1 Multilayer Neural Networks

Multi-Layer Perceptron
Solves the limitations of a single-layer perceptron by introducing: Backpropagation, Hidden Layers

Universal Approximation Theorem
Equivalent to the **Universality Theorem**, which states that 'Neural networks with a single hidden layer can be used to approximate any continuous function to any desired precision.' The high-level idea is to prove that **the structure of MLP can identify any continuous non-linear patterns.**

Activation Functions
The primary motivation for using activation functions is to introduce **non-linearity** into the network.

Requirement for Activation Functions
1. **Must be nonlinear**, otherwise it is equivalent to a linear classifier.
2. **Continuous and differentiable** almost everywhere, to allow backpropagation.
3. **Must be monotonic**, otherwise it introduces additional local extrema in the error surface.

Issues of Sigmoid Function
1. **Sigmoids saturate and kill gradients.** A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. (same to Tanh)
2. Sigmoid outputs are not zero-centered.

Hyperbolic Tangent Function (Tanh)
The main advantage provided by the function is that it **produces zero-centred output** thereby aiding the back-propagation process.

ReLU Function
When the input is positive, the gradients don't vanish. This helps to mitigate the vanishing gradient problem
The **main advantage** of using the ReLU in computation is that, they **guarantee faster computation**
Another property of the ReLU is that it **introduces sparsity in the hidden units** as it squishes the values between zero to maximum.

Dead ReLUs (Issues of ReLU)
This occurs **when a large negative gradient flows through a ReLU neuron**, causing it **never activates on any data point again**.

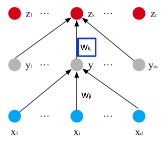
Leaky ReLU
To address the 'dying ReLU' problem
These variants **allow small negative values** when the input is less than zero, providing a way for 'dead' neurons to come back to life and continue learning.

Backpropagation
Hidden-to-output weight wkj

- Chain rule

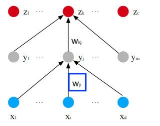
$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} y_j$$

$$\begin{cases} J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{t}-\boldsymbol{z}\|^2 \\ z_k = f(net_k) \\ net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} \end{cases}$$



Input-to hidden weight wji

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{w_{ji}}$$



$$\frac{\partial J}{\partial y_j} = \sum_{k=1}^{C}\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial y_j}$$

$$net_j = \sum_{i=1}^{d} x_j w_{ji} + w_{j0}$$

$$y_j = f(net_j)$$

2 Optimization for Training Deep Models

Gradient Descent

Batch Gradient Descent
calculated across the **entire** dataset. It can be very **slow on large datasets**
The model might **get stuck in** shallow local minima or **saddle points** in case of non-convex loss surfaces (**common for deep networks**).

Pure SGD (batch size is just 1):
Noise often results in better solutions, because it can escape from shallow local minima or saddle points due to the inherent noise in the gradient estimation.
Faster than batch learning

Mini-batch based SGD
Randomly permute mini-batches and take a **mini-batch** sequentially to approximate the gradient.
The **estimated gradient** at each **iteration** is **more reliable**.

Challenges for Gradient Descent
1. Choose a proper learning rate can be difficult. 2. The same learning rate applies to all parameter updates.3. Easily get trapped in numerous saddle points.

Momentum
tries to **accelerate** SGD in the relevant direction and dampens oscillations.

$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta). v_t = \gamma v_{t-1} + \eta\nabla_\theta J(\theta - \gamma v_{t-1})$ $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t+\epsilon}} \odot g_t.$

$\theta_t = \theta_{t-1} - v_t.$ $\theta = \theta - v_t$

Nesterov accelerated gradient (NAG) compare to Momentum
Momentum first **computes the current gradient** J(θ), and **then takes a big jump** in the direction of the updated accumulated gradient (θ = θ − vt).
NAG first makes a big jump in the direction of the previous accumulated gradient (θ − γvt−1), measures the gradient J(θ − γvt−1) and then makes a correction (θ = θ − vt) which results in the complete NAG update.

Adagrad
It **eliminates the need to manually tune the learning rate**. It adapts the learning rate to the parameters.
main weakness is its **accumulation** of the squared gradients in the denominator: causes the learning rate to shrink and eventually become infinitesimally small

Adadelta
restricts the window of accumulated past gradients to some fixed size w
sum of gradients is recursively defined as a **decaying average** of all past squared gradients.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2, \qquad \Delta\theta_t = -\frac{RMS[\delta\theta]_{t-1}}{RMS[g]_t}g_t.$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t+\epsilon}}g_t = \theta_t - \frac{\eta}{RMS[g]_t}g_t. \qquad \theta_{t+1} = \theta_t + \Delta\theta_t.$$

The first step in Adadelta is the **introduction of a moving average of the squared updates**.
Adadelta has one more step, which is the calculation of parameter updates based on the **ratio** of the accumulated average squared updates and the accumulated average squared gradients
This allows the **learning rate to be more adaptive** and better scaled for each parameter. **no need to set a default learning rate**
RMSprop in fact is identical to the first update vector of Adadelta

Adam
Adam also keeps an **exponentially decaying average** of past gradients mt, **similar to momentum**.
mt and vt are are **estimates** of the **first moment (the mean)** and the **second moment (the uncentered variance)** of the gradients respectively
As mt and vt are initilizaed as vectors of 0's, they are biased towards zero. They counteract these biases by **computing bias-corrected first and second moment estimates**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t. \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t}. \qquad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t.$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2. \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

Benefits of Adam
Adaptive Learning Rate: adapts the learning rate for each weight individually, through the use of moving averages of the gradient and squared gradient
Momentum Incorporation: Adam implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in RMSProp, essentially adding momentum
Bias Correction: Adam includes bias correction terms to adjust for the fact that both the first and second moment estimates are initialized at zero

Methods	Adaptive lnearning rate	Consistent units	Easily hand Saddle points
SGD	No	No	No
Momentum	No	No	No
Nesterov	No	No	No
Adagrad	Yes	No	Yes
Adadelta	Yes	Yes	Yes
Rmsprop	Yes	No	Yes
Adam	Yes	No	Yes

Parameter Initialization
Motivation: Parameter initialization can significantly affect the performance of the model. **Avoiding Symmetry**: allows each neuron to learn different features, **Mitigating Vanishing/Exploding Gradients**
All zero initialization will not work: Because every neuron computes the same output, so that they will also compute the same gradients and undergo the exact same parameter updates.
Xavier: The **mean** of the activations should be **zero**.
The **variance of the activations** should stay the **same across every layer.**
Assume the activation function is **tanh**
He: Assume the activation function is **ReLU**. Assure variance but mean.

$$\text{Xavier Normal}: \mathcal{N}(0, \frac{2}{n^{[l-1]}+n^{[l]}}). \qquad \text{Kaiming Normal}: \mathcal{N}(0, \frac{2}{n}).$$

$$\text{Xavier Uniform}: \mathcal{U}(-\frac{\sqrt{6}}{\sqrt{n^{l-1}+n^{[l]}}}, \frac{\sqrt{6}}{\sqrt{n^{l-1}+n^{[l]}}}). \quad \text{Kaiming Uniform}: \mathcal{U}(-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}).$$

3 Regularization for Deep Models
Motivation: Avoid Overfitting, Improve Generalization, Feature Selection

Weight Decay
By reducing the size of the weights, the model is **less likely to fit the noise** in the data and is **more likely to generalize** well to new data.
Weight decay is a form of **L2 regularization** that aims to prevent overfitting in a neural network.

$$\theta \leftarrow \theta - \eta\nabla\hat{L}_R(\theta) = \theta - \eta\nabla\hat{L}(\theta) - \eta\alpha\theta = (1-\eta\alpha)\theta - \eta\nabla\hat{L}(\theta).$$

It can be directly used in updating weights.

$$\min_\theta \hat{L}_R(\theta) = \hat{L}(\theta) + \frac{\alpha}{2}||\theta||_2^2.$$

Noise to the input
This can make the model **more robust** to small variations in the input data and improve generalization. Noise injection can be seen as **a form of data augmentation. Equivalence to the Weight Decay.**

Noise to the Weight
Weight noise helps the model explore a wider region of the parameter space
Early Stopping: When validation error not improved for some time, stop.

Dropout
• At training (each iteration): Each unit is retained with a probability p.
• At test: The network is used as a whole. The weights are scaled-down by a factor of p (e.g. 0.5).
Inverted Dropout: weight-scaled-down in training
Dropout has more advantages over weight decay
• **is scale-free**: dropout does not penalize the use of large weights when needed.
• **is invariant to parameter scaling**: dropout is unaffected if weights in a certain layer are scaled up by a constant c, and weights in another layer are scaled down.

Difference Between DropConnect and Dropout
Dropconnect randomly disconnects some links. The link of a neuron to a neuron in the next layer is disconnected, but this neuron still works on other neurons in the next layer.

Batch Normalization
Whitening (Classical Normalizations)
standardization vs normalization

$$\frac{x-\mu}{\sigma}. \qquad \frac{x-x_{min}}{x_{max}-x_{min}}.$$

Internal Covariate Shift
distribution of each layer's inputs changes during training, as the parameters of the previous layers change.
main issues: 1. hard to set an appropriate learning rate. 2. exacerbate the problem of vanishing or exploding gradients. 3. lead to a need for careful initialization and the use of smaller learning rates. 4. meaning the inputs to a layer can fall into the flat region of the activation function

Batch Normalization
Issues: 1. **Sensitive to batch size**. BN's error increases rapidly when the batch size becomes smaller, caused by inaccurate batch statistics estimation.
2. **Inconsistency at training and testing time**. At inference time, the mean and variance are pre-computed from the training set.
3. **Not suitable for Recurrent Connections**. Because activations of each time step have different statistic. Hence using BN in recurrent connections is complicated and computationally expensive.

$$\begin{aligned} \mu_\mathcal{B} &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i && \text{//mini-batch mean} \\ \sigma_\mathcal{B}^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_\mathcal{B})^2 && \text{//mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i-\mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2+\epsilon}} && \text{//normalize} \\ y_i &\leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) && \text{//scale and shift} \end{aligned}$$

Group Normalization
To solve the problem of sensitive to batch size, Group Normalization divides the channels into groups and computes within each group the mean and variance for normalization. **GN's computation is independent of batch sizes.**
Layer normalization is equal to GN with group = 1

Why LN in RNN not BN
1. **Sequence Dependency**. BN normalizes across the batch dimension, can disrupt the temporal dependencies.
2. **Consistent Normalization**. LN is consistent across all time steps, preserving the sequential nature of the data.
3. **Batch Size Flexibility**. RNNs often work with varying batch sizes, especially when dealing with time series data where sequences can be of different lengths.
4. **Training Stability**. LN tends to provide more stable training for RNNs.
5. **Handling Variable Length Sequences**. BN's dependency on batch statistics makes it less effective in scenarios with variable-length sequences.

4 Convolutional Neural Networks
Problem of MLP: the number of **weights grows largely** with the size of the input image, and **pixels in distance** are **less correlated.** Traditional NNs do not consider the **spatial structure of the image** and sensitive to the **position of an object in the image.**
To solve these problems, CNN propose:
1. **Shared Weights**: Hidden nodes at different locations share the same weights. increase the translation invariance.
2. **Multiple Filters**: Multiple filters provide the probability of detecting the spatial distributions of multiple visual patterns.

Convolutional Layer
Benefits:1. Dimensionality Reduction/Augmentation. 2. Reduce computational load by reducing parameter map. 3. Add additional **non-linearity** to the network. 4. can capture l**ocal patterns and features**. 5. Better Performance on **Spatial** Data

$$H_{out} = \left\lfloor\frac{H_{in}-K+2P}{S}\right\rfloor + 1 \qquad W_{out} = \left\lfloor\frac{W_{in}-K+2P}{S}\right\rfloor + 1$$

Dilated Convolution
The "**holes**" basically define a spacing between the values of the kernel.

Pooling
Benefits: 1. Pooling helps the representation become **slightly invariant to small translations of the input**. 2. Pooling is used for **down sampling**, it can be used to handle inputs of varying sizes. 3. Responsible for **reducing the spatial size** of the Convolved Feature. it is useful for **extracting dominant features which are rotational and positional invariant**,

Other Pooling
- Other pooling
  - L_p **pooling** (preserves the class-specific spatial/geometric information in the pooled features)

    $$y_i = \left(\sum_j w_j {x_{i,j}}^p\right)^{1/p}$$

  - **Mixed pooling** (addresses the over-fitting problem)

    $$y_i = a\max(x_{i,1},\dots,x_{i,n}) + (1-a)\text{mean}(x_{i,1},\dots,x_{i,n})$$

  - **Stochastic pooling** (hyper-parameter free, regularizes large CNNs)

    $$y_i = x_l, \text{ where } l \sim P(p_1,\cdots,p_n) \text{ and } p_j = \frac{x_{i,j}}{\sum_j x_{i,j}}$$

  - **Spectral pooling** (preserves considerably more information per parameter than other pooling strategies)

    $$\mathbf{y} = \mathcal{F}(\mathbf{x}) \in \mathbb{C}^{M\times N}, \hat{\mathbf{y}} = \mathcal{F}^{-1}(\mathbf{y} \in \mathbb{C}^{H\times W})$$

  - …

Receptive Field
The receptive field in CNN is the **region of the input space** that affects a particular unit of the network.

Ways to Increase the RF
• Add **more convolutional layers** (make the network deeper).
• **Add pooling layers** or **higher stride convolutions** (sub-sampling)
• Use **dilated convolutions**.
• **Depth-wise convolutions** (see MobileNet [1]).

Why Increase RF?
1. **Capture larger patterns**: By increasing the size of the receptive field, a neuron can capture larger and more complex patterns in the input data.
2. **Reduce computational cost**: By using a larger receptive field, a network can reduce the number of layers or the number of neurons it needs to capture the same amount of information.
3. **Incorporate context**: A larger receptive field allows a neuron to incorporate more context from the input data.
4. **Robustness to object scale and position**

5 Neural Network Architectures

Winning CNN Architectures

Model	AlexNet	ZF Net	GoogLeNet	Resnet
Year	2012	2013	2014	2015
#Layer	8	8	22	152
Top 5 Acc	15.4%	11.2%	6.7%	3.57%
Data augmentation	√	√	√	√
Dropout	√	√		
Batch normalization				√

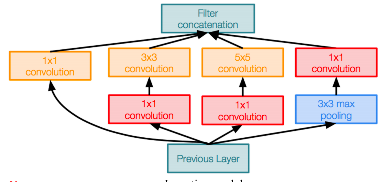
LeNet
Problems it solves: LeNet is designed to recognize handwritten and machine-printed characters.
LeNet was one of the **first successful applications of CNNs**.

AlexNet
Benefits: It **introduced several key concepts** such as ReLU activations, dropout, and multiple GPUs for training.
Issues: It has **many parameters and can be computationally intensive**.
Key Features
Using **ReLU** as the activation function, use **Data Augmentation**, use **Dropout**, use **Local Response Normalization**, and **Overlapping Pooling**
Overlapping Pooling: When pooling stride < pooling kernel size, the overlapping pooling occurs. The **advantage of overlapping pooling** is that it can help to reduce the risk of overfitting by providing a form of implicit data augmentation and making the representation slightly more translation invariant.

ZFNet
ZFNet was developed to improve the **visualization of higher-level feature maps** and **understand what features the network learned**. It is an improved version of AlexNet

VGGNet
Key Features
Use **3 × 3 convolutional layers** with **stride=1** and **padding=1**, and **2×2 max-pooling** with **stride=2**
Prefer a stack of small filters: **two 3x3 layer = 5x5 layer, three 3x3 layer = 7x7 layer**. Use more small filters can make the network deeper which **introduces more non-linearity and results in less parameters**. Also, small filters can result in larger receptive fields, which can capture larger patterns.

GoogLeNet
The **inception module** was introduced to **reduce the computational cost** while **increasing the depth and width** of the network.
Key Features
• **Inception Module**:
Before performing 3x3 and 5x5 convolutions, a **1x1 convolution (bottleneck convolution)** is applied to **reduce the dimensionality of the input**. This can **dramatically reduce the computational cost** of the subsequent larger convolutions, making the network faster and less memory-intensive without significantly sacrificing performance.
The **benefits** of the Inception module are:
– **Efficiency**: The use of 1x1 convolutions before larger convolutions reduces computational cost.
– Multi-level feature extraction: By performing **convolutions of different sizes** in parallel, the module can **capture information at different scales** in input.
• **1x1 convolution**:
This effect of cross channel down-sampling is called 'Dimensionality reduction'.



Inception module

ResNet
The motivation is that solve **vanishing gradient problem**. As a result, the **weights in the earlier layers of the network are updated very slowly** and the network can become stuck during training, leading to underperformance.
Issues
Residual connections **help** with the vanishing gradient problem, **but** they **do not eliminate it entirely**
Key Features
• **Residual Connections**: Rather than feeding the output of a layer directly into the next one, the original input is added to the output of the layer, creating a "shortcut" or "skip connection". We **make the layer learn F(x) = H(x) - x (the residual)**, and then we add x back to get the final output H(x) = F(x) + x.
The benefits of residual connections include:
– **Ease of Training**: Residual connections alleviate the vanishing gradient problem
– **Improved Performance**: By allowing the model to learn more complex functions, residual connections can improve the performance of the network.

Inception v2
Key Features: The usage of **batch normalization** (compare to v1) and **small kernels (3*3)**(inspired by VGG), and remove LRN and dropout compared to naive Inception module. 7x7 filter is factorized into **7x1 and 1x7 filters (same Receptive Field but smaller parameter)**, 3x3 filter is factorized into 3x1 and 1x3 filters.

Inception v3
Key Features: Inception uses **label smoothing** technique.

Xception
Xception was developed to improve the Inception architecture by using depthwise separable convolutions (In the **first step**, **each channel is processed using a separate convolution kernel**, which greatly reduces the amount of calculation and the number of parameters. In the **second step**, the results of the deep convolution are **combined and reshaped using a 1x1 convolution kernel**, which is equivalent to integrating the features of each channel).
It **improves the efficiency** of the network in terms of computation and number of parameters.

Wide ResNet
Key Features: **Increasing width** instead of depth can be more computationally efficient.

ResNeXt
Key Features: The key innovation in ResNeXt is the introduction of the **"cardinality" dimension.** In a ResNeXt block, the **input is split into several different paths**, each of which **undergoes a series of transformations** (typically a bottleneck convolution followed by a 3x3 convolution) **independently**. ResNeXt uses the **same transformation in each path**.
Group convolution: Split the dimension into many groups, and then convolve each group. The **number of groups is cardinality**.
The benefits of ResNeXt:
The **representational power** of the network can **be increased without significantly increasing the complexity** of the network.

DenseNet
The key innovation in DenseNet is the idea of **dense connectivity**. In a DenseNet, **each layer is connected to every other layer in a feed-forward fashion**. Specifically, the **output of each layer is concatenated with the inputs of all subsequent layers**.
The benefits of DenseNet:
• **Parameter Efficiency**: Each layer in a DenseNet has direct access to the gradients from the loss function and the original input signal, leading to an improved flow of information and gradients throughout the network, which can allow for better performance with fewer parameters.
• **Feature Reuse**: The dense connectivity leads to a form of implicit deep supervision, as the features learned by the early layers are directly used by the later layers.
• **Improved Performance**

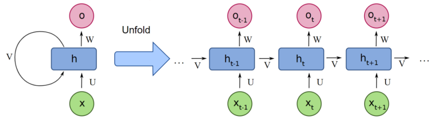
SqueezeNet
Key Features
SqueezeNet introduces a new building block called a "**Fire module**" to construct the network. A Fire module consists of a "squeeze" convolutional layer (with 1x1 filters) which feeds into an "expand" layer that has a mix of 1x1 and 3x3 convolutional filters.
• **Use of 1x1 Filters**: reduce the dimension in "squeeze" stage
• **Decrease in the Number of Input Channels (use 1*1 before) to 3x3 Filters**: The number of parameters can be further reduced.
• **Downsampling Late in the Network**: Performing downsampling late in the network, ensures that convolutional layers have large activation maps.
The main advantage of SqueezeNet is its **small size and high efficiency**. SqueezeNet was designed to achieve the **same level of accuracy as AlexNet, but with 50 times fewer parameters**.

MobileNet
The key idea behind MobileNet is the use of **depthwise separable convolutions** instead of standard convolutions. A depthwise separable convolution consists of two parts: **a depthwise convolution followed by a pointwise convolution**.
• **Depthwise Convolution**: Each feature layer has an independent kernel for convolution, and the width of the output is the width of the input.
• **Pointwise Convolution**: Use 1*1 convolution to increase dimension
Two hyperparameters that can be tuned to trade off between latency and accuracy: **Width Multiplier, Resolution Multiplier**
The benefits of MobileNet include:
• **Efficiency**: MobileNet is more computationally efficient and smaller in size, making it suitable for mobile and edge devices.
• **Adaptability**: MobileNet can be easily adapted to different use cases by adjusting the width and resolution multipliers.

ShuffleNet
Key Features
• **Pointwise Group Convolutions**: In a group convolution, the input channels are split into several groups, and each group of channels is convolved separately (Replace the traditional 1*1 convolution with group convolution).
• **Channel Shuffle**: The channel shuffle operation rearranges the output channels from the group convolution (The features generated by group convolution are only related to the feature layer in the channel of the group. Channel shuffle is used to shuffle the channels between different groups to achieve the purpose of interaction between groups).
The benefits of ShuffleNet include: Efficiency, Performance.

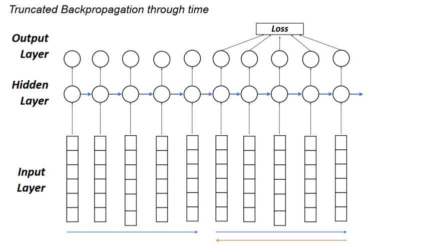
6 Recurrent Neural Network
Motivation - The Usage of Memory
• **Sequential Dependency**: In tasks such as natural language processing or time series prediction, the current output often depends not just on the current input, but also on the preceding inputs.
• **Variable Length**: Sequences can vary in length, and important information can appear anywhere in the sequence.
• **Temporal Dynamics**: In tasks such as speech recognition or video processing, temporal dynamics are important.
An RNN uses its memory to maintain this context/ handle sequences of any Length/ allows it to capture these dynamics.

Neural Network + Memory = RNN



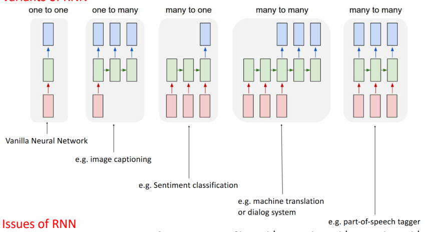
$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \qquad y_t = W_{hy}h_t + b_y$$

What about very long sequence?
Use truncated backpropagation through time: run forward and backward **through chunks of the sequence instead of the whole sequence**.



Why do we use Tanh in RNN?
The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1. **If we do not use tanh, the values will become either exploding or very small (even disappearing) during the training.**

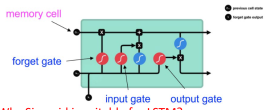
Variants of RNN



Issues of RNN
• **Vanishing Gradient Issue**. if 0 < Whh < 1, [(tanh)′ Whh(tanh)′ Whh(tanh)′ Whh] will approach to 0.
• **Exploding Gradients**. if Whh > 1, [(tanh)′ Whh(tanh)′ Whh(tanh)′ Whh] will approach to infinity. In an RNN, error gradients can **accumulate** during an update and result in very large gradients.
• **The problem of learning long-range dependencies**. For a long sequence, RNN may not be able to capture the information of the previous information if the distance is too far.

Long Short-Term Memory (LSTM)
Motivation
To solve the problem of learning long-term dependencies in sequence data LSTMs introduce a way to maintain a **"long-term memory"** and thereby mitigate the vanishing gradient problem.
Methodology
Forget gate layer: Decides what information from the long-term state Ci-1 should be thrown away or kept
Input gate layer: Decides what new information should be stored in cell state
Update the cell state: Update the old cell state Ct−1 to produce the new state Ct by combining the previous 2 results (the values of the forget and input layers)

Output gate layer: Decides what exactly to output - the output will be based on the cell state, but it will be a filtered version



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$\tilde{c}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot \tanh(C_t)$$

Why Sigmoid is suitable for LSTM?
Each of these gates uses a sigmoid activation function to output values between 0 and 1, indicating how much information to forget or keep. **The closer to 0 means to forget, and the closer to 1 means to keep.**

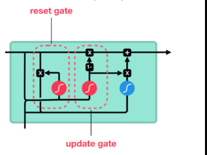
Issues of LSTM
LSTM is **computationally expensive** as it has **4 times more parameters than an RNN** model because it introduces three gates and one cell state.

Gated Recurrent Unit (GRU)
Motivation
The motivation behind GRUs is to provide a model that can learn long-term dependencies like an LSTM **but with less computational complexity**

Benefits of GRU (Comparison to LSTM)
• **Fewer Gates**: GRUs simplify the LSTM architecture by using two gates (update and reset)
• **Combining Cell State and Hidden State**: In LSTMs, the cell state and hidden state are separate, whereas in a GRU, there is no separate cell state.
• **Efficiency**: GRUs have shown to perform on par with LSTMs on certain tasks despite their simplicity

Methodology
1. **Update Gate (z)**: This gate determines how much of the previous hidden state to keep and how much new information to add.
2. **Reset Gate (r)**: The reset gate determines how much of the previous hidden state to forget.
3. **New Memory Content**: The network calculates the new memory content, which is the new information that is to be added to the hidden state.
4. **Final Hidden State (ht)**: Finally, the network calculates the new hidden state (ht), which is a combination of the previous hidden state (ht−1) and the new memory content (h′ t).

$$z_t = \sigma(w_z[x_t, h_{t-1}] + b_z)$$
$$r_t = \sigma(w_r[x_t, h_{t-1}] + b_r)$$
$$\tilde{h}_t = \tanh(w_h[x_t, r_t \odot h_{t-1}] + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$



Bidirectional RNN
Bidirectional Recurrent Neural Networks (BRNNs) are an extension of standard RNNs that are able to **capture information from both past and future states**.

Benefits of BRNN compared to RNN
• **Better Performance on Certain Tasks**: BRNNs generally outperform standard RNNs on tasks where context from both the past and future is useful.
• **More Comprehensive Information Utilization**: BRNNs take both past and future input features into account at each time step.

Issues of BRNN compared to RNN
• **Increased Computational Complexity**: As BRNNs are essentially two RNNs combined, they require more computational resources and take longer to train.
• **Not Suitable for Real-Time Applications**: Unlike standard RNNs, BRNNs require the **entire sequence** for making predictions.
• **Overfitting**: Because they are more complex models, BRNNs are more prone to overfitting, especially on smaller datasets.
• **Memory Requirements**: Due to the need to store the outputs of the forward and backward RNNs, BRNNs can require more memory capacity.

Stacking RNN
Benefits of Stacking RNN compared to RNN
• **Ability to Capture Complex Patterns**: Stacked RNNs can potentially capture more complex patterns in the data.
• **Hierarchical Feature Learning**: In theory, different layers of the RNN might learn to represent different levels of abstraction in the data.

Issues of Stacking RNN compared to RNN
• **Difficulty of Training**: Deep RNNs can be harder to train effectively.
• **Risk of Overfitting**: With the increased complexity and capacity of the model, stacked RNNs might overfit to the training data
• **Increased Memory Usage**: Stacked RNNs require more memory to store the intermediate outputs for each layer.

Applications of Seq2Seq Learning
• Speech Recognition.• Movie Frame Labelling.• Math Expression. • Machine Translation.• Sentence Completion.• Conversation Modelling.• Dialog System.

Other Applications (Not Belong To Seq2Seq Learning)
Visual Question Answering
Given an image and a natural language question about the image, the task of the VQA system is to **provide an accurate natural language answer**.
Difference to Seq2Seq Learning: Image processing and multimodal feature combination, which aren't typically part of Seq2Seq tasks.

7 Transformer Neural Networks
Intuition Behind Attention: **Attention is a weighted average**, where the weights are determined by comparing the values with a query.

Benefits of Using Attention
• **Improve performance.**• **Solving the bottleneck problem.**• Reducing **vanishing gradient** problem.• **Providing some interpretability**: Inspect attention distribution, and show what the decoder was focusing on.

Self-Attention
The self-attention mechanism in the Transformer model uses three vectors derived from the input: Query (Q), Key (K), and Value (V). **These vectors are created by applying linear transformations** (i.e., multiplication by a learned weight matrix) to the input embeddings.
• **Query**: This vector represents the current word or token we're focusing on. It's used to score the relationship or relevance of the current word to all other words in the sequence.
• **Key**: This vector is used to represent all words or tokens in the sequence. Each word has a Key vector, and these vectors are compared with the Query vector to determine the word's relevance.
• **Value**: This vector represents the actual content of the words in the sequence. The Value vector of each word gets weighted by the attention score to form the final output representation of the sequence.

Why divide by square root dk
Forward propagation: If not scaled, after softmax, some attention scores may be large (1.0) and some may be small (0.0), which does not conform to the assumption of temporal correlation.
The **premise of the model** is that the data of each layer conforms to the normal distribution with mean 0 and variance 1. After multiplying qk, it becomes a distribution with **mean 0 and variance dk**. Dividing by square root dk can pull the variance back (to prevent internal covariate shift)

Scaled Dot-Product Attention: $\alpha_{1,i} = (q^1 \cdot k^i)/\sqrt{d}$,

where **d is the dimension of q and k**, followed by a softmax operation to get the weights to sum to one:
The final step in the self-attention process is to compute a weighted sum of the Value vectors, using the attention scores as weights:

$$\hat{\alpha}_{1,i} = \exp(\alpha_{1,i})/\sum_j \exp(\alpha_{1,j}) \quad b^1 = \sum_j \hat{\alpha}_{1,i} v^i,\ b^2 = \sum_j \hat{\alpha}_{2,i} v^i \ldots$$

Multi-head Self-Attention
The motivation for multi-head self-attention in the Transformer model is to allow the model to focus on **different types of information and capture various aspects** of the input data.
In the context of natural language processing, different words in a sentence can have **different types of relationships** with each other
Another important aspect is that multi-head attention allows the model to focus on **information at different positions**.

Benefits of Multi-head Self-Attention
• **Diverse Attention**: The model can pay attention to different parts of the input simultaneously, allowing it to **capture various aspects of the data**.
• **Different Scales of Attention**: Multi-head attention allows the model to focus on information at different positions.
• **Increased Capacity**: Adding more heads increases the capacity of the model without increasing the computational complexity
• **Improved Performance**: In practice, models with multi-head self-attention tend to perform better than those with single-head self-attention.

Masked Multi-Head Attention
Motivation
It's used to **prevent the model from seeing future tokens** in a sequence when generating a prediction. When applying the self-attention mechanism, mask ensures that **only the current and preceding positions can be attended to**.

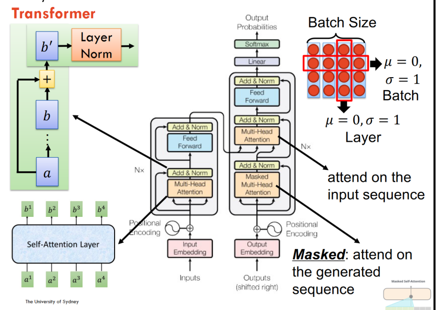
Positional Encoding
Motivation: Without the recurrence, there is **no indication of the order of inputs**. The **positional encoding provides information about the order**.
Methodology: The positional encodings have the **same dimension as the embeddings** so that they can **be summed together**.
The original Transformer paper uses **a specific type of positional encoding** that uses sine and cosine functions of different frequencies.

$$e^i(i, 2j) = \sin(i/10000^{2j/d})$$
$$e^i(i, 2j+1) = \cos(i/10000^{2j/d})$$

Transformer
Why the Inputs of the Decoder is the Outputs of encoder?
During training, the model is provided with the correct target output sequence as input to the decoder, which guides the model to **learn how to generate the output sequence correctly**, effectively learning the dependencies between the tokens and improving its ability to **generate coherent and meaningful output sequences**.
However, to allow for **efficient parallelization during training**, the target output sequence is shifted by one position to the right.

Benefit
Parallelization: Transformers can process tokens in parallel, unlike recurrent neural networks (RNNs) where computations are sequential.
Long-range dependencies: Transformers are capable of capturing long-range dependencies in sequences more effectively than traditional sequential models like RNNs or LSTMs.
Positional encoding: By incorporating positional encoding, Transformers can encode the position information of tokens in the input sequence
Versatility: Transformers have achieved state-of-the-art performance across various natural language processing tasks

Limits
High computational cost: The self-attention mechanism used in Transformers requires quadratic time and space complexity with respect to sequence length
Data inefficiency: Transformers often require large amounts of data for training due to their high parameter count and complexity.
Limited interpretability: The inner workings of the model can be less interpretable compared to simpler models like decision trees or linear models.
Lack of sequential memory: Unlike RNNs, which have an inherent sequential memory due to their recurrent nature, Transformers lack explicit sequential memory.



Embeddings from Language Model (ELMO)
ELMO uses the bidirectional LSTM.
The main innovation of ELMo is the use of **contextual word embeddings**.
ELMo achieves this by using a type of recurrent neural network called a long short-term memory (LSTM) network to process **a text sequence from left to right and from right to left**.
This bidirectional processing allows ELMo to **capture both the forward and backward context** for each word.

Bidirectional Encoder Representations from Transformers (BERT)
BERT vs. ELMO
BERT also uses contextual embeddings, but it goes a step further than ELMo by directly modeling the **interactions between all words in a sentence.**
It achieves this **using a Transformer architecture**, which allows it to attend to all parts of the input sequence simultaneously.

Training Methodology
• **Masked Language Modeling (MLM)**: During the training of BERT, a certain percentage of the input data gets replaced with a [MASK] token. This task encourages the model to learn a representation that fuses left and right context.
• **Next Sentence Prediction (NSP)**: This task is important for tasks that require understanding of the relationship between sentences, like question answering and natural language inference.
During pre-training, BERT learns a lot of information about the language which can then **be fine-tuned on a specific task with a much smaller amount of training data**.
How To Use BERT: 1. Input: Single sentence, Output: Class. 2. Input: Single sentence, Output: Class of each word. 3. Input: Two sentences, output: Class. 4. Input: Document, Query, Output: Answer.

Generative Pre-Training (GPT)
Motivation
Supervised models have two major limitations:
• They need large amount of annotated data for learning a particular task which is often not easily available.
• They fail to generalize for tasks other than what they have been trained for.
This paper proposed learning a generative language model **using unlabeled data and then fine-tuning the model.**

GPT-1
Task Specific Input Transformations: To make minimal changes to the architecture of the model during fine tuning, inputs to the specific downstream tasks were transformed into ordered sequences.

GPT-2
The developments in GPT-2 model were mostly in terms of **using a larger dataset and adding more parameters** to learn even stronger language model.

GPT-3
GPT-3 and GPT-2 are both large transformer-based language models developed by OpenAI, and they share the same core architecture. However, there are several **key differences between the two**:
• **Model Size**: GPT-3 is significantly larger than GPT-2. GPT-3 has 175 billion parameters
• **Training Data**: While both models were trained on a diverse range of internet text, GPT-3 was trained on more data.
• **Performance**: Due to its larger size and more extensive training data, GPT-3 generally outperforms GPT-2 on a wide range of natural language processing tasks without task-specific training data.
• **Few-Shot Learning**: This means that GPT-3 can understand the task it needs to perform by seeing a few examples.

GPT-3.5
GPT-3.5 aligns LLMs with user intent by fine-tuning **with human feedback**.

Reinforcement Learning with Human Feedback (RLHF)
• **Pretraining a Language Model (LM)**: RLHF generally use pre-trained LMs as the starting point (e.g., GPT-3 is used for GPT-3.5).
• **Gathering data and training a Reward Model (RM)**: RM in RLHF: takes in a sequence of text and returns a scalar reward that represent the human preference. Human annotators rank the outputs from the LM.
• **Fine-tuning the LM by reinforcement learning**
1. Sample a prompt
2. Pass it through both initial LM and the RL Policy (Copy of the initial LM)
3. Pass the output from policy to RM to calculate reward, and use output from both initial LM and policy to calculate shift penalty.
4. Use reward and penalty together to update the policy by PPO (Proximal Policy optimisation)

ChatGPT
Same method as InstructGPT, but **slightly different data collection setup**:
• **For initial supervised fine-tuning data**:
– The trainers acted as both users and AI assistants to create conversation dataset with help of model-written suggestions.
– Mix the new dialogue dataset with InstructGPT dataset, which is also converted into dialogue format.
• **For reward model data (comparison data)**:
– Took conversations that AI trainers had with the chatbot.
– Randomly selected a model-written message, sampled several alternative completions, and had AI trainers rank them.

The best way to provide inputs into LLM's
1. Clear and Concise Prompts: Provide clear and specific instructions. Explicitly state what you want the model to do.
2. Provide Context: Include relevant context within the input. If you require a specific style or tone, mention it in the prompt.
3. Use Examples: Show examples of the desired output.
4. Structure and Format: Use bullet points, numbered lists, or other structures to organize the input clearly.
5. Iterative Refinement: If the initial response isn't satisfactory, refine your prompt and try again.
6. Multi-turn Interactions: For ongoing conversations, maintain coherence by referencing earlier parts of the dialogue.
7. Control Output Length and Detail: Indicate the desired length of response.

Vision Transformer
Motivation-difference with CNN
Traditionally, Convolutional Neural Networks (**CNNs**) have been the go-to model for computer vision tasks. They **are specifically designed to handle grid-like data** and **have built-in assumptions** about the nature of images (such as the proximity of pixels and the idea that parts of image are more important than others).
Transformers treat input data as a sequence and do not have these built-in assumptions. This can be an advantage because it allows the model to **learn the important features from the data itself without being constrained by prior assumptions**. It also means that Transformers can potentially model **long-range dependencies between pixels in an image**, which can be difficult for CNNs.

Methodology
• **Patch extraction and embedding**: Images are divided into fixed-size patches, typically 16x16 pixels.
• **Position embeddings**: Since the Transformer model doesn't have any inherent notion of the spatial arrangement of the patches, position embeddings are added to the patch embeddings to retain positional information.
• **Class token**: This token is used to accumulate information about the entire image and its output embedding is used for classification.
• **Transformer encoder**: The sequence of patch embeddings, along with the class token, is passed through a standard Transformer encoder.
This procedure is essentially **a translation of the standard Transformer training procedure to the domain of images.**

8 Graph Convolutional Networks
Motivation (Why GCN)-CNN underlying
Standard Convolutional Neural Networks (**CNNs**) are primarily **designed for grid-like data structures** where the data has a clear spatial or temporal order. This allows them to **take advantage of locality and stationarity** in the data.
Graph data does not have these properties. Graphs are typically irregular and non-Euclidean, meaning they don't have a standard grid-like structure and the concept of locality is not as straightforward as in images. The convolution operation in a standard CNN, which involves applying a filter to a local region of the input, doesn't translate directly to graph data.
The **order of nodes in a graph is not meaningful**, and nodes can be arbitrarily relabeled without changing the structure of the graph.
Spatial Approach to GCNs would involve **gathering features from the neighboring nodes**, **applying some sort of transformation or weighting to these features**, and then using these to **update the features of the original node**. This process is **analogous to the convolution operation** in Convolutional Neural Networks where local neighborhoods are considered in input data.
Spectral Approach is based on spectral graph theory, which studies the properties of graphs **in the spectral domain**, specifically using the graph Laplacian and its eigenvectors. The spectral approach defines convolution operations in the spectral domain by **multiplying** the **spectral representation of the graph signals** with **a learned spectral filter**.

Spatial vs. Spectral
• **Interpretability**: The spatial approach operates directly on the graph structure and can be more interpretable than the spectral approach
• **Flexibility**: The spatial approach can handle different graph structures, including dynamic graphs, more easily than the spectral approach because it doesn't rely on the graph Laplacian. If the graph structure is not fixed or if there is a need to **generalize to unseen graph structures**, the spatial approach might be a better choice.
• **Efficiency**: The spectral approach can be more computationally efficient for certain types of graph structures
• **Theoretical grounding**: The spectral approach has a stronger connection to the underlying theory of graphs

9 Deep Learning Applications
Region CNN (R-CNN)
The original R-CNN algorithm works as follows:
• **Region Proposal**: First, it **generates potential bounding boxes in the image**. This was typically done using a separate algorithm like **Selective Search**.
• **Feature Extraction**: For **each region proposal**, the R-CNN algorithm **extracts a fixed-length feature vector** using a Convolutional Neural Network (CNN).
• **Classification**: **Each feature vector is fed into a separate classifier** (like a Support Vector Machine) to determine the object class, and **a regression model** to refine the bounding box coordinates.

Some limitations:
• **It's slow**: Since it applies the CNN to each region proposal separately
• **Training is complex**: The R-CNN model involves training several separate components (the CNN, the classifier, and the bounding box regressor)
• **It requires a lot of disk space**: For each region proposal of each image, a feature vector is stored on disk which makes the model space inefficient.
• **Duplication of calculation**: Since the region proposals often overlap, the same pixels can be included in multiple region proposals.
• **Not in end-to-end manner**: The original R-CNN model is not trained in an end-to-end manner because its components are trained separately and sequentially

Selective Search
The idea behind Selective Search is to **replace the exhaustive search of sliding window approaches** with a more efficient and effective method that's guided by the image content itself. Here's how it works:
• **Segmentation**: The algorithm starts by over-segmenting the image based on intensity of the pixels using a method such as the Felzenszwalb and Huttenlocher algorithm.
• **Hierarchical Grouping**: Following the initial segmentation, regions are hierarchically merged based on similarities in color, texture, size, etc.
• **Bounding Box Proposals**: After the hierarchical grouping, the bounding boxes of all the regions at all the levels of the hierarchy are output as the final set of region proposals.

Fast R-CNN
It introduces several improvements that **address the limitations of R-CNN**.
Here are the key differences and advantages:
• **Shared Computation**: Fast R-CNN applies the CNN to the entire image once, and then extracts features from the resulting feature map for each region proposal.
• **End-to-End Training**: Fast R-CNN is trained end-to-end on a multi-task loss. This combines the losses for classification and bounding box regression.
• **ROI Pooling**: ROI Pooling is a type of max-pooling. The output always has the same size, which allows it to extract fixed-size feature vectors from region proposals of any size.
• **Speed and Efficiency**: Due to the shared computation and other improvements, Fast R-CNN is much faster and more efficient than R-CNN.
• **Improved Accuracy**: Fast R-CNN achieves higher accuracy than R-CNN

Faster R-CNN = Fast R-CNN + RPN
Region Proposal Network
The RPN in Faster R-CNN is **a fully convolutional network** that is designed to generate region proposals **directly from the feature map** of the input image.
The RPN scans the feature map with a sliding window and for each window, **outputs a number of potential bounding boxes** (called "anchors") and **scores for each bounding box** representing **how likely it is to contain an object**.
By integrating the region proposal step into the model, **Faster R-CNN allows for end-to-end training of the whole model**, including the region proposal step, which can lead to better performance.
Additionally, because the RPN operates on the feature map of the image, it can **share computation with the rest of the model**, making Faster R-CNN **more efficient** than Fast R-CNN when the number of region proposals is large.

Mask R-CNN = Instance Segmentation + Faster R-CNN.
Motivation
The authors observed that the **bounding box outputs of Faster R-CNN** could be supplemented **with a parallel branch for predicting binary masks**, without a significant increase in computational complexity.

Extension to Faster R-CNN
• **Additional Mask Branch**: Mask R-CNN adds an extra branch into Faster R-CNN, which outputs a binary mask that specifies whether or not each pixel in the object's bounding box belongs to the object. This is done **in parallel** with the existing branch for bounding box recognition in Faster R-CNN.
• **ROIAlign**: Mask R-CNN introduces a method called "ROIAlign" to address a misalignment problem between the extracted features and the input.

10 Deep Generation Models
Why Generative Models?
• **Data Generation**: One of the main reasons we study generative models is for the ability to generate new data samples that resemble the training data.
• **Understanding Data**: Generative models can help understand the underlying structure and distribution of the data.
• **Semi-Supervised Learning**: Generative models can be used in semi-supervised learning settings, where we have a large amount of unlabeled data and a small amount of labelled data.
• **Anomaly Detection**: Generative models can be used for anomaly detection because they can learn the normal distribution of data.
• **Missing Data Imputation**: Generative models can be used to fill in missing values in a dataset.

Generative Adversarial Networks (GAN)
GAN consists of two models: **a discriminator D** and **a generator G**. These two models compete against each other during the training process
• **Generator (G)**: The generator's role is to create new data instances. The goal of the generator is to **produce data that is as close as possible to the real data distribution**.
• **Discriminator (D)**: The discriminator's role is to **classify instances as real** (coming from the real data distribution) **or fake** (coming from the generator).
Training GANs involves **carefully maintaining the balance** between G and D to ensure that both continue to learn.

Issues of GAN
The process is known to be **slow and unstable**. Training a GAN faces a **dilemma**: If the discriminator behaves badly, the generator does not have accurate feedback and the loss function cannot represent reality; If the discriminator does a great job, the gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed.

Deep Convolutional GANs (DCGANs)
What's New?
• Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
• Use batchnorm in both the generator and the discriminator.
• Remove fully connected hidden layers for deeper architectures.
• Use ReLU activation in generator for all layers except for the output, which uses Tanh.
• Use LeakyReLU activation in the discriminator for all layers.

Motivation
DCGAN came from the desire to **improve upon the stability of training traditional GANs** and generate higher quality samples.

Benefits of DCGAN
• **Improved Training Stability**: DCGANs introduced several architectural choices and training procedures that improved the stability of training GANs.
• **Higher Quality Samples**: By leveraging the power of convolutional networks, DCGANs were able to generate higher quality samples.
• **Feature Learning**: DCGANs were found to learn useful hierarchical representations from images.
• **Spatially Coherent Outputs**: DCGANs were found to generate images with better spatial coherence due to the use of convolutional layers.

Wasserstein GAN
The motivation behind WGANs comes from the problems associated with the standard GANs, such as mode collapse, where the generator only produces a limited variety of samples, and unstable training, where the generator and discriminator loss functions oscillate wildly.

Benefits of WGAN
• **More Stable Training**: The Wasserstein loss function provides smoother gradients and hence more stable training.
• **Avoids Mode Collapse**: WGANs are less susceptible to mode collapse compared to standard GANs.
• **Loss Correlates with Quality**: The Wasserstein loss correlates better with the quality of generated samples.
• **No Need for Logarithm in Loss**: In WGAN, the loss function is simpler and doesn't involve a logarithm, making it less prone to such issues.

Variational Auto-Encoder (VAE)
Benefits and motivations of VAEs compared to AEs (autoen-coders):
• **Stochasticity and Continuity**: Unlike AEs, VAEs are designed to model a continuous probability distribution over the input data.
• **Generative Capability**: VAEs are explicitly designed as generative models.
• **Data Compression**: Like AEs, VAEs can be used for data compression and noise reduction.
• **Solving Overfitting**: VAEs introduce a regularization term in the loss function (the KL-divergence term) that forces the model to learn a well-structured latent space and avoid overfitting.

The purpose of Sampling
• **Sampling a normal distribution**: The reparameterization trick involves sampling from a normal distribution. The reparameterization trick **allows the model to backpropagate** through the random sampling operation
• **Learning mean and variance parameters**: The purpose of learning these parameters is to capture the structure and variability of the data.

Methodology
• **Encoding**: In this step, the input data (for example, an image) is passed through an encoder neural network. **Outputs two vectors** of the same dimensionality as the desired latent space: one for the **means (mu)** and one for the **standard deviations (sigma)**.
• **Sampling**: Once we have the parameters of the Gaussian distribution, we **sample a point from this distribution**.
• **Reparameterization**: Instead of directly sampling from the Gaussian distribution, we **sample from a standard normal distribution**, and then scale and **shift the sample using the mean and standard deviation vectors output by the encoder**.
• **Decoding**: The sampled point from the latent space is then passed through a decoder neural network, which **reconstructs the original input data**.

GAN vs. VAE
There are some reasons why GANs are often perceived as generating higher-quality results than VAEs, especially for tasks like generating realistic images:
• **Sharpness of generated samples**: GANs tend to produce sharper, more detailed images than VAEs.
• **Ability to model complex distributions**: The adversarial training procedure of GANs allows them to model more complex distributions than VAEs.
• **Absence of an explicit likelihood**: VAEs optimize a lower bound on the data likelihood, which can be computationally challenging and may not directly correspond to the quality of the generated samples.
However, It's worth noting that **GANs** also have their **own challenges**, such as **mode collapse and difficulty in training**. **VAEs** are typically **easier to train** and provide **a clear measure of the quality** of the learned model in the form of the data likelihood or its lower bound.

```
class RNNModel(nn.Module):
   def __init__(self, in_feature, hidden_size, n_class):
      super(RNNModel, self).__init__()
      self.in_feature = in_feature
      self.hidden_size = hidden_size
      self.n_class = n_class
  self.fully_connected = nn.Linear(in_feature+self.hidden_size, self.hidden_size)
      self.pred_layer = nn.Linear(self.hidden_size, self.n_class)
      self.tanh = nn.Tanh()
   def forward(self, input, dtype=torch.float):
      T = input.shape[0], batch_size = input.shape[1]
      outputs = torch.zeros(size=(T, batch_size, self.hidden_size), dtype=dtype)
      state = torch.zeros(size=(batch_size, self.hidden_size), dtype=dtype)
      for t in range(T):
         concat = torch.cat([input[t], state], dim=1)
         state = self.tanh(self.fully_connected(concat))
         outputs[t] = state
      return outputs, state
   def predict(self, input_state, dtype=torch.float):
      _, last_state = self.forward(input_state)
      predict = self.tanh(self.pred_layer(last_state))
      return predict
def create_maps():
      dic = {}, counter = 1
      for i in range(ord('a'), ord('z') + 1):
            dic[chr(i)] = counter
            counter += 1
      return dic
def word_embedding(input_seq, vob_size, dtype=torch.float):
      word_embed = nn.Embedding(vob_size, embedding_size)
      embeddings = word_embed(input_seq.long())
      return embeddings
```