

Feed Forward Neural Networks

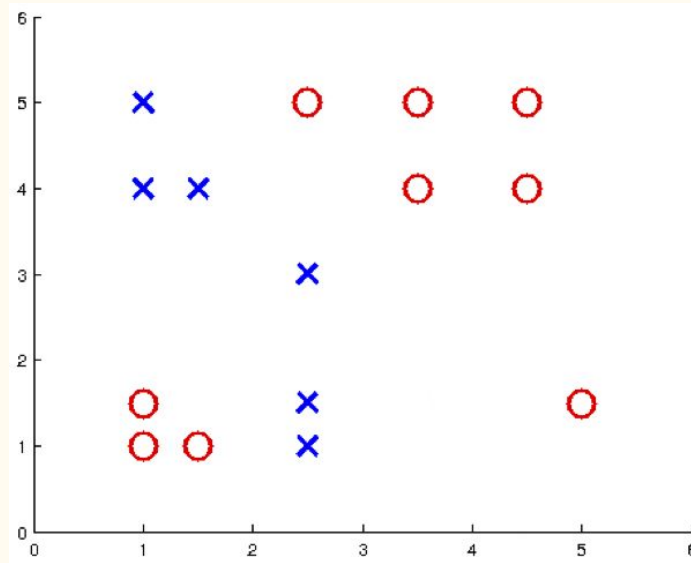
L101: Machine Learning for Language Processing
Andreas Vlachos



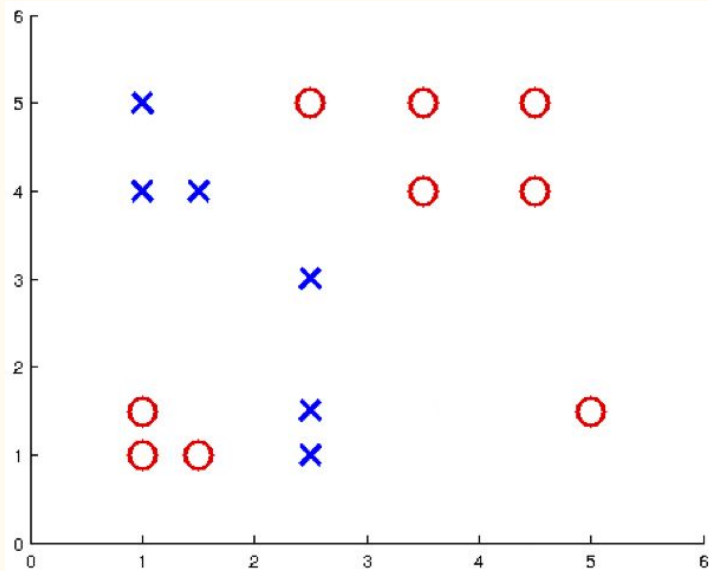
Linear classifiers

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

And their limitations:



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:

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

Terminology: input units x , hidden units h

Can think of the hidden units as learned features

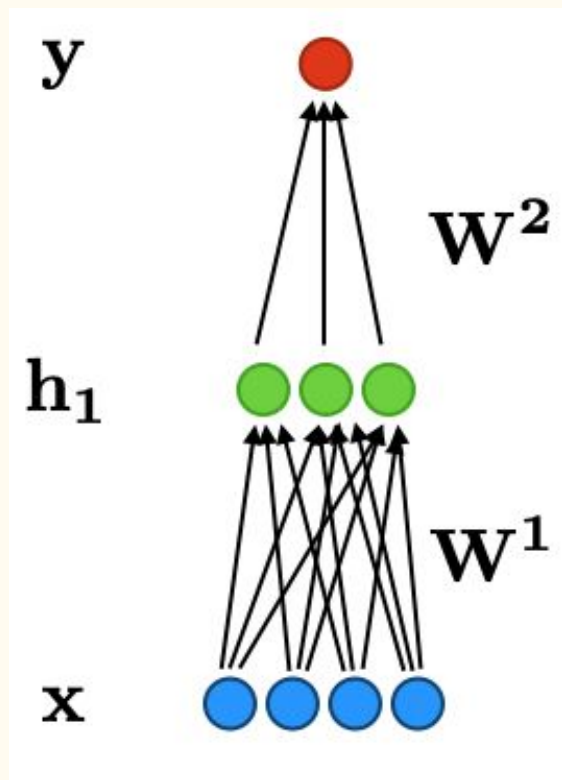
More compactly for

$$h^1 = \sigma(W^1 x + b^1)$$

k layers :

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

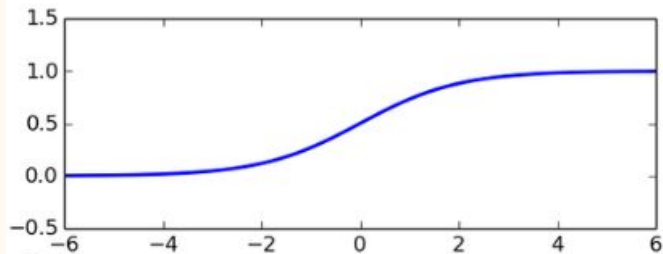
Feed forward neural networks: Graphical view



Feedforward: no cycles, the information flows forwards

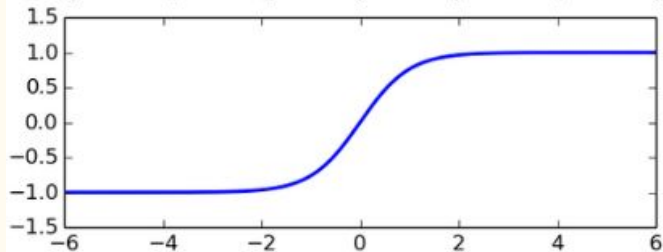
Fully connected layers

Activation functions



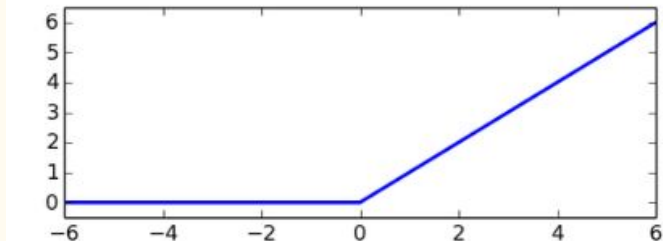
Sigmoid

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



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 bit of a misnomer

Hughes and Correll (2016)

How to learn the parameters?

Supervised learning! Given labeled training data of the form:

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

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

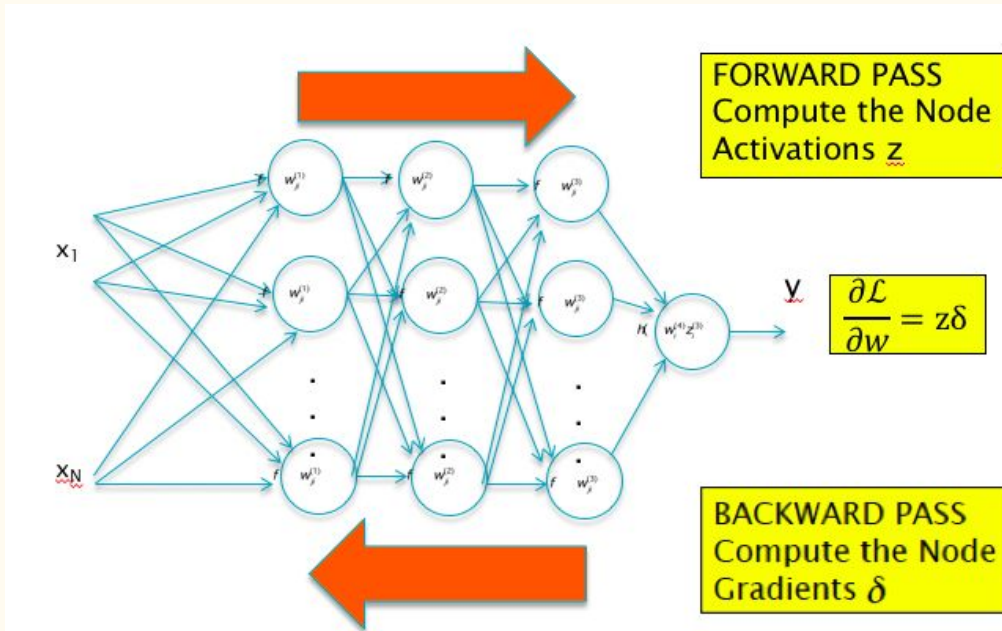
$$NLL(\hat{y}, y) = -y \log P(\hat{y} = 1) - (1 - y) \log(1 - P(\hat{y} = 1))$$

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**.

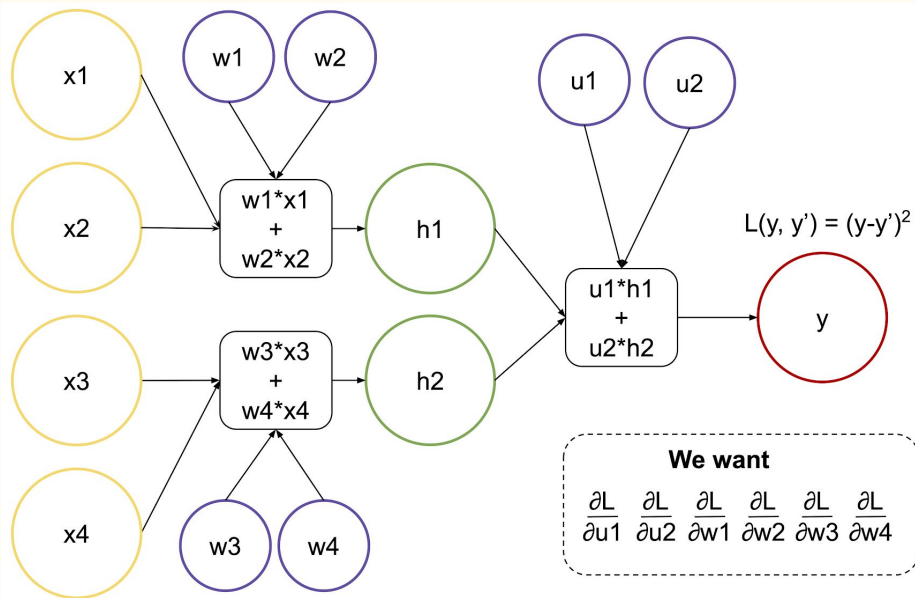
Backpropagation

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

$$\frac{\partial L}{\partial u_1} \quad \frac{\partial L}{\partial u_2} \quad \frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2} \quad \frac{\partial L}{\partial w_3} \quad \frac{\partial L}{\partial w_4}$$

All base derivatives

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

$$\frac{\partial y}{\partial h_1} = u_1 \quad \frac{\partial y}{\partial h_2} = u_2$$

$$\frac{\partial y}{\partial u_1} = h_1 \quad \frac{\partial y}{\partial u_2} = h_2$$

$$\frac{\partial h_1}{\partial w_1} = x_1 \quad \frac{\partial h_1}{\partial w_2} = x_2$$

$$\frac{\partial h_1}{\partial x_1} = w_1 \quad \frac{\partial h_1}{\partial x_2} = w_2$$

$$\frac{\partial h_2}{\partial w_3} = x_3 \quad \frac{\partial h_2}{\partial w_4} = x_4$$

$$\frac{\partial h_2}{\partial x_3} = w_3 \quad \frac{\partial h_2}{\partial x_4} = w_4$$

Full derivation examples

$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial u_1} = 2(y - y') * h_1$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_1} \frac{\partial h_1}{\partial w_1} = 2(y - y') * u_1 * x_1$$

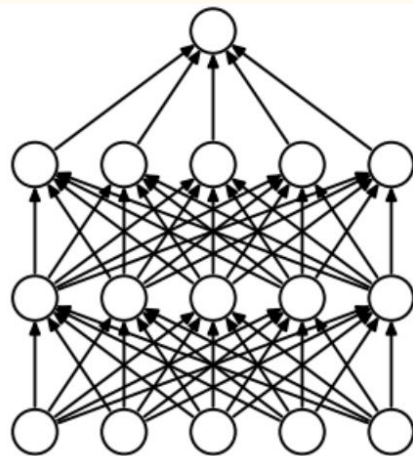
Regularization

L2 is standard

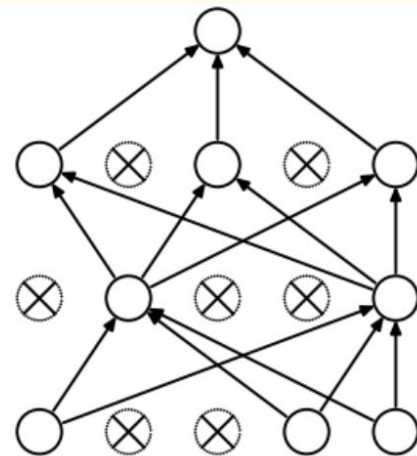
Early stopping based on validation error

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

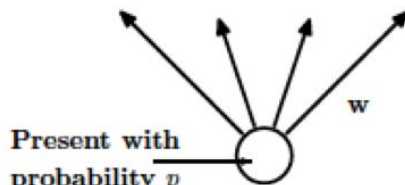
Same in testing: MC-dropout uncertainty estimates



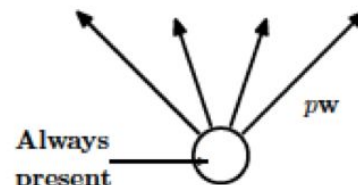
(a) Standard Neural Net



(b) After applying dropout.



(a) At training time



(b) At test time

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 the random seed ([Reimers and Gurevych, 2017](#)) or some other implementation detail ([Narang et al. 2021](#))
- Initialize with small non-zero values
- Greater learning capacity makes overfitting more likely: start making sure you can (over-)fit the data, then regularize

[Let's try some of this](#)

From discrete features to neural

Remember multiclass logistic regression:

$$P(y|x) = \textit{softmax}(\mathbf{W} \cdot \phi(\mathbf{x})), \mathbf{W} \in \mathfrak{R}^{|\mathbf{Y}| \times |\phi(\mathbf{x})|}$$

For large number of labels with many sparse features, difficult to learn. Factorize!

$$P(y|x) = \textit{softmax}((\mathbf{B} \cdot \mathbf{A}) \cdot \phi(\mathbf{x})), \mathbf{B} \in \mathfrak{R}^{|\mathbf{Y}| \times \mathbf{k}}, \mathbf{A} \in \mathfrak{R}^{\mathbf{k} \times |\phi(\mathbf{x})|}$$

\mathbf{A} contains the feature embeddings and \mathbf{B} maps them to labels

The feature embeddings can be initialized to word embeddings

[FastText \(Joulin et al., 2017\)](#) was popular baseline for classification (pre-BERT)

Sentence pair modelling

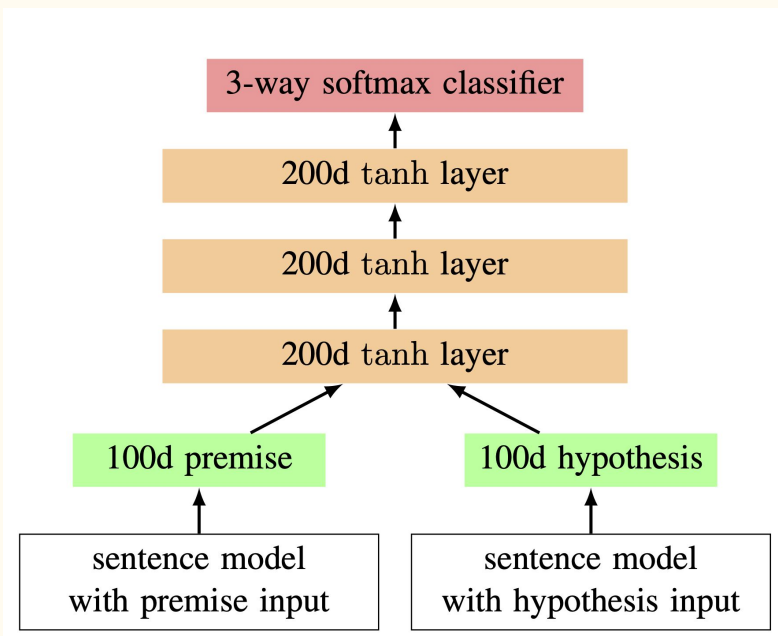
We can use FFNNs to perform tasks involving comparisons between two sentences, e.g. textual entailment: does the premise support the hypothesis?

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?
- Faithfulness, i.e. does the explanation contain the information used in the prediction, is important too (e.g. for debugging)

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 ([Turian et al., 2010](#))

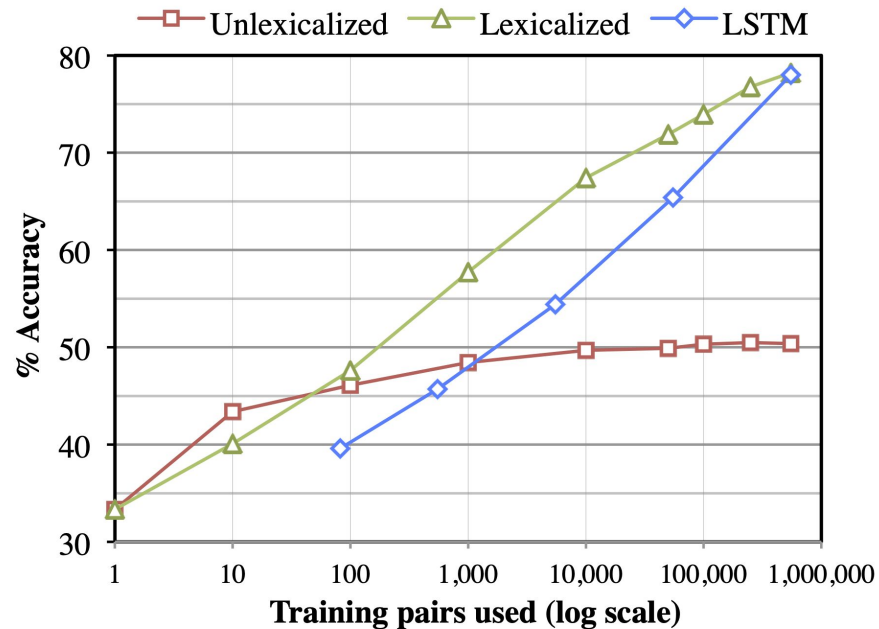
Why not be excited?

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)

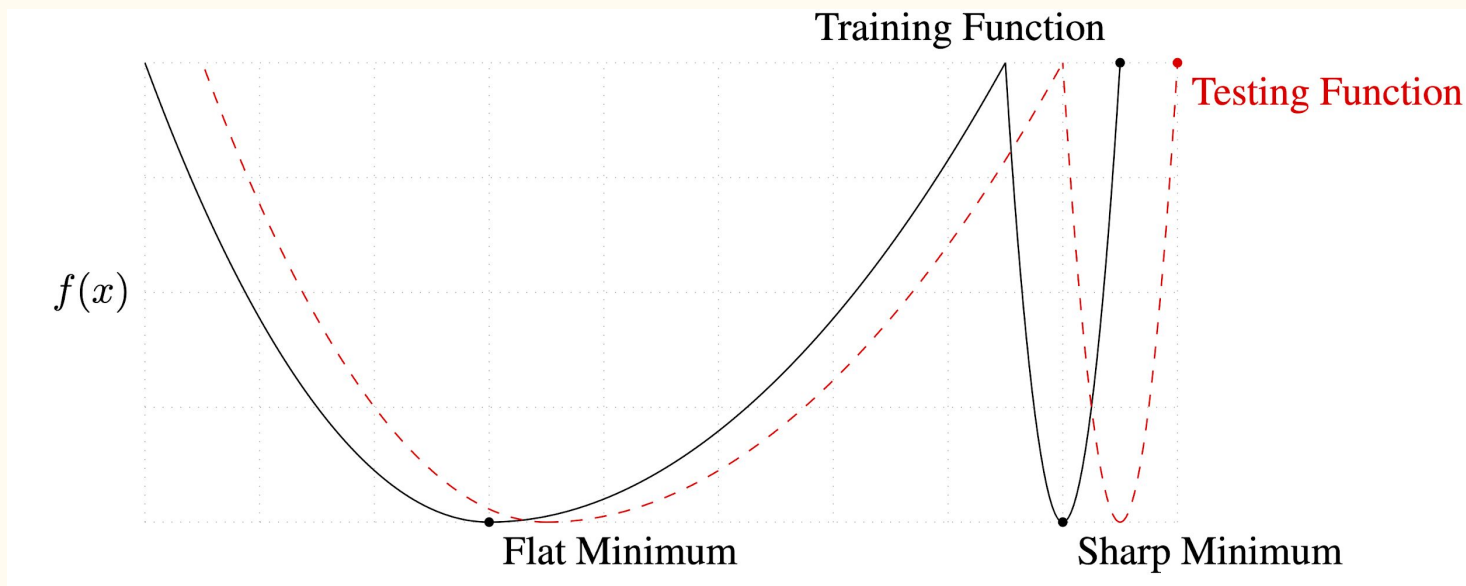
Need for (more) data

Feature engineering is replaced by architecture engineering



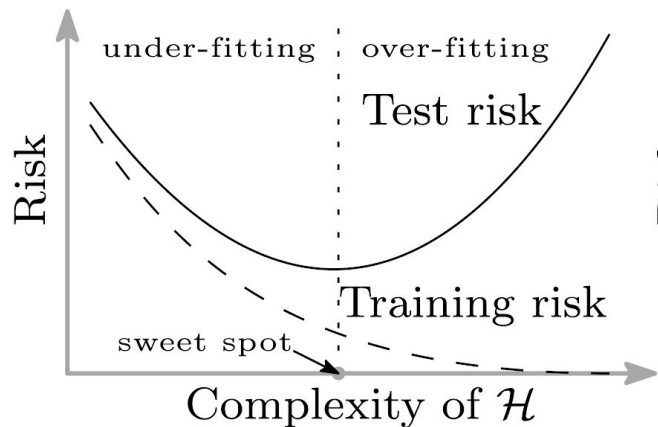
Bowman et al. (2015)

Optimization

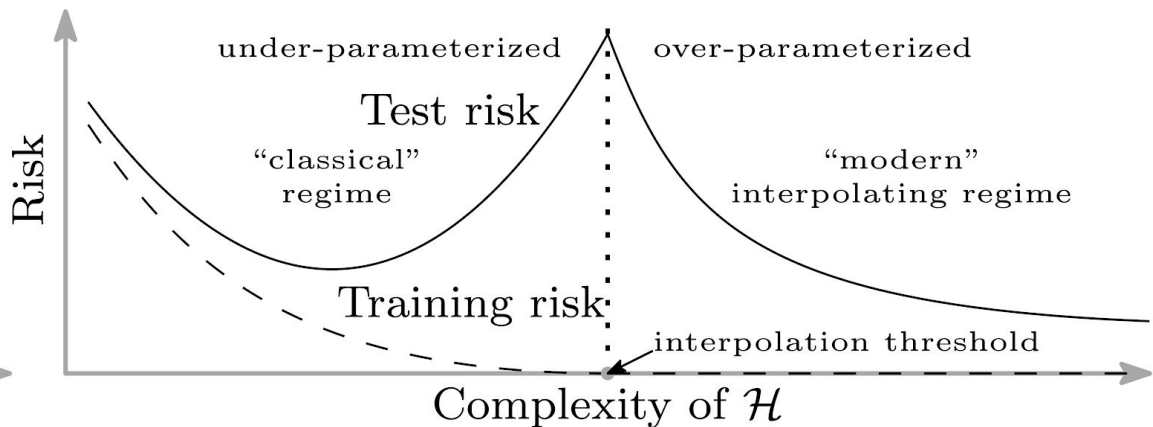


Noise from being stochastic in gradient descent can be beneficial as it avoid sharp local minima ([Keskar et al. 2017](#))

Double descent



(a) U-shaped "bias-variance" risk curve



(b) "double descent" risk curve

[Belkin et al. \(2018\)](#): Number of parameters is not the only factor in determining complexity; their size (*norm*) matters

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). Why new NNs then?

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

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

Larger networks with many network weights to tune have more chances to learn the few network weights needed for the task/dataset ([lottery ticket hypothesis](#))

Bibliography

A [simple implementation](#) in python of backpropagation (the nonlinear function derivative there is a bit of a misnomer, but the code works, why?)

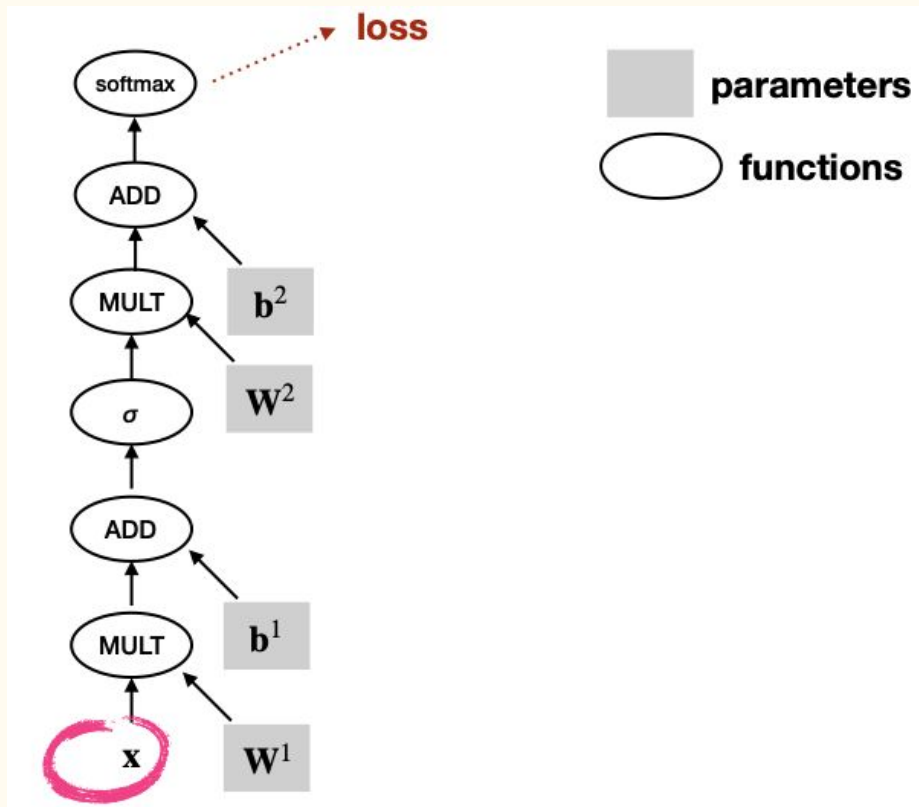
The [tutorial](#) of Quoc V. Le

[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](#)

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!