Neural Net Basics

Neural Networks

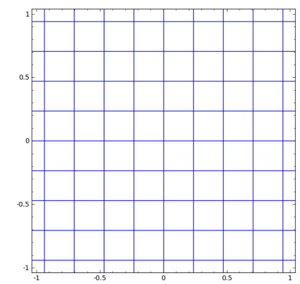
Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear transformation | Warp space | Shift

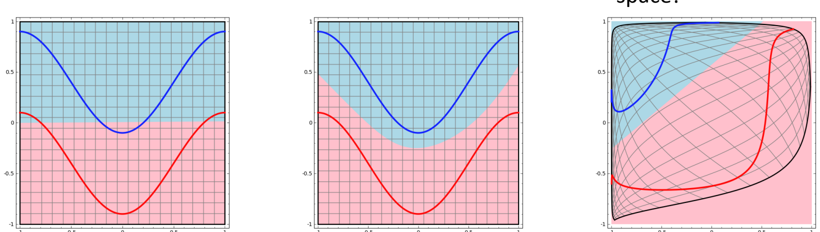
$$\mathrm{pred} = \mathbf{w}'^{\top}\mathbf{y}$$



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

Neural Networks

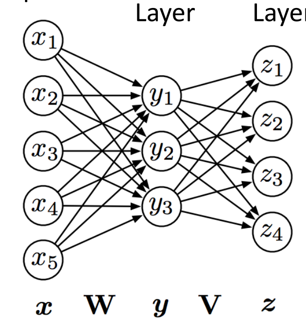
Linear classifier | Neural network | Linear classification in the transformed space!



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

Deep Neural Networks

Input | First Layer | Second Layer



$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

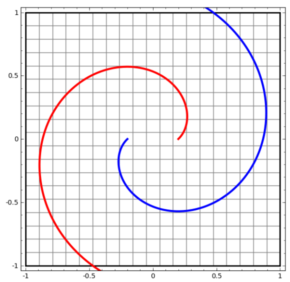
$$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

"Feedforward" computation (not recurrent)

Adopted from Chris Dyer

Deep Neural Networks



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

Feedforward Networks, Backpropagation

Vectorization and Softmax

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

- Single scalar probability

- Three classes, "different weights"

$$\begin{array}{ccccc} \mathbf{w}_1^\top \mathbf{x} & & -1.1 & & 0.036 \\ \mathbf{w}_2^\top \mathbf{x} & = & 2.1 & \xrightarrow{\text{softmax}} & 0.89 \\ \mathbf{w}_3^\top \mathbf{x} & & -0.4 & & 0.07 \end{array}$$

class probs

- Softmax operation = "exponentiate and normalize"
- We write this as: $\mathrm{softmax}(W\mathbf{x})$

Logistic Regression with NNs

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

- Single scalar probability

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wf(\mathbf{x}))$$

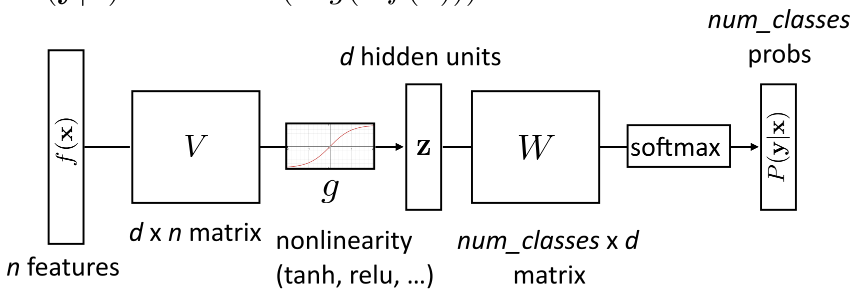
- Weight vector per class; W is [num classes x num feats]

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$

- Now one hidden layer

Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$



Training Neural Networks

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

- Maximize log likelihood of training data

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

- *i**: index of the gold label
- e_i: 1 in the *i*th row, zero elsewhere. Dot by this = select *i*th index

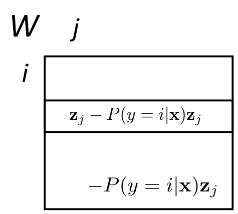
$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_j \exp(W\mathbf{z}) \cdot e_j$$

Computing Gradients

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

- Gradient with respect to *W*

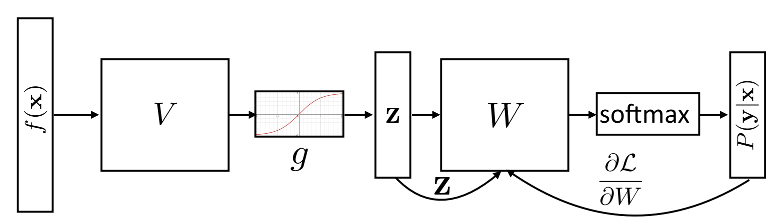
$$\frac{\partial}{\partial W_{ij}} \mathcal{L}(\mathbf{x}, i^*) = \begin{cases} \mathbf{z}_j - P(y = i|\mathbf{x})\mathbf{z}_j & \text{if } i = i^* \\ -P(y = i|\mathbf{x})\mathbf{z}_j & \text{otherwise} \end{cases}$$



- Looks like logistic regression with ***z*** as the features!

Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$



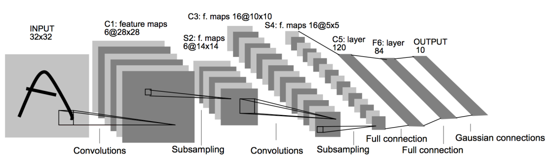
Backpropagation

- Gradients of output weights *W* are easy to compute — looks like logistic regression with hidden layer ***z*** as feature vector
- Use the chain rule from calculus to compute an update for *V*. Looks like running the network in reverse
- Need to remember the values from the forward computation
- Autodiff tools mean you never need to implement this!

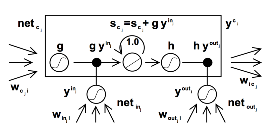
Neural Nets History

History: NN "dark ages"

- Convnets: applied to MNIST by LeCun in 1998



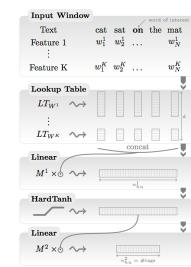
- LSTMs: Hochreiter and Schmidhuber (1997)



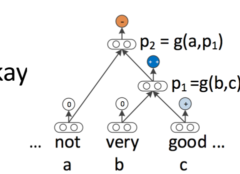
- Henderson (2003): neural shift-reduce parser, not SOTA

2008-2013: A glimmer of light…

- Collobert and Weston 2011: "NLP (almost) from scratch"
  - Feedforward neural nets induce features for sequential CRFs ("neural CRF")
  - 2008 version was marred by bad experiments, claimed SOTA but wasn't, 2011 version tied SOTA



- Krizhevskey et al. (2012): AlexNet for vision
- Socher 2011-2014: tree-structured RNNs working okay



2014: Stuff starts working

- Kim (2014) + Kalchbrenner et al. (2014): sentence classification / sentiment (convnets work for NLP?)
- Sutskever et al. + Bahdanau et al.: seq2seq for neural MT (LSTMs work for NLP?)
- Chen and Manning transition-based dependency parser (even feedforward networks work well for NLP?)
- 2015: explosion of neural nets for everything under the sun

Why didn't they work before?

- **Datasets too small**: for MT, not really better until you have 1M+ parallel sentences (and really need a lot more)
- **Optimization not well understood**: good initialization, per-feature scaling + momentum (Adagrad / Adadelta / Adam) work best out-of-the-box
  - **Regularization**: dropout is pretty helpful
  - **Computers not big enough**: can't run for enough iterations
- **Inputs**: need word representations to have the right continuous semantics

Next Time

- More implementation details: practical training techniques
- Word representations / word vectors
- word2vec, GloVe