Deep Learning Foundations

Kris Sankaran

Nepal Winter School in AI

December 25, 2018

Outline

Introduction

Representation Learning

Optimization

Debugging Deep Learning

Deep Learning and Poetry

- Algorithms and words are only as remarkable as human creativity
  - Discover patterns and make connections unseen by others
- Knowing SGD is like knowing meter
- I want to teach you the form, so you can write the poetry

Cheerios

BY BILLY COLLINS

One bright morning in a restaurant in Chicago
as I waited for my eggs and toast,
I opened the *Tribune* only to discover
that I was the same age as Cheerios.

Learning Objectives

- Understand goals of deep learning, especially representation learning
- Logistic Regression to Backpropagation: Learn the modular-components view of deep learning
- Learn tricks for doing (and debugging) deep learning
- How to study deep learning algorithms: toy examples, drawing pictures, ...

Ask Questions

Truth emerges more readily from error than from confusion.

Francis Bacon (as cited in [3])

Flipping Coins

- Imagine n flips of a coin with probability π of coming up heads.
- Loglikelihood function (independent bernoulli trials)

$$\log p\left(y|\pi\right) = \log\left[\prod_{i=1}^{n} \pi^{\mathbb{I}(y_i=1)}\left(1-\pi\right)^{\mathbb{I}(y_i=0)}\right]$$

$$= \sum_{i=1}^{n} \mathbb{I}\left(y_i = 1\right)\log\pi + \mathbb{I}\left(y_i = 0\right)\log\left(1-\pi\right)$$

where we use the shorthand $y = (y_1, \ldots, y_n)$.

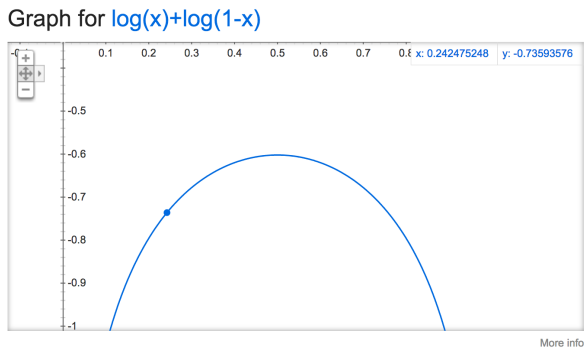


Figure: Logliklihoods over π when we see one head and one tail. Seems most likely that $\pi \approx 0.5$.

Flipping Coins

- Imagine n flips of a coin with probability π of coming up heads.
- Loglikelihood function (independent bernoulli trials)

$$\log p\left(y|\pi\right) = \log\left[\prod_{i=1}^{n} \pi^{\mathbb{I}(y_i=1)} \left(1-\pi\right)^{\mathbb{I}(y_i=0)}\right]$$

$$= \sum_{i=1}^{n} \mathbb{I}\left(y_i = 1\right) \log \pi + \mathbb{I}\left(y_i = 0\right) \log\left(1-\pi\right)$$

where we use the shorthand $y = (y_1, \ldots, y_n)$.

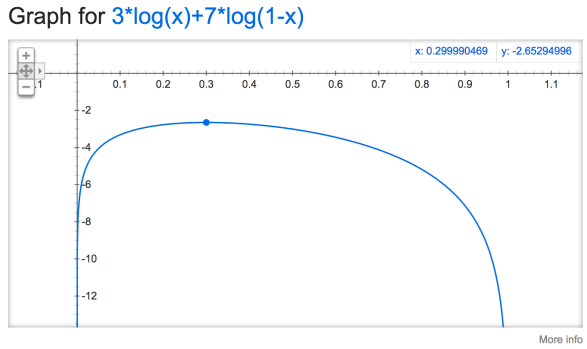


Figure: Logliklihoods over π when we see 3 heads and 7 tails. Seems most likely that $\pi \approx 0.3$.

Flipping Coins

- Imagine n flips of a coin with probability π of coming up heads.
- Loglikelihood function (independent bernoulli trials)

$$\log p\left(y|\pi\right) = \log\left[\prod_{i=1}^{n} \pi^{\mathbb{I}(y_i=1)}\left(1-\pi\right)^{y_i=0}\right]$$

$$= \sum_{i=1}^{n} \mathbb{I}\left(y_i = 1\right)\log\pi + \mathbb{I}\left(y_i = 0\right)\log\left(1-\pi\right)$$

where we use the shorthand $y = (y_1, \ldots, y_n)$.

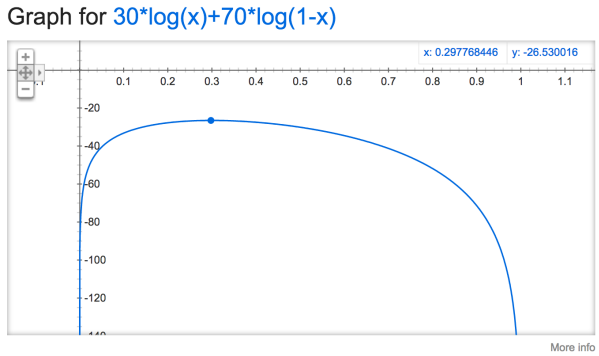


Figure: Logliklihoods over π when we see 30 heads and 70 tails. Again seems most likely that $\pi \approx 0.3$, but everything is scaled by $10\times$.

Using Input Data

- What if the probability depended on input data?
- Depending on some input x_i, we have probability $\pi(x_i)$ of heads

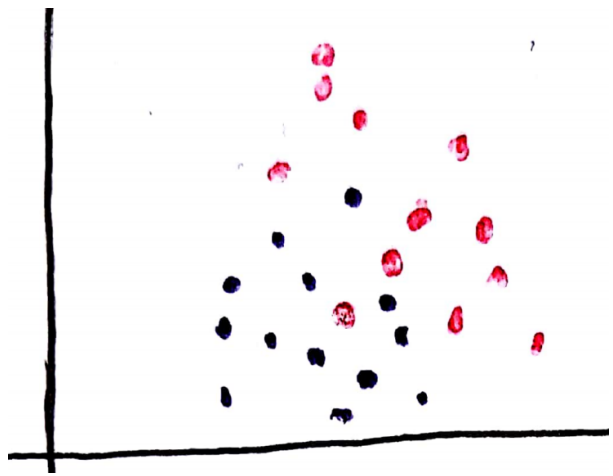


Figure: Example of a classification problem when x_i are two dimensional, heads are green, and tails are red.

Using Input Data

- What if the probability depended on input data?
- Depending on some input x_i, we have probability $\pi(x_i)$ of heads

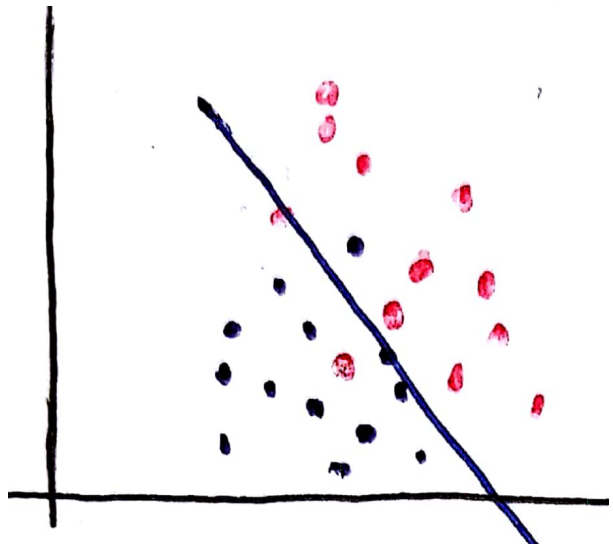


Figure: Example of a classification problem when x_i are two dimensional, heads are green, and tails are red.

Using Input Data

- What if the probability depended on input data?
- Depending on some input x_i, we have probability $\pi(x_i)$ of heads

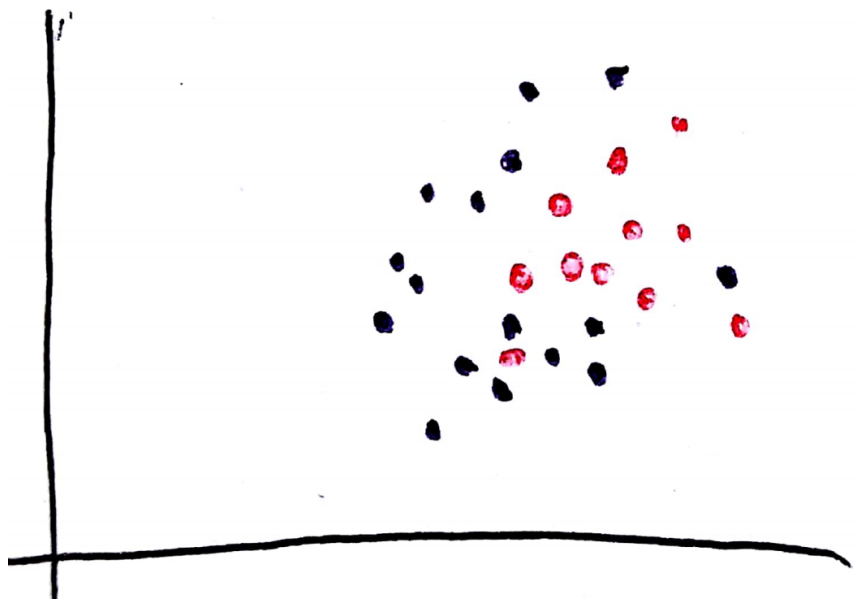


Figure: Example with a nonlinear boundary.

Using Input Data

- What if the probability depended on input data?
- Depending on some input x_i, we have probability $\pi(x_i)$ of heads

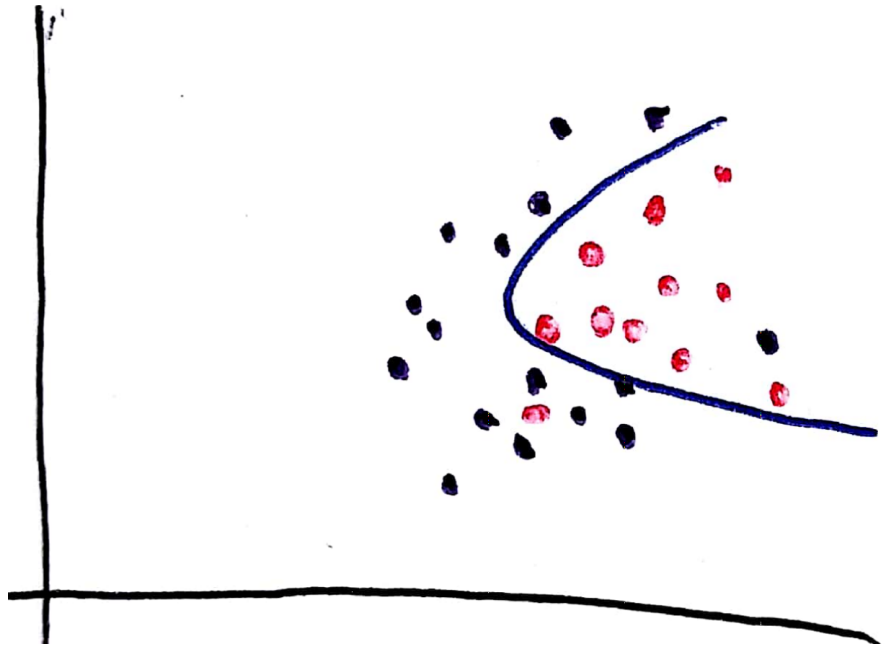


Figure: Example with a nonlinear boundary.

Using Input Data

- Can adapt previous loglikelihood to this setting

$$\log p\left(y|x\right) = \sum_{i=1}^{n} \mathbb{I}\left(y_i = 1\right) \log \pi\left(x\right) + \mathbb{I}\left(y_i = 0\right) \log\left(1 - \pi\left(x\right)\right)$$

- Need to learn $\pi\left(x\right)$, just like we learned π before

Sigmoid Function

- Consider the collection of functions (one for each θ)

$$\pi_\theta(x) = \frac{1}{1+\exp(-\theta^T x)} \overset{\text{defined}}{=} \sigma(\theta^T x)$$

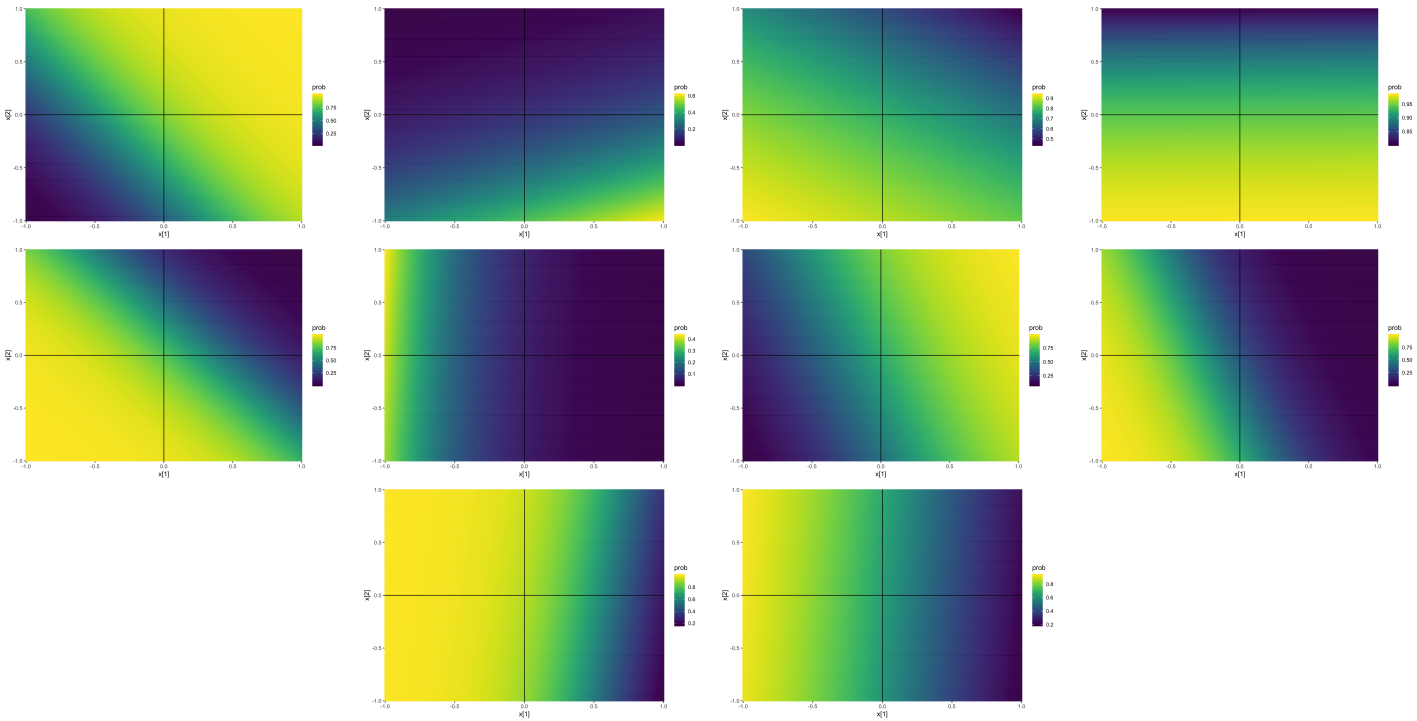


Figure: Example sigmoid functions π_θ for random draws of θ, when we assume x has an intercept term. Notice that there are no nonlinear boundaries...

Logistic Regression

- Logistic Regression $\rightarrow$ Finding good θ's for this likelihood (i.e., the ones that maximize it)

$$\log p_\theta\left(y|x\right) = \sum_{i=1}^{n} \mathbb{I}\left(y_i = 1\right) \log \pi_\theta\left(x\right) + \mathbb{I}\left(y_i = 0\right) \log\left(1 - \pi_\theta\left(x\right)\right)$$

- These are the directions like those in the previous figure that do a good job separating heads from tails

Optimization

- Need to find some θ's that maximize $\log p_\theta\left(y|s\right)$
- Use gradient descent on $-\log p_\theta\left(y|x\right)$
- How to get the gradients?
- Approach 1: Courageously differentiate,

$$
\begin{aligned}
&\frac{\partial}{\partial\theta}\log p_\theta\left(y|x\right)\\
=&\frac{\partial}{\partial\theta}\left[\sum_{i=1}^{n} y_i\log\sigma\left(\theta^T x_i\right)+\left(1-y_i\right)\log\left(1-\sigma\left(\theta^T x_i\right)\right)\right]\\
=&\frac{\partial}{\partial\theta}\left[\sum_{i=1}^{n} y_i\log\left(\frac{1}{1+\exp\left(-\theta^T x_i\right)}\right)+\left(1-y_i\right)\log\left(\frac{\exp\left(-\theta^T x_i\right)}{1+\exp\left(-\theta^T x_i\right)}\right)\right]
\end{aligned}
$$

Optimization

- Approach 2: Chain rule magic!

$$\begin{aligned}\log p_\theta\left(y_i|x_i\right) &= y_i \log\left(\theta^T x_i\right) + \left(1-y_i\right)\log\left(1-\sigma\left(\theta^T x_i\right)\right) \\ &= \ell\left(\sigma\left(\theta^T x_i\right)\right)\end{aligned}$$

- This is the composition of three relatively simple functions

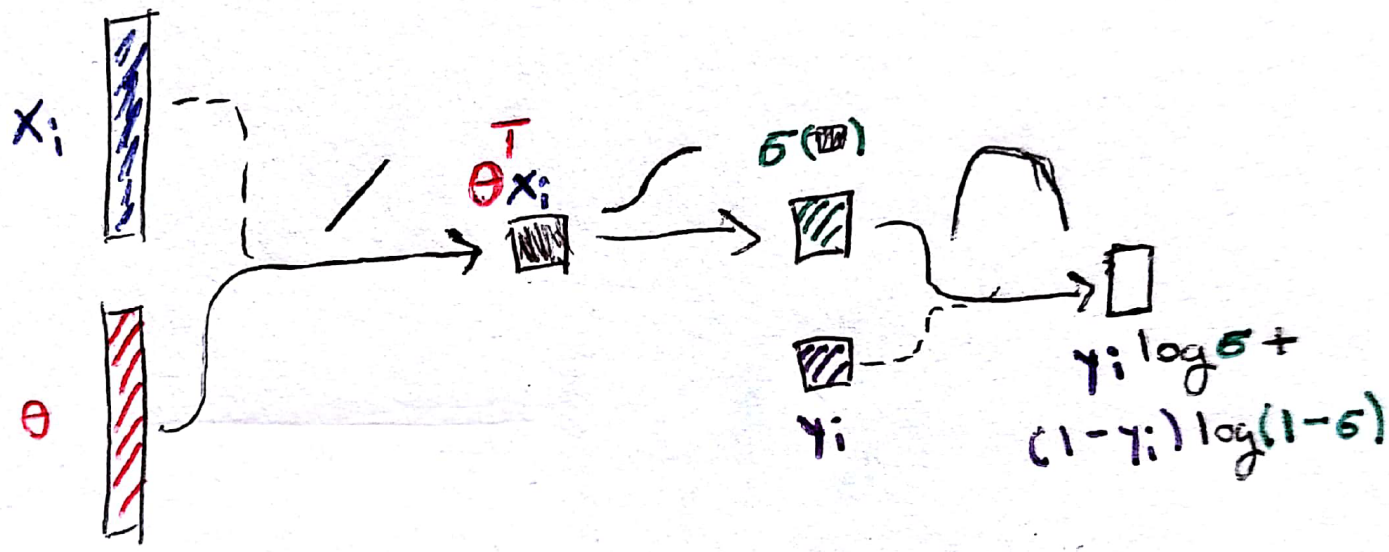


Figure: The loss function is a series of compositions of relatively simple functions. This is an example of a flow graph, which we will revisit in more generality soon.

Review: The Chain Rule

- Derivative of composition $\rightarrow$ multiplication of derivatives

$$D\left(f \circ g\right)(x) = Df\left(g\left(x\right)\right) Dg\left(x\right)$$

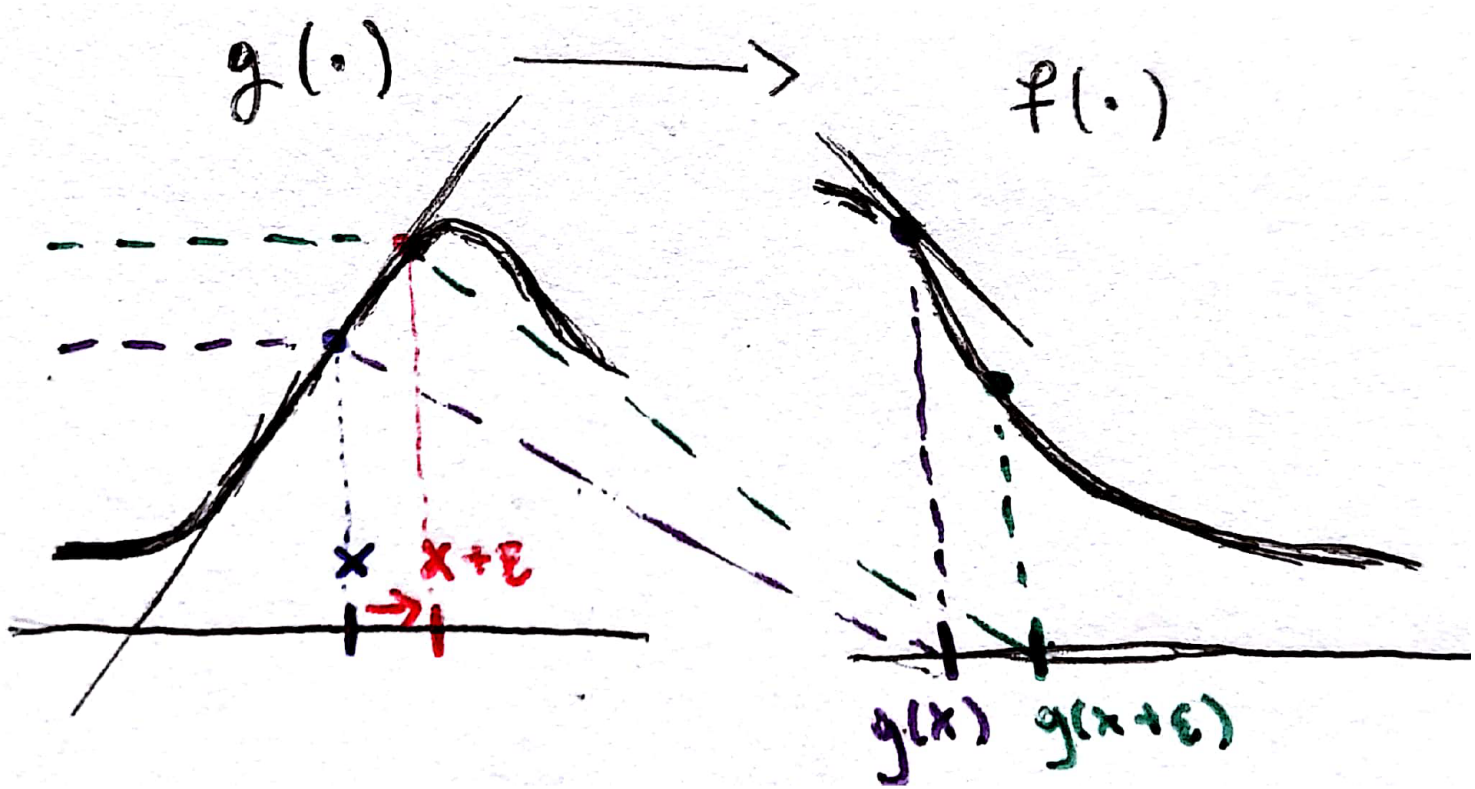


Figure: The chain rule tells us how a small change in x affects the downstream output of a composition of functions, in terms of the changes it causes within each intermediate function.

Review: The Chain Rule

- Derivative of composition $\rightarrow$ multiplication of derivatives

$$D\left(f^K \circ f^{K-1} \cdots \circ f^1\right)(x) = \left[\prod_{k=2}^{K} Df^k\left(h_{k-1}\right)\right] Df^1(x)$$

where $h_k = f^k\left(h_{k-1}\right)$ is output of previous level (and $h_0 = x$)

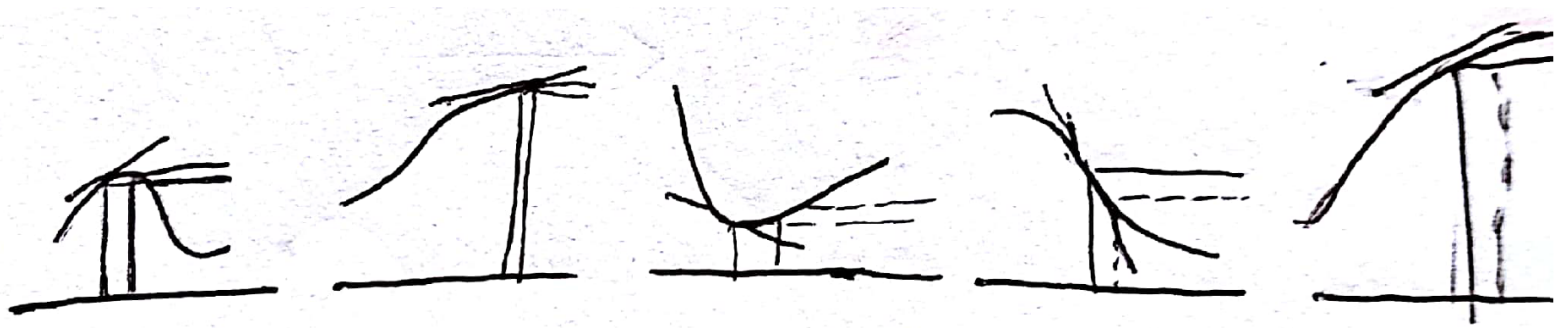


Figure: The chain rule tells us how a small change in x affects the downstream output of a composition of functions, in terms of the changes it causes within each intermediate function.

Chain Rule for Logistic Regression

- We can now find the derivative of the logistic regression loss much more elegantly
- Remember that it has these intermediate functions
  1. $\ell(\sigma) = y_i \log \sigma + (1 - y_i) \log(1 - \sigma)$
  2. $\sigma(z) = \frac{1}{1+\exp(-z)}$
  3. $\theta^T x_i$
- Their derivatives are (check this!)
  1. $\frac{\partial}{\partial \sigma} \ell(\sigma) = \frac{y_i}{\sigma} - \frac{1-y_i}{1-\sigma}$
  2. $\frac{\partial}{\partial z} \sigma(z) = -\sigma(z)(1 - \sigma(z))$
  3. $\nabla_\theta \theta^T x_i = x_i$

Chain Rule for Logistic Regression

- Multiply the derivatives and substitute input from the flow graph
- Gives the gradient,

$$\begin{aligned}
&\left(\frac{y_i}{\sigma\left(\theta^T x_i\right)} - \frac{1-y_i}{1-\sigma\left(\theta^T x_i\right)}\right)\sigma\left(\theta^T x_i\right)\left(1-\sigma\left(\theta^T x_i\right)\right)x_i \\
&= \left[y_i\left(1-\sigma\left(\theta^T x_i\right)\right) - \left(1-y_i\right)\sigma\left(\theta^T x_i\right)\right]x_i \\
&= \left(y_i - \pi_\theta\left(x_i\right)\right)x_i
\end{aligned}$$

  recalling that by definition $\pi_\theta\left(x_i\right) = \sigma\left(\theta^T x_i\right)$

- Nice interpretation: change θ a lot when y_i is far from the prediction probability $\pi_\theta\left(x_i\right)$

Outline

Introduction

Representation Learning

Optimization

Debugging Deep Learning

Finding Meaningful Features

- Want to estimate $p(y_i = \text{yak}|x_i)$
- Could use logistic regression if the input x_i were meaningful

has_mountains = 1
has_river = 0
furry_animal = 1
horned_animal = 1
scary_animal = 0
.....

Figure: It would be nice if our images came with qualitative annotation about what was in them.

Finding Meaningful Features

- Want to estimate $p\left(y_i = \text{yak} | x_i\right)$
- If our x_i's are unstructured, this becomes much more difficult
- Logistic regression will fail

pixel_1 = 132
pixel_2 = 140
pixel_3 = 141
pixel_4 = 139
pixel_5 = 130
.....

Figure: In reality, we can't manually provide all this annotation.

Finding Meaningful Features

- If our x_i's are unstructured, this becomes much more difficult
- Logistic regression will fail

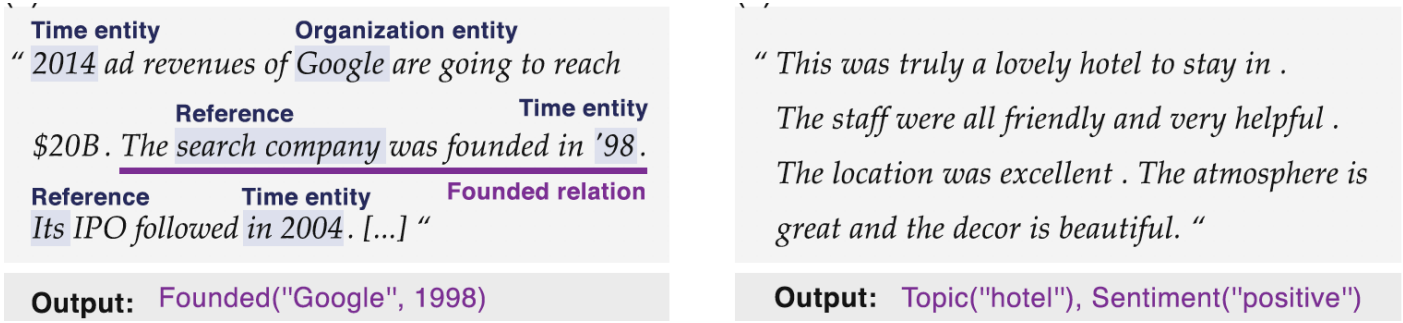


Figure: Similarly, extracting meaningful structure from raw text would be useful in a variety of downstream tasks.

Our Goal

- Try to automatically learn meaningful features
- Start with simple features at low layers
- Increase feature complexity in a compositional way

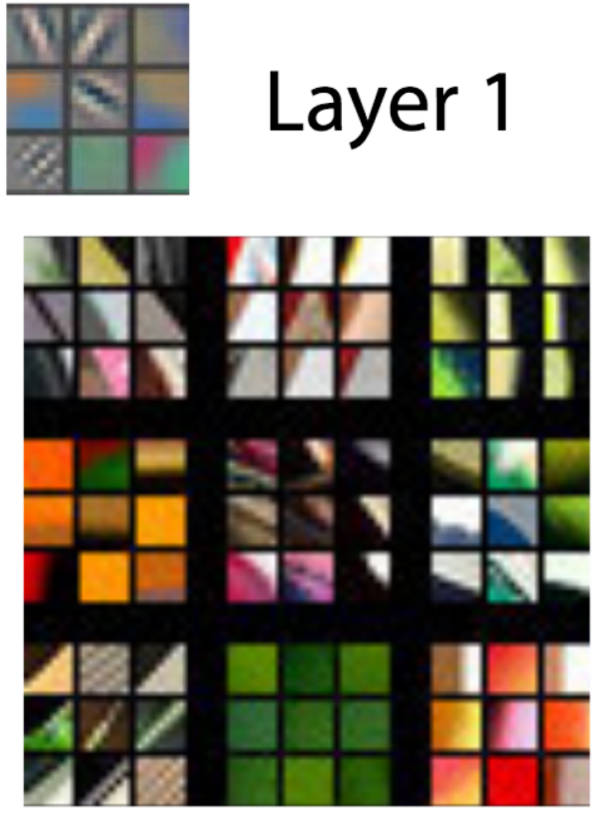


Figure: Early layers might create features corresponding to edges of different orientations.

Our Goal

- Try to automatically learn meaningful features
- Start with simple features at low layers
- Increase feature complexity in a compositional way

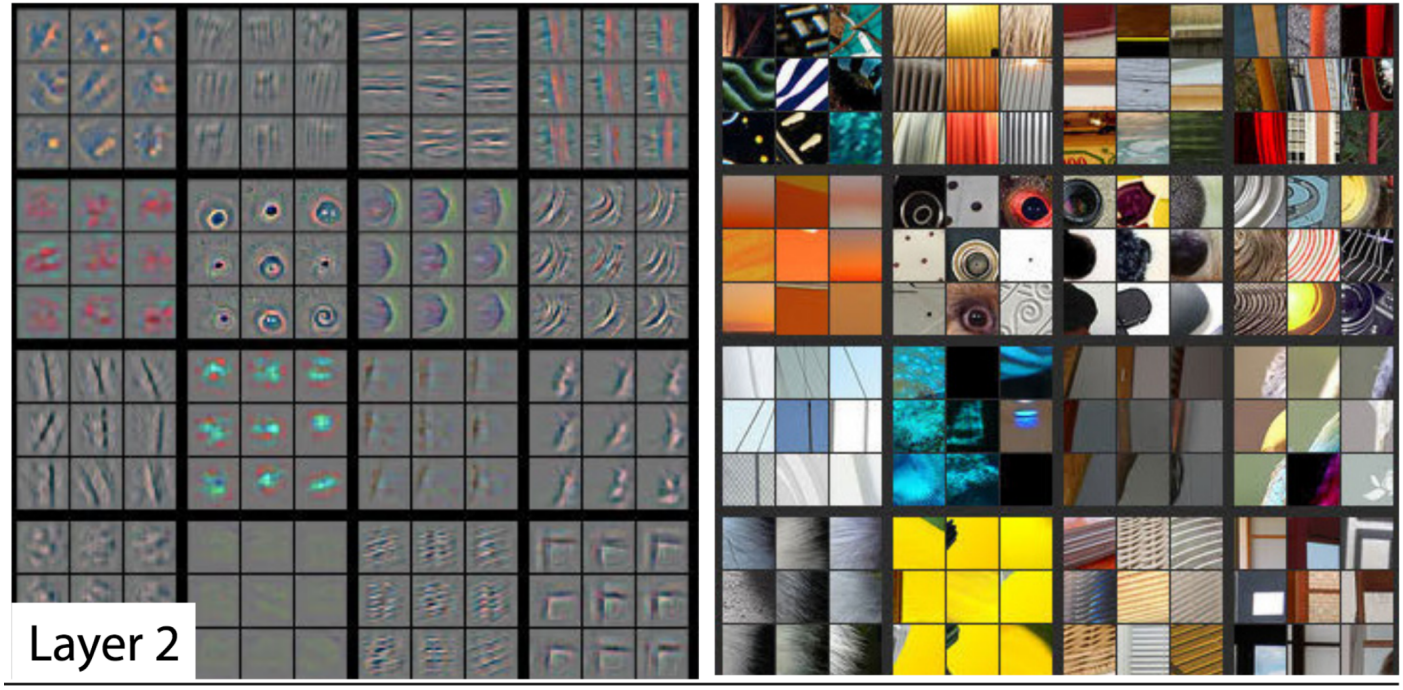


Figure: Features increase in complexity as you get deeper in the network.

Our Goal

- Try to automatically learn meaningful features
- Start with simple features at low layers
- Increase feature complexity in a compositional way

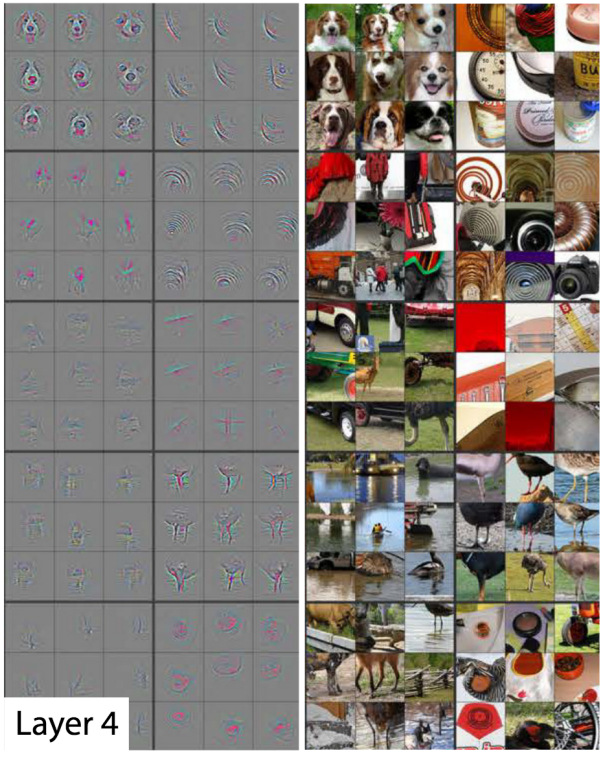


Figure: Eventually we will have meaningful high-level features.

Our Goal

- Try to automatically learn meaningful features
- Start with simple features at low layers
- Increase feature complexity in a compositional way

Figure: Similar kinds of complex features can emerge in text applications too.

Mixing & Stacking Logistic Regressions

- Simple combinations of simple functions quickly become complex
- These complex compositions could capture meaningful features

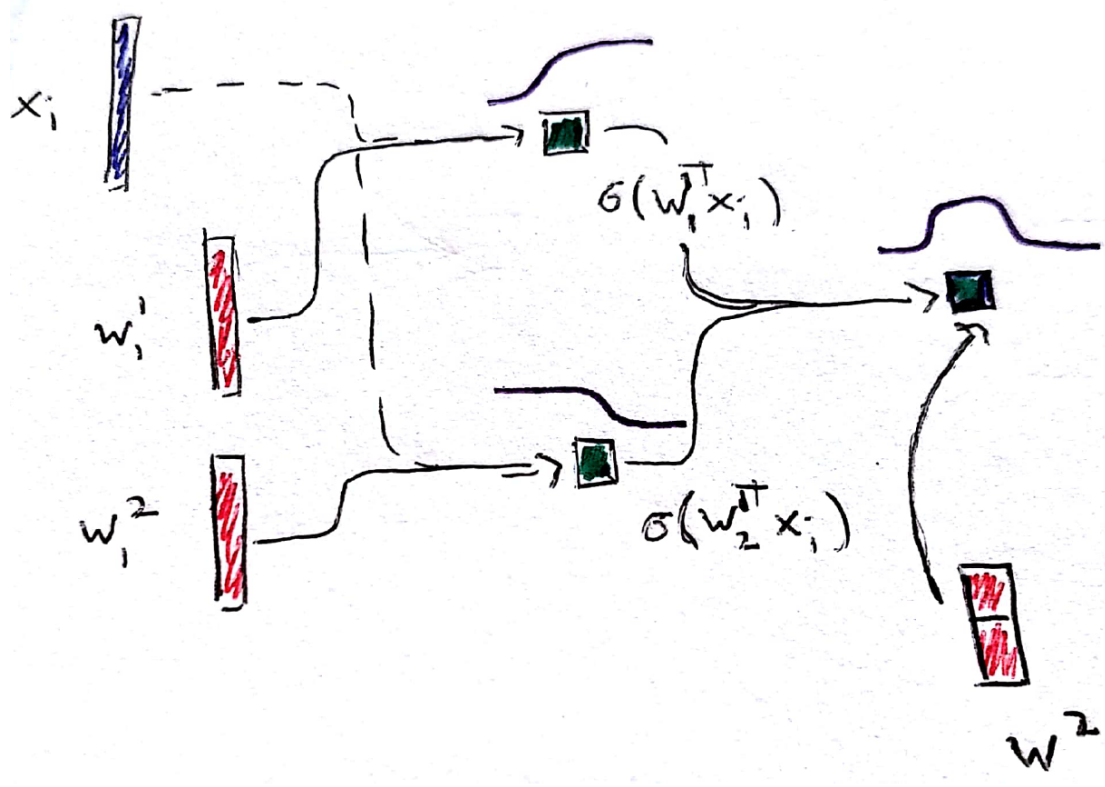
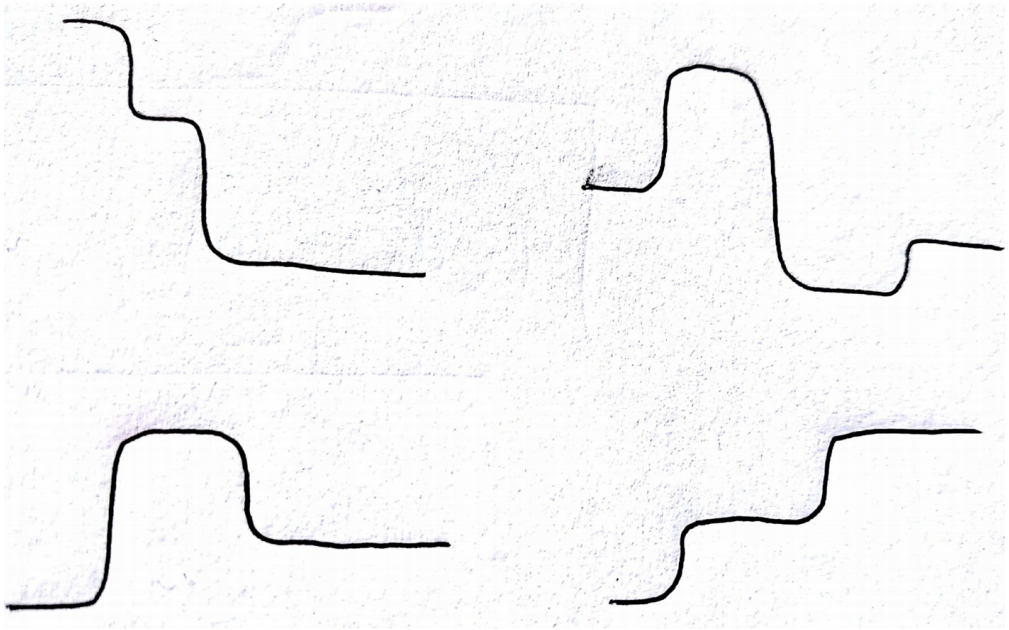


Figure: Simply mixing two logistic regressions, we can represent nonmonotonic functions.

Discussion

- Which of the following functions *cannot* be represented by the mixture of two logistic regressions?
- Can you think of other functions that cannot be learned this way?



Mixing & Stacking Logistic Regressions

- Simple combinations of simple functions can quickly become complex
- These complex compositions could capture meaningful features

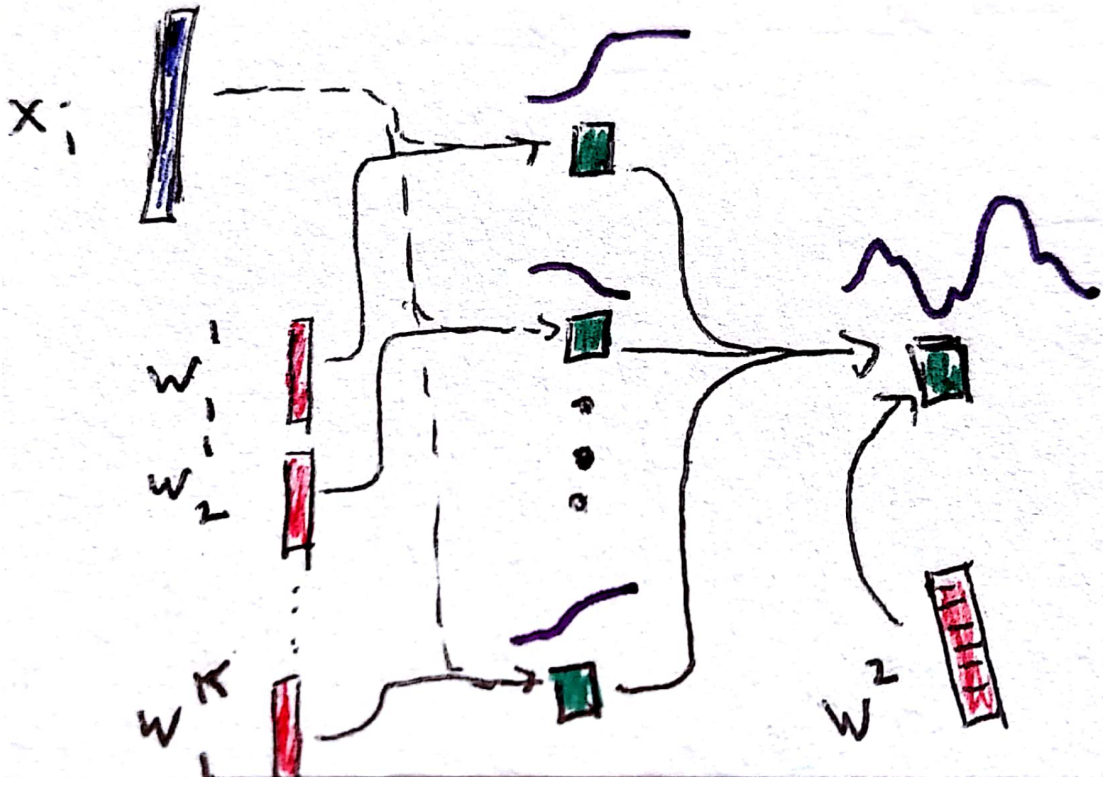


Figure: Mixing more sigmoids, we can represent even more complicated functions.

Mixing & Stacking Logistic Regressions

- New notation: Representation at layer k,

$$h_k = f^k\left(h_{k-1}; W_k\right) \overset{\text{defined}}{=} \sigma\left(W_k h_{k-1}\right)$$

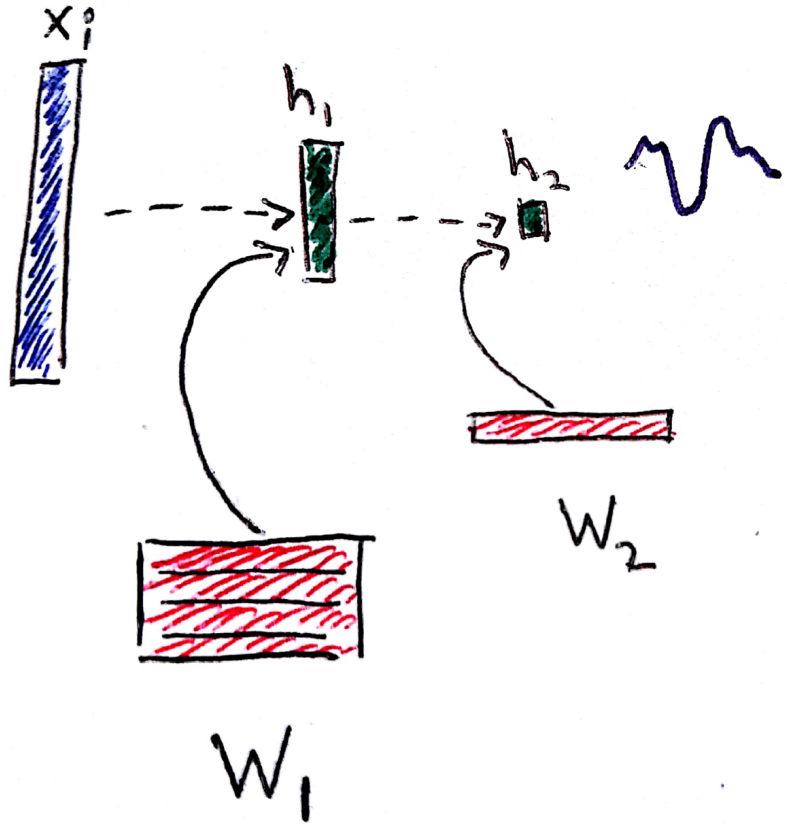


Figure: Mixing more sigmoids, we can represent even more complicated functions.

Mixing & Stacking Logistic Regressions

- New notation: Representation at layer k,

$$h_k = f^k\left(h_{k-1}; W_k\right) \overset{\text{defined}}{=} \sigma\left(W_k h_{k-1}\right)$$

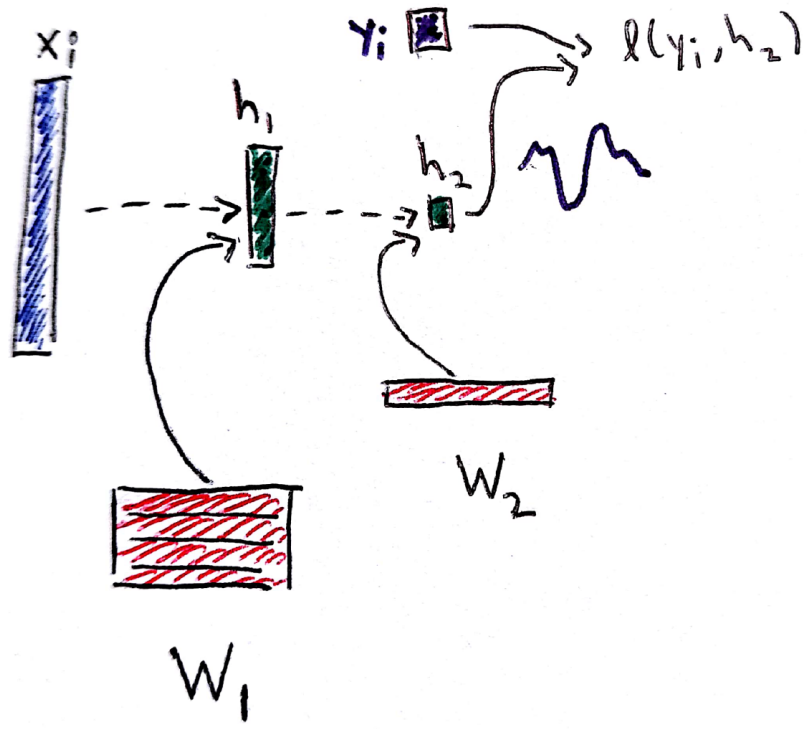


Figure: Mixing more sigmoids, we can represent even more complicated functions.

Mixing & Stacking Logistic Regressions

- Simple combinations of simple functions can quickly become complex
- These complex compositions could capture meaningful features

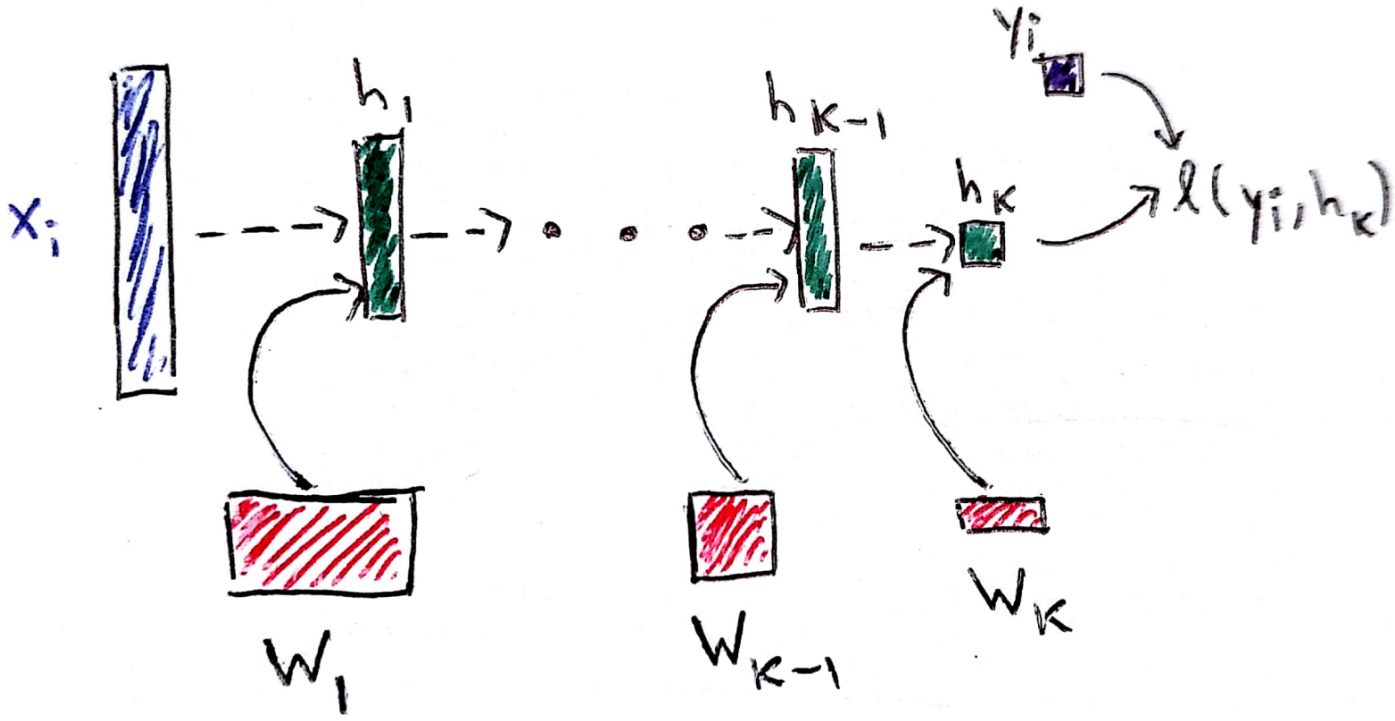


Figure: The basic idea of deep learning is to compose nonlinearities in a way that encourages discovery of complex intermediate representations h_k.

Aside: Activations

- For nonlinearity, we used sigmoid
- Many types of nonlinearities can be used instead
- The basic idea of mixing together simple functions remains the same
- Most common these days are Rectified Lienar Unites (ReLUs)
  - They are easier to optimize (alleviate vanishing gradient problem)

Figure: Some example nonlinearities from `https://en.wikipedia.org/wiki/Activation_function`.

Outline

Introduction

Representation Learning

Optimization

Debugging Deep Learning

Optimization: Stochastic Gradient Descent

- Optimize by moving in a direction expected to decrease loss
- SGD: Estimated direction of gradient, after looking at small batch of data
- These days, other methods used too – Adam, AdaGrad, AdaDelta, ...

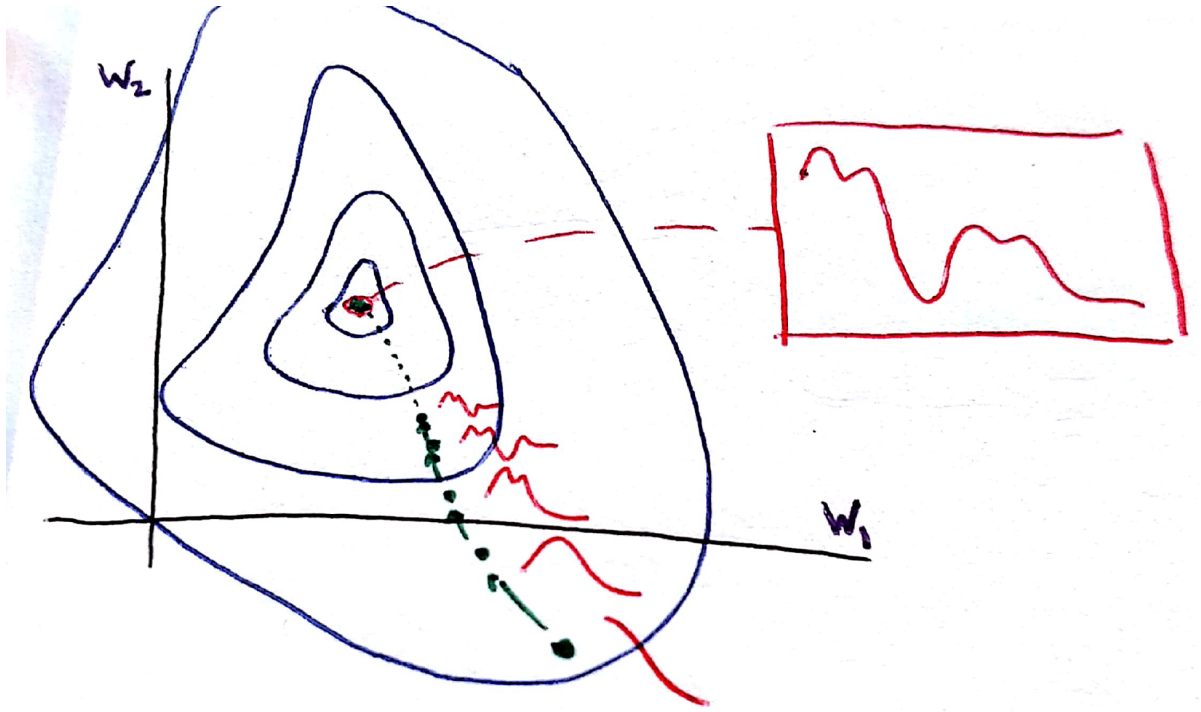


Figure: Each optimization step modifies model parameters to make the fitted function closer ot the true (local) minimizer of the loss.

Backpropagation via Flow Graphs

- For each iteration of SGD, we need to estimate the gradient of the loss with respect to the parameters
- When you perturb W_k a little, it changes all the downstream $h_k, h_{k+1}, \ldots, h_K$ up to the prediction, which uses that final representation
- There is a lot of composition going on here... courage alone won't help you take these derivatives
- Chain rule saves the day again!

Chain Rule Revisited

- Chain rule for vector-valued transformations $x \xrightarrow{g} h \xrightarrow{f} y$,

$$D\left(f \circ g\right)(x) = Df\left(h\right) Dg\left(x\right)$$

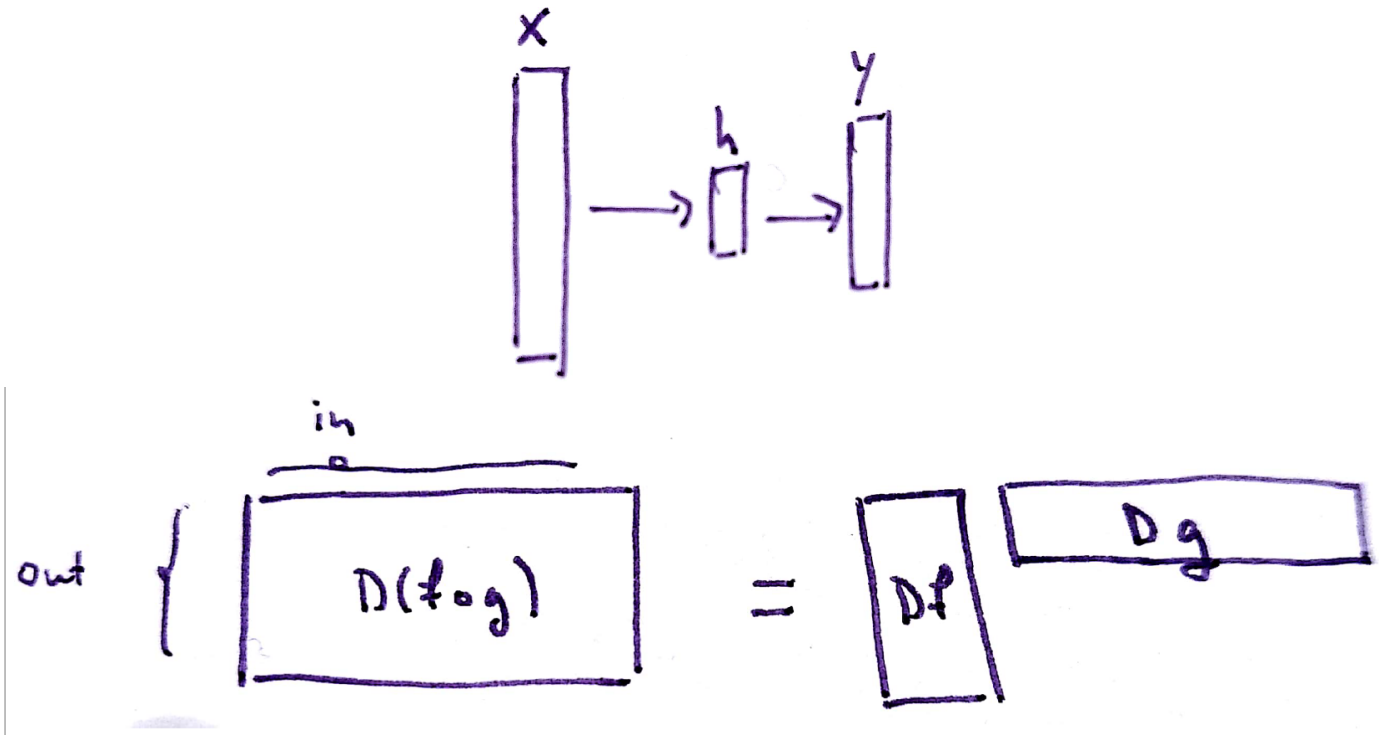


Figure: The first row shows the sequence of vector-valued transformations. The ij^{th} cell of the derivative matrices tells you how the i^{th} output changes when you perturb the j^{th} input slightly.

Flow Graphs

- A flow graph shows how different vectors are computed from one another
  - Nodes are vectors
  - Edges are specific operations on input, which when combined with other incoming edges gives the output
- Crucial: Operations along each edge should be easily differentiable with respect to input

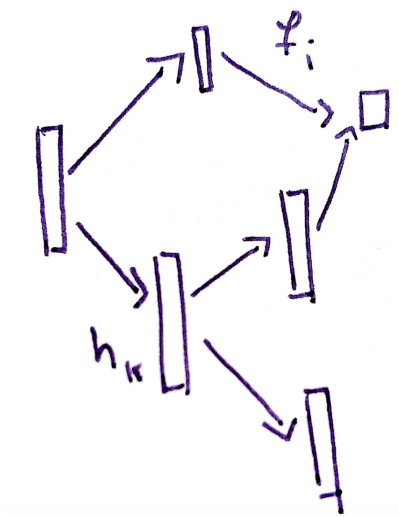


Figure: The main idea of a flow graph.

Chain Rule on Graphs

- When differentiating function f_i with respect to upstream h_k, the chain rule implies

$$Df_i(h_k) = \sum_{j \in C(i)} Df_i(h_j)\, Dh_j(h_i)$$

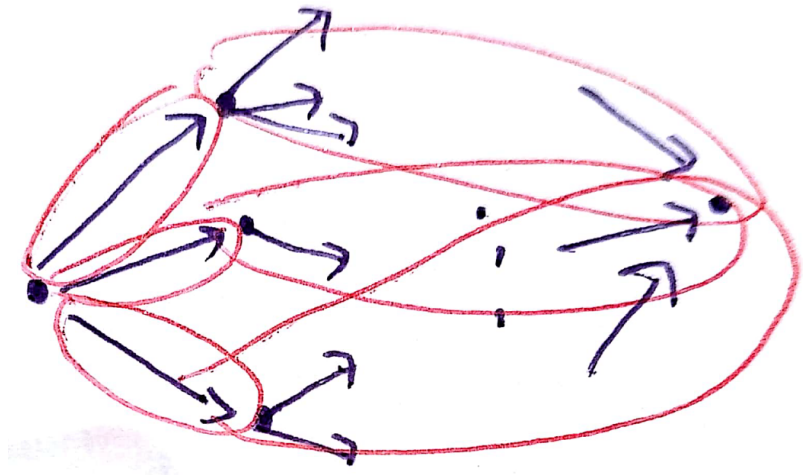


Figure: The graph version of the chain rule sums over the immediate children, referring recursively to similar gradients.

Argument for Formula

- This may look unfamiliar, but it's really the earlier matrix product in disguise
- Consider the case where we conceptually split h into h^1 and h^2

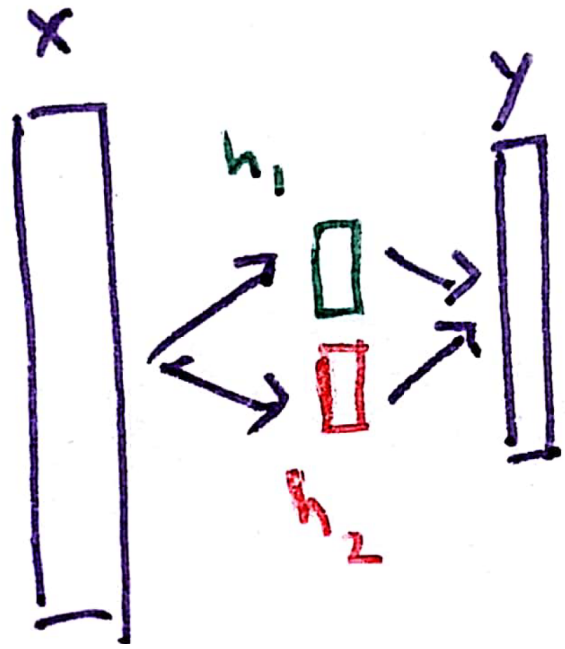


Figure: If we are splitting h, we can split all the derivatives in the matrix product.

Argument for Formula

- This may look unfamiliar, but it's really the earlier matrix product in disguise
- Consider the case where we conceptually split the intermediary h into h^1 and h^2

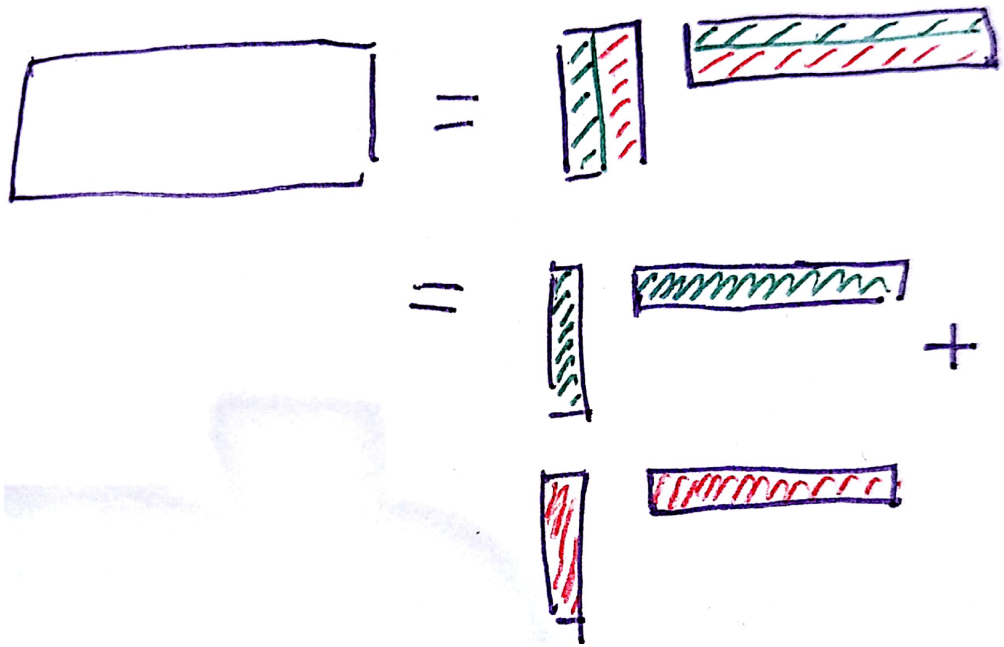


Figure: If we are splitting h, we can split all the derivatives in the matrix product. This can be written exactly as a sum, as in the earlier formula.

Organized Gradient Computation

$$Df_i(h_k) = \sum_{j \in C(i)} Df_i(h_j)\, Dh_j(h_i)$$

- This gives a natural way of organizing gradient computation
- Start with output nodes and flow back towards inputs
- Remember each edge w.r.t input is straightforwards

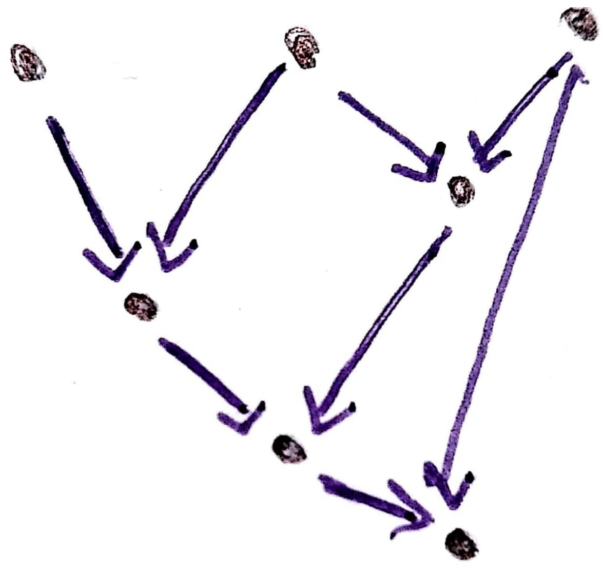


Figure: Organization of computation to get gradients with respect to all inputs.

Organized Gradient Computation

$$Df_i(h_k) = \sum_{j \in C(i)} Df_i(h_j)\, Dh_j(h_i)$$

- This gives a natural way of organizing gradient computation
- Start with output nodes and flow back towards inputs
- Remember each edge w.r.t input is straightforwards

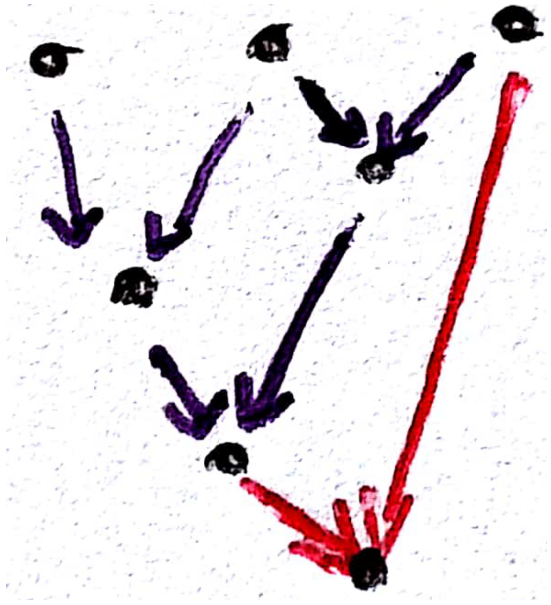


Figure: Organization of computation to get gradients with respect to all inputs.

Organized Gradient Computation

$$Df_i(h_k) = \sum_{j \in C(i)} Df_i(h_j)\, Dh_j(h_i)$$

- This gives a natural way of organizing gradient computation
- Start with output nodes and flow back towards inputs
- Remember each edge w.r.t input is straightforwards

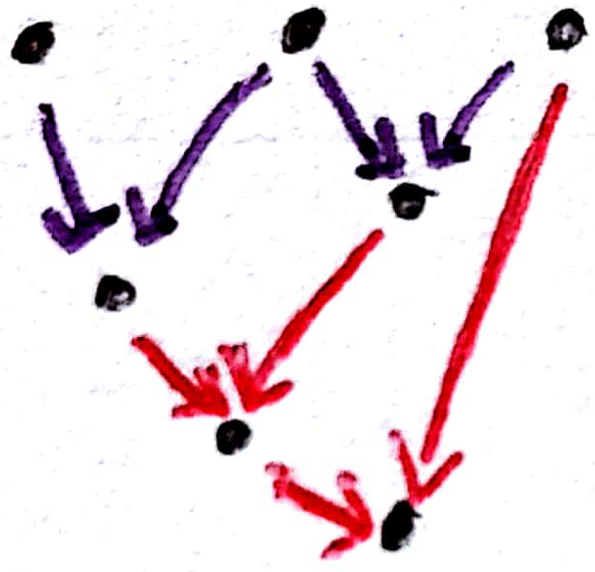


Figure: Organization of computation to get gradients with respect to all inputs.

Organized Gradient Computation

$$Df_i(h_k) = \sum_{j \in C(i)} Df_i(h_j)\, Dh_j(h_i)$$

- This gives a natural way of organizing gradient computation
- Start with output nodes and flow back towards inputs
- Remember each edge w.r.t input is straightforwards

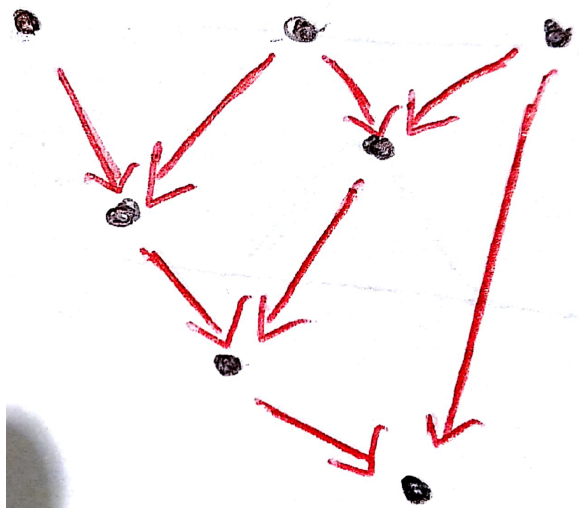


Figure: Organization of computation to get gradients with respect to all inputs.

Consequence: Backpropagation

- The flow graph for a standard deep neural net is straightforwards

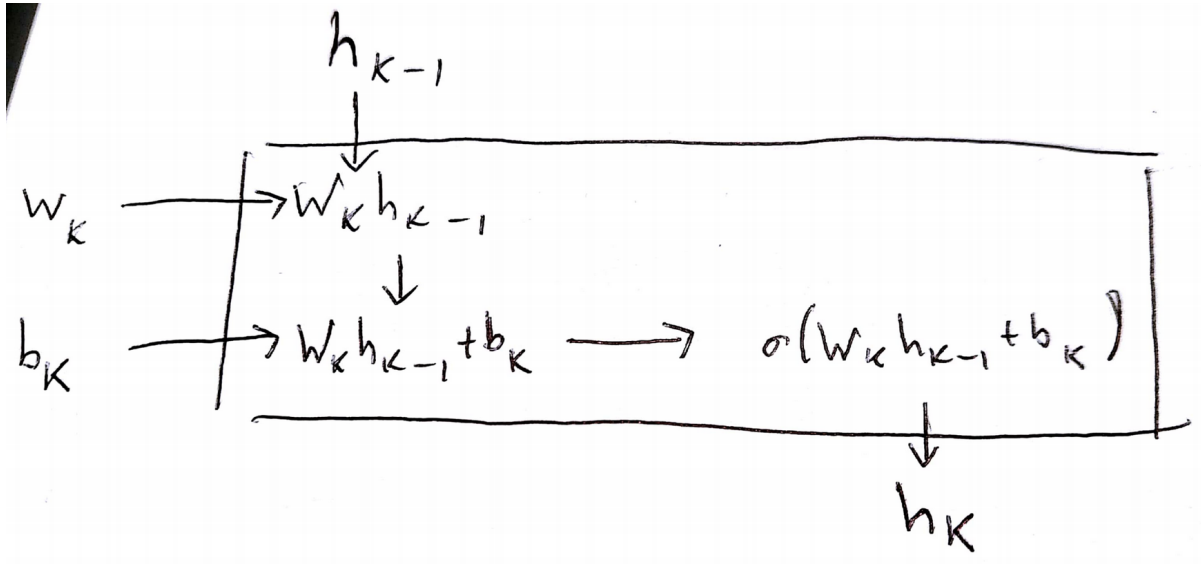


Figure: The flow of computation within a single layer.

Modularity

- Gradients make use only of *local* information
- Can propose arbitrary new layers, as long as you can define gradient of output with respect to input
- Has let people experiment widely in literature,
  - Convolution layers [7]
  - Attention mechanisms [8]
  - LSTM cells [9]
  - Residual layers [6]
  - ...
- And all sorts of mixing and matching between these

Outline

Introduction

Representation Learning

Optimization

Debugging Deep Learning

Hyperparameter Tuning

- Deep learning models can be hard to train
- Depending on various hyperparameters,
  - Sequence of (per-layer) learning rates
    - Other optimization parameters (e.g., momentum)
  - Type, number, and width of each layer
  - Batch size
  - Regularization
  - Weight initialization
  - Activation functions
  - Preprocessing (e.g., standardization, PCA, or quantile transformation)

  the model may or may not train properly

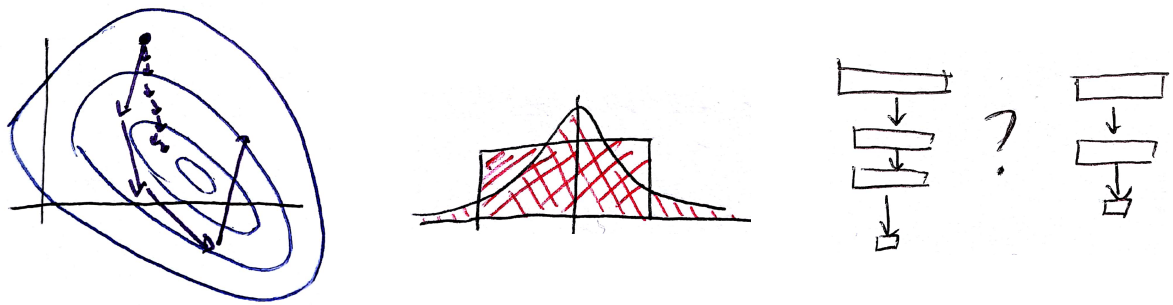


Figure: Three types of hyperparameters that can affect training: Those associated with optimization, regularization of the loss, and model architecture.

Search Strategy

- Multiresolution search: Coarse exploration followed by detailed investigation
- Semi-automated strategy: Analyze simple sweeps, launch new jobs accordingly
- Alternatives to grid search: Search in random grid, or use Bayesian Optimization
- Cheap validation alternatives: Instead of full training in all experiments, use cheap proxies

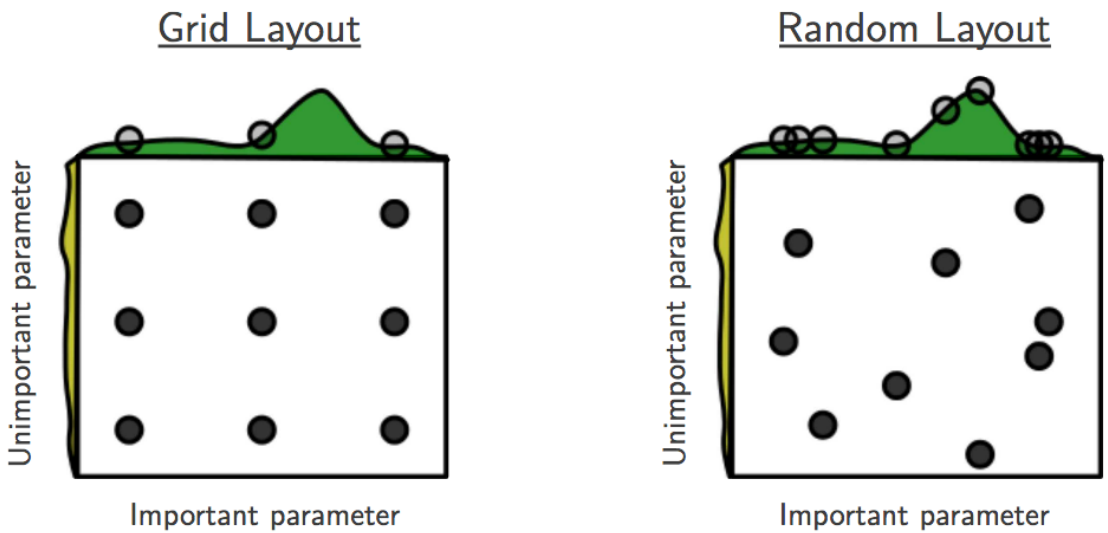


Figure: A comparison of grid and random search, from [2].

Search Strategy

- Multiresolution search: Coarse exploration followed by detailed investigation
- Semi-automated strategy: Analyze simple sweeps, launch new jobs accordingly
- Alternatives to grid search: Search in random grid, or use Bayesian Optimization
- Cheap validation alternatives: Instead of full training in all experiments, use cheap proxies

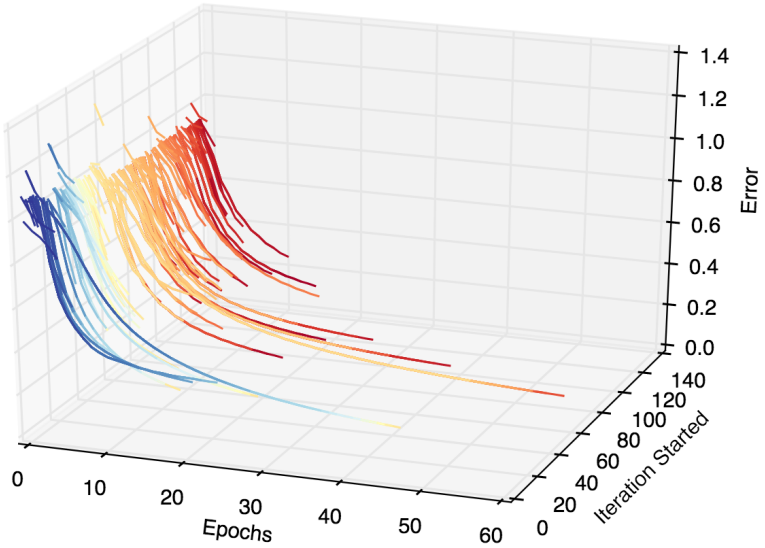


Figure: An example of freeze-thaw bayesian optimization, from [10].

Training Analysis

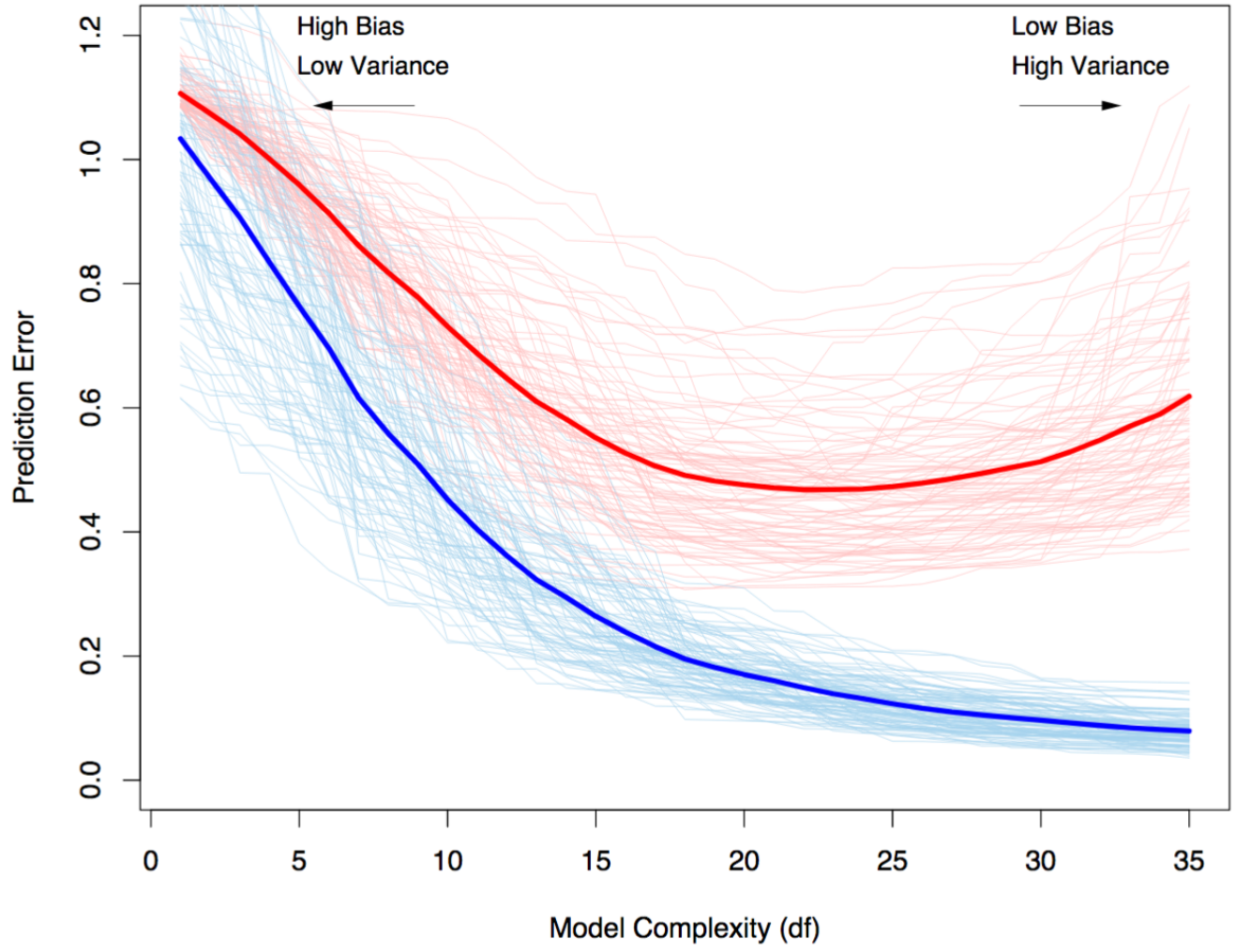


Figure: Typical strategy is to try to first overfit the model, then regularize to ensure validation performance.

Exercise

Discuss the bias-variance tradeoffs associated with these training parameters

- Number and widths of layers
- Learning rates
- Number of iterations

Training Analysis

- It can be helpful to collect statistics
  - Gradients
  - Activation values
- Save your model at checkpoints! Helps both debugging and restarting training

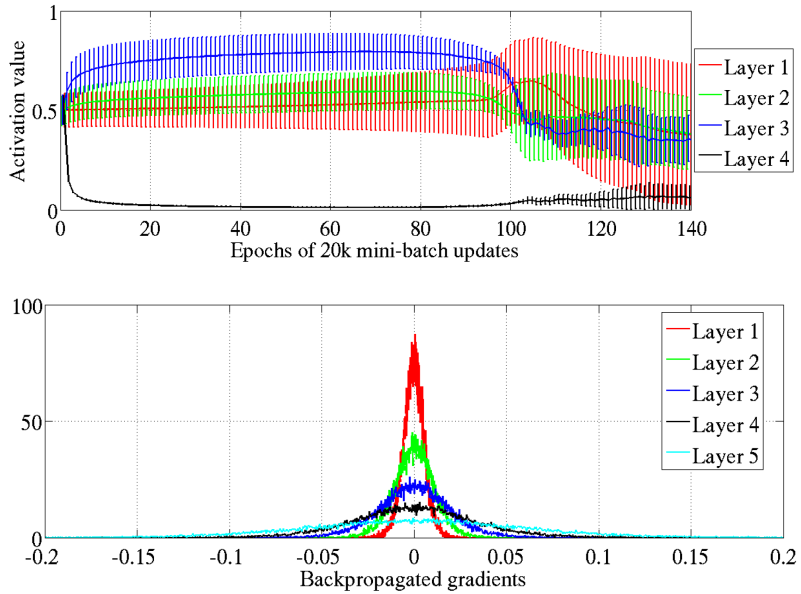


Figure: Detective work to understand failures in training, from [5].

Training Analysis

Other (more sophisticated) forms of visualization that can be useful,

- Study low-dimensional representation of function as it trains
- Vary subset of weights for a given example, see how maps change

Many, many more ideas in [1].

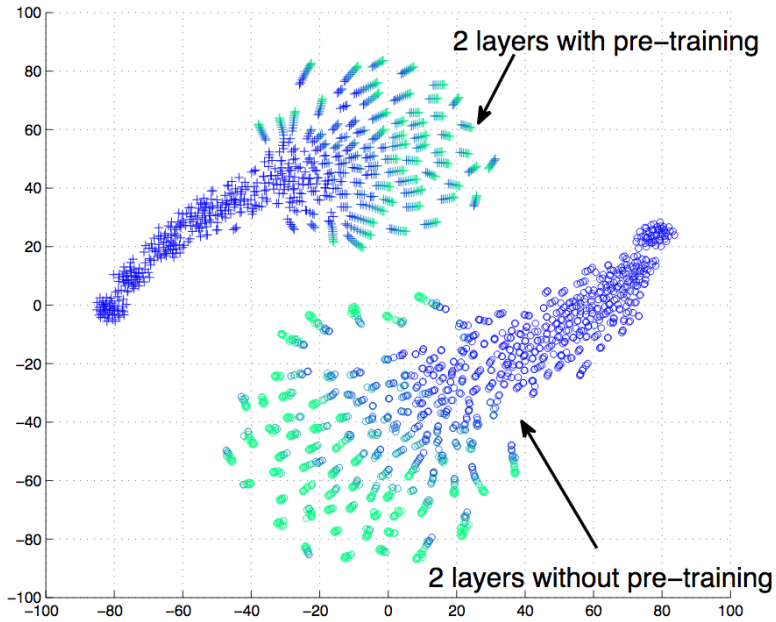


Figure: The model may be invariant to certain transformations of the parameters, but you can directly visualization a dimensionality reduction done on the functions over the course of training, as discussed in [4].

Philosophy for Learning Deep Learning

- Don't ignore history!
  - No need to keep up with every retweeted paper
  - There were many creative people long before deep learning
- Whenever you encounter a new algorithm or formula,
  - Sketch a toy example (one-dimensional, binary instead of multiclass, discrete instead of continuous, ...)
  - Draw picture of a toy example
  - Try to plug in simple values (0, 1, $\pm\infty$ are good choices)
- Genius is overrated
  - First of all, there's stackoverflow
  - No amount of natural talent can make up for dedicated effort
  - You can't learn everything overnight, but it's a journey worth taking

[1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] Persi Diaconis. Theories of data analysis: From magical thinking through classical statistics. *Exploring data tables, trends, and shapes*, pages 1–36, 1985.

[4] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[8] Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. *Distill*, 1(9):e1, 2016.

[9] Jürgen Schmidhuber and Sepp Hochreiter. Long short-term memory. *Neural Comput*, 9(8):1735–1780, 1997.

[10] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.