Deep Learning

Ronald Parr
CompSci 370

Late 1990’s: Neural Networks Hit the Wall

• Recall that a 3 layer network can approximate any function arbitrarily closely (caveat: might require many, many hidden nodes)

• Q: Why not use big networks for hard problems?
• A: It didn’t work in practice!
  – Vanishing gradients
  – Not enough training data (overfitting)
  – Not enough training time (computers too slow to handle huge data sets, even if they were available)
Why Deep?

- Deep learning is a family of techniques for building and training large neural networks

- Why deep and not wide?
  - Deep sounds better than wide 😊
  - While wide is always possible, deep may require fewer nodes to achieve the same result
  - May be easier to structure with human intuition: think about layers of computation vs. one flat, wide computation

Examples of Deep Learning Today

- Object/face recognition in your phone, your browser, autonomous vehicles, etc.
- Natural language processing (speech to text, parsing, information extraction, machine translation)
- Product recommendations (Netflix, Amazon)
- Fraud detection
- Medical imaging
- Image enhancement or restoration (e.g., Adobe Super resolution) [https://blog.adobe.com/en/publish/2021/03/10/from-the-acr-team-super-resolution.html](https://blog.adobe.com/en/publish/2021/03/10/from-the-acr-team-super-resolution.html)
- Quick Draw: [https://quickdraw.withgoogle.com](https://quickdraw.withgoogle.com)
- Large language models, e.g., chatGPT
Vanishing Gradients

• Recall backprop derivation:

\[ \delta_j = \sum_i \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial a_j} = h'(a_j) \sum w_{kj} \delta_i \]

• Activation functions often between -1 and +1
• In huge networks, any individual weight often has only a tiny contribution to the output
• The further you get from the output layer, the smaller the gradient gets

Related Problem: Saturation

• Sigmoid gradient goes to 0 at tails
• Extreme values (saturation) anywhere along backprop path causes gradient to vanish
Summary of the Challenges

- Not enough training data in the 90’s to justify the complexity of big networks (recall dangers of overfitting)
- Slow to train big networks
- Vanishing gradients, saturation

Summary of Changes

- Massive data available
- Massive computation available
- Faster training methods
- Different training methods
- Different network structures
- Different activation functions
Estimating the Gradient Efficiently

• Recall: Backpropagation is gradient descent
• Computing exact gradient of the loss function requires iterating over all training samples

• Thought experiment: What if you randomly sample one (or more) data point(s) and compute the gradient?
  – Called online or stochastic gradient descent (SGD)
  – Expected value of sampled gradient = true value of gradient
  – Sampled gradient ≠ true gradient + noise
  – As sample size increases, noise decreases, sampled gradient → true
  – Practical idea: For massive data sets, estimate gradient using sampled training points to trade off computation vs. accuracy in gradient calculation
  – Possible pitfalls:
    • What is the right sampling strategy?
    • Does the noise prevent convergence or lead to slower convergence?

Batch/Minibatch Methods

• Find a sweet spot by estimating the gradient using a subset of the samples
• Randomly sample subsets of the training data and sum gradient computations over all samples in the subset
• Take advantage of parallel architectures (multicore/GPU)

• Still requires careful selection of step size and step size adjustment schedule – art vs. science
Other Tricks for Speeding Things Up

• Second order methods, e.g., Newton’s method – may be computationally intensive in high dimensions

• Momentum: Use a combination of previous gradients to smooth out noise in the gradient estimate

• Line search: (Binary) search in gradient direction to find biggest worthwhile step size

• Current best method in pytorch is likely Adam, which is a variant of SGD that combines various acceleration tricks

Convolutional Neural Networks (CNNs)

• Championed by LeCun (1998)

• Originally used for handwriting recognition

• Now used in state of the art systems in many computer vision applications

• Well-suited to data with a grid-like structure
Convolutions

- What is a convolution?
- Way to combine two functions, e.g., \( x \) and \( w \):

\[
s(t) = \int x(a)w(t - a)\,da
\]

- Discrete version

\[
s(t) = \sum x(a)w(t - a)
\]

Example: Suppose \( s(t) \) is a decaying average of values of \( x \) around \( t \), with \( w \) decreasing as \( a \) gets further from \( t \)

Convolution on Grid Example

Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called “valid” convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Figure 9.1 from Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville
Convolutions on Grids

• For image \( I \)
• Convolution “kernel” \( K \):

\[
S(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n)
\]

Examples:
A convolution can blur/smooth/noise-filter an image by averaging neighboring pixels.
A convolution can also serve as an edge detector
https://en.wikipedia.org/wiki/Kernel_(image_processing)

Application to Images & Nets

• Images have huge input space: 1000x1000=1M
• Fully connected layers = huge number of weights, slow training

• Convolutional layers reduce connectivity by connecting only an \( m \times n \) window around each pixel
• Can use weight sharing to learn a common set of weights so that same convolution is applied everywhere (or in multiple places)
Advantages of Weight Sharing

- Reduces # of weights that must be learned
  - Speeds up learning
  - Fewer local optima
  - Less risk of overfitting
- Enforces uniformity in what is learned
- Enforces translation invariance — learns the same thing for all positions in the image

Additional Stages & Different Activation Functions

- Convolutional stages (may) feed to detector stages
- Detectors are nonlinear, e.g., ReLU
- Detectors feed to pool stages
- Pooling stages summarize upstream nodes, e.g., average, 2-norm, max
Additional Stages & Different Activation Functions

- Convolutional stages (may) feed to intermediate stages

- Detectors stages are nonlinear, e.g., ReLU

- Pooling stages summarize upstream nodes, e.g., average (shrinking image), max (thresholding)

Example Convolutional Network

From, Convolutional Networks for Images, Speech, and Time-Series, LeCun & Bengio

N.B.: Subsampling = averaging

Weight sharing results in 2600 weights shared over 100,000 connections.
Why This Works

• ConvNets can use weight sharing to reduce the number of parameters learned – mitigates problems with big networks

• Combination of convolutions with shared weights and subsampling can be interpreted as learning position and scale invariant features

• Final layers combine feature to learn the target function

• Can be viewed as doing **simultaneous feature discovery and classification**

ConvNets in Practice

• Work surprisingly well in many examples, even those that aren’t images

• Number of convolutional layers, form of pooling and detecting units may be application specific – art & science here
Other Tricks

• Convnets and ReLUs tend to help with vanishing gradient problem, but don’t eliminate it
• Residual nets introduce connections across layers, which tends to mitigate the vanishing gradient problem
• Techniques such as image perturbation and drop out reduce overfitting and produce more robust solutions

Putting It all Together

• Why is deep learning succeeding now when neural nets lost momentum in the 90’s?
• New architectures (e.g., ConvNets) are better suited to (some) learning tasks, reduce # of weights
• Smarter algorithms for smarter gradient descent
• Massive amounts of data make overfitting less of a concern (but still always a concern)
• Massing amounts of computation make handling massive amounts of data possible
• Large and growing bag of tricks to mitigating overfitting, vanishing gradient issues
Limitations

- Deep learning results are not easily human-interpretable
- Computationally intensive
- Combination of art, science, rules of thumb
- Can be tricked:
  - “Intriguing properties of neural networks”, Szegedy et al. [2013]

...and it gets worse!

- From “Universal adversarial perturbations”, Moosavi-Dezfooli et al., 2017
- A single, barely perceptible noise pattern can cause misclassifications in all categories!
Concerns About Reproducibility

• Obvious sources of randomness:
  – Random initial weights
  – Stochastic gradient descent

• Hidden sources of randomness:
  – Rounding in single precision GPU calculations
  – Non-determinism in asynchronous parallel computations

• Other hidden factors:
  – Different GPU hardware
  – Different software libraries
  – Lots of tunable parameters in network training

• Human factors in reporting:
  – Cherry picking of results

Beyond Classification

• Deep networks (and other techniques) can be used for unsupervised learning

• Example: Autoencoder tries to compress inputs to a lower dimensional representation
Recurrent Networks

• Recurrent networks feed (part of) the output of the network back to the input

• Why?
  – Can learn (hidden) state, e.g., in a hidden Markov model
  – Useful for parsing language
  – Can learn a program

• LSTM: Variation on RNN that handles long term memories better

Deep Learning Conclusions

• Deep learning provides state-of-the-art results on many problems of practical significance

• We get impressive results but we don’t always understand why or whether we really need all of the data and computation used

• Hard to explain results and hard to guard against adversarial special cases

• Not clear how logic, high level reasoning could be incorporated
  – Neurosymbolic methods (active research area) tries to take steps in this direction
  – LLMs like chatGPT give the illusion of logical reasoning but don’t explicitly reason logically

• Not clear how to incorporate prior knowledge in a principled way
I have 7 measuring cups spread out on a table. Five of the measuring cups have maximum capacity of 250 ml. One has a maximum capacity of 500 ml, and the last one has a maximum capacity of 1 liter. I fill the five 250 ml cups with liquid water. I then empty these into the 500 ml cup. Finally, I empty this into the 1 liter cup. How full is the 1 liter cup?

The five 250 ml cups hold a total of 5 x 250 ml = 1250 ml of water.

When you pour this into the 500 ml cup, the 500 ml cup will be completely filled, and you will have 1250 ml - 500 ml = 750 ml of water remaining.

When you pour this 750 ml of water into the 1 liter cup, the cup will be filled to a total of 750 ml + 500 ml = 1250 ml.

So the 1 liter cup is filled to 1250 ml / 1000 ml = 1.25 or 125% of its maximum capacity.