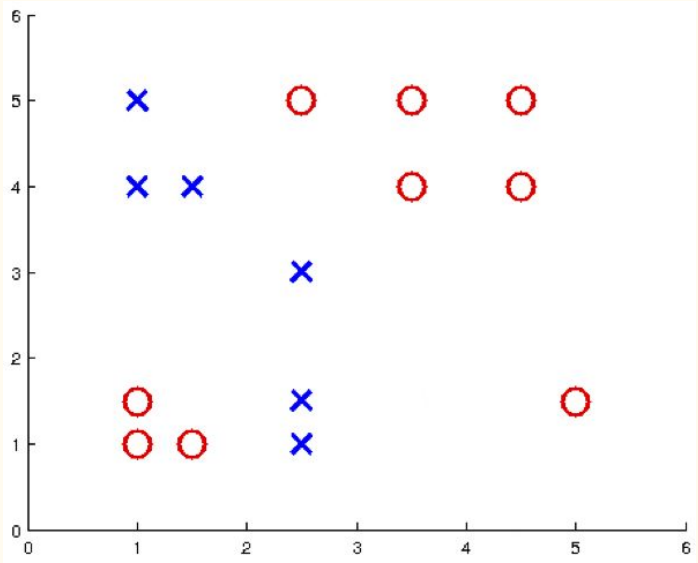
L101: Feed Forward Neural Networks

—

Linear classifiers

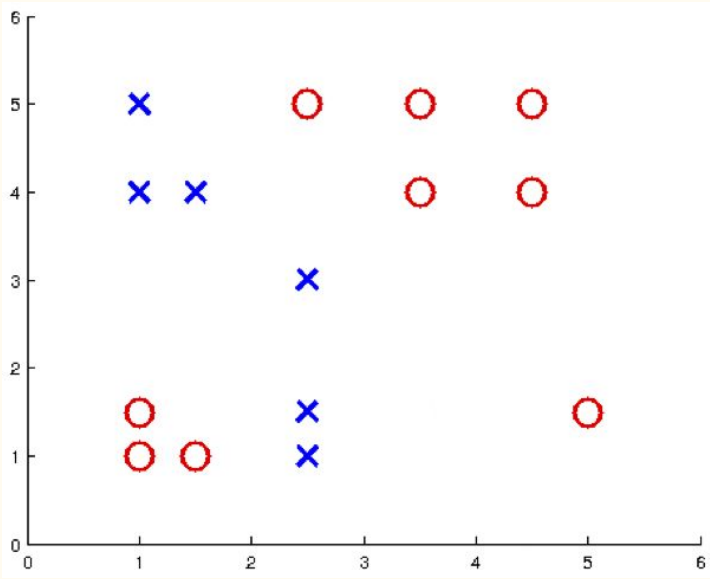
e.g. binary logistic regression: $P(\hat{y} = 1) = \sigma(w \cdot \phi(x))$

And their limitations:



http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html

What if we could use multiple classifiers?



Decompose predicting red vs blue in 3 tasks:

- top-right red circles vs. rest
- bottom-left red circles vs. rest
- If one of the above is red circle, then it is red circle, otherwise blue cross

Transform non-linearly into linearly separable!

Feed forward neural networks

More concretely:

$$h_1 = f_1(x) = \sigma(w_1 x + b_1)$$
$$h_1 = f_2(x) = \sigma(w_2 x + b_2)$$
$$P(\hat{y} = 1) = \sigma(w \cdot [h_1; h_2] + b)$$

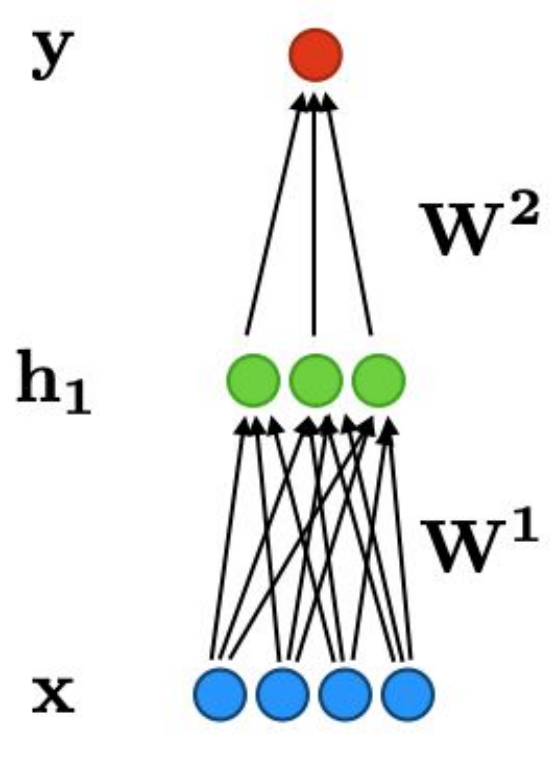
Terminology: input units x, hidden units h

Can think of the hidden units as learned features

More compactly for k layers :

$$h^1 = \sigma(W^1 x + b^1)$$
$$\cdots$$
$$P(\hat{y} = 1) = \sigma(W^k \cdot h^{k-1} + b^k)$$

Feed forward neural networks: Graphical view

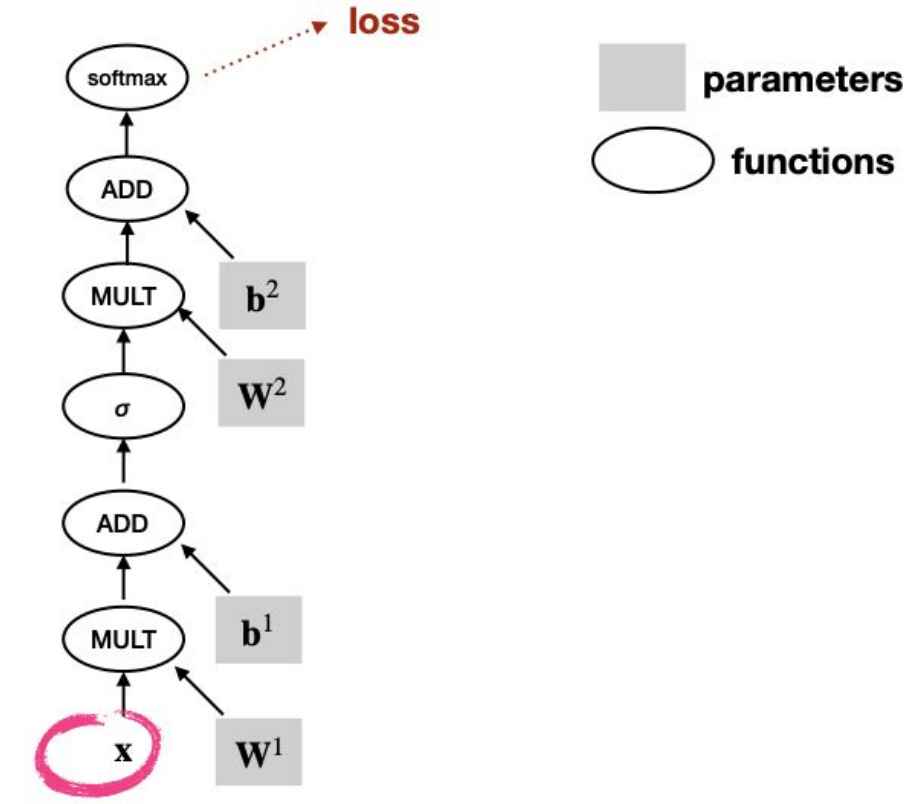


Feedforward: no cycles, the information flows forwards

Fully connected layers

Barbara Plank (AthNLP lecture)

Computation Graph view



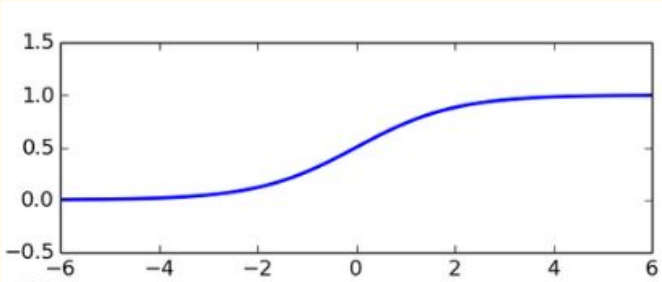
It is useful when differentiating and/or optimizing the code for speed

What should input x be for text classification?

Word embeddings!

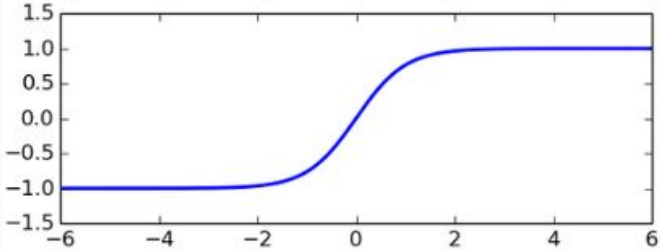
Barbara Plank (AthNLP lecture)

Activation functions



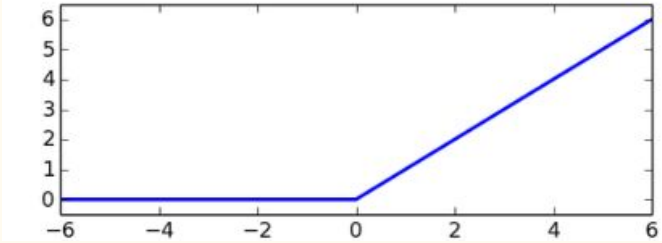
Sigmoid

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



Hyperbolic Tangent

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



Rectified Linear

$$\phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

Non-linearity is key: without it we still do linear classification

Multilayer perceptron is a misnomer

Hughes and Correll (2016)

How to learn the parameters?

Supervised learning! Given labeled training data of the form:

$$D = \{(x^1, y^1), \ldots (x^M, y^M)\}$$

Optimize the Negative Log-Likelihood, e.g. with gradient descent:

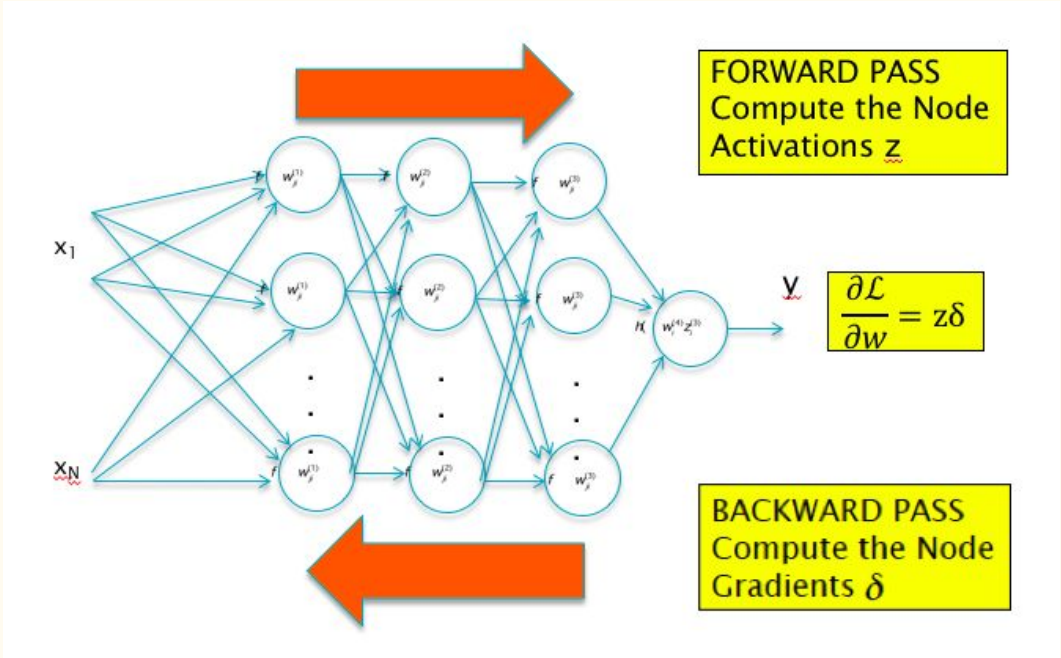
$$NLL(\hat{y}, y) = -y \log P(\hat{y} = 1) - (1 - y) \log(1 - P(\hat{y} = 1))$$

What could go wrong?

We can only calculate the derivatives of the loss for the final layer, we do not know the correct values for the hidden ones. The latter with non-linear activations make the objective **non-convex.**

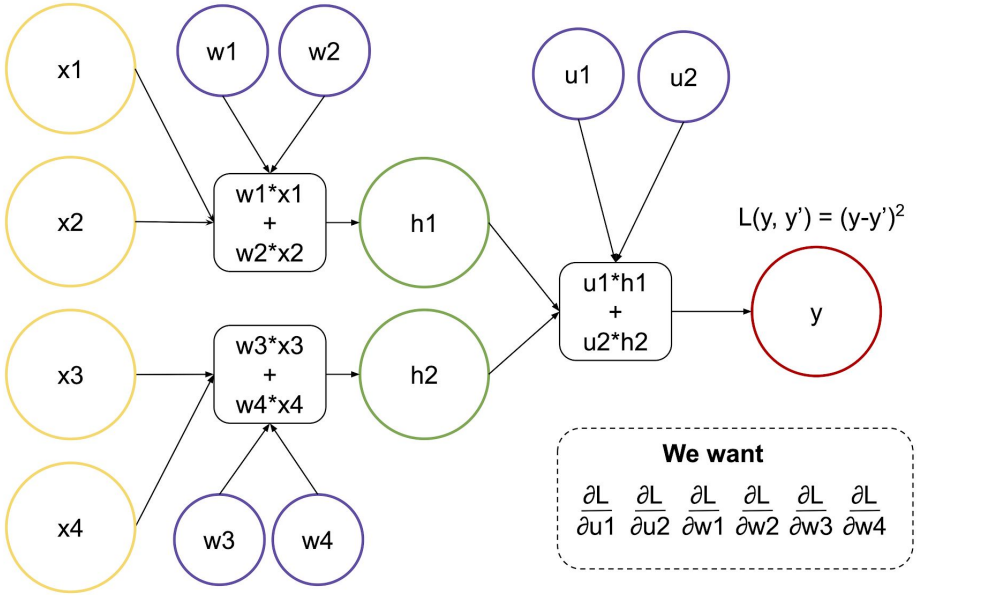
Backpropagation

We can obtain temporary values for the hidden layer and final loss (forward pass) and then calculate the gradients backwards:



https://srdas.github.io/DLBook/TrainingNNsBackprop.html

Backpropagation (toy example)



We want

$$\frac{\partial L}{\partial u1} \quad \frac{\partial L}{\partial u2} \quad \frac{\partial L}{\partial w1} \quad \frac{\partial L}{\partial w2} \quad \frac{\partial L}{\partial w3} \quad \frac{\partial L}{\partial w4}$$

All base derivatives

$$\frac{\partial L}{\partial y} = 2(y-y')$$

$$\frac{\partial y}{\partial h1} = u1 \qquad \frac{\partial y}{\partial h2} = u2$$

$$\frac{\partial y}{\partial u1} = h1 \qquad \frac{\partial y}{\partial u2} = h2$$

$$\frac{\partial h1}{\partial w1} = x1 \qquad \frac{\partial h1}{\partial w2} = x2$$

$$\frac{\partial h1}{\partial x1} = w1 \qquad \frac{\partial h1}{\partial x2} = w2$$

$$\frac{\partial h2}{\partial w3} = x3 \qquad \frac{\partial h1}{\partial w4} = x4$$

$$\frac{\partial h1}{\partial x3} = w3 \qquad \frac{\partial h1}{\partial x4} = w4$$

Full derivation examples

$$\frac{\partial L}{\partial u1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial u1} = 2(y-y')*h1$$

$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h1}\frac{\partial h1}{\partial w1} = 2(y-y')*u1*x1$$

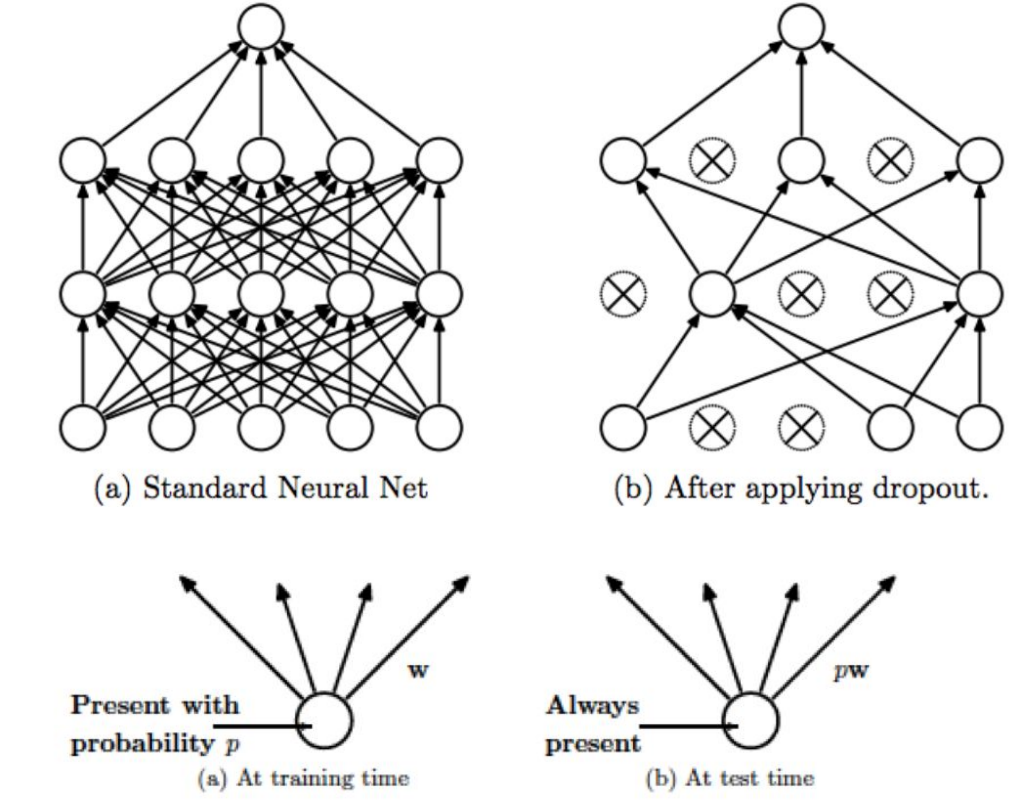
Ryan McDonald (AthNLP 2019)

Regularization

L2 is standard

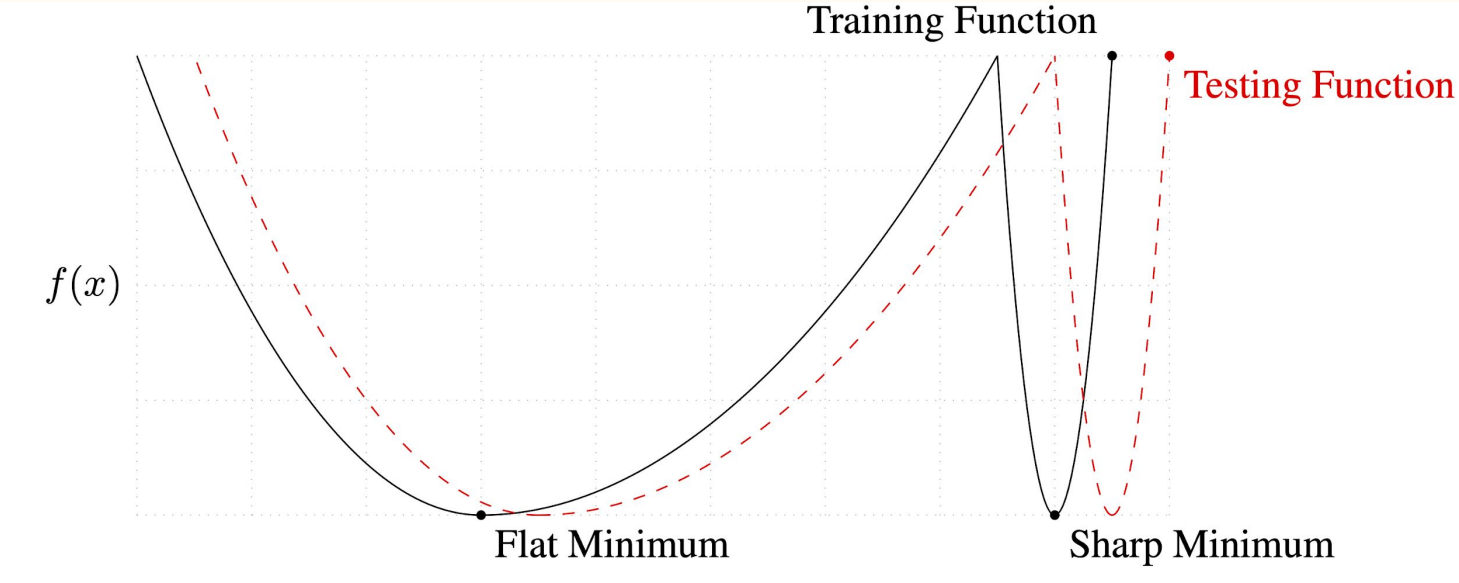
Early stopping based on validation error

Dropout (Srivastava et al., 2014): remove some connections (at random, different each time) in order to make the rest work harder



https://srdas.github.io/DLBook/ImprovingModelGeneralization.html#ImprovingModelGeneralization

Optimization



Noise from being stochastic in gradient descent can be beneficial as it avoid sharp local minima (Keskar et al. 2017)

Implementation

- Learning rates in (S)GD with backprop need to be small (we don't know the values for the hidden layer, we hallucinate them)
- Batching the data points allows us to be faster on GPUs
- Learning objective non-convex: initialization matters
  - Random restarts to escape local optima
  - When arguing for the superiority of an architecture, ensure it is not just the random seed (Reimers and Gurevych, 2017)
- Initialize with small non-zero values
- Greater learning capacity makes overfitting more likely: regularize

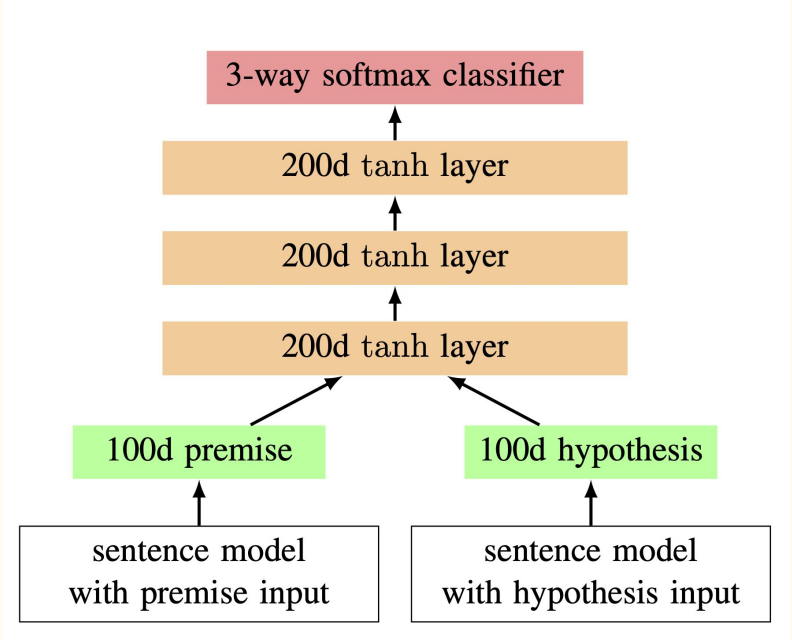
Let's try some of this

Sentence pair modelling

We can use FFNNs to perform tasks involving comparisons between two sentences, e.g. textual entailment: does the premise support the hypothesis?

Premise: Children smiling and waving at a camera
Hypothesis: The kids are frowning
Label: Contradiction

Well-studied task in NLP, was revolutionized



Bowman et al. (2015)

Interpretability

What do they learn?

Two families of approaches:

- Black box: alter the inputs to expose the learning, e.g. LIME
- White box: interpret the parameters directly, e.g. learn the decision tree
  - Alter the model to generate an explanation in natural language
  - Encourage parameters to be explanation-like

What is an explanation?

- Explains the model prediction well?
- What a human would have said to justify the label?

Why should we be excited about NNs?

Continuous representations help us achieve better accuracy

Open avenues to work on more tasks that were not amenable with discrete features:

- Multimodal NLP
- Multi-task learning

Pretrained word embeddings are the most successful semi-supervised learning method I know of (Turian et al., 2010)

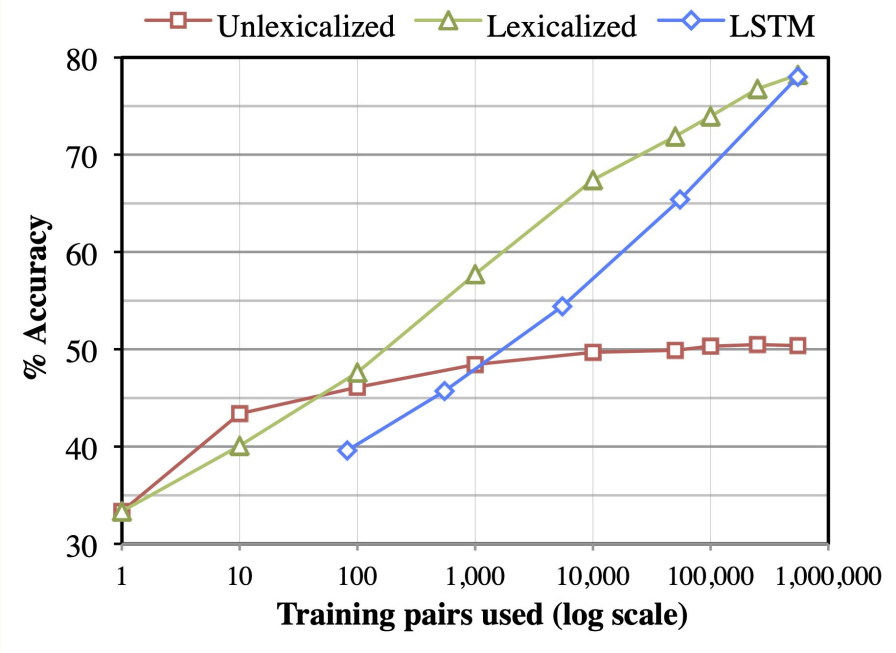
Why not be excited?

We don't quite understand them: arguments about architecture/regularization suitability to task do not seem to be tight

(the field is working on it)

Need for (more) data

Feature engineering is replaced by architecture engineering



Bowman et al. (2015)

What can we learn with FFNNs?

Universal approximation theorem tells us that with one hidden layer with enough capacity can represent any function (map between two spaces).

Then why do we design new architectures?

Being able to represent, doesn't mean able to **learn** the representation:

- Adding more hidden units becomes infeasible/impractical
- Optimization can find poor local optimum, or overfit

Different architectures can be better to learn with for different tasks/datasets

We can compress large trained models with simple ones, but not learn the simpler ones directly (Ba and Caruana, 2014)

Bibliography

A simple implementation in python of backpropagation

The tutorial of Quoc V. Le

A nice, full-fledged explanation of back-propagation

Similar material from an NLP perspective is covered in Yoav Goldberg's tutorial, sections 3-6

Chapter 6, 7 and 8 from Goodfellow, Bengio and Courville (2016) Deep Learning