University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 10: October 2

Lecturer: Alexander Schwing Scribe: Daguan Chen

1 Outline

Lecture 10 focuses on deep nets and deep learning. It explores the motivation for deep nets over other alternatives, such as logistic regression. We examine the different building blocks that combine to form deep nets: fully connected layers, convolutions, ReLUs, pooling, dropout layers, and softmax layers. We also briefly discuss how to train and optimize deep nets, although that particular topic will be covered in greater detail subsequently, in Lecture 11.

2 Motivation

At this point, we have a current framework for how to train classification systems, using the cost function:

$$\min_{\mathbf{w}} \frac{C}{2}||\mathbf{w}||_2^2 + \sum_{i \in D} \epsilon ln \sum_{\hat{y}} exp \frac{L(y^{(i)}, \hat{y}) + \mathbf{w}^T \psi(x^{(i)}, \hat{y})}{\epsilon} - \mathbf{w}^T \psi(x^{(i)}, y^{(i)})$$

Let us take a minute to understand what these things mean. We assume we are provided with training dataset $D = \{(x^{(i)}, y^{(i)})\}$, where x is the data, and y is the classification. These pairs are index by "i". Here,C is a constant that we can choose for regularization purposes. $\mathbf{w}$ are weight parameters that we seek to adjust in order to optimize our classifier - that is, to minimize our formulation of the cost/error function. ϵ is another constant we pick. $\hat{y}$ is the predicted label, L() is the loss function. ψ is a function we operate upon x and y to get to the feature space; we have flexibility in choosing this function; it may return a matrix. However, the parameters of the function itself are fixed, and depend on $\mathbf{w}$ to "modulate" it.

Note that the first term, $\frac{C}{2}||\mathbf{w}||_2^2$, is a regularization term, and its importance is scaled via choosing C.

What is a possible issue/limitation with this framework? We see that we have been restricted to linearity in the feature space $\psi(x, y)$. Even if we use kernels to address this problem, $\mathbf{w}$ is still a matrix, and therefore we are learning a model linear in the paramaters of $\mathbf{w}$. To break such a linearity constraint, we replace $\mathbf{w}^T \psi(x^{(i)}, \hat{y})$ with a more general function, F($\mathbf{w}$, x, y); note that the formulation $\mathbf{w}^T \psi(x^{(i)}, \hat{y})$ constitutes merely a subspace of such more general functions.

We then are left with a newer formulation:

$$\min_{\mathbf{w}} \frac{C}{2}||\mathbf{w}||_2^2 + \sum_{i \in D} \epsilon ln \sum_{\hat{y}} exp \frac{L(y^{(i)}, \hat{y}) + F(\mathbf{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\mathbf{w}, x^{(i)}, y^{(i)})$$

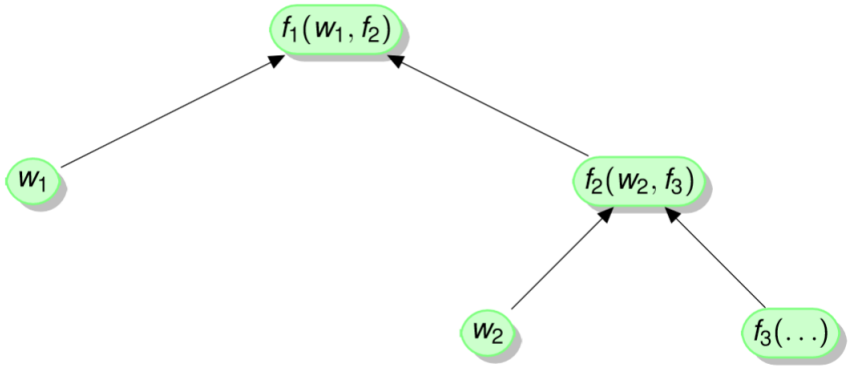
Indeed, by choosing properly the parameters and sub-functions within this function, we are able to formualte logistic regression, binary SVM, multiclass regression, and multiclass SVM. We will focus on a new, alternate formulation enabled by such a generalization: deep learning.

3 Introduction to Deep Learning

Deep learning is a way for us to obtain a suitable function F(**w**, x, y) and for us to optimize **w**. For such task, we choose F() to be any differentiable composite function

$$F(\mathbf{w}, x, y) = f_1(\mathbf{w}_1, y, f_2(\mathbf{w}_2, f_3(...f_n(\mathbf{w}_n, x)...)))$$

Note that we have kept y not too deeply nested; this allows for faster computation of F() for different values of y.
More generally, functions can be represented by an acyclic, computational graph. For example, the function $F(\mathbf{w}, x, y) = f_1(\mathbf{w}_1, f_2(\mathbf{w}_2, f_3(...)))$ can be represented by the following graph:



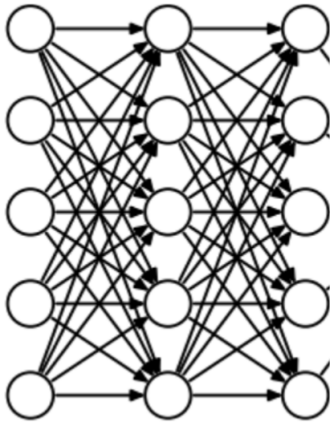
Here, the nodes are weights, data, and functions. Each node's output serves as input to the function it points to. In the next section we will explore the different functions that are frequently used in deep nets.

4 Functions (Layers) of Deep nets

We will go over the following functions/layers/blocks that are used in deep nets:

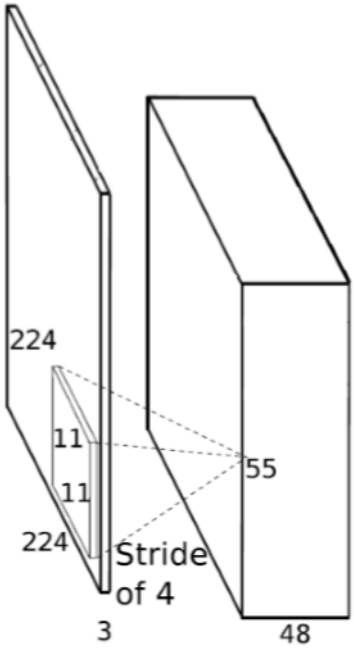
- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): max{0,x}
- Maximum/Average pooling
- Soft-max layer
- Dropout

4.1 Fully conected layers



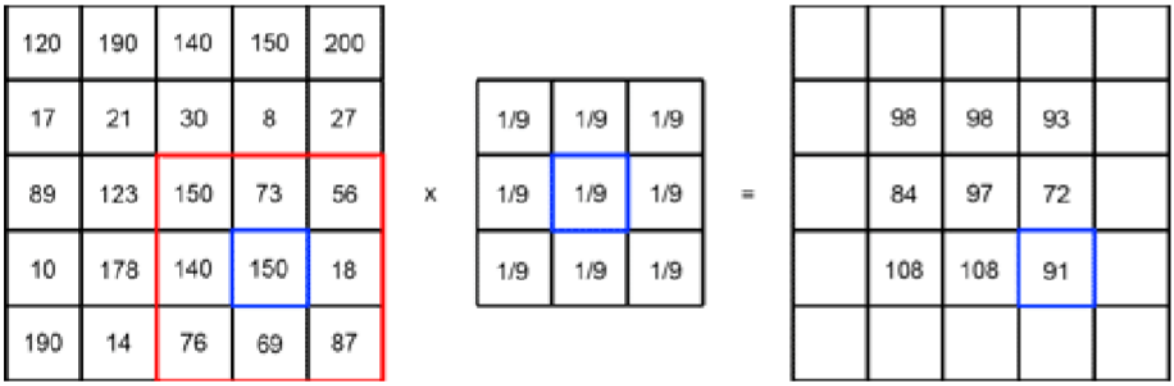
A fully connected layer has all nodes from one level connected to all nodes on the next level, as the diagram shows. Such a layer can be described using weights (of each edge) and a series of biases injected into the recipient nodes as offsets. Conveniently, such a layer is linear, and can be described as $\mathbf{Wx} + \mathbf{b}$, where $\mathbf{W}$ is a matrix of edge weights, $\mathbf{x}$ is the vector containing outputs from the previoua layer of nodes, and $\mathbf{b}$ are the biases. The output of the equation is the "raw" value injected into each node on the next level. These raw values are then passed to the functions of those particular nodes. Unfortunately, for layers with many nodes, such as images (where each pixel can be a node), fully connected layers are too dense to be computationally feasible.

4.2 Convolutions



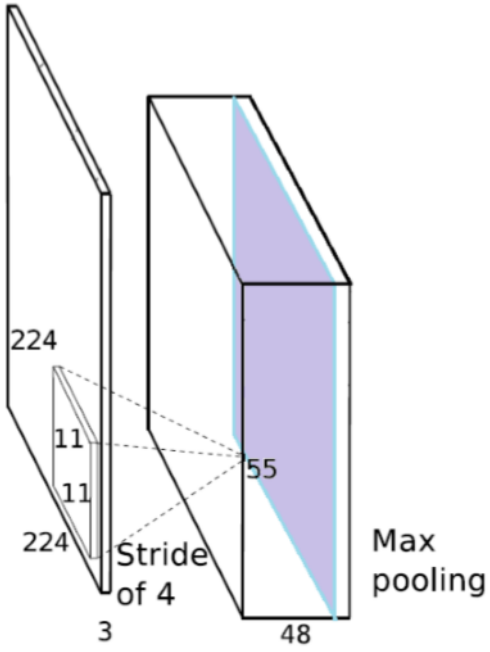
A convolution layer is one way to reduce the number of parameters by sharing weights. In the case of a image-processing deep net, the convolution layer can be understood as performing a two-dimensional convolution on the image using a convolutoin "template" filter. These filters, along

with injected biases, become the trainable parameters. The figure below gives a brief demonstration of what 2D convolutions look like with a square matrix (in this case, representing an image with different pixel values). The first grid can be understood as the original image; the smaller 3x3 grid is the filter, and they comvolve to produce the output on the right. We observe that to get the output of the convolution, at each pixel we perform a weighed summation of the pixels in the original images, with weights dictated by the values in the filter. Thus, the idea of a convolution is similar to that of taking dot products between the filter and small patches of the original image.



4.3 Maximum/average pooling

The output of this layer is the maximum or average value over a spatial region of the original values. Such a layer has no trainable parameters. However, care ought to go into designing the size and shape of the spatial region over which pooling is carried out.



4.4 Soft-max layer

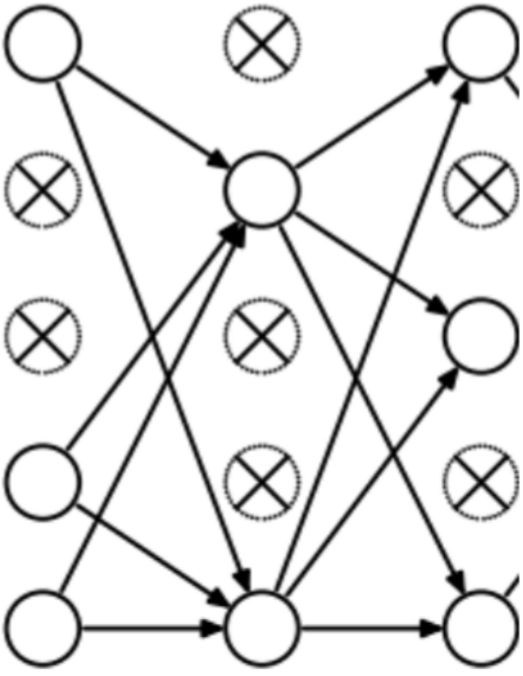
The soft-max layer can be described by the function

$$softmax([x_1, x_2, ...x_i]) = \frac{1}{\sum_{j=1}^{i} exp(x_j)}[exp(x_1), exp(x_2), ...exp(x_i)]$$

Note that the soft-max layer has no traininable parameters.

The soft-max layer is often used at the output to turn classification scores into probabilities. It has the feature that the sum of its output values total to 1, as probability distributions should. Furthermore, when the scores are formulated in a way that corresponds with the log-likelihood probability of the classifications being true, the soft-max will take the log-probabilities and transform them into actual probability values [2]. This, and other useful features, make the soft-max layer oftentimes more preferrable to alternate ways of making the output values sum to 1, such as dividing by the sum.

4.5 Dropout

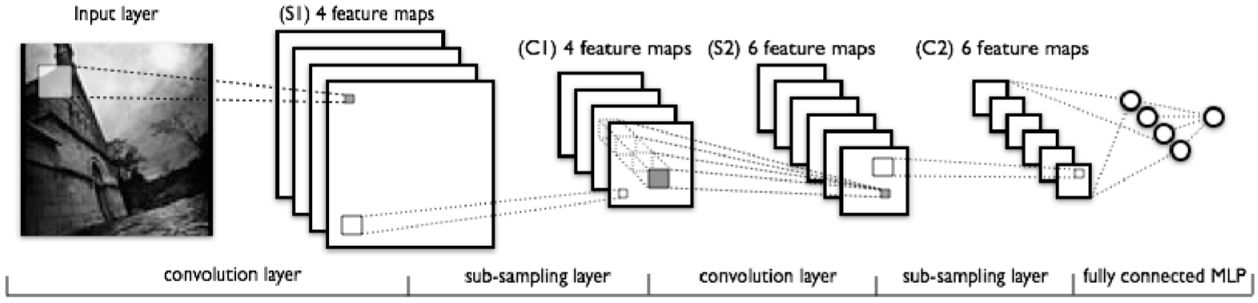


The dropout layer, as its name suggests, randomly drops out certain activations by setting them to zero. Such layers help to prevent the deep net from overfitting the training data.

5 Example Function Architectures

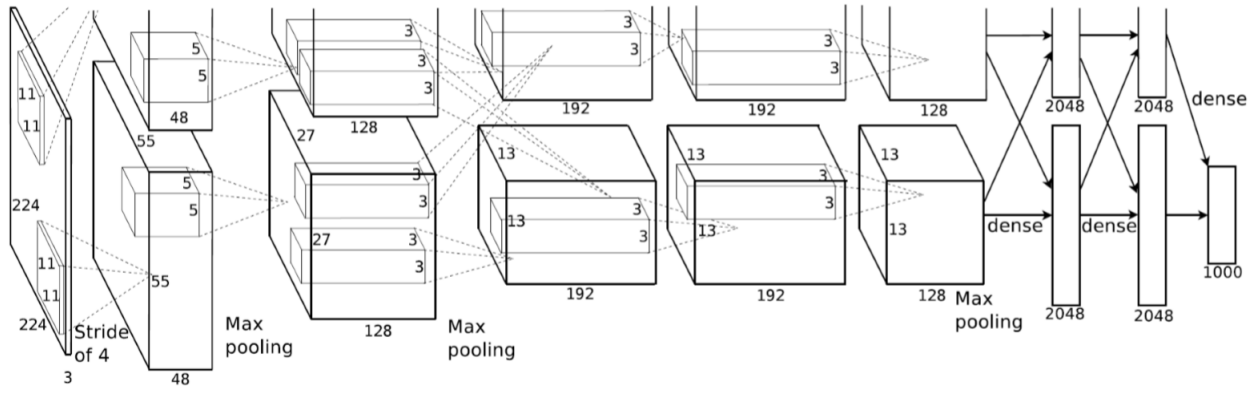
5.1 LeNet

LeNet is an example of a deep net that operates on images. It uses convolution layers, sub-sampling layers, and fully-connected MLPs. Like many image-processing deep nets, we see a trend as we go through the layers: the spatial resolution of the image-object decreases, while the number of channels increases.



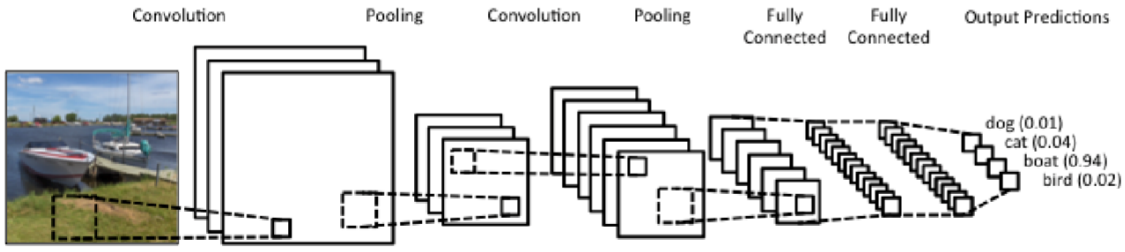
5.2 AlexNet

AlexNet is another computer-vision related deep net, albeit a bigger one. It works well and is well-regarded within the computer vision community [1].
Again, we notice the trend of progressively decreasing spatial resolution and increasing number of channels. We note that the output is 1000-dimensional. This is because the original image-classification task AlexNet performed had 1000 potential categories.



5.3 Another deep net

Another deep net for computer vision. This one classifies images into one of four categories: dog, cat, boat, and bird. Note that each category is given a score. Here, they sum up to roughly 1, with the "boat" category correctly having the highest score at 0.94. Although it is not explicitly shown, such a summation to 1 is likely caused by an implicit soft-max layer.



Although all three examples have to do with computer vision, it is entirely possible for deep nets to operate on data that is neither an image nor 2D. In many situations, a deep net may not contain convolutional layers.

6 Preview: Training Deep Nets

For deep nets, we can formulate the following cost function minimization:

$$\min_{\mathbf{w}} \frac{C}{2}||\mathbf{w}||_2^2 + \sum_{i \in D} ln \sum_{\hat{y}} expF(\mathbf{w}, x^{(i)}, \hat{y}) - F(\mathbf{w}, x^{(i)}, y^{(i)})$$

Such training is often also referred to as maximizing the regularized cross entropy; we can write:

$$\max_{\mathbf{w}} \frac{C}{2}||\mathbf{w}||_2^2 + \sum_{i \in D} \sum_{\hat{y}} p_{GT}^{(i)}(\hat{y}) lnp(\hat{y}|x^{(i)})$$

where $p_{GT}^{(i)}(\hat{y}) = \delta((\hat{y} = y^{(i)}$ and $p(\hat{y}|x^{(i)}) \propto expF(\mathbf{w}, x, \hat{y})$.
We can write the maximization problem as a minimization problem of itself multiplied by -1:

$$\min_{\mathbf{w}} \frac{C}{2}||\mathbf{w}||_2^2 \sum_{i \in D} \sum_{\hat{y}} p_{GT}^{(i)}(\hat{y}) lnp(\hat{y}|x^{(i)})$$

Here, $\frac{C}{2}||\mathbf{w}||_2^2$ is called the weight decay, C is called the regularization constant, and $\sum_{i \in D} \sum_{\hat{y}} p_{GT}^{(i)}(\hat{y}) lnp(\hat{y}|x^{(i)})$ is the cross entropy loss, which in Pytorch is represented by torch.nn.CrossEntropyLoss(gt, F). This particular formulation adheres to the equation for information entropy.

Aside from CrossEntropyLoss, there are many other loss functions to pick from:

CrossEntropyLoss:

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j]))) = -x[class] + log(\sum_j exp(x[j]))
```

NLLLoss:

```
  loss(x, class) = -x[class]
```

MSELoss:

```
  loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

BCELoss:

```
  loss(o,t)=-1/n \sum_i i(t[i]*log(o[i])+(1-t[i])*log(1-o[i]))
```

BCEWithLogitsLoss:

```
  loss(o,t)=-1/n\sum_i(t[i]*log(sigmoid(o[i])) +(1-t[i])*log(1-sigmoid(o[i])))
```

L1Loss

KLDivLoss

Let us take a closer look at NLLLoss: it is formulated as

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

so that it can work well with LogSoftMax:

$$f_i(x) = log \frac{exp(x_i)}{\sum_j exp(x_j)}$$

This gives our computation numerical robustness through the log-sum-exp trick, which states

$$log \sum_j exp(x_j) = c + log \sum_j exp(x_j - c)$$

Do not try to go without such robustness, since things will fail.

Finally, here is an example of a deep net implemented in Pytorch, PyTorchECE544.py:

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(1, 6, 5)
    self.conv2 = nn.Conv2d(6, 16, 5)
    self.fc1 = nn.Linear(16 * 5 * 5, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

  def forward(self, x):
    x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
    x = F.max_pool2d(F.relu(self.conv2(x)), 2)
    x = x.view(-1, self.num_flat_features(x))
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
return x
```

References

[1] G. E. H. Alex Krizhevsky, Kya Sutskever. Image net classication with deep convolutional neural networks. 2012.

[2] A. Karpathy. Softmax ciassifier, cs231n convolutional neural networks for visual recognition.