S&DS 365 / 665

Intermediate Machine Learning

Neural Networks

(continued)

September 21

Reminders

- Assignment 1 is out, due a week from today
- Quiz 2 available at 10:30 am today on Canvas; 48 hours/20 minutes
- Any material covered in class (up to and including Monday)
- OH schedule posted to Canvas/EdD
- Questions or concerns?

Last time: Basics of neural nets

1. Basic architecture of feedforward neural nets
2. Backpropagation
3. Examples: np-complete and TensorFlow

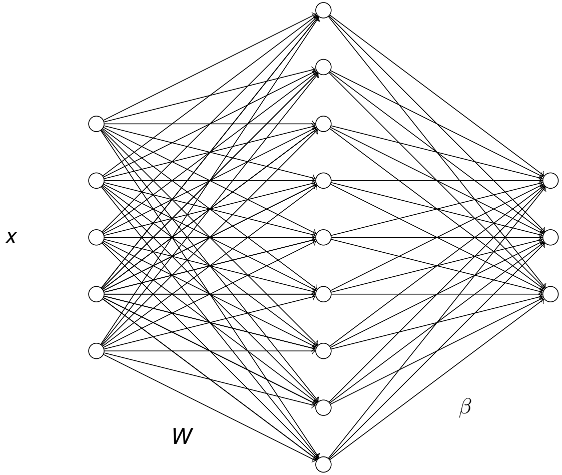
Today:

- Continue with NTK and double descent

Next up: Convolutional neural nets

- Mechanics of convolutional networks
- Filters and pooling and flattening (oh my!)
- Example: Classifying Ca2+ brain scans
- Other examples

Two-layer dense network (multi-layer perceptron)



Equivalent to linear model

With just the weights, this is just another linear model

$$f(x) = \widetilde{\beta}^T x + \widetilde{\beta}_0$$

We get a reparameterization of a linear model; nothing new.

Need to add *nonlinearities*

Nonlinearities

Add nonlinearity

$$h(x) = \varphi(Wx + b)$$

applied component-wise.

For regression, the last layer is just linear:

$$f(x) = \beta^T h(x) + \beta_0$$

Nonlinearities

Commonly used nonlinearities:

$$\varphi(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

$$\varphi(u) = \mathrm{sigmoid}(u) = \frac{e^u}{1 + e^u}$$

$$\varphi(u) = \mathrm{relu}(u) = \max(u, 0)$$

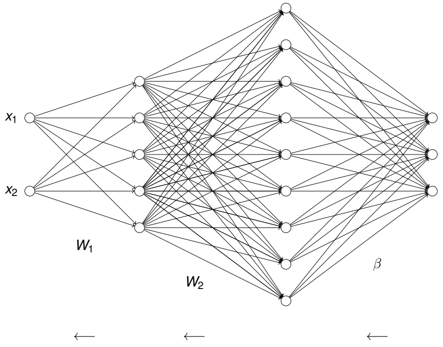
Nonlinearities

So, a neural network is nothing more than a parametric regression model with a restricted type of nonlinearity

Training

- The parameters are trained by stochastic gradient descent.
- To calculate derivatives we just use the chain rule, working our way backwards from the last layer to the first.

High level idea



Start at last layer, send error information back to previous layers

Classification

For classification we use softmax to compute probabilities

$$(p_1, p_2, p_3) = \frac{1}{e^{f_1} + e^{f_2} + e^{f_3}} \left(e^{f_1}, e^{f_2}, e^{f_3} \right)$$

The loss function is

$$\mathcal{L} = -\log P(y \mid x) = \log \left(e^{f_1} + e^{f_2} + e^{f_3} \right) - f_y$$

So, we have

$$\frac{\partial \mathcal{L}}{\partial f_k} = p_k - \mathbb{1}(y = k)$$

4: Demos

Interactive examples

https://playground.tensorflow.org/

What’s going on?

- These models are curiously robust to overfitting
- Why is this?
- Some insight: Kernels and double descent

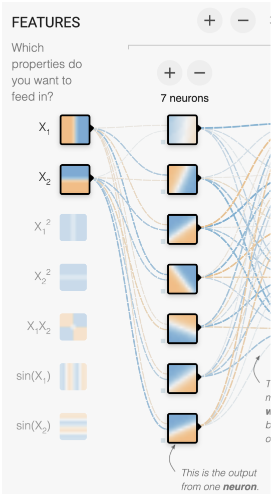
Fruit flies

Drosophila melanogaster

- Model scientific organism
- Eight Nobel prizes for research using *Drosophila*

The statistical fruit fly

A fruit fly for deep learning: Random features


FEATURES
+
−
Which properties do you want to feed in?
+
−
7 neurons
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
This is the output from one **neuron**.

http://www.argmin.net/2017/12/05/kitchen-sinks/

Double descent

We’ll go over notes on the double descent phenomenon on the board, which will allow you to complete Problem 4 on the first assignment.

`https://github.com/YData123/sds365-fa22/raw/main/notes/double-descent.pdf`

OLS and minimal norm solution

OLS: $p < n$

$$\widehat{\beta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T Y$$

Minimal norm solution: $p > n$:

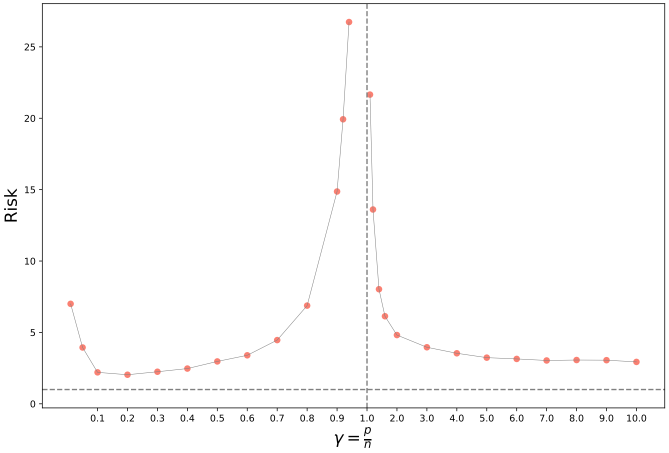
$$\widehat{\beta}_{\text{mn}} = \mathbb{X}^T(\mathbb{X}\mathbb{X}^T)^{-1} Y$$

“Ridgeless regression”

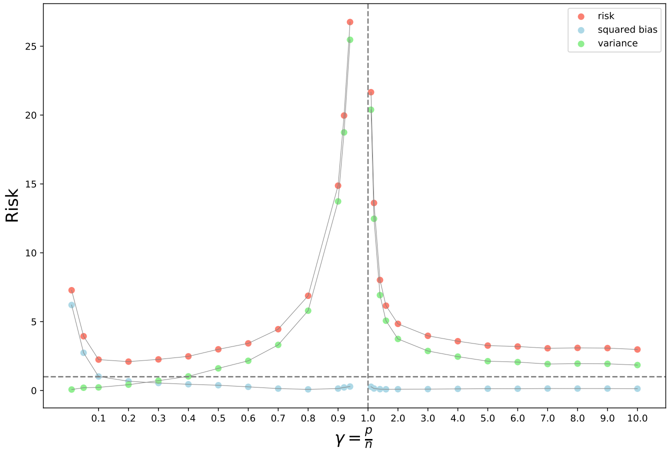
As λ decreases to zero, the ridge regression estimate:

- Converges to OLS in the “classical regime” $\gamma < 1$
- Converges to $\widehat{\beta}_{\text{mn}}$ in “overparameterized regime” $\gamma < 1$

Double descent



Double descent



Neural tangent kernel

There is a kernel view of neural networks that has been useful in understanding the dynamics of stochastic gradient descent for neural networks.

This is based on the *neural tangent kernel (NTK)*

Parameterized functions

Suppose we have a parameterized function $f_\theta(x) \equiv f(x; \theta)$

Almost all machine learning takes this form — for classification and regression, these give us estimates of the regression function

For neural nets, the parameters θ are all of the weight matrices and bias (intercept) vectors across the layers.

Feature maps

Suppose we have a parameterized function $f_\theta(x) \equiv f(x;\theta)$

We then define a *feature map*

$$x \mapsto \varphi(x) = \nabla_\theta f(x;\theta) = \begin{pmatrix} \frac{\partial f(x;\theta)}{\partial \theta_1} \\ \frac{\partial f(x;\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(x;\theta)}{\partial \theta_p} \end{pmatrix}$$

This defines a Mercer kernel

$$K(x, x') = \varphi(x)^T \varphi(x') = \nabla_\theta f(x;\theta)^T \nabla_\theta f(x';\theta)$$

Feature maps

This defines a Mercer kernel

$$K(x, x') = \varphi(x)^T \varphi(x') = \nabla_\theta f(x; \theta)^T \nabla_\theta f(x'; \theta)$$

What is the NTK for the random features model?

NTK and SGD

- The NTK has been used to study the dynamics of stochastic gradient descent
- Upshot: As the number of neurons in the layers grows, the parameters in the network barely change during training, even though the training error quickly decreases to zero

Summary

- Neural nets are layered linear models with nonlinearities added
- Trained using stochastic gradient descent with backprop
- Can be automated to train complex networks (with no math!)
- Key to understanding risk properties: Double descent
- Kernel connection: NTK