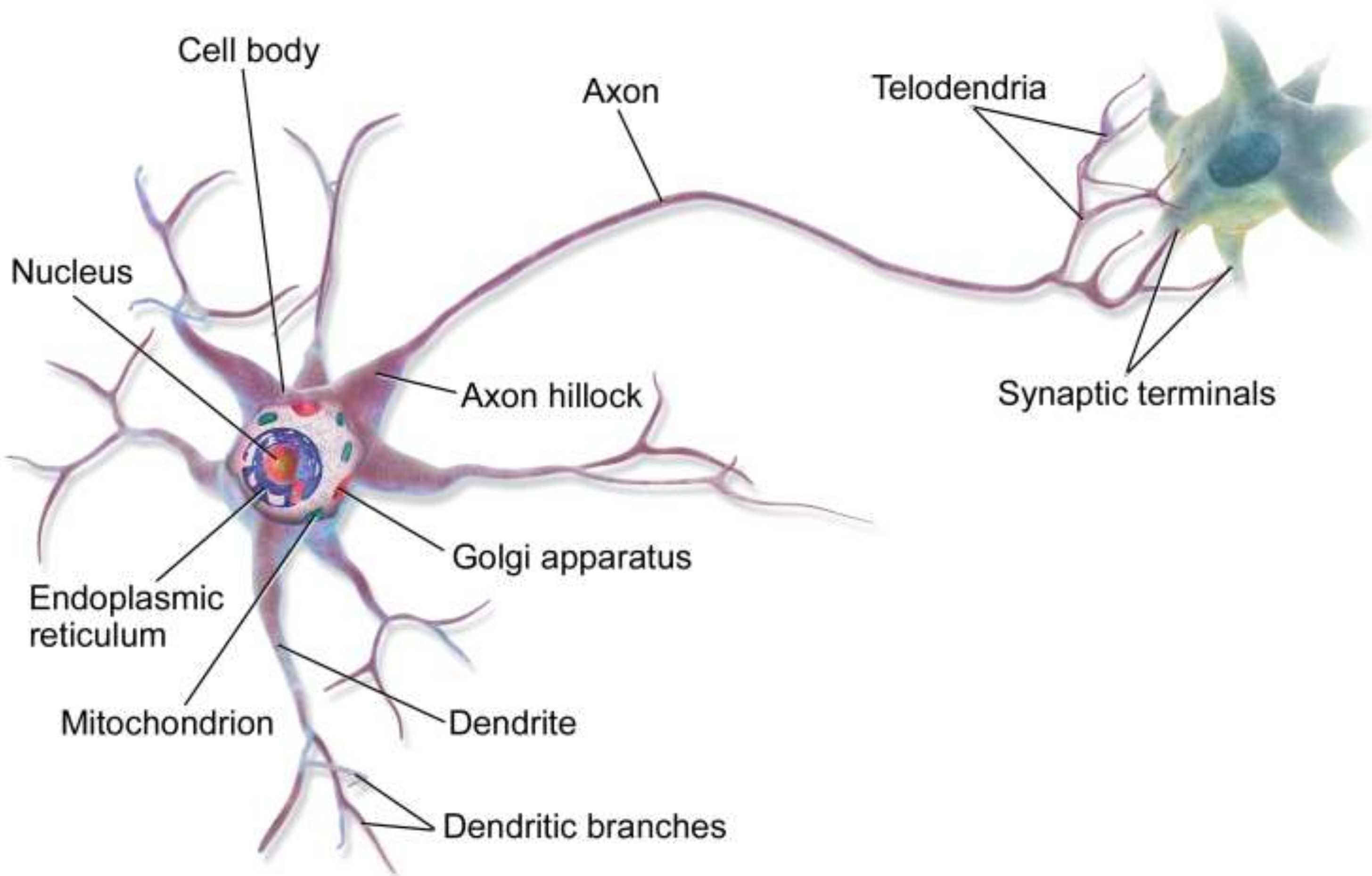
Lecture 6: Basic Artificial Neural Networks

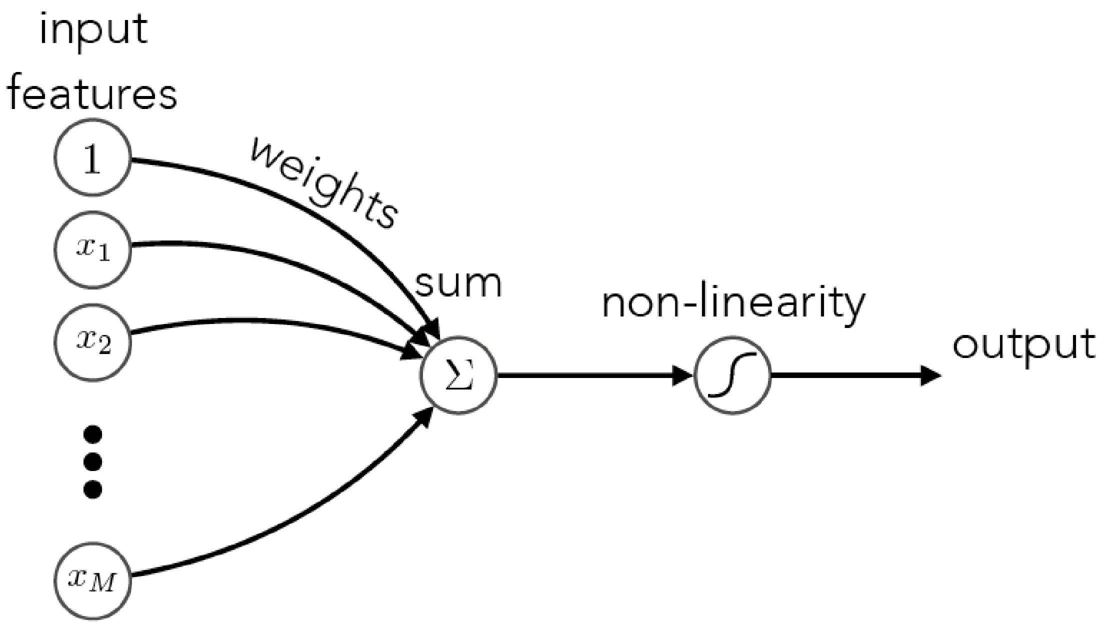
Shenghua Gao

neuron



Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic reticulum
Mitochondrion
Dendrite
Dendritic branches

Mathematical model of a neuron



artificial neuron: *weighted sum and non-linearity*

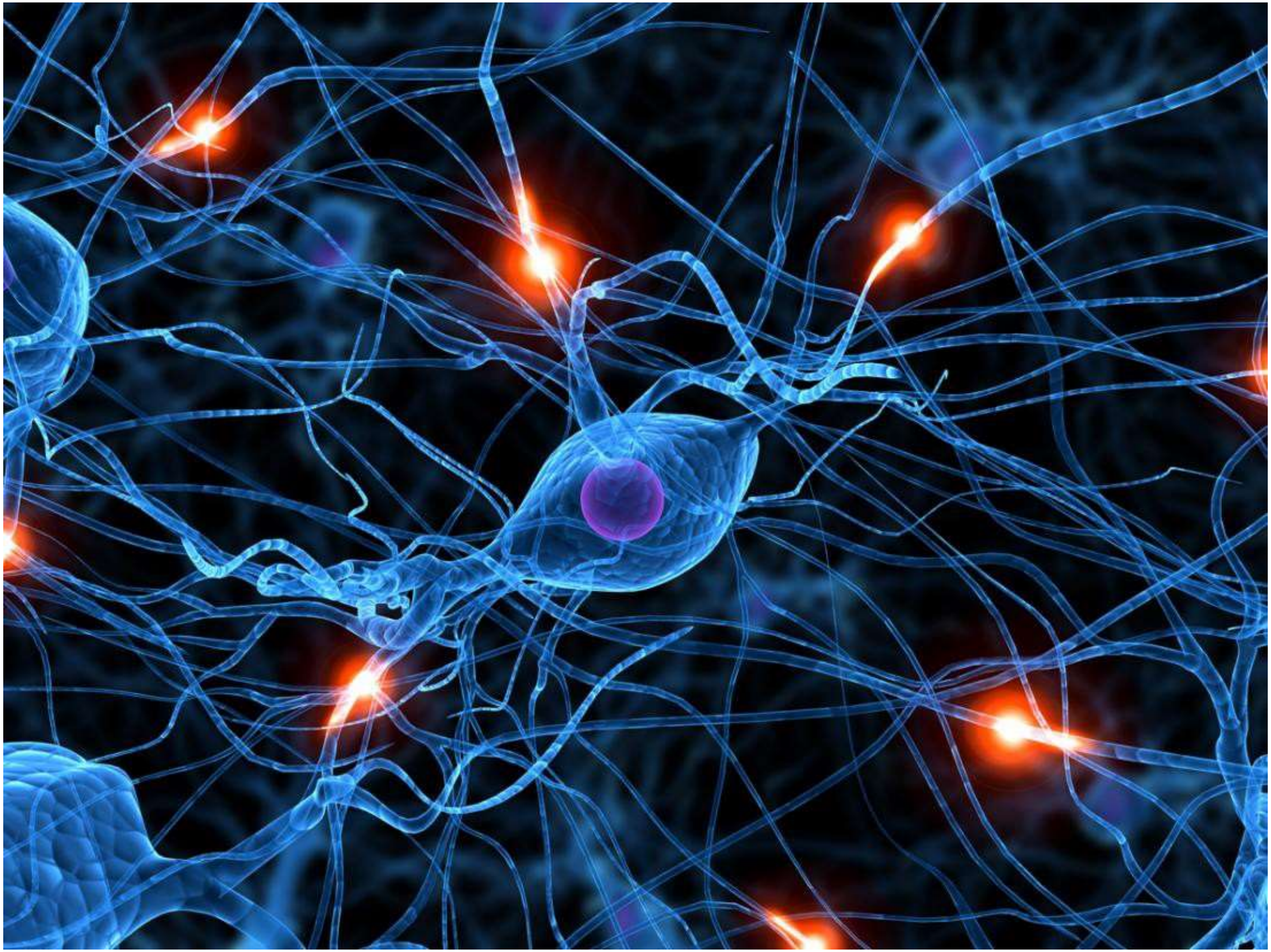
bias — input features

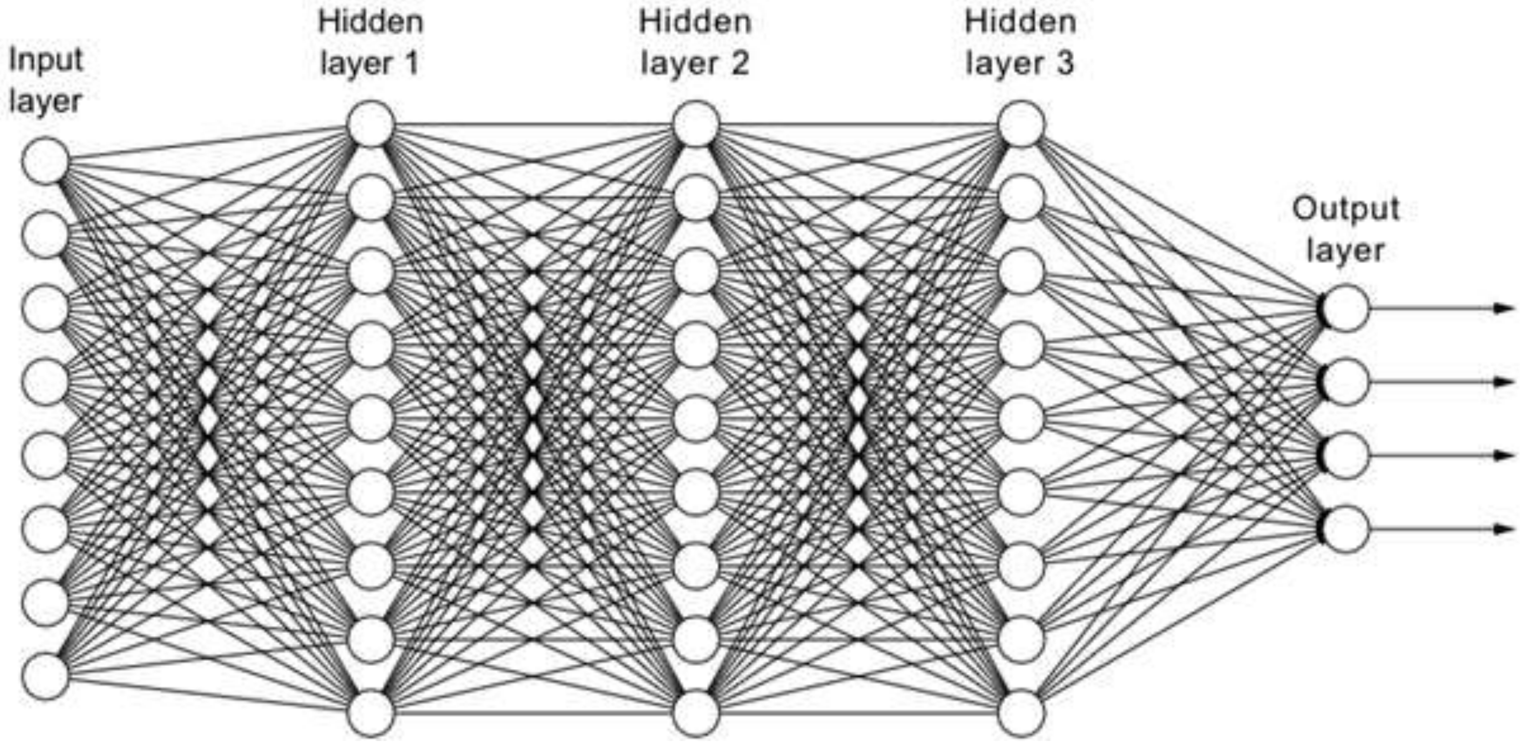
$$s = b + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M = \mathbf{w}^\mathsf{T}\mathbf{x}$$

sum — weights

$$h = g(s)$$

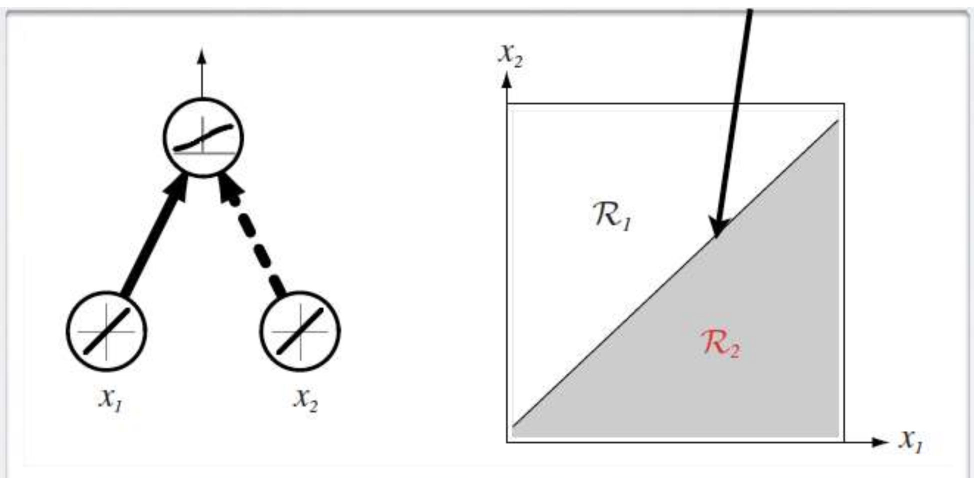
output — non-linearity — sum

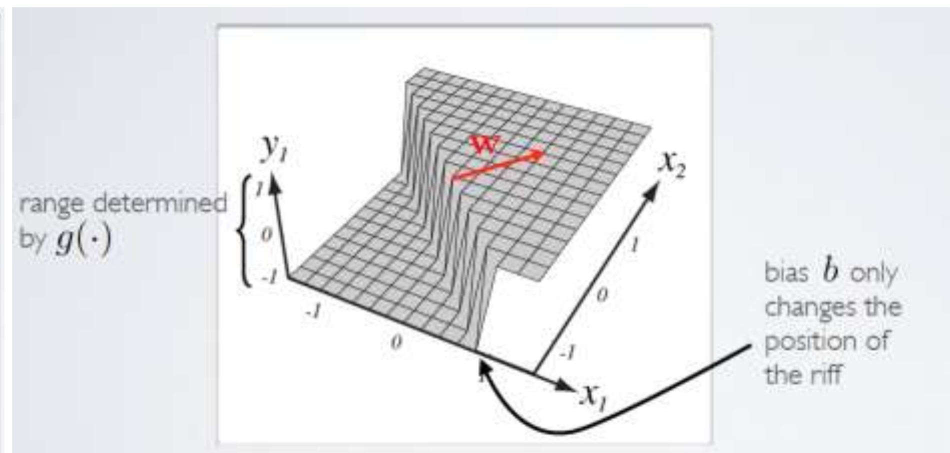



Input
layer
Hidden
layer 1
Hidden
layer 2
Hidden
layer 3
Output
layer

Capacity of single neuron

- Binary classification
  - A neuron estimates $P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$
  - Its decision boundary is linear, determined by its weights





Capacity of neural network

- Universal approximation
  - Theorem (Hornik, 1991)

    A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units.
  - The result applies for sigmoid, tanh and many other hidden layer activation functions

- Caveat: good result but not useful in practice
  - How many hidden units?
  - How to find the parameters by a learning algorithm?

General neural network

- Multi-layer neural network

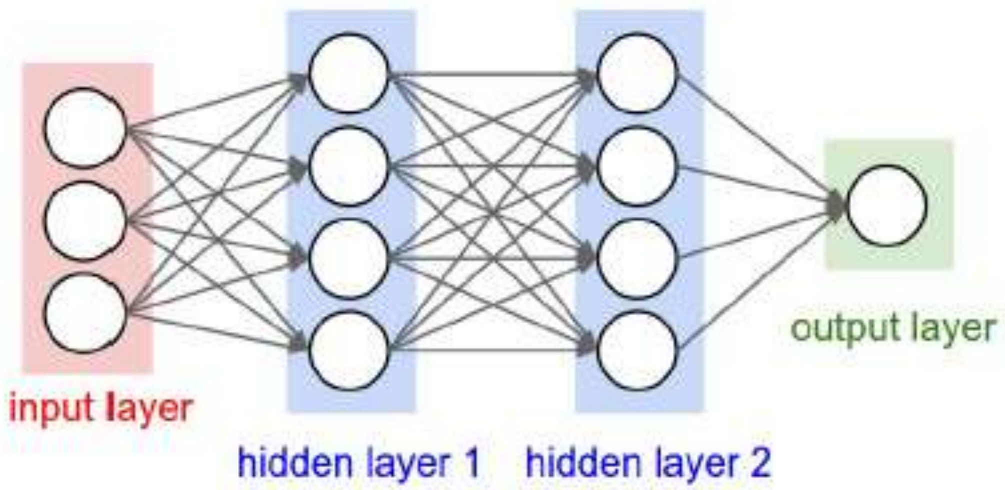
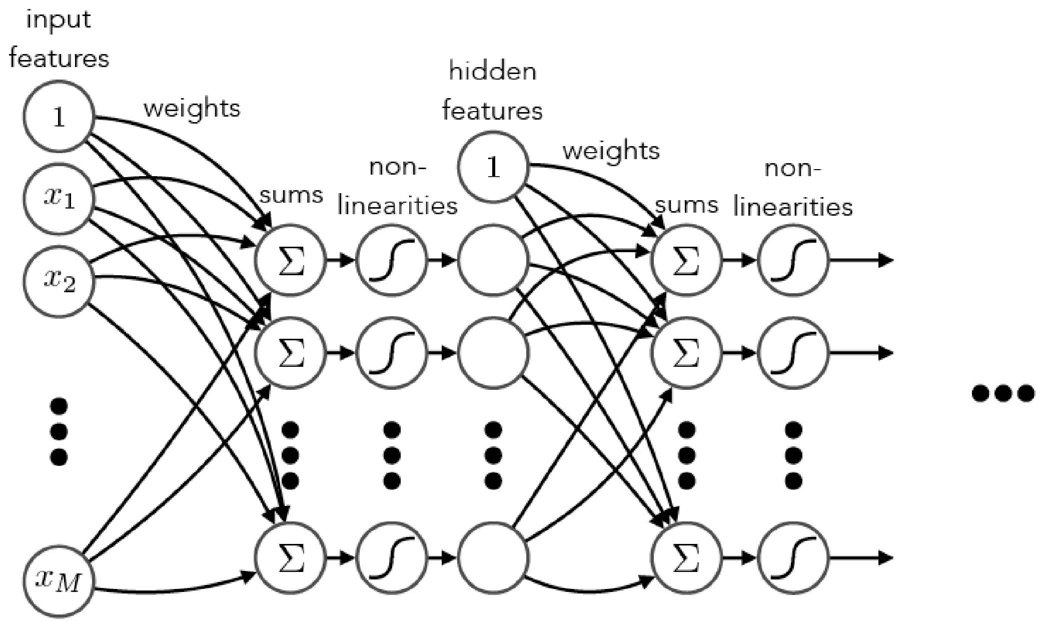


Figure : A 3-layer neural net with 3 input units, 4 hidden units in the first and second hidden layer and 1 output unit

- Naming conventions; a N-layer neural network:
  - $N-1$ layers of hidden units
  - One output layer

Multilayer networks



network: *sequence of parallelized weighted sums and non-linearities*

DEFINE $\mathbf{x}^{(0)} \equiv \mathbf{x}$, $\mathbf{x}^{(1)} \equiv \mathbf{h}$, ETC.

1st layer

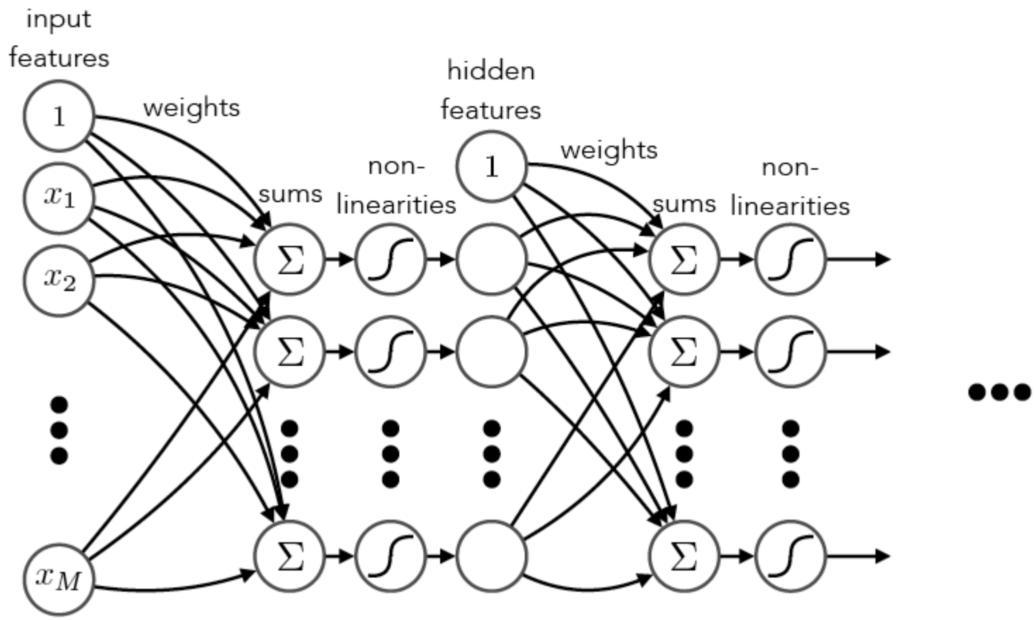
$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)\intercal}\mathbf{x}^{(0)}$$
$$\mathbf{x}^{(1)} = \sigma(\mathbf{s}^{(1)})$$

2nd layer

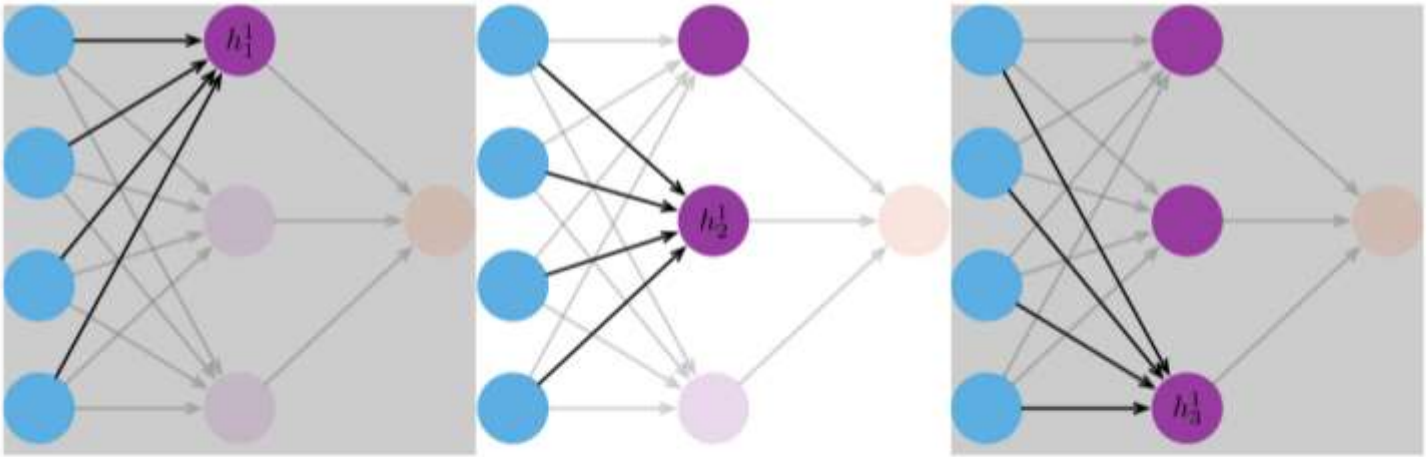
$$\mathbf{s}^{(2)} = \mathbf{W}^{(2)\intercal}\mathbf{x}^{(1)}$$
$$\mathbf{x}^{(2)} = \sigma(\mathbf{s}^{(2)})$$

•••

Multilayer networks



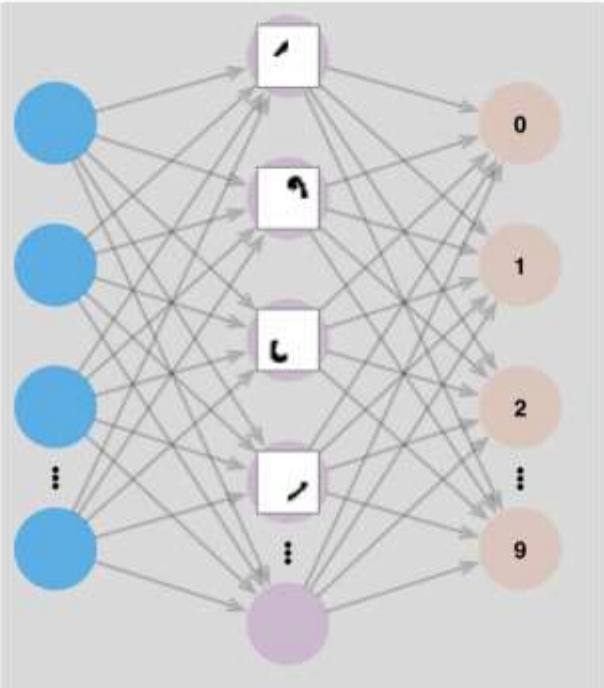
network: *sequence of parallelized weighted sums and non-linearities*



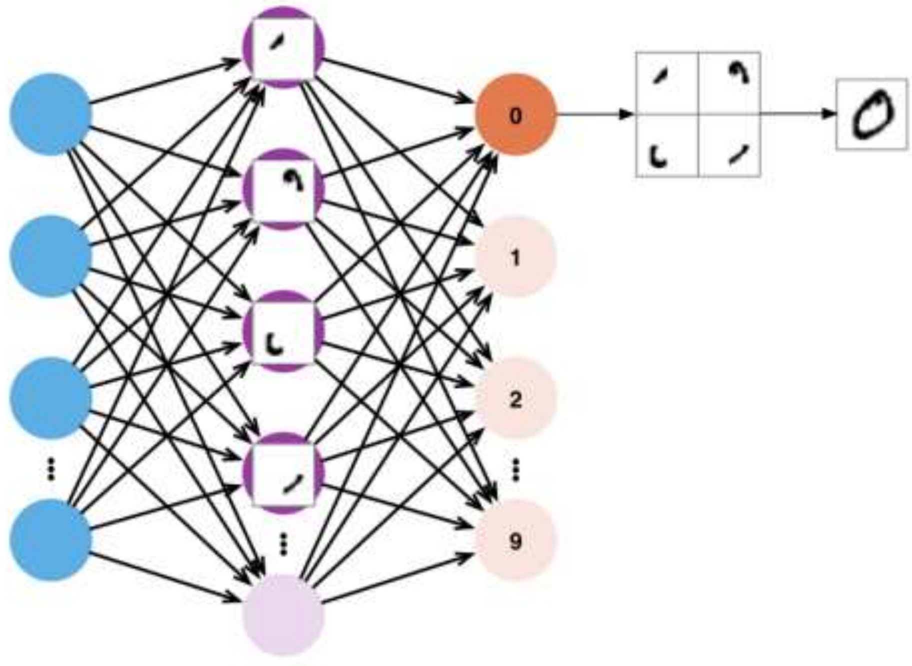
- Each hidden neuron is an **output** of a perceptron
- So you will have

$$\begin{bmatrix} h_1^1 \\ h_2^1 \\ \cdots \\ h_n^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 & \cdots & w_{1n}^1 \\ w_{21}^1 & w_{22}^1 & \cdots & w_{2n}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^1 & w_{m2}^1 & \cdots & w_{mn}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

Interpreting the Hidden Layer



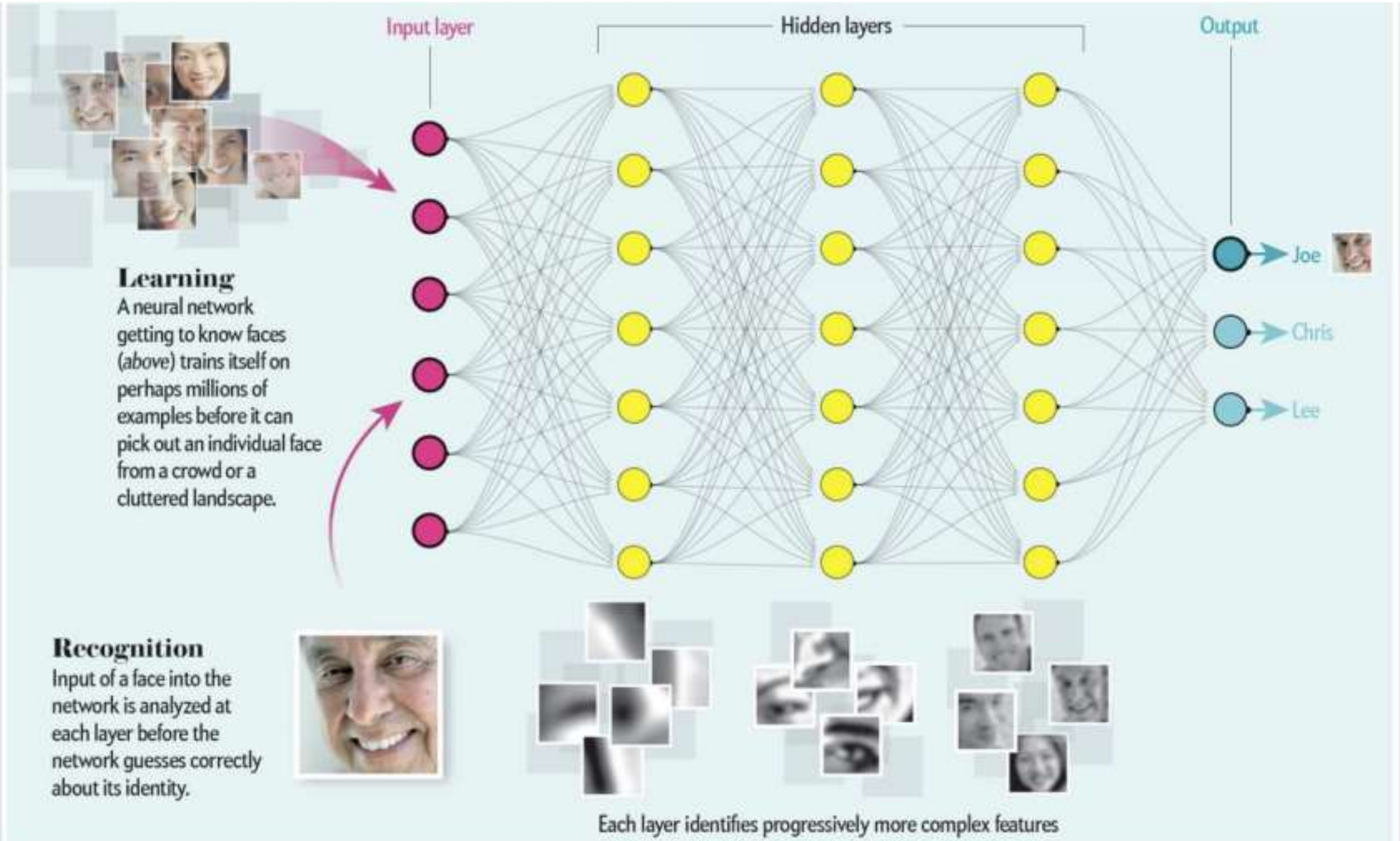
A Trained Neural Network in which each hidden neuron classifies certain pattern



Feeding a handwritten digit of 0 should trigger the 4 hidden layer neurons, and then the first output neuron

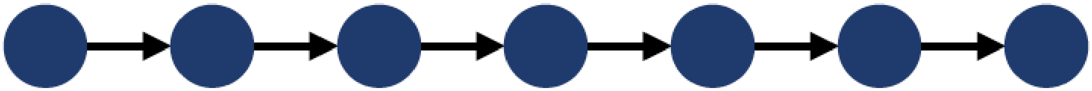
- Each hidden neuron is responsible for certain features.
- Given an object, the network identifies the most likely features.

Interpreting the Hidden Layer



Other network connectivity

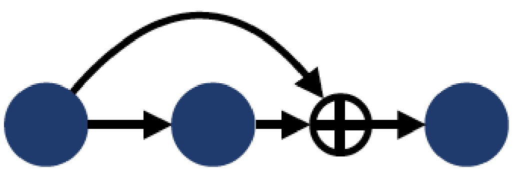
sequential connectivity: *information must flow through the entire sequence to reach the output*



information may not be able to propagate easily

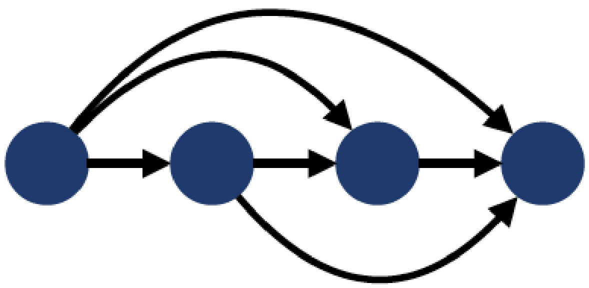
→ *make shorter paths to output*

residual & highway connections



Deep residual learning for image recognition, He et al., 2016
Highway networks, Srivastava et al., 2015

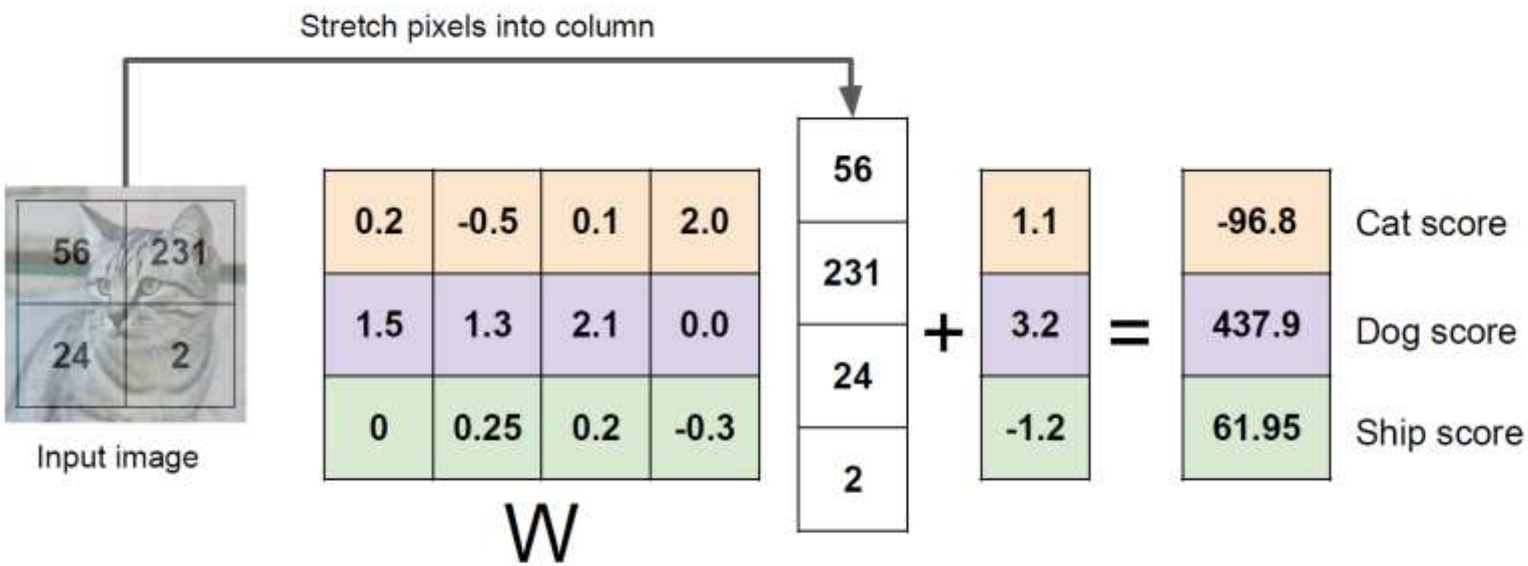
dense (concatenated) connections



Densely connected convolutional networks, Huang et al., 2017

Multiclass linear classifiers

- Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



- The winer-takes-all (WTA) prediction: one-hot encoding of its predicted label

$$y = 1 \Leftrightarrow y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad y = 2 \Leftrightarrow y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad y = 3 \Leftrightarrow y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Probabilistic outputs

scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad \boxed{s=f(x_i;W)}$$

unnormalized probabilities

	unnormalized log probabilities		unnormalized probabilities		probabilities
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

Scores are also termed as logits or unnormalized log probabilities of the classes

Example: Logistic Regression

- Learning loss: negative log likelihood

scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad \boxed{s=f(x_i;W)}$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$\boxed{L_i=-\log P(Y=y_i|X=x_i)}$$

- Given training data $\{(x_i, y_i): 1 \le i \le n\}$ i.i.d. from distribution D
- Find $y = f(x) \in \mathcal{H}$ that minimizes $\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$
- s.t. the expected loss is small

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

Empirical loss

Logistic Regression

- Learning loss: example

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$ unnormalized probabilities	$\xrightarrow{\text{normalize}}$ probabilities
cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

→ L_i = -log(0.13)
= **0.89**

Logistic Regression

- Learning loss: questions

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q: What is the min/max possible loss L_i?

cat	**3.2**		**24.5**		**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	exp →	164.0	normalize →	0.87	
frog	-1.7		0.18		0.00	

unnormalized log probabilities

probabilities

Logistic Regression

- Learning loss: questions

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q2: Usually at initialization W is small so all s ≈ 0.
What is the loss?

	unnormalized log probabilities			probabilities
cat	**3.2**	**24.5**	**0.13**	L_i = -log(0.13) = **0.89**
car	5.1	164.0	0.87	
frog	-1.7	0.18	0.00	

$\xrightarrow{\text{exp}}$ $\xrightarrow{\text{normalize}}$

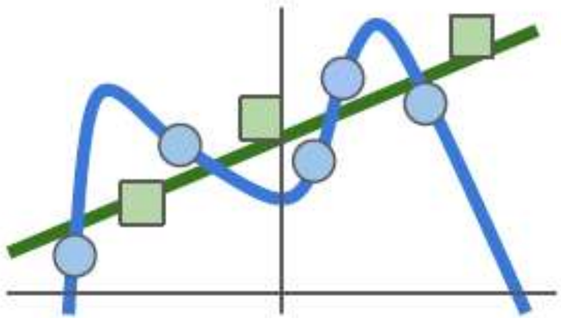
Learning with regularization

- Constraints on hypothesis space
  - Similar to Linear Regression

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Learning with regularization

- Regularization terms

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Max norm regularization (might see later)

- Priors on the weights
  - Bayesian: integrating out weights
  - Empirical: computing MAP estimate of W

Optimization: gradient descent

- Stochastic gradient descent

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

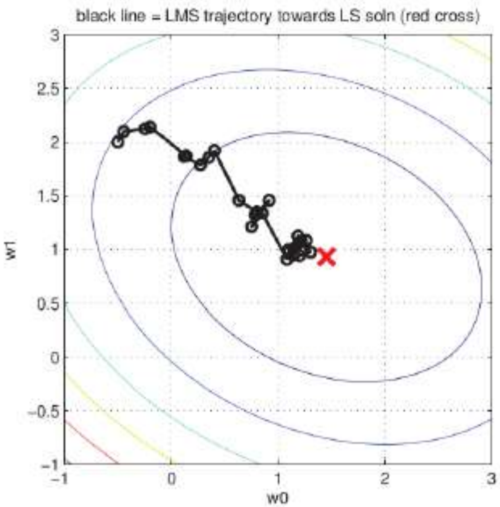
Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

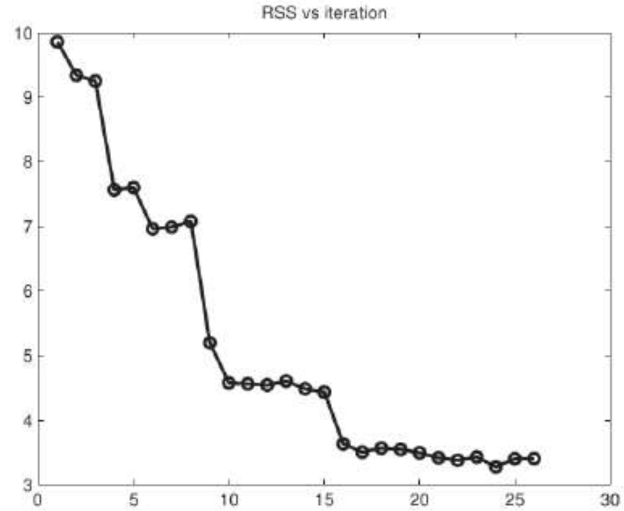
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Optimization: gradient descent

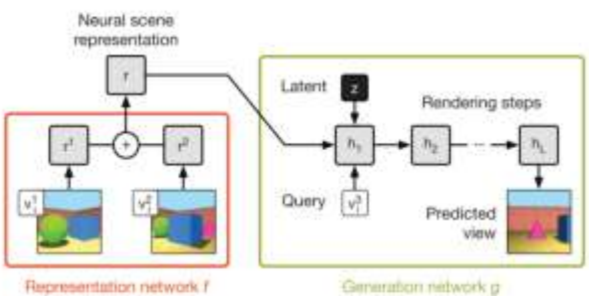
- Stochastic gradient descent





- the objective does not always decrease for each step
- comparing to GD, SGD needs more steps, but each step is cheaper
- mini-batch, say pick up 100 samples and do average, may accelerate the convergence

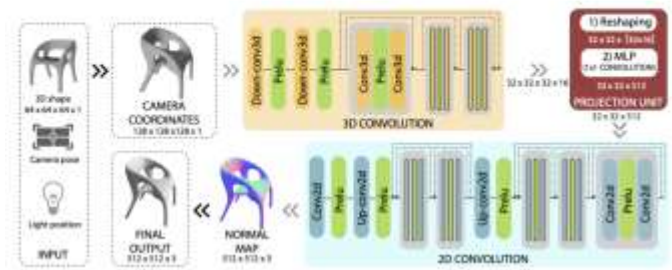
Modern MLP as Implicit Representation



Generative Query Networks
[Eslami et al. 2018]

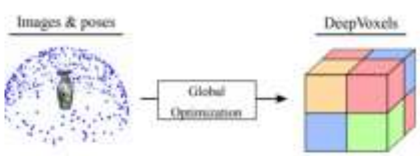
[Flynn et al., 2016; Zhou et al., 2018b;
Mildenhall et al. 2019]

Multiplane Images (MPIs)

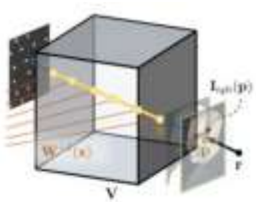


RenderNet [Nguyen-Phuoc et al. 2018]

Voxel Grids + CNN decoder

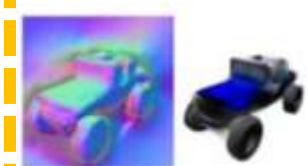


DeepVoxels
[Sitzmann et al. 2019]

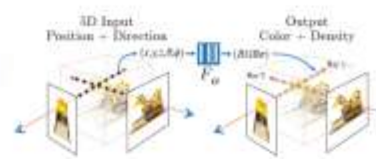


Neural Volumes
[Lombardi et al. 2019]

Voxel Grids + Ray Marching



SRN
[Sitzmann et al. 2019b]



NeRF
[Mildenhall et al. 2020]

IDR
[Yariv et al. 2020]

Implicit Fields