Assignment-13

Dhanush V Nayak
ee23btech11015@iith.ac.in

1 Convolutional Networks

Convolutional Neural Networks (CNNs) are designed for structured input data like images. A fully connected neural network would need a vast number of weights to process large images, which would be computationally expensive. CNNs handle this complexity by using local connections and sharing parameters across different parts of the image, significantly reducing the number of parameters to learn.

CNNs leverage the concept of spatial invariance, allowing them to detect patterns in different regions of the input image using convolution operations. A kernel or in general referred as the filter is applied to local regions of the input image to detect features, such as edges, corners, or textures. The convolution operation can be expressed mathematically as:

$$z_i = \sum_{j=1}^{l} k_j x_{j+i-\frac{l+1}{2}}$$

where z_i is the output at position i, k_j is the kernel, and x is the input.

1.1 Pooling and Downsampling

Pooling layers reduce the spatial dimensions of the data while preserving important features. This operation can be either average-pooling, where the average value of the inputs is taken, or max-pooling, where the maximum value is taken. Pooling typically uses a kernel of size l and a stride s, reducing the size of the image by a factor of s.

For example, after applying pooling, an object that occupied $10s$ pixels would occupy only 10 pixels in the downsampled image. Max-pooling acts like a logical disjunction, focusing on the most important feature detected in a local region.

1.2 Tensor Operations in CNNs

Tensors, which are multidimensional arrays, are widely used in deep learning to represent data shapes. In Convolutional Neural Networks (CNNs), the structure of tensors helps track how data evolves through the layers of a network. The fundamental idea behind CNNs is that adjacent elements in data are semantically related, which makes it useful to apply operations like convolution to localized regions of the data. By using tensors, CNN layers can be concisely described as mappings from one tensor to another, allowing for efficient mathematical representation and computational optimization.

This allows CNNs to process large batches of images simultaneously, improving training speed. For example, if we process a minibatch of 64 RGB images, each with dimensions 256×256, the input tensor has a size of 256×256×3×64. After applying convolutional filters, we get an output tensor, or feature map, of size 128×128×96×64, which contains information about the extracted features from each image. CNNs use these feature maps to make predictions about the data.

1.3 Residual Networks

Residual networks (ResNets) introduce the idea of learning a residual function rather than a full transformation at each layer. This is mathematically represented as:

$$z^{(i)} = g^{(i)}(z^{(i-1)} + f(z^{(i-1)}))$$

where f is the residual function and $g^{(i)}$ is the activation function at layer i. This approach helps prevent the vanishing gradient problem by allowing gradients to flow through the identity connections in deep networks.

2 Learning Algorithms

Training neural networks typically involves minimizing a loss function using gradient-based optimization techniques like Stochastic Gradient Descent (SGD). The update rule for gradient descent is:

$$w \leftarrow w - \alpha \nabla_w L(w)$$

where α is the learning rate, and $L(w)$ is the loss function. For SGD, instead of computing the gradient with respect to the entire training set, we compute it with respect to a random minibatch of m examples.

2.1 Computing Gradients in Computation Graphs

The back-propagation algorithm is used to compute gradients in deep neural networks. For a node h in a computation graph, the partial derivatives of the loss function L with respect to h are calculated as:

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial h_j} + \frac{\partial L}{\partial h_k}$$

where h_j and h_k are the next nodes in the graph. The backward pass propagates these derivatives from the output layer to the input layer.

2.2 Batch Normalization

Batch normalization is used to speed up convergence by normalizing the outputs of each layer. For a node z, batch normalization adjusts its value as:

$$\hat{z}_i = \gamma \frac{z_i - \mu}{\sqrt{\epsilon + \sigma^2}} + \beta$$

where μ is the mean of z in the minibatch, σ is the standard deviation, and ϵ is a small constant to avoid division by zero. γ and β are learnable parameters.

3 Generalization

Generalization refers to the network's ability to perform well on unseen data. Three key methods to improve generalization are selecting the right network architecture, penalizing large weights, and introducing random perturbations during training.

3.1 Choosing a Network Architecture

A great deal of effort in deep learning research has gone into finding network architectures that generalize well. Deeper networks with the same number of weights often generalize better than shallow networks. For instance, Figure 22.7 of the text shows that an 11-layer network outperforms a 3-layer network for the same number of parameters.However, they can make surprising errors, like misclassifying images when slightly altered (adversarial examples). Despite progress, these models can still be vulnerable to adversarial attacks.

3.2 Weight Decay

In deep learning, regularization techniques, such as weight decay, help improve generalization by limiting model complexity. Weight decay adds a penalty term to the loss function, specifically:

$$\lambda \sum_{i,j} W_{i,j}^2$$

where λ is a hyperparameter that controls the strength of the penalty. A value of $\lambda = 0$ means no weight decay, while larger λ values encourage smaller weights. Typical values for λ are around 10^{-4}.

In networks with sigmoid activations, weight decay is believed to keep activations near the linear region, preventing vanishing gradients. However, with ReLU activations and residual connections, the mechanism is different—weight decay promotes small differences between layers rather than small absolute weights.

Another interpretation of weight decay is that it implements maximum a posteriori (MAP) learning. For inputs X and outputs y, the MAP hypothesis is given by:

$$h_{\text{MAP}} = \arg\max_w P(y|X, W)P(W) = \arg\min_w \left[-\log P(y|X, W) - \log P(W)\right].$$

Here, the first term corresponds to the cross-entropy loss, while the second term applies a prior on the weights. If we assume a zero-mean Gaussian prior on the weights, this is equivalent to regularizing the loss with:

$$\log P(W) = -\lambda \sum_{i,j} W_{i,j}^2.$$

3.3 Dropout

Dropout is another regularization technique that randomly deactivates a subset of neurons during training, forcing the network to learn robust representations. For each minibatch, each node has a probability p of being dropped, with typical values being $p = 0.5$ for hidden layers and $p = 0.8$ for input layers. Dropout simulates training multiple networks by approximating an ensemble model.

4 Conclusion

Convolutional Neural Networks (CNNs) have proven highly effective for image processing tasks by leveraging local spatial relationships in data. Techniques like pooling, residual connections, and batch normalization further improve their performance and efficiency. With the help of learning algorithms like stochastic gradient descent, CNNs can be trained to recognize patterns in data, while generalization techniques like weight decay and dropout help prevent overfitting.