CNNs, Learning Algorithms, Generalization

Satyanarayana Gajjarapu
AI24BTECH11009
Department of Artificial Intelligence
ai24btech11009@iith.ac.in

If the first hidden layer is constructed so that each hidden unit receives input from only a small, local region of the image, then adjacency is followed locally and also reduces the number of weights. Images exhibit approximate **spatial invariance**, means eyes or blades of grass look similar regardless of their position within a small region. If l weights are connecting a local region, for hidden units i and j, the weights $w_{1,i}, \cdots, w_{l,i}$ are the same as $w_{1,j}, \cdots, w_{l,j}$.

1 Convolutional Neural Network

A **convolutional neural network** (CNN) contains spatially local connections in the early layers and has patterns of weights that replicates in multiple regions in each layer called **kernel**. The process of applying the kernel to the pixels of the image is called **convolution**. Consider an input vector $\mathbf{x}$ of size n, corresponding to n pixels in one dimensional image and a vector kernel $\mathbf{k}$ of size l. The convolution operation written using $*$, $\mathbf{z} = \mathbf{x} * \mathbf{k}$ is defined as:

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

If the pixels on which the kernels are centered are separated by a distance of p pixels then **stride** is defined as $s = p$, then n pixels are reduced to n/s in one dimension. In two dimensions, n pixels are reduced to $n/s_x s_y$, where s_x and s_y are the strides in the x and y directions in the image. There will be d kernels, with a stride of 1, the output will be d times larger. This means that a 2-D input array becomes a 3-D array of hidden units, where the third dimension is of size d.

CNNs were inspired originally by models of the visual cortex proposed in neuroscience. The **receptive field** of a neuron is the portion of the sensory input that can affect that neuron's activation. The receptive field of a unit in the first hidden layer is small, but larger in the deeper layers of the network. When the stride is 1, a node in the mth hidden layer will have a receptive field of size $(l-1)m+1$.

A **pooling layer** in a neural network summarizes a set of adjacent units from the preceding with a single value. The two types of pooling are **max pooling** and **average pooling**. **Downsampling** refers to reducing the resolution of the feature map by sampling fewer points. A reason for describing CNNs in terms of tensor operations is computational efficiency, deep learning workloads often run on GPUs or TPUs.

Residual networks are a popular and successful approach to building very deep networks that

Preprint. Under review.

avoid the problem of vanishing gradients. These are often used with convolutional layers in vision applications. Using the matrix–vector notation, with $\mathbf{z}^{(i)}$ being the values of the units in layer i

$$\mathbf{z}^{(i)} = f(\mathbf{z}^{(i-1)}) = \mathbf{g}^{(i)}(\mathbf{W}^{(i)}\mathbf{z}^{(i-1)})$$

The key idea of residual networks is that a layer should perturb the representation from the previous layer rather than replace it entirely. $\mathbf{g}_r$ is the activation functions for the residual layer, f is **residual**, **W** and **V** are learned weight matrices with the usual bias weights added.

$$\mathbf{z}^{(i)} = \mathbf{g}_r^{(i)}(\mathbf{z}^{(i-1)} + f(\mathbf{z}^{(i-1)}))$$
$$f(\mathbf{z}) = \mathbf{V}\mathbf{g}(\mathbf{W}\mathbf{z})$$

If **V** = **0** for a particular layer in order to disable that layer, then

$$\mathbf{z}^{(i)} = \mathbf{g}_r(\mathbf{z}^{(i-1)})$$

2 Learning Algorithms

The goal is to minimize the loss L(**w**), where **w** represents all of the parameters of the network. The update algorithm for training is

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\Delta_{\mathbf{w}}L(\mathbf{w})$$

Important characteristics relevant to training neural networks:
•Both the dimensionality of **w** and the size of the training set are very large.
•The gradient contribution of each training example in the SGD minibatch can be computed independently.
•To improve convergence, the learning rate which decreases overtime is used.

The backward message $\frac{\partial L}{\partial h_j}$ is the partial derivative of L with respect to j's first input, which is the forward message from h to j. Now, h affects L through both j and k.

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

$\frac{\partial h}{\partial f_h}$ and $\frac{\partial h}{\partial g_h}$ are just the derivatives of h with respect to its first and second arguments. The back-propagation process begins with the output nodes, where each initial message $\frac{\partial L}{\partial \hat{y}_j}$ is calculated directly from the expression for L in terms of the predicted value $\hat{\mathbf{y}}$ and the true value **y** from the training data.

Batch normalization is a commonly used technique that improves the rate of convergence of SGD. Consider a node z in the network, the values of z for the m examples in a minibatch are $z_1, \cdots, z_m$. Batch normalization replaces each z_i with a new quantity $\hat{z}_i$.

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

where μ is the mean value of z across the minibatch, σ is the standard deviation of $z_1, \cdots, z_m$, ϵ is a small constant added to prevent division by zero, and γ and β are learned parameters.

3 Generalization

A good deal of the deep learning progress in performance has come from exploring different kinds of network architectures and varying the number of layers, their connectivity, and the types of node in each layer. Some neural network architectures are explicitly designed to generalize well on particular types of data. **Neural architecture search** are used to explore the state space of possible network architectures.

In the context of neural networks, regularization is usually called **weight decay**. When weight decay is assumed as maximum a posteriori (MAP) learning, the first term is the usual cross-entropy loss and the second term represents weights. At each step of training, **dropout** applies one step of back-propagation learning to a new version of the network that is created by deactivating a randomly chosen subset of the units.