CS 3710: Advanced Topics in AI

Introduction

Prof. Adriana Kovashka
University of Pittsburgh
August 28, 2023
Course Info

- **Course website:**
  http://people.cs.pitt.edu/~kovashka/cs3710_fa23

- **Instructor:** Adriana Kovashka
  (kovashka@cs.pitt.edu)

- **Office:** Sennott Square 5325

- **Class:** Mon/Wed, 9:30am-10:45am

- **Office hours:** by appointment

- **Zoom (for some appointments):**
  https://pitt.zoom.us/s/4168010698
About the Instructor

Born 1985 in Sofia, Bulgaria

Got BA in 2008 at Pomona College, CA (Computer Science & Media Studies)

Got PhD in 2014 at University of Texas at Austin (Computer Vision)
Course Goals

• To learn about state-of-the-art approaches to computer vision tasks
• To think critically about vision approaches, see strengths, weaknesses, and connections between works
• To conduct research with contributions including novel methods, method comparison and method/data analysis
Note on Course Climate

• Some may be more familiar with the subject of the seminar, some less, and that’s ok!
• Please respect each other and listen to each other carefully
• You will need to collaborate on the course project—work hard, be fair and honest, and try to talk through problems
• Talk to instructor openly
Plan for First Three Classes

• Course logistics
• Students introduce themselves
• What is computer vision?
• Preview readings
• Paper presentation signups (by Sept. 1)
• Some basics
• Preliminary project pitches exercise
Policies and Schedule

https://people.cs.pitt.edu/~kovashka/cs3710_fa23/

Highlights:

• Paper presentations
• Project logistics
• Readings and schedule
Questions?
Blitz introductions (30 sec)

• What is your name?
• What one thing outside of school are you passionate about?
• What is your current or planned research about?
• What do you hope to get out of this class? (Optional)

• When you speak, please remind me your name
What is Computer Vision?
What is computer vision?

Done?

"We see with our brains, not with our eyes“ (Oliver Sacks and others)

Kristen Grauman (adapted)
What is computer vision?

- Automatic understanding of images and video
  - Algorithms and representations to allow a machine to recognize objects, people, scenes, and activities
  - Algorithms to mine, search, and interact with visual data
  - Computing properties and navigating within the 3D world using visual data
  - Generating realistic synthetic visual data

Adapted from Kristen Grauman
Perception and interpretation

Objects
- The Wicked Twister
- Lake Erie
- Cedar Point

Activities
- Ferris wheel
- 12 E

Scenes
- sky
- water
- deck
- bench
- tree

Locations
- amusement park
- people waiting in line

Text / writing
- Kristen Grauman

Faces
- people sitting on ride

Gestures
- umbrellas

Motions
- maxair

Emotions
Visual search, organization

Query -> Image or video archives -> Relevant content

Kristen Grauman
Measurement

Real-time stereo

Structure from motion

Multi-view stereo for community photo collections

Pollefeys et al.

Goesele et al.
Generation

Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018
Recognition in novel modalities

Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.
Learning from weak supervision

The elephant are about to march through them. The spiders themselves have a span as wide as a

But the love serenade is over once a dog arrives.

Australian camels appear sick and emaciated.

Tigers are one of the few cats that actually enjoy swimming.

Male koalas play no role in parenting.

About 50 animals have died in just three months, including this adult orangutan on the day we

Unlike mechanics, langurs are the friends of spotted deer.

There's a turf war going on and the koalas are losing. (dog)

The mayor has declined offers of assistance and expert advice from animal welfare groups. (elephant)
Understanding activities and intents
Understanding stories in film

Video Clip

Legend
- Character
- Attribute
- Relationship
- Interaction
- Summary Int.
- Topic
- Reason
- Timestamp

Scene: Field Road
Situation: Bullying
Description:
As Jenny and Forrest are on the road, three boys start throwing rocks at Forrest. Jenny urges him to run from them. While Forrest runs, his leg braces fall apart.
Reasoning and acting: Embodied question answering
Related disciplines

- Artificial intelligence
- Machine learning
- Cognitive science
- Computer vision
- Algorithms
- Image processing
- Graphics

Kristen Grauman
Why vision?

• Images and video are everywhere!

- Personal photo albums
- Movies, news, sports
- Surveillance and security
- Medical and scientific images

Adapted from Lana Lazebnik
Why vision?

• As image sources multiply, so do applications
  – Relieve humans of boring, easy tasks
  – Perception for robotics / autonomous agents
  – Organize and give access to visual content
  – Description of content for the visually impaired
  – Human-computer interaction
  – Fun applications (e.g. art styles to my photos)
  – ...
  – What else?

Adapted from Kristen Grauman
Seeing AI

Microsoft Cognitive Services: Introducing the Seeing AI project
Why are these tasks challenging?
Recognition: What objects do you see?

- building
- street
- carriage
- horse
- table
- person
- car
- balcony
Detection: Where are the cars?
Activity: What is this person doing?
Scene: Is this an indoor scene?
Instance: Which city? Which building?
Visual question answering:
Why is there a carriage in the street?
Why is vision difficult?

• Ill-posed problem: real world much more complex than what we can measure in images
  – 3D $\rightarrow$ 2D
  – Motion $\rightarrow$ static
• Impossible to literally “invert” image formation process with limited information
  – Need information outside of this particular image to generalize what image portrays (e.g. to resolve occlusion)

Adapted from Kristen Grauman
Challenges: many nuisance parameters

- Illumination
- Object pose
- Clutter
- Occlusions
- Intra-class appearance
- Viewpoint
Challenges: intra-class variation

slide credit: Fei-Fei, Fergus & Torralba
Challenges: importance of context

slide credit: Fei-Fei, Fergus & Torralba
Challenges: Complexity

- Thousands to millions of pixels in an image
- 3,000-30,000 human recognizable object categories
- 30+ degrees of freedom in the pose of articulated objects (humans)
- Billions of images indexed by Google Image Search
- 1.424 billion smart camera phones sold in 2015
- About half of the cerebral cortex in primates is devoted to processing visual information [Felleman and van Essen 1991]
Challenges: Limited supervision

Less

Unlabeled, multiple objects

Classes labeled, some clutter

More

Cropped to object, parts and classes
Problem with categorization
(Borges' Animals)

“These ambiguities, redundancies and deficiencies recall those that Dr. Franz Kuhn attributes to a certain Chinese dictionary entitled *The Celestial Emporium of Benevolent Knowledge*. In its remote pages it is written that animals can be divided into (a) those belonging to the Emperor, (b) those that are embalmed, (c) those that are tame, (d) pigs, (e) sirens, (f) imaginary animals, (g) wild dogs, (h) those included in this classification, (i) those that are crazy-acting, (j) those that are uncountable, (k) those painted with the finest brush made of camel hair, (l) miscellaneous, (m) those which have just broken a vase, and (n) those which, from a distance, look like flies."

Preview of Readings
Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset’s classes.
Figure 2: An overview of using ViLD for open-vocabulary object detection. ViLD distills the knowledge from a pretrained open-vocabulary image classification model. First, the category text embeddings and the image embeddings of cropped object proposals are computed, using the text and image encoders in the pretrained classification model. Then, ViLD employs the text embeddings as the region classifier (ViLD-text) and minimizes the distance between the region embedding and the image embedding for each proposal (ViLD-image). During inference, text embeddings of novel categories are used to enable open-vocabulary detection.
Figure 1. VISPROG is a modular and interpretable neuro-symbolic system for compositional visual reasoning. Given a few examples of natural language instructions and the desired high-level programs, VISPROG generates a program for any new instruction using in-context learning in GPT-3 and then executes the program on the input image(s) to obtain the prediction. VISPROG also summarizes the intermediate outputs into an interpretable visual rationale (Fig. 4). We demonstrate VISPROG on tasks that require composing a diverse set of modules for image understanding and manipulation, knowledge retrieval, and arithmetic and logical operations.

Fig. 1 Prompt engineering vs Context Optimization (CoOp). The former needs to use a held-out validation set for words tuning, which is inefficient; the latter automates the process and requires only a few labeled images for learning.

Figure 1: Schematic of the method. (Left) The standard method of a zero-shot open vocabulary image classification model (e.g., CLIP (Radford et al., 2021)). (Right) Our method of CuPL. First, an LLM generates descriptive captions for given class categories. Next, an open vocabulary model uses these captions as prompts for performing classification.
Figure 2: Cycle-consistent adversarial adaptation overview. By directly remapping source training data into the target domain, we remove the low-level differences between the domains, ensuring that our task model is well-conditioned on target data. We depict here the image-level adaptation as composed of the pixel GAN loss (green), the source cycle loss (red), and the source and target semantic consistency losses (black dashed) – used when needed to prevent label flipping. For clarity the target cycle is omitted. The feature-level adaptation is depicted as the feature GAN loss (orange) and the source task loss (purple).

Figure 1. We propose to separate and localize audio sources based on a natural language query, by learning to align the modalities on completely unlabeled videos. In comparison, prior audio-visual sound separation approaches require object label supervision.
Figure 1. Ego4D is a massive-scale egocentric video dataset of daily life activity spanning 74 locations worldwide. Here we see a snapshot of the dataset (5% of the clips, randomly sampled) highlighting its diversity in geographic location, activities, and modalities. The data includes social videos where participants consented to remain unblurred. See https://ego4d-data.org/fig1.html for interactive figure.

Fig. 1: Example advertisements from our dataset that require challenging visual recognition and reasoning. Despite the potential applications of understanding the messages of ads, this problem has not been tackled in computer vision.
Figure 1. Make-A-Scene: Samples of generated images from text inputs (a), and a text and scene input (b). Our method is able to both generate the scene (a, bottom left) and image, or generate the image from text and a simple sketch input (b, center).
Figure 1: LLMs have not interacted with their environment and observed the outcome of their responses, and thus are not grounded in the world. SayCan grounds LLMs via value functions of pretrained skills, allowing them to execute real-world, abstract, long-horizon commands on robots.
Paper Presentation Sign-ups

https://docs.google.com/spreadsheets/d/14g6_finym215LqDZ2qnAE57bU5j5M9uHz3Tmmgq484U/edit?usp=sharing
Some Basics
(breezing through to establish common ground...)

Convolutional networks
Recurrent networks
Transformers
Self-supervised learning
Image formation

Illumination (energy) source

Imaging system

Scene element

(Internal) image plane (film)

Slide credit: Derek Hoiem
Digital images

Figure 2.17 (a) Continuous image projected onto a sensor array. (b) Result of image sampling and quantization.

- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
Digital images

- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
- What does quantizing signal look like?

Image thus represented as a matrix of integer values.

Adapted from S. Seitz
Digital color images

Bayer filter

© 2000 How Stuff Works

Slide credit: Kristen Grauman
Digital color images

Color images, RGB color space:

Split image into three channels

Adapted from Kristen Grauman
Images as Matrices

• Vectors and matrices are just collections of ordered numbers that represent something: movements in space, word counts, movie ratings, pixel brightnesses, etc.
Vectors have two main uses

- Vectors can represent an offset in 2D or 3D space
- Points are just vectors from the origin
- Data can also be treated as a vector
- Such vectors don’t have a geometric interpretation, but calculations like “distance” still have value
Vector

- A column vector $\mathbf{v} \in \mathbb{R}^{n \times 1}$ where
  $$
  \mathbf{v} = \begin{bmatrix}
  v_1 \\
  v_2 \\
  \vdots \\
  v_n
  \end{bmatrix}
  $$

- A row vector $\mathbf{v}^T \in \mathbb{R}^{1 \times n}$ where
  $$
  \mathbf{v}^T = \begin{bmatrix}
  v_1 & v_2 & \ldots & v_n
  \end{bmatrix}
  $$

$T$ denotes the transpose operation.
Norms

• L1 norm

\[ \| \mathbf{x} \|_1 := \sum_{i=1}^{n} |x_i| \]

• L2 norm

\[ \| \mathbf{x} \| := \sqrt{x_1^2 + \cdots + x_n^2} \]

• L^p norm (for real numbers p ≥ 1)

\[ \| \mathbf{x} \|_p := \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p} \]
Distances

• L1 (Manhattan) distance

\[ d_1(p, q) = \| p - q \|_1 = \sum_{i=1}^{n} |p_i - q_i|, \]

• L2 (Euclidean) distance

\[ d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \]

\[ d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}. \]
Example: Feature representation

- A vector representing measurable characteristics of a data sample we have
- E.g. a glass of juice can be represented via its color = \{\text{yellow}=1, \text{red}=2, \text{green}=3, \text{purple}=4\} and taste = \{\text{sweet}=1, \text{sour}=2\}
- A given glass $i$ can be represented as a vector: $x_i = [3 \ 2]$ represents green, sour juice
- For $D$ features, this defines a $D$-dimensional space where we can measure similarity between samples
Example: Feature representation

E.g. a glass of juice can be represented via its color = \{yellow=1, red=2, green=3, purple=4\} and taste = \{sweet=1, sour=2\}

L2 distance:
\[ d(x_1, x_2) = \sqrt{4+0} \]
\[ d(x_1, x_3) = \sqrt{0+1} \]
\[ d(x_2, x_3) = \sqrt{4+1} \]

L1 distance:
\[ d(x_1, x_2) = 2+0 \]
\[ d(x_1, x_3) = 0+1 \]
\[ d(x_2, x_3) = 2+1 \]
Inner (Dot) Product

• Multiply corresponding entries of two vectors and add up the result

\[ x^T y = [x_1 \ \ldots \ \ x_n] \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i \quad \text{(scalar)} \]

• \( x \cdot y \) is also \( |x| \ ||y|| \cos(\text{angle between } x \text{ and } y) \)

• If \( B \) is a unit vector, then \( A \cdot B \) gives the length of \( A \) which lies in the direction of \( B \) (projection)

(if \( B \) is unit-length hence norm is 1)
Traditional Recognition Approach

- Features are key to recent progress in recognition, but research shows they’re flawed... Where next?

What about learning the features?

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly
“Shallow” vs. “deep” architectures

Traditional recognition: “Shallow” architecture

Image/Video Pixels → Hand-designed feature extraction → Trainable classifier → Object Class

Deep learning: “Deep” architecture

Image/Video Pixels → Layer 1 → … → Layer N → Simple classifier → Object Class
Neural network definition

- Activations:
  \[ a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \]

- Nonlinear activation function \( h \) (e.g. sigmoid, RELU):
  \[ z_j = h(a_j) \]

Recall SVM:
\[ w^T x + b \]

Figure 5.1: Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables \( x_0 \) and \( z_0 \). Arrows denote the direction of information flow through the network during forward propagation.

Figure from Christopher Bishop
Neural network definition

• Layer 2

\[ a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \]

• Layer 3 (final)

\[ a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \]

• Outputs (e.g. sigmoid/softmax)

\[
\begin{align*}
(y_k = \sigma(a_k) &= \frac{1}{1 + \exp(-a_k)} \\
y_k &= \frac{\exp(a_k)}{\sum_j \exp(a_j)}
\end{align*}
\]

• Finally:

\[
y_k(x, w) = \sigma \left( \sum_{j=1}^{M} w_{kj}^{(2)} h \left( \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)
\]
Activation functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \text{max}(0, x) \]

**Leaky ReLU**
\[ \text{max}(0.1x, x) \]

**Maxout**
\[ \text{max}(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ f(x) = \begin{cases} 
   x & \text{if } x > 0 \\
   \alpha (\exp(x) - 1) & \text{if } x \leq 0 
\end{cases} \]

Andrej Karpathy
A multi-layer neural network

- **Nonlinear** classifier
- Can approximate any continuous function to arbitrary accuracy given sufficiently many hidden units
Inspiration: Neuron cells

- **Neurons**
  - accept information from multiple inputs,
  - transmit information to other neurons.
- **Multiply inputs by weights along edges**
- **Apply some function to the set of inputs at each node**
- **If output of function over threshold, neuron “fires”**

Text: HKUST, figures: Andrej Karpathy
Biological analog

A biological neuron

An artificial neuron

Sigmoid function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$
Biological analog

Hubel and Weisel’s architecture

Multi-layer neural network

Adapted from Jia-bin Huang
Deep neural networks

- Lots of hidden layers
- Depth = power (usually)
- “With great power comes great responsibility”
How do we train deep neural networks?

• The goal is to find such a set of weights that allow the activations/outputs to match the desired output: \( f(W, x_i) \sim y_i \)

• Unfortunately, no closed-form solution for weights \( W \), but we can express our objective

• We want to minimize a loss function (a function of the weights in the network), we’ll do so iteratively

• For now let’s simplify and assume there’s a single layer of weights in the network
Classification goal

Example dataset: CIFAR-10
10 labels
50,000 training images
each image is 32x32x3
10,000 test images
Classification scores

\[ f(x, W) = Wx \]

[32x32x3]
array of numbers 0...1
(3072 numbers total)
Linear classifier

\[ f(x, W) = Wx + (b) \]

[32x32x3] array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”
Linear classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

input image

stretch pixels into single column

<table>
<thead>
<tr>
<th>W</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>56</td>
</tr>
<tr>
<td>-0.5</td>
<td>1.1</td>
</tr>
<tr>
<td>0.1</td>
<td>231</td>
</tr>
<tr>
<td>2.0</td>
<td>3.2</td>
</tr>
<tr>
<td>1.5</td>
<td>24</td>
</tr>
<tr>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>1.2</td>
</tr>
<tr>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>-0.3</td>
<td></td>
</tr>
</tbody>
</table>

\[ f(x_i; W, b) \]

cat score: -96.8

dog score: 437.9

ship score: 61.95
Going forward: Loss function/Optimization

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function *(optimization)*

<table>
<thead>
<tr>
<th></th>
<th>1.3</th>
<th>2.2</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td></td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>
Linear classifier

Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th>Category</th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Adapted from Andrej Karpathy
Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Hinge loss:

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$

the loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Want: $s_{y_i} \geq s_j + 1$, for $j \neq y_i$

i.e. $s_j - s_{y_i} + 1 \leq 0$

If true, loss is 0
If false, loss is magnitude of violation

Adapted from Andrej Karpathy
## Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>scores</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Losses:** 2.9

### Hinge loss:

Given an example $\mathbf{(x_i, y_i)}$ where $\mathbf{x}_i$ is the image and $\mathbf{y}_i$ is the (integer) label,

and using the shorthand for the scores vector: $\mathbf{s} = f(\mathbf{x}_i, W)$

the loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, \mathbf{s}_j - \mathbf{s}_{y_i} + 1)$$

$$= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$

Adapted from Andrej Karpathy
### Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
<td></td>
</tr>
</tbody>
</table>

**Hinge loss:**

Given an example $\left(x_i, y_i\right)$ where $x_i$ is the image and $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)$$

$$= \max(0, -2.6) + \max(0, -1.9)$$

$$= 0 + 0$$

$$= 0$$

Adapted from Andrej Karpathy
Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes. With some \( W \) the scores \( f(x, W) = Wx \) are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Losses: 2.9 0 12.9

Hinge loss:

Given an example \( (x_i, y_i) \) where \( x_i \) is the image and \( y_i \) is the (integer) label, and using the shorthand for the scores vector: \( s = f(x_i, W) \)

the loss has the form:

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\]

\[
= \max(0, 2.2 - (-3.1) + 1)
+ \max(0, 2.5 - (-3.1) + 1)
= \max(0, 5.3 + 1)
+ \max(0, 5.6 + 1)
= 6.3 + 6.6
= 12.9
\]

Adapted from Andrej Karpathy
# Linear classifier: Hinge loss

Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Losses:</td>
<td>2.9</td>
<td>0</td>
<td>12.9</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Hinge loss:**

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

$$L = \frac{(2.9 + 0 + 12.9)}{3} = \frac{15.8}{3} = 5.3$$

Adapted from Andrej Karpathy
Linear classifier: Hinge loss

\[
f(x, W) = W x
\]

\[
L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)
\]
Linear classifier: Hinge loss

Weight Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

$\lambda =$ regularization strength (hyperparameter)

In common use:

**L2 regularization**

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

**L1 regularization**

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

**Dropout** (will see later)

Adapted from Andrej Karpathy
Another loss: Softmax (cross-entropy)

scores = unnormalized log probabilities of the classes

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

where \( s = f(x_i; W) \)

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

\[ L_i = -\log P(Y = y_i | X = x_i) \]

Adapted from Andrej Karpathy
Another loss: Softmax (cross-entropy)

\[ L_i = -\log \left( \frac{e^{sy_i}}{\sum_j e^{sj}} \right) \]

<table>
<thead>
<tr>
<th>Unnormalized Log Probabilities</th>
<th>Unnormalized Probabilities</th>
<th>Normalized Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat 3.2</td>
<td>24.5</td>
<td>0.13</td>
</tr>
<tr>
<td>car 5.1</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>frog -1.7</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>

\[ L_i = -\log(0.13) = 0.89 \]

Adapted from Andrej Karpathy
Other losses

- Triplet loss (Schroff, FaceNet)

\[
\sum_{i}^{N} \left[ \| f(x^a_i) - f(x^p_i) \|_2^2 - \| f(x^a_i) - f(x^n_i) \|_2^2 + \alpha \right]_+
\]

a denotes anchor
p denotes positive
n denotes negative

Figure 3. The Triplet Loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

- Anything you want!
How to minimize the loss function?
How to minimize the loss function?

In 1-dimension, the derivative of a function is:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

In multiple dimensions, the gradient is the vector of (partial derivatives):

That is, for $f: \mathbb{R}^n \to \mathbb{R}$, its gradient $\nabla f: \mathbb{R}^n \to \mathbb{R}^n$ is defined at the point $p = (x_1, \ldots, x_n)$ in $n$-dimensional space as the vector:[b]

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

The nabla symbol $\nabla$, written as an upside-down triangle and pronounced "del", denotes the vector differential operator.

Adapted from Andrej Karpathy, definition/equation from https://en.wikipedia.org/wiki/Gradient
Loss gradients

• Different notations: \( \frac{\partial E}{\partial w_{ji}^{(1)}} \) \( \nabla_W L \)

• i.e. how does the loss change as a function of the weights

• We want to change weights in a way that makes the loss decrease as fast as possible
Gradient descent

• We’ll update weights
• Move in direction opposite to gradient:

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \]

Figure from Andrej Karpathy
Gradient descent

- Iteratively *subtract* the gradient with respect to the model parameters (w)
- I.e. we’re moving in a direction opposite to the gradient of the loss
- I.e. we’re moving towards *smaller* loss
How to compute the loss/gradient?

• In classic gradient descent, we compute the gradient from the loss for all training examples

• Mini-batch gradient descent: Only use some of the data for each gradient update, cycle through training examples multiple times
  • Each time we’ve cycled through all of them once is called an ‘epoch’
  • Allows faster training (e.g. on GPUs), parallelization
  • Some benefits for learning due to randomness
Learning rate selection

The effects of step size (or “learning rate”)

https://www.deeplearning.ai/ai-notes/optimization/

Andrej Karpathy
Gradient descent in multi-layer nets

- We’ll update weights
- Move in direction opposite to gradient:

\[ w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \]

- How to update the weights at all layers?
- Answer: backpropagation of error from higher layers to lower layers
Backpropagation: Graphic example

First calculate error of output units and use this to change the top layer of weights.

Calculate how to update weights into $j$ (update at end of iter)

Adapted from Ray Mooney, equations from Chris Bishop
Backpropagation: Graphic example

Next calculate error for hidden units based on errors on the output units it feeds into.

Adapted from Ray Mooney, equations from Chris Bishop
Backpropagation: Graphic example

Finally update bottom layer of weights based on errors calculated for hidden units.

Update weights into $i$

Adapted from Ray Mooney, equations from Chris Bishop
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

E.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:
\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}
\]
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \, y = 5, \, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

Chain rule:
\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}
\]
activations
activations

\[ \frac{\partial z}{\partial x} \]

“local gradient”

\[ \frac{\partial z}{\partial y} \]

generates

\[ f \]

\[ z \]

generates

 \[ x \]

 \[ y \]
activations

\[ \frac{\partial z}{\partial x} \]

“local gradient”

\[ \frac{\partial z}{\partial y} \]

\[ z \]

\[ \frac{\partial L}{\partial z} \]

gradients
activations

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

“local gradient”

\[
\frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial z}
\]

gradients
activations

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

“local gradient”

\[
\frac{\partial z}{\partial x}
\]

\[
\frac{\partial z}{\partial y}
\]

\[
\frac{\partial L}{\partial z}
\]

gradients
Convolutional Neural Networks (CNN)

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant, *more abstract* features
- Classification layer at the end


Adapted from Rob Fergus
Convolutional Neural Networks (CNN)

• Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Apply non-linearity
  3. Spatial pooling (downsample)

• Supervised training of convolutional filters by back-propagating classification error

Adapted from Lana Lazebnik
1. Convolution

- Apply learned filter weights
- One feature map per filter
- Stride can be greater than 1 (faster, less memory)

Adapted from Rob Fergus
2. Non-Linearity

- Per-element (independent)
- Some options:
  - Tanh
  - Sigmoid: $1/(1+\exp(-x))$
  - Rectified linear unit (ReLU)
    - Avoids saturation issues

Adapted from Rob Fergus
3. Spatial Pooling

- Sum or max over non-overlapping / overlapping regions

Rob Fergus, figure from Andrej Karpathy
3. Spatial Pooling

- Sum or max over non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (neurons see more of input)

Adapted from Rob Fergus
Convolutions: More detail

32x32x3 image

32 \text{ height}

32 \text{ width}

3 \text{ depth}
Convolutions: More detail

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolutions: More detail

Convolution Layer

The result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5\times5\times3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolutions: More detail

Convolution Layer

- 32x32x3 image
- 5x5x3 filter
- Convolve (slide) over all spatial locations

activation map
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps
Convolutions: More detail

For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Convolutions: More detail

**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

![Diagram of ConvNet layers with dimensions and activation functions](image-url)
ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

**Preview:**

- CONV, ReLU (e.g., 6x5x5x3 filters)
- CONV, ReLU (e.g., 10x5x5x6 filters)
- CONV, ReLU
Convolutions: More detail

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Convolutions: More detail

We call the layer convolutional because it is related to convolution of two signals:

$$G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v]F[i + u, j + v]$$

Element-wise multiplication and sum of a filter and the signal (image)

Adapted from Andrej Karpathy, Kristen Grauman
Convolutions: More detail

A closer look at spatial dimensions:

- 32x32x3 image
- 5x5x3 filter
- convolve (slide) over all spatial locations

activation map
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied \textbf{with stride 2} => 3x3 output!
Convolutions: More detail

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Convolutions: More detail

Output size: 
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):
- stride 1 \(\Rightarrow (7 - 3)/1 + 1 = 5\)
- stride 2 \(\Rightarrow (7 - 3)/2 + 1 = 3\)
- stride 3 \(\Rightarrow (7 - 3)/3 + 1 = 2.33 \)\
In practice: Common to zero pad the border

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & & & & & \\
0 & & & & & \\
0 & & & & & \\
0 & & & & & \\
0 & & & & & \\
\end{array}
\]

E.g. input 7x7
3x3 filter, applied with \textbf{stride 1}
\textbf{pad with 1 pixel} border \implies what is the output?

(recall:)
\[(N - F) / \text{stride} + 1\]
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

7x7 output!
In practice: Common to zero pad the border

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border \(\Rightarrow\) what is the output?

7x7 output!
in general, common to see CONV layers with
stride 1, filters of size FxF, and zero-padding with
\((F-1)/2\). (will preserve size spatially)
e.g. \(F = 3 \Rightarrow\) zero pad with 1
\(F = 5 \Rightarrow\) zero pad with 2
\(F = 7 \Rightarrow\) zero pad with 3

\[(N + 2\text{padding} - F) / \text{stride} + 1\]
Convolutions: More detail

Examples time:

Input volume: **32x32x3**
10 5x5x3 filters with stride 1, pad 2

Output volume size: ?
Convolutions: More detail

Examples time:

Input volume: $32\times32\times3$
10 $5\times5\times3$ filters with stride 1, pad 2

Output volume size:
$(32+2*2-5)/1+1 = 32$ spatially, so $32\times32\times10$
Convolutions: More detail

Examples time:

Input volume: \(32\times32\times3\)
10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?
Convolutions: More detail

Examples time:

Input volume: 32x32x3
10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has $5*5*3 + 1 = 76$ params (+1 for bias)
=> $76*10 = 760$
Putting it all together
Layer 1

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 2

- Activations projected down to pixel level via deconvolution
- Patches from validation images that give maximal activation of a given feature map

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 3

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 4 and 5

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Occlusion experiments

(a) Input Image
True Label: Pomeranian

(d) Classifier, probability of correct class

(as a function of the position of the square of zeros in the original image)

True Label: Car Wheel

True Label: Afghan Hound

[Zeiler & Fergus 2014]
Occlusion experiments

(as a function of the position of the square of zeros in the original image)

[Zeiler & Fergus 2014]
What image maximizes a class score?

Repeat:
1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an “image update”
What image maximizes a class score?

[Understanding Neural Networks Through Deep Visualization, Yosinski et al., 2015]
http://yosinski.com/deepvis
Shape vs texture

ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness [Geirhos et al., ICLR 2019]
A Basic Architecture: AlexNet

Figure from http://www.mdpi.com/2072-4292/7/11/14680/htm
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

- [227x227x3] INPUT
- [55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0
- [27x27x96] **MAX POOL1**: 3x3 filters at stride 2
- [27x27x96] **NORM1**: Normalization layer
- [27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2
- [13x13x256] **MAX POOL2**: 3x3 filters at stride 2
- [13x13x256] **NORM2**: Normalization layer
- [13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1
- [13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1
- [13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1
- [6x6x256] **MAX POOL3**: 3x3 filters at stride 2
- [4096] **FC6**: 4096 neurons
- [4096] **FC7**: 4096 neurons
- [1000] **FC8**: 1000 neurons (class scores)

**Details/Retrospectives:**
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error
Case Study: GoogLeNet

[ Szegedy et al., 2014 ]

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: ResNet

[He et al., 2015]
ILSVRC 2015 winner (3.6% top 5 error)

MSRA @ ILSVRC & COCO 2015 Competitions

• 1st places in all five main tracks
  • ImageNet Classification: “Ultra-deep” (quote Yann) 152-layer nets
  • ImageNet Detection: 16% better than 2nd
  • ImageNet Localization: 27% better than 2nd
  • COCO Detection: 11% better than 2nd
  • COCO Segmentation: 12% better than 2nd

*improvements are relative numbers

Slide from Kaiming He’s presentation https://www.youtube.com/watch?v=1PGLj-uKT1w
Case Study: ResNet

Revolution of Depth

ImageNet Classification top-5 error (%)


(slide from Kaiming He’s presentation)
Case Study: ResNet

[He et al., 2016]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC’15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC’15 and COCO’15!
Case Study: ResNet

[He et al., 2016]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

Q: What’s strange about these training and test curves? 
[Hint: look at the order of the curves]

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!
Case Study: ResNet

[He et al., 2016]

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.
Case Study: ResNet

[He et al., 2016]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

\[ H(x) = F(x) + x \]

Use layers to fit residual \( F(x) = H(x) - x \) instead of \( H(x) \) directly.
Case Study: ResNet

[He et al., 2016]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
Training: Best practices

• Data
  • Center (subtract mean from) your data
  • Use data augmentation
  • Use mini-batch

• Weights/activations
  • To initialize weights, use “Xavier initialization”
  • Use regularization
  • Use RELU (most common), don’t use sigmoid

• Hyperparameters:
  • Learning rate: too high? Too low?
  • Use cross-validation to pick
Over-training prevention

• Running too many epochs can result in over-fitting.

• Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.

Adapted from Ray Mooney
Regularization: Dropout

- Randomly turn off some neurons
- Allows individual neurons to independently be responsible for performance

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Adapted from Jia-bin Huang
Data Augmentation (Jittering)

Create *virtual* training samples

- Horizontal flip
- Random crop
- Color casting
- Geometric distortion

Jia-bin Huang, Image: [https://github.com/aleju/imgaug](https://github.com/aleju/imgaug)
Transfer Learning

“You need a lot of data if you want to train deep CNNs”
Transfer Learning with CNNs

• The more weights you need to learn, the more data you need
• That’s why with a deeper network, you need more data to train than for a shallower net
• One possible solution:

Set these to the already learned weights from another network

Learn these on your own task
Transfer Learning with CNNs

<table>
<thead>
<tr>
<th>more generic</th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>more specific</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very little data</th>
<th>Use simple classifier from top layer</th>
<th>You’re in trouble… Try simple classifier from different stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>quite a lot of data</td>
<td>Finetune a few layers</td>
<td>Finetune a larger number of layers</td>
</tr>
</tbody>
</table>

Adapted from Andrej Karpathy
Pre-training on ImageNet

- Have a source domain and target domain
- Train a network to classify ImageNet classes
  - Coarse classes and ones with fine distinctions (dog breeds)
- Remove last layers and train layers to replace them, that predict target classes

*Oquab et al., “Learning and Transferring Mid-Level Image Representations…”, CVPR 2014*
Transfer learning with CNNs is pervasive…

Object Detection
Ren et al., “Faster R-CNN“, NIPS 2015

Image Captioning

CNN pretrained on ImageNet

Adapted from Andrej Karpathy
Semantic segmentation

Extract patch → Run through a CNN → Classify center pixel

Repeat for every pixel

Lecture 13 - 28
Analysis of pre-training on ImageNet

• **Source:**
  • distinguish 1000 ImageNet categories (incl. many dog breeds)

• **Target tasks:**
  • object detection and action recognition on PASCAL
  • scene recognition on SUN

• Pre-training with 500 images per class is about as good as pre-training with 1000

• Pre-training with 127 classes is about as good as pre-training with 1000

• Pre-training with (fewer classes, more images per class) > (more classes, fewer images)

• Small drop in if classes with fine-grained distinctions removed from pre-training set

_Huh et al., “What makes ImageNet good for transfer learning?”, arxiv 2016_
Recurrent Networks offer a lot of flexibility:

e.g. image captioning
image -> sequence of words
Recurrent Neural Network

We can process a sequence of vectors $\mathbf{x}$ by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

where $h_t$ is the new state, $h_{t-1}$ is the old state, $x_t$ is the input vector at some time step, and $f_W$ is some function with parameters $W$. 

Andrej Karpathy
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”

What kind of loss can we formulate?
What do we still need to specify, for this to work?
Neural Machine Translation (NMT)

The sequence-to-sequence model

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding.*

Source sentence (input)

Decoder RNN produces an encoding of the source sentence.

Target sentence (output)

Note: This diagram shows *test time* behavior: decoder output is fed in ...⇒ as next step’s input.
Sequence-to-sequence: the bottleneck problem

Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

Source sentence (input)

Target sentence (output)

<START> he hit me with a pie <END>

Abigail See
Attention

- **Attention** provides a solution to the bottleneck problem.

- **Core idea:** on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First we will show via diagram (no equations), then we will show with equations
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

il a m’ entarté <START>

Attention scores

dot product
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

Abigail See
Sequence-to-sequence with attention

Encoder RNN

Decoder RNN

Attention scores

dot product

Source sentence (input)

Abigail See
Sequence-to-sequence with attention

Encoder RNN

{ Attention scores

dot product

Decoder RNN

Source sentence (input)

il a m’ entarté <START>

Abigail See
Sequence-to-sequence with attention

On this decoder timestep, we’re mostly focusing on the first encoder hidden state (“he”)

Take softmax to turn the scores into a probability distribution
Sequence-to-sequence with attention

Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

Source sentence (input)

Abigail See
**Sequence-to-sequence with attention**

- **Encoder RNN**:
  - Input sequence: *il, a, m’, entarté*  
  - Attention scores
  - Attention distribution

- **Attention output**
  - Connected to the decoder RNN

- **Decoder RNN**
  - Output: *he*

**Attention**

- Concatenate **attention output** with **decoder hidden state**, then use to compute $y_1$ as before.

**Source sentence (input)**

- *il, a, m’, entarté, <START>*

Abigail See
Attention: in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention score $e^t$ for this step:
  $$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$
- We take softmax to get the attention distribution $\alpha^t$ for this step (this is a probability distribution and sums to 1)
  $$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$
- We use $\alpha^t$ to take a weighted sum of the encoder hidden states to get the attention output $a_t$
  $$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$
- Finally we concatenate the attention output $a_t$ with the decoder hidden state $s_t$ and proceed as in the non-attention seq2seq model
  $$[a_t; s_t] \in \mathbb{R}^{2h}$$

Abigail See
Attention is great

- Attention significantly improves NMT performance
  - It’s very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself
Attention is a general Deep Learning technique

- We’ve seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- **However**: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

**More general definition of attention:**
- Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the *query* *attends to* the *values*.
- For example, in seq2seq + attention model, each decoder hidden state (query) *attends to* all encoder hidden states (values).
Image Captioning

CVPR 2015:  
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei  
Show and Tell: A Neural Image Caption Generator, Vinyals et al.  
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.  
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Adapted from Andrej Karpathy
Image Captioning

Recurrent Neural Network

Convolutional Neural Network
Image Captioning

Andrej Karpathy
before:
\[ h = \tanh(W_{xh} \times x + W_{hh} \times h) \]

now:
\[ h = \tanh(W_{xh} \times x + W_{hh} \times h + W_{ih} \times im) \]
Image Captioning

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

Caption generated: “straw hat”

<START> straw hat <END> token => finish.

Adapted from Andrej Karpathy
Issues with recurrent models: Linear interaction distance

- $O(\text{sequence length})$ steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is “baked in”; not necessarily the right way to think about sentences...

Adapted from John Hewitt
Issues with recurrent models: Lack of parallelizability

- Forward and backward passes have $O(\text{sequence length})$ unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can’t be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!

Numbers indicate min # of steps before a state can be computed
If not recurrence, then what?

How about attention?

• **Attention** treats each word’s representation as a **query** to access and incorporate information from a **set of values**.
  • We saw attention from the **decoder** to the **encoder**; today we’ll think about attention **within a single sentence**.
  • If **attention** gives us access to any state... maybe we can just use attention and don’t need the RNN?
• Number of unparallelizable operations not tied to sequence length.
• All words interact at every layer!

![Diagram of attention and embedding]

All words attend to all words in previous layer; most arrows here are omitted

Adapted from John Hewitt
Self-Attention

- Attention operates on **queries**, **keys**, and **values**.
  - We have some **queries** $q_1, q_2, ..., q_T$. Each query is $q_i \in \mathbb{R}^d$
  - We have some **keys** $k_1, k_2, ..., k_T$. Each key is $k_i \in \mathbb{R}^d$
  - We have some **values** $v_1, v_2, ..., v_T$. Each value is $v_i \in \mathbb{R}^d$
- In **self-attention**, the queries, keys, and values are drawn from the same source.
  - For example, if the output of the previous layer is $x_1, ..., x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)
- The (dot product) self-attention operation is as follows:

$$e_{ij} = q_i^T k_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij})} \quad \text{output } i = \sum_j \alpha_{ij} v_j$$

Compute **key-query** affinities

Compute attention weights from affinities

(softmax)

Compute outputs as weighted sum of **values**

The number of queries can differ from the number of keys and values in practice.
In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers.

Can self-attention be a drop-in replacement for recurrence?

No. It has a few issues, which we’ll go through.

First, self-attention is an operation on sets. It has no inherent notion of order.

Self-attention doesn’t know the order of its inputs.
**Barriers**

- Doesn’t have an inherent notion of order!
- No nonlinearities for deep learning magic! It’s all just weighted averages
- Need to ensure we don’t “look at the future” when predicting a sequence
  - Like in machine translation
  - Or language modeling

**Solutions**

- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self-attention output.
- Mask out the future by artificially setting attention weights to 0!
Fixing the first self-attention problem: Sequence order

- Since self-attention doesn’t build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each sequence index as a vector
  \[ p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2, \ldots, T\} \] are position vectors
- Don’t worry about what the \( p_i \) are made of yet!
- Easy to incorporate this info into our self-attention block: just add the \( p_i \) to our inputs!
- Let \( v_i', k_i', q_i' \) be our old values, keys, and queries.

\[
\begin{align*}
  v_i &= v_i' + p_i \\
  q_i &= q_i' + p_i \\
  k_i &= k_i' + p_i
\end{align*}
\]

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...
Position representation vectors through sinusoids

- **Sinusoidal position representations**: concatenate sinusoidal functions of varying periods:

  \[
  p_i = \begin{pmatrix}
  \sin(i/10000^{2\pi/d}) \\
  \cos(i/10000^{2\pi/d}) \\
  \vdots \\
  \sin(i/10000^{2\pi/d}) \\
  \cos(i/10000^{2\pi/d})
  \end{pmatrix}
  \]

- **Pros:**
  - Periodicity indicates that maybe “absolute position” isn’t as important
  - Maybe can extrapolate to longer sequences as periods restart!

- **Cons:**
  - Not learnable; also the extrapolation doesn’t really work!

Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/
Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors.

- Easy fix: add a **feed-forward network** to post-process each output vector.

\[ m_i = MLP(output_i) = W_2 \ast \text{ReLU}(W_1 \times output_i + b_1) + b_2 \]

Intuition: the FF network processes the result of attention.
To use self-attention in **decoders**, we need to ensure we can’t peek at the future.

At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)

To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

We can look at these (not greyed out) words

For encoding these words

---

John Hewitt
Necessities for a self-attention building block:

- **Self-attention:**
  - The basis of the method.

- **Position representations:**
  - Specify the sequence order, since self-attention is an unordered function of its inputs.

- **Nonlinearities:**
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.

- **Masking:**
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from “leaking” to the past.

- That’s it! But this is not the Transformer model we’ve been hearing about.
Transformer Overview

Attention is all you need. 2017. Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin


- Non-recurrent sequence-to-sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

This and related figures from paper ↑
Next, let’s look at the Transformer Encoder and Decoder Blocks

What’s left in a Transformer Encoder Block that we haven’t covered?

1. **Key-query-value attention**: How do we get the $k, q, v$ vectors from a single word embedding?
2. **Multi-headed attention**: Attend to multiple places in a single layer!
3. **Tricks to help with training!**
   1. Residual connections
   2. Layer normalization
   3. Scaling the dot product
   4. These tricks **don’t improve** what the model is able to do; they help improve the training process
The Transformer Encoder: 
**Key-Query-Value Attention**

- We saw that self-attention is when keys, queries, and values come from the same source. The Transformer does this in a particular way:
  - Let \(x_1, \ldots, x_T\) be input vectors to the Transformer encoder; \(x_i \in \mathbb{R}^d\)

- Then keys, queries, values are:
  - \(k_i = Kx_i\), where \(K \in \mathbb{R}^{d \times d}\) is the key matrix.
  - \(q_i = Qx_i\), where \(Q \in \mathbb{R}^{d \times d}\) is the query matrix.
  - \(v_i = Vx_i\), where \(V \in \mathbb{R}^{d \times d}\) is the value matrix.

- These matrices allow *different aspects* of the \(x\) vectors to be used/emphasized in each of the three roles.
The Transformer Encoder: Key-Query-Value Attention

- Let’s look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \ldots; x_T] \in \mathbb{R}^{T \times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{T \times d}$, $XQ \in \mathbb{R}^{T \times d}$, $XV \in \mathbb{R}^{T \times d}$.
  - The output is defined as $\text{output} = \text{softmax}(XQ(XK)^T) \times XV$.

First, take the query-key dot products in one matrix multiplication: $XQ(XK)$

Next, softmax, and compute the weighted average with another matrix multiplication.

All pairs of attention scores!
The Transformer Encoder: Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word $i$, self-attention “looks” where $x^T Q^T K x_j$ is high, but maybe we want to focus on different $j$ for different reasons?
- We’ll define **multiple attention “heads”** through multiple $Q,K,V$ matrices
- Let, $Q_P, K_P, V_P \in \mathbb{R}^{d \times d}$, where $h$ is the number of attention heads, and $P$ ranges from 1 to $h$.
- Each attention head performs attention independently:
  - output$_P = \text{softmax}(X Q_P K_P^T X^T) \times X V_P$, where output$_P \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - output = $Y [\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to “look” at different things, and construct value vectors differently.
The Transformer Encoder: Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word $i$, self-attention “looks” where $x^T Q^T K x_j$ is high, but maybe we want to focus on different $j$ for different reasons?
- We’ll define **multiple attention “heads”** through multiple Q,K,V matrices
- Let, $Q_P, K_P, V_P \in \mathbb{R}^{d \times d}$, where $h$ is the number of attention heads, and $P$ ranges from 1 to $h$.

---

**Single-head attention**
(just the query matrix)

$$X \quad Q = XQ$$

**Multi-head attention**
(just two heads here)

$$X \quad Q_1 Q_2 = XQ_1 XQ_2$$

*Same amount of computation as single-head self-attention!*

John Hewitt
Attention visualization in layer 5

- Words start to pay attention to other words in sensible ways
In 5th layer. Isolated attentions from just the word ‘its’ for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Christopher Manning
I kicked the ball

Who did what?

To whom?

Parallel attention heads

Ashish Vaswani
One of the benefits of self-attention over recurrence was that it’s highly parallelizable.

However, its total number of operations grows as $O(T^2d)$, where $T$ is the sequence length, and $d$ is the dimensionality.

- Think of $d$ as around 1,000.
  - So, for a single (shortish) sentence, $T \leq 30$; $T^2 \leq 900$.
  - In practice, we set a bound like $T = 512$.
  - But what if we’d like $T \geq 10,000$? For example, to work on long documents?
Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*
- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows, looking at everything**, and **random interactions**.

![Diagrams showing different attention patterns: (a) Random attention, (b) Window attention, (c) Global Attention, (d) BigBird)](image)
In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.
- This has been exceptionally effective at building strong:
  - representations of language
  - parameter initializations for strong NLP models.

[This model has learned how to represent entire sentences through pretraining]
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.

- **Encoders**
  - Gets bidirectional context – can condition on future!
  - Wait, how do we pretrain them?

- **Decoders**
  - Language models! What we’ve seen so far.
  - Nice to generate from; can’t condition on future words

- **Encoder-Decoder**
  - Good parts of decoders and encoders?
  - What’s the best way to pretrain them?

Adapted from John Hewitt
Pretraining through language modeling

[Dai and Le, 2015]

Recall the language modeling task:
- Model $p_\theta(w_t \mid w_{1:t-1})$, the probability distribution over words given their past contexts.
- There’s lots of data for this! (In English.)

Pretraining through language modeling:
- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.
Pretraining can improve NLP applications by serving as parameter initialization.

**Step 1: Pretrain (on language modeling)**
Lots of text; learn general things!

**Step 2: Finetune (on your task)**
Not many labels; adapt to the task!

... the movie was ...

John Hewitt
Capturing meaning via context: What kinds of things does pretraining learn?

There’s increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language:

• Stanford University is located in__________, California. [Trivia]
• I put____fork down on the table. [syntax]
• The woman walked across the street, checking for traffic over____shoulder. [coreference]
• I went to the ocean to see the fish, turtles, seals, and_____. [lexical semantics/topic]
• Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was____. [sentiment]
• Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the_____. [some reasoning – this is harder]
• I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21,_____[some basic arithmetic; they don’t learn the Fibonnaci sequence]
• Models also learn – and can exacerbate racism, sexism, all manner of bad biases.

Adapted from John Hewitt
Pretraining encoders: What pretraining objective to use?

So far, we’ve looked at language model pretraining. But **encoders get bidirectional context**, so we can’t do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

\[ h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T) \]
\[ y_i \sim Aw_i + b \]

Only add loss terms from words that are “masked out.” If \( x' \) is the masked version of \( x \), we’re learning \( p_\theta(x|x') \). Called **Masked LM**.

[Devlin et al., 2018]
Devlin et al., 2018 proposed the “Masked LM” objective and released the weights of a pretrained Transformer, a model they labeled BERT.

Some more details about Masked LM for BERT:

• Predict a random 15% of (sub)word tokens.
  • Replace input word with [MASK] 80% of the time
  • Replace input word with a random token 10% of the time
  • Leave input word unchanged 10% of the time (but still predict it!)
• Why? Doesn’t let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)
• Too little masking: Too expensive to train
• Too much masking: Not enough context

[Devlin et al., 2018]
BERT: Bidirectional Encoder Representations from Transformers

- Additional task: Next sentence prediction
- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Adapted from Christopher Manning
The pretraining input to BERT was two separate contiguous chunks of text:

- In addition to masked input reconstruction, BERT was trained to predict whether one chunk follows the other or is randomly sampled.
- Later work has argued this “next sentence prediction” is not necessary.

[Devlin et al., 2018, Liu et al., 2019]

Adapted from John Hewitt
BERT: Bidirectional Encoder Representations from Transformers

Details about BERT

• Two models were released:
  • BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  • BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.

• Trained on:
  • BooksCorpus (800 million words)
  • English Wikipedia (2,500 million words)

• Pretraining is expensive and impractical on a single GPU.
  • BERT was pretrained with 64 TPU chips for a total of 4 days.
  • (TPUs are special tensor operation acceleration hardware)

• Finetuning is practical and common on a single GPU
  • “Pretrain once, finetune many times.”
2018’s GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym “GPT” never showed up in the original paper; it could stand for “Generative PreTraining” or “Generative Pretrained Transformer”
How do we format inputs to our decoder for finetuning tasks?

**Natural Language Inference:** Label pairs of sentences as **entailing/contradictory/neutral**

Premise: *The man is in the doorway*  
Hypothesis: *The person is near the door*  
\[
\text{entailment}
\]

Radford et al., 2018 evaluate on natural language inference.  
Here’s roughly how the input was formatted, as a sequence of tokens for the decoder.

*[START] The man is in the doorway [DELIM] The person is near the door [EXTRACT]*

The linear classifier is applied to the representation of the [EXTRACT] token.
Increasingly convincing generations (GPT2)  
[Radford et al., 2018]

We mentioned how pretrained decoders can be used in their capacities as language models. GPT-2, a larger version of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

**Context (human-written):** In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

**GPT-2:** The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.
Let’s take a look at the assumptions we’ve made about a language’s vocabulary.

We assume a fixed vocab of tens of thousands of words, built from the training set. All novel words seen at test time are mapped to a single UNK.
Aside: Word structure and subword models

Finite vocabulary assumptions make even less sense in many languages.

- Many languages exhibit complex morphology, or word structure.
- The effect is more word types, each occurring fewer times.

Example: Swahili verbs can have hundreds of conjugations, each encoding a wide variety of information. (Tense, mood, definiteness, negation, information about the object, ++) Here’s a small fraction of the conjugations for *ambia* – to tell.

[Wikitionary]
Aside: The byte-pair encoding algorithm

Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of *parts of words (subword tokens)*.
- At training and testing time, each word is split into a sequence of known subwords.

**Byte-pair encoding** is a simple, effective strategy for defining a subword vocabulary.

1. Start with a vocabulary containing only characters and an “end-of-word” symbol.
2. Using a corpus of text, find the most common adjacent characters “a,b”; add “ab” as a subword.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

[Sennrich et al., 2016, Wu et al., 2016]
Aside: Word structure and subword models

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.

<table>
<thead>
<tr>
<th>word</th>
<th>vocab mapping</th>
<th>embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common words</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hat</td>
<td>hat</td>
<td></td>
</tr>
<tr>
<td>learn</td>
<td>learn</td>
<td></td>
</tr>
<tr>
<td>Variations</td>
<td>taaaaasty</td>
<td></td>
</tr>
<tr>
<td>misspellings</td>
<td>laern</td>
<td></td>
</tr>
<tr>
<td>novel items</td>
<td>Transformerify</td>
<td></td>
</tr>
<tr>
<td></td>
<td>taa## aaa## sty</td>
<td></td>
</tr>
<tr>
<td></td>
<td>la## ern##</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transformer## ify</td>
<td></td>
</tr>
</tbody>
</table>
GPT-3, in-context learning, very large models

So far, we’ve interacted with pretrained models in two ways:
• Sample from the distributions they define (maybe providing a prompt)
• Fine-tune them on a task we care about, and take their predictions.

Very large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.

GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters. 
**GPT-3 has 175 billion parameters.**
Very large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

**Input (prefix within a single Transformer decoder context):**

```
“ thanks -> merci
    hello -> bonjour
    mint -> menthe
    otter ->    ”
```

**Output (conditional generations):**

loutre...”
Transformers in vision

Dosovitskiy, ICLR 2021, [https://github.com/google-research/vision_transformer](https://github.com/google-research/vision_transformer)  [https://www.youtube.com/watch?v=TrdevFK_am4](https://www.youtube.com/watch?v=TrdevFK_am4)
Cross-modal transformers

Figure 1: Overview of the proposed UNITER model (best viewed in color), consisting of an Image Embedder, a Text Embedder and a multi-layer self-attention Transformer, learned through three pre-training tasks.

Chen et al., “UNITER: Learning UNiversal Image-TExt Representations”, ECCV 2020
Cross-modal transformers

Figure 1: Our ViLBERT model consists of two parallel streams for visual (green) and linguistic (purple) processing that interact through novel co-attentional transformer layers. This structure allows for variable depths for each modality and enables sparse interaction through co-attention. Dashed boxes with multiplier subscripts denote repeated blocks of layers.
Cross-modal transformers

Figure 1: The LXMERT model for learning vision-and-language cross-modality representations. ‘Self’ and ‘Cross’ are abbreviations for self-attention sub-layers and cross-attention sub-layers, respectively. ‘FF’ denotes a feed-forward sub-layer.

Tan and Bansal, “LXMERT: Learning Cross-Modality Encoder Representations from Transformers”, EMNLP 2019
Self-Supervised Learning

• Learn representations from context in raw data
• Language – predict nearby words [*already covered*]
  – Transformers, BERT
• Vision – predict pixels from other pixels
  – Predict nearby patches in an image
  – Predict order of frames in a video
  – Predict what you will see as you move
  – Predict physics

Jitendra Malik: "**Supervision** is the opium of the AI researcher"
Alyosha Efros: "The AI revolution will not be **supervised**"
Yann LeCun: “**Self-supervised** learning is the cake, **supervised** learning is the icing on the cake, **reinforcement learning** is the cherry on the cake"
Unsupervised Visual Representation Learning by Context Prediction

Carl Doersch, Alexei Efros and Abhinav Gupta

ICCV 2015
ImageNet + Deep Learning

- Image Retrieval
- Detection (RCNN)
- Segmentation (FCN)
- Depth Estimation
- ...

Do we even need semantic labels?

Context as Supervision
[Collobert & Weston 2008; Mikolov et al. 2013]

house, where the professor lived without his wife and child; or so
he said jokingly sometimes: “Here’s where I live. My house.” His
daughter often added, without resentment, for the visitor’s infor-
mation, “It started out to be for me, but it’s really his.” And she
might reach in to bring forth an inch-high table lamp with fluted
shade, or a blue dish the size of her little fingernail, marked
“Kitty” and half full of eternal milk, but she was sure to replace
these, after they had been admired, pretty near exactly where they
had been. The little house was very orderly, and just big enough
for all it contained, though to some tastes the bric-à-brac in the
parlor might seem excessive. The daughter’s preference was for the
store-bought gimmicks and appliances, the toasters and carpet
sweepers of Lilliput, but she knew that most adult visitors would
Context Prediction for Images

Semantics from a non-semantic task

Relative Position Task

8 possible locations

What is learned?

<table>
<thead>
<tr>
<th>Input</th>
<th>Ours</th>
<th>ImageNet AlexNet</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Input Image" /></td>
<td><img src="image2" alt="Ours Image" /></td>
<td><img src="image3" alt="ImageNet AlexNet Image" /></td>
</tr>
<tr>
<td><img src="image4" alt="Input Image" /></td>
<td><img src="image5" alt="Ours Image" /></td>
<td><img src="image6" alt="ImageNet AlexNet Image" /></td>
</tr>
<tr>
<td><img src="image7" alt="Input Image" /></td>
<td><img src="image8" alt="Ours Image" /></td>
<td><img src="image9" alt="ImageNet AlexNet Image" /></td>
</tr>
<tr>
<td><img src="image10" alt="Input Image" /></td>
<td><img src="image11" alt="Ours Image" /></td>
<td><img src="image12" alt="ImageNet AlexNet Image" /></td>
</tr>
</tbody>
</table>

Pre-Training for R-CNN

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

Pre-train on relative-position task, w/o labels

VOC 2007 Performance
(pretraining for R-CNN)

<table>
<thead>
<tr>
<th>Condition</th>
<th>% Average Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet Labels</td>
<td>54.2</td>
</tr>
<tr>
<td>Rel. Position</td>
<td>46.3</td>
</tr>
<tr>
<td>No Pretraining</td>
<td>40.7</td>
</tr>
</tbody>
</table>

Which will be better?

- Option 1: pretrain (unsup) on dataset B
- Option 2: pretrain (sup) on dataset A
- Test on dataset B
Shuffle and Learn: Unsupervised Learning using Temporal Order Verification

Ishan Misra, C. Lawrence Zitnick, and Martial Hebert

ECCV 2016
Fig. 1: (a) A video imposes a natural temporal structure for visual data. In many cases, one can easily verify whether frames are in the correct temporal order (shuffled or not). Such a simple sequential verification task captures important spatiotemporal signals in videos. We use this task for unsupervised pre-training of a Convolutional Neural Network (CNN). (b) Some examples of the automatically extracted positive and negative tuples used to formulate a classification task for a CNN.
Fig. 2: (a) We sample tuples of frames from high motion windows in a video. We form positive and negative tuples based on whether the three input frames are in the correct temporal order. (b) Our triplet Siamese network architecture has three parallel network stacks with shared weights up to the fc7 layer. Each stack takes a frame as input, and produces a representation at the fc7 layer. The concatenated fc7 representations are used to predict whether the input tuple is in the correct temporal order.
Table 2: Mean classification accuracies over the 3 splits of UCF101 and HMDB51 datasets. We compare different initializations and finetune them for action recognition.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Initialization</th>
<th>Mean Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCF101</td>
<td>Random</td>
<td>38.6</td>
</tr>
<tr>
<td></td>
<td>(Ours) Tuple verification</td>
<td><strong>50.2</strong></td>
</tr>
<tr>
<td>HMDB51</td>
<td>Random</td>
<td>13.3</td>
</tr>
<tr>
<td></td>
<td>UCF Supervised</td>
<td>15.2</td>
</tr>
<tr>
<td></td>
<td>(Ours) Tuple verification</td>
<td><strong>18.1</strong></td>
</tr>
</tbody>
</table>
Learning image representations tied to ego-motion

Dinesh Jayaraman and Kristen Grauman
ICCV 2015
The kitten carousel experiment
[Held & Hein, 1963]

active kitten

passive kitten

Key to perceptual development:
**self-generated motion + visual feedback**
Problem with today’s visual learning

**Status quo:** Learn from “disembodied” bag of labeled snapshots.

**Our goal:** Learn in the context of acting and moving in the world.

Our idea: **Ego-motion ↔ vision**

**Goal:** Teach computer vision system the connection: “how I move” ↔ “how my visual surroundings change”

Ego-motion $\leftrightarrow$ vision: view prediction

After moving:

Ego-motion ↔ vision for recognition

Learning this connection requires:

➢ Depth, 3D geometry
➢ Semantics
➢ Context

Can be learned without manual labels!

Our approach: unsupervised feature learning using egocentric video + motor signals

Approach idea: Ego-motion equivariance

**Invariant features**: unresponsive to some classes of transformations

\[ z(gx) \approx z(x) \]

\[ z \] is a function, \( g \) is a class of transformations, and \( x \) is the input.

**Equivariant features**: predictably responsive to some classes of transformations, through simple mappings (e.g., linear)

\[ z(gx) \approx M_g z(x) \]

\( M_g \) is the “equivariance map”

Invariance discards information; equivariance organizes it.

Approach idea: Ego-motion equivariance

**Training data**
Unlabeled video + motor signals

**Equivariant embedding**
organized by ego-motions

Pairs of frames related by similar ego-motion should be related by same feature transformation

Approach overview

Our approach: unsupervised feature learning using egocentric video + motor signals

1. Extract training frame pairs from video
2. Learn ego-motion-equivariant image features
3. Train on target recognition task in parallel

Training frame pair mining

Discovery of ego-motion clusters

\( g = \text{left turn} \)

\( g = \text{forward} \)

\( g = \text{right turn} \)

Ego-motion equivariant feature learning

**Given:**
- $\mathbf{x}_i$ and $\mathbf{x}_k$
- Class $y_k$

**Desired:** for all motions $g$ and all images $\mathbf{x}$,

$$z_\theta(g\mathbf{x}) \approx M_g z_\theta(\mathbf{x})$$

**Unsupervised training**
- $\mathbf{x}_i \xrightarrow{\theta} z_\theta(\mathbf{x}_i)$
- $\mathbf{g} \xrightarrow{\theta} z_\theta(g\mathbf{x}_i)$
- $\| M_g z_\theta(\mathbf{x}_i) - z_\theta(g\mathbf{x}_i) \|_2$

**Supervised training**
- $\mathbf{x}_k \xrightarrow{\theta} z_\theta(\mathbf{x}_k)$
- $\mathbf{M}_g \xrightarrow{\text{max loss}} \min L_C(\mathbf{x}_k, y_k)$
- $\theta, M_g$ and $W$ jointly trained

Results: Recognition

Learn from **unlabeled car video** (KITTI)

Exploit features for **static scene classification** (SUN, 397 classes)


Xiao et al, CVPR ’10

Geiger et al, IJRR ’13
Results: Recognition

Do ego-motion equivariant features improve recognition?

Up to 30% accuracy increase over state of the art!

The Curious Robot: Learning Visual Representations via Physical Interactions

Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta
ECCV 2016
Embodied representations

Physical Interaction Data

Learned Visual Representation

Fig. 2. Examples of successful (left) and unsuccessful grasps (right). We use a patch based representation: given an input patch we predict a 18-dim vector which represents whether the center location of the patch is graspable at $0^\circ$, $10^\circ$, $\ldots$ $170^\circ$. 

**Pushing**

Objects and push action pairs

![Pushing Diagram](image)

**Fig. 4.** Examples of initial state and final state images taken for the push action. The arrows demonstrate the direction and magnitude of the push action.

Fig. 6. Examples of the data collected by the poking action. On the left we show the object poked, and on the right we show force profiles as observed by the tactile sensor.
Fig. 8. Our shared convolutional architecture for four different tasks.
Classification/retrieval performance

**Fig. 10.** The first column corresponds to query image and rest show the retrieval. Note how the network learns that cups and bowls are similar (row 5).

# Classification/retrieval performance

**Table 1.** Classification accuracy on ImageNet Household, UW RGBD and Caltech-256

<table>
<thead>
<tr>
<th>Method</th>
<th>Household</th>
<th>UW RGBD</th>
<th>Caltech-256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root network with random init.</td>
<td>0.250</td>
<td>0.468</td>
<td>0.242</td>
</tr>
<tr>
<td>Root network trained on robot tasks (ours)</td>
<td>0.354</td>
<td>0.693</td>
<td>0.317</td>
</tr>
<tr>
<td>AlexNet trained on ImageNet</td>
<td>0.625</td>
<td>0.820</td>
<td>0.656</td>
</tr>
</tbody>
</table>

**Table 2.** Image Retrieval with Recall@k metric

<table>
<thead>
<tr>
<th>Method</th>
<th>Instance level</th>
<th>Category level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k=1</td>
<td>k=5</td>
</tr>
<tr>
<td>Random Network</td>
<td>0.062</td>
<td>0.219</td>
</tr>
<tr>
<td>Our Network</td>
<td>0.720</td>
<td>0.831</td>
</tr>
<tr>
<td>AlexNet</td>
<td>0.686</td>
<td>0.857</td>
</tr>
</tbody>
</table>

SimCLR - A Simple Framework for Contrastive Learning of Visual Representations

SimCLR Framework

Figure 1. **Our MAE architecture.** During pre-training, a large random subset of image patches (e.g., 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images (full sets of patches) for recognition tasks.
Project Pitches