Neural Networks

Berkeley
N L P

Dan Klein, John DeNero
UC Berkeley

Slides adapted from Greg Durrett

Neural Net Basics

Neural Networks

- Linear classification: $\mathrm{argmax}_y w^\top f(x, y)$

- Want to learn intermediate conjunctive features of the input

the movie was ***not*** *all that* ***good***

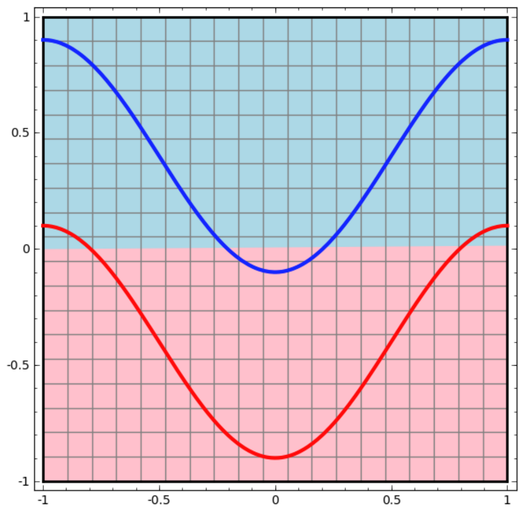
I[contains *not* & contains *good*]

- How do we learn this if our feature vector is just the unigram indicators?

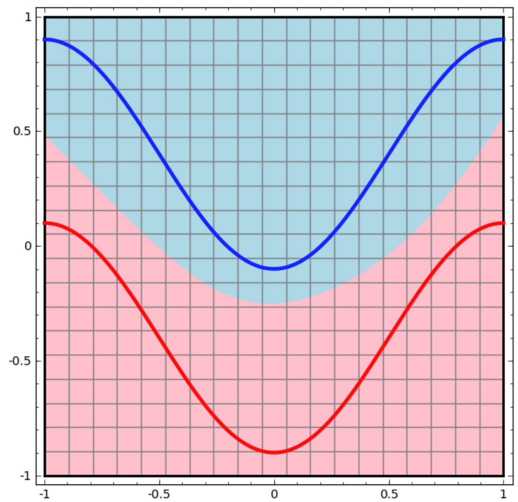
I[contains *not*], I[contains *good*]

Neural Networks

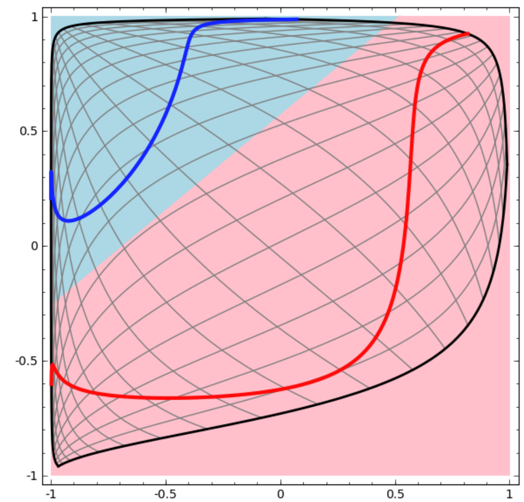
Linear classifier



Neural network



...possible because we transformed the space!



Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Logistic Regression with NNs

$$P(y|\mathbf{x}) = \frac{\exp(w^\top f(\mathbf{x}, y))}{\sum_{y'} \exp(w^\top f(\mathbf{x}, y'))}$$

- Single scalar probability

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}\left([w^\top f(\mathbf{x}, y)]_{y \in \mathcal{Y}}\right)$$

- Compute scores for all possible labels at once (returns vector)

$$\text{softmax}(p)_i = \frac{\exp(p_i)}{\sum_{i'} \exp(p_{i'})}$$

- softmax: exps and normalizes a given vector

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W f(\mathbf{x}))$$

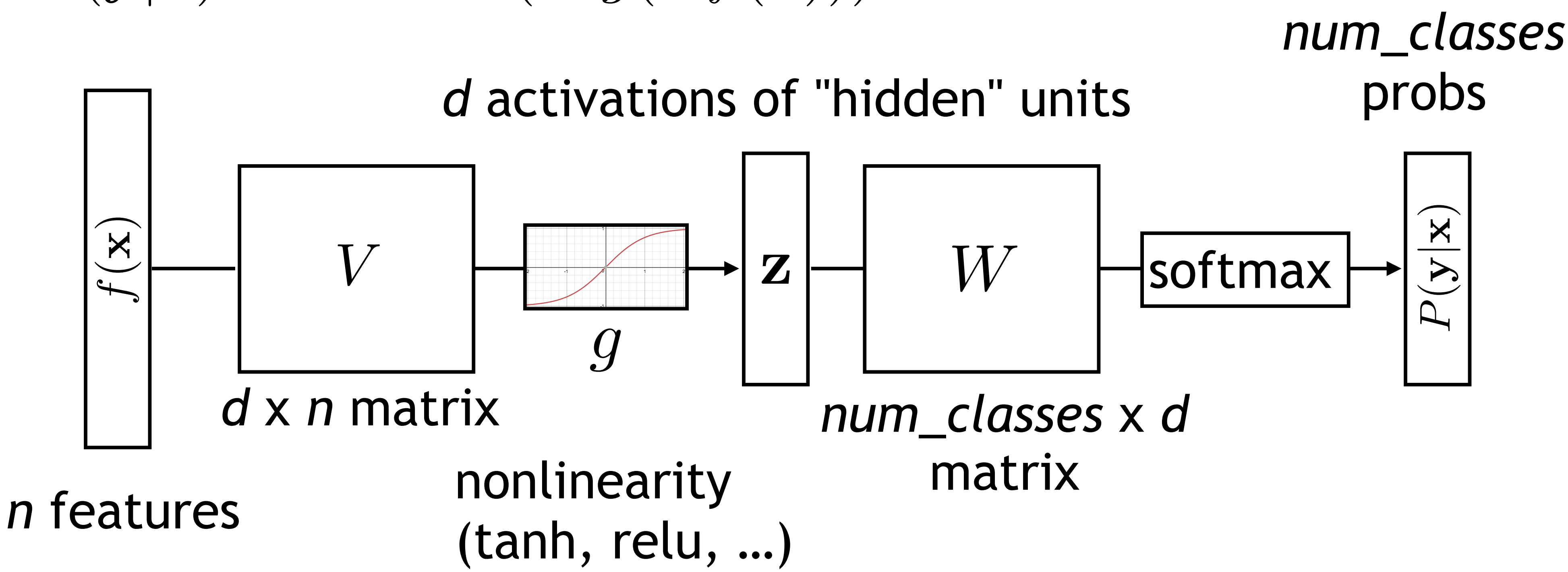
- Weight vector per class;
  W is [num classes x num feats]

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W g(V f(\mathbf{x})))$$

- Now one hidden layer

Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(W g(V f(\mathbf{x})))$$



Objective Function

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(W\mathbf{z}) \qquad \mathbf{z} = g(Vf(\mathbf{x}))$$

- Maximize log likelihood of training data observations

$$\mathcal{L}(\mathbf{x}, i^*) = \log P(y = i^*|\mathbf{x}) = \log\left(\mathrm{softmax}(W\mathbf{z}) \cdot e_{i^*}\right)$$

- *i**: index of the gold label
- e_i: 1 in the *i*th row, zero elsewhere. This dot selects the *i**th index

$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_j \exp(W\mathbf{z}) \cdot e_j$$

Training Procedure

- Initialize parameters
- For each epoch (one pass through all the training examples):
  - Shuffle the examples
  - Group them into mini-batches
  - For each mini-batch (these days often just called a "batch"):
    - Compute the loss over the mini-batch
    - Compute the gradient of the loss w.r.t. the parameters
    - Update parameters according to a gradient-based optimizer
  - Evaluate the current network on a held-out validation set

Training Tips

Batching

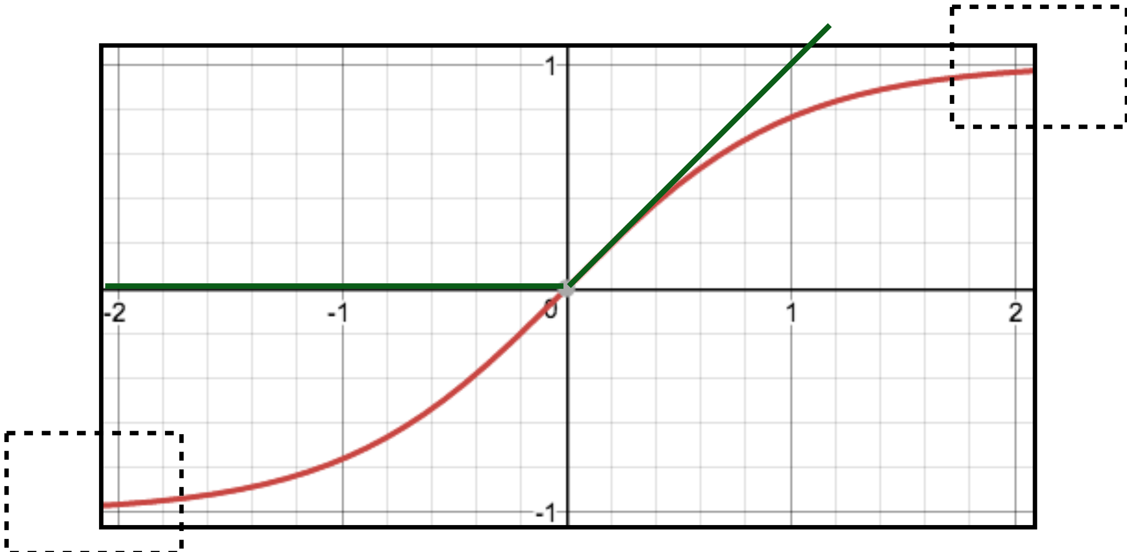
- Batching data gives speedups due to more efficient matrix operations
- Need to process a batch at a time

```
# input is [batch_size, num_feats]
# gold_label is [batch_size, num_classes]
def make_update(input, gold_label)
    ...
    probs = ffnn.forward(input) # [batch_size, num_classes]
    loss = torch.sum(torch.neg(torch.log(probs)).dot(gold_label))
    ...
```

- A batch size of 32 is typical, but the best choice is model & application dependent

Initialization

- Nonlinear model…how does this affect things?



- If cell activations are large in absolute value, gradients are small.
- ReLU: Zero gradient when activation is negative.

Initialization

1) Can’t use zeroes for parameters to generate hidden layers: all values in that hidden layer are always 0 and have zero gradients.

2) Initialize too large and cells are saturated

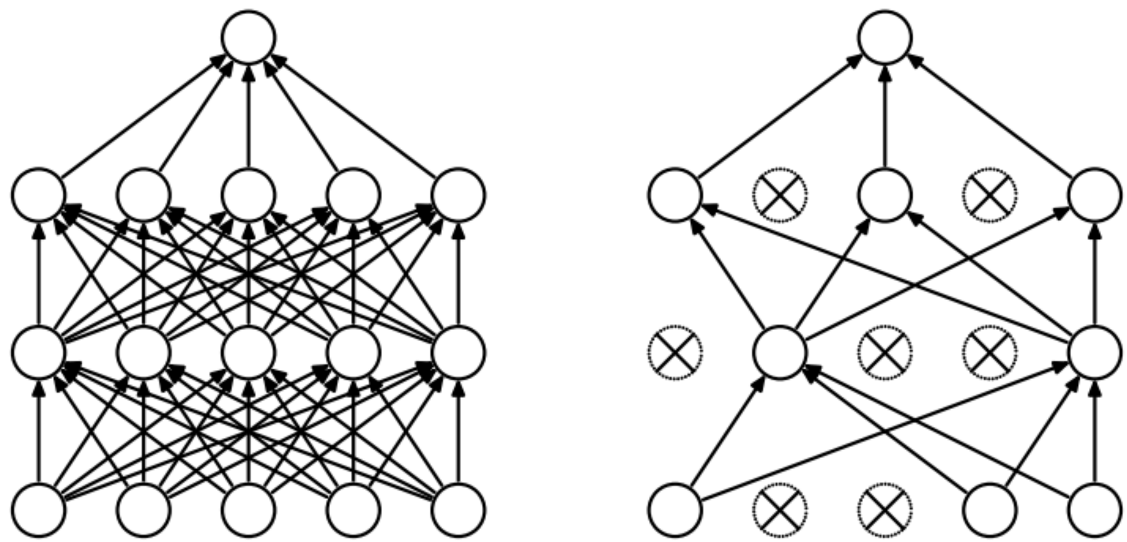
- A common approach is random uniform/normal initialization with appropriate scale (small is typically good)
- Xavier Glorot (2010) uniform initializer: $U\left[-\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}, +\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}\right]$
- Want variance of inputs and gradients for each layer to be similar

Dropout

- Probabilistically zero out some activations during training to prevent overfitting, but use the whole network at test time

- Form of stochastic regularization

- Similar to benefits of ensembling: network needs to be robust to missing signals, so it has redundancy



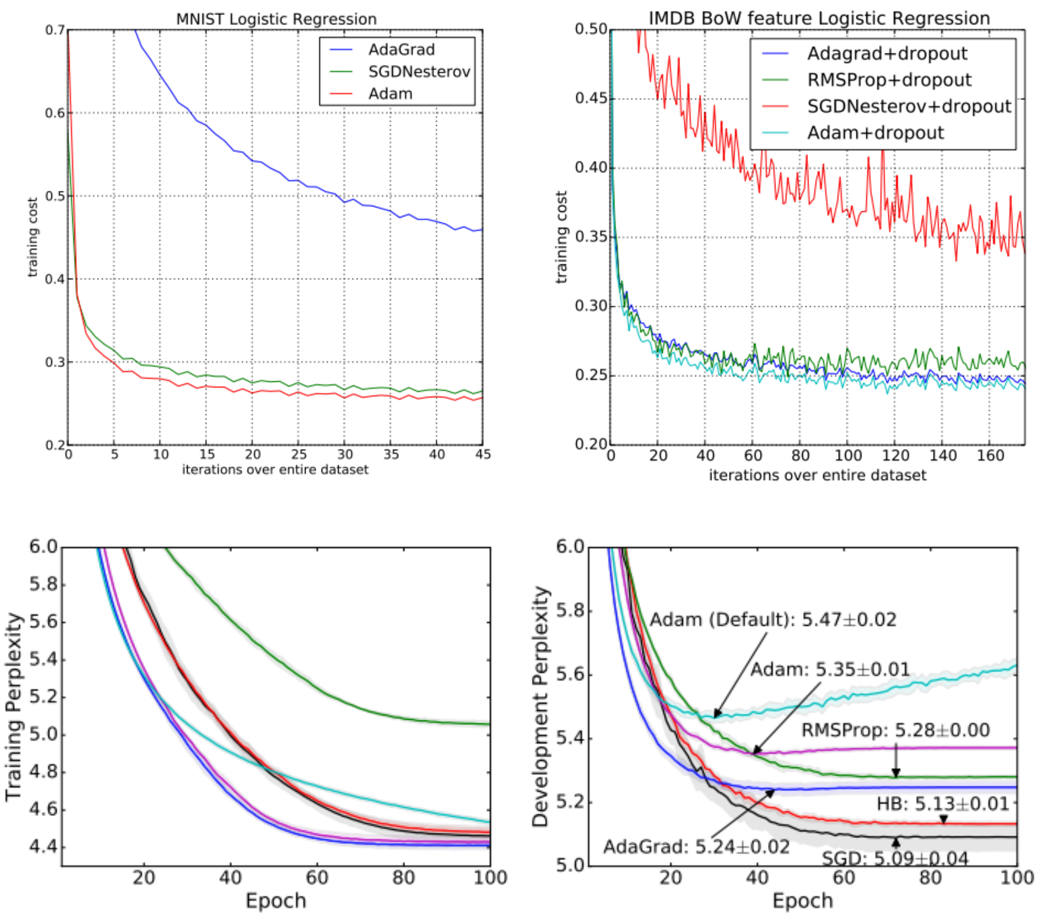
(a) Standard Neural Net (b) After applying dropout.

- One line in Pytorch/Tensorflow

Srivastava et al. (2014)

Optimizer

- Adam (Kingma and Ba, ICLR 2015): very widely used. Adaptive step size + momentum
- Wilson et al. NIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)
- One more trick: **gradient clipping** (set a max value for your gradients)

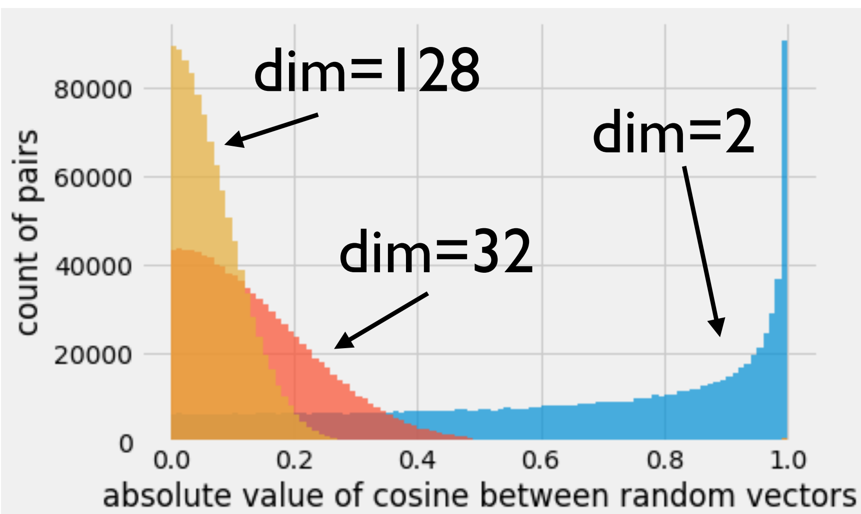


(e) Generative Parsing (Training Set) (f) Generative Parsing (Development Set)

Embeddings

Symbol Embeddings

- Words and characters are discrete symbols, but input to a neural network must be real-valued
- Different symbols in language do have common characteristics that correlate with their distributional properties
- An "embedding" for a symbol: a learned low-dimensional vector



Intuition: Low-rank approximation to a co-occurrence matrix

$$m\left\{\left[\overbrace{\qquad A \qquad}^{n}\right]\right. \approx m\left\{\left[\overbrace{X}^{k}\right]\right.\left.\left[\overbrace{\qquad Y \qquad}^{n}\right]\right\}k$$