CS 446/ECE 449: Machine Learning

A. G. Schwing

University of Illinois at Urbana-Champaign, 2020

Scribe & Exercises

L8: Deep Neural Networks

Goals of this lecture

- Getting to know deep nets
- Understanding forward and backward pass
- Learning about deep net components

Reading material

- I. Goodfellow et al.; Deep Learning; Chapters 6-9

Our current framework:

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

What is a possible issue/limitation?

Linearity in the feature space $\psi(x,y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to fix this?

Replace $\boldsymbol{w}^T\psi(x,y)$ with a general function $F(\boldsymbol{w},x,y)$

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

General framework:

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

How to get to

- **Logistic regression**
- **Binary SVM**
- **Multiclass regression**
- **Multiclass SVM**
- **Deep Learning**

Deep Learning:

What function $F(\boldsymbol{w}, x, y) \in \mathbb{R}$ to choose?

- Choose any differentiable composite function

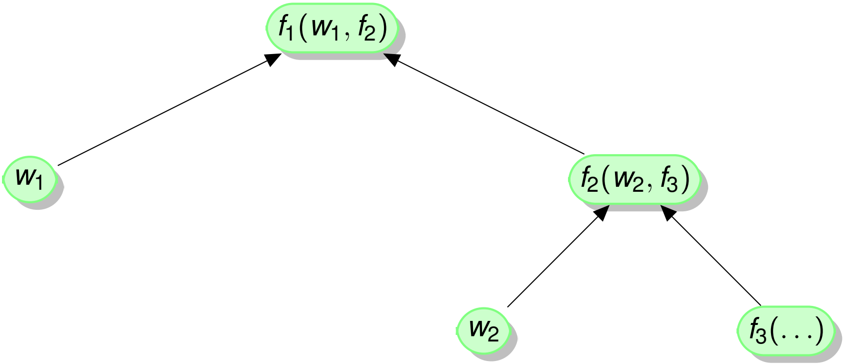
$$F(\boldsymbol{w}, x, y) = f_1(\boldsymbol{w}_1, y, f_2(\boldsymbol{w}_2, f_3(\ldots f_n(\boldsymbol{w}_n, x)\ldots)))$$

- More generally: functions can be represented by an acyclic graph (computation graph)

Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, f_2(w_2, f_3(\ldots)))$$

Nodes are weights, data, and functions:

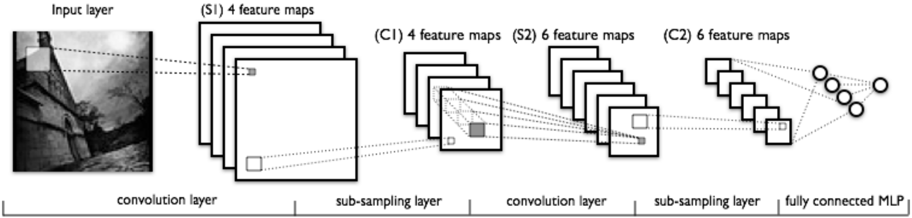


Internal representation used by deep net packages.

What are the individual functions/layers f_1, f_2 etc.?

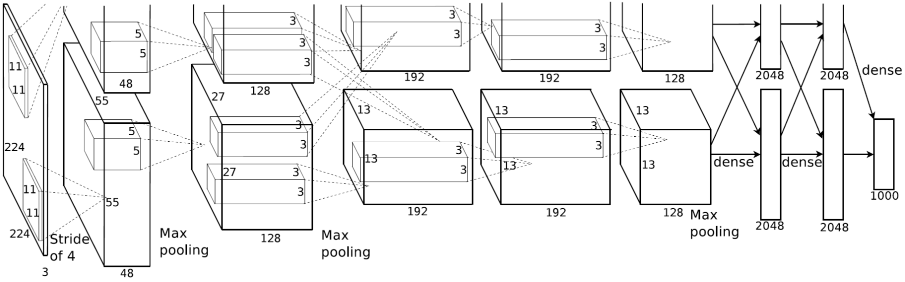
- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling
- Soft-max layer
- Dropout

Example function architecture: LeNet



Decreasing spatial resolution and the increasing number of channels

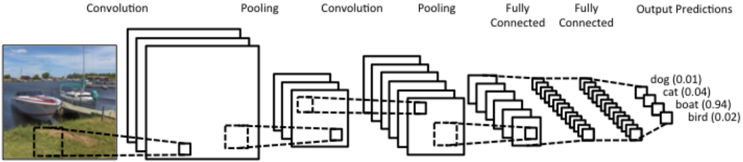
Example function architecture: AlexNet



Decreasing spatial resolution and the increasing number of channels

Why is the output 1000-dimensional?

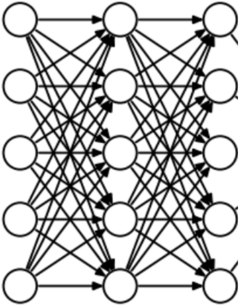
Another deep net:



Those nets are structurally simple in that a layer's output is used as input for the next layer. This is not required.

Fully connected layer:

$$\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$



Trainable parameters $\boldsymbol{w}$:

- Matrix
- Bias

What's an issue with fully connected layers?

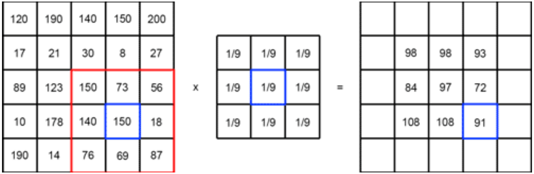
Issue with fully connected layers:

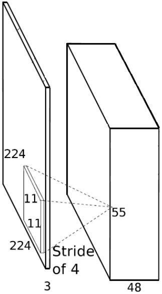
- Suppose the input is an image of size 256×256
- Let the output of this layer have identical size
- How many weights are necessary?

$$2^{32} = 4,294,967,296$$

- How to share weights?

Convolutions:





Trainable parameters ***w***:

- Filters (width, height, depth, number)
- Bias

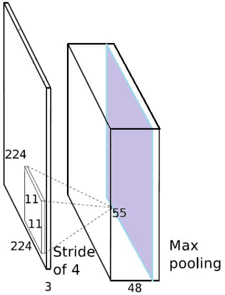
width+2*padding+..+1   55

Maximum-/Average pooling

Maximum or average over a spatial region

Trainable parameters $\boldsymbol{w}$:

- None



Soft-max layer:

$$x \quad \longrightarrow \quad \frac{\exp x_i}{\sum_j \exp x_j}$$

Trainable parameters $\boldsymbol{w}$:

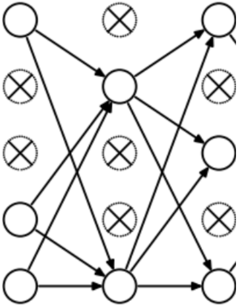
- None

Dropout layer:

Randomly set activations to zero

Trainable parameters $\boldsymbol{w}$:

- None



Deep net training:

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

ground truth

Often also referred to as maximizing the regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \quad \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is C? Weight decay (aka regularization constant)

$$\min_{\boldsymbol{w}} \underbrace{\frac{C}{2}\|\boldsymbol{w}\|_2^2}_{\text{weight decay}} \underbrace{-\sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)})}_{\text{torch.nn.CrossEntropyLoss(gt, } F\text{)}}$$

Deep learning choices:

- Design a composite function $F(\boldsymbol{w}, x, y)$
- Use an appropriate loss function

Know what you are doing, i.e., know all the dimensions.

Many more loss functions:

- 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]
```

log–likelihood, x is log softmax probablity

- 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

Why this form for the NLLLoss?

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

Intended to be used in combination with 'LogSoftmax':

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

Why? Numerical robustness ('log-sum-exp trick')

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

Don't try without, it **will** fail!

Example (PyTorch.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
```

What are the input dimensions?

Quiz:

- What are deep nets?
- How do deep nets relate do SVMs and logistic regression
- What components of deep nets do you know?
- What algorithms are used to train deep nets?

Important topics of this lecture

- Deep nets
- Relationship to SVMs and logistic regression
- Components of deep nets

Up next:

- Backpropagation