Convolutional Neural Networks

John Thickstun

Convolutional neural networks (convnets) are a family of functions introduced by LeCun et al. [1989] that we can use to parameterize models. They have a bias towards translation-invariance, which has made them particularly suitable for visual and audio data that exhibit local self-similarity. They also have a locality bias, although this bias is mitigated with depth. Convnets pervade machine learning research (especially in vision); the modern, canonical version of a convnet described in this document was introduced by He et al. [2016]. Maybe the most notable application of convnets is AlexNet [Krizhevsky et al., 2012], which popularized the principle of deep learning.

Image Modeling

A digital image can be represented as tensor $\mathbf{x} \in \mathbb{R}^{C \times h \times w}$, where h and w are the height and width of the image respectively (measured in pixels) and C are the color channels of the image. Conventionally $C = 3$ with color channels in R/G/B (red/green/blue) format; for grayscale images, $C = 1$. We typically normalize the dynamic range of color intensities to the range $[0, 1]$, i.e. $\mathbf{x} \in [0, 1]^{C \times h \times w}$. It's also common to represent images with discrete intensities by linearly quantizing the range $[0, 1]$. For example, 8-bit color images take values $\mathbf{x} \in \mathcal{X}^{C \times h \times w}$ with $|\mathcal{X}| = 2^8 = 256$.

Visual data was the motivating example for the development of convnets, and it's a useful example to keep in mind as we explore convolutional architectures. For convenience, we'll assume that $h = w = d$, i.e. we assume that our images are square. Note that the convnets that we discuss trivially generalize to non-square images. These models apply equally well to one-dimensional data, simply setting $h = 1$ to drop the second spatial dimension [Oord et al., 2016]. Convnets have also been applied to higher-dimensional data, e.g 3d voxel data [Tian et al., 2019].

A Simple Convnet Architecture

A **convolutional layer** is a parameterized function class $f_\theta : \mathbb{R}^{C_{\text{in}} \times d \times d} \to \mathbb{R}^{C_{\text{out}} \times d \times d}$. If $\mathbf{x} \in \mathbb{R}^{C_{\text{in}} \times d \times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$\mathbf{z}_{c,i,j} = \text{ReLU}\left(\sum_{c'=1}^{C_{\text{in}}} \langle W_{c,c'}, \text{Pad}(\mathbf{x})_{i:i+k,j:j+k} \rangle\right), \qquad W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k \times k}. \tag{1}$$

Each index of the weight tensor W_c is referred to as a **convolution filter**. The idea is that we transform the values $\mathbf{x}_{i,j} \in \mathbb{R}^{C_{\text{in}}}$ into values $\mathbf{z}_{i,j} \in \mathbb{R}^{C_{\text{out}}}$ based on local information from the input in a neighborhood of $\mathbf{x}_{i,j}$. The hyper-parameter k is called the **receptive field** of the convolution, and controls the size of the neighborhood that we use to compute this transformation. The padding function pad : $\mathbb{R}^{C_{\text{in}} \times d \times d} \to \mathbb{R}^{C_{\text{in}} \times (d+k) \times (d+k)}$ is defined by

$$\text{Pad}(\mathbf{x})_{d,i,j} = \begin{cases} \mathbf{x}_{d,i-\lfloor k/2 \rfloor, j-\lfloor k/2 \rfloor} & \text{if } i \geq \lfloor k/2 \rfloor \text{ and } j \geq \lfloor k/2 \rfloor, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

This should be interpreted as a zero-padded version of the input $\mathbf{x}$; typically we choose an odd value for the receptive field k, in which case the pad is symmetric. If we set $C = C_{\text{in}} = C_{\text{out}}$ then the convolutional layer is endomorphic and can be stacked. A **convnet** is a composition of L convolutional layers, each with their own parameters: $f_{\theta_L} \circ \cdots \circ f_{\theta_1}(\mathbf{x}) \in \mathbb{R}^{C\times d\times d}$.

Observe that a convolution layer exhibits locality bias: $\mathbf{z}_{i,j}$ is only a function of input in a local neighborhood of $\mathbf{x}_{i,j}$. By restricting the receptive field to a bounded $k \times k$ patch, and using the same weights W to compute the relationship between inputs and outputs at each index (i, j), our parameter counts do not scale with the dimensionality of the input $d \times d$. By stacking multiple convolution layers on top of each other, we achieve a wider receptive field. It's worth taking a moment to contrast the convolutional layer with a transformer block or a recurrent network. Each architecture breaks the correspondence between parameter counts and input dimensionality, but each achieves this independence in a very different way: local connectivity in the convolutional network, versus attention in the transformer, versus state in the recurrent network.

Deep Residual Convnets

A surprising, and surprising consistent, empirical finding across many applications is that the performance of convnets increases pretty monotonically with the depth L of the network; this is the success of deep learning. But to take advantage of depth, we need a couple of modern tricks. We need a good optimizer; vanilla SGD can fail, or become exceedingly difficult to tune for deep networks, leading to the development of modern adaptive optimizers like Adam [Kingma and Ba, 2015]. People have also reported that initialization is important for optimizing deep networks [Mishkin and Matas, 2016], although in my own experiments I've found that a simple Gaussian init generally works fine. Beyond the optimizer and init, there are two modifications of the basic convolutional architecture that enable optimization of very deep networks: batch normalization [Ioffe and Szegedy, 2015] and residual connections [He et al., 2016].

A **residual convolutional block** is a parameterized function class $f_\theta : \mathbb{R}^{C\times d\times d} \to \mathbb{R}^{C\times d\times d}$. If $\mathbf{x} \in \mathbb{R}^{C\times d\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$\mathbf{u}'_{c,i,j} = \sum_{c'=1}^{C} \langle W^1_{c,c'}, \mathrm{Pad}(\mathbf{x})_{i:i+k,j:j+k}\rangle, \qquad W^1 \in \mathbb{R}^{C\times C\times k\times k}, \tag{3}$$

$$\mathbf{u} = \mathrm{ReLU}\left(\mathrm{BatchNorm}(\mathbf{u}'; \gamma_1, \beta_1)\right), \qquad \gamma_1, \beta_1 \in \mathbb{R}^C, \tag{4}$$

$$\mathbf{z}'_{c,i,j} = \sum_{c'=1}^{C} \langle W^2_{c,c'}, \mathrm{Pad}(\mathbf{u})_{i:i+k,j:j+k}\rangle, \qquad W^2 \in \mathbb{R}^{C\times C\times k\times k}. \tag{5}$$

$$\mathbf{z} = \mathrm{ReLU}\left(\mathbf{x} + \mathrm{BatchNorm}(\mathbf{z}'); \gamma_2, \beta_2\right), \qquad \gamma_2, \beta_2 \in \mathbb{R}^{\mathbb{C}}. \tag{6}$$

The BatchNorm function [Ioffe and Szegedy, 2015] is applied directly after every convolution operation, and is defined for a batch of B samples $\mathbf{z} \in \mathbb{R}^{B\times C\times d\times d}$ by

$$\mathrm{BatchNorm}(\mathbf{z}; \gamma, \beta)_{i,c} = \gamma_c \frac{(\mathbf{z}_{i,c} - \mu_{\mathbf{z},c})}{\sigma_{\mathbf{z},c}} + \beta_c, \qquad \gamma, \beta \in \mathbb{R}^C. \tag{7}$$

$$\mu_{\mathbf{z}} = \frac{1}{B}\sum_{i=1}^{B} \mathbf{z}_i, \quad \sigma_{\mathbf{z}} = \sqrt{\frac{1}{k}\sum_{i=1}^{B}(\mathbf{z}_i - \mu_{\mathbf{z}})^2}. \tag{8}$$

The residual connection, defined in Equation (6) by reintroducing the input $\mathbf{x}$ directly at the output of the block, is motivated by the principle that we should make it easy for a neural network to fit the identify function; for further discussion of this principle see Hardt and Ma [2017].

Contrast batch normalization, which centers the mean and normalizes the variance of independent samples across a minibatch, with the layer normalization used in transformers, which centers the mean and normalizes the variance of *features* across a single sample. Batch-normalization requires use of a minibatch during training (when the minibatch gets very small, maybe 4 samples or less, the behavior of BatchNorm gets weird). After training, when we evaluate the network we replace minibatch statistics $\mu_{\mathbf{z}}$ and $\sigma_{\mathbf{z}}$ with large-batch versions computed across the whole training dataset (or approximated by an exponential moving average of these statistics accumulated during training). This is a little weird, since it means that we optimize a different function than we evaluate; this behavior is comparable to the behavior of the Dropout function [Srivastava et al., 2014] and if it makes you feel better you could think of BatchNorm as a strange form of regularization that happens to also help us optimize. The reason BatchNorm helps us optimize remains unclear, five years after its introduction, despite its widespread adoption.

1x1 Convolutions

A final modern trick for improving the performance of convnets is the idea of a 1x1 convolution. These 1x1 convolutions first appeared (as far as I can tell) in Lin et al. [2013] and were rapidly integrated into models like Inception [Szegedy et al., 2015] before appearing in the form presented here in He et al. [2016]. The idea of a 1x1 convolution is that we want to make the capacity of out network C large, but if C is large then the weight matrices $W^1, W^2 \in \mathbb{R}^{C\times C\times k\times k}$ appearing in the residual convolutional block have a lot of parameters, and convolving with these matrices can be computationally intensive. The idea of a 1x1 convolution is to maintain a large network capacity C, but project down to a smaller collection of feature maps D before convolving.

A **residual block with 1x1 convolutions** is a parameterized function class $f_\theta : \mathbb{R}^{C\times d\times d} \to \mathbb{R}^{C\times d\times d}$. If $\mathbf{x} \in \mathbb{R}^{C\times d\times d}$ then $f_\theta(\mathbf{x}) = \mathbf{z}$ where

$$\mathbf{u}'_{c,i,j} = \sum_{c'=1}^{C} \langle W^1_{c,c'}, \mathbf{x}_{i:i+1,j:j+1}\rangle, \qquad W^1 \in \mathbb{R}^{C\times D\times 1\times 1}, \tag{9}$$

$$\mathbf{u} = \mathrm{ReLU}\left(\mathrm{BatchNorm}(\mathbf{u}'; \gamma_1, \beta_1)\right), \qquad \gamma_1, \beta_1 \in \mathbb{R}^D, \tag{10}$$

$$\mathbf{v}'_{c,i,j} = \sum_{c'=1}^{D} \langle W^2_{c,c'}, \mathrm{Pad}(\mathbf{u})_{i:i+k,j:j+k}\rangle, \qquad W^2 \in \mathbb{R}^{D\times D\times k\times k}, \tag{11}$$

$$\mathbf{v} = \mathrm{ReLU}\left(\mathrm{BatchNorm}(\mathbf{v}'; \gamma_1, \beta_1)\right), \qquad \gamma_2, \beta_2 \in \mathbb{R}^D, \tag{12}$$

$$\mathbf{z}'_{c,i,j} = \sum_{c'=1}^{D} \langle W^3_{c,c'}, \mathrm{Pad}(\mathbf{v})_{i:i+1,j:j+1}\rangle, \qquad W^3 \in \mathbb{R}^{D\times C\times 1\times 1}. \tag{13}$$

$$\mathbf{z} = \mathrm{ReLU}\left(\mathbf{x} + \mathrm{BatchNorm}(\mathbf{z}'); \gamma_3, \beta_3\right), \qquad \gamma_2, \beta_2 \in \mathbb{R}^C. \tag{14}$$

The 1x1 convolution W^1 applies a projection at each coordinate (i, j) of the input from C channels down to D channels (with the understanding that $D < C$). We then perform a $k \times k$ convolution with a reduced number of filter maps D, before projecting back up to C channels again with a second 1x1 convolution operation using weights W^3. Drawing a very loose analogy to transformer networks, we could compare the inner $k \times k$ convolution to the attention layer of a transformer block, and the 1x1 convolutions to the fully-connected layer. This perspective puts residual blocks with 1x1 convolutions into the same loose convection-diffusion framework described for transformers by Lu et al. [2019].

References

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *International Conference for Learning Representations*, 2017. (document)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (document)

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015. (document)

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015. (document)

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012. (document)

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. (document)

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. (document)

Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. Understanding and improving transformer from a multi-particle dynamic system point of view. *arXiv preprint arXiv:1906.02762*, 2019. (document)

Dmytro Mishkin and Jiri Matas. All you need is a good init. *International Conference for Learning Representations*, 2016. (document)

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. (document)

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014. (document)

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015. (document)

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *International Conference for Learning Representations*, 2019. (document)