Introduction to Deep Learning – Week 3

CNNs operate differently from traditional NNs in which every neuron in a layer is connected to every neuron in the previous layer. The MLP type of connection is general purpose and makes no assumptions about the features in the input data, meaning that it tends to be expensive in terms of memory and computation. CNNs take advantage of the properties of spatial data and assemble more complex patterns using smaller and simpler patterns. **Convolutional Neural Networks** contain multiple copies of the same neuron, and all the neurons share the same weights, bias and activation function. As a result, the neuron can be 'reused' in different parts of the input.

I’ll describe each of the successive parts of a CNN before summarising.

1. Input

We start with an input of Images, each of which is essentially an array of pixel values or intensities.

1. Kernels

In spatial data it makes sense to apply a small window around a set of pixels in order to look for image features. Each neuron in a CNN is only connected to a few nearby local neurons in the previous layer. By using the same weights we are assuming that the feature is equally likely to occur at every input position. To

*Diagram of convoluting a matrix and a kernel.*

The stride is the distance you shift the kernel between each convolution – a bigger stride leading to a small output matrix.

To account for the fact that at the edges you would lose values you add padding to an input image.

Common kernels: edges: [[1,0,-1],[2,0,-2],[1,0,-1]] and blur: [[1/9, etc.]]

Receptive field gets larger for each layer.

The number of feature maps, i.e. depth of output, is determined by the number of fitlers used. The product of pixels and maps are kept roughly constant.

The number of parameters to train a convolutional layer becomes (kernel\_width \* kernel\_height \* kernel\_depth + 1) \* output\_depth

 As a result, the network learns filters that activate when it detects some specific type of [feature](https://en.wikipedia.org/wiki/Feature_(machine_learning)) at some spatial position in the input.

1. Dropout

Dropout is a means of regularising a neural network. During the training process, specific neurons may become influenced solely by particular neurons in the previous layer, leading to overfitting. Dropout prevents this by randomly cutting the connections between neurons in successive layers during training. All it does is set a fraction of the inputs to 0.

1. Pooling

The objective of max pooling is to down-sample the dimensionality of an input representation (e.g. an image), by using assumptions about the features in sub-regions. It is done to both help overfitting and reduce the computational cost by reducing the number of parameters to learn. It is often achieved with a max filter, which just replaces a number of values with the single largest value in a specific kernel shape.

*Diagram of max pooling.*

They are non-overlapping. Pooling is also used in CNNs to make the detection of certain features invariant to scale and orientation changes. They achieve this as they generalise over lower level, complex information.

We do pooling depth wise – i.e. we don’t change the number of output channels just the dimensions.

1. Fully Connected layer(s) for classification
2. Back Propagation.

Number of weights = number of terms in mask. A weight term is updated using the derivative of the loss function in regards to shared weights (we sum the gradients of the same shared weights)

Weight initialisation: if the weights are initialised at the same value, because the chain rule means that in an MLP backpropagation will mean that they will update in the same way – the symmetry problem. Variance grows with consecutive layers – this hurts convergence in NNs. Ill conditioning is when gradients of different outputs become of different scales – gradient descent methods slow down dramatically. Xavier initialisation multiplies weights by by sqrt 2 / sqrt(n\_in + n\_out) of hidden layer. Batch normalisation: (i) normalise neuron output before activation, (ii) during training we use an exponential moving average for mu and sigma. Dropout: keep neurons active with probability p. Data augmentation – to get around small datasets, colour shifts, scaling – talk last week where generalising to real images did not work.

Normalising the input data to have a mean of zero means that most of the hidden neurons start with their outputs in the linear region of the activation functions, where training progresses fast. It also improves the convergence properties of backpropagation.

Backpropagation of max pooling is not differentiable – instead non-maximum patch neurons have a gradient of 0 as changing their value does not change the output, whereas the max patch neuron has a gradient of 1. Neurons of deep convolutional layers learn complex representations that. Can be used as features for classification with MLP.

*NNs like outputs have zero mean and have standard variance. Also exponential is expensive with lots of neurons. The hyperbolic tangent is zero-centred but suffers from the same problems as sigmoid. In practice, ReLu provides the fastest convergence. However, this will also suffer from dying neurons – negative values that get given z value of zero and therefore never update. This is dealt with by leaky ReLu, which has a small gradient for when f(x) = max (ax, x).*

Summary:

* **Feature engineering** is the process of extracting useful patterns from the input data in order to help the predictive model understand the true nature of the problem. CNNs are extremely good at not only finding features but also combining them together to increase the complexity of the patterns. The final layers use these generated features for the task at hand. Good feature learning significantly increases the accuracy of the applied machine learning algorithms in a way that would be too expensive for the machine learning itself.
* CNNs make the explicit assumption that the entries are images, which allows one to encode certain properties in the architecture to recognise specific elements in the images.
* CNNs can learn spatial hierarchies of patterns by preserving spatial relationships, i.e. a second layer can learn patterns comprised of the basic elements of the first layer.
* Each neuron of the first hidden layer is connected to a small region (5x5) of the input image.
* The particular thing about CNNs is that we use the same filter for all the neurons in a hidden layer. This drastically reduces the number of parameters in a network.

Transfer Learning

There are 101 different types of transfer learning but the basic principle is that you attempt to utilise similarities in task domains to reuse a model trained on one task in another.

One approach is to reuse the first layers and only change the later convolutional layers and fully connected layers. Instead of randomly initialising parameters you instead reuse those learned from the original model.

Again you propagate the gradients but with a smaller learning rate so don’t lose that original training.

Other similar tasks:

Semantic segmentation: an input image to a class label for each pixel.

No pooling layers are used because this downsamples, i.e. pixelates the image. Instead keep width and height, final layer has number output channels = number clusters for segmentation, i.e. each depth slice will be duck, water, grass, etc. Naïve: softmax per pixels. This is too expensive. By adding pooling layer we increase depth and decrease width and height, so we do unpooling: upsampling so we can create transformation back to original size. Methods: fill pixels with same values, max-pooling: remember which element was maximum during pooling and fill that position during unpooling. These are not data driven as they don’t use image. Rather could use convolution to do unpooling: e.g. 2x2 to 4x4 use 3x3 kernel with stride 2 (transposed convolution)

Image classification and localisation: a bounding box where the object is.

Object localisation and classification: bounding box with x,y,w,h values. Use regression for these parameters. CNN trained with cross-entropy. We can reuse convolutional layers to train box wit MSE instead of cross-entropy. To train with different losses we use the sum of the losses for backpropagation.