CS 446/ECE 449: Machine Learning

A. G. Schwing

University of Illinois at Urbana-Champaign, 2020

Scribe & Exercises

L9: Deep Neural Networks

Goals of this lecture

- Understanding forward and backward pass
- Learning about backpropagation

Reading material

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

Recap: Our earlier framework:

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

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 $F(\boldsymbol{w}, x, y) \in \mathbb{R}$

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

General framework:

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

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? ($y \in \{1, \ldots, K\}$)

- Choose any differentiable composite function

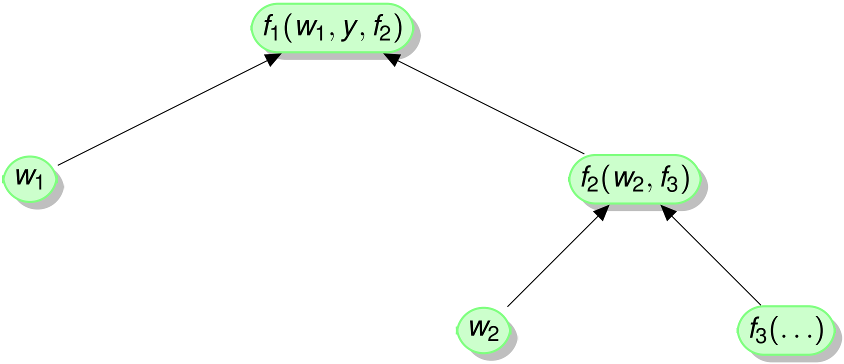
$$F(\boldsymbol{w}, x, y) = f_1(\boldsymbol{w}_1, y, f_2(\boldsymbol{w}_2, f_3(\ldots f_n(\boldsymbol{w}_n, x)\ldots))) \in \mathbb{R}$$

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

Example:

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

Nodes are weights, data, and functions:

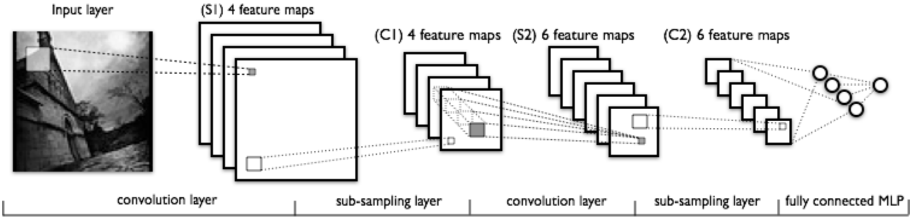


Internal representation used by deep net packages.

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

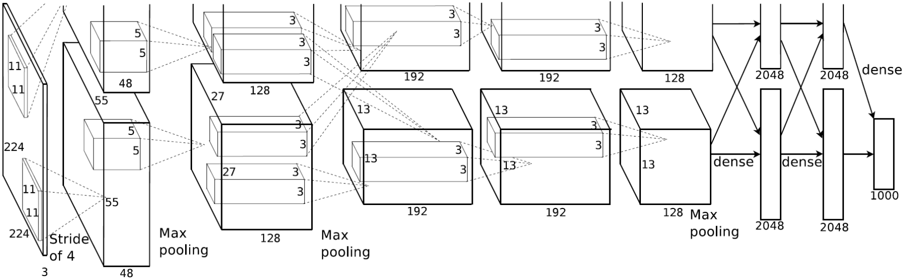
- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling
- Soft-max layer
- Dropout

Example function architecture: LeNet



Decreasing spatial resolution and the increasing number of channels

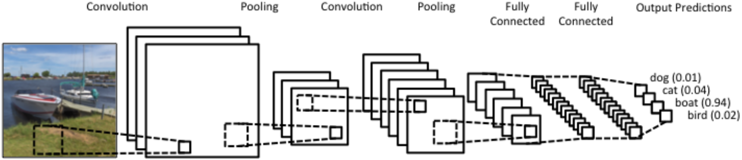
Example function architecture: AlexNet



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.

Deep net training:

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

Often also referred to as maximizing the regularized cross entropy:

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

What is C? Weight decay (aka regularization constant)

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

Program:

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

How to optimize this?

Stochastic gradient descent with momentum: What was this again?

heavy ball rolling down

Gradient of

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

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(\underbrace{p(\hat{y}|x^{(i)})}_{\text{softmax}} - \underbrace{\delta(\hat{y} = y^{(i)})}_{\text{ground truth}} \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes logits F as input
- $\frac{\partial F(\boldsymbol{w},x,\hat{y})}{\partial \boldsymbol{w}}$ via backpropagation

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 = f_3(w_3, x) \\ x_1 = f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

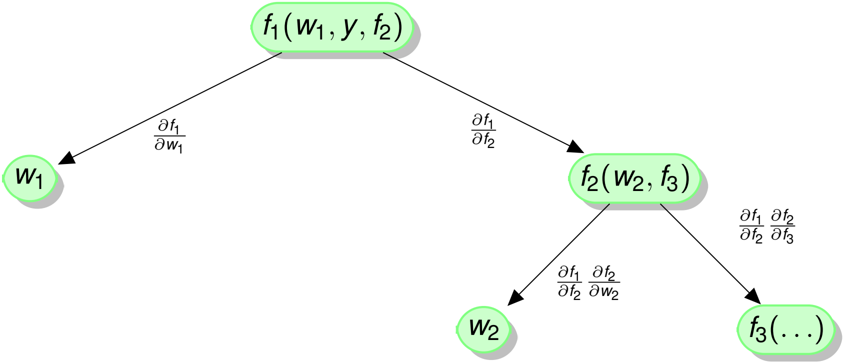
What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial w_2}$$

Generally: To avoid repeated computation, backpropagation on an acyclic graph. Nodes in this graph are weights, data, and functions.

Composite function represented as acyclic graph

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



Repeated use of chain rule for efficient computation of all gradients

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result
- Learning:
  - Store intermediate results for fully connected layer, convolution
  - Some functions can be combined to reduce intermediate storage, e.g., X + ReLU, X + Sigmoid, X + tanh (inplace computation)

Difference between activation functions and layers

Recommendation: implement a simple deep net framework yourself

Remark:

Since $F(\boldsymbol{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

- We are no longer guaranteed to find the global optimum
- Initialization of $\boldsymbol{w}$ matters if w is bad, the solution is not good.

Initialization:

- Not well understood in general
- Needs to break symmetry
- Random uniform

$$\text{Uniform}\left(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}}\right)$$

- Glorot and Bengio (2010)

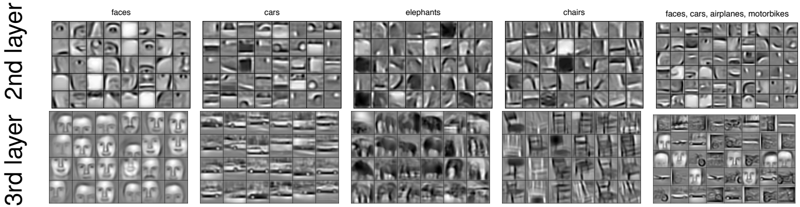
$$\text{Uniform}\left(-\sqrt{\frac{6}{\text{fan in + fan out}}}, \sqrt{\frac{6}{\text{fan in + fan out}}}\right)$$

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



Disadvantage of deep nets compared to usage of features:

Deep nets are computationally demanding (GPUs) and require significant amounts of training data

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances
- Sufficient evidence that it works

This combination lead to significant performance improvements on many datasets

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Normalizes data when training really deep nets
  - Normalize by subtracting mean and dividing by standard deviation

Choices in deep learning packages:

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

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

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]
```

- 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':

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

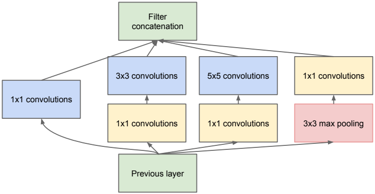
Why? Numerical robustness ('log-sum-exp trick')

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

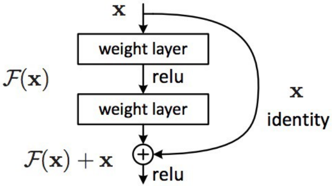
Don't try without, it **will** fail!

Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)



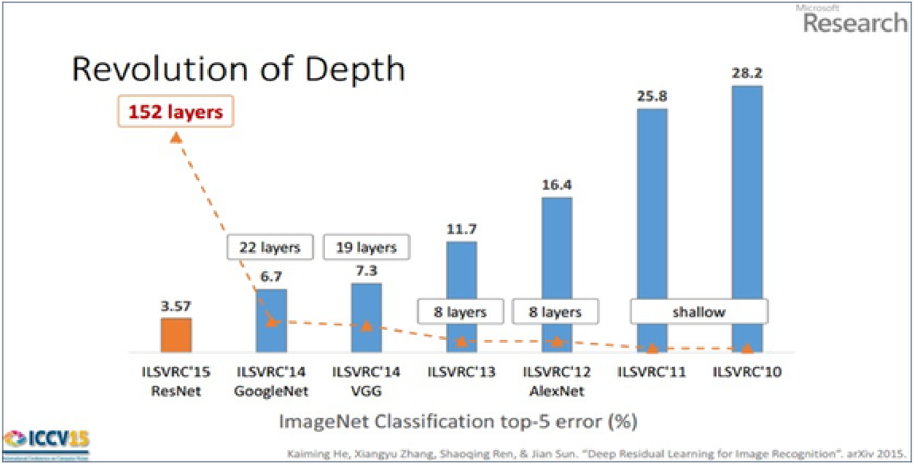
- ResNet (residual connections)



Imagenet Challenge:

- A large dataset: 1.2M images, 1000 categories
- AlexNet was run on the GPU, i.e., sufficient computational resources
- Rectified linear units rather than sigmoid units simplify optimization

Results:


Microsoft
Research
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Quiz:

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

Important topics of this lecture

- Deep nets
- Backpropagation

Up next:

- Pytorch