PHY 835: Machine Learning in Physics
Lecture 10: Convolutional Neural Network
February 22, 2024

AI ∩ Universe

Gary Shiu
Outline for today

- Convolutional neural networks (CNNs)
- Convolutional Layer and Pooling layer
- Workflow for Deep Learning

References: Deep Learning Book, 1803.08823

Stanford CS23 (Andrej Karpathy & Fei-Fei Li): https://cs231n.github.io
Benchmark Datasets

- **MNIST** database: images of digits
- **ImageNet** database: \( \geq 1.4 \times 10^7 \) images (hand annotated), \( \geq 20,000 \) categories (e.g. screwdriver, each with \( \mathcal{O}(1000) \) examples), http://image-net.org
- Performance of algorithms measured on benchmark datasets.
- Other datasets for different problems (e.g. 3D object recognition, Language: Wordnet)
- Think of some physics examples?
Learning with Symmetries

- **Locality:** features that define a “cat” are local in the picture: whiskers, tail, paws, ...

- **Translational invariance:** Cats can be anywhere in the image.

- **Rotational invariance:** Relative position of features must be respected (e.g. whiskers and tail should appear on opposite sides)

- Our classifier should exhibit all these high-level structures.
Learning with Symmetries

- Consider classification of digits:

  - What symmetries should be built-in in ML classifiers?

Translation, scaling, small rotations, smearing, elastic deformations.
Locality and Symmetries

- **Locality & Symmetries**: basic principles underlying physical laws.
- Physics is governed by **local interactions**. Think about QFT, relativity, and statistical physics:

![Vision tasks: local features matters, e.g., whiskers, edge of a table, ...](image)

\[ H = -J \sum_{\langle ij \rangle} S_i S_j \]
Locality and Symmetries

- **Symmetries** are at the heart of physics. For example, translation invariance allows to work in momentum space \( \rightarrow \) less parameters.

- In relativity and quantum field theory, Poincare-symmetry (translations, rotations, boosts) is essential.

- Gauge symmetries are ubiquitous in QFT and gravity. Equivariant CNNs (Cohen, Welling 2016). We will come back to this...

- \( f(x) \) is equivariant if we change the input in a particular way as \( x' = g \cdot x \), the output changes in the same way: \( f(g \cdot x) = g \cdot f(x) \):

\[
\begin{array}{ccc}
X & \xrightarrow{g \cdot} & X \\
\downarrow & & \downarrow \\
Y & \xrightarrow{g \cdot} & Y
\end{array}
\]

\[ f \]

\[ f \]
Convolutional Neural Networks

• The simplest approach would be to input the images to a **fully connected NN** which given enough training data (and time) would **learn the symmetries** by example.

• However, a crucial property is ignored: **nearby pixels are strongly correlated** we should aim instead first to **identify local features** that depend on small subregions.

• For example, treating the spin configuration of the 2d Ising model as a $L \times L$ dimensional vector ($L =$ number of sites in each linear direction) throws away spatial information (e.g., domain wall)

• Convolutional Neural Networks (CNNs) are architectures that **take advantage of this additional high-level structures** that all-to-all coupled networks fail to exploit.
Convolutional Neural Networks

A CNN is a translationally invariant neural network that respects locality of the input data.

Depth: number of input channels (not depth of neural network)

Convolution

Coarse-graining (pooling)

Convolution

Coarse-graining (pooling)

Fully Connected Layer

D=3 for RGB images

Height (H) and Width (W) determined by # of pixels

pooling layers reduce H, W while preserving D

neuron activation state:
convolution with local spatial filter (e.g., 3 x 3 pixel grid)
Convolutional Neural Networks

CNNs are composed by **two** kinds of layers

Convolution of input with filters

non-linearity

example of convolutional layer

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.

In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. For instance, a square filter of size $F$ is a three-dimensional array of size $F \times F \times D_l$ (marked by the different colors in Fig. 42) – to the number of filters in that layer. All neurons corresponding to a particular filter have the same parameters (i.e. shared weights and bias).

In practice, one often inserts a BatchNorm layer before the non-linearity, cf. Section 9.4.3.

These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say $2 \times 2$ neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step of RG (Iso et al., 2018; Koch-Janusz and Ringel, 2017; Lin et al., 2017; Mehta and Schwab, 2014). This generally reduces the dimension of outputs.

For example, if the region we pool over is $2 \times 2$, then both the height and the width of the output layer will be halved.

Generally, pooling operations do not reduce the depth of the convolutional layers because pooling is performed separately at each depth. A simple example of a max-pooling operation is shown in Fig. 44. There are some studies suggesting that pooling might be unnecessary (Springenberg et al., 2014), but pooling layers remain a staple of most CNNs.

**Example of convolutional layer**

Convolution of input with filters

non-linearity

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.

In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. For instance, a square filter of size $F$ is a three-dimensional array of size $F \times F \times D_l$ (marked by the different colors in Fig. 42) – to the number of filters in that layer. All neurons corresponding to a particular filter have the same parameters (i.e. shared weights and bias).

In practice, one often inserts a BatchNorm layer before the non-linearity, cf. Section 9.4.3.

These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say $2 \times 2$ neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step of RG (Iso et al., 2018; Koch-Janusz and Ringel, 2017; Lin et al., 2017; Mehta and Schwab, 2014). This generally reduces the dimension of outputs.

For example, if the region we pool over is $2 \times 2$, then both the height and the width of the output layer will be halved.

Generally, pooling operations do not reduce the depth of the convolutional layers because pooling is performed separately at each depth. A simple example of a max-pooling operation is shown in Fig. 44. There are some studies suggesting that pooling might be unnecessary (Springenberg et al., 2014), but pooling layers remain a staple of most CNNs.

**Example of convolutional layer**

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.

In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. For instance, a square filter of size $F$ is a three-dimensional array of size $F \times F \times D_l$ (marked by the different colors in Fig. 42) – to the number of filters in that layer. All neurons corresponding to a particular filter have the same parameters (i.e. shared weights and bias).

In practice, one often inserts a BatchNorm layer before the non-linearity, cf. Section 9.4.3.

These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say $2 \times 2$ neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step of RG (Iso et al., 2018; Koch-Janusz and Ringel, 2017; Lin et al., 2017; Mehta and Schwab, 2014). This generally reduces the dimension of outputs.

For example, if the region we pool over is $2 \times 2$, then both the height and the width of the output layer will be halved.

Generally, pooling operations do not reduce the depth of the convolutional layers because pooling is performed separately at each depth. A simple example of a max-pooling operation is shown in Fig. 44. There are some studies suggesting that pooling might be unnecessary (Springenberg et al., 2014), but pooling layers remain a staple of most CNNs.

**Example of convolutional layer**

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.

In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. For instance, a square filter of size $F$ is a three-dimensional array of size $F \times F \times D_l$ (marked by the different colors in Fig. 42) – to the number of filters in that layer. All neurons corresponding to a particular filter have the same parameters (i.e. shared weights and bias).

In practice, one often inserts a BatchNorm layer before the non-linearity, cf. Section 9.4.3.

These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say $2 \times 2$ neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step of RG (Iso et al., 2018; Koch-Janusz and Ringel, 2017; Lin et al., 2017; Mehta and Schwab, 2014). This generally reduces the dimension of outputs.

For example, if the region we pool over is $2 \times 2$, then both the height and the width of the output layer will be halved.

Generally, pooling operations do not reduce the depth of the convolutional layers because pooling is performed separately at each depth. A simple example of a max-pooling operation is shown in Fig. 44. There are some studies suggesting that pooling might be unnecessary (Springenberg et al., 2014), but pooling layers remain a staple of most CNNs.

**Example of convolutional layer**

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.

In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. For instance, a square filter of size $F$ is a three-dimensional array of size $F \times F \times D_l$ (marked by the different colors in Fig. 42) – to the number of filters in that layer. All neurons corresponding to a particular filter have the same parameters (i.e. shared weights and bias).

In practice, one often inserts a BatchNorm layer before the non-linearity, cf. Section 9.4.3.

These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say $2 \times 2$ neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step of RG (Iso et al., 2018; Koch-Janusz and Ringel, 2017; Lin et al., 2017; Mehta and Schwab, 2014). This generally reduces the dimension of outputs.

For example, if the region we pool over is $2 \times 2$, then both the height and the width of the output layer will be halved.

Generally, pooling operations do not reduce the depth of the convolutional layers because pooling is performed separately at each depth. A simple example of a max-pooling operation is shown in Fig. 44. There are some studies suggesting that pooling might be unnecessary (Springenberg et al., 2014), but pooling layers remain a staple of most CNNs.

**Example of convolutional layer**

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.

In general, we will be concerned with local spatial filters (often called a receptive field in analogy with neuroscience) that take as inputs a small spatial patch of the previous layer at all depths. For instance, a square filter of size $F$ is a three-dimensional array of size $F \times F \times D_l$ (marked by the different colors in Fig. 42) – to the number of filters in that layer. All neurons corresponding to a particular filter have the same parameters (i.e. shared weights and bias).

In practice, one often inserts a BatchNorm layer before the non-linearity, cf. Section 9.4.3.

These convolutional layers are interspersed with pooling layers that coarse-grain spatial information by performing a subsampling at each depth. One common pooling operation is the max pool. In a max pool, the spatial dimensions are coarse-grained by replacing a small region (say $2 \times 2$ neurons) by a single neuron whose output is the maximum value of the output in the region. In physics, this pooling step is very similar to the decimation step of RG (Iso et al., 2018; Koch-Janusz and Ringel, 2017; Lin et al., 2017; Mehta and Schwab, 2014). This generally reduces the dimension of outputs.

For example, if the region we pool over is $2 \times 2$, then both the height and the width of the output layer will be halved.

Generally, pooling operations do not reduce the depth of the convolutional layers because pooling is performed separately at each depth. A simple example of a max-pooling operation is shown in Fig. 44. There are some studies suggesting that pooling might be unnecessary (Springenberg et al., 2014), but pooling layers remain a staple of most CNNs.

**Example of convolutional layer**

Convolution of input with ReLU non-linearity. Convolutional layer for a spatial filter of size $F$ for a one-dimensional input of width $W$ with stride $S$ and padding $P$ followed by a ReLU non-linearity.
F = receptive field size of the Conv Layer neurons
S = stride
P = amount of zero padding on the border

Number of neurons (outputs) in the layer:

\[(W - F + 2P)/S + 1\]
CHAPTER 9. CONVOLUTIONAL NETWORKS

Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.
The Convolution Operation

For example, estimate the position of a spaceship from several measurements:

\[ s(t) = \int x(a)w(t - a)\,da \]

This operation known as \textit{convolution} is denoted by an asterisk:

\[ s(t) = (x \ast w)(t) \]

In a data set, “time” is discretized:

\[ s(t) = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \]
Convolutional Neural Networks

CNNs are composed by **two** kinds of layers.

1. **Convolution layer** of input with filters.
2. **Pooling** layer that coarse-grains the input while maintaining locality and spatial structure.

   - **MaxPool**: the spatial dimensions are coarse-grained by replacing a small region by a single neuron whose output is the maximum value of the output in the region.
   - **Average Pooling**: one averages over the output in the region.

   ~ decimation in RG

   reduces the dim. of outputs (depth is kept fixed)

In this example, by pooling over 2x2 blocks, H and W are reduced by half.

max pooling

average pooling
Convolutional Neural Networks

CNNs are composed by two kinds of layers:

- **Convolution** layer that computes the convolution of the input with filters
- **Pooling** layer that coarse-grains the input while maintaining locality and spatial structure

These layers are followed by an all-to-all connected layer and a high-level classifier, so that one can train CNNs using the standard backpropagation algorithm:
Convolutional Neural Networks

- Significantly reduce the number of parameters: determined by the number and size of the filters. Further reduced by pooling.

- Only problems characterized by spatial locality are amenable to CNNs, e.g., Ising model and MNIST but not SUSY datasets.

![Image of ordered, critical, and disordered phases of the Ising model](image_url)

- See Notebook 14: Pytorch CNN (Ising); MNIST example (later).
  
  ![Notebook 14: Pytorch CNN (Ising); MNIST example](https://physics.bu.edu/~pankajm/MLnotebooks.html)

- Can you think of the types of datasets in particle physics and cosmology that are amendable to CNNs?
Organizing a workflow for Deep Learning. Schematic illustrating a deep learning workflow inspired by navigating the bias–variance tradeoff (figure based on Andrew Ng’s talk at the 2016 Deep Learning School available at https://www.youtube.com/watch?v=F1ka6a13S9I).

We have assumed that there is no mismatch between the distributions the training and test sets are drawn from.

In the second part of this section, we shift gears and ask the question, why have neural networks been so successful? We provide three different high-level explanations that reflect current dogmas. Finally, we end the section by discussing the limitations of supervised learning methods and current neural network architectures.

11.1. Organizing deep learning workflows using the bias–variance tradeoff

Imagine that you are given some data and asked to design a neural network for learning how to perform a supervised learning task. What are the best practices for organizing a systematic workflow that allows us to efficiently do this? Here, we present a simple deep learning workflow inspired by thinking about the bias–variance tradeoff (see Fig. 46). This section draws heavily on Andrew Ng’s tutorial at the Deep Learning School (available online at https://www.youtube.com/watch?v=F1ka6a13S9I) which readers are strongly encouraged to watch.

The first thing we would like to do is divide the data into three parts. A training set, a validation or dev (development) set, and a test set. The test set is the data on which we want to make predictions. The dev set is a subset of the training data we use to check how well we are doing out-of-sample, after training the model on the training dataset. We use the validation error as a proxy for the test error in order to make tweaks to our model. It is crucial that we do not use any of the test data to train the algorithm. This is a cardinal sin in ML. We thus suggest the following workflow:

- **Estimate optimal error rate (Bayes rate).**
  - The first thing one should establish is the difficulty of the task and the best performance one can expect to achieve. No algorithm can do better than the “signal” in the dataset. For example, it is likely much easier to classify objects in high-resolution images than in very blurry, low-resolution images. Thus, one needs to establish a proxy or baseline for the optimal performance that can be expected from any algorithm. In the context of Bayesian statistics, this is often called the Bayes rate. Since we do not know this a priori, we must get an estimate of this.
  - For many tasks such as speech or object recognition, we can approximate this by the performance of humans on the task. For a more specialized task, we would like to ask how well experts, trained at the task, perform. This expert performance then serves as a proxy for our Bayes rate.

- **Minimize underfitting (bias) on training dataset.**
  - After we have established the Bayes rate, we want to make sure that we are using a sufficiently complex model to avoid underfitting on the training dataset. In practice, this means comparing the training error rate to the Bayes rate. Since the training error does not care about generalization (variance), our model should approach the Bayes rate on the training set. If it does not, the bias of the DNN model is too large and one should try training the model longer and/or using a larger model. Finally, if none of these techniques work, it is likely that the model architecture is not well suited to the dataset, and one should modify the neural architecture in some way to better reflect the underlying structure of the data (symmetries, locality, etc.).

- **Make sure you are not overfitting.**
  - Next, we run our algorithm on the validation or dev set. If the error is similar to the training error rate and Bayes rate, we are done. If it is not, then we are overfitting the training data. Possible solutions include, regularization and, importantly, collecting more data. Finally, if none of these work, one likely has to change the DNN architecture.

If the validation and test sets are drawn from the same distributions, then good performance on the validation set should lead to similarly good performance on the test set. (Of course performance will typically be slightly worse on the test set because the hyperparameters were fit to the validation set.) However, sometimes the training data and test data.