Lecture 09:

Early Stopping: To prevent overfitting, training should be stopped when the validation loss ceases to decrease. This involves monitoring the performance on a validation set that is not used during training.

Dropout: A regularization technique that randomly drops neurons (with a probability p) during training to prevent overfitting. For each training step, a new "thinned" network is used, leading to a kind of ensemble effect. During testing, no dropout is applied, but the weights are adjusted (multiplied by 1-p) to compensate for the dropout rate used during training. This method is akin to having an ensemble of networks with shared weights, offering a robust way to generalize the model.

Normalization: Techniques like Max-Min Normalization and Standardization (Z-score normalization) are used to scale the input features. Batch normalization is specifically highlighted for its ability to accelerate training by reducing internal covariate shift, which is the shifting distribution of layer inputs during training. Batch normalization standardizes the inputs to a layer for each mini-batch, thus stabilizing the learning process.

Data Augmentation: The process of transforming input data in various ways to enhance the diversity of the training set, which helps in improving the model's generalization. It includes techniques like spatial transformations learned by the network to automatically adjust images for better processing.

Regularization - Focal Loss: Introduced for dealing with class imbalance by focusing more on hard-to-classify samples and less on well-classified ones. It automatically balances the contribution of different samples to the loss, making the model more robust to imbalanced data.

Visualization Techniques:

T-SNE (t-Distributed Stochastic Neighbor Embedding): A tool for visualizing high-dimensional data by mapping each data point to a two or three-dimensional space. It helps in understanding how well the classification model is performing by visualizing the separation between different classes.

Class Activation Map: Helps in understanding whether the model is capturing class-specific features by visualizing areas of the image that are important for predicting a particular class.

These techniques collectively aim to improve the training process, reduce overfitting, enhance model generalization, and provide insights into model performance and behavior.

Lecture 10:

CNNs are designed to automatically and adaptively learn spatial hierarchies of features from images for tasks like image classification.

The presentation discusses popular vision tasks and datasets, such as MNIST, CIFAR-10, CIFAR-100, and ImageNet, highlighting the challenges of image classification due to variations in viewpoint, scale, deformation, occlusion, illumination conditions, background clutter, and intra-class variation.

Convolution Operation:

Convolution layers use filters or kernels to extract features from images, with each filter activating certain features present in the image.

Key properties of convolution in CNNs include the ability to recognize patterns without seeing the whole image, reuse of parameters across different regions of the image (translational invariance), and robustness to changes in the object's appearance or position within the image.

Pooling:

Pooling layers reduce the spatial size of the representation, decrease the number of parameters and computation in the network, and hence control overfitting. Max pooling and average pooling are the two common types of pooling techniques.

CNN Architecture:

A typical CNN architecture consists of several convolution and pooling layers followed by fully connected layers. The convolution layers act as feature extractors, and the fully connected layers perform classification based on these features.

The presentation explains how convolution over volume, stride, padding, and the calculation of output size are essential aspects of designing CNN architectures.

Convolution vs. Fully Connected Layers:

Convolution layers use much fewer parameters compared to fully connected layers due to the localized nature of the connections and shared weights, leading to more efficient learning and better generalization.

Max Pooling vs. Average Pooling:

The presentation contrasts max pooling, which preserves the maximum value, with average pooling, which computes the average of values in a certain region. Max pooling is more common because it tends to preserve the detected features better.

CNN in Keras:

Demonstrates how to implement a simple CNN in Keras, showing the syntax for adding convolution and pooling layers. The example emphasizes how the input format changes from a vector to a 3-D tensor, accommodating the spatial dimensions of the image and the color channels.

Advanced Concepts in Convolution:

Further discussions include advanced concepts like stride, padding, dilation, and groups within convolution operations, explaining their significance and impact on the model's performance.

AlexNet Example:

A brief overview of AlexNet, one of the first successful CNN architectures for large scale image classification, is provided, including its structure and how it processes inputs through its layers.

Training and Inference with CNNs:

The process of training a CNN involves loading data, defining the network architecture, specifying the loss function and optimizer, and iteratively training the network. Inference involves passing new images through the trained network to predict labels.

The presentation encapsulates the essence of CNNs, their operational principles, key architectural elements, and their implementation using frameworks like Keras, providing a foundational understanding for those interested in deep learning for visual recognition tasks.

Lecture 12:

RNN Introduction and Applications:

RNNs are well-suited for applications like slot filling in natural language understanding, where the goal is to identify specific pieces of information (slots) from the input text, such as dates or locations.

Basic Concept of RNN:

Traditional feedforward neural networks process inputs independently, without considering any order or sequence. In contrast, RNNs maintain a form of memory by using their output as part of the input for the next step, making them ideal for sequential data.

RNN Structure:

An RNN has a simple structure where each node is like a mini neural network that takes two inputs: the current input and the previous output. This allows the network to maintain information across the sequence.

Challenges with RNNs:

Training RNNs can be difficult due to vanishing and exploding gradients. These issues make it hard for the network to learn long-range dependencies.

LSTM and GRU:

To overcome the limitations of traditional RNNs, LSTM (Long Short-Term Memory) units introduce gates that control the flow of information, making it easier for the network to remember or forget information. GRUs (Gated Recurrent Units) are a simpler variant of LSTMs.

Bidirectional RNNs:

These networks consist of two RNNs going in opposite directions, allowing the model to have information from both past and future states simultaneously, improving the context understanding.

Sequence to Sequence Models:

RNNs can be used for tasks that require understanding a sequence input and producing another sequence as output, such as machine translation or summarization. This is achieved through encoder-decoder architectures where one RNN encodes the input sequence, and another decodes it to the target sequence.

Training RNNs

Training involves backpropagation through time (BPTT), a variation of traditional backpropagation used for feedforward networks, tailored for the sequential nature of RNNs. However, this can lead to computational challenges.

Gradient Clipping:

A technique used to prevent exploding gradients by limiting the size of the gradients to a defined range.

Applications Beyond Text:

RNNs are not limited to text and can be applied to any sequence data, including time series analysis, speech recognition, and even in the generation of music.

Advanced RNN Features in Keras:

Keras provides high-level APIs for building RNNs, including support for LSTMs, GRUs, and bidirectional RNNs, along with options to control output sequences, return states, and manage the internal states of the networks for complex sequence modeling tasks.

Lecture 13:

Auto-Encoders:

Concept and Application: Auto-encoders are unsupervised learning models designed to learn compact representations (encodings) of data, then reconstruct the original input from these encodings. They consist of an encoder that compresses the input and a decoder that reconstructs the input from the compressed form.

Deep Auto-Encoders: By stacking multiple layers, deep auto-encoders can learn more complex representations, offering better compression and reconstruction capabilities compared to shallow networks or traditional dimensionality reduction techniques like PCA.

De-noising Auto-Encoders: A variant that learns to reconstruct the original clean input from corrupted versions, thus learning robust features.

Encoder-Decoder Models

Basic Architecture: RNN-based encoder-decoder architectures where the encoder generates a contextual representation of the input, and the decoder generates output sequences from this representation. The final state of the encoder serves as the initial context for the decoder.

Challenges: Encoder-decoder models initially struggled with maintaining context over long sequences, leading to the development of attention mechanisms.

Attention Mechanism

Purpose and Function: The attention mechanism allows models to dynamically focus on different parts of the input sequence when generating each word of the output sequence, effectively solving the long-range dependency problem.

Implementation: Attention involves computing a context vector for each output time step by weighting the importance of each input state based on the current output state.

Transformers

Introduction: The Transformer model, introduced by Vaswani et al. in "Attention is All You Need" (2017), eschews recurrence and relies solely on attention mechanisms to draw global dependencies between input and output.

Components: Transformers consist of an encoder and a decoder, both composed of multiple layers of self-attention and position-wise, fully connected layers for both the encoder and decoder.

Key Concepts and Applications

Auto-regressive Models: Models that predict future outputs based on a linear combination of past values, widely used in NLP for predicting the next token in a sequence.

Sequence-to-Sequence Learning: The encoder-decoder architecture is foundational for sequence-to-sequence (seq2seq) tasks, such as machine translation, where the input sequence (e.g., a sentence in English) is converted into an output sequence (e.g., the corresponding sentence in French) with the help of attention mechanisms to improve accuracy and fluency.

Advanced Topics

De-noising for Image and Text Retrieval: Auto-encoders are not only used for compression but also for tasks like de-noising images and text retrieval, where they can significantly enhance performance by focusing on relevant features.

Cross-Attention: A type of attention that focuses on the interaction between the encoder and decoder states, allowing for more nuanced control and better understanding of the input sequence in generating the output.

This presentation provides a comprehensive overview of the progression from basic encoder-decoder models to the sophisticated Transformer models that dominate NLP tasks today, highlighting the evolution of techniques to address the challenges of sequence modeling and representation learning.