Some content adapted from material from Andrej Karpathy, Sean Bell, Kavita Bala, and ...
Announcements

- Project 5 (Neural Radiance Fields) due Weds, May 1 by 8pm
- In class final on May 7
  - Allowed two sheets of notes (front and back sides)
- Course evaluations are open starting Monday, April 29
  - We would love your feedback!
  - Small amount of extra credit for filling out
    - What you write is still anonymous, instructors only see whether students filled it out
  - Link coming soon
Readings

• Convolutional neural networks
  – Szeliski (2nd Edition) Chapter 5.4

• Best practices for training CNNs
Deep networks can be used for...

Image classification

\[
f(\text{🍎}) = \text{“apple”} \\
f(\text{🍅}) = \text{“tomato”} \\
f(\text{🐄}) = \text{“cow”}
\]

View synthesis

And much more!
A Recent Example: *Segment Anything*

*Segment Anything*
Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, Ross Girshick
Another Recent Example: Tracking Everything Everywhere All at Once

Tracking Everything Everywhere All At Once

Paper ID: 2206
(with audio 🎧)

Tracking Everything Everywhere All At Once
Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, Noah Snavely
ICCV 2023
Back to convolutional neural networks

Layer types:
- Convolutional layer
- Pooling layer
- Fully-connected layer
Training a network

• Given a network architecture (CNN, MLP, etc) and some training data, how do we actually set the weights of the network?
Gradient descent: iteratively follow the slope

Stochastic gradient descent (SGD)

- Computing the exact gradient over the training set is expensive
- Train on batches of data (e.g., 32 images or 32 rays) at a time
- A full pass through the dataset (i.e., using batches that cover the training data) is called an \textit{epoch}
- Usually need to train for multiple epochs, i.e., multiple full passes through the dataset to converge
- Stochastic gradient descent only approximates the true gradient, but works remarkably well in practice
- Use \textit{backpropagation} to automatically compute gradients on each batch
How do you actually train these things?

Roughly speaking:

- Gather labeled data
- Find a ConvNet architecture
- Minimize the loss

But lots of details to get right!
Training a convolutional neural network

• Split and preprocess your data
• Choose your network architecture
• Initialize your network weights
• Find a learning rate and regularization weight
• Minimize the loss and monitor progress
• Fiddle with knobs...
Why so complicated?

• Training deep networks can be finicky – lots of parameters to learn, complex, non-linear optimization function
What makes training deep networks hard?

• It’s easy to get high training accuracy:
  • Use a huge, fully connected network with tons of layers
  • Let it memorize your training data

• It’s harder to get high test accuracy

This would be an example of overfitting
**Related Question: Why Convolutional Layers?**

- A fully connected layer can generally represent the same functions as a convolutional one
  - Think of the convolutional layer as a version of the FC layer with constraints on parameters

- What is the advantage of CNNs?
Overfitting: More Parameters, More Problems

• Non-Deep Example: consider the function $x^2 + x$
• Let’s take some noisy samples of the function...
Overfitting: More Parameters, More Problems

• Now let's fit a polynomial to our samples of the form $P_N(x) = \sum_{k=0}^{N} x^k p_k$
Overfitting: More Parameters, More Problems

- A model with more parameters can represent more functions

- E.g., if $P_N(x) = \sum_{k=0}^{N} x^k p_k$, then $P_{15}$

- More parameters will often reduce training error but increase testing error. This is overfitting.

- When overfitting happens, models do not generalize well
Deep Learning: More Parameters, More Problems?

• More parameters let us represent a larger space of functions

• The larger that space is, the harder our optimization becomes

• This means we need:
  • More data
  • More compute resources
  • Etc.
Deep Learning: More Parameters, More Problems?

A convolutional layer looks for components of a function that are spatially-invariant.
Overfitting in view synthesis

• What happens if you directly optimize an MPI to reconstruct a small set of input views?
Overfitting in view synthesis

- Answer: you can exactly reconstruct the input views, but produce garbage for new views

DeepView: View synthesis with learned gradient descent

John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, Richard Tucker

Fitting a multi-plane image to a set of views using gradient descent
Overfitting in view synthesis

• Reminiscent of shadow sculptures

Anamorphic Star Wars Shadow Art by Red Hong Yi, via TKSST
Overfitting in view synthesis

SHADOW ART
Niloy J. Mitra, Mark Pauly
ACM SIGGRAPH Asia 2009
Overfitting in view synthesis

- MPI with 64 layers, each storing a 1024 x 768 RGBA image $\rightarrow$ $\sim$200M parameters
- If we have 32 input RGB images of 1024x768 resolution $\rightarrow$ $\sim$75M inputs
- Many more parameters than measurements $\rightarrow$ risk of overfitting

- Compare to NeRF: $\sim$500K - 1M parameters
How to Avoid Overfitting: Regularization

• In general:
  • More parameters means higher risk of overfitting
  • More constraints/conditions on parameters can help

• If a model is overfitting, we can
  • Collect more data to train on
  • Regularize: add some additional information or assumptions to better constrain learning

• Regularization can be done through:
  • the design of architecture
  • the choice of loss function
  • the preparation of data
  • …
• “Bigger” architectures (typically, those with more parameters) tend to be more at risk of overfitting.

• But, we’ll see much bigger architectures (transformers) soon that work well when given lots of training data.
Regularization reduces overfitting

\[ L = L_{\text{data}} + L_{\text{reg}} \]

\[ L_{\text{reg}} = \lambda \frac{1}{2} ||W||_2^2 \]

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]
(1) **Data preprocessing**

Preprocess the data so that learning is better conditioned:

- **original data**
- **zero-centered data**
- **normalized data**

\[
X \leftarrow np.mean(axis=0, \text{keepdims=True})
\]

\[
X \leftarrow np.std(axis=0, \text{keepdims=True})
\]

*Figure: Andrej Karpathy*
(1) Data preprocessing

In practice, often perform a single mean RGB value, and divide by a per-channel standard deviation (recall MOPS, Normalized 8-Point Algorithm)
(1) Data preprocessing

```python
# Data loading code
if args.dummy:
    print("=> Dummy data is used!"
    train_dataset = datasets.FakeData(1281167, (3, 224, 224), 1000, transforms.ToTensor())
    val_dataset = datasets.FakeData(50000, (3, 224, 224), 1000, transforms.ToTensor())
else:
    traindir = os.path.join(args.data, 'train')
    valdir = os.path.join(args.data, 'val')
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])

    train_dataset = datasets.ImageFolder(
        traindir,
        transforms.Compose([
            transforms.RandomResizedCrop(224),
```

Batch normalization

• Side note – can also perform normalization after each layer of the network to stabilize network training ("batch normalization")
(1) Data preprocessing

**Augment the data** — extract random crops from the input, with slightly jittered offsets. Without this, typical ConvNets (e.g. [Krizhevsky 2012]) overfit the data.

**E.g.** 224x224 patches extracted from 256x256 images

Randomly reflect horizontally

Perform the augmentation live during training

*Figure: Alex Krizhevsky*
(2) Choose your architecture

https://playground.tensorflow.org/
(2) Choose your architecture

Very common modern choice for classification problems

“AlexNet” [Krizhevsky et al. NIPS 2012]

“GoogLeNet” [Szegedy et al. CVPR 2015]

“VGG Net” [Simonyan & Zisserman, ICLR 2015]

“ResNet” [He et al. CVPR 2016]
(3) Initialize your weights

Set the weights to small random numbers:

\[ W = \text{np.random.randn}(D, H) * 0.001 \]

(matrix of small random numbers drawn from a Gaussian distribution)

Set the bias to zero (or small nonzero):

\[ b = \text{np.zeros}(H) \]

(if you use ReLU activations, folks tend to initialize bias to small positive number)

*Slide: Andrej Karpathy*
(4) Overfit a small portion of the data

The above code:
- take the first 20 examples from CIFAR-10
- turn off regularization (reg = 0.0)
- use simple vanilla ‘sgd’
(4) Overfit a small portion of the data

```python
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
model, two_layer_net,
num_epochs=200, reg=0.0,
update='sgd', learning_rate_decay=1,
sample_batches = False,
learning_rate=1e-3, verbose=True)
```

**Details:**

- `'sgd'`: vanilla gradient descent (no momentum etc)
- `learning_rate_decay = 1`: constant learning rate
- `sample_batches = False` (full gradient descent, no batches)
- `epochs = 200`: number of passes through the data

*Slide: Andrej Karpathy*
(4) Overfit a small portion of the data

100% accuracy on the training set (good)

Slide: Andrej Karpathy
(4) Find a learning rate

Q: Which one of these learning rates is best to use?
Learning rate schedule

How do we change the learning rate over time?

Various choices:

- Step down by a factor of 0.1 every 50,000 mini-batches (used by SuperVision [Krizhevsky 2012])
- Decrease by a factor of 0.97 every epoch (used by GoogLeNet [Szegedy 2014])
- Scale by sqrt(1-t/max_t) (used by BVLC to re-implement GoogLeNet)
- Scale by 1/t
- Scale by exp(-t)
Summary of things to fiddle with

- Network architecture
- Learning rate, decay schedule, update type (+batch size)
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization
Questions?
Transfer learning

“You need a lot of data if you want to train/use CNNs for a new classification task”
Transfer learning

“You need a lot of data if you want to train/use CNNs for a new classification task”
Transfer learning with CNNs

Step 1: Take a model trained on ImageNet

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeur
Transfer learning with CNNs

Step 2a: If you have a small amount of new data, adjust a small number of network weights

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeung
Transfer learning with CNNs

Step 2b: If you have a larger amount of new data, adjust a larger number of network weights

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeung
<table>
<thead>
<tr>
<th></th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeung
<table>
<thead>
<tr>
<th></th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>Use Linear Classifier on top layer</td>
<td>?</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>Finetune a few layers</td>
<td>?</td>
</tr>
</tbody>
</table>

More specific

More generic

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeung
<table>
<thead>
<tr>
<th>More specific</th>
<th>More generic</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>Use Linear Classifier on top layer</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>Finetune a few layers</td>
</tr>
</tbody>
</table>

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeung
Transfer learning with CNNs is pervasive

- It’s the norm, not the exception
Transfer learning with CNNs is pervasive

• It’s the norm, not the exception
Other pre-trained models are starting to become standard

- Swin-transformer pre-trained on ImageNet-21K
- DINO features
- Foundation models (Stable Diffusion, etc)
Takeaway for your projects and beyond

Have some dataset of interest, but it has \(< < \sim 1M\) images?

1. Find a large dataset with similar data (e.g., ImageNet), train a large CNN
2. Apply transfer learning to fine-tune on your data

For step 1, many existing models exist in “Model Zoos”

Common modern approach: start with a ResNet architecture pre-trained on ImageNet, and fine-tune on your (smaller) dataset

Slide credit: Fei-Fei Li, Justin Johnson, and Serena Yeung
Questions?