Module 1: Artificial Neural Network An Artificial Neural Network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information. They are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown.

- **Perceptron**: The simplest type of artificial neuron, which takes a set of inputs and returns an output based on a linear combination of the inputs.
- **Backpropagation**: The primary algorithm for performing gradient descent on neural networks. It iteratively adjusts the weights of connections in the network in order to minimize the difference between the actual and desired output.

Module 2: Deep Learning Deep Learning is a subfield of machine learning that uses algorithms inspired by the structure and function of the brain's neural networks.

- **Hyperparameters**: These are parameters whose values are set prior to the commencement of the learning process. Examples include learning rate, batch size, number of layers, number of units in a layer, etc.
- **Dropout**: A technique used in neural networks to prevent overfitting. It temporarily drops out neurons (along with their connections) from the network during training.
- **Weight Initialization**: The process of setting the initial values of the weights in a neural network.

Module 3: Convolutional Neural Networks (CNN) CNNs are a type of deep learning model that can process grid-like data (like images) with their spatial relationships intact.

- **Sparse Connectivity**: In CNNs, each neuron in a layer is connected only to a small region of the layer before it, instead of all of the neurons in the previous layer. This is also referred to as local receptive fields.
- **Popular Architectures**: Architectures in deep learning refers to the design of the neural network. Examples include AlexNet, VGG, and ResNet.

Module 4: Sequence Modeling in Neural Network Sequence modeling is a type of learning where the order of data matters. Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks are examples of sequence models.

- **RNN**: A type of neural network that has an internal loop, which allows information to be passed from one step in the network to the next.
- **LSTM**: A type of RNN that can learn and remember over long sequences and is not sensitive to the gap length.

Module 5: Autoencoders with Deep Learning Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional code and then reconstruct the output from this representation.

- **Regularized Autoencoders**: Autoencoders that have a constraint added to the loss function to prevent overfitting.
- **Variational Autoencoders (VAE)**: A type of autoencoder that produces a compressed representation of the input and can generate new data that has the same characteristics as the original input.

Module 6: Generative deep learning models Generative models are a subclass of unsupervised learning that generate new sample/data by understanding true data distribution of the training set.

- **Boltzmann Machine**: A type of stochastic recurrent neural network.

- **Generative Approach with Autoencoder and VAE**: Autoencoders and VAEs can be used to generate new data that is similar to the training data.
- **GAN Applications**: Generative Adversarial Networks (GANs) are deep neural net architectures composed of two nets, pitting one against the other

Module 1: Artificial Neural Networks (ANN).

1. **Introduction:** ANNs are computing systems that are modeled after the human brain's network of neurons. They're composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements.
2. **Background of NN:** Neural Networks have been around since the 1940s. The earliest models were simple circuits that tried to replicate the neuron activity of the human brain. Over time, these models have evolved and have been used in various applications from pattern recognition, forecasting, data compression to controls.
3. **Introduction to Perceptron:** A perceptron is a simple model of a biological neuron in an artificial neural network. Perceptron takes multiple binary inputs x and produces a single binary output.
4. **Perceptrons in Layers and Network:** Perceptrons can be arranged in layers to form a neural network. The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes connected to multiple layers of input nodes.
5. **Perceptron Training (Delta Rule):** This is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network. It is a simple form of the more general backpropagation algorithm.
6. **Importance of Non-Linearity:** Non-linear activation functions allow neural networks to compute arbitrary functions, whereas with a linear activation function, no matter how many layers the network has, it can still compute only linear mappings, which are simple tasks.
7. **Backpropagation:** This is the primary learning algorithm for neural networks. It works by calculating a weight change amount (ΔW) for every weight in the network that proportionally reduces the error in the output.
8. **Generalization and stopping criteria:** Generalization is the model's ability to give reasonable outputs for inputs not present in the training dataset. Stopping criteria is a condition to stop training the model. If we don't set a stopping criterion, the model can overfit the training data, leading to poor generalization to new, unseen data.

Module 2: Deep Learning.

1. **Introduction to Deep Learning:** Deep learning is a subset of machine learning that's based on artificial neural networks with representation learning. It can process a wide range of data resources, requires less preprocessing by humans, and can often produce more accurate results than traditional machine learning approaches.
2. **Deep Learning and ANN:** Deep learning utilizes large neural networks with many layers (hence the 'deep' in deep learning) to learn complex patterns from large amounts of

data. These deep networks have significantly more power to fit complex patterns in data as compared to shallow or less deep networks.

3. **Hyperparameters:** Hyperparameters are variables that define the network structure (e.g., number of hidden units, learning rate, etc.) and how the network is trained (e.g., number of epochs, batch size, etc.). They are set before training (optimizing the weights and biases).
4. **Loss Landscape:** This refers to the high-dimensional surface created by the loss function. Each dimension corresponds to a weight in the network, and the height corresponds to the loss. The aim of training a neural network is to find the minimum of this surface, which would correspond to the best weights for the network.
5. **Weight Decay:** This is a regularization technique by adding an additional term to the loss function that penalizes certain configurations of the weights (typically, it discourages large weights).
6. **Dropout:** This is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is an efficient way to combine the predictions of many different large neural nets at test time.
7. **Other Methods:** There are many other methods and techniques in deep learning, such as batch normalization, data augmentation, etc. These methods help to improve the performance of the network and speed up the training process.
8. **Weight Initialization and Training:** The weights in a neural network must be initialized to small random numbers. The network is then trained by iteratively adjusting these weights to minimize the difference between the predicted and actual outcome.

Deep learning is a powerful tool for solving problems of prediction, image classification, and natural language processing. It is widely used in various fields including healthcare, finance, transportation, and more.

Module 3: Convolutional Neural Networks (CNN) with Deep Learning.

1. **CNN Prerequisites: Computer Vision:** Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. CNNs are widely used in computer vision tasks like image classification, object detection, etc.
2. **CNN 1-D, CNN 2-D:** CNNs can be applied to 1-D data (like time series or audio signals) and 2-D data (like images). The difference is in how convolutional layers, pooling layers, and the filters/kernels are structured.
3. **CNN Classification Pipeline:** This involves preprocessing the data, designing the model architecture (convolutional layers, pooling layers, fully connected layers), training the model, and evaluating its performance.
4. **Sparse Connectivity in CNN:** This refers to the idea that each neuron in a convolutional layer is connected only to a small region of the input volume and not the whole input, which is often the case in fully connected neural networks. This helps reduce the number of parameters and computational complexity.
5. **Popular Architectures:** There are many pre-defined architectures in the field of Deep Learning which have been proposed over the years, such as AlexNet, VGG-16, GoogleNet, and ResNet. These architectures are designed to solve specific problems.

6. **CNN Case Studies (U-Net, R-CNN, Fast R-CNN, Yolo):** These are different architectures or approaches used for specific tasks. For example, U-Net is used for biomedical image segmentation, while R-CNN, Fast R-CNN, and Yolo are used for object detection in images.

CNNs have been successful in various text and image recognition tasks and are now widely used in the field of computer vision. They have been a key component behind the leap in the performance of AI systems over the past few years.

Module 4: Sequence Modeling in Neural Networks.

1. **Sequence Data:** Sequence data is a type of data where the order of values is important. Examples of sequence data include time series data, sentences, protein sequences, and more.
2. **Modelling Sequence:** Sequence modeling involves using statistical methods to predict the next value in the sequence or the whole sequence itself. Models for these types of data are called sequence models. Examples of sequence models include Hidden Markov Models (HMMs), Recurrent Neural Networks (RNNs), and Transformers.
3. **Recurrent Neural Network (RNN):** RNNs are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, or the spoken word. They are networks with loops in them, allowing information to persist.
4. **Training RNN:** Training an RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a twist. Because of the recurrent nature of an RNN, we perform backpropagation for each time step in the input sequence - this is known as Backpropagation Through Time (BPTT).
5. **LSTM (Long Short Term Memory):** Long Short Term Memory (LSTM) is a type of RNN that solves the problem of learning long-term dependencies. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network.
6. **Applications of LSTM:** LSTMs are widely used for sequence prediction problems and have proven to be extremely effective. They are used for time series prediction, natural language processing, speech recognition, among others.

In summary, sequence modeling in neural networks is an essential concept especially when dealing with data where temporal dynamics and the order of inputs contain information. LSTM networks are a great way to achieve this as they overcome the challenges posed by the standard RNNs.

1. **Introduction to Reinforcement Learning (RL):** RL is a type of machine learning where an agent learns to make decisions by taking actions in an environment to achieve a goal. The agent learns from the consequences of its actions, rather than from being explicitly taught and it selects its actions based on its past experiences (exploitation) and also by new choices (exploration).

2. **Markov Decision Process (MDP):** Most reinforcement learning problems can be modeled as MDPs. MDPs provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision-maker.
3. **Policy & Value Iteration:** Policy iteration and value iteration are two fundamental methods to solve MDPs. Both methods consist of a sequence of computations involving the values of states until convergence.
4. **Q-Learning:** Q-learning is a values based algorithm in reinforcement learning. It is a table of values for every state (row) and action (column) possible in the environment. The Q-value represents the expected future reward of that action at that state.
5. **Deep Q-Learning:** Deep Q-Learning is the combination of Q-Learning and deep learning. It uses neural networks to estimate Q-values and make the algorithm work with high-dimensional states and/or actions.
6. **Policy Gradients:** Policy gradients are an alternative to Q Learning. Instead of learning a value function that tells us what is the expected sum of rewards given a state and an action, policy gradients aim to learn directly the policy function that will tell us what action to take given a state.
7. **Applications of RL:** RL can be used in a variety of applications including robotics, video games, finance, and healthcare. It's particularly good at problems that require a sequence of decisions.

Reinforcement Learning is a crucial part of AI. It helps software agents and machines to automatically determine the ideal behavior within a specific context, to maximize its performance.

Module 6: Generative Adversarial Networks (GANs).

1. **Introduction to GANs:** GANs are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework. They were introduced by Ian Goodfellow and other researchers at the University of Montreal in 2014.
2. **Structure of GANs:** A GAN consists of two parts - a generator network and a discriminator network. The generator network takes in random noise as input and produces a data instance as output. The discriminator takes in both real and fake (generated by the generator) data instances and predicts whether they are real or fake.
3. **Training GANs:** The GAN is trained by alternately optimizing the discriminator to correctly classify real and fake data, and optimizing the generator to fool the discriminator. This can be seen as a game between the generator and the discriminator where the generator tries to fool the discriminator, and the discriminator tries to catch the generator.
4. **Applications of GANs:** GANs have been used for various applications like image synthesis, semantic image editing, style transfer, image super-resolution and much more.
5. **Problems & Solutions:** Despite their potential, GANs are notoriously difficult to train. Problems may include mode collapse, vanishing gradients, and instability. There are several techniques to mitigate these issues, such as Wasserstein GANs, gradient penalty, and spectral normalization.

6. **Popular GAN models:** There are several variations of GANs, including DCGANs (Deep Convolutional GANs), CycleGANs, and StyleGANs, each with their own unique properties and use cases.

GANs are a very exciting development in the field of machine learning and AI, and have opened up new possibilities in the areas of creativity, design, and personalization.

1. Perceptron Networks

Formula:

- **Weighted Sum:** $z = \sum_{i=1}^{n} w_i x_i + b$
  - w_i: weights
  - x_i: input features
  - b: bias
- **Activation Function (Step Function):**
  $$\text{output} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Example:

- Inputs: $x = [1, 0, 1]$
- Weights: $w = [0.5, -0.6, 0.2]$
- Bias: $b = 0.1$
- $z = 0.5 \cdot 1 + (-0.6) \cdot 0 + 0.2 \cdot 1 + 0.1 = 0.8$
- Output: 1 (since $z > 0$)

2. Backpropagation

Formulas:

- **Loss Function (Mean Squared Error):**
  $L = \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
- **Gradient of Loss w.r.t Weights:**
  $\frac{\partial L}{\partial w_j} = \sum_{i=1}^{n} (y_i - \hat{y}_i) \cdot \hat{y}_i \cdot (1 - \hat{y}_i) \cdot x_{ij}$

Example:

- Predicted output: $\hat{y} = [0.4, 0.6]$
- True output: $y = [0.5, 0.5]$
- $L = \frac{1}{2} \left[(0.5 - 0.4)^2 + (0.5 - 0.6)^2 \right] = \frac{1}{2} [0.01 + 0.01] = 0.01$

3. Convolutional Neural Networks (CNNs)

Formulas:

- **Convolution Operation:**
  $(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n)$
  - I: input image
  - K: kernel/filter
- **Pooling Operation (Max Pooling):**
  $\text{output}(i, j) = \max(\text{region})$

Example:

- Input Image: $I = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
- Kernel: $K = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
- Convolution Output:
  $\begin{bmatrix} 1*1+2*0+4*(-1)+5*1 & 2*1+0*0+5*(-1)+6*1 \\ 4*1+5*0+7*(-1)+8*1 & 5*1+6*0+8*(-1)+9*1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$

4. Recurrent Neural Networks (RNNs)

Formulas:

- **Hidden State:**
  $h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$
  - σ: activation function (e.g., tanh, ReLU)
  - W_{xh}: weight matrix for input to hidden state
  - W_{hh}: weight matrix for hidden state to hidden state
  - x_t: input at time step t
  - h_{t-1}: hidden state at previous time step
  - b_h: bias
- **Output:**
  $o_t = \sigma(W_{ho}h_t + b_o)$
  - W_{ho}: weight matrix for hidden state to output
  - b_o: bias

Example:

- Input: $x_t = [1, 0.5]$
- Previous hidden state: $h_{t-1} = [0.2, -0.1]$
- Weights: $W_{xh} = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & -0.1 \end{bmatrix}, W_{hh} = \begin{bmatrix} 0.5 & 0.4 \\ 0.3 & 0.2 \end{bmatrix}$
- Bias: $b_h = [0.1, -0.1]$
- $h_t = \tanh(\begin{bmatrix} 0.1 & 0.3 \\ 0.2 & -0.1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.4 \\ 0.3 & 0.2 \end{bmatrix} \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.1 \end{bmatrix}) = \tanh(\begin{bmatrix} 0.45 \\ 0.23 \end{bmatrix})$

5. Long Short-Term Memory (LSTM)

Formulas:

- **Forget Gate:**
  $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- **Input Gate:**
  $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
  $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
- **Cell State:**
  $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
- **Output Gate:**
  $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
  $h_t = o_t * \tanh(C_t)$

Example:

- Input: $x_t = [1, 0.5]$
- Previous hidden state: $h_{t-1} = [0.2, -0.1]$
- Previous cell state: $C_{t-1} = [0.5, 0.3]$
- Weights: W_f, W_i, W_C, W_o are weight matrices
- Biases: b_f, b_i, b_C, b_o are biases

6. Autoencoders

Formulas:

- **Encoding:**
  $h = \sigma(W_e x + b_e)$
- **Decoding:**
  $\hat{x} = \sigma(W_d h + b_d)$

Example:

- Input: $x = [1, 0.5]$
- Weights: $W_e = \begin{bmatrix} 0.5 & 0.4 \\ 0.3 & 0.2 \end{bmatrix}, W_d = \begin{bmatrix} 0.5 & 0.3 \\ 0.4 & 0.2 \end{bmatrix}$
- Biases: $b_e = [0.1, -0.1], b_d = [0.2, 0.1]$
- $h = \tanh(\begin{bmatrix} 0.5 & 0.4 \\ 0.3 & 0.2 \end{bmatrix} \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.1 \end{bmatrix}) = \tanh(\begin{bmatrix} 0.85 \\ 0.4 \end{bmatrix})$

7. Variational Autoencoders (VAE)

Formulas:

- **Encoder:**
  $\mu = f_\mu(x), \quad \log \sigma^2 = f_{\log \sigma^2}(x)$
- **Sampling:**
  $z = \mu + \sigma \cdot \epsilon$
  - $\epsilon \sim \mathcal{N}(0, 1)$
- **Decoder:**
  $\hat{x} = g(z)$
- **Loss:**
  $L = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)||p(z))$

Example:

- Input: $x = [1, 0.5]$
- Encoder networks provide μ and $\log \sigma^2$
- Sample z from $\mathcal{N}(\mu, \sigma^2)$
- Decoder reconstructs $\hat{x}$

8. Generative Adversarial Networks (GANs)

Formulas:

- **Generator Loss:**
  $L_G = -\mathbb{E}_{z \sim p_z(z)}[\log D(G(z))]$
- **Discriminator Loss:**
  $L_D = -\mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] - \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$

Example:

- Random noise z is fed into the generator G
- Generated sample $G(z)$ is evaluated by the discriminator D
- Discriminator updates its parameters to differentiate real data from generated data
- Generator updates its parameters to improve the realism of generated samples

These formulas cover the essential concepts and operations in deep learning. Let me know if you need more detailed explanations or additional examples!