

Day 6: Neural Networks

Summer STEM: Machine Learning

Department of Electrical and Computer Engineering
NYU Tandon School of Engineering
Brooklyn, New York

Outline

1 Review

2 Non-linear Optimization

3 Neural Networks

4 Stochastic Gradient Descent

5 Overparameterization

6 Lab

Machine Learning Problem Pipeline

- 1** Formulate the problem: regression, classification, or others?
- 2** Gather and visualize the data
- 3** Design the model and the loss function
- 4** Train your model
 - (a) Perform feature engineering
 - (b) Construct the design matrix
 - (c) Choose regularization techniques
 - (d) Tune hyper-parameters using a validation set
 - (e) If the performance is not satisfactory, go back to step (a).
- 5** Evaluate the model on a test set

Outline

1 Review

2 Non-linear Optimization

3 Neural Networks

4 Stochastic Gradient Descent

5 Overparameterization

6 Lab

Motivation

- Cannot rely on closed form solutions
 - Computation efficiency: operations like inverting a matrix is not efficient
 - For more complex problems such as neural networks, a closed-form solution is not always available
- Need an optimization technique to find an optimal solution
 - Machine learning practitioners use **gradient**-based methods

Gradient Descent Algorithm

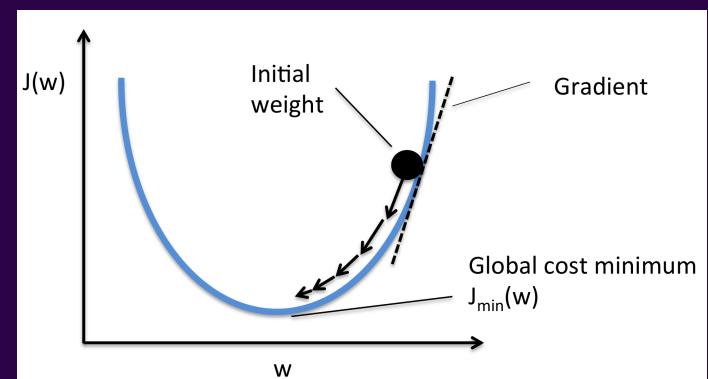
■ Update Rule

Repeat{

$$\mathbf{w}_{new} = \mathbf{w} - \alpha \nabla J(\mathbf{w})$$

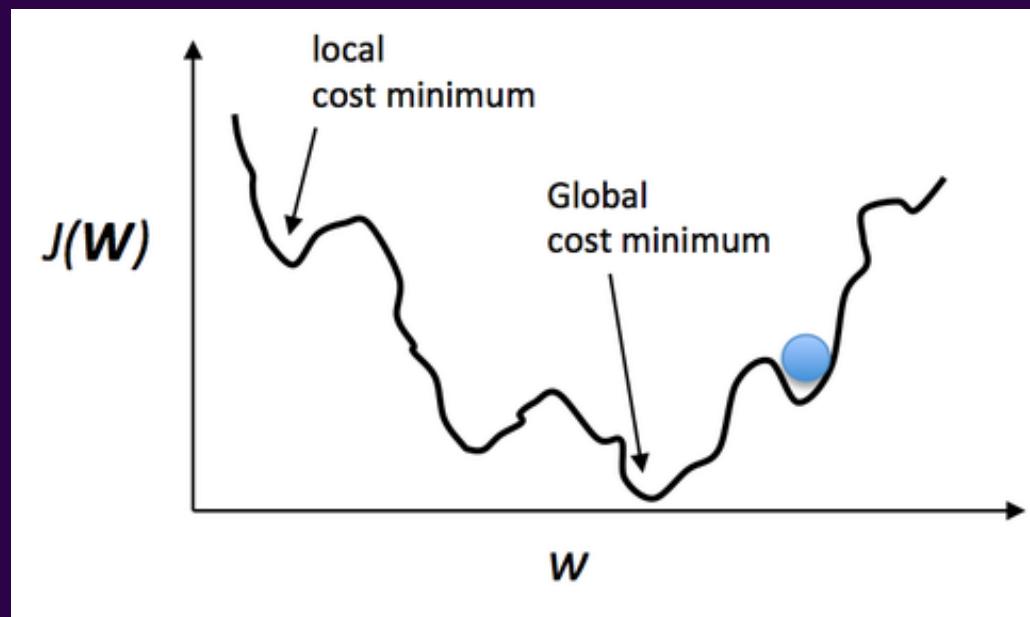
}

α is the learning rate

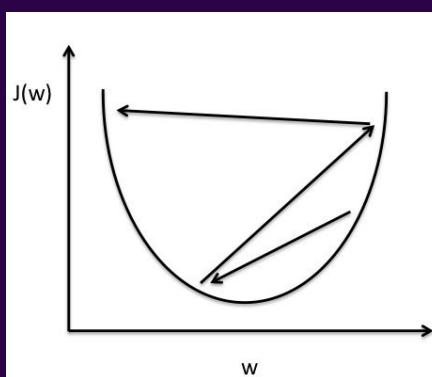


General Loss Function Contours

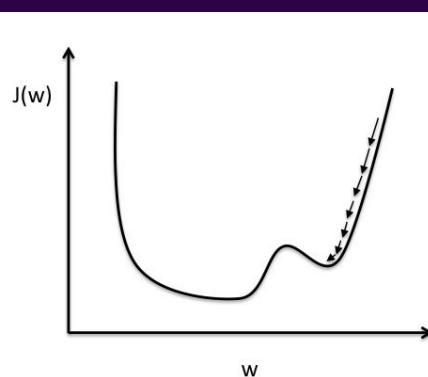
- Most loss function contours are not perfectly parabolic
- Our goal is to find a solution that is very close to global minimum by the right choice of hyper-parameters



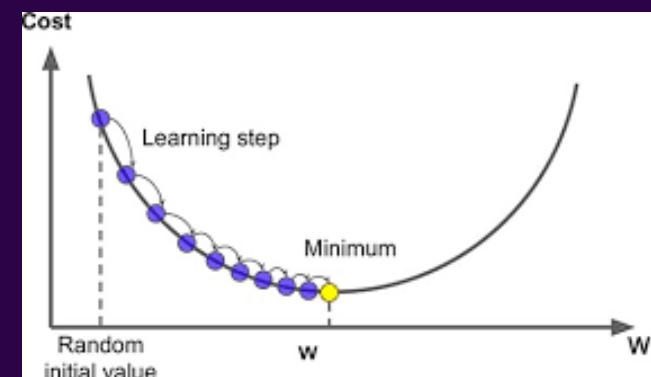
Understanding Learning Rate



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.



Correct learning rate

Some Animations

- Demonstrate gradient descent animation

Outline

1 Review

2 Non-linear Optimization

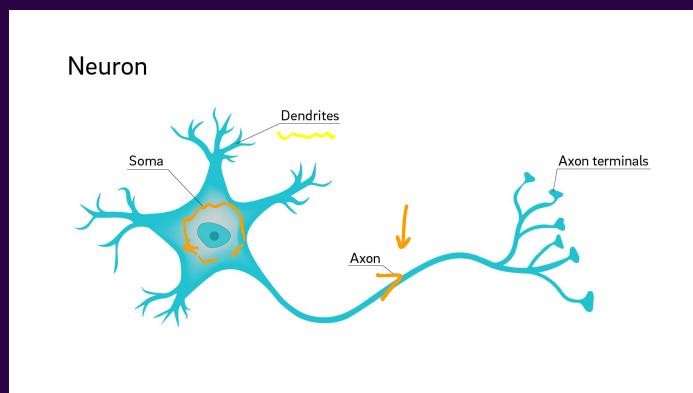
3 Neural Networks

4 Stochastic Gradient Descent

5 Overparameterization

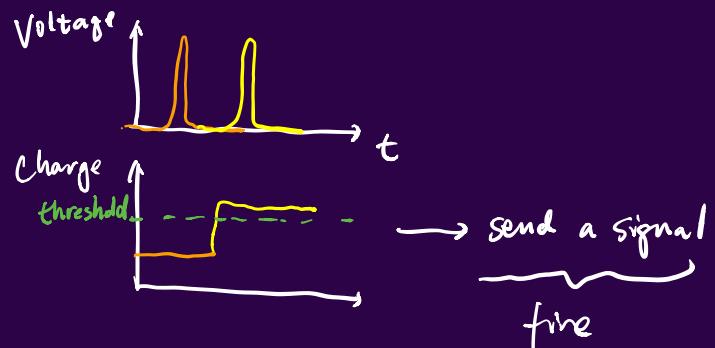
6 Lab

Biological Neuron



Source: David Baillot/UC San Diego

- Communicate with other cells
- A neuron can send electrochemical signals to other neurons

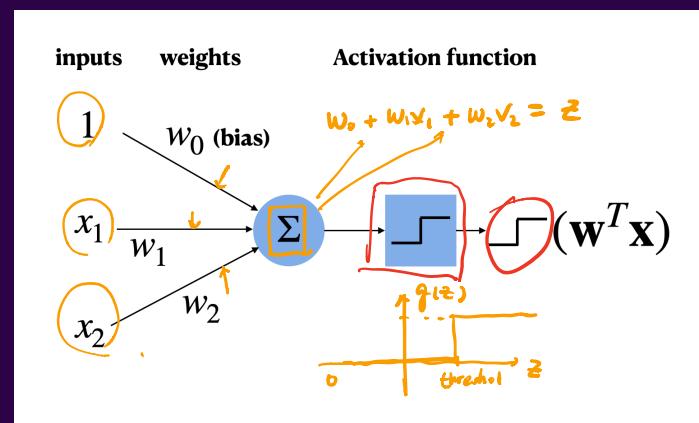


Mathematical Neuron: Perceptron

Biological Neuron:

- A neuron can receive electrochemical signals from other neurons;
- A neuron fires once its accumulated electric charge passes a certain threshold.
- Neurons that fire together wire together.

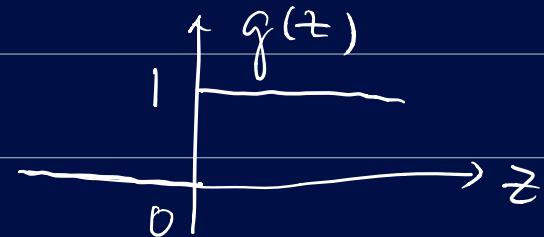
Mathematical Neuron
(Perceptron)



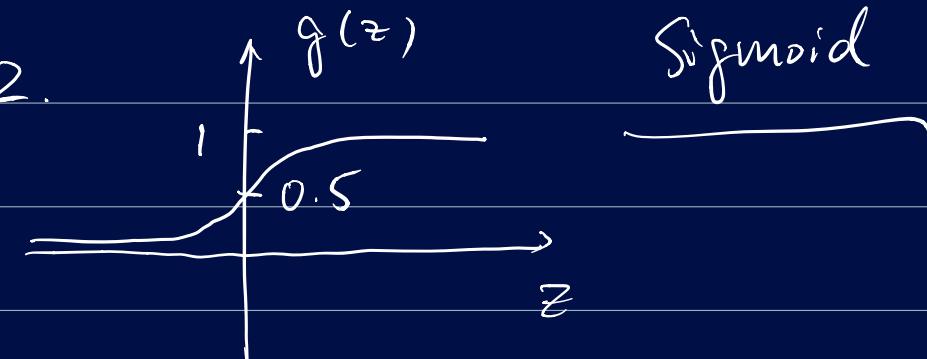
$$g(\mathbf{w}^T \mathbf{x}) \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Activation function

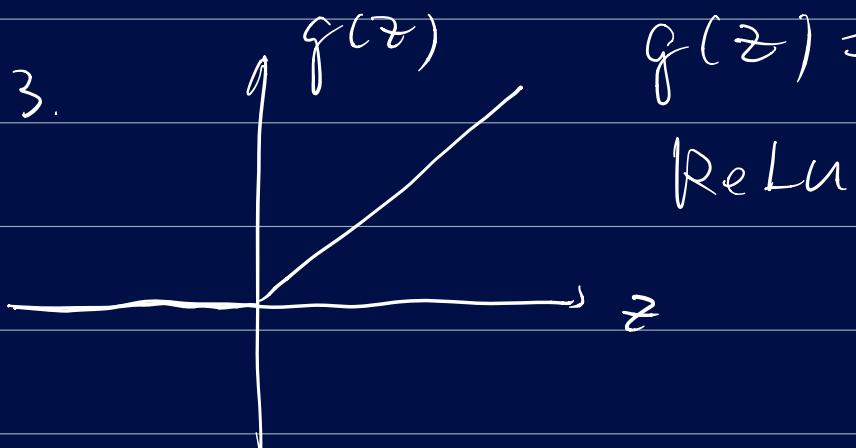
1. Step function



2.



3.



$$g(z) = \max(z, 0)$$

ReLU

If we use sigmoid activation
our model is simply
 $f(x) = \sigma(w^T x)$

Relation to Logistic Regression

What if we use the sigmoid function as the activation?

$$f(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

Decision boundary is a line: how is this supposed to revolutionize machine learning?

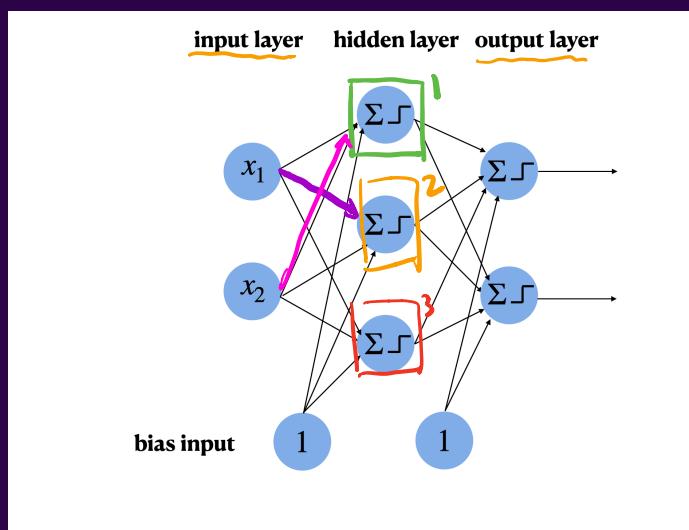
Multi-layer Perceptron (MLP)

We need more neurons and we need to connect them together!

- Many ways to do that...
- Today: multi-layer perceptron/fully connected feed-forward network.

$$\sigma(\omega^T x)$$

MLP Example



- What is the shape of the input and output?
- How many parameters does this model have?
- What activation function would you use for the output layer? Why?

w_{ij} : the weight associated with the arrow that points from x_j to the neuron i .

$$z_1 = w_{10}x_0 + w_{11}x_1 + w_{12}x_2$$

$$x_0 = 1 \quad z_2 = w_{20}x_0 + w_{21}x_1 + w_{22}x_2$$

$$z_3 = w_{30}x_0 + w_{31}x_1 + w_{32}x_2$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \underbrace{\begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \\ w_{30} & w_{31} & w_{32} \end{bmatrix}}_W \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Let's say we don't use any activation function,

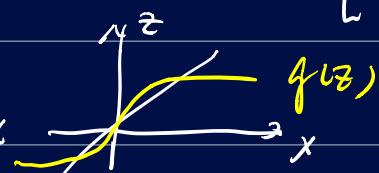
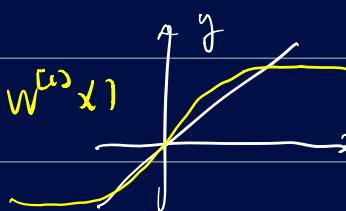
$$z = \underline{w^T x}$$

$$y = \underline{w^{[2]} g(z)} = w^{[2]} \underbrace{g(w^{[1]} x)}$$

$$z = w^{[1]} x$$

$$y = w^{[2]} g(w^{[1]} x)$$

$$g(z) = \begin{bmatrix} f(z_1) \\ f(z_2) \\ \vdots \end{bmatrix}$$



More about MLPs

- Many choices for the activation function: Sigmoid, Tanh, ReLU, Swish, etc.
- Many choices for the number of hidden layers and the number of neurons per layer.
- MLPs can approximate any continuous function given enough data.
- MLPs can overfit, but we know many effective ways of regularization.

Exercise: TensorFlow Keras Basics

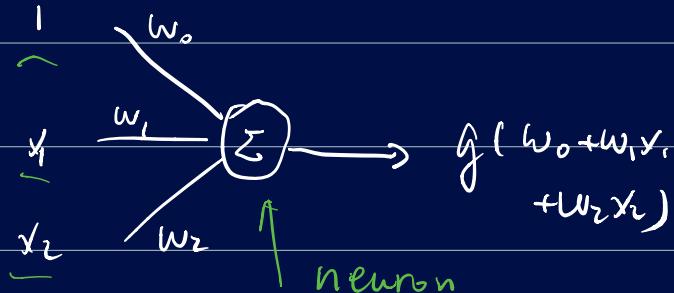
Open `demo_tf_keras_basics.ipynb`

Review

$$x = [1 \ x_1 \ x_2 \ \dots]^T$$

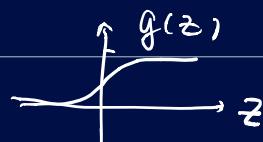
• Perceptron

$$\sigma(w^T x) \quad \sigma: \text{Sigmoid}$$

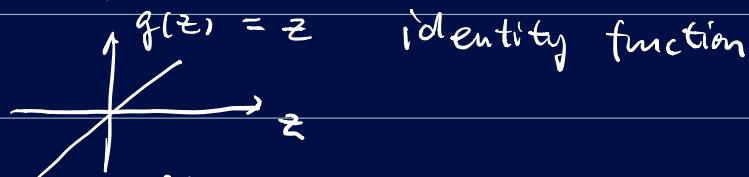


Activation functions.

Sigmoid

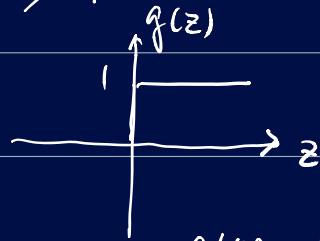


linear

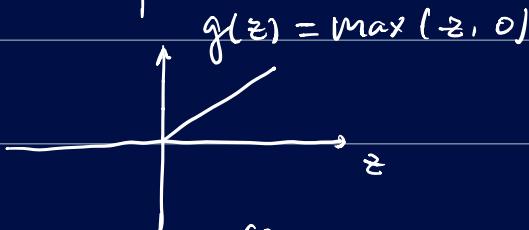


identity function

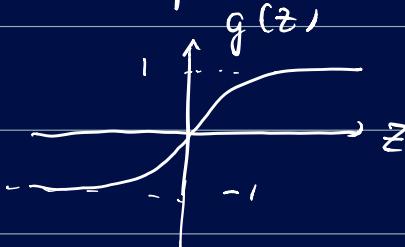
Step function



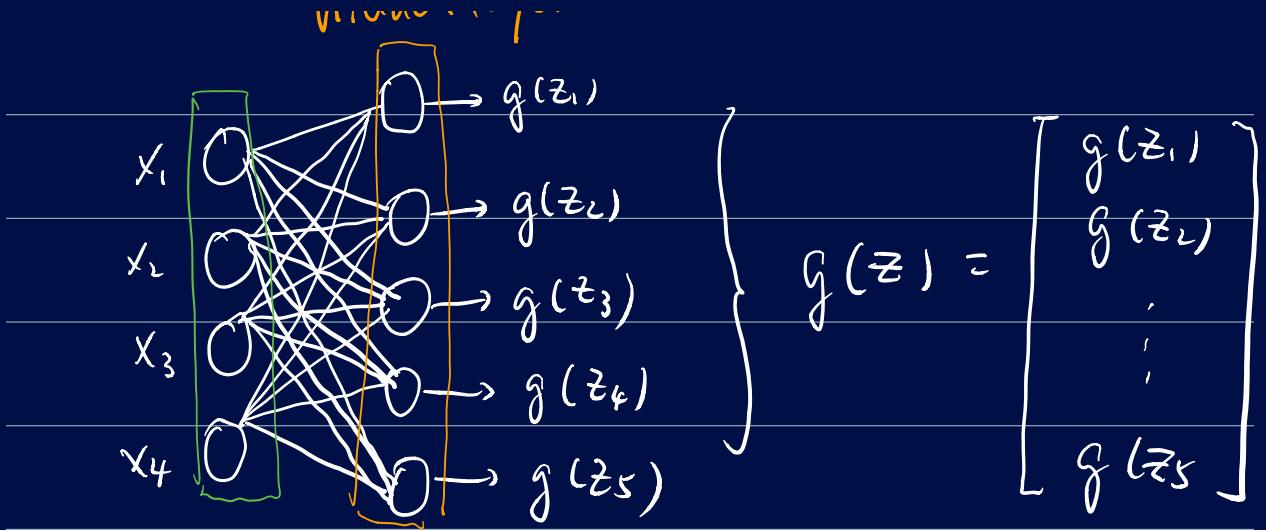
ReLU



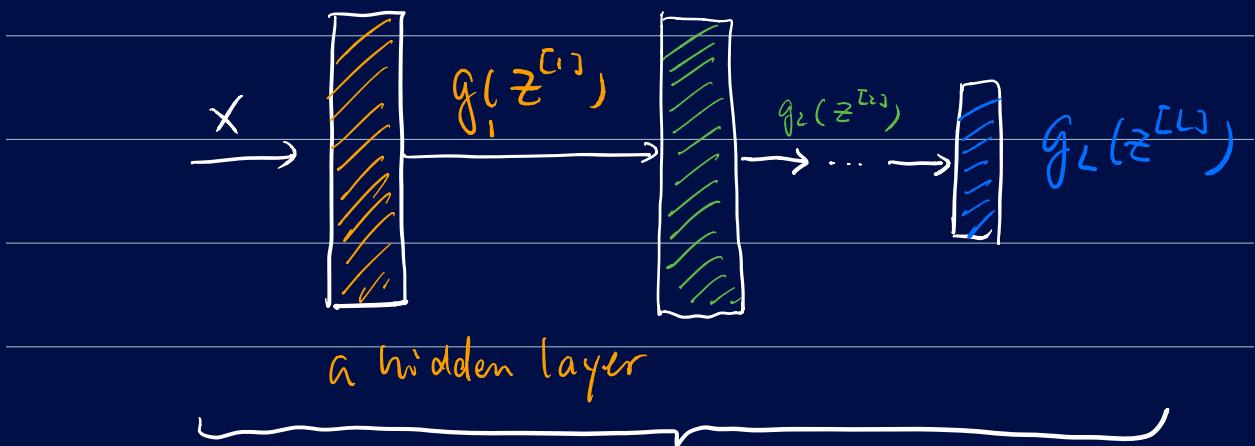
Tanh



In hidden layer



$$z_i = \sum w_{ij} x_j \quad w_{ij} \text{ points from } x_j \text{ to } z_i$$



$$\hat{y} = f(x) = \underbrace{g_L(z^{[L]})}_{\sim}$$

the choice of g_L depends on the task:

- regression / classification
- our knowledge about the output

Ex. • binary classification ($y \in \{0, 1\}$)

g_L : Sigmoid

• multi-class classification

g_L : Softmax

• regression ($y \in \mathbb{R} (-\infty, +\infty)$)

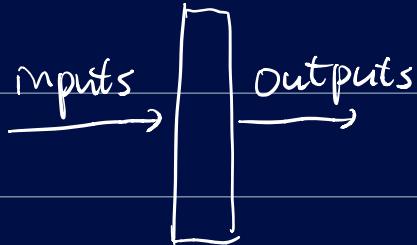
g_L : Linear

from tensorflow.keras.layers import Dense
from tensorflow.keras import layers

• fully-connected layer : `layers.Dense()`

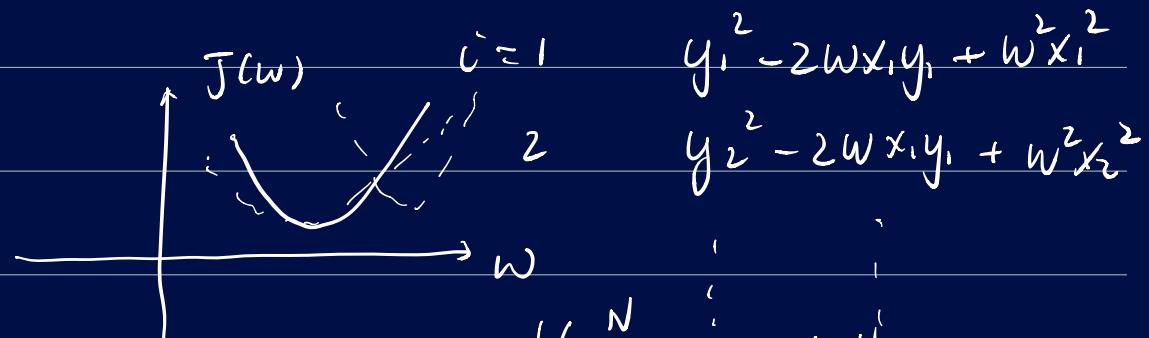
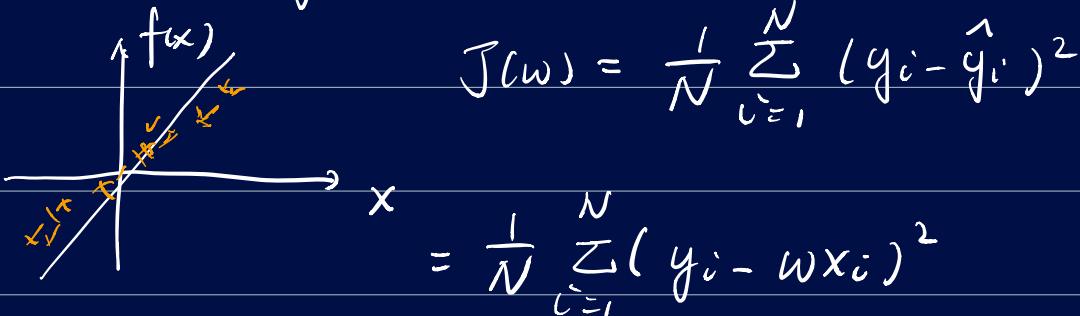
$l_1 = \text{layers.Dense(arg1, activation="relu", input_shape=(4,))$

arg1: size of outputs
of neurons



Review : Gradient Descent

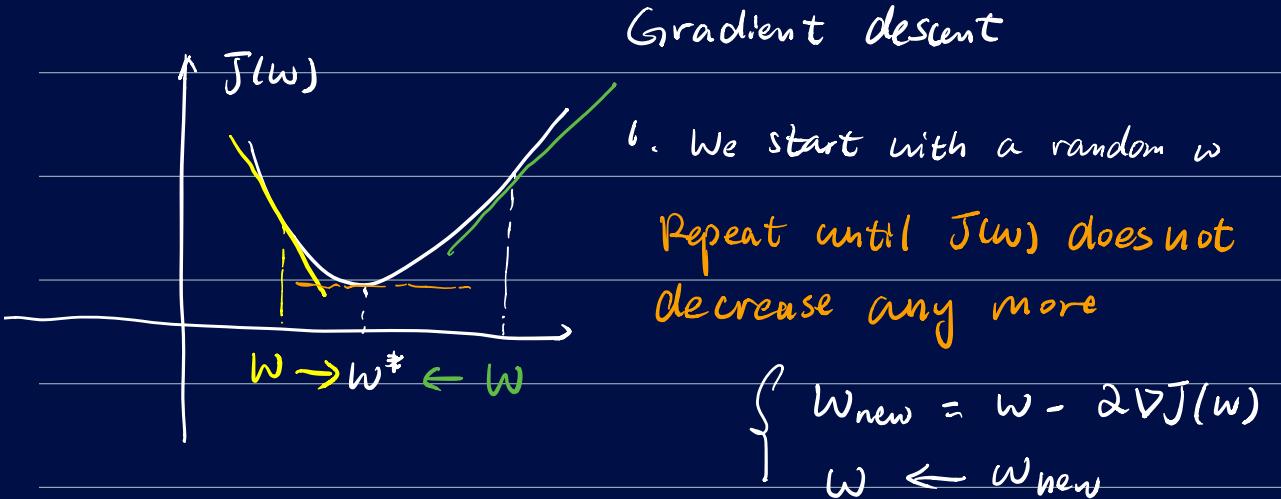
Toy example : Consider scalar-valued features x , labels y , and suppose our model is $\hat{y} = f(x) = w \cdot x$ w : a scalar



\checkmark convex

\wedge concave

$$+ \underbrace{\left(\sum_{i=1}^N x_i^2 \right) w^2}_{>0}$$



α : learning rate $\alpha > 0$

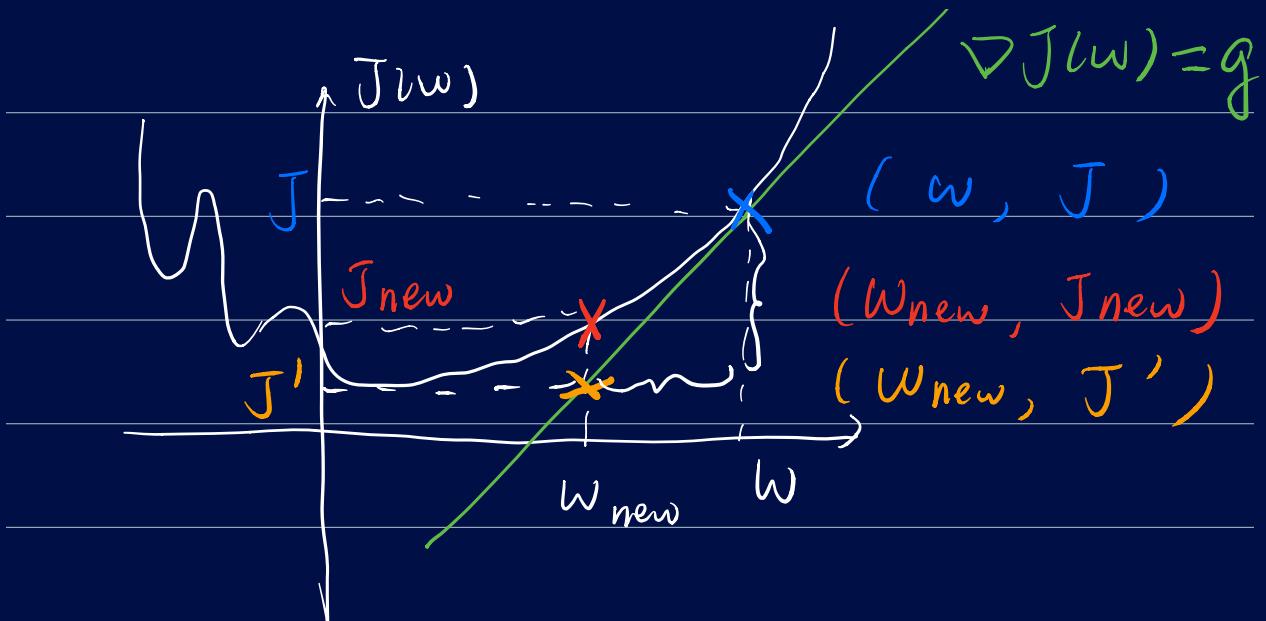
$\nabla J(w)$: the gradient for $J(w)$ at w

Training a model: Finding the optimal weights (model parameters) w^* such that $J(w^*)$ is the lowest possible value.

$$\nabla J(w) > 0 \quad w_{\text{new}} = w - \underbrace{\alpha \nabla J(w)}_{< 0}$$

$$\nabla J(w) < 0 \quad w_{\text{new}} = w - \underbrace{\alpha \nabla J(w)}_{> 0}$$

$$\nabla J(w) = 0 \quad w_{\text{new}} = w$$



$$g = \frac{J - J'}{\omega - \omega_{\text{new}}}$$

$$g(\omega - \omega_{\text{new}}) = J - J'$$

$$J' = J - g(\omega - \omega_{\text{new}})$$

$$J_{\text{new}} \approx J' = J + g(\omega_{\text{new}} - \omega)$$

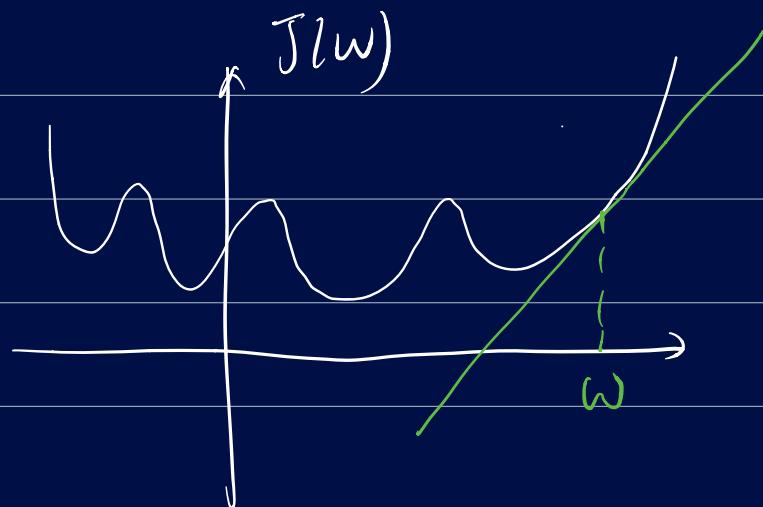
We want $J_{\text{new}} < J$ ($J_{\text{new}} - J < 0$)

$$J_{\text{new}} - J \approx g(\omega_{\text{new}} - \omega) < 0$$

$$g(\omega - \alpha \cdot g - \omega) = -\alpha g^2 < 0$$

$\alpha > 0$

Derivatives and Gradients

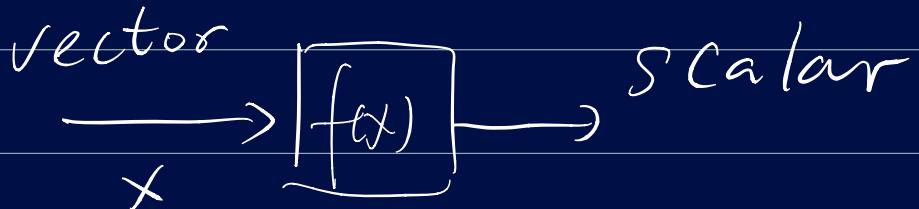


The green line : tangent line
to the curve $J(w)$

$\nabla J(w)$: the slope of the
 $(\nabla \text{ nabla})$ tangent line
 $(\Delta \text{ delta})$

Multivariable Case

$$f(x_1, x_2, x_3, \dots) = f(\mathbf{x})$$



Suppose we have \mathbf{x} and \mathbf{x}'

very close to each other

$$\|\mathbf{x} - \mathbf{x}'\| \approx 0, \text{ then we}$$

can approximate

$$\underbrace{f(\mathbf{x}) - f(\mathbf{x}')}_{\text{Scalar}} \approx \underbrace{(\nabla f(\mathbf{x}))^\top}_{\text{now vector col.}} \underbrace{(\mathbf{x} - \mathbf{x}')}_{\text{vector}}$$

$$\left. \frac{\partial f}{\partial x_i} \right|_{\mathbf{x}} = (\nabla f(\mathbf{x}))^\top \text{ derivatives}$$

Outline

1 Review

2 Non-linear Optimization

3 Neural Networks

4 Stochastic Gradient Descent

5 Overparameterization

6 Lab

Deep Learning

What does deep learning stand for?

- Deep: Neural network architectures with many hidden layers.
- Learning: Optimizing model parameters given a dataset.

In general, the deeper the model is, the more parameters we need to learn and the more data is needed.

Large-Scale Machine Learning

For deep learning systems to perform well, large datasets are required

- COCO 330K images
- ImageNet 14 million images

Challenges:

- Memory limitation: GeForce RTX 2080 Ti has 11 GB memory, while ImageNet is about 300 GB.
- Computation: Calculating gradients for the whole dataset is computationally expensive (slow), and we need to do this many times.

Stochastic Gradient Descent

Idea: Instead of calculating the gradients from the whole dataset, do it only on a subset.

- Randomly select B samples from the dataset
- The loss for this subset

$$\tilde{J}(\omega) = \frac{1}{N} \sum_{i=1}^N \|y - \hat{y}_i\|^2$$

$$\tilde{J}(\omega) = \frac{1}{B} \sum_{i=1}^B \|y - \hat{y}_i\|^2$$

$$B \ll N$$

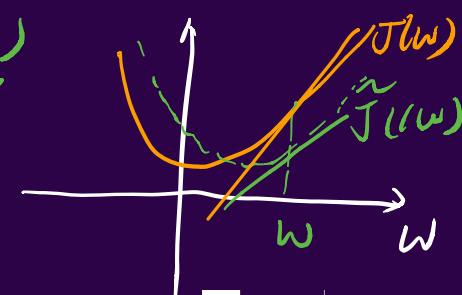
$$\tilde{J}'(\omega) = \frac{1}{B} \sum_{i=1}^B \|y - \hat{y}_i\|^2$$

- Update Rule
Repeat{

$$\mathbf{w}_{new} = \mathbf{w} - \alpha \nabla \tilde{J}(\mathbf{w})$$

}

$$\nabla \tilde{J}(\omega)$$



Yet Another Hyper-Parameter

This gives a noisy gradient

$$\nabla \tilde{J}(\mathbf{w}) = \nabla J(\mathbf{w}) + \epsilon$$

$\nabla J(\mathbf{w})$ ϵ *epsilon*

- **SGD:** $B = 1$, gives very noisy gradients
- **(batch) GD:** $B = N$, $\epsilon = 0$, expensive to compute
- **Mini-batch GD:** Pick a small B , typical values are 32, 64, rarely more than 128 for image inputs

i	x
0	
1	
2	
3	
4	
5	

$$N = 6$$

B would be 16, 32,

$$B = 2$$

64 ... 128

It's also a hyper-parameter

Randomly pick $B=2$ samples from
the dataset, for example

Epoch 1

$$\text{iteration 1 } \tilde{J}(\omega) = \|y_1 - \hat{y}_1\|^2 + \|y_3 - \hat{y}_3\|^2$$

$$\text{iteration 2 } \tilde{J}(\omega) = \|y_2 - \hat{y}_2\|^2 + \|y_0 - \hat{y}_0\|^2$$

$$\text{iteration 3 } \tilde{J}(\omega) = \|y_4 - \hat{y}_4\|^2 + \|y_5 - \hat{y}_5\|^2$$

Epoch 2

$$\text{iteration 1 } \tilde{J}(\omega) = \|y_4 - \hat{y}_4\|^2 + \|y_1 - \hat{y}_1\|^2$$

$$\text{iteration 2 } \tilde{J}(\omega) = \|y_2 - \hat{y}_2\|^2 + \|y_3 - \hat{y}_3\|^2$$

$$\text{iteration 3 } \tilde{J}(\omega) = \|y_0 - \hat{y}_0\|^2 + \|y_5 - \hat{y}_5\|^2$$

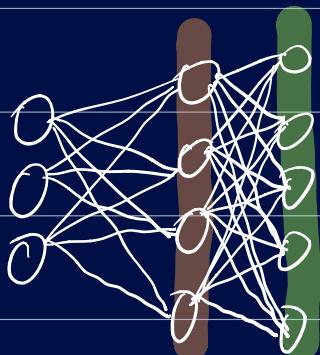
each iteration pick B samples

each epoch iterate through all
N samples.

Hyper-parameters in Deep Learning

- B : batch size, # of epochs
- α : learning rate
- # of hidden layers / # neurons per layer
- Type of the activation functions for each layer

Model parameters



each edge / arrow represents
a weight

all the weights / bias

$$\begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

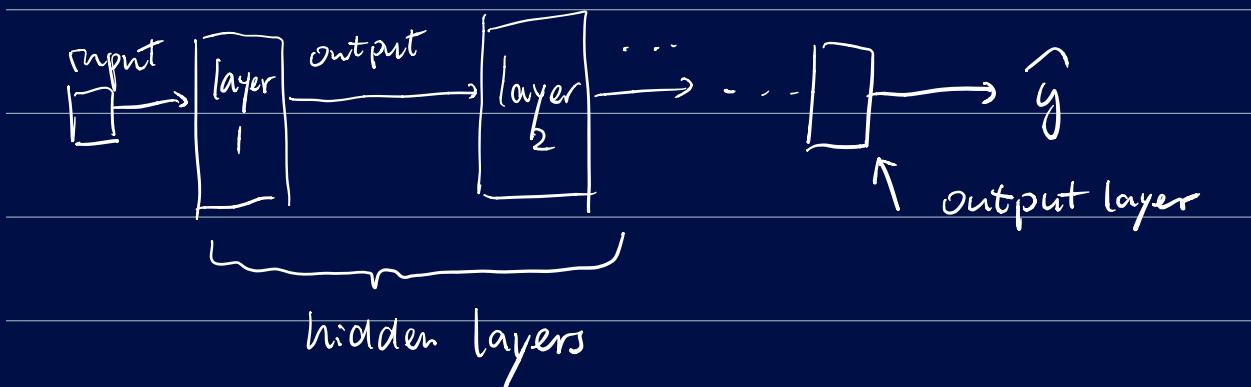
Training: to find w^*

such that $J(w^*)$ is = $\underbrace{\begin{bmatrix} w_{01} & w_{02} \\ w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\text{weights}} + \underbrace{\begin{bmatrix} w_{00} \\ w_{10} \\ w_{20} \end{bmatrix}}_{\text{bias}}$

the least possible value
(e.g. via SGD)

if your initial w is very close to w^* , then you will converge faster

Sometimes, it might be helpful to simply restart training with a different initialization of w (model.compile())



When you define a layer in tensorflow

Dense (10, activation = "relu", input_shape=(2,))

→ creating all the weights and bias

Input ($B, \underline{2}$) Output = Weights · input

Output ($B, \underline{10}$) + bias

Some Noise Helps

If $\nabla \tilde{J}(w) = 0$ accidentally, at the next iteration $\nabla \tilde{J}(w)$ will

What would w_{new} be if $\nabla J(w) = 0$ be different

$$\begin{aligned} \text{GD } w_{\text{new}} &= w - \alpha \nabla J(w) \\ &= w \end{aligned}$$

$$|\epsilon| > 0$$

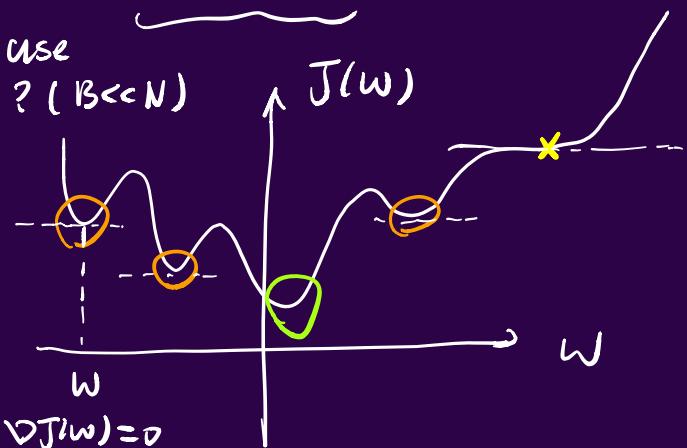
Even if we can, we rarely set $B = N$. In fact, some noise in the gradients might help to

what if we use SGD? ($B < N$)

- escape from local minima,
- escape from saddle points, and
- improve generalization.

global minimum

(All global minima are also local minima)



Adam

noise

$$\nabla \tilde{J}(w) \neq \nabla J(w)$$

$$w_{\text{new}} = w - \alpha \nabla \tilde{J}(w) \neq w$$

escape

Outline

1 Review

2 Non-linear Optimization

3 Neural Networks

4 Stochastic Gradient Descent

5 Overparameterization

6 Lab

Overparameterized Models

- Modern deep learning models are heavily overparameterized, i.e. the number of learnable parameters is much larger than the number of the training samples.

$$\# \text{ parameters per layer} = \# \text{ weights} + \# \text{ bias}$$

Ex. input size 300 $\# \text{ weights}$ 300×300
 output size 300 $\# \text{ bias}$ 300

$$\underbrace{\text{Output}}_{\substack{\text{vector of} \\ \text{size 300}}} = \underbrace{\text{Weights} \cdot \text{input}}_{\substack{\downarrow \\ \text{matrix}}} + \text{bias} \quad \left(\begin{array}{l} \text{it does not depend} \\ \text{on } B \end{array} \right)$$

vector of size 300

Overparameterized Models

- Modern deep learning models are heavily overparameterized, i.e. the number of learnable parameters is much larger than the number of the training samples.
 - ResNet: State-of-the-art vision model, 10-60 million parameters

Overparameterized Models

- Modern deep learning models are heavily overparameterized, i.e. the number of learnable parameters is much larger than the number of the training samples.
 - ResNet: State-of-the-art vision model, 10-60 million parameters
 - GPT-3: State-of-the-art language model, 175 billion parameters

Overparameterized Models

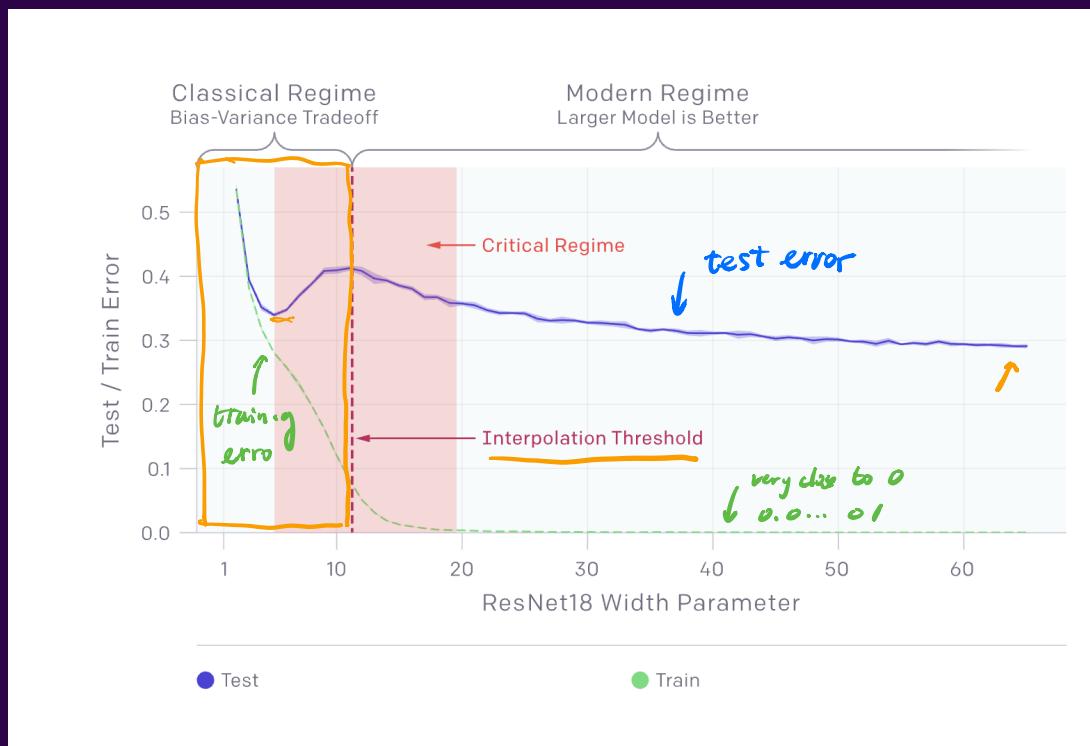
- Modern deep learning models are heavily overparameterized, i.e. the number of learnable parameters is much larger than the number of the training samples.
 - ResNet: State-of-the-art vision model, 10-60 million parameters
 - GPT-3: State-of-the-art language model, 175 billion parameters
- Conventional wisdom: Such models overfit.

Overparameterized Models

- Modern deep learning models are heavily overparameterized, i.e. the number of learnable parameters is much larger than the number of the training samples.
 - ResNet: State-of-the-art vision model, 10-60 million parameters
 - GPT-3: State-of-the-art language model, 175 billion parameters
- Conventional wisdom: Such models overfit.
- It is not the case in practice!

Double Descent Curve

If possible, always use wider / deeper models



Source: OpenAI

If you do have $J(\omega) = 0$, there is no need
to update ω

Outline

1 Review

2 Non-linear Optimization

3 Neural Networks

4 Stochastic Gradient Descent

5 Overparameterization

6 Lab

Let's solve the mini-project with MLPs!

Open `lab_mlp_fish_market_keras.ipynb`