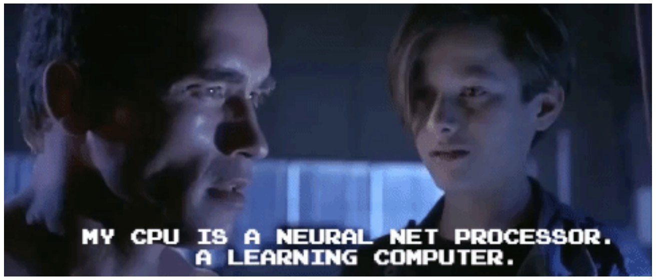
Neural Net Basics

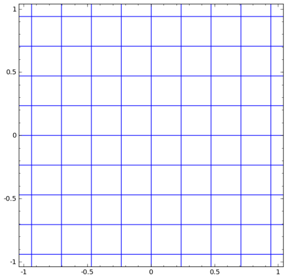


Neural Networks

$$\mathbf{z} = \underset{\substack{\nearrow \\ \text{Nonlinear} \\ \text{transformation}}}{g}(\underset{\substack{\nwarrow \\ \text{Warp} \\ \text{space}}}{V} f(\mathbf{x}) + \underset{\substack{\uparrow \\ \text{Shift}}}{\mathbf{b}})$$

$$y_{\text{pred}} = \text{argmax}_y \mathbf{w}_y^\top \mathbf{z}$$

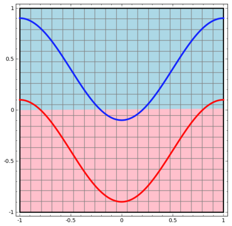
- Ignore shift / +**b** term for the rest of the course



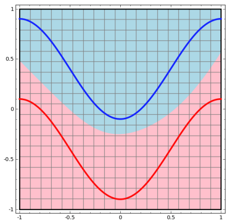
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

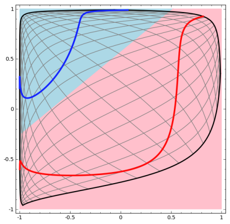
Linear classifier



Neural network



Linear classification in the transformed space!



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

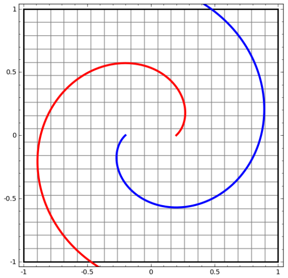
Deep Neural Networks

$$\mathbf{z}_1 = g(V_1 f(\mathbf{x}))$$

$$\mathbf{z}_2 = g(V_2 \mathbf{z}_1)$$

$$\ldots$$

$$y_{\text{pred}} = \text{argmax}_y \mathbf{w}_y^\top \mathbf{z}_n$$



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

Feedforward Networks

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

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

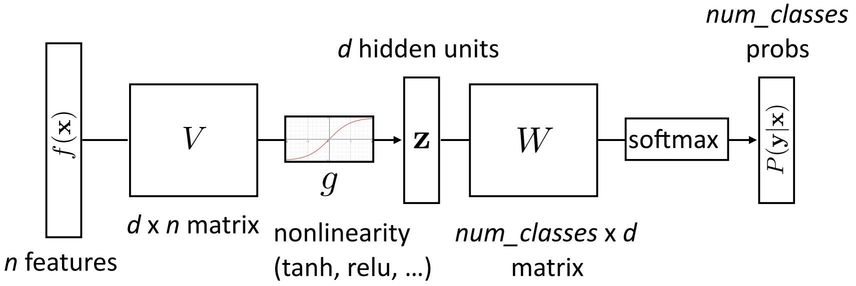
- Weight vector per class; *W* is [num classes x num feats]

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

- Now one hidden layer

Neural Networks for Classification

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



Backpropagation
(we’ll go quickly — derivations at end of slides)

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

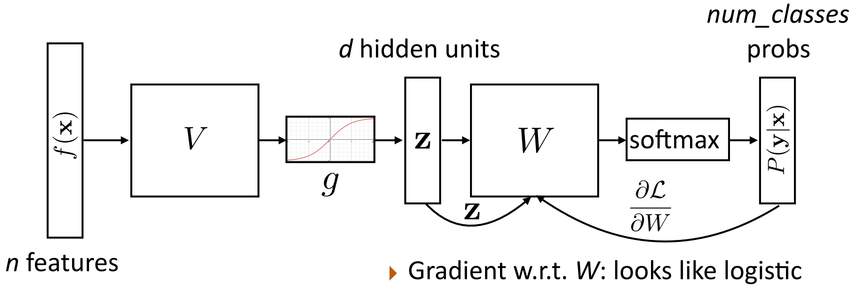
$$\mathcal{L}(\mathbf{x}, i^*) = \log P(y = i^*|\mathbf{x}) = \log\left(\text{softmax}(W\mathbf{z}) \cdot e_{i^*}\right)$$

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

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

Backpropagation: Picture

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

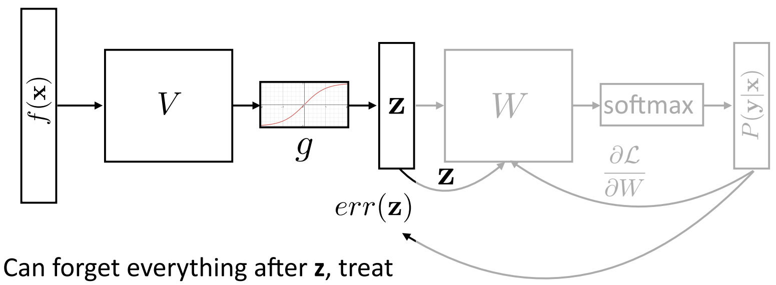


- Gradient w.r.t. *W*: looks like logistic regression, can be computed treating **z** as the features

---

Backpropagation: Picture

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



- Can forget everything after **z**, treat it as the output and keep backpropping

Computing Gradients: Backpropagation

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

Activations at hidden layer

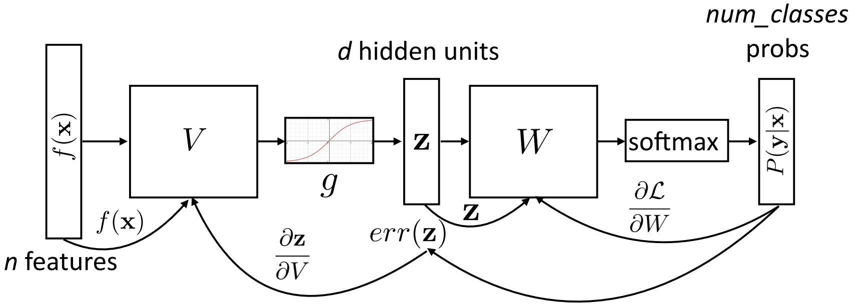
- Gradient with respect to *V*: apply the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial V_{ij}} = \boxed{\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}}} \boxed{\frac{\partial \mathbf{z}}{\partial V_{ij}}} \qquad \frac{\partial \mathbf{z}}{\partial V_{ij}} = \boxed{\frac{\partial g(\mathbf{a})}{\partial \mathbf{a}}} \boxed{\frac{\partial \mathbf{a}}{\partial V_{ij}}} \qquad \mathbf{a} = Vf(\mathbf{x})$$

- First term: *err*(**z**); represents gradient w.r.t. **z**
- First term: gradient of nonlinear activation function at ***a*** (depends on current value)
- Second term: gradient of linear function

Backpropagation: Picture

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

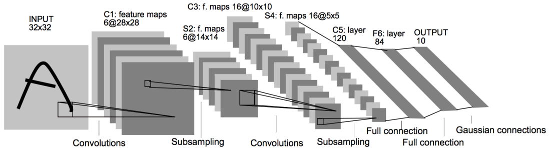


- Combine backward gradients with forward-pass products

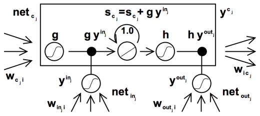
Neural Nets History

History: NN "dark ages"

- Convnets: applied to MNIST by LeCun in 1998



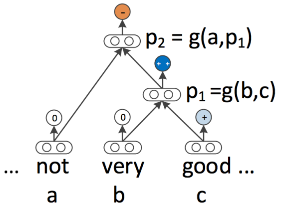
- LSTMs: Hochreiter and Schmidhuber (1997)



- 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")
- 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. (2014) + Bahdanau et al. (2014): seq2seq for neural MT (LSTMs work for NLP?)
- Chen and Manning transition-based dependency parser (feedforward)
- 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

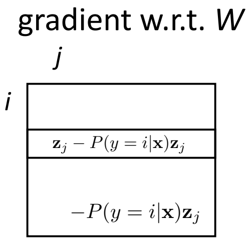
Backpropagation — Derivations (not covered in lecture, optional but useful for Assignment 2)

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:*

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


gradient w.r.t. *W*

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

Computing Gradients: Backpropagation

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

$$\mathbf{z} = g(Vf(\mathbf{x}))$$

Activations at hidden layer

- Gradient with respect to *V*: apply the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial V_{ij}} = \boxed{\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}}} \frac{\partial \mathbf{z}}{\partial V_{ij}}$$

[some math…]

$$err(\text{root}) = e_{i^*} - P(\mathbf{y}|\mathbf{x})$$

dim = *num_classes*

$$\boxed{\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} = err(\mathbf{z}) = W^\top err(\text{root})}$$

dim = *d*

Computing Gradients: Backpropagation

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

$$\mathbf{z} = g(Vf(\mathbf{x}))$$

Activations at hidden layer

- Gradient with respect to *V*: apply the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial V_{ij}} = \boxed{\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}}} \boxed{\frac{\partial \mathbf{z}}{\partial V_{ij}}} \qquad \frac{\partial \mathbf{z}}{\partial V_{ij}} = \boxed{\frac{\partial g(\mathbf{a})}{\partial \mathbf{a}}} \boxed{\frac{\partial \mathbf{a}}{\partial V_{ij}}} \qquad \mathbf{a} = Vf(\mathbf{x})$$

- First term: gradient of nonlinear activation function at ***a*** (depends on current value)
- Second term: gradient of linear function
- First term: *err*(**z**); represents gradient w.r.t. **z**

Backpropagation

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

- Step 1: compute $err(\mathrm{root}) = e_{i^*} - P(\mathbf{y}|\mathbf{x})$ (vector)
- Step 2: compute derivatives of *W* using *err*(root) (matrix)
- Step 3: compute $\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} = err(\mathbf{z}) = W^\top err(\mathrm{root})$ (vector)
- Step 4: compute derivatives of *V* using *err*(**z**) (matrix)
- Step 5+: continue backpropagation if necessary
- See `optimization.py` in the homework to understand this more