Last time:
NB
BOW LR

Neural Networks (INLP ch. 3)

CS 685, Spring 2021
Advanced Topics in Natural Language Processing
http://brenocon.com/cs685
https://people.cs.umass.edu/~brenocon/cs685_s21/

Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

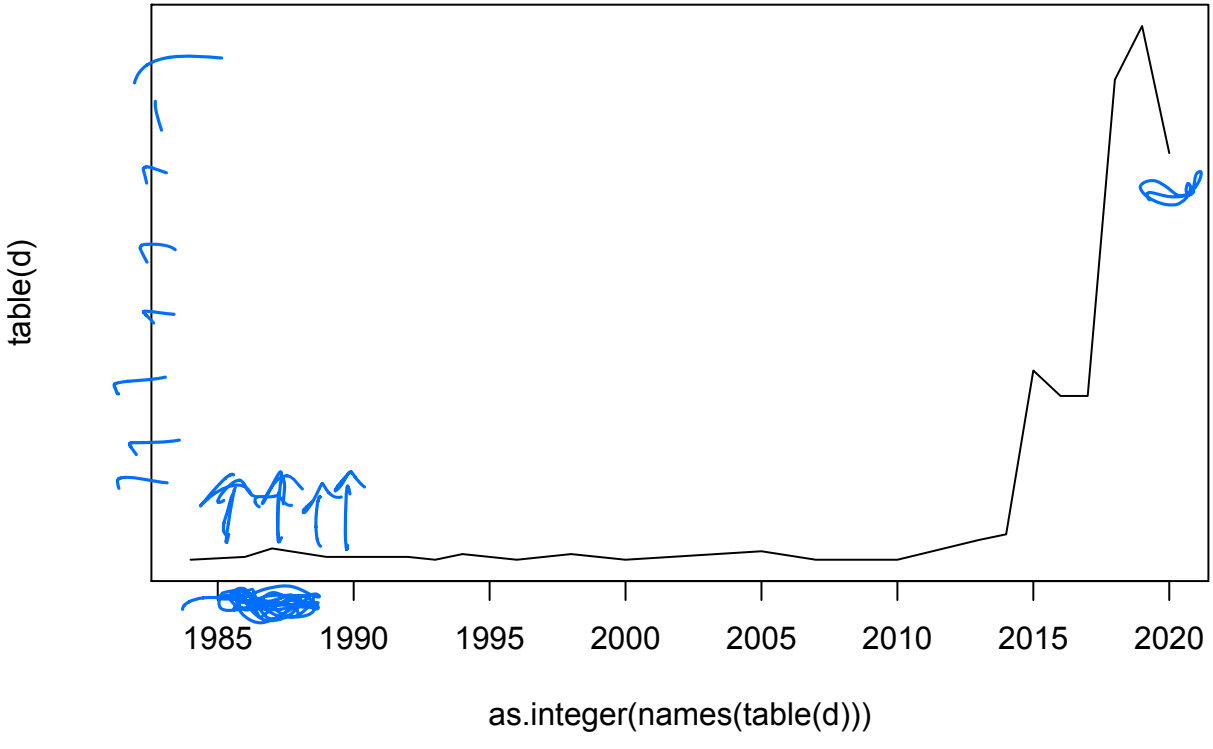
some slides adapted from Mohit Iyyer, Jordan Boyd-Graber, Richard Socher, Eisenstein (2019)

Neural Networks in NLP

- Motivations:
  - Word sparsity => denser word representations
  - Nonlinearity
- Models
  - BoE / Deep Averaging
- Learning
  - Backprop
  - Dropout

The Second Wave: NNs in NLP

- % of ACL paper titles/venues with “connectionist/connectionism”, “parallel distributed”, “neural network”, or “deep learning”
  - https://www.aclweb.org/anthology/



NN Text Classification

- Goals:
  - Avoid feature engineering
  - Generalize beyond individual words
- General model architectures that work well for many different datasets (and tasks!)
- For medium-to-large labeled training datasets, deep learning methods generally outperform feature-based LogReg

Word sparsity

- Alternate view of Bag-of-Words classifiers: every word has a "one-hot" representation.
  - Represent each word as a vector of zeros with a single 1 identifying the index of the word
- Doc BOW **x** = average of all words' vectors

vocabulary
i
hate
love
the
movie
film

movie = <0, 0, 0, 0, 1, 0>

film = <0, 0, 0, 0, 0, 1>

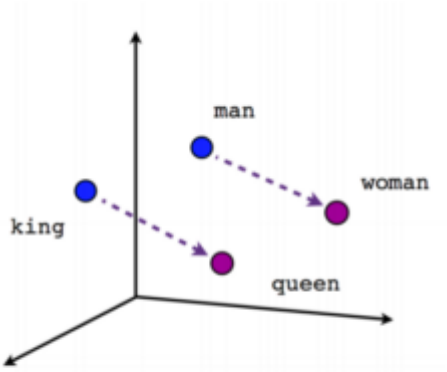
what are the issues of representing a word this way?

all wavelengths are orthogonal

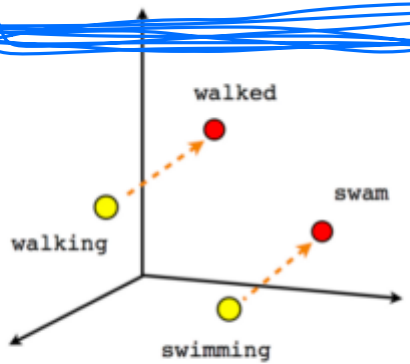
Word embeddings

- Represent words with low(ish)-dimensional vectors called **embeddings**
- Today: word embeddings are the first "lookup" layer in an NN. Every word in vocabulary has a vector — these are model parameters.
  - Ideally: semantically similar words get similar vectors. Or other semantic properties??

king =
[0.23, 1.3, -0.3, 0.43]



Male-Female

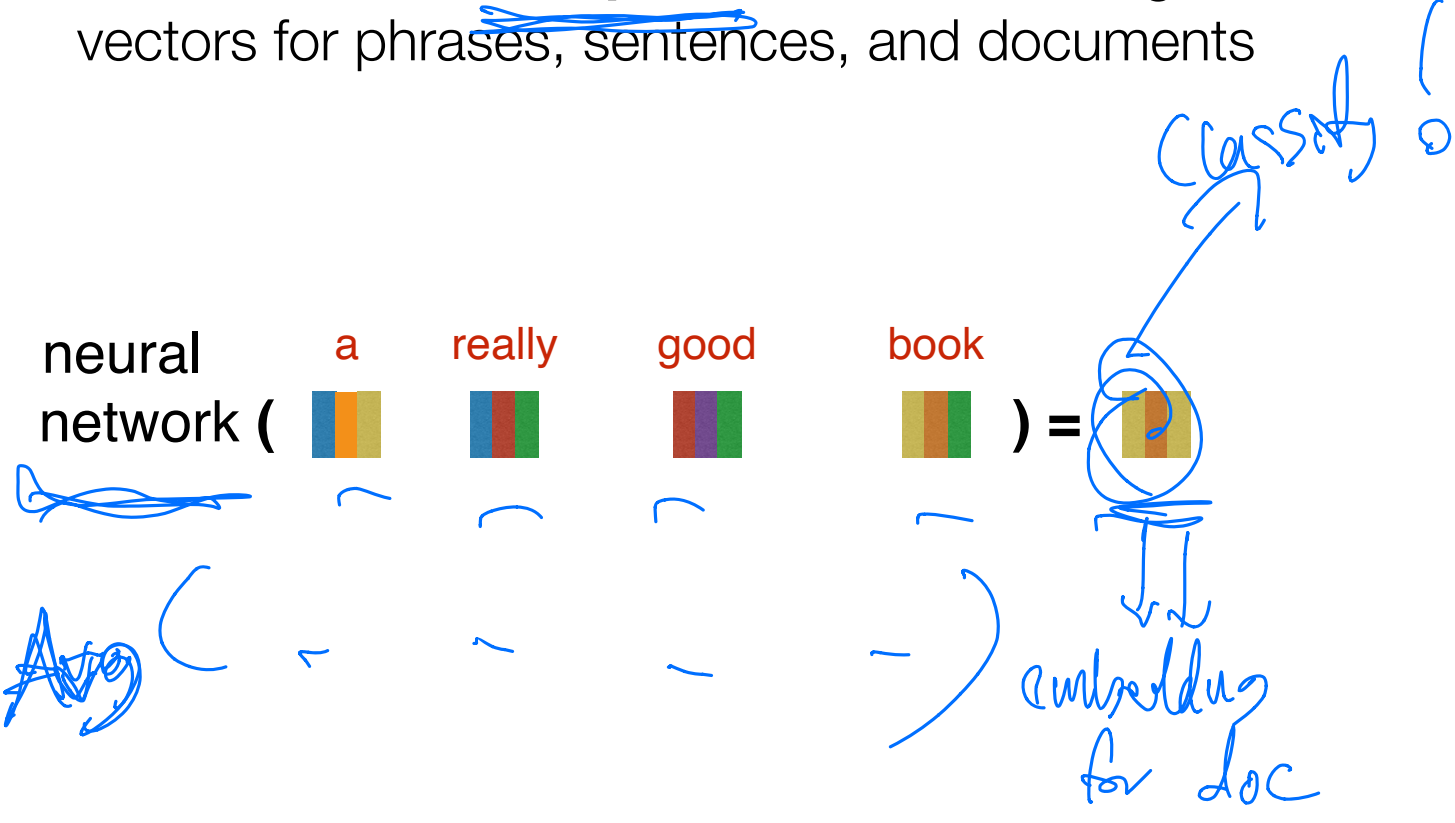


Verb tense

Country-Capital

composing embeddings

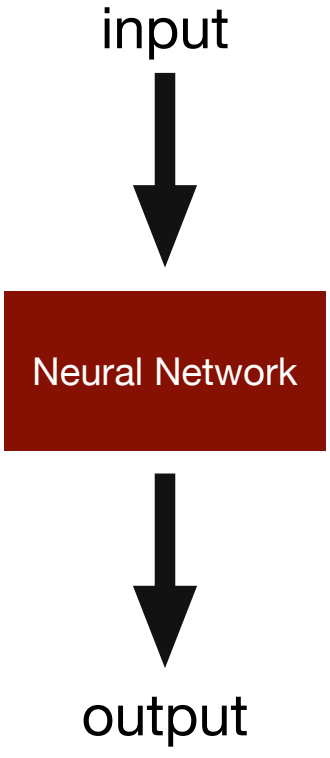
- neural networks **compose** word embeddings into vectors for phrases, sentences, and documents



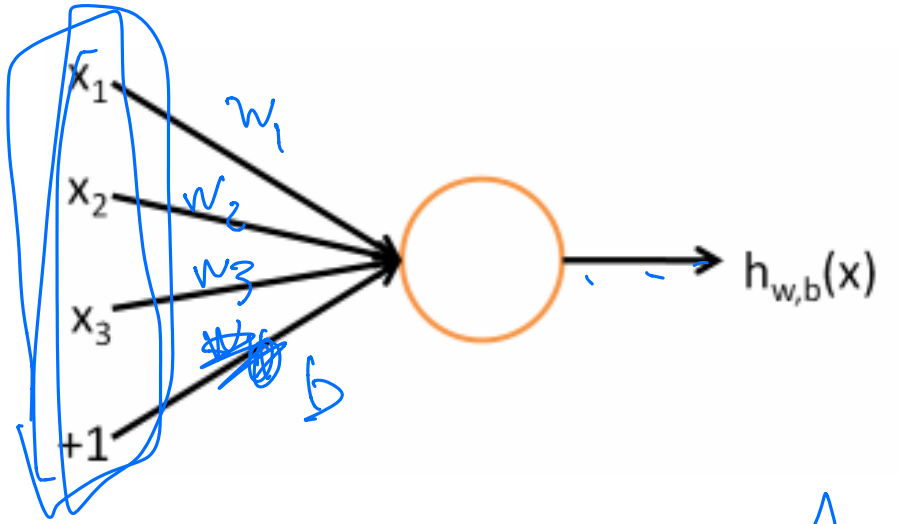
what is **deep learning**?

f (input) = output

what is **deep learning**?



Logistic Regression by Another Name: Map inputs to output



Input

$x_1 \dots x_d = \vec{x}$

Output

$f\left(\sum_i x_i w_i + b\right)$

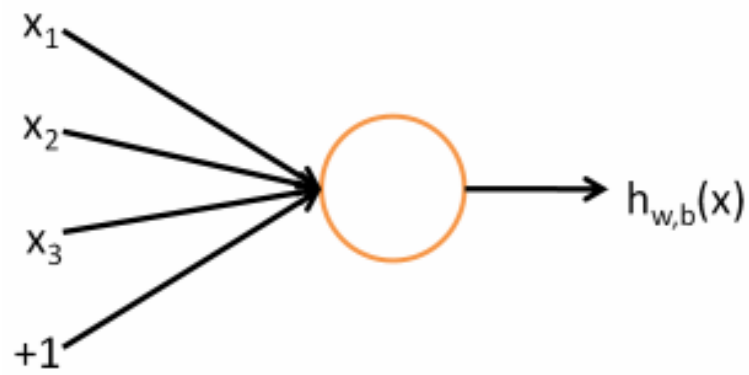
$\in \mathbb{R}$

Activation

$f(z) = \frac{1}{1+e^{-z}}$

Logistic Sigmoid

Logistic Regression by Another Name: Map inputs to output



Input

Vector $x_1 \ldots x_d$

Output

$$f\left(\sum_i W_i x_i + b\right)$$

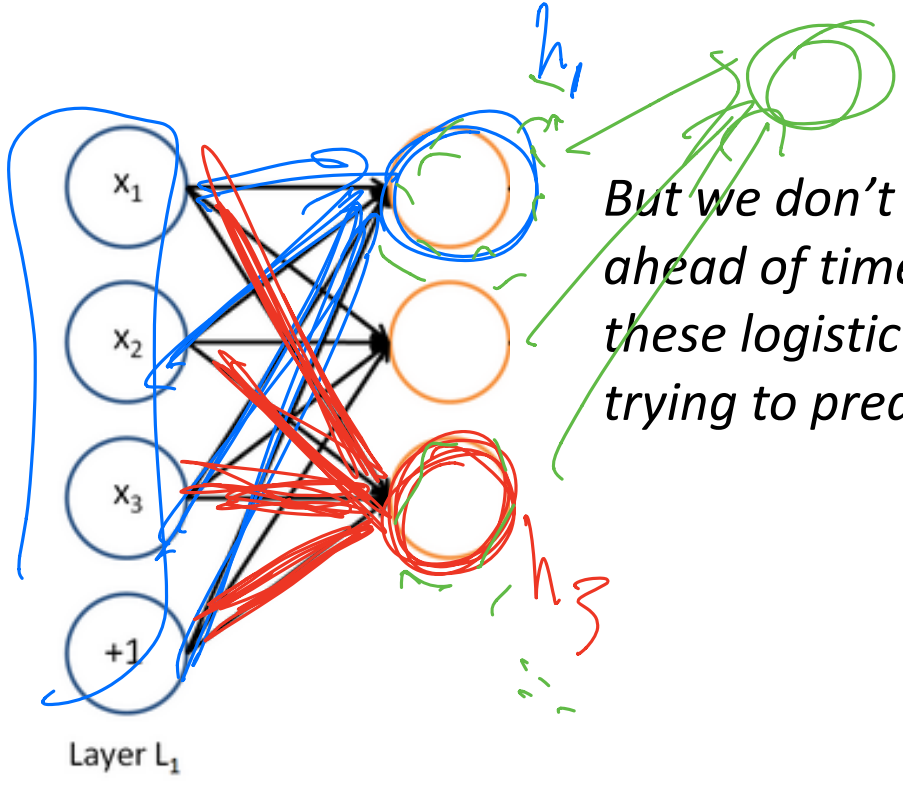
Activation

$$f(z) \equiv \frac{1}{1+\exp(-z)}$$

pass through nonlinear sigmoid

NN: kind of like several intermediate logregs

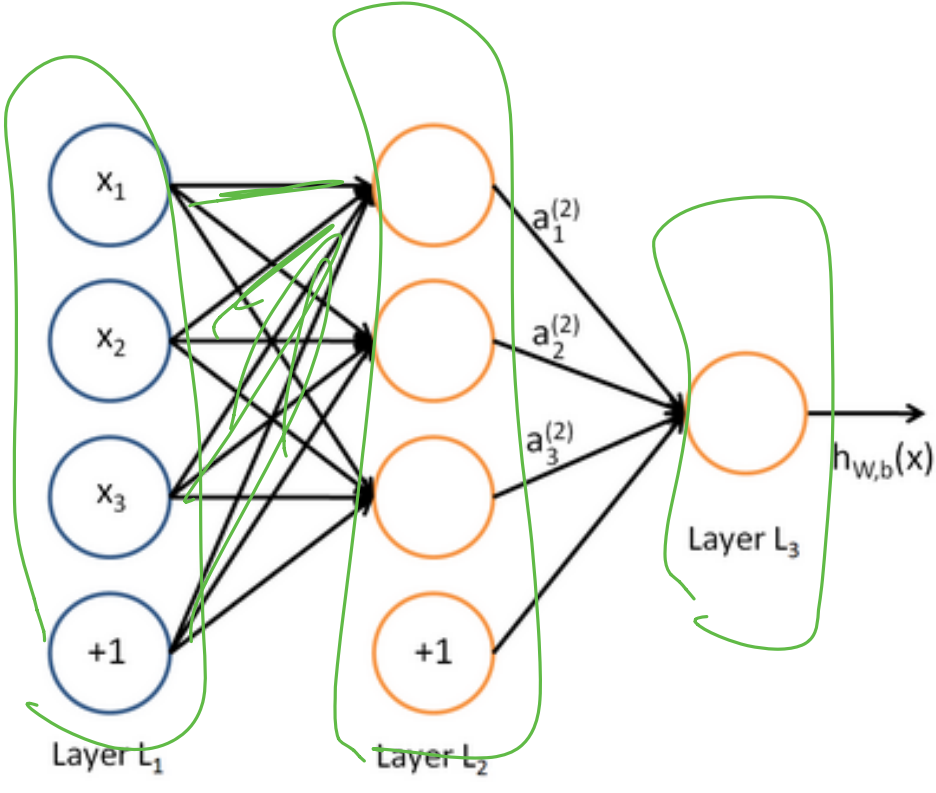
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

NN: kind of like several intermediate logregs

… which we can feed into another logistic regression function

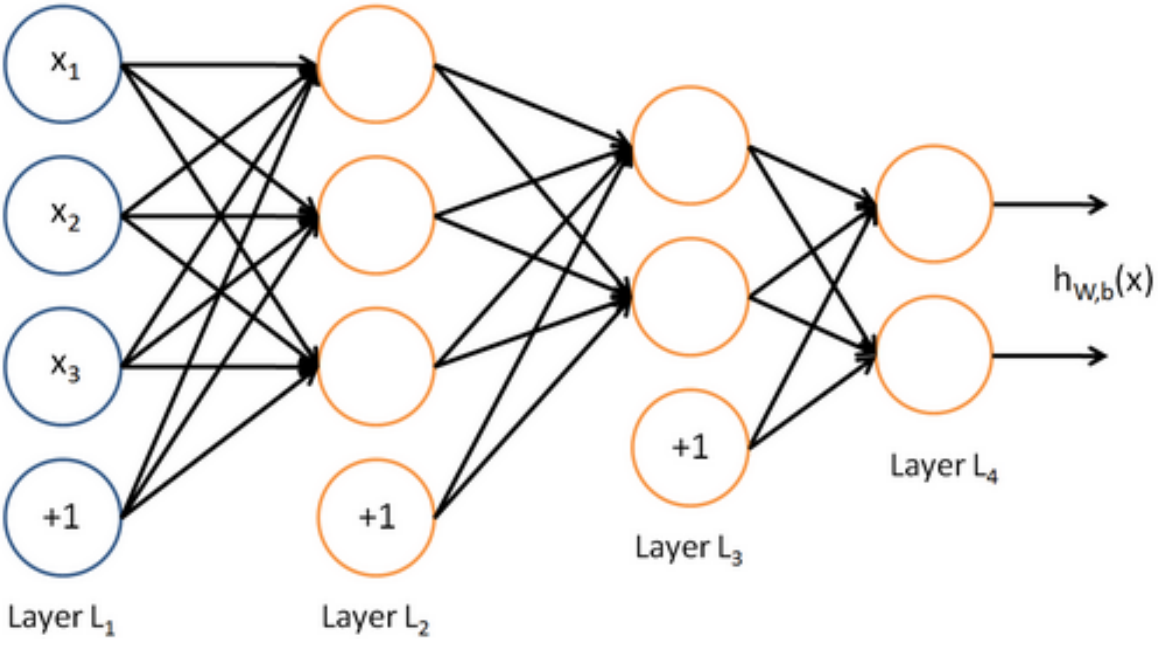


It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

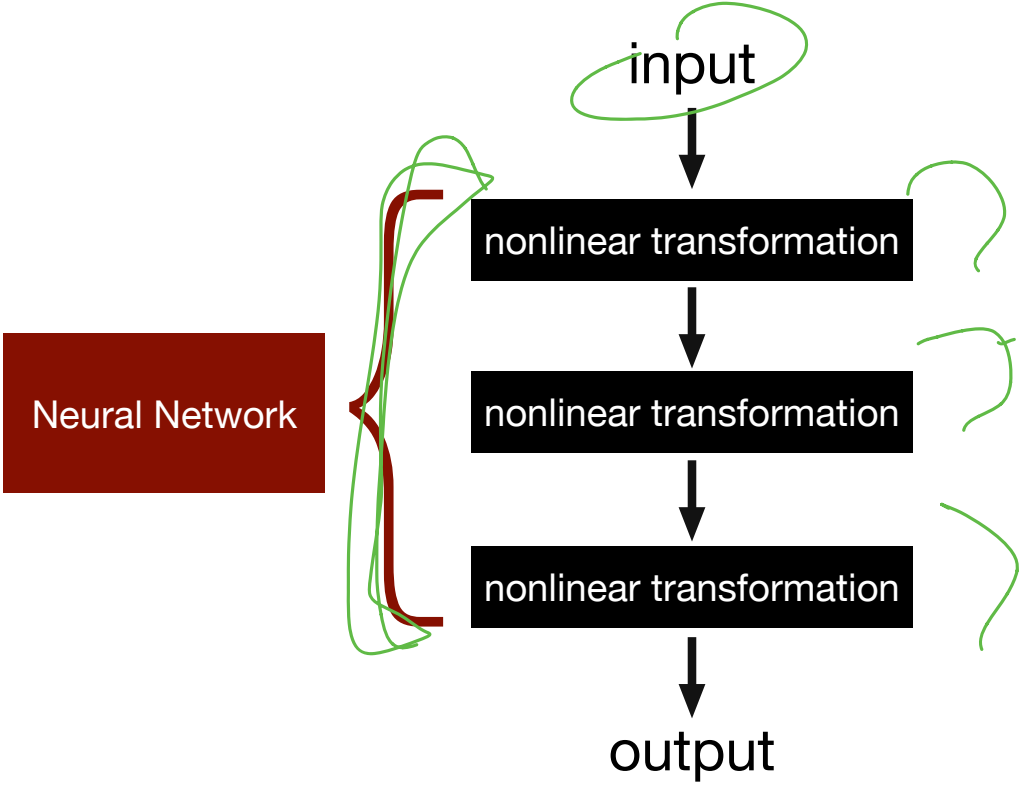
NN: kind of like several intermediate logregs

Before we know it, we have a multilayer neural network….

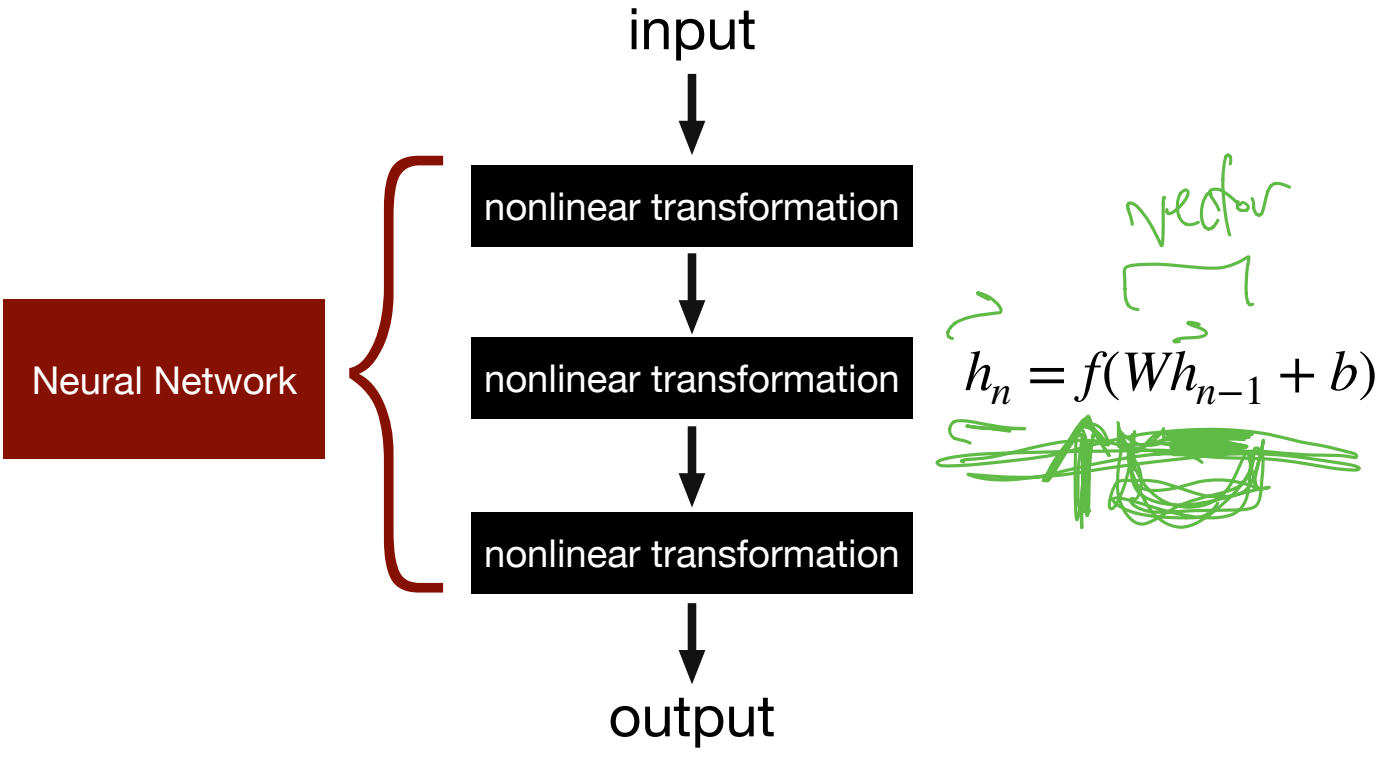
a.k.a. **feedforward network** (see INLP on terminology)



what is **deep learning**?

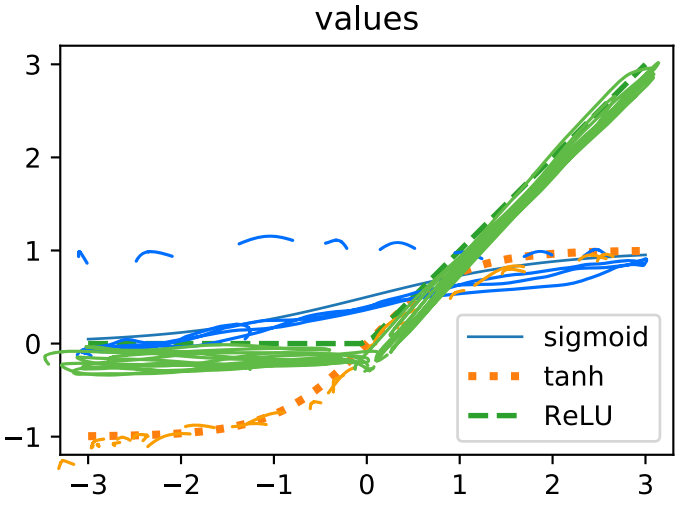


what is **deep learning**?



Nonlinear activations

- "Squash functions"!



- Logistic / Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

- tanh

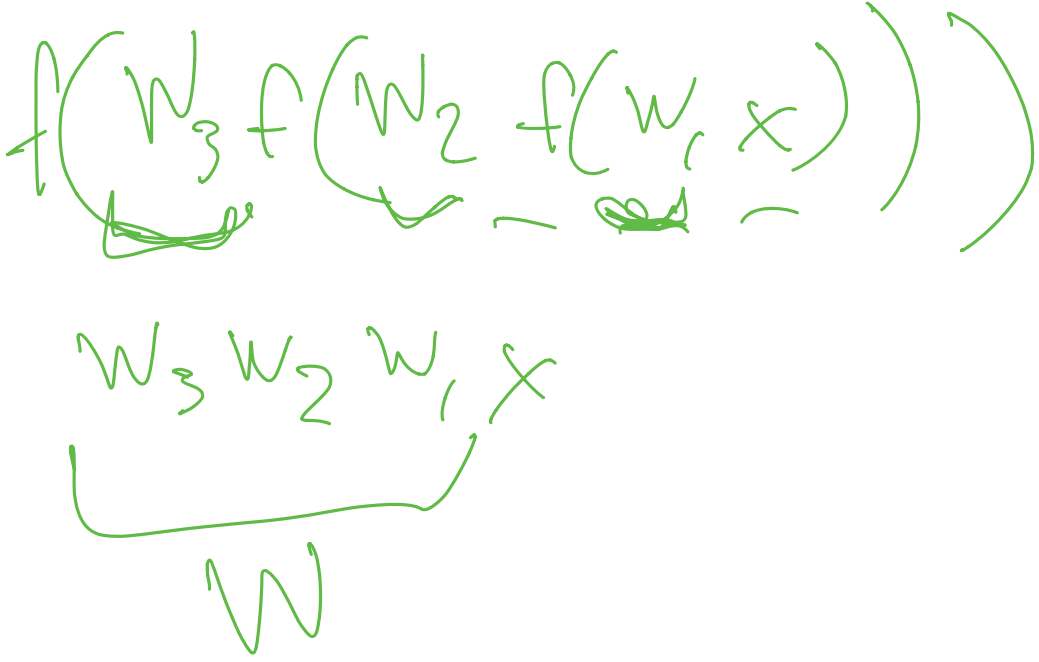
$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2}$$

- ReLU

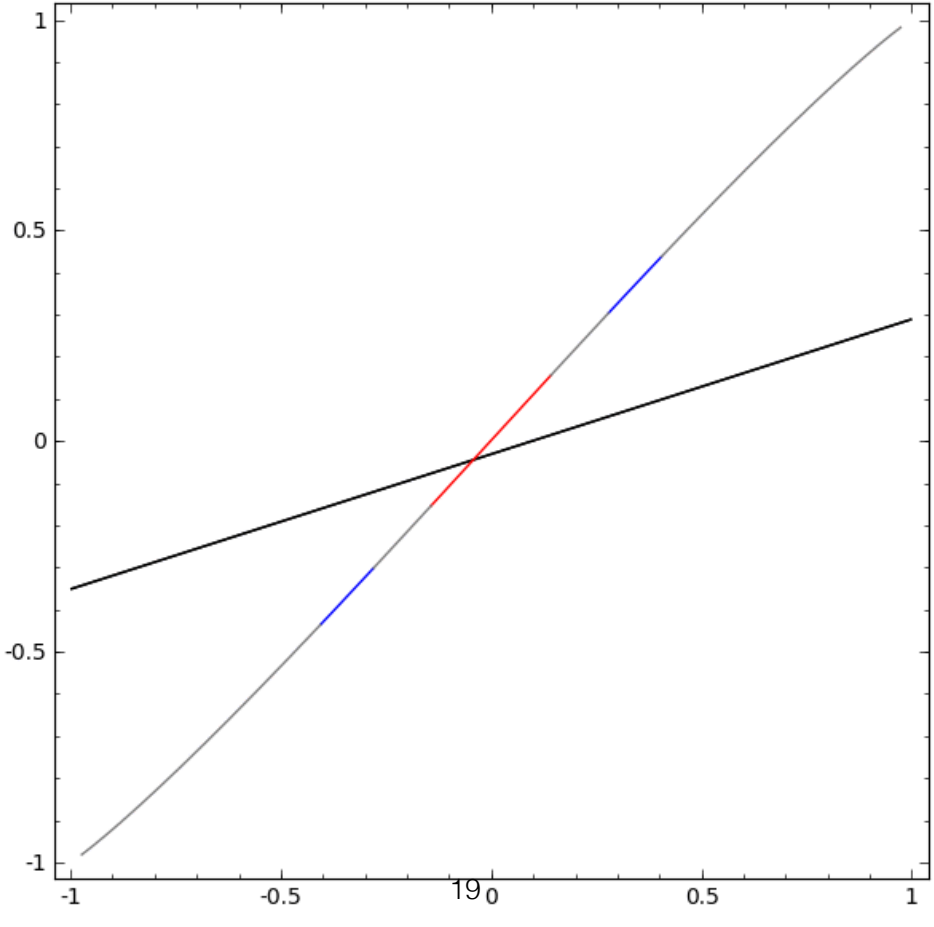
$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{3}$$

"Softplus"

is a multi-layer neural network with no nonlinearities
(i.e., f is the identity $f(\mathbf{x}) = \mathbf{x}$)
more powerful than a one-layer network?

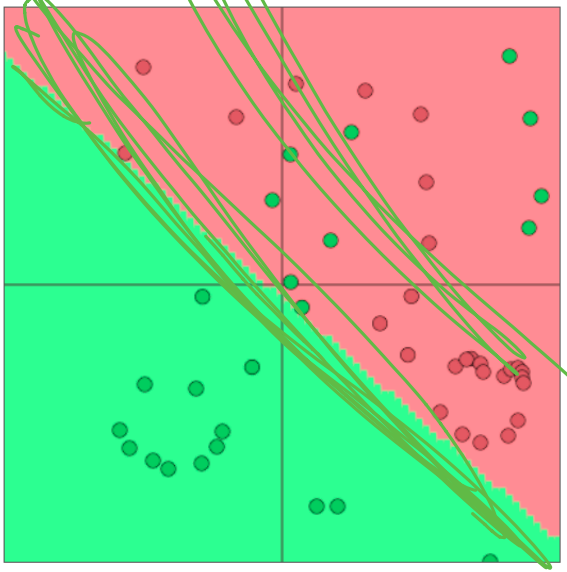


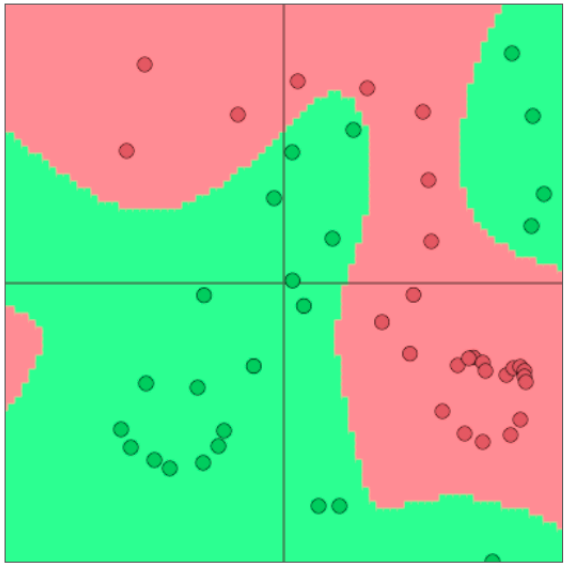
why nonlinearities?



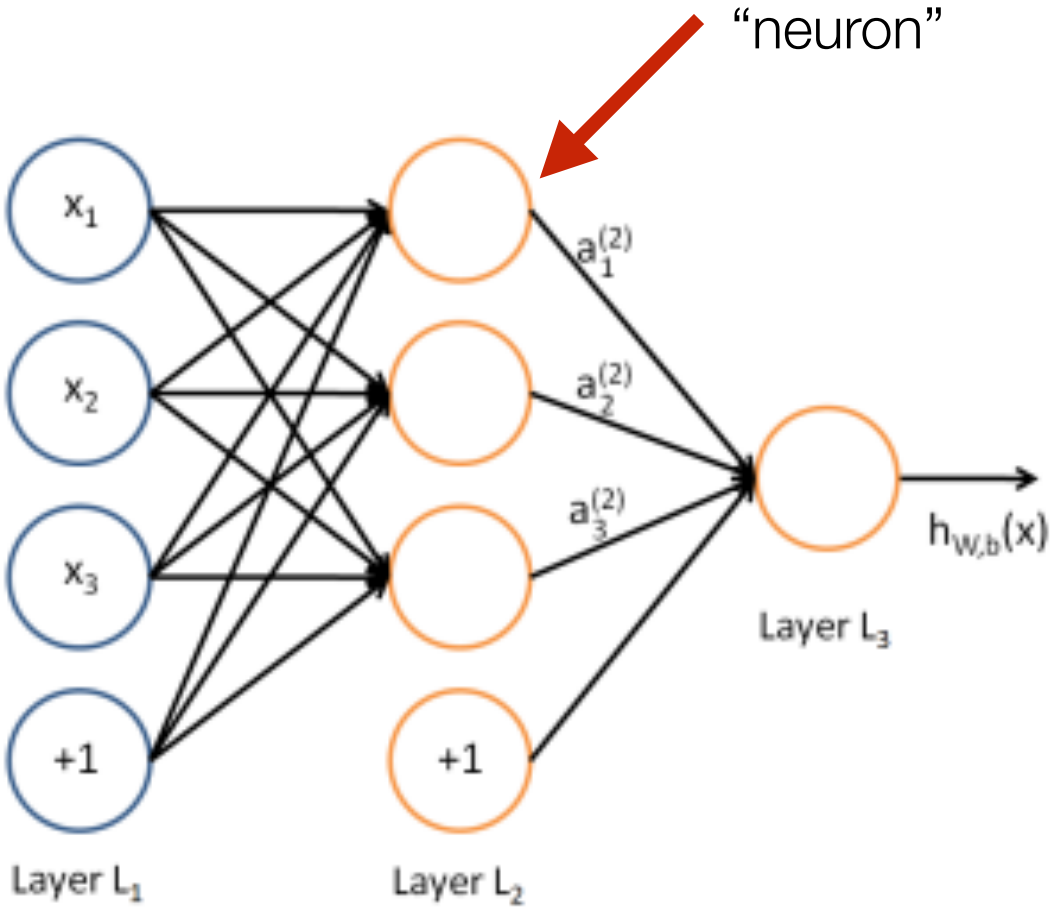
credit for figure:
Christopher Olah

why nonlinearities?

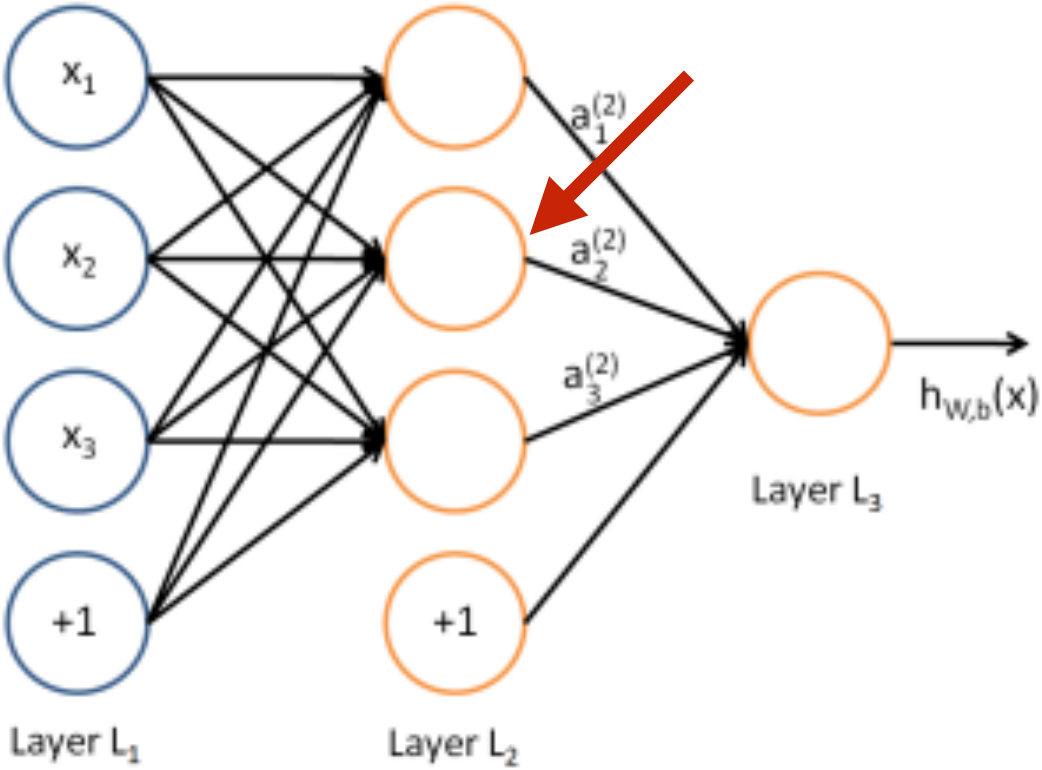




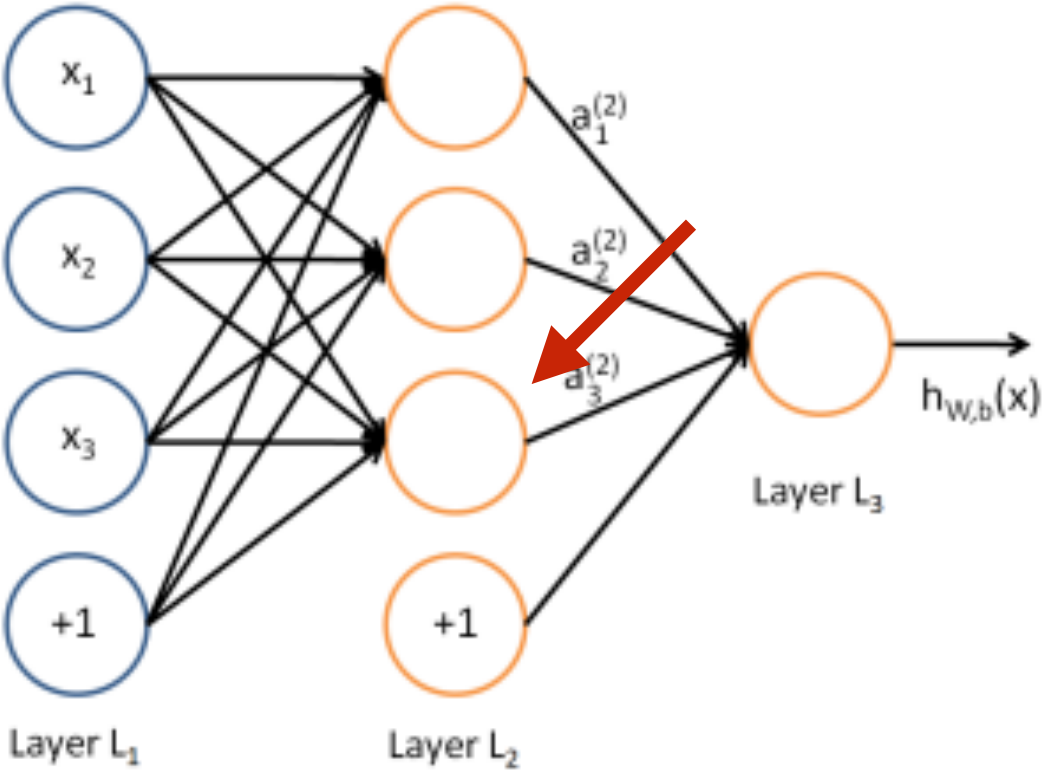
Tensorflow Playground



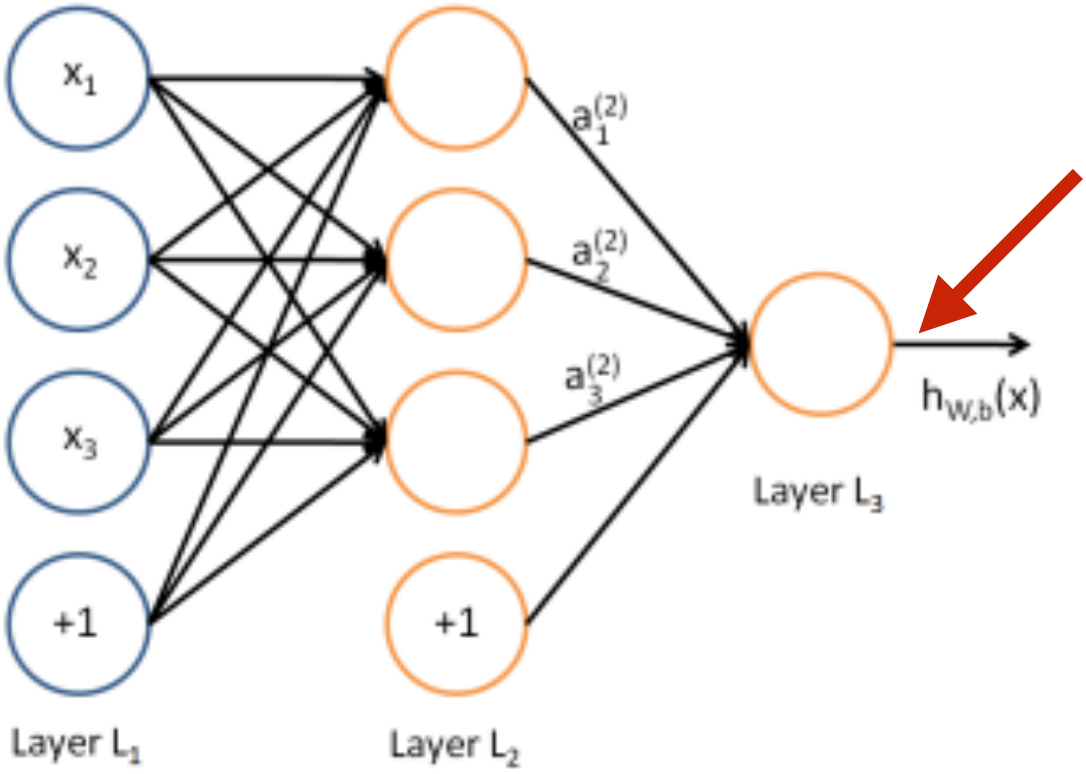
$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$



$$a_2^{(2)} = f\left(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}\right)$$

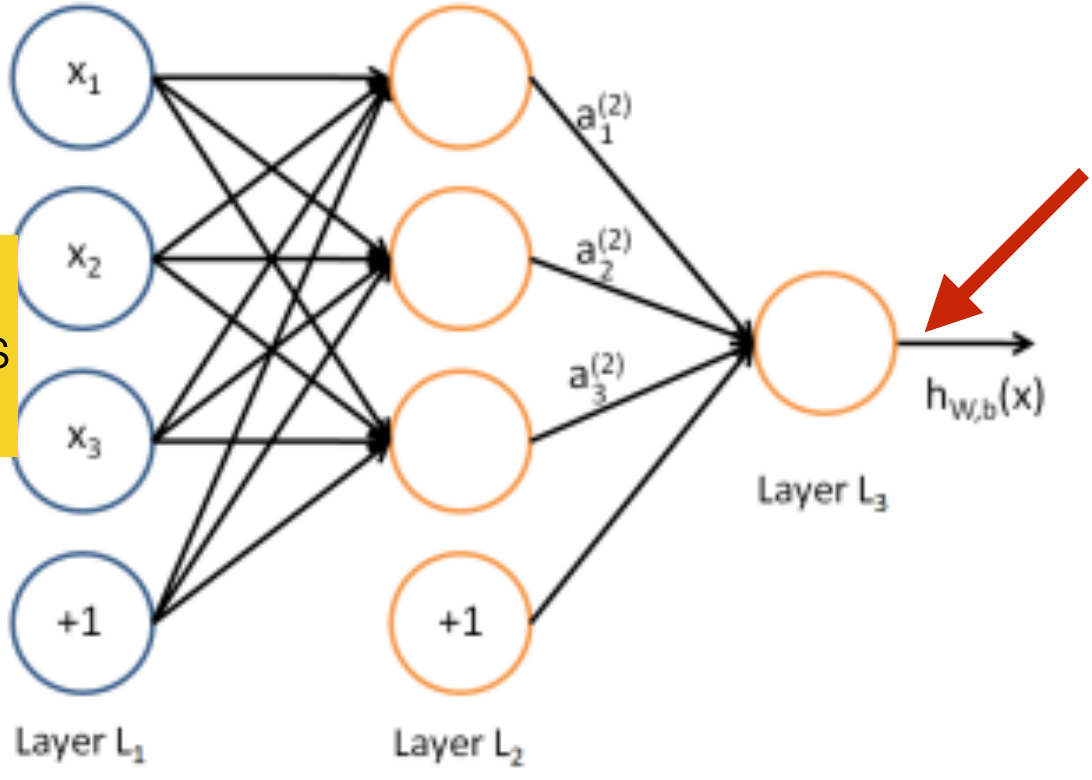


$$a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}\right)$$



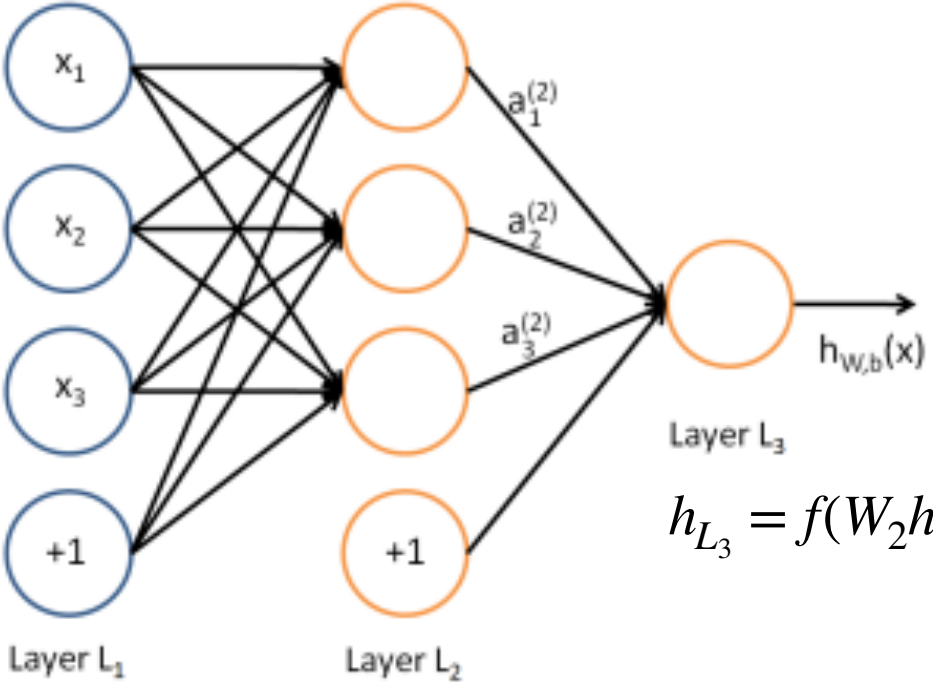
$$h_{W,b}(x) = a_1^{(3)} = f\left(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)}\right)$$

we will be learning the x's and the W's!



$$h_{W,b}(x) = a_1^{(3)} = f\left(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)}\right)$$

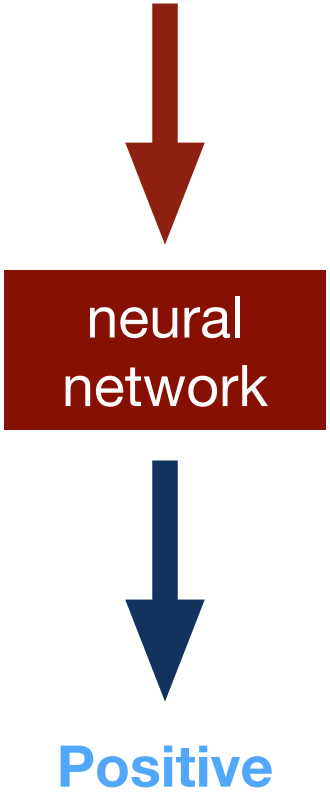
in matrix-vector notation…



$$h_{L_3} = f(W_2 h_{L_2} + b)$$

$$h_{L_2} = f(W_1 x + b)$$

Dracula is a really good book!



softmax function

- let's say I have 3 classes (e.g., positive, neutral, negative)
- use multiclass logreg with "cross product" features between input vector **x** and 3 output classes. for every class *c*, i have an associated weight vector β_c , then

$$P(y = c \mid \mathbf{x}) = \frac{e^{\beta_c \mathbf{x}}}{\sum_{k=1}^{3} e^{\beta_k \mathbf{x}}}$$

softmax function

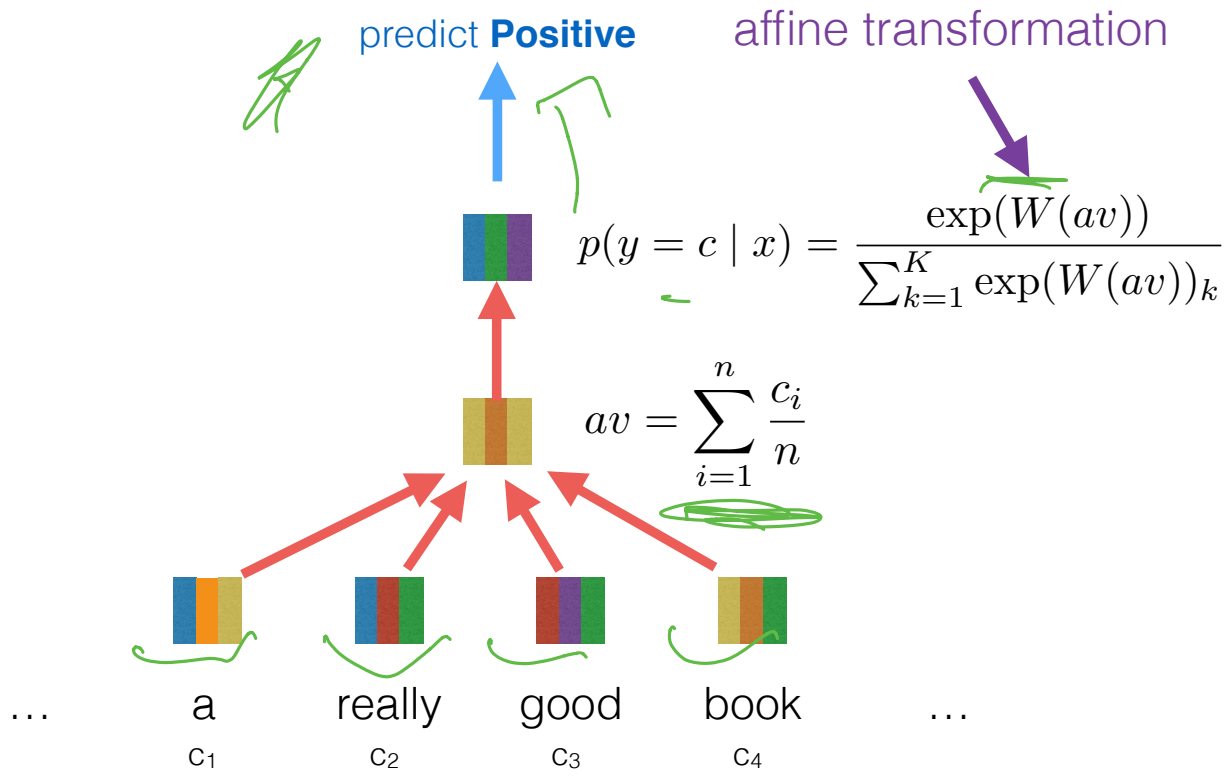
$$\text{softmax}(x) = \frac{e^{x}}{\sum_{j} e^{x_j}}$$

x is a vector

x_j is dimension j of x

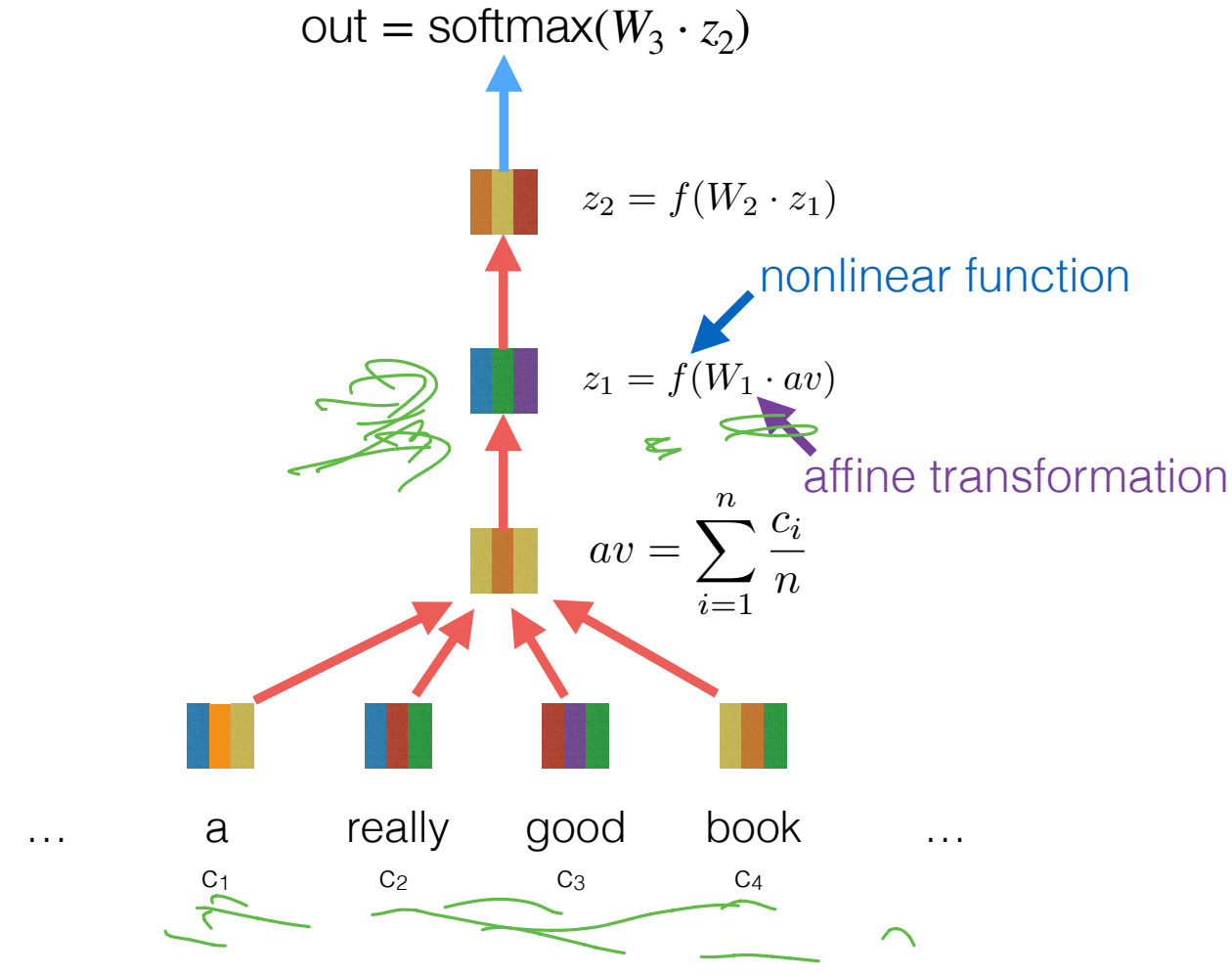
each dimension j of the softmaxed output represents the probability of class j

“bag of embeddings”



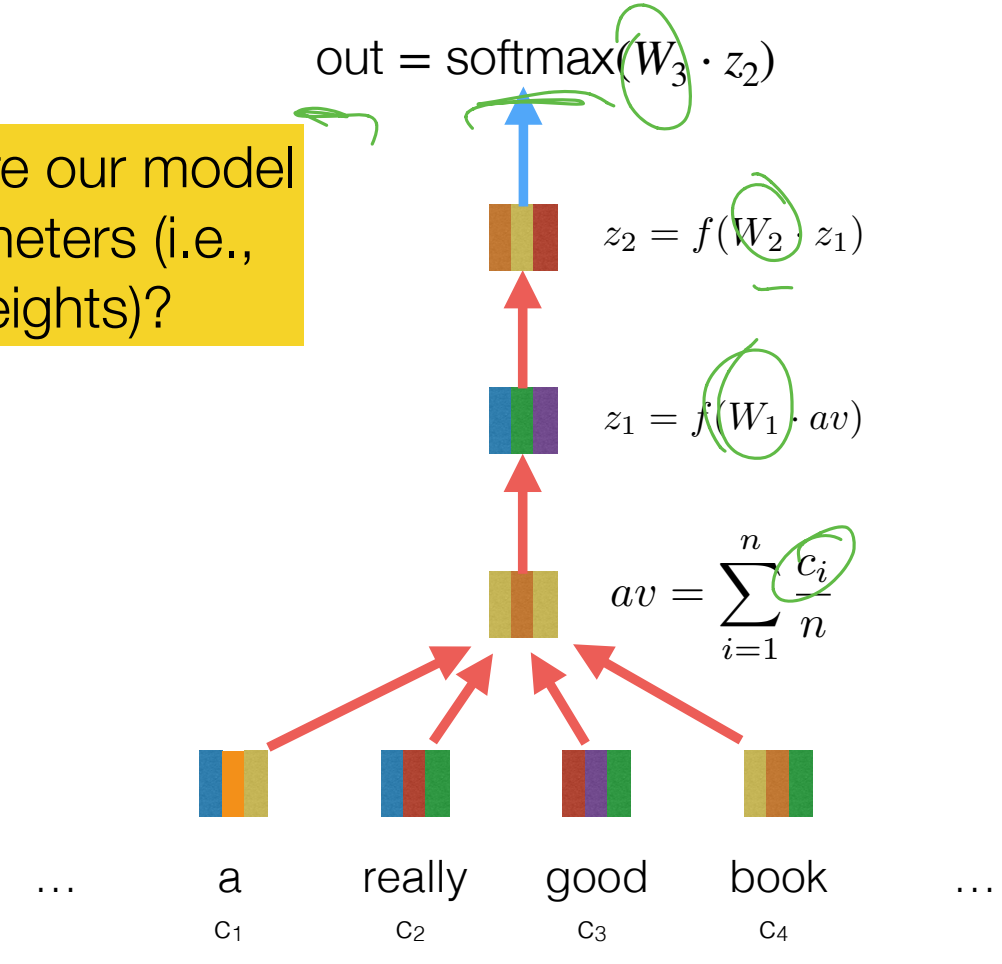
Iyyer et al., ACL 2015

deep averaging networks

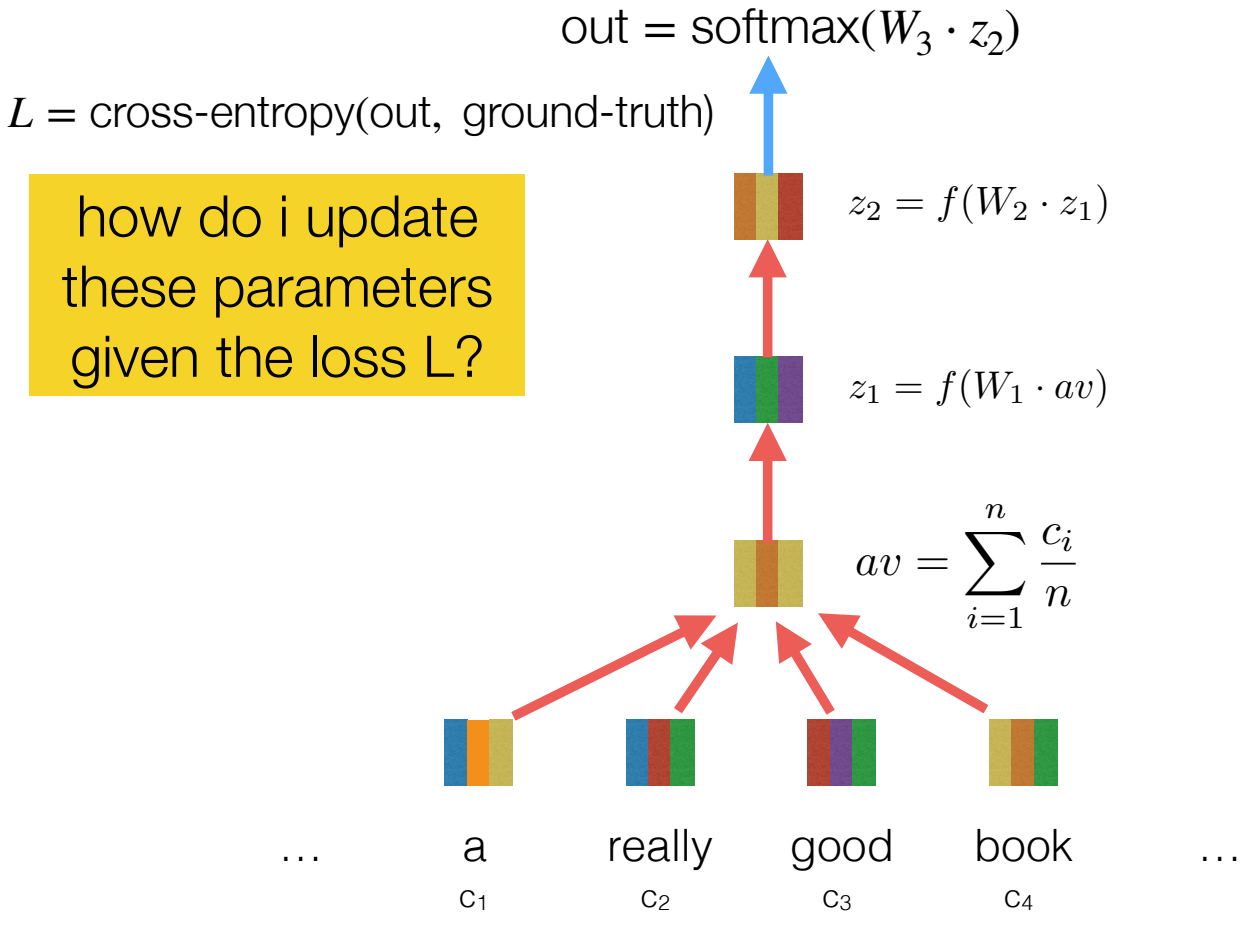


deep averaging networks

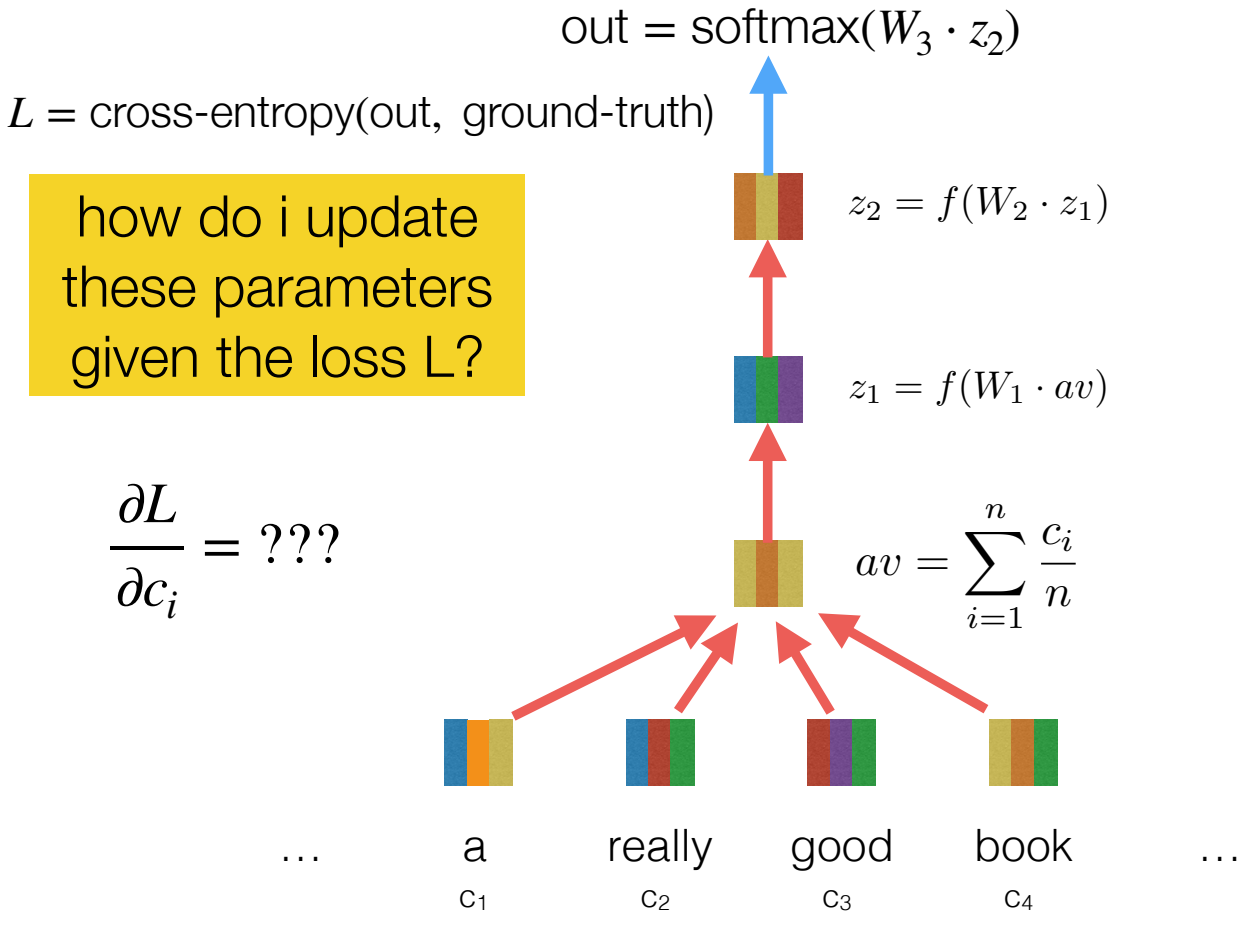
what are our model parameters (i.e., weights)?



deep averaging networks



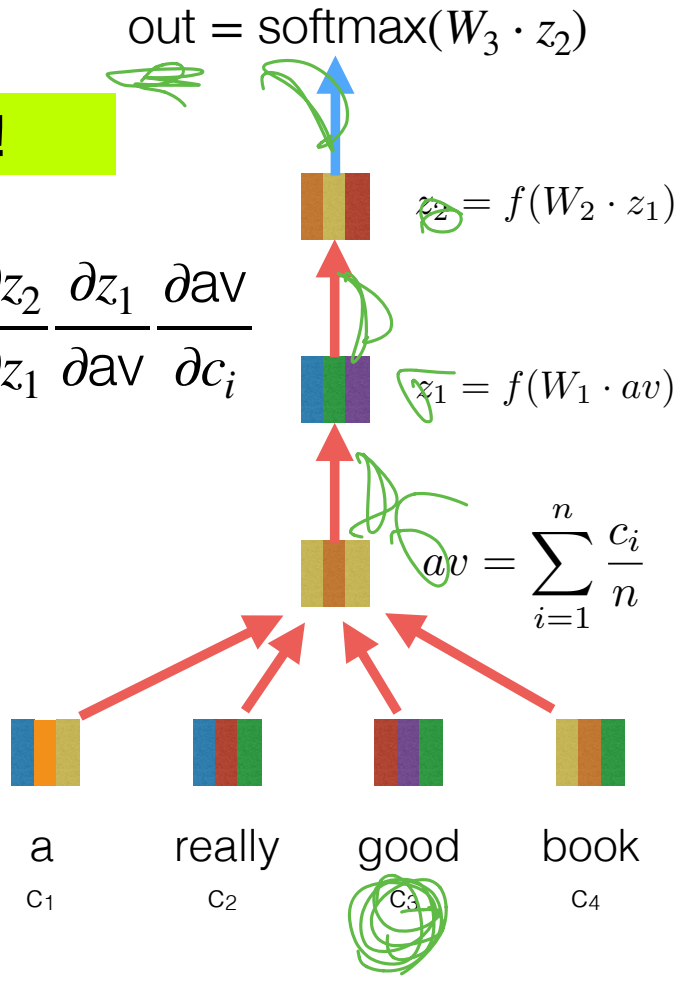
deep averaging networks



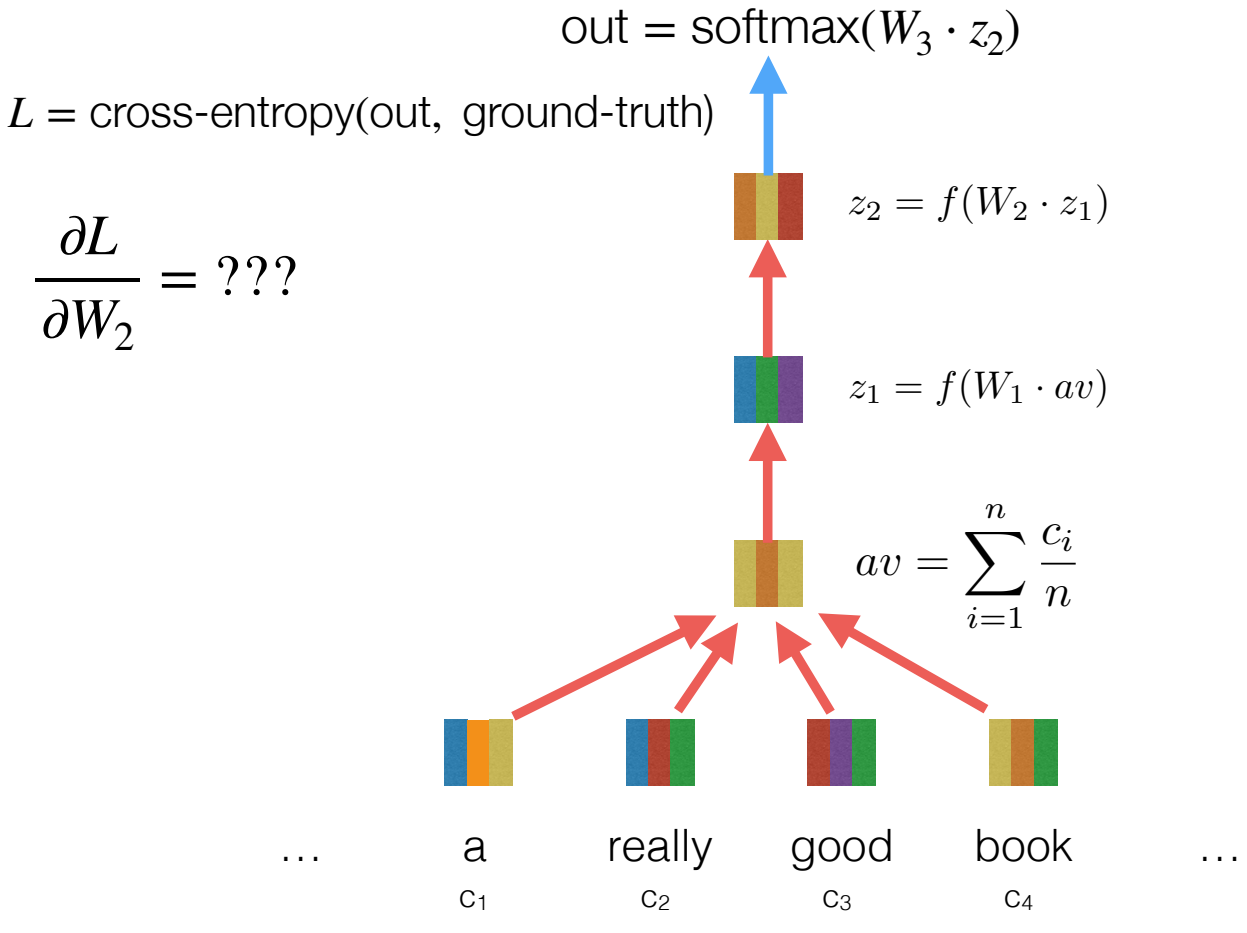
deep averaging networks

chain rule!!!

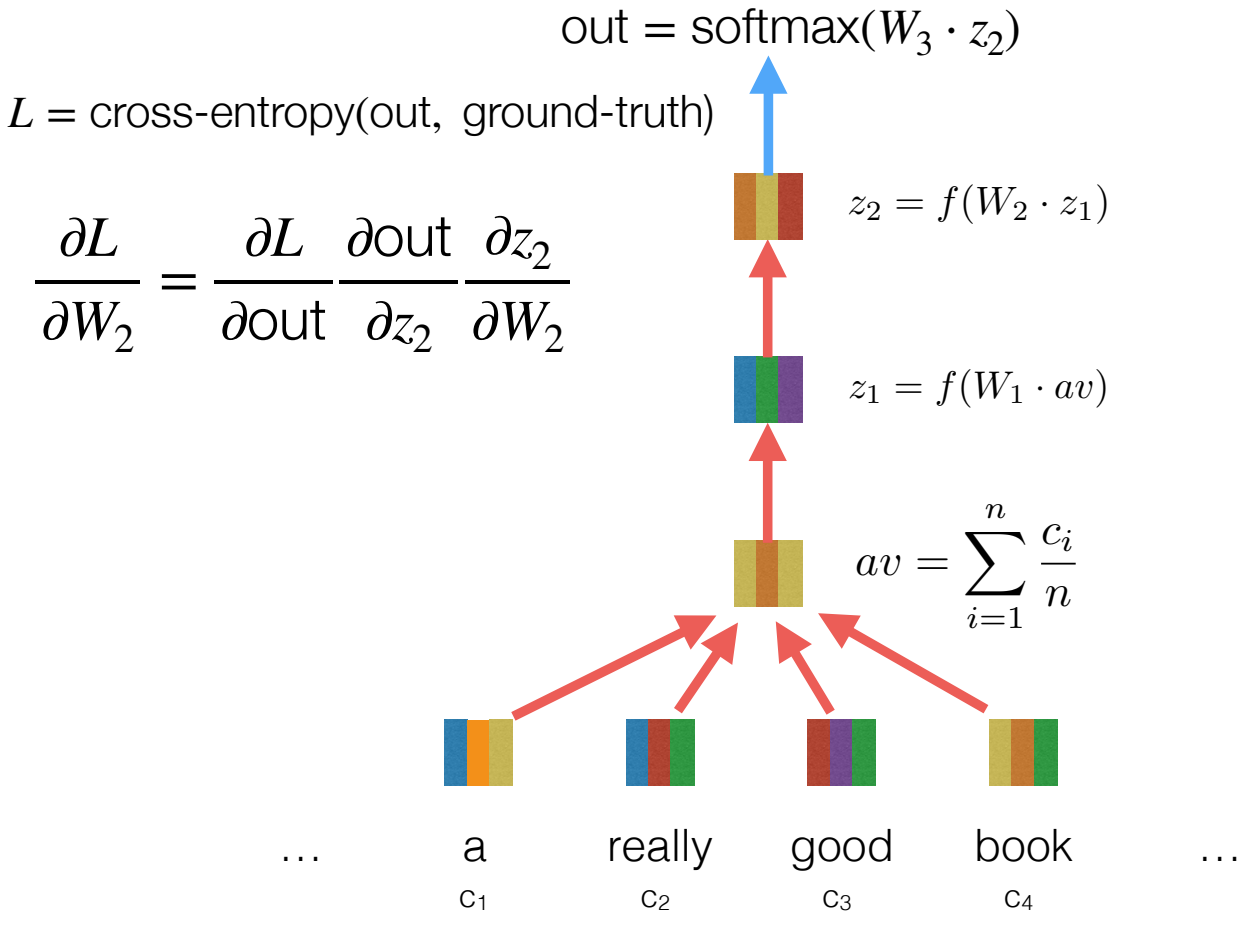
$$\frac{\partial L}{\partial c_i} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial \text{av}} \frac{\partial \text{av}}{\partial c_i}$$



deep averaging networks



deep averaging networks



backpropagation

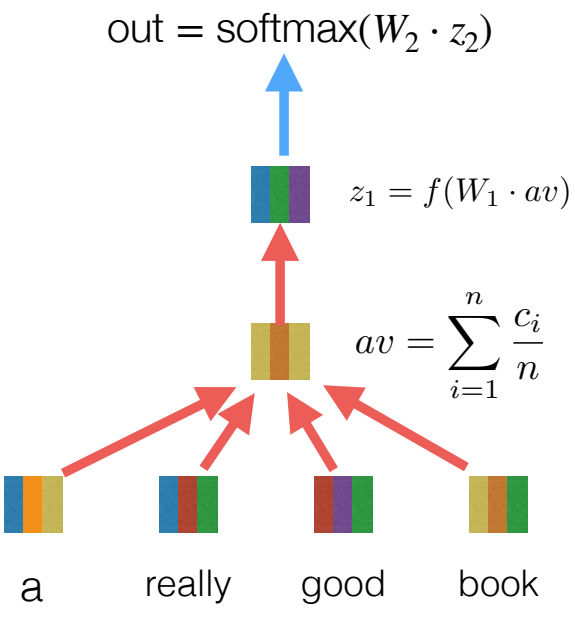
- use the chain rule to compute partial derivatives w/ respect to each parameter
- trick: re-use derivatives computed for higher layers to compute derivatives for lower layers!

$$\frac{\partial L}{\partial c_i} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial z_1} \frac{\partial z_1}{\partial \text{av}} \frac{\partial \text{av}}{\partial c_i}$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial z_2} \frac{\partial z_2}{\partial W_2}$$

Rumelhart et al., 1986

deep learning frameworks make building NNs super easy!

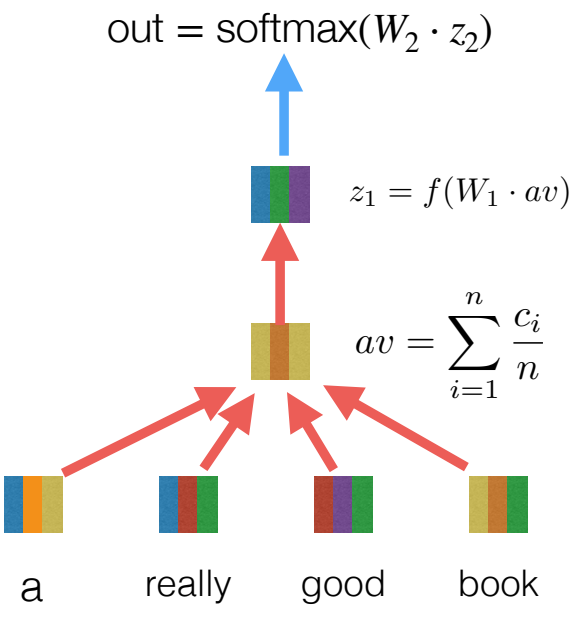


set up the network

```
def __init__(self, n_classes, vocab_size, emb_dim=300,
             n_hidden_units=300):
    super(DanModel, self).__init__()
    self.n_classes = n_classes
    self.vocab_size = vocab_size
    self.emb_dim = emb_dim
    self.n_hidden_units = n_hidden_units
    self.embeddings = nn.Embedding(self.vocab_size,
                                   self.emb_dim)
    self.classifier = nn.Sequential(
          nn.Linear(self.n_hidden_units,
                    self.n_hidden_units),
          nn.ReLU(),
          nn.Linear(self.n_hidden_units,
                    self.n_classes))
    self._softmax = nn.Softmax()
```

deep learning frameworks make building NNs super easy!

PyTorch

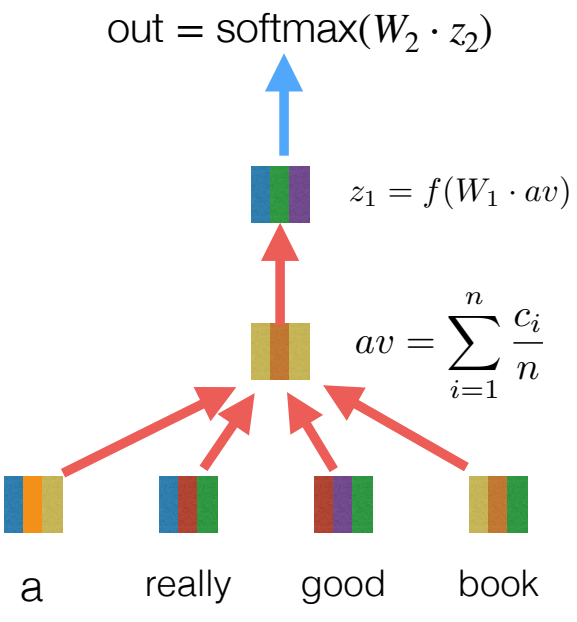


do a forward pass to compute prediction

```
def forward(self, batch, probs=False):
    text = batch['text']['tokens']
    length = batch['length']
    text_embed = self._word_embeddings(text)
    # Take the mean embedding. Since padding results
    # in zeros its safe to sum and divide by length
    encoded = text_embed.sum(1)
    encoded /= lengths.view(text_embed.size(0), -1)

    # Compute the network score predictions
    logits = self.classifier(encoded)
    if probs:
        return self._softmax(logits)
    else:
        return logits
```

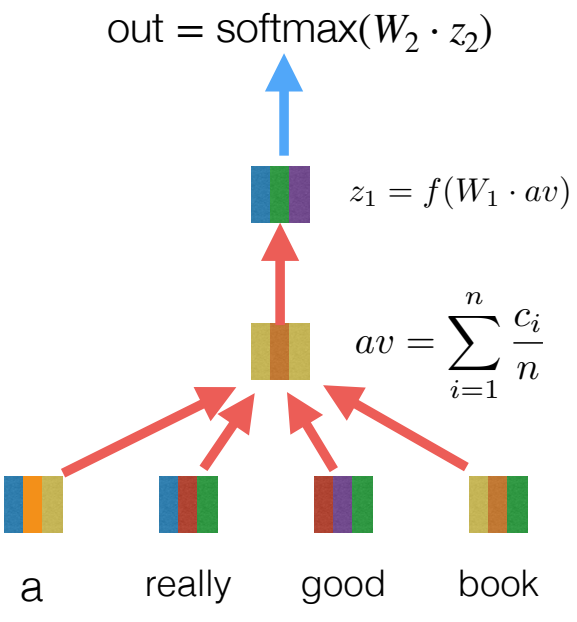
deep learning frameworks make building NNs super easy!



do a backward pass to update weights

```
def _run_epoch(self, batch_iter, train=True):
    self._model.train()
    for batch in batch_iter:
        model.zero_grad()
        out = model(batches)
        batch_loss = criterion(out,
                               batch['label'])
        batch_loss.backward()
        self.optimizer.step()
```

deep learning frameworks make building NNs super easy!



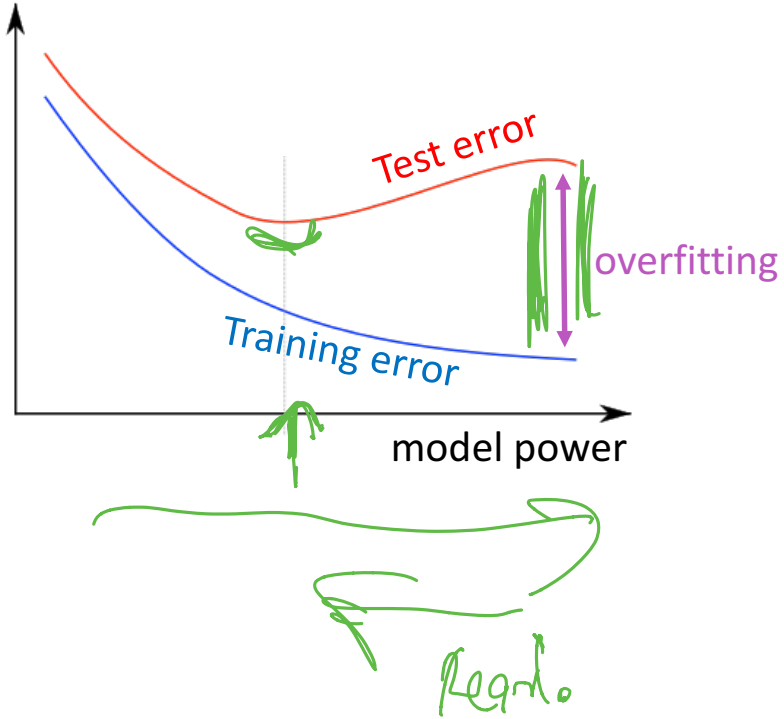
do a backward pass to update weights

```
def _run_epoch(self, batch_iter, train=True):
    self._model.train()
    for batch in batch_iter:
        model.zero_grad()
        out = model(batches)
        batch_loss = criterion(out,
                               batch['label'])
        batch_loss.backward()
        self.optimizer.step()
```

that's it! no need to compute gradients by hand!

Regularization

- Regularization prevents overfitting when we have a lot of features (or later a very powerful/deep model,++)



L2 regularization

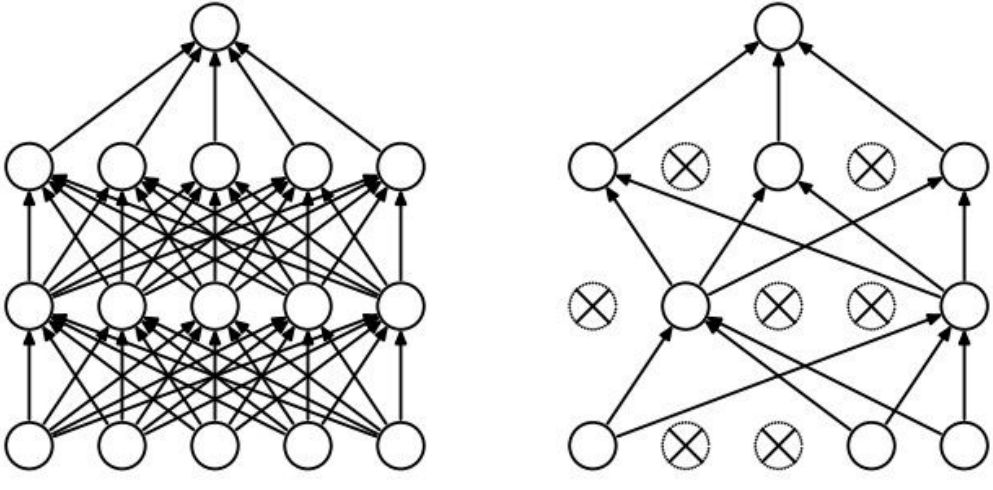
$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) + \lambda \sum_k \theta_k^2$$

θ represents all of the model's parameters!

penalizing their norm leads to smaller weights
we are constraining the parameter space
we are putting a prior on our model

Dropout for NNs

randomly set p% of neurons to 0 in the forward pass



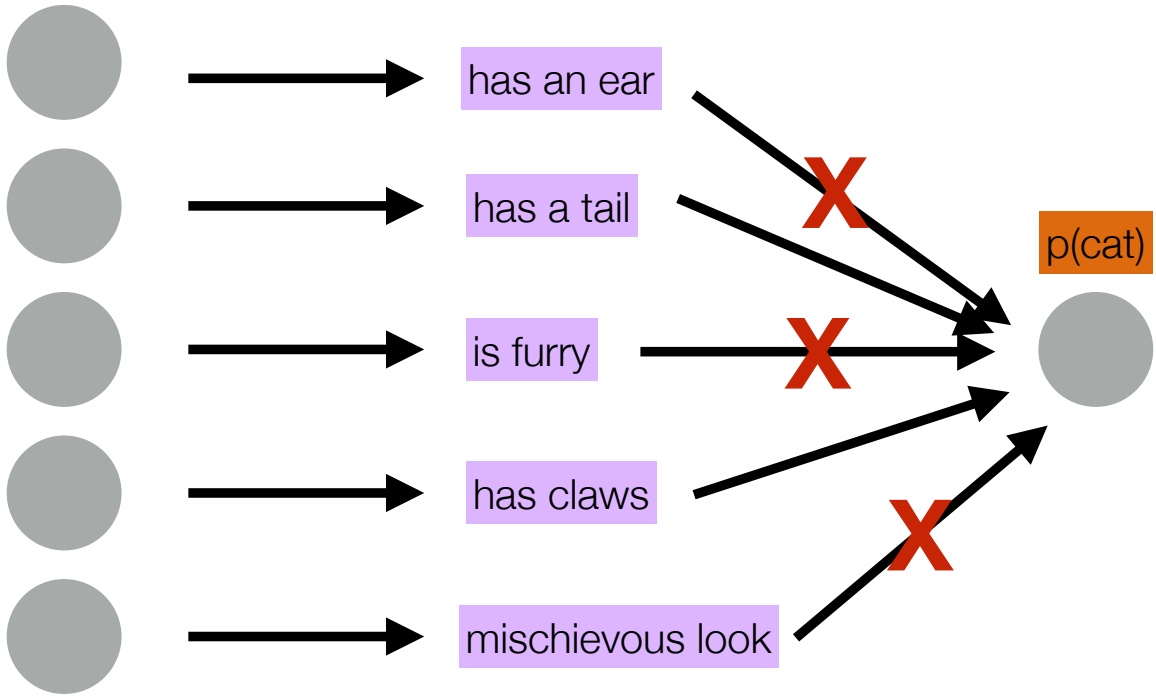
(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

Why?

randomly set *p*% of neurons to 0 in the forward pass

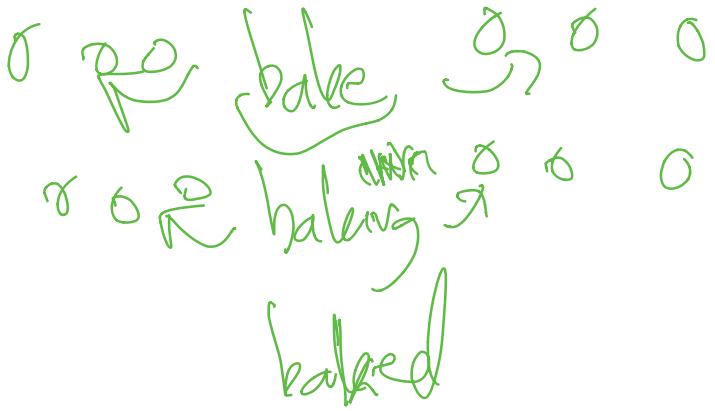


 network can't just rely on one neuron!

Addressing instability

- Training can be unstable! Therefore some tricks.
  - Initialization — random small but reasonable values can help.
  - Layer normalization (very important for some recent architectures)
- Since performance variance is high, you need to evaluate *multiple runs*
  - whether you're averaging or taking max performance
  - esp for comparisons!

- A few unresolved questions about NNs in NLP
  - Useful architectures?
    - Many: Convolutional, Recurrent, Self/cross-attention
  - Modular systems?
  - Interpretability / explainability?
  - Incorporate prior knowledge?
  - Transferring information across datasets/ languages/etc?
- These are major questions for NLP modeling right now!


bake
baking
baked