Lecture 21
Neural networks

Simple neural networks; activation functions; training

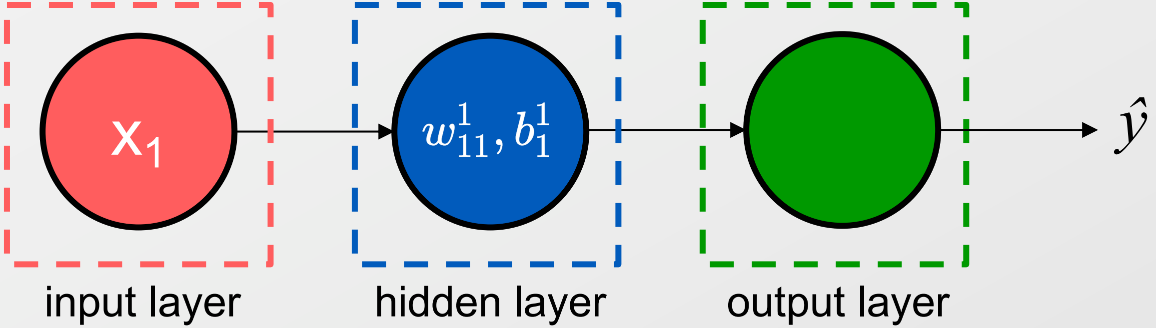
Prof. David A. Kofke
CE 500 – Modeling Potential-Energy Surfaces
Department of Chemical & Biological Engineering
University at Buffalo

University at Buffalo
The State University of New York

Here is perhaps the simplest example of a neural network

- We have a set of training and test data

- We define a neural network to represent it



Training

x	y
1.0	5.082
2.0	7.950
3.0	10.96
4.0	14.05
5.0	17.04

Test

x	y
1.2	5.601
6.	19.98

- Output from x_1 is transformed according to $\hat{y} = \underbrace{w^1_{11}}_{\text{weight}} x_1 + \overbrace{b_1}^{\text{bias}}$

- The NN "learns" from the training data, meaning it determines values of w^1_{11}, b^1_1 that best describe the data

Learning is done by evaluating a loss function and adjusting parameters using gradient

$$\hat{y} = w_{11}^{1} x_1 + b_1$$

- Loss function

$$L(w_{11}^{1}, b_1^{1}) = \sum \left(y_i - \hat{y}(x_{1i}; w_{11}^{1}, b_1^{1})\right)^2$$

- A general parameter θ is updated according to

$$\theta(\text{new}) = \theta(\text{old}) - \gamma \frac{\partial L}{\partial \theta}$$

  - γ is a hyperparameter that controls the learning process
    - Too-large γ risks moving parameters too far from current values
    - Too-small γ increases amount of iterations needed to learn
  - Gradients are

$$\frac{\partial L}{\partial w_{11}^{1}} = -2 \sum (y_i - \hat{y}(x_{1i})) x_{1i} \qquad \frac{\partial L}{\partial b_1^{1}} = -2 \sum (y_i - \hat{y}(x_{1i}))$$

Iteration entails calculation of loss function, update of parameters, repeat

W_{11}^1	b_1^1	$\frac{\partial L}{\partial W_{11}^1}$	$\frac{\partial L}{\partial b_1^1}$	L
1	0	−281.	−80.2	361.391
3.80552	0.801755	52.1079	12.0077	13.6995
3.28444	0.681677	−8.81311	−4.8254	1.97225
3.37257	0.729932	2.32893	−1.69893	1.53215
3.34929	0.746921	0.276786	−2.22772	1.46799

ŷ(1)	ŷ(2)	ŷ(3)	ŷ(4)	y
4.60728	3.96612	4.10251	4.09621	5.082
8.4128	7.25056	7.47508	7.44549	7.950
12.2183	10.535	10.8477	10.7948	10.96
16.0238	13.8195	14.2202	14.1441	14.05
19.8294	17.1039	17.5928	17.4933	17.04

$$L(w_{11}^1, b_1^1) = \sum \left(y_i - \hat{y}(x_{1i}; w_{11}^1, b_1^1)\right)^2$$

$$\theta(\text{new}) = \theta(\text{old}) - \gamma \frac{\partial L}{\partial \theta}$$

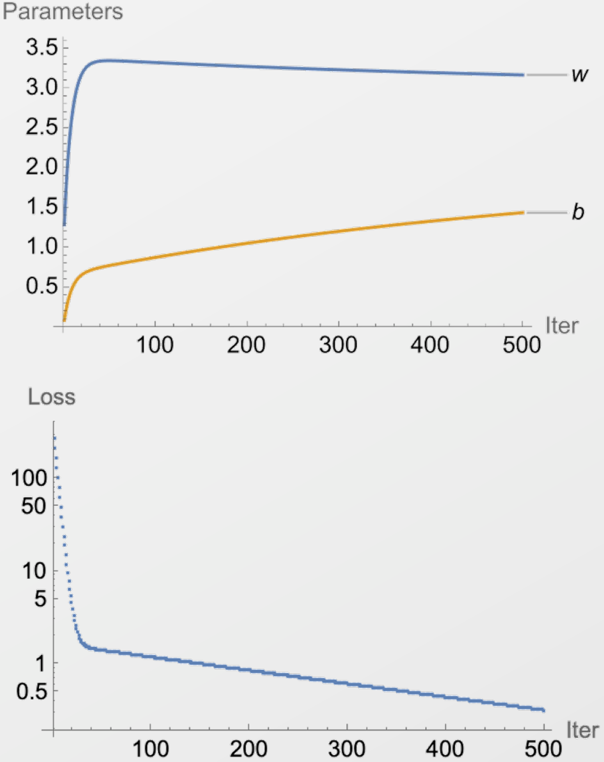
$$\frac{\partial L}{\partial w_{11}^1} = -2 \sum (y_i - \hat{y}(x_{1i}))x_{1i}$$

$$\frac{\partial L}{\partial b_1^1} = -2 \sum (y_i - \hat{y}(x_{1i}))$$

Learning rate can affect performance

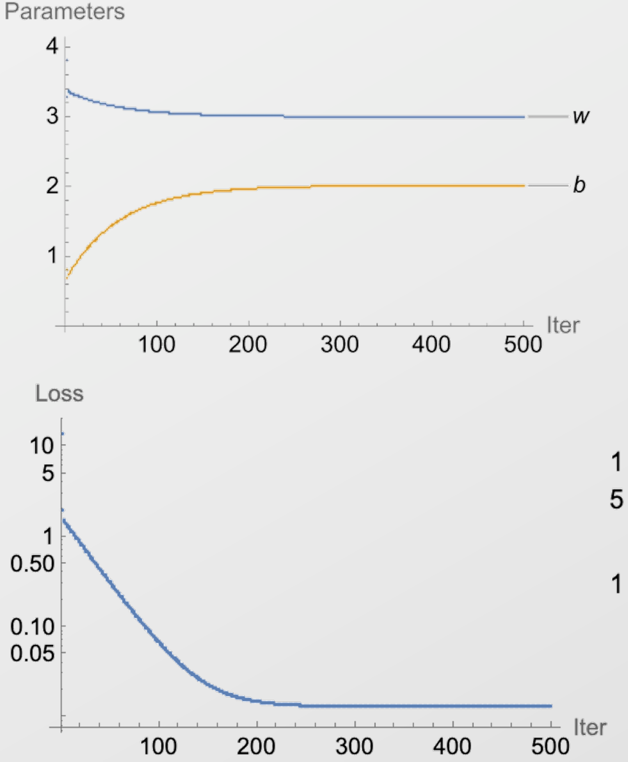
Learning too slow

γ =0.001



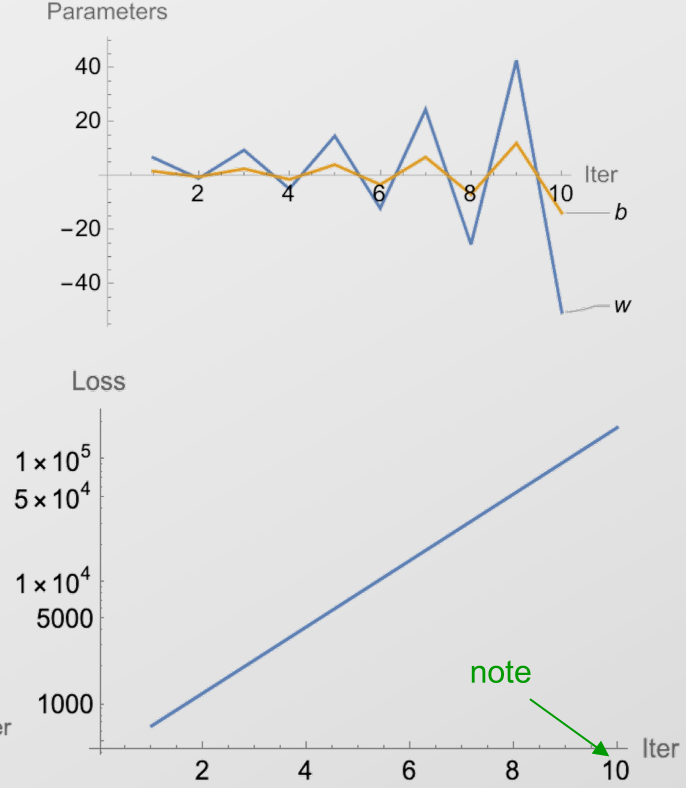
Learning just right

γ =0.01



Learning too fast

γ =0.02



Comparison to test set is used to evaluate NN model

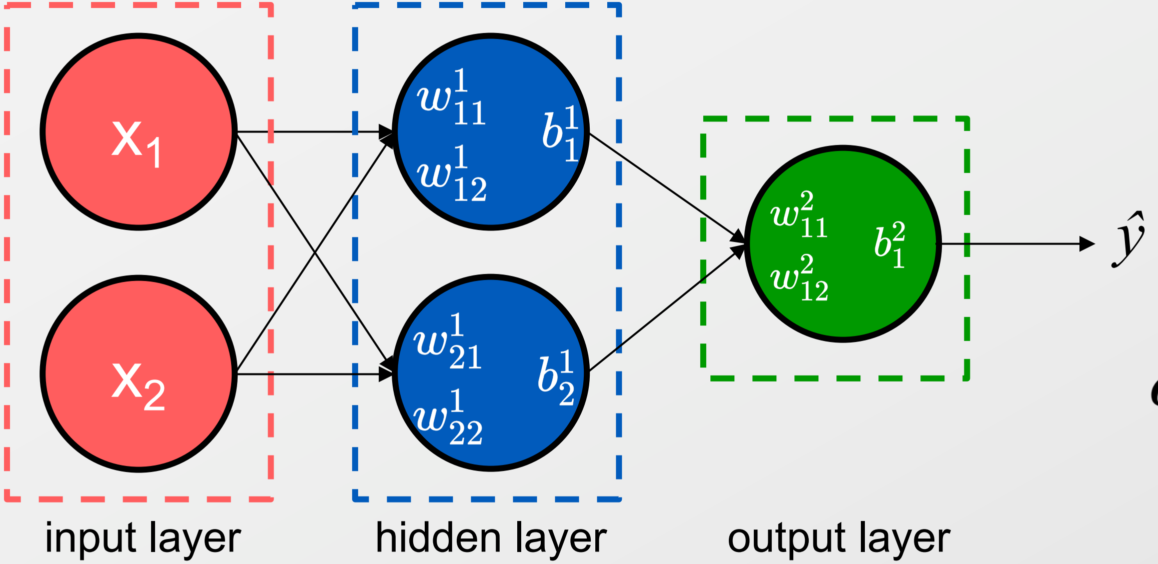
Test

x	y
1.2	5.601
6.	19.98

NN model

$\hat{y}$
5.61505
20.0216

We can make a more complicated NN by adding another input, and another node



Number of nodes in hidden layer is independent of number in input layer; they happen to have been equal in our two examples so far

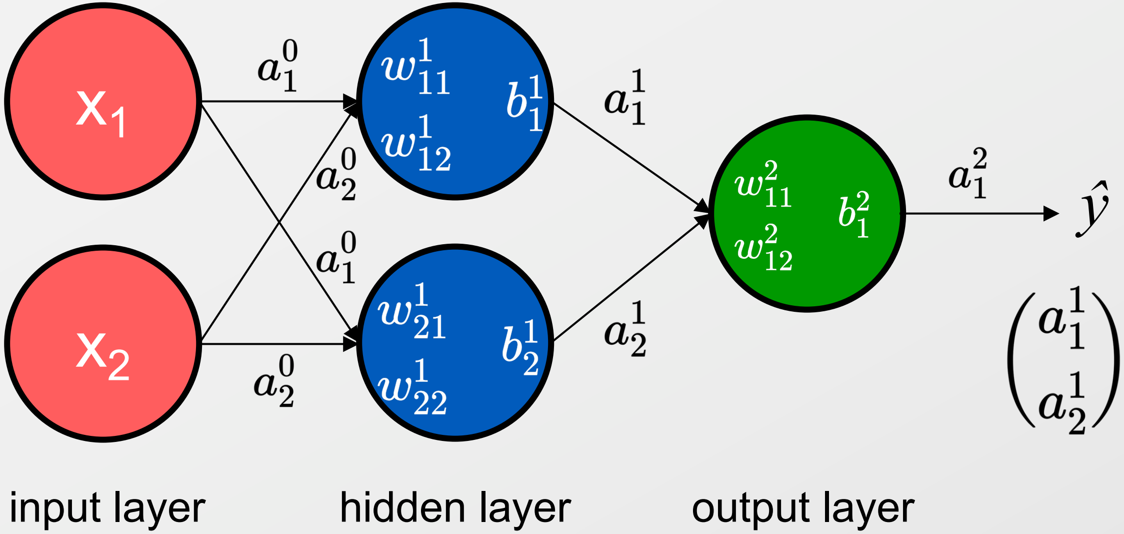
- Now we have 9 parameters
- The output (activation, a) of each node is given from all of its inputs

$$a^1_1 = w^1_{11}x_1 + w^1_{12}x_2 + b^1_1$$

- All nodes in layer l, in matrix form:

$$a^l = W^l a^{l-1} + b^l$$

The combining of the inputs to produce the output can be described by matrix operations



$$a^l = W^l a^{l-1} + b^l$$

1$^{\text{st}}$ layer

$$\begin{pmatrix} a^1_1 \\ a^1_2 \end{pmatrix} = \begin{pmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \end{pmatrix} \begin{pmatrix} a^0_1 \\ a^0_2 \end{pmatrix} + \begin{pmatrix} b^1_1 \\ b^1_2 \end{pmatrix}$$

2$^{\text{nd}}$ (output) layer

$$\begin{pmatrix} a^2_1 \end{pmatrix} = \begin{pmatrix} w^2_{11} & w^2_{12} \end{pmatrix} \begin{pmatrix} a^1_1 \\ a^1_2 \end{pmatrix} + \begin{pmatrix} b^2_1 \end{pmatrix}$$

Weight indexes

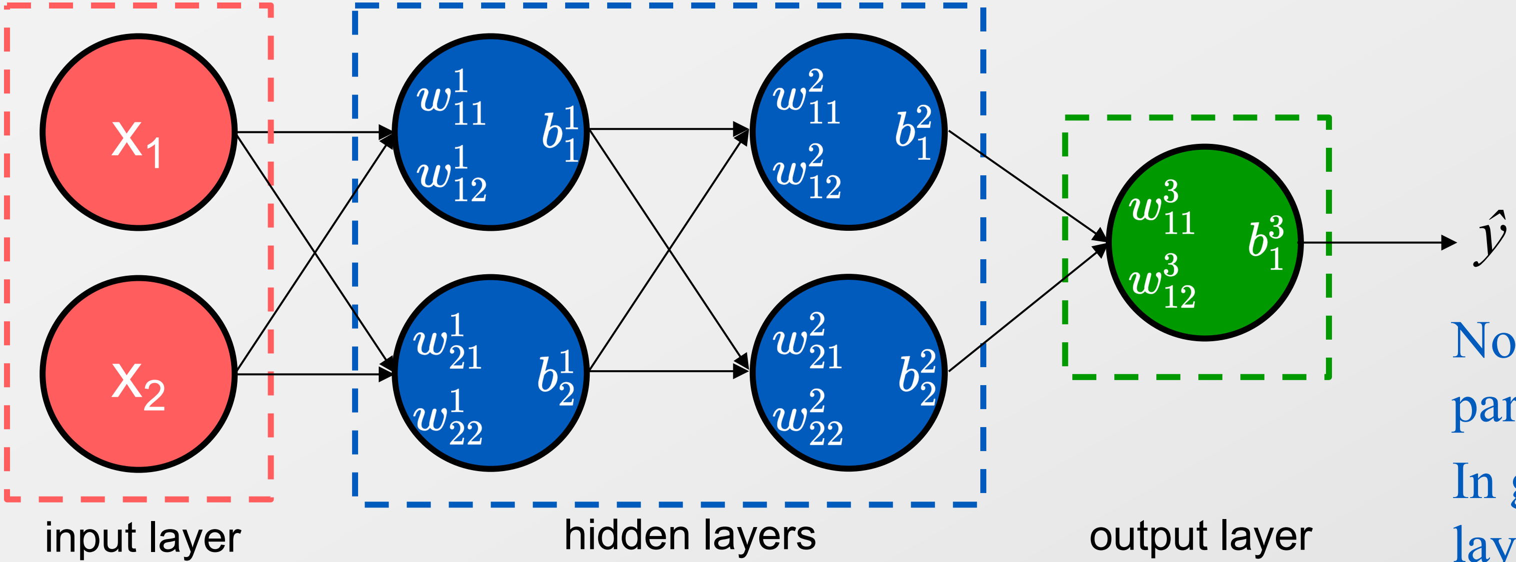
$$w^l_{jk}$$

Layer number

Number of node receiving input in layer l

Number of node providing output from layer l-1

More complexity can be introduced by adding more hidden layers



Number of nodes in hidden layer is independent of number in input layer; they happen to have been equal in our two examples so far

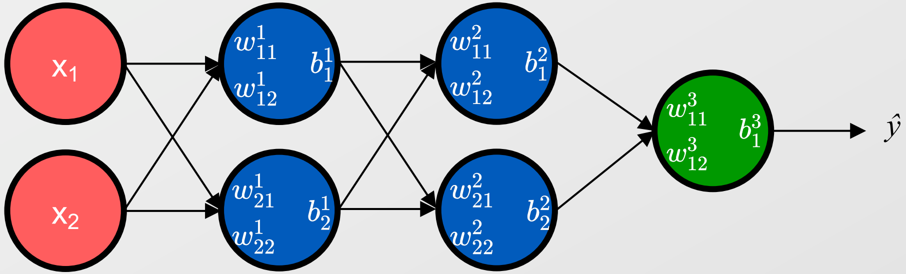
Now we have 15 parameters

In general, for L layers with N_l nodes in layer l

$$N_{\text{params}} = \sum_{l=1}^{L} (N_{l-1} + 1) N_l$$

So far, neural networks appear to be exactly the same as a simple multilinear regression

- The output vector can be written succinctly as a set of nested linear operations

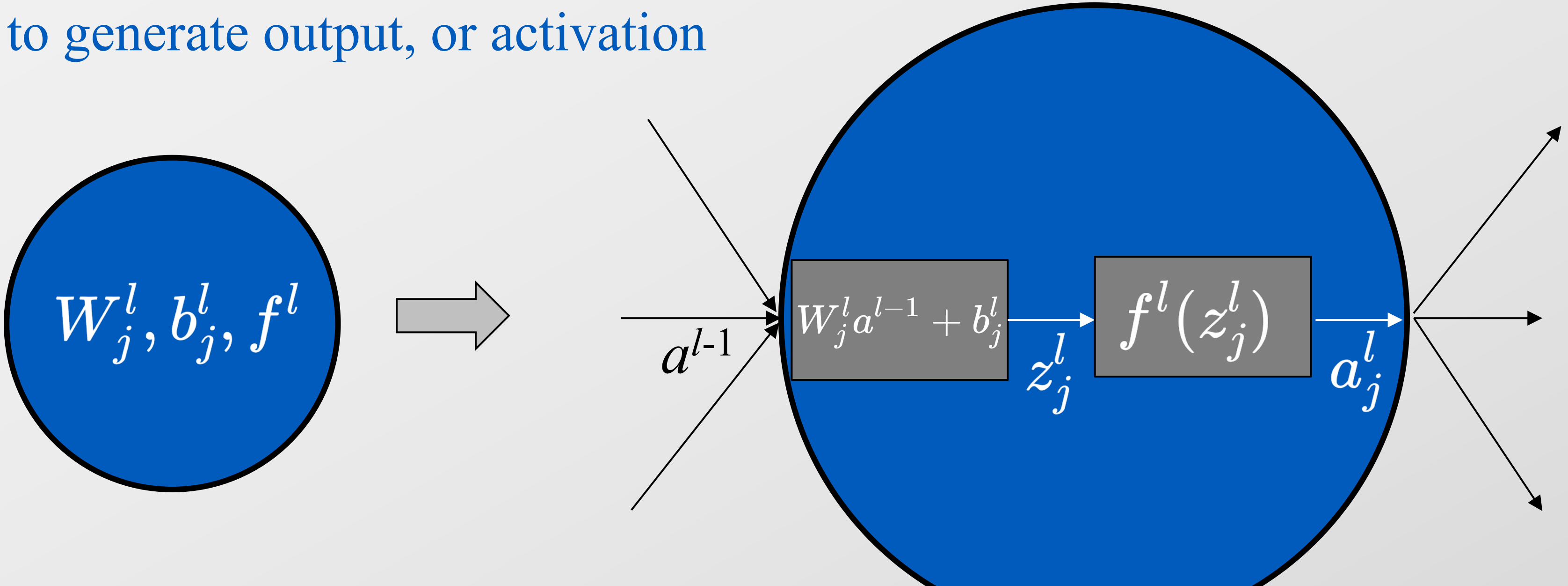


$$
\begin{aligned}
\hat{y} &= W^3 a^2 + b^3 \\
&= W^3(W^2 a^1 + b^2) + b^3 \\
&= W^3(W^2(W^1 a^0 + b^1) + b^2) + b^3 \\
&= W^3 W^2 W^1 x + (W^3 W^2 b^1 + W^3 b^2 + b^3) \\
&= Ax + c
\end{aligned}
$$

- This is just a linear combination of the input data x
  - We examined this already, and showed it has an explicit solution

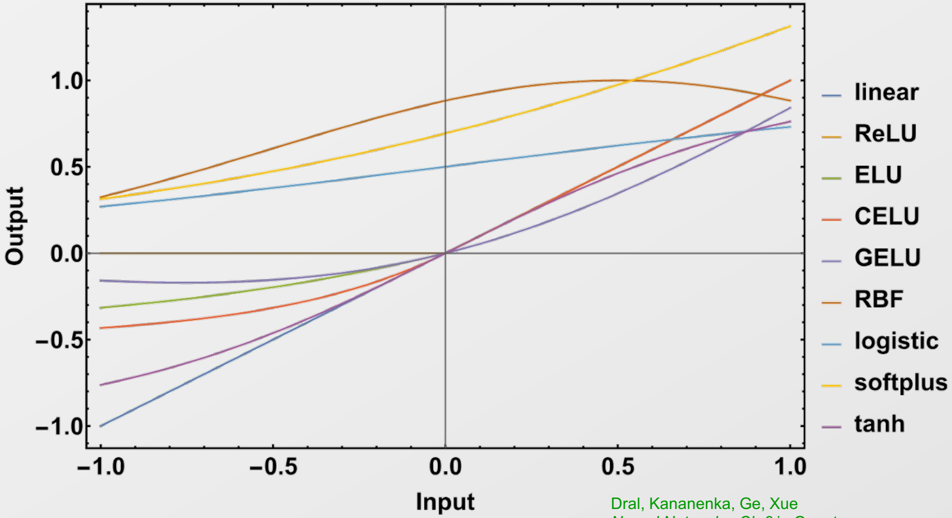
Neural networks are made nonlinear by introducing an *activation function* with each node

- The node performs two operations
  - Calculate linear combination of inputs
  - Pass combined inputs though a nonlinear function to generate output, or activation



The activation function is what gives neural networks their versatility

- Several forms are in use
  - Same for for all nodes in a layer
  - May differ in different layers



Dral, Kananenka, Ge, Xue
Neural Networks, Ch.8 in Quantum Chemistry in the Age of Machine Learning
https://doi.org/10.1016/B978-0-323-90049-2.00011-1

TABLE 1 Overview of a selection of popular activation functions.

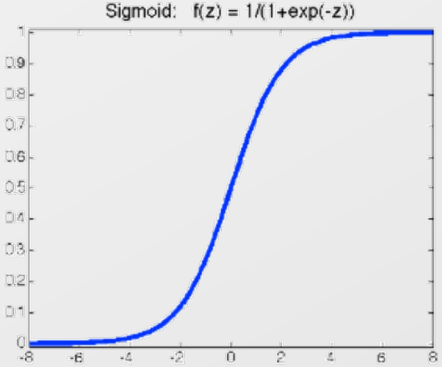
Names	Equation
Linear function Identity function [2]	$g(v)=v$
Rectified linear unit (ReLU) [2]	$g(v)=\max(0,v)$
Exponential linear unit (ELU) [5]	$g(v)=\begin{cases} v & \text{if } v\geq 0 \\ a(\exp(v)-1) & \text{otherwise} \end{cases}$ where a is a parameter
Continuously differentiable exponential linear unit (CELU) [6]	$g(v)=\begin{cases} v & \text{if } v\geq 0 \\ a(\exp(v/a)-1) & \text{otherwise} \end{cases}$ where a is a parameter
Gaussian error linear unit (GELU) [7]	$g(v)=v\cdot\frac{1}{2}\left[1+\text{erf}\left(\frac{v}{\sqrt{2}}\right)\right]$ Faster approximated versions: $g(v)=0.5v\left(1+\tanh\left[\sqrt{2/\pi}\left(v+0.044715v^3\right)\right]\right)$ $g(v)=v\cdot\sigma(1.702v)=v\cdot 1/[1+\exp(-1.702v)]$
Radial basis function (RBF) [1,2]	$g(v)=\exp(-a(v-c)^2)$ where a and c are parameters
Logistic sigmoid function [1,2]	$g(v)=\sigma(v)=1/[1+\exp(-v)]$
Softplus function [2]	$g(v)=\log(1+\exp(v))$
Hyperbolic tangent function [2]	$g(v)=T(v)=\tanh(v)$

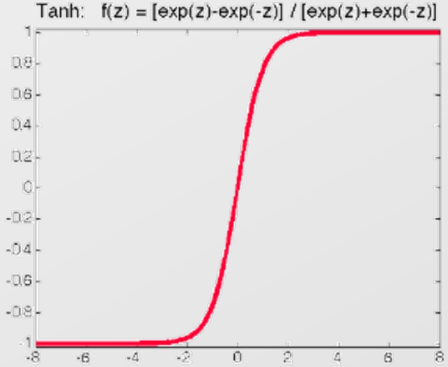
The activation function is what gives neural networks their versatility

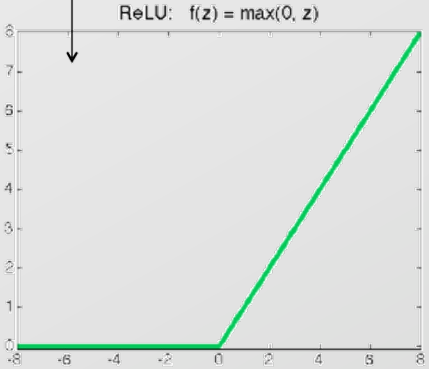
Most commonly used activation functions:

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$

Most popular recently for deep learning





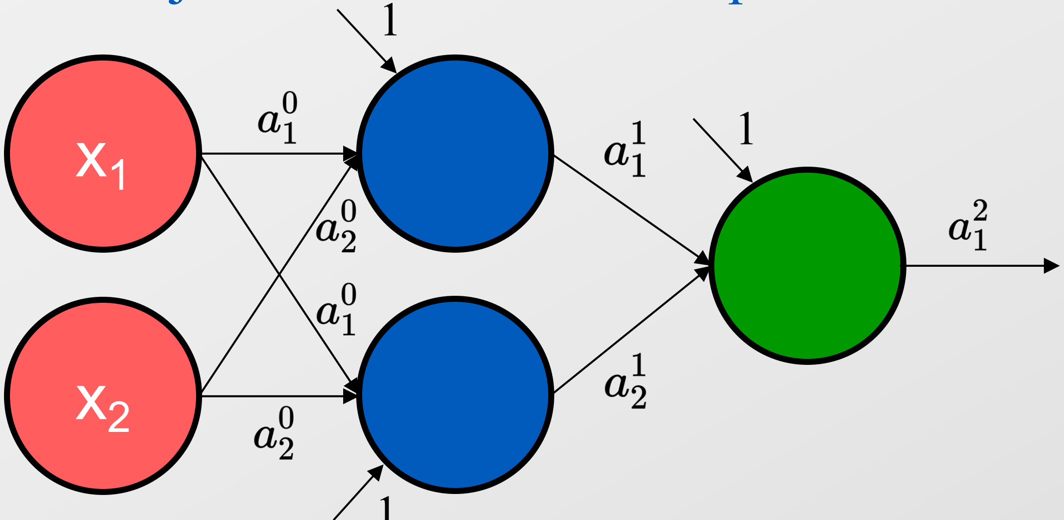


Shengyang Sun
CSC 411, U Toronto

Before going further, let's simplify notation by absorbing *b* into *W*

- Each node implicitly has a unit input, along with inputs from previous layer
  - Last column of *W* plays the role of *b*
  - This just makes notation simpler

$$\begin{pmatrix} a_1^l \\ a_2^l \end{pmatrix} = \begin{pmatrix} w_{11}^l & w_{12}^l & w_{13}^l \\ w_{21}^l & w_{22}^l & w_{23}^l \end{pmatrix} \begin{pmatrix} a_1^{l-1} \\ a_2^{l-1} \\ 1 \end{pmatrix}$$



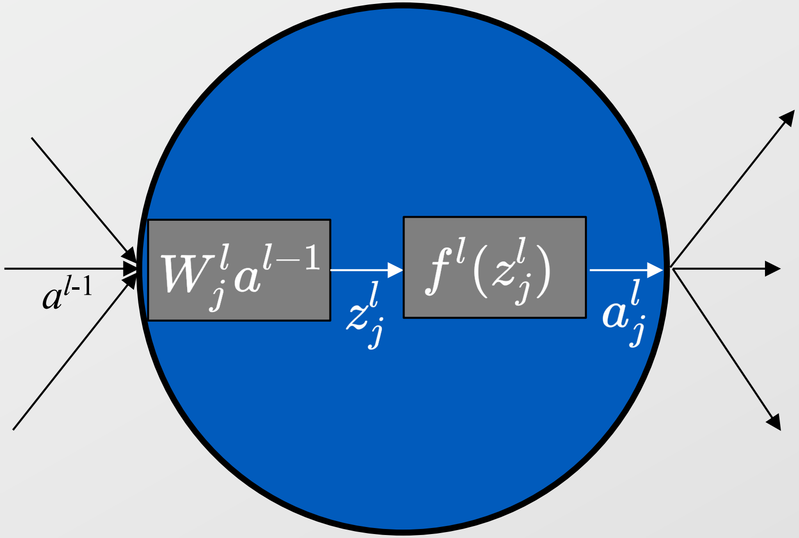
$$a^l = W^l a^{l-1}$$

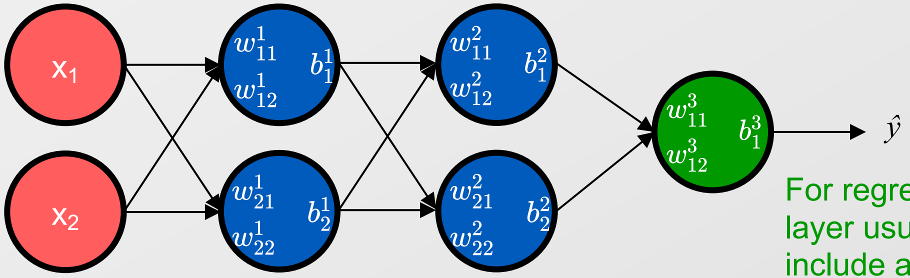
instead of

$$a^l = W^l a^{l-1} + b^l$$

Neural network can be written as a system of nested functions

$$\begin{aligned}\hat{y} &= W^3 a^2 \\ &= W^3 f^2(z^2) \\ &= W^3 f^2(W^2 a^1) \\ &= W^3 f^2(W^2 f^1(z^1)) \\ &= W^3 f^2(W^2 f^1(W^1 x))\end{aligned}$$





For regression, the output layer usually does not include an activation function

Backpropagation is used to compute weight derivatives efficiently

- Regardless of network configuration, weights are computed using gradient descent, modulated by the learning rate γ

$$\theta(\text{new}) = \theta(\text{old}) - \gamma \frac{\partial L}{\partial \theta}$$

loss function

- The necessary derivatives can be complicated to compute
- Working backwards, using the chain rule, makes it easy

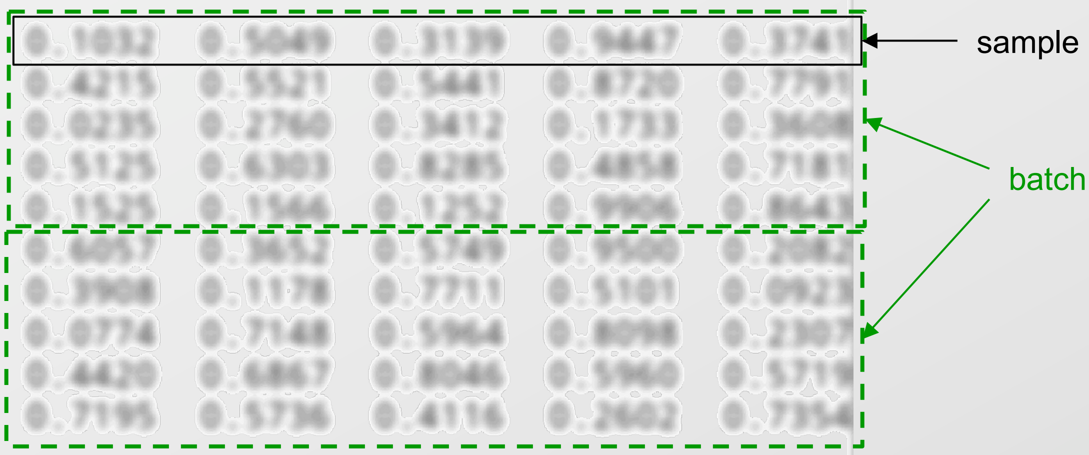
$$\frac{dC}{da^L} \cdot \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \cdot \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \cdot \ldots \cdot \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x}$$

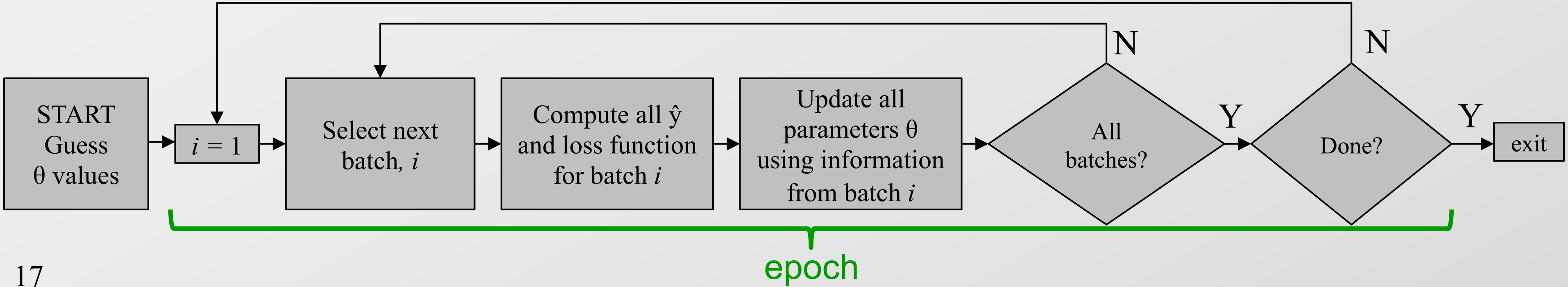
number of layers

Samples, batches and epochs each describe an amount of training data

- Learning is performed on batches
- An *epoch* is reached when all batches in training set are used

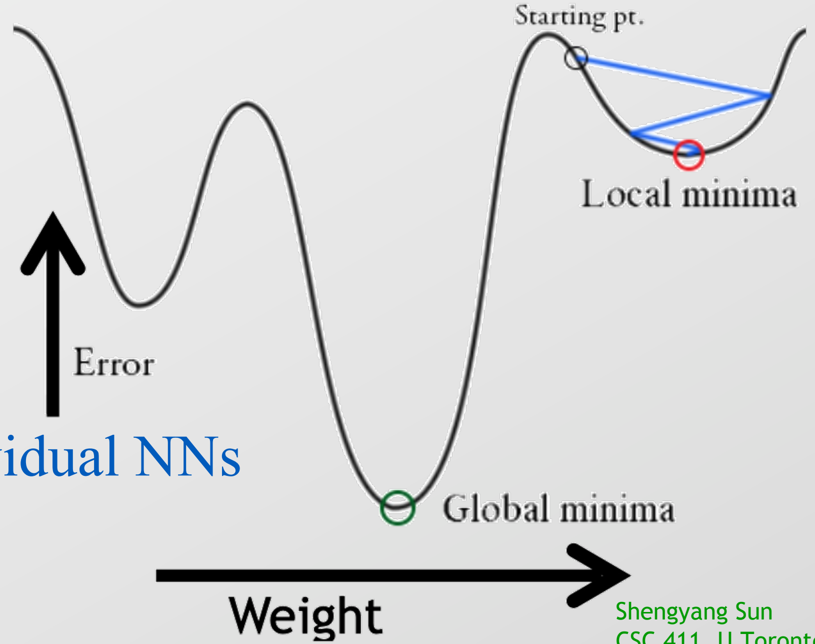
Training set





Training neural networks is a non-convex optimization problem

- This means we can run into many local minima during training
- Having many solutions is not necessarily bad
  - very different parameters can give NN models making similar predictions for points similar to training points



- Combine NNs in an ensemble
  - ensemble mean more stable than individual NNs
  - use deviation within ensemble for uncertainty quantification

Suggested Reading/Viewing

- Dral, Pavlo O; Kananenka, Alexei A; Ge, Fuchun; Xue, Bao-Xin, Chapter 8, Neural Networks. In *Quantum Chemistry in the Age of Machine Learning*.
  - https://doi.org/10.1016/B978-0-323-90049-2.00011-1
  - Posted on UBLearns
- https://en.wikipedia.org/wiki/Backpropagation