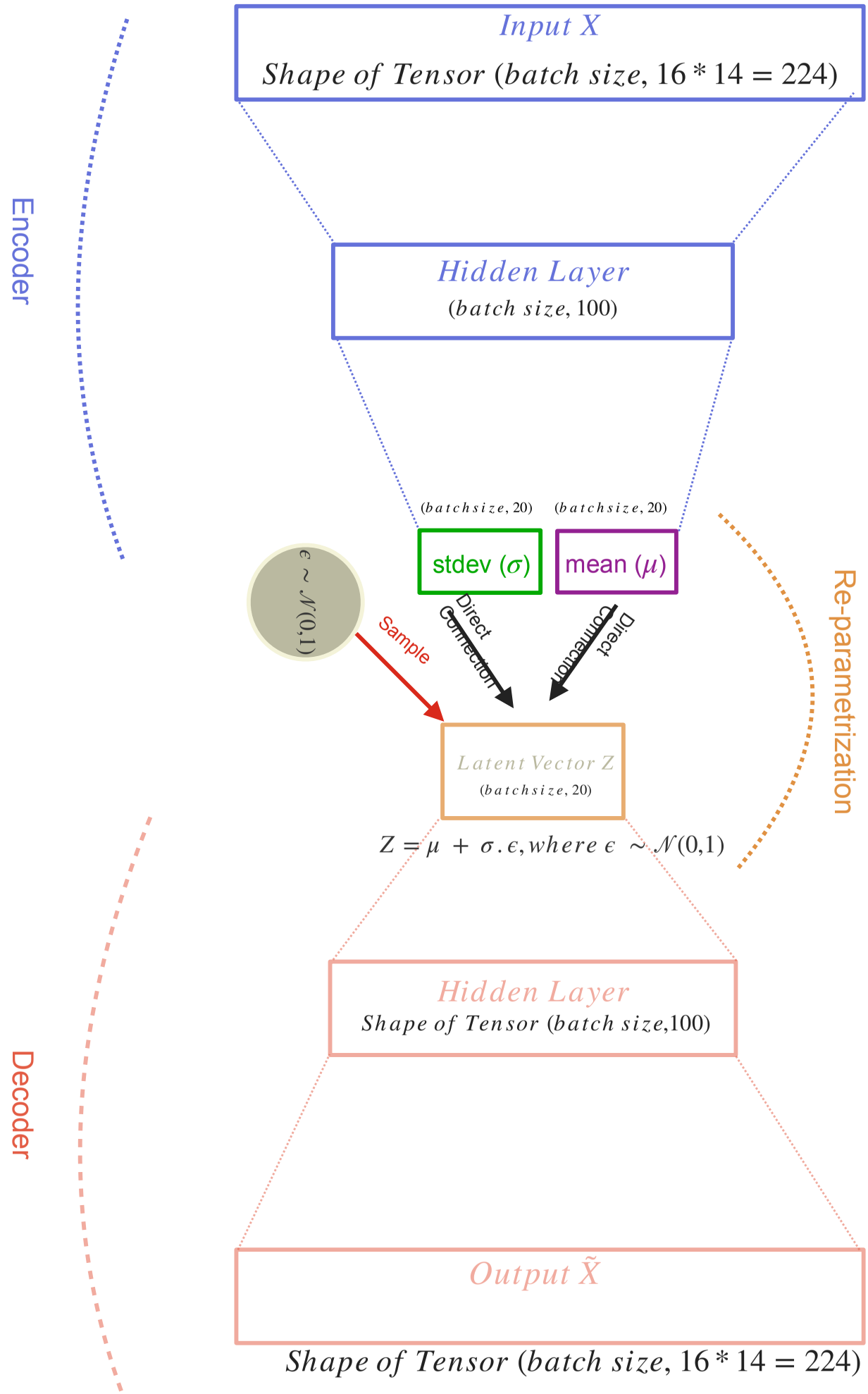
How can we come up with Probability Distribution ϕ?

We start from a prior assumption about the distribution. Let's assume it follows a Gaussian distribution.

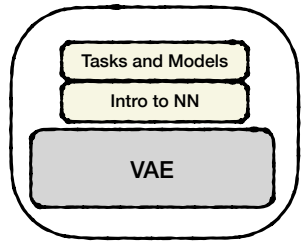
Any Gaussian Distribution can be represented using two variables: mean (μ), standard deviation (σ) —> We can easily predict these in the encoder section

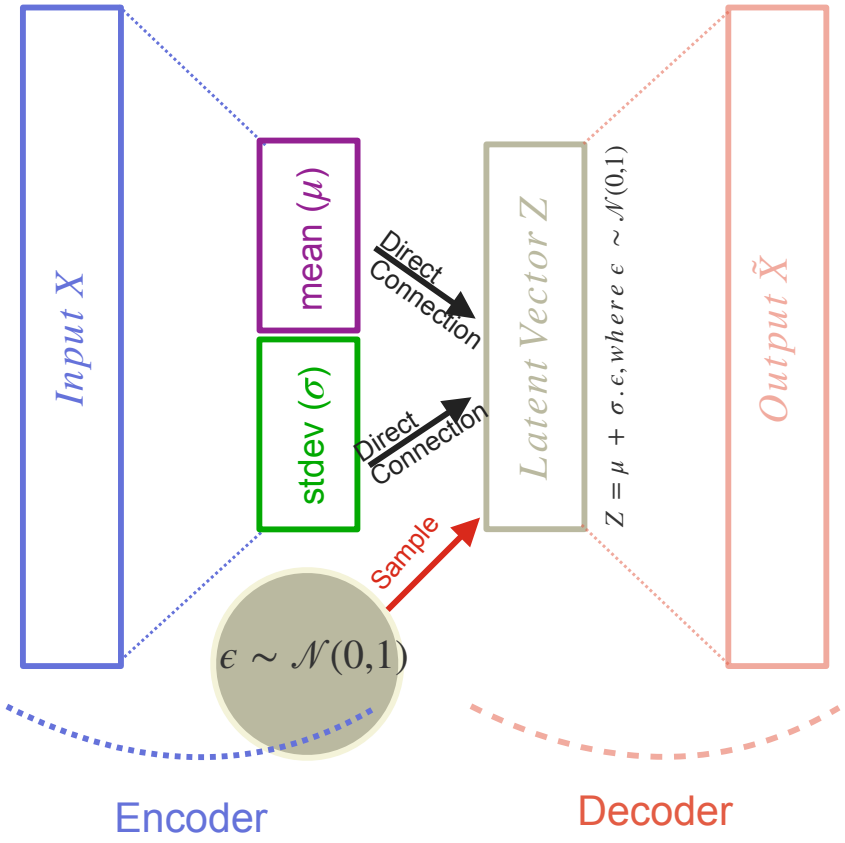
Note: A sampling operation is not differentiable! So this is problematic for training the model, as back-propagation requires every component to be differentiable. How do we fix this?!


Input X
Shape of Tensor (batch size,784)
Encoder
Hidden Layer
(batch size, 400)
(batchsize, 20)
(batchsize, 20)
stdev (σ)
mean (μ)
ε ~ N(0,1)
Sample
Direct Connection
Direct Connection
Re-parametrization
Latent Vector Z
(batchsize, 20)
Z = μ + σ.ε, where ε ~ N(0,1)
Hidden Layer
Shape of Tensor (batch size,400)
Decoder
Output X̃
Shape of Tensor (batch size,784)


Input X
Shape of Tensor (batch size, 16 * 14 = 224)
Encoder
Hidden Layer
(batch size, 100)
(batchsize, 20)
(batchsize, 20)
stdev (σ)
mean (μ)
ϵ ~ 𝒩(0,1)
Sample
Direct Connection
Direct Connection
Re-parametrization
Latent Vector Z
(batchsize, 20)
Z = μ + σ.ϵ, where ϵ ~ 𝒩(0,1)
Decoder
Hidden Layer
Shape of Tensor (batch size,100)
Output X̃
Shape of Tensor (batch size, 16 * 14 = 224)

Variational Auto-Encoders





How do we make the probabilistic components differentiable?!

By using some mathematical manipulations, we move "sampling" operations out of the back-propagation path —> This is called "Re-parametrization"

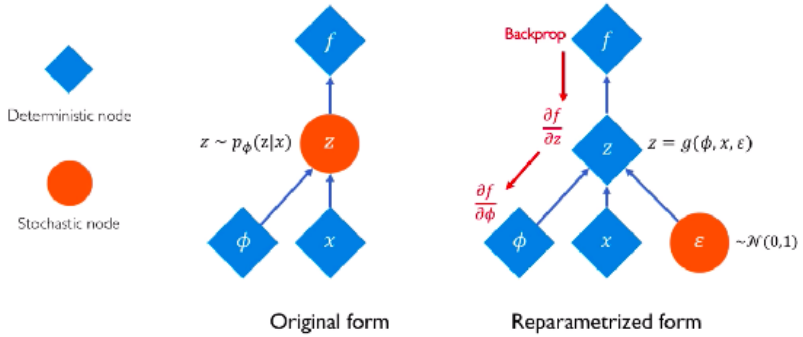
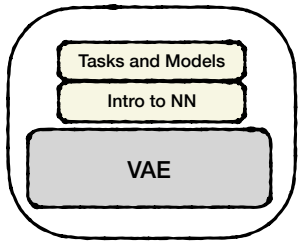
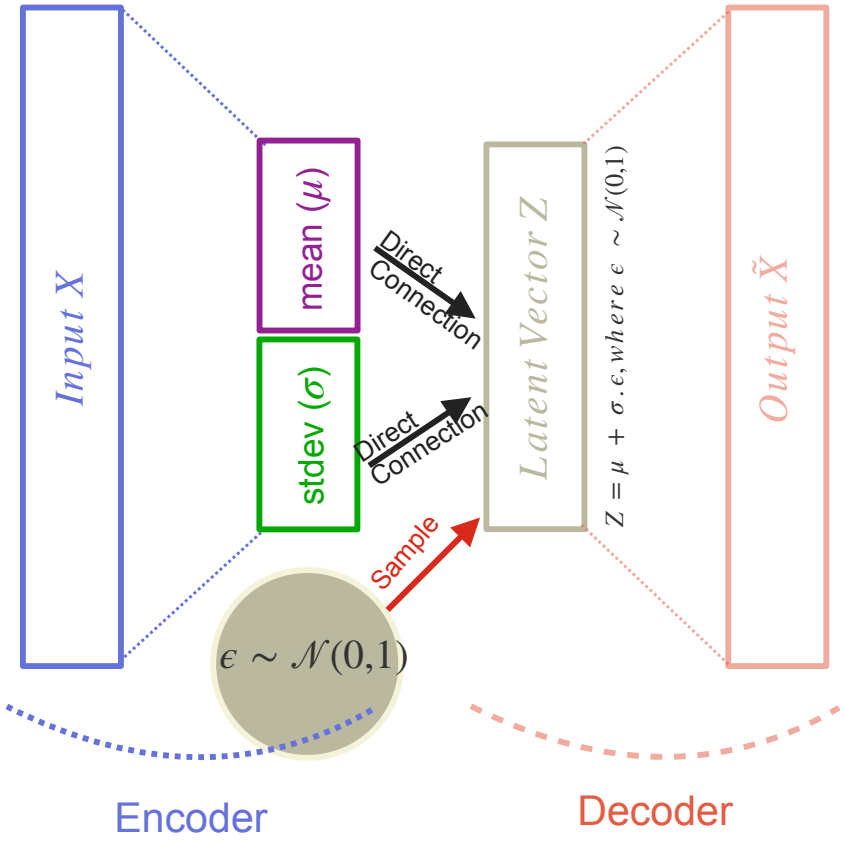


Image from https://towardsdatascience.com/reparameterization-trick-126062cfd3c3

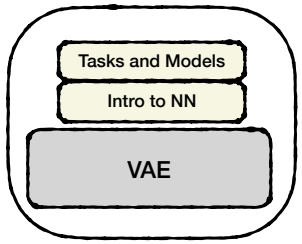
Variational Auto-Encoders





One more issue?! If we train the model as is, then it turns out that the model decides to set the stdev nodes to zero and only pass information through the mean (i.e. the model turns itself back into a deterministic auto-encoder)

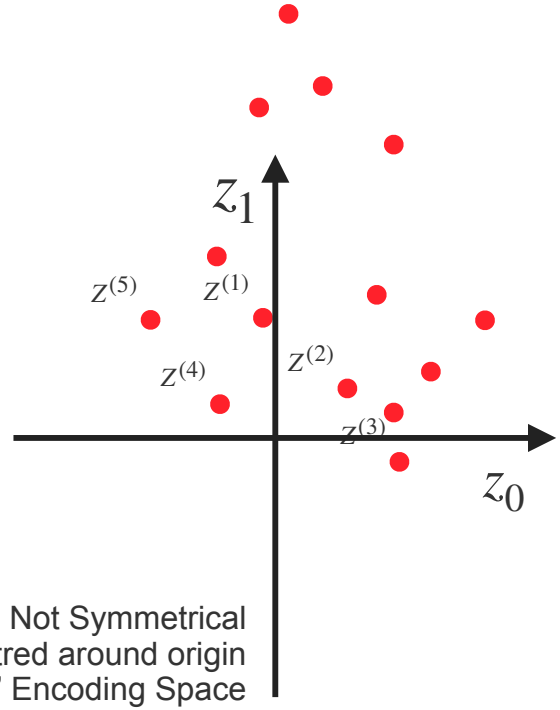
To solve this, we add a loss term to our loss function so as to ensure the predicted distributions to be as close to a standard normal distribution as possible



AE vs. VAE

$Input\ X = (x_0, x_1, x_2, \ldots, x_{n-1})$

$Latent\ Vector\ Z = (z_0, z_1)$



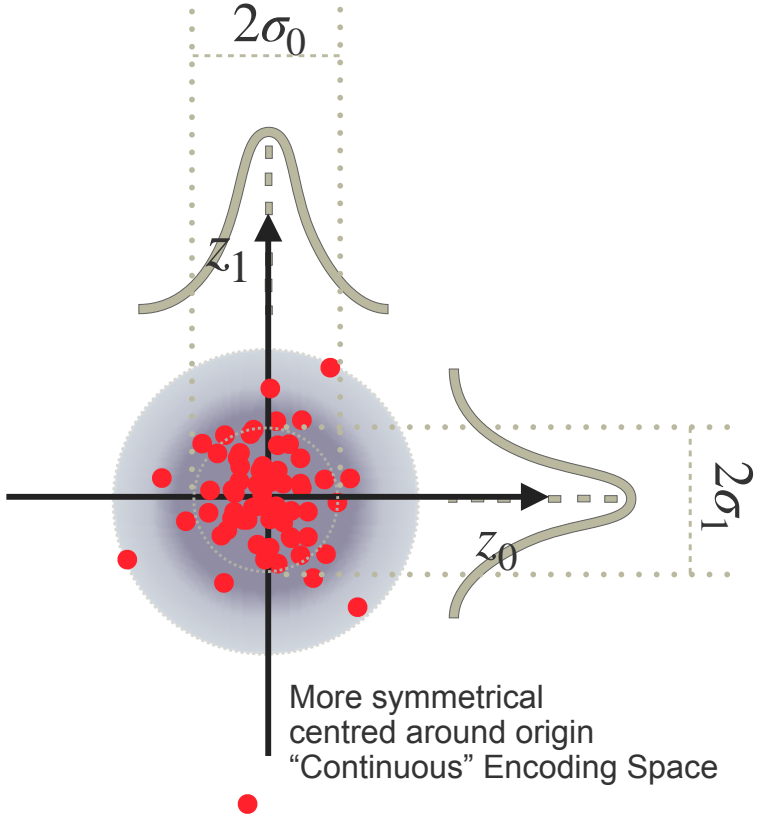
Not Symmetrical
Not centred around origin
"Discreet" Encoding Space

$Z^{(K)}$: $Latent\ Encoding\ of\ K^{th}\ Input$

$Input\ X = (x_0, x_1, x_2, \ldots, x_{n-1})$

$Latent\ Distribution\ \phi(\mu, \sigma) = (\mathcal{N}(\mu_0, \sigma_0), \mathcal{N}(\mu_1, \sigma_1))$

$Latent\ Vector\ Z\ is\ sampled\ from\ \phi$



More symmetrical
centred around origin
"Continuous" Encoding Space

$Z^{(K)}$: $Latent\ Encoding\ of\ K^{th}\ Input$

Read this if you're interested: https://towardsdatascience.com/generating-new-faces-with-variational-autoencoders-d13cfcb5f0a8