NEURAL NETWORK MODELS

Dr. Aric LaBarr

Institute for Advanced Analytics

NEURAL NETWORK STRUCTURE

Neural Networks Overview

- Neural networks are considered “black-box” models.
  - Complex and hard to decipher relationships between variables and the target.

- Potential to model very complicated patterns in datasets – both for continuous and categorical targets.

Neural Networks History

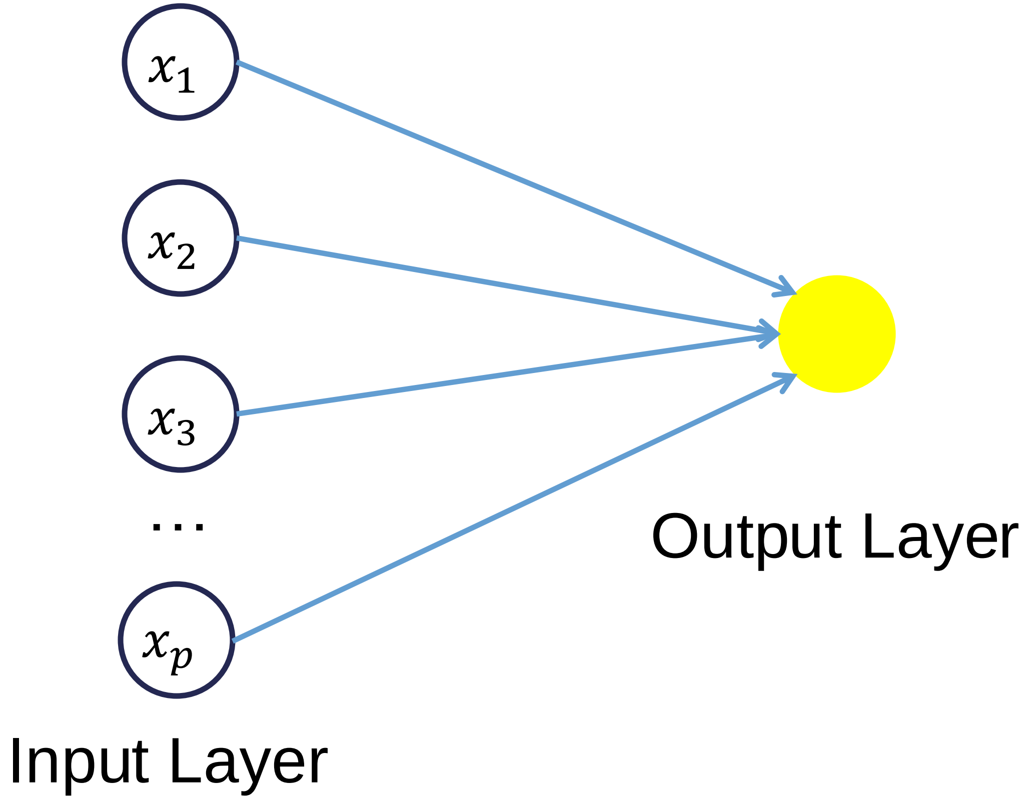
- Concept was well received in 1980’s.
- Didn’t live up to expectations… ☹
- Support Vector Machines (SVM’s) overtook neural networks in the early 2000’s as the popular “black-box” model.

- Revitalized with the growth in image and visual recognition problems.
  - Tons of research
  - “Deep Learning” → Recurrent NN, Convolutional NN, Feedforward NN, etc.

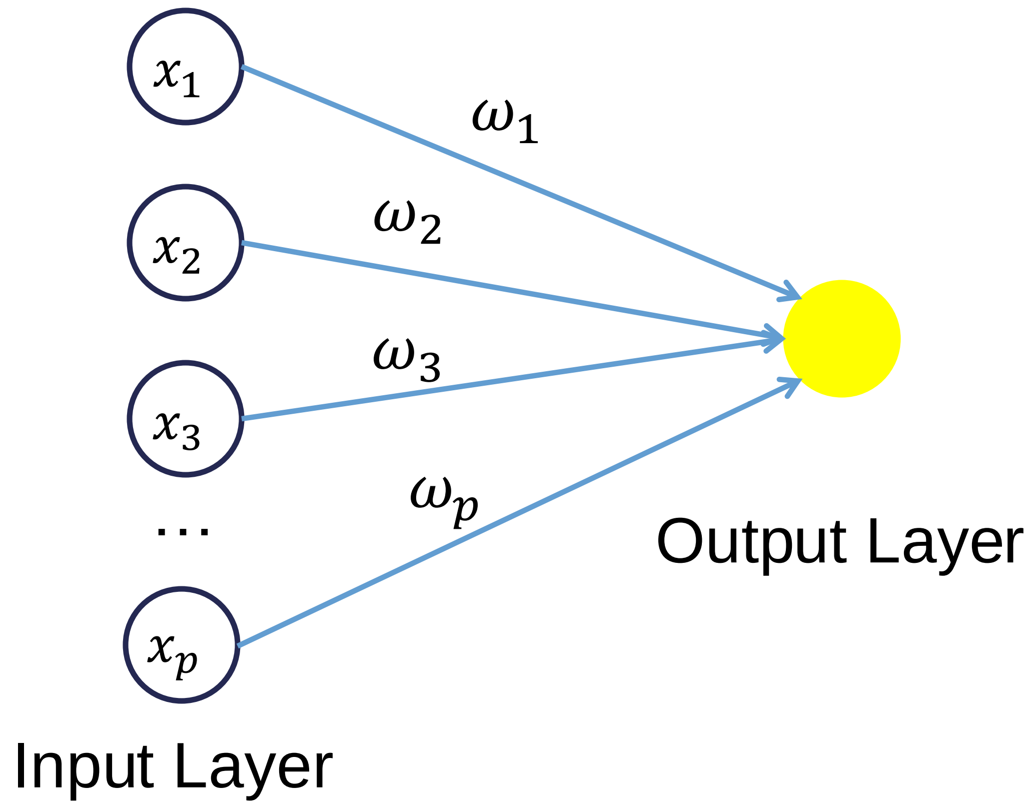
Neural Networks Structure

- They are organized in a network of **neurons** through **layers**.
- The input variables are considered the neurons on the **bottom layer**.
- The output variable is considered the neuron on the **top layer**.
- The layers in between, called **hidden layers**, transform the input variables through non-linear methods to try and best model the output variable.

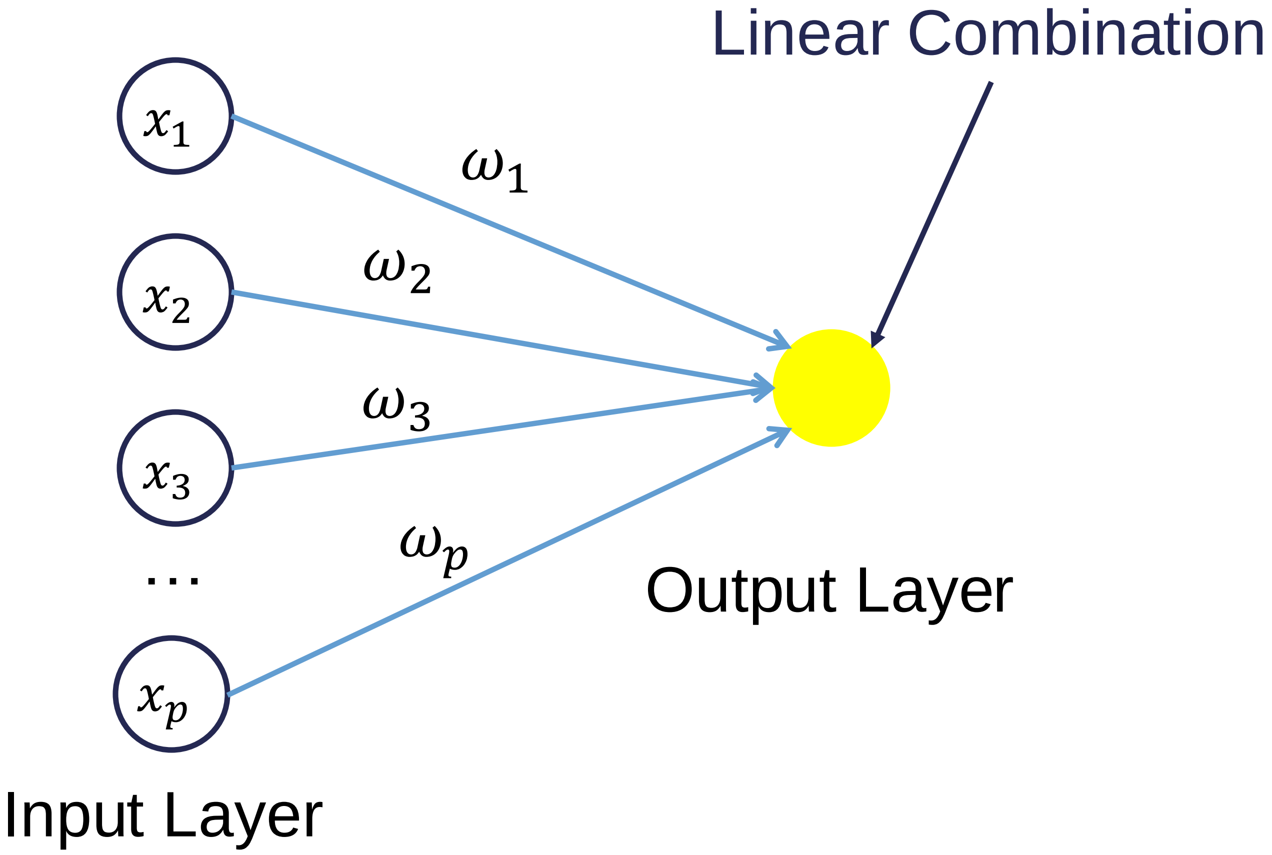
Neural Network Structure



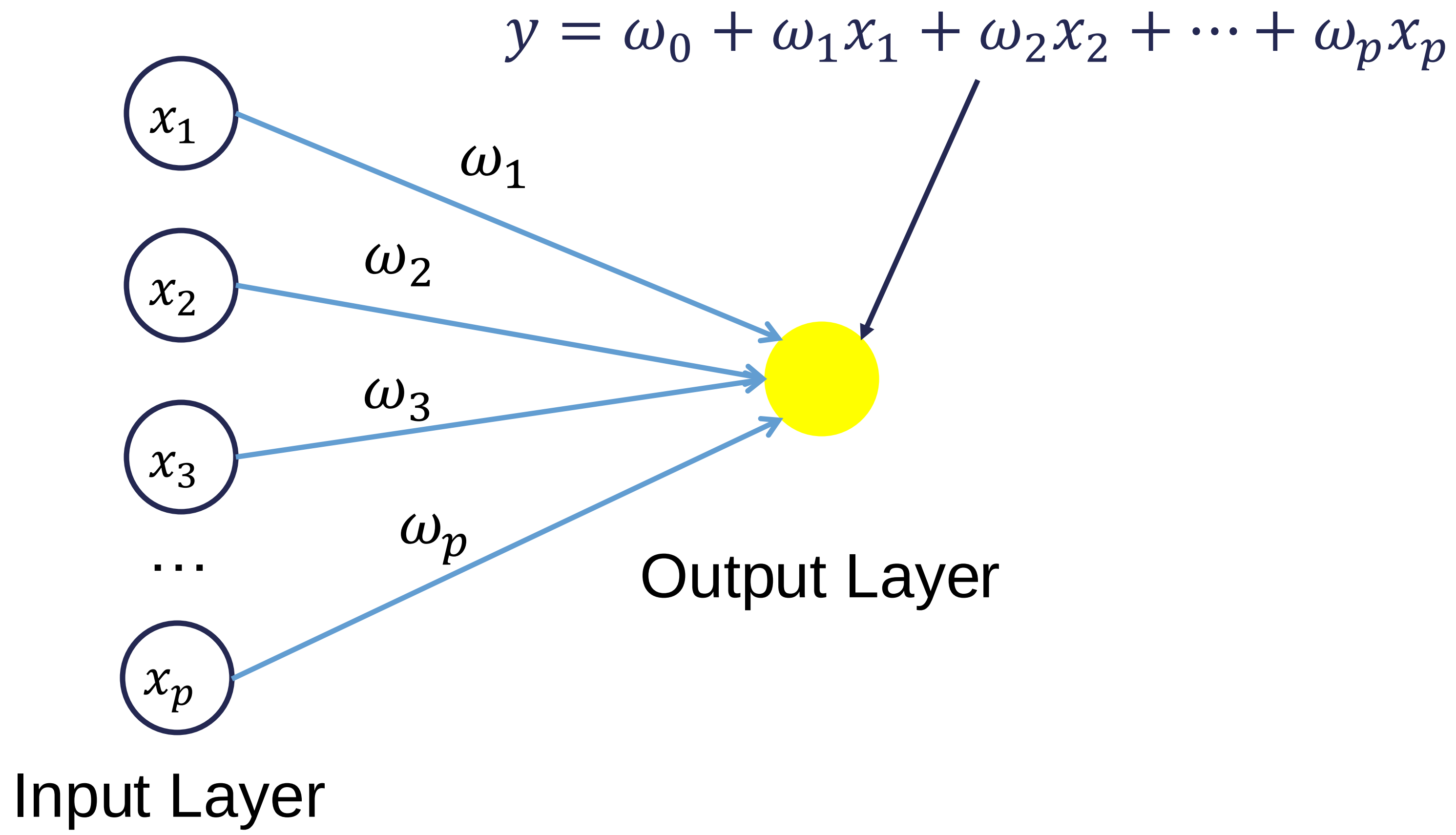
Neural Network Structure



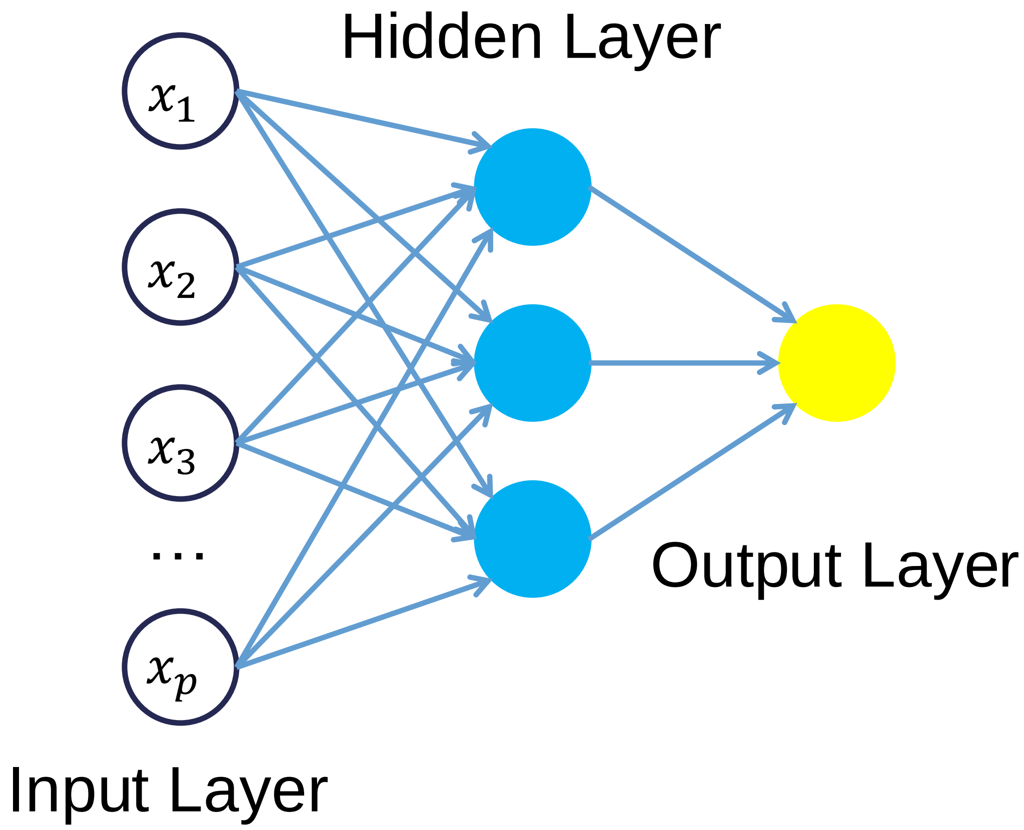
Neural Network Structure



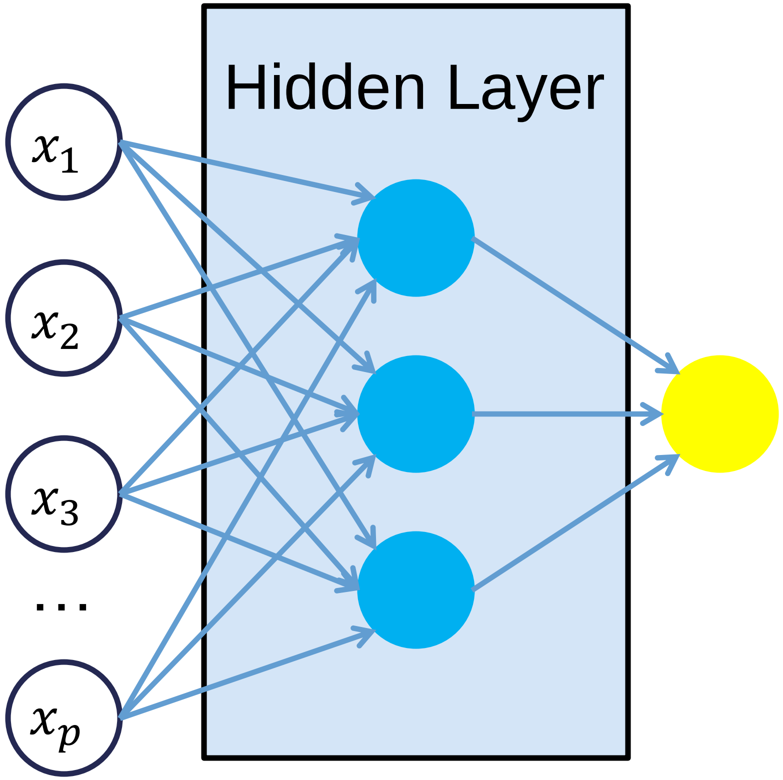
Neural Network Structure



Neural Networks

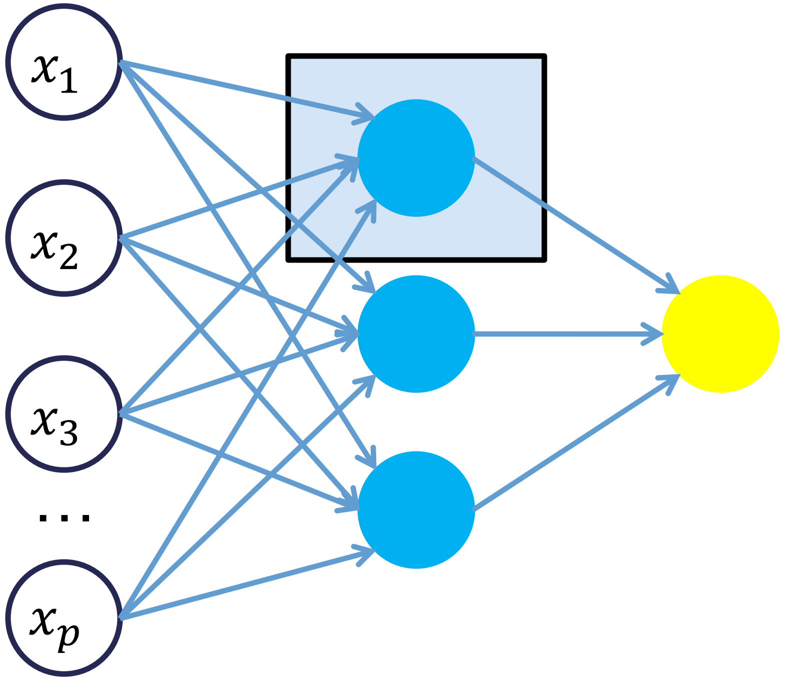


Neural Networks

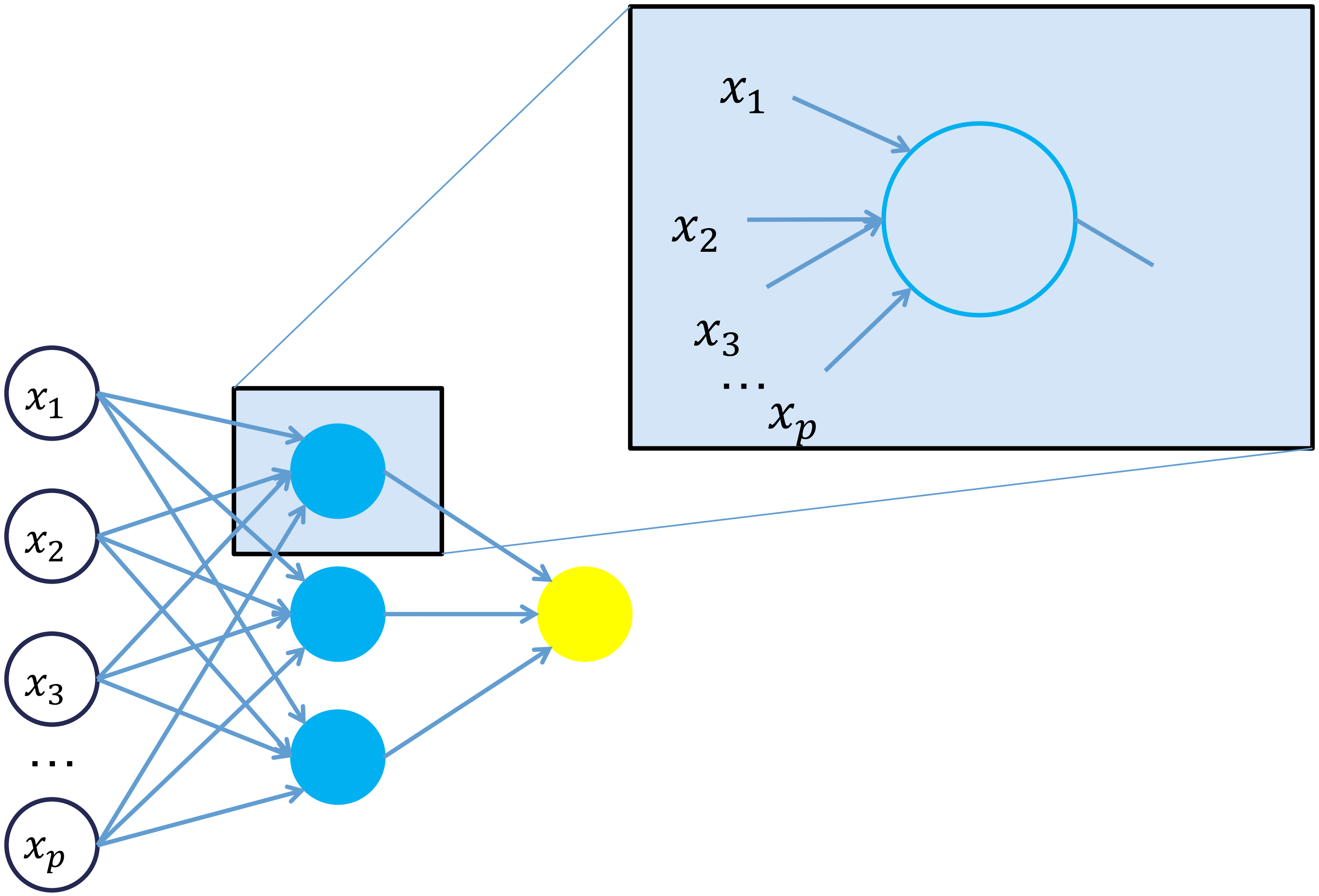


All of the nonlinearities and complication of the variables get added to the model here.

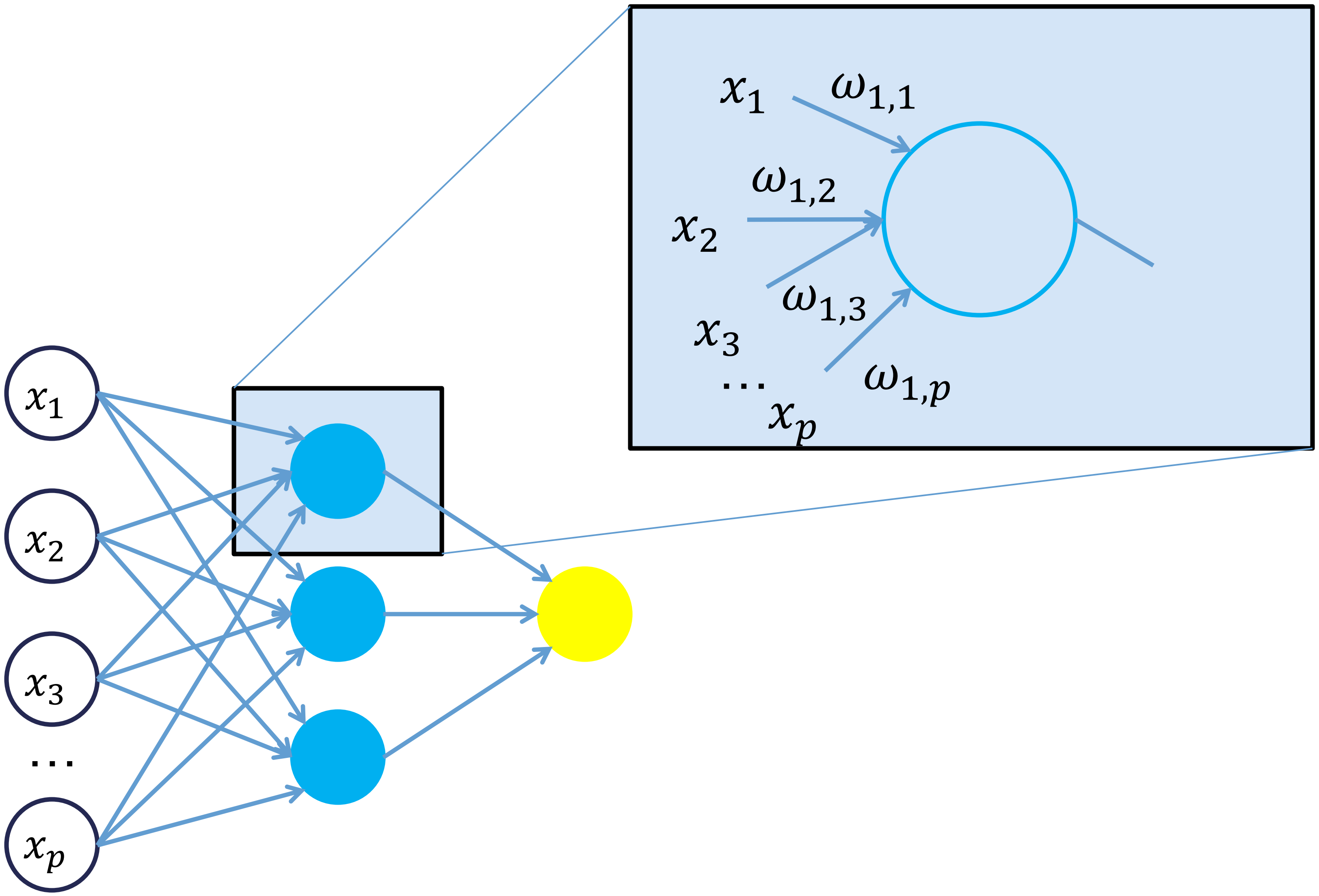
Neural Networks



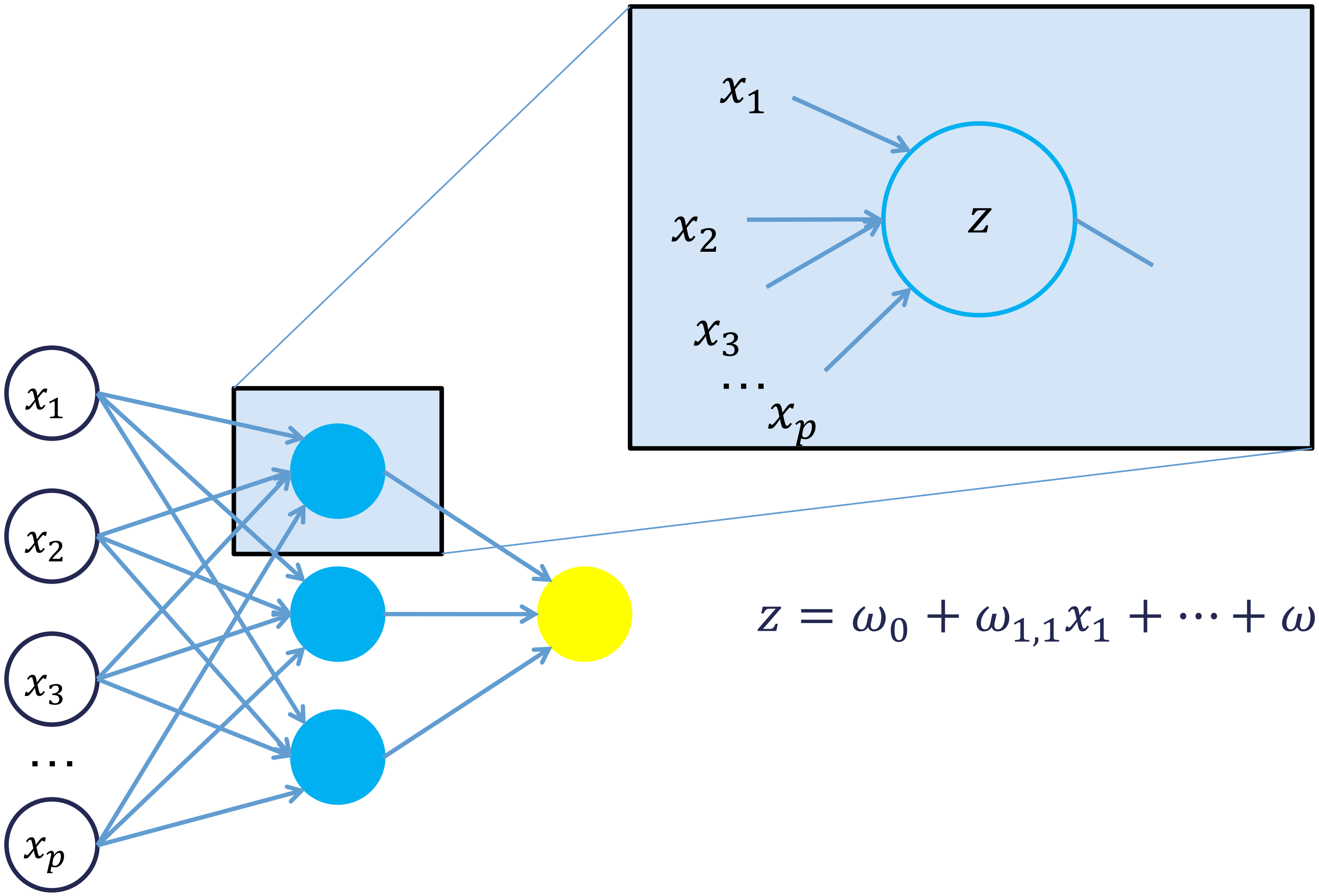
Neural Networks



Neural Networks

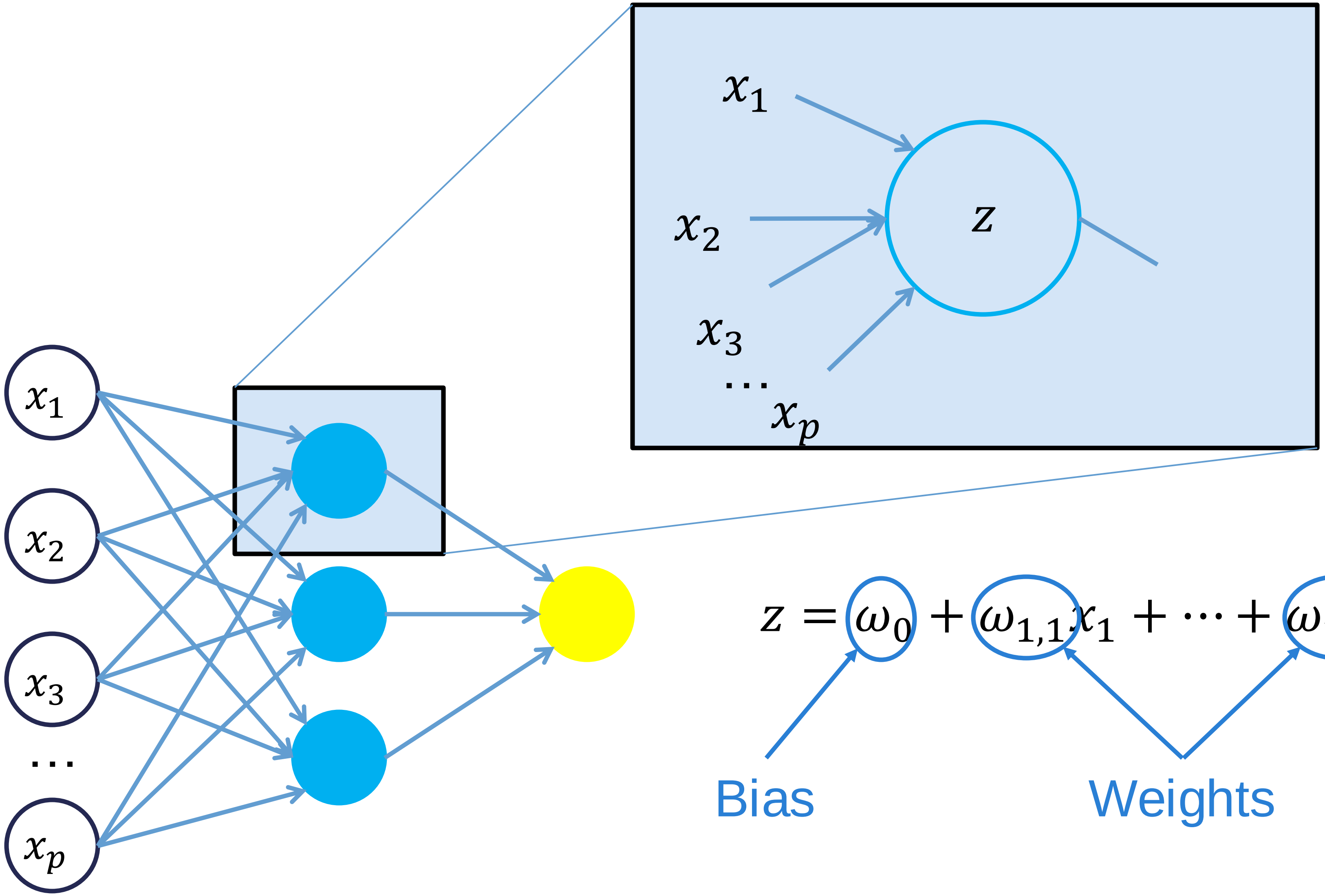


Neural Networks



$$z = \omega_0 + \omega_{1,1} x_1 + \cdots + \omega_{1,p} x_p$$

Neural Networks

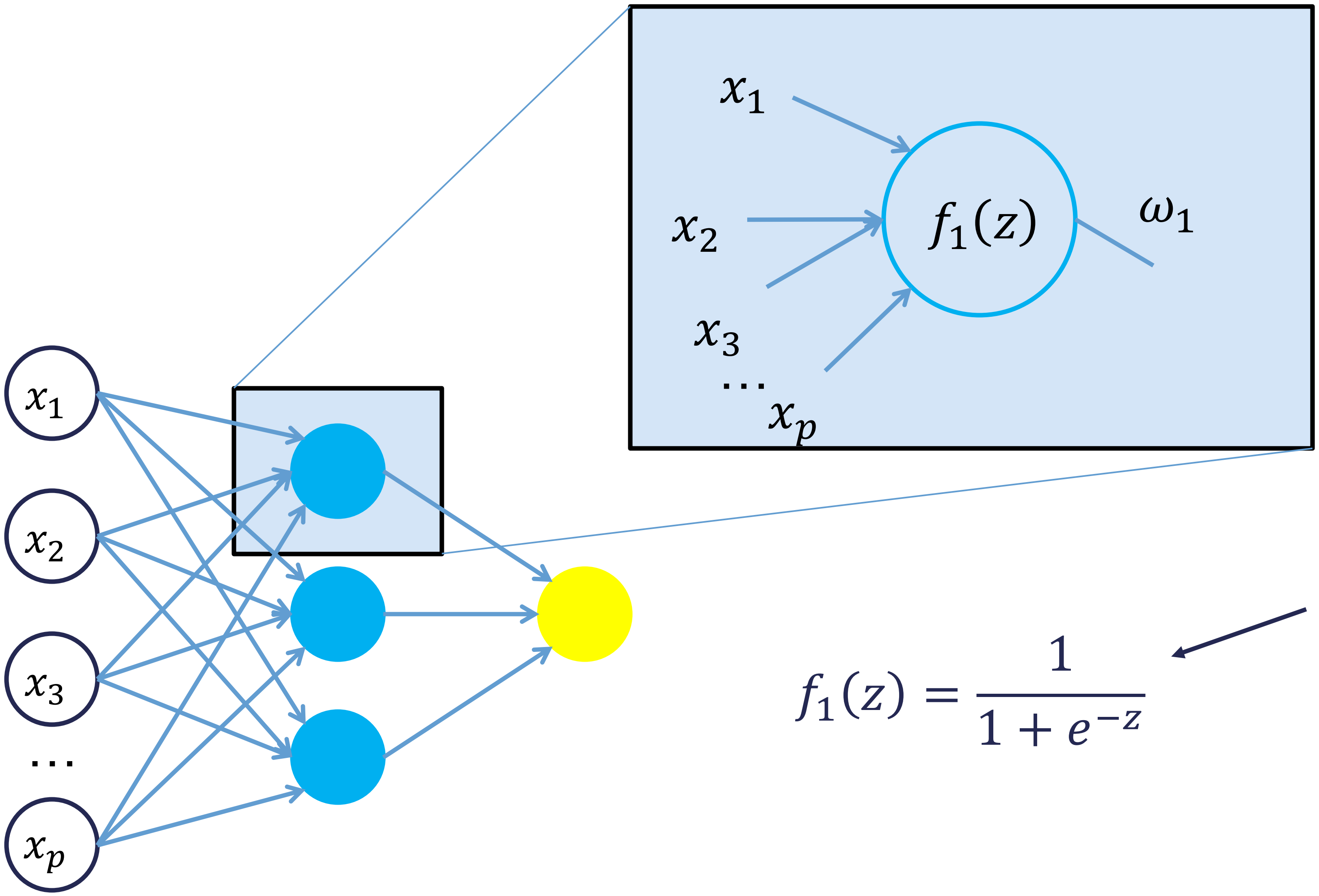


$$z = \omega_0 + \omega_{1,1} x_1 + \cdots + \omega_{1,p} x_p$$

Bias

Weights

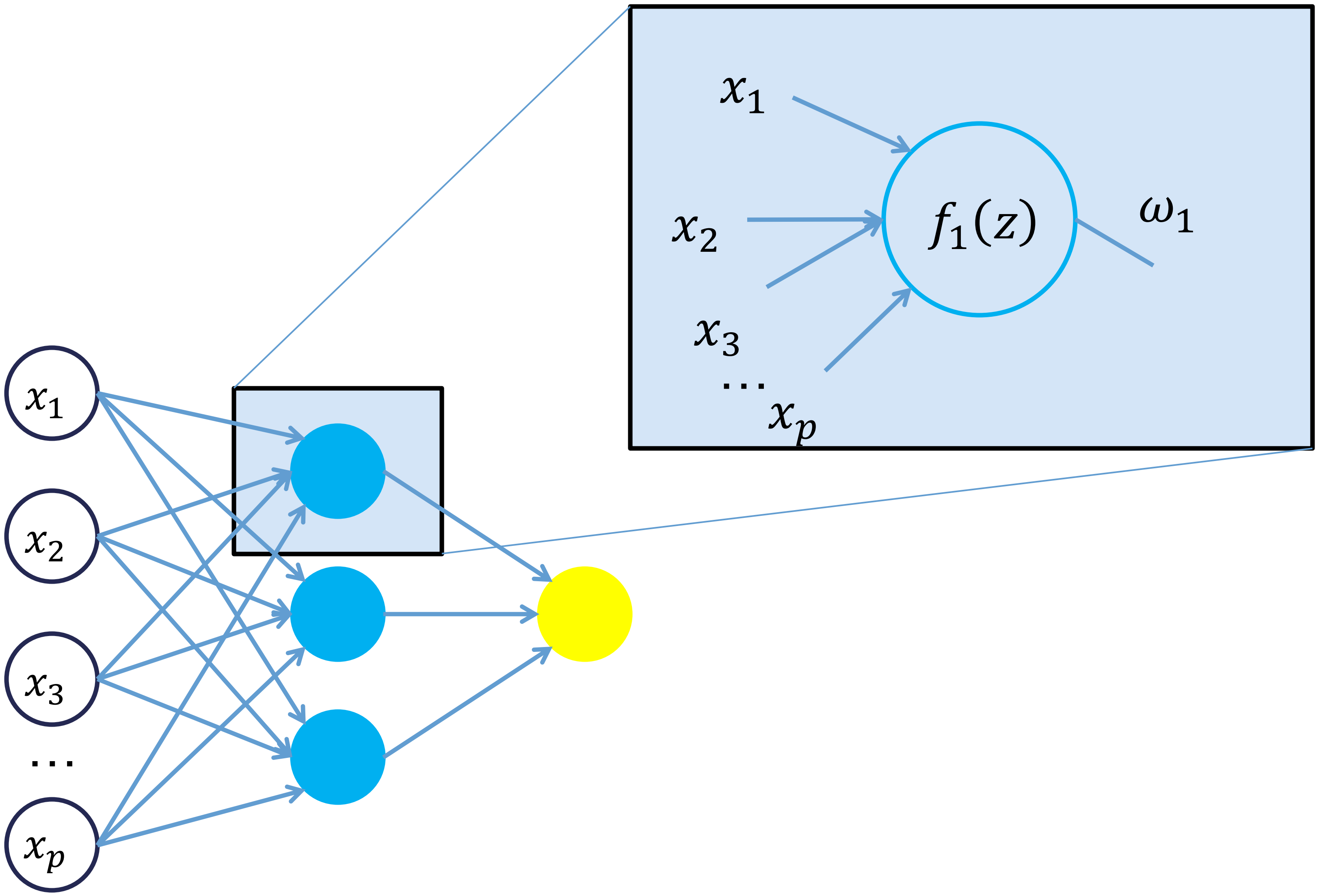
Neural Networks



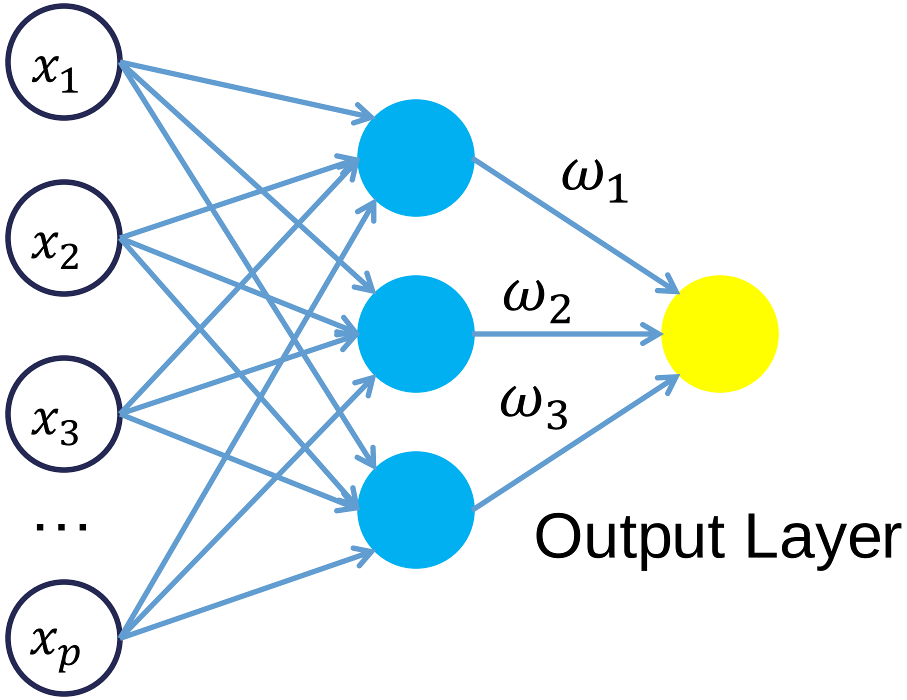
$$f_1(z) = \frac{1}{1 + e^{-z}}$$

Or one of many other functions…

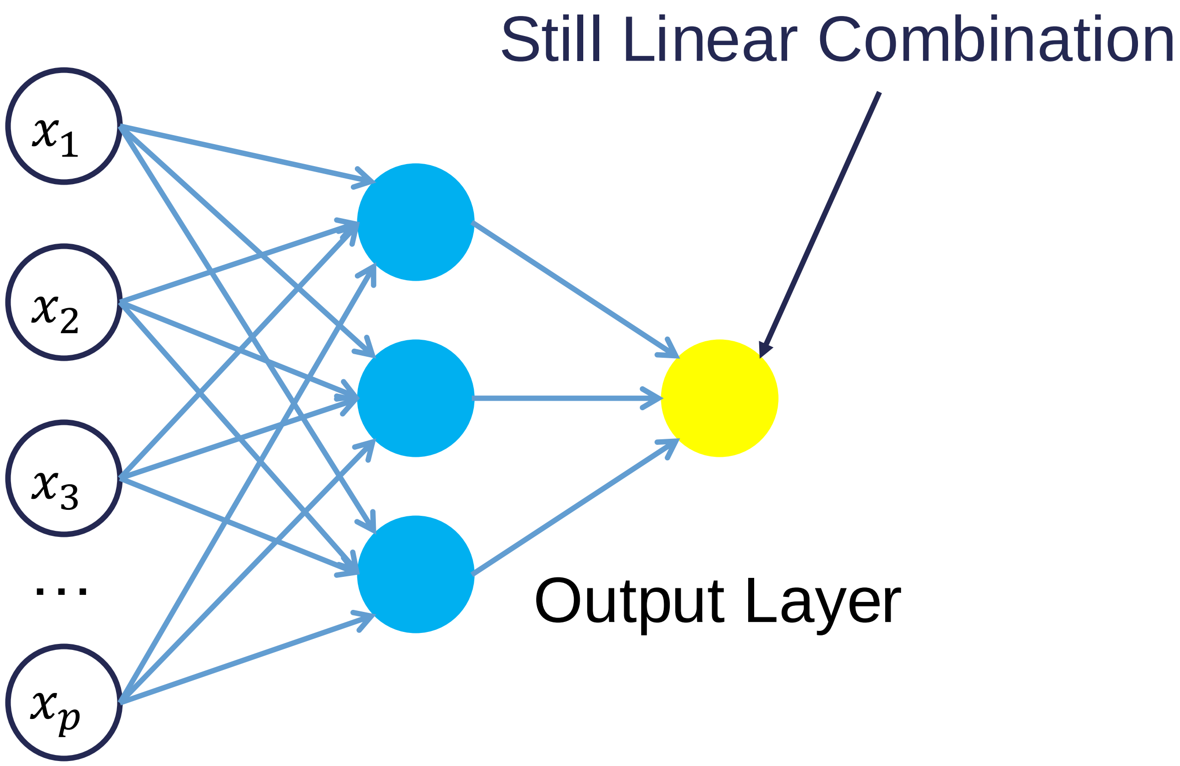
Neural Networks



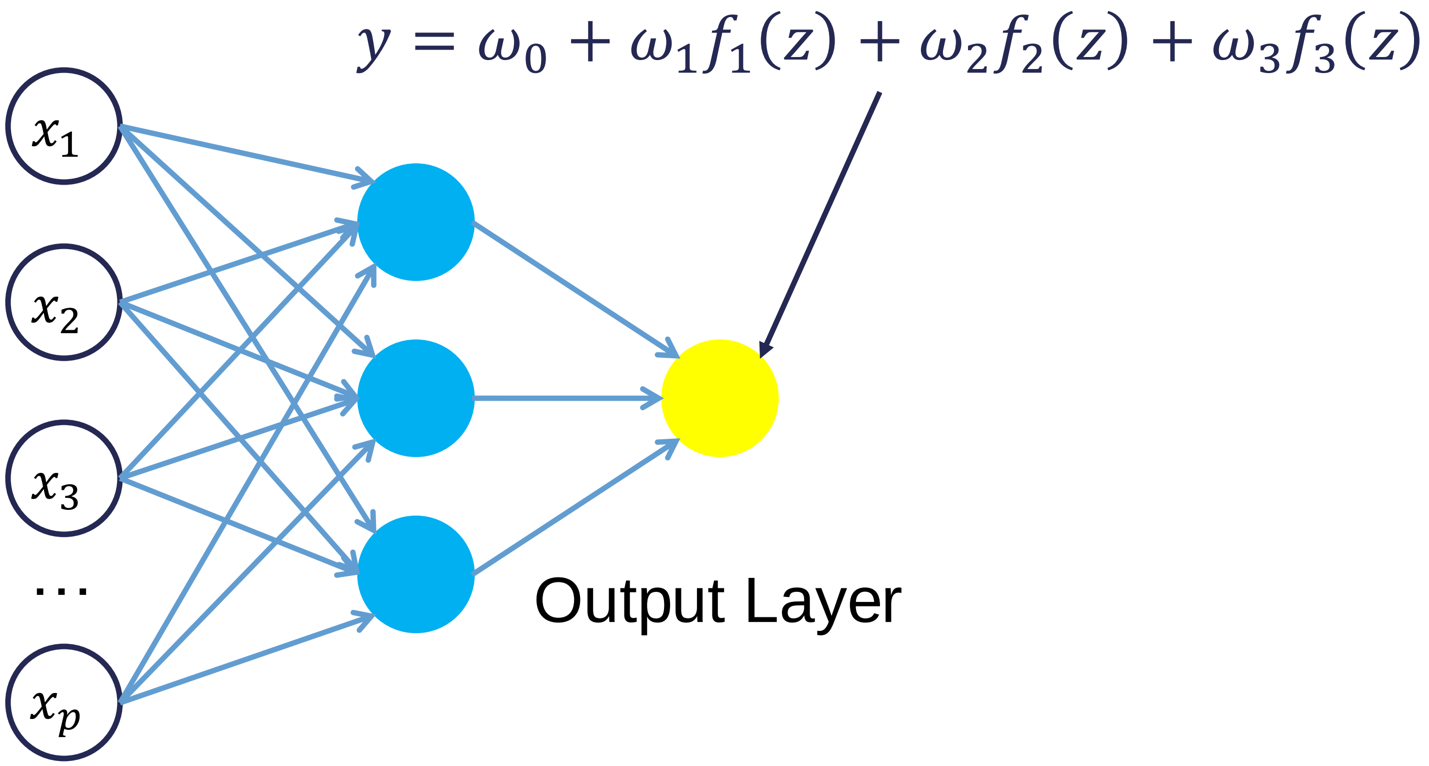
Neural Networks



Neural Networks



Neural Networks



BACKPROPAGATION

Backpropagation Algorithm

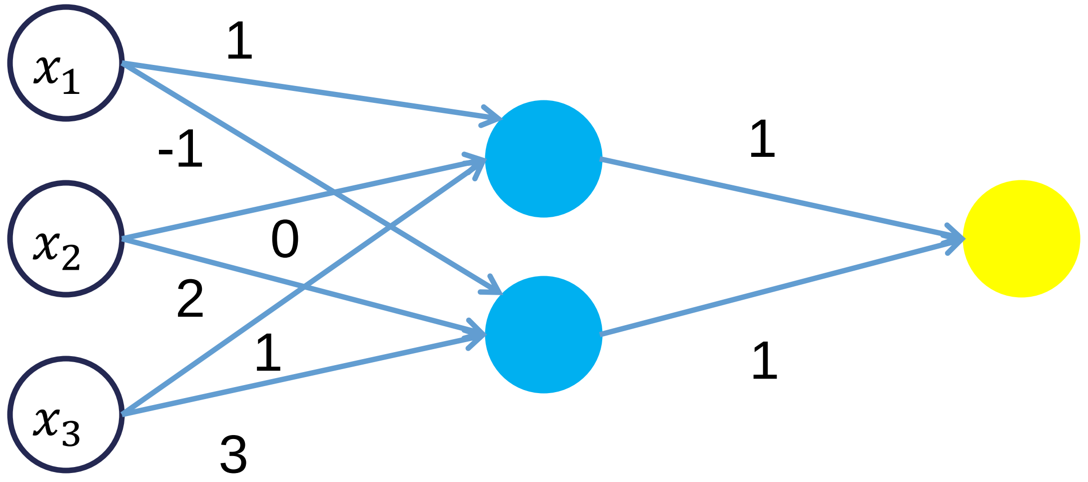
- Forward phase:
  1. Start with some initial weights (often random).
  2. Calculations passed through network.
  3. Predicted value computed.

- Backward phase:
  1. Predicted value compared with actual value (error).
  2. Work backwards through network to adjust weights to make the prediction better.

- **Repeat until some notion of convergence!**

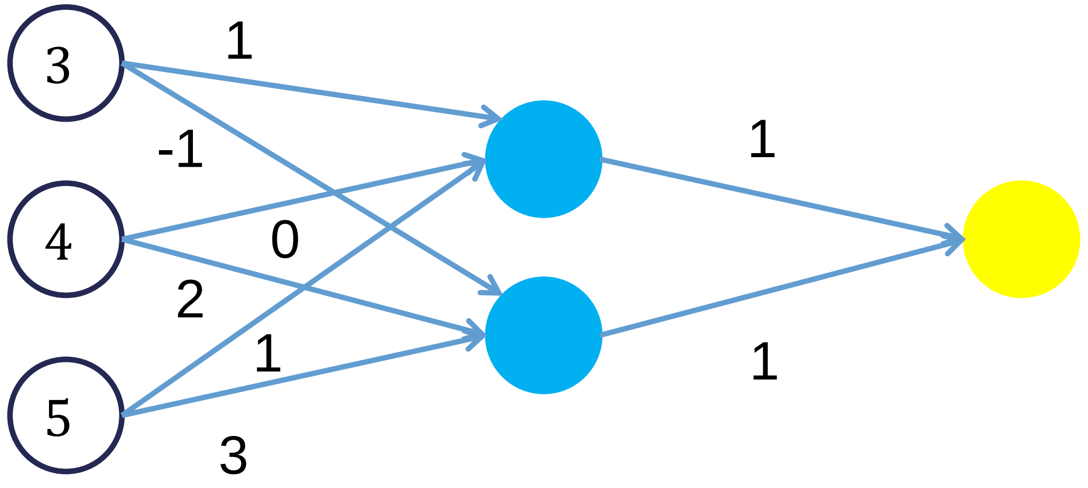
Forward Phase

1. Start with some initial weights (often random).
2. Calculations passed through network.
3. Predicted value computed.



Forward Phase

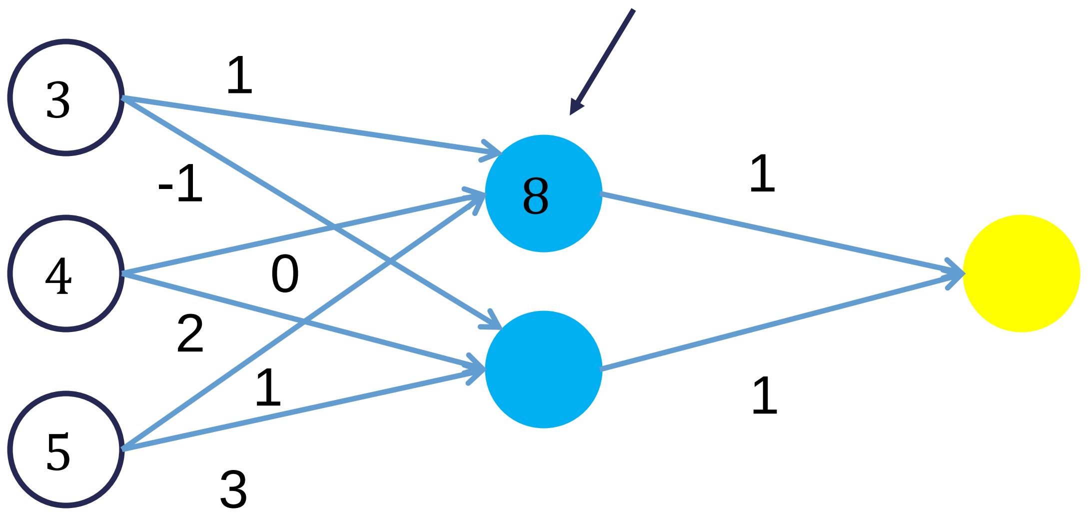
1. Start with some initial weights (often random).
2. Calculations passed through network.
3. Predicted value computed.



Forward Phase

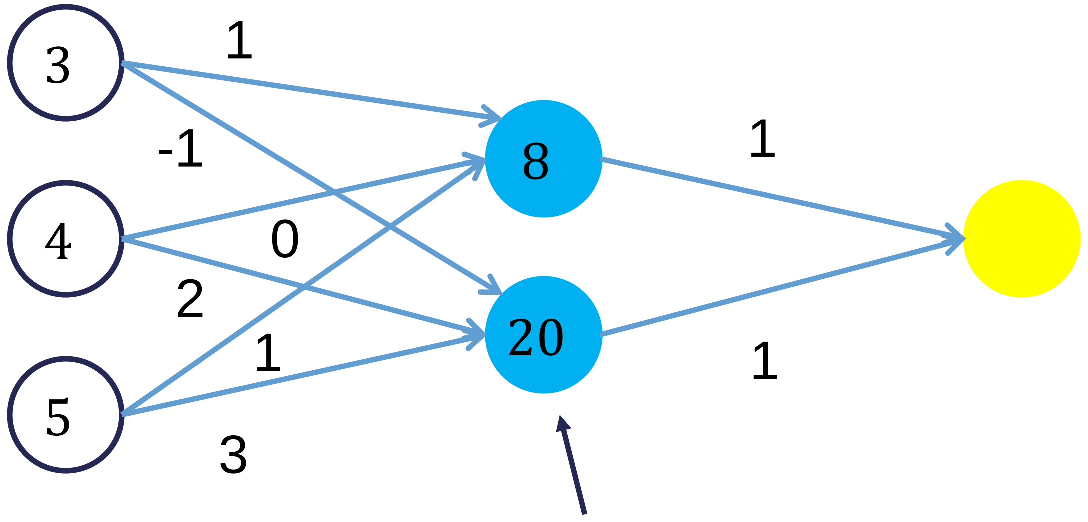
1. Start with some initial weights (often random).
2. Calculations passed through network.
3. Predicted value computed.

$3 * 1 + 4 * 0 + 5 * 1$



Forward Phase

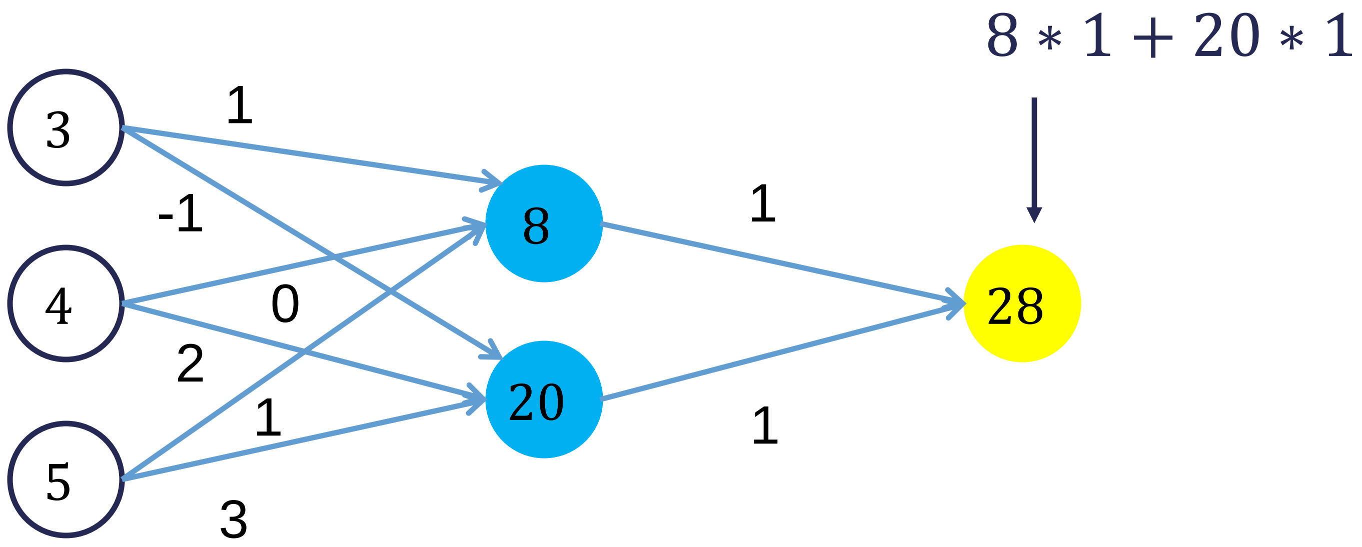
1. Start with some initial weights (often random).
2. Calculations passed through network.
3. Predicted value computed.



$$3 * -1 + 4 * 2 + 5 * 3$$

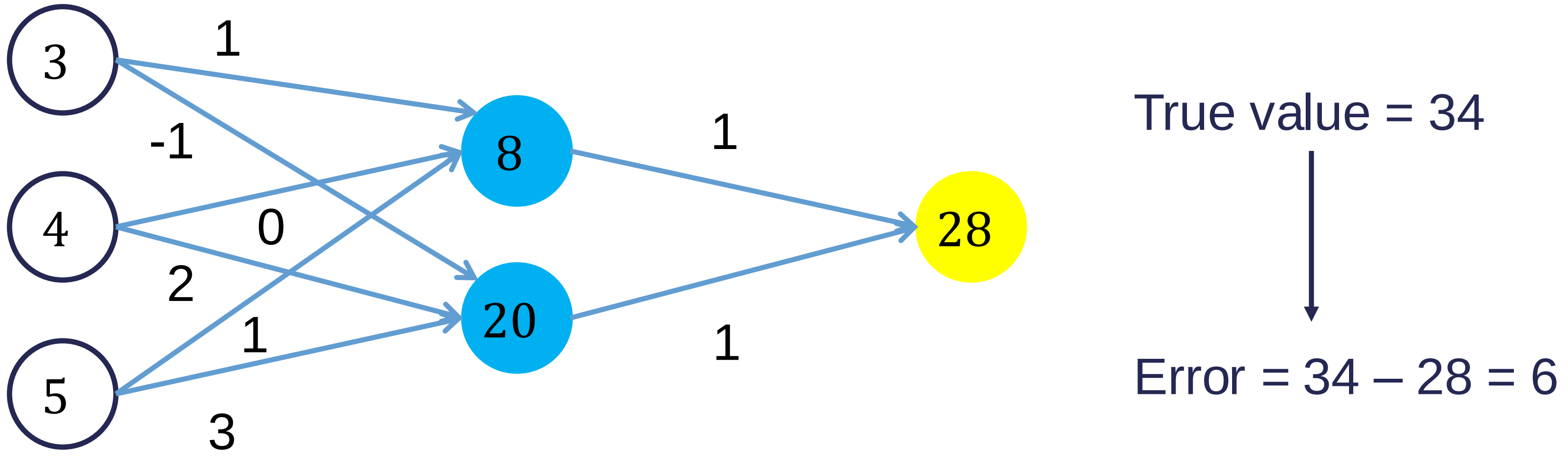
Forward Phase

1. Start with some initial weights (often random).
2. Calculations passed through network.
3. Predicted value computed.



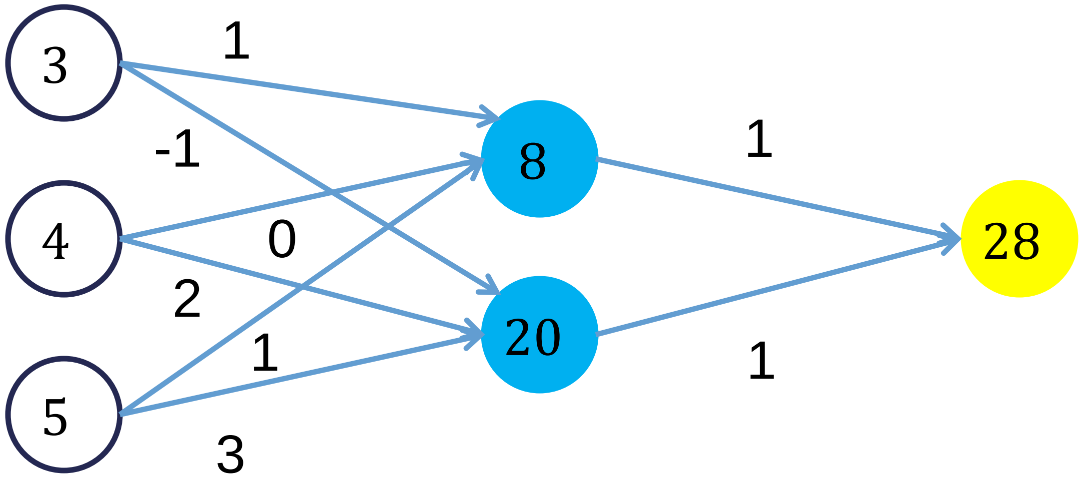
Backward Phase

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better.

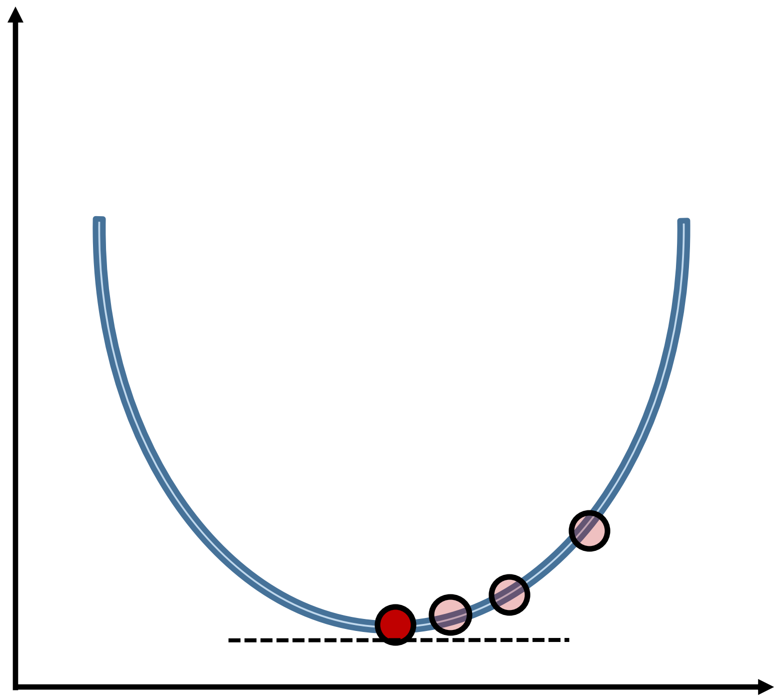


Backward Phase

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better.

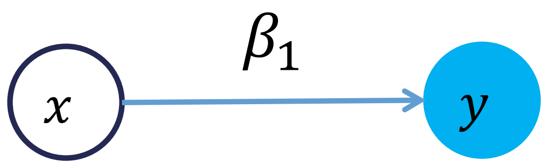


Error = 34 – 28 = 6



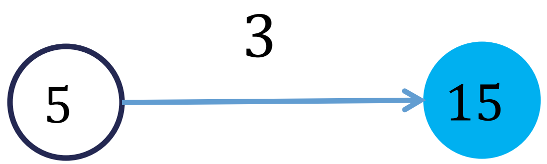
Easy Example

- Let's work through a very, very, very, very watered-down example…
- You know that you have $y = \beta_1 x$.
- You know that $x = 5$ and $y = 20$, but you don't know division…
- Use backpropagation to solve what β_1 is.



Easy Example – Forward

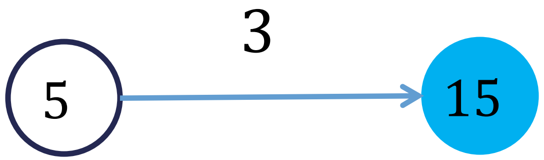
1. Start with some initial weights (often random).
2. Calculations passed through network.
3. Predicted value computed.



True value = 20

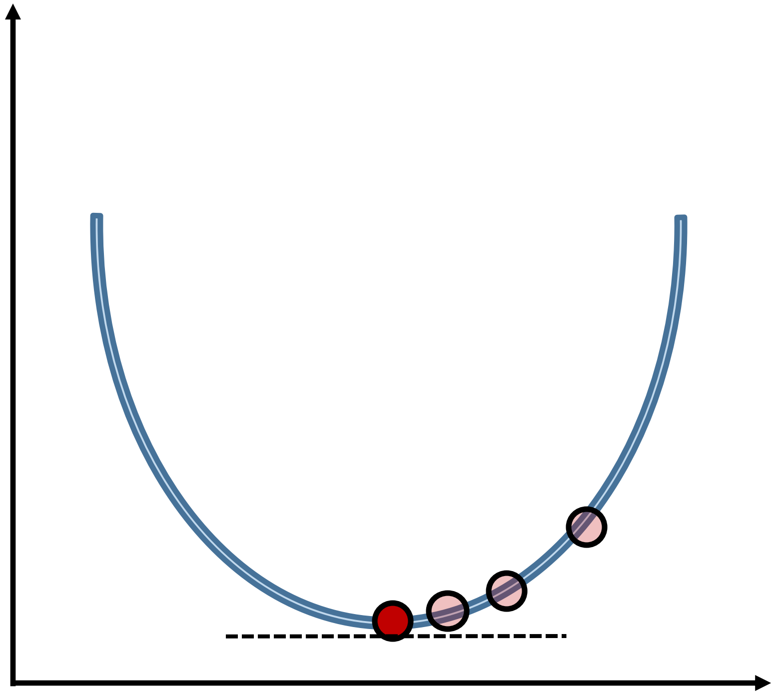
Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!



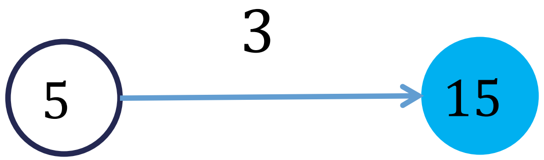
True value = 20

Error = 20 – 15 = 5



Easy Example – Backward

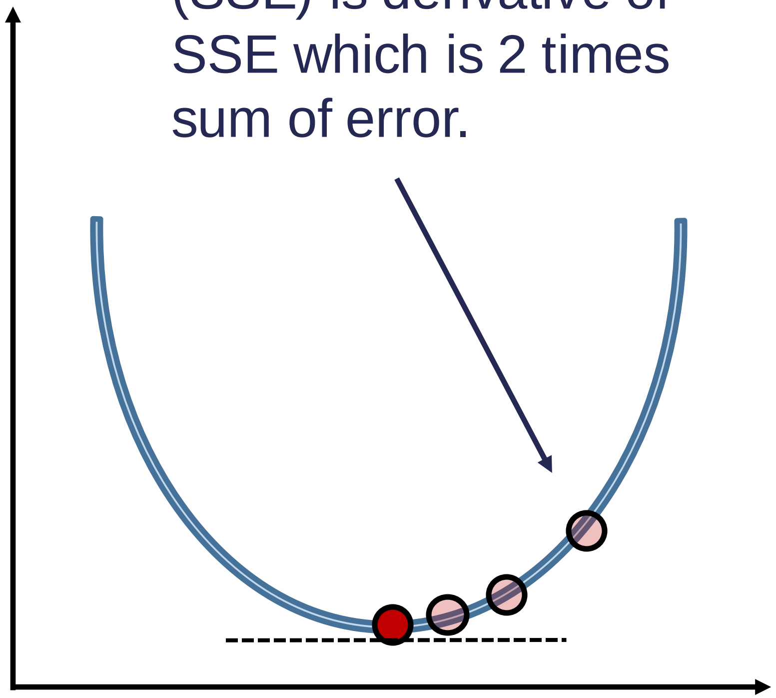
1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!



True value = 20

Error = 20 – 15 = 5

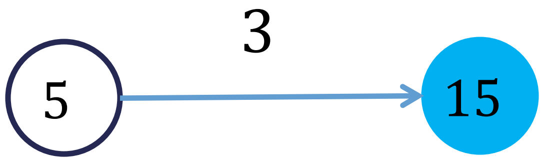
Slope of error curve (SSE) is derivative of SSE which is 2 times sum of error.



Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

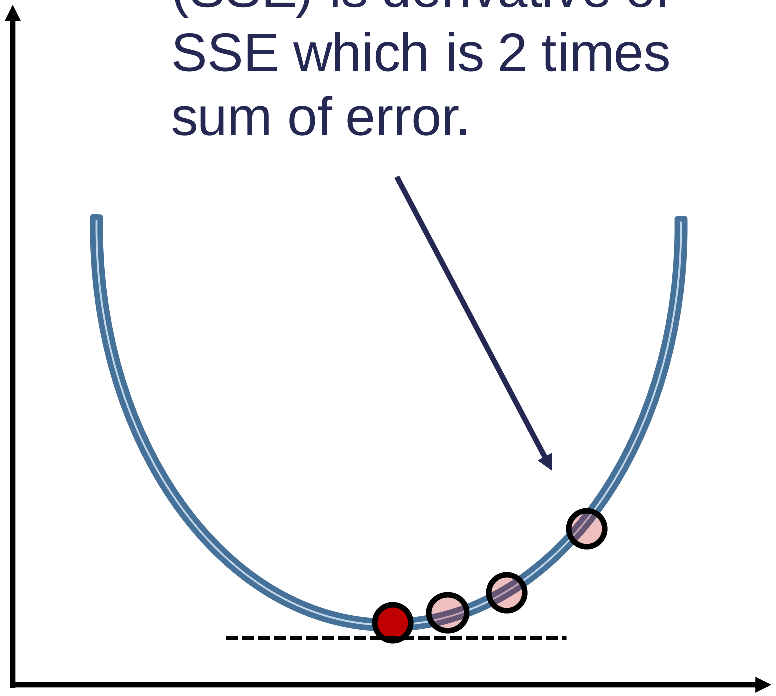
Slope of error curve (SSE) is derivative of SSE which is 2 times sum of error.



True value = 20

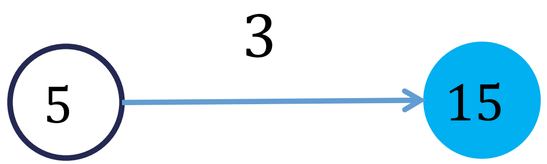
Error = 20 – 15 = 5

2(Error) = 10



Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

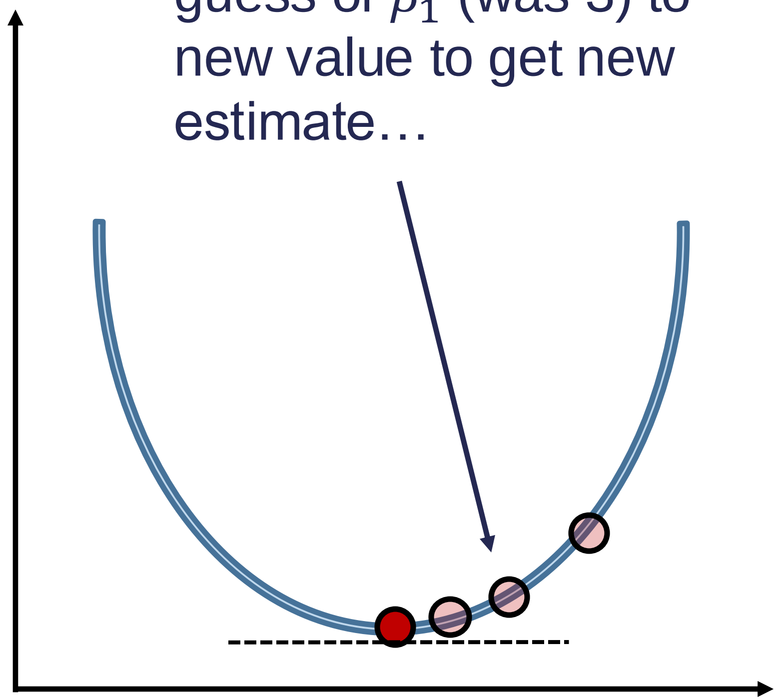


True value = 20

Error = 20 – 15 = 5

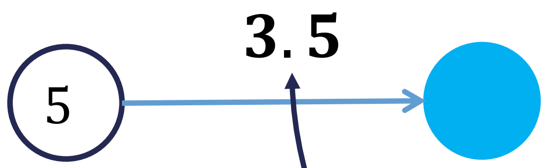
2(Error) = 10

Need to adjust original guess of β_1 (was 3) to new value to get new estimate…



Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

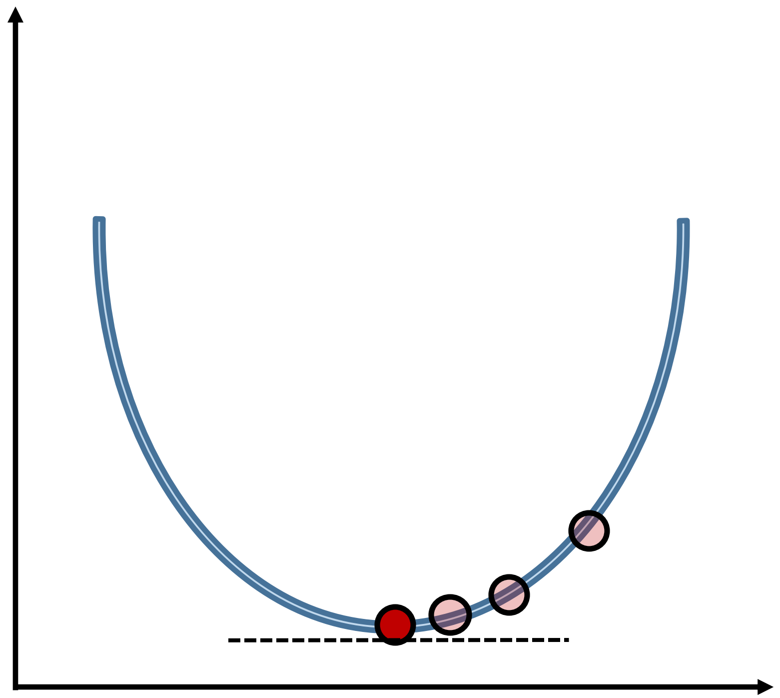


True value = 20

Error = 20 – 15 = 5

2(Error) = 10

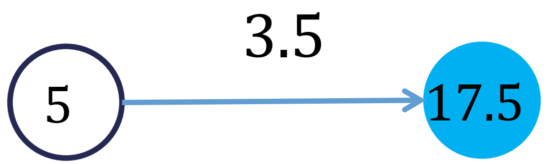
$$\mathbf{3 + 0.05(10) = 3.5}$$



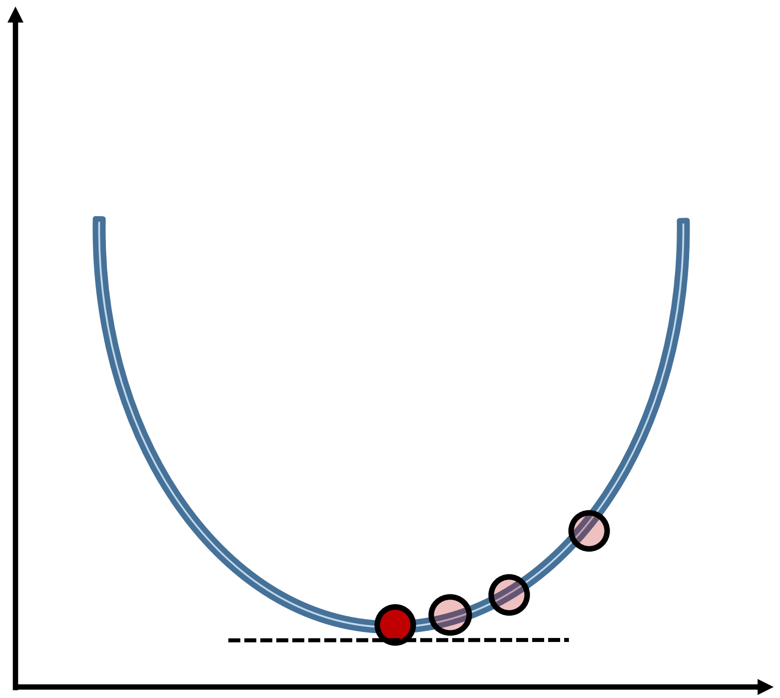
Multiply slope of error curve by **learning rate** and add to original estimate…

Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!



True value = 20

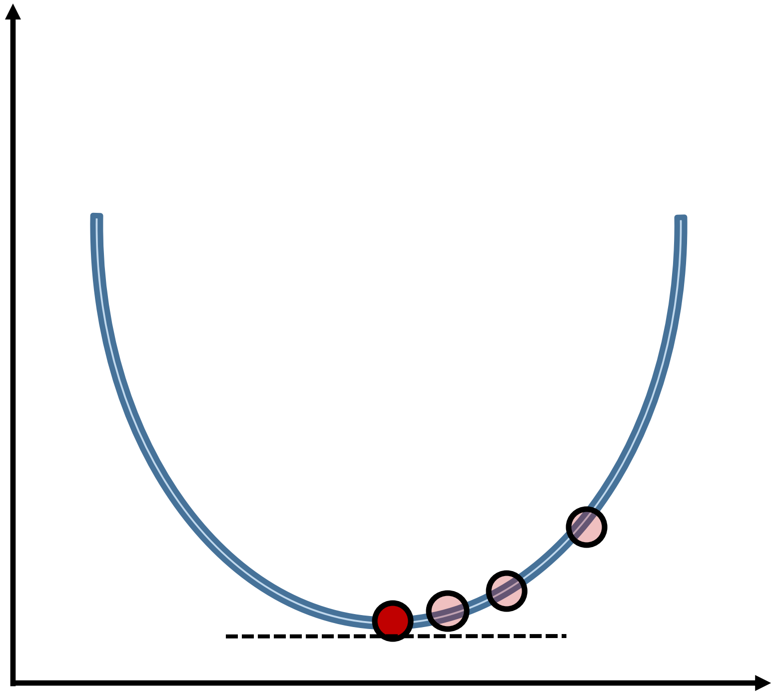


Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

True value = 20

Error = 20 – 17.5 = 2.5



Easy Example – Backward

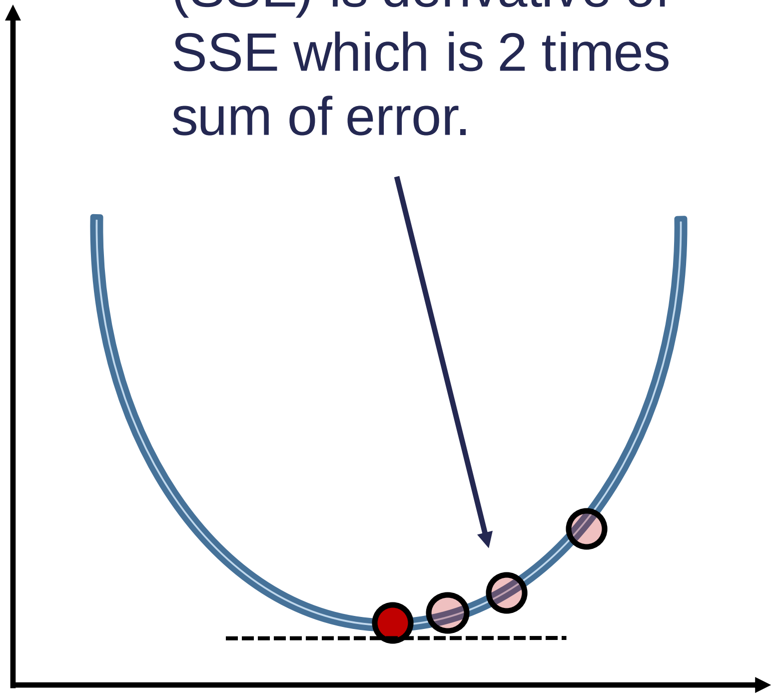
1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

True value = 20

Error = 20 – 17.5 = 2.5

2(Error) = 5

Slope of error curve (SSE) is derivative of SSE which is 2 times sum of error.



Easy Example – Backward

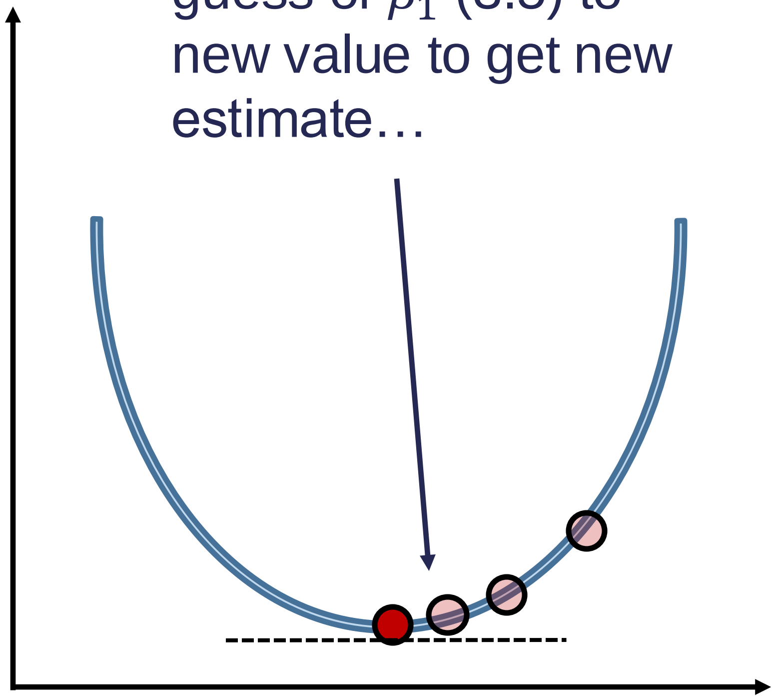
1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

True value = 20

Error = 20 – 17.5 = 2.5

2(Error) = 5

Need to adjust original guess of β_1 (3.5) to new value to get new estimate…



Easy Example – Backward

1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better. BUT HOW?!?!?!?!

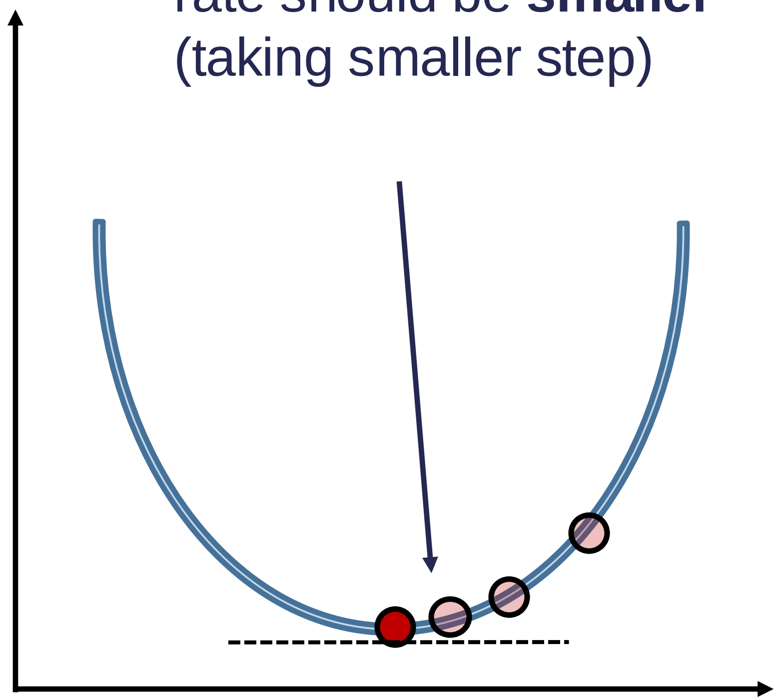
New value of learning rate should be **smaller** (taking smaller step)

True value = 20

Error = 20 – 17.5 = 2.5

2(Error) = 5

$$\mathbf{3.5 + 0.04(5) = 3.7}$$

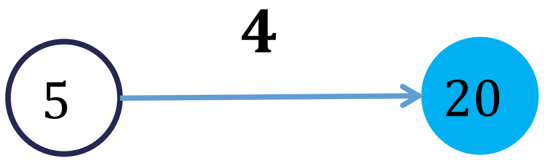


Multiply slope of error curve by **learning rate** and add to original estimate…

Easy Example – Backward

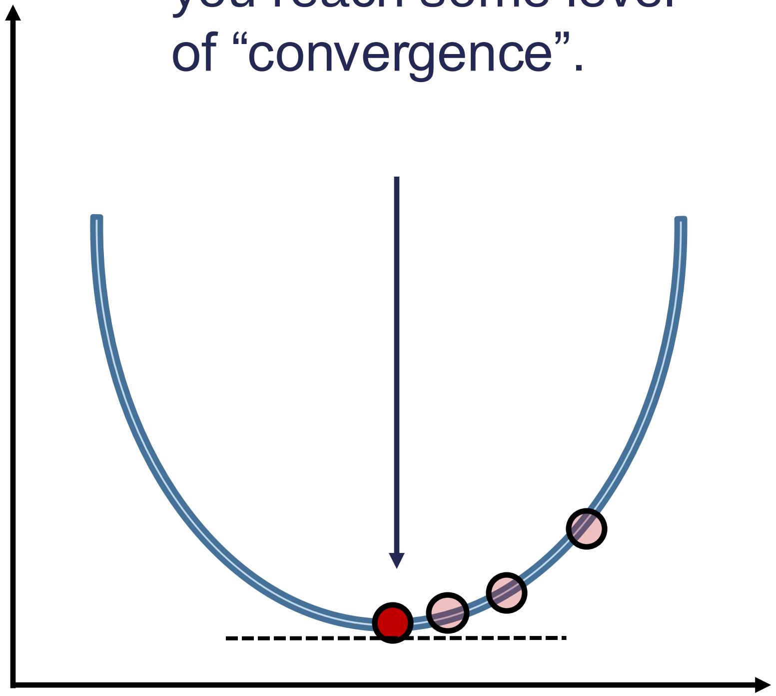
1. Predicted value compared with actual value (error).
2. Work backwards through network to adjust weights to make the prediction better.
3. Repeat, repeat, repeat,…

Continue process until you reach some level of “convergence”.



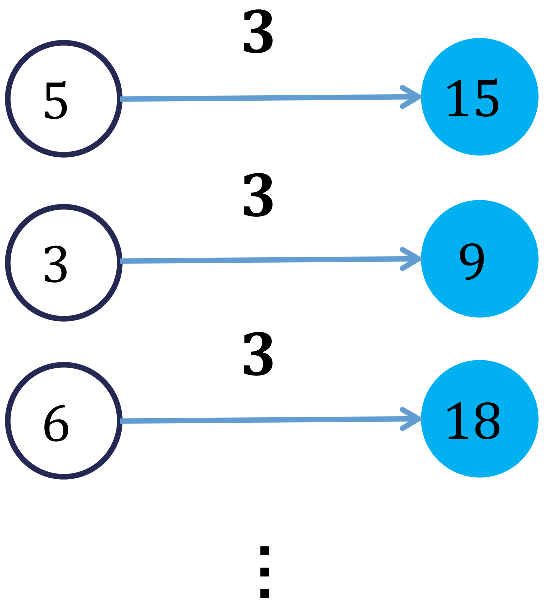
True value = 20

Error = 20 – 20 = 0



Not So Easy In Practice…

- Multiple values of x and y to optimize across, not just one observation…
  - Aggregate errors across all observations (SSE).



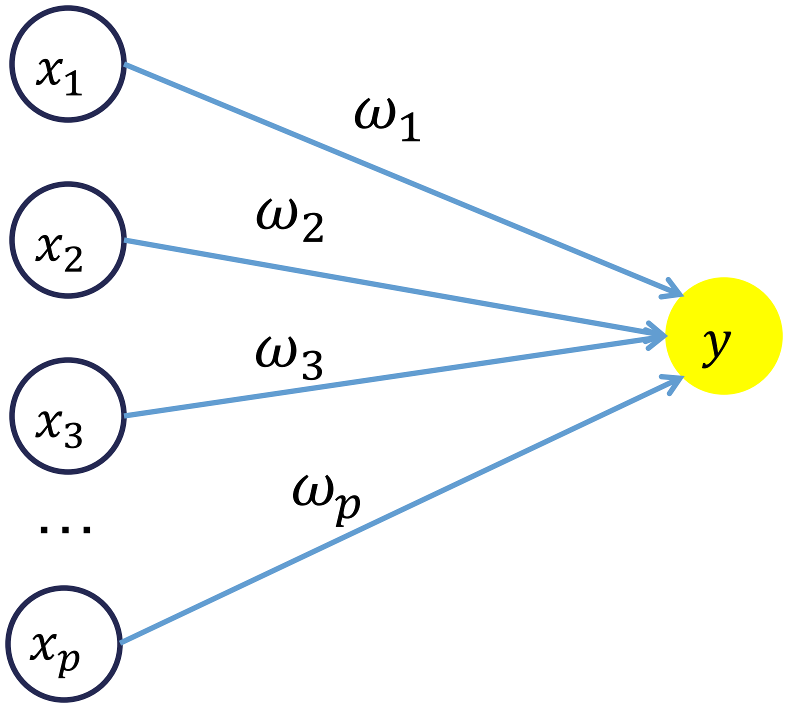
True value = 20, Error = 20 – 15 = 5

True value = 15, Error = 15 – 9 = 6

True value = 25, Error = 25 – 18 = 7

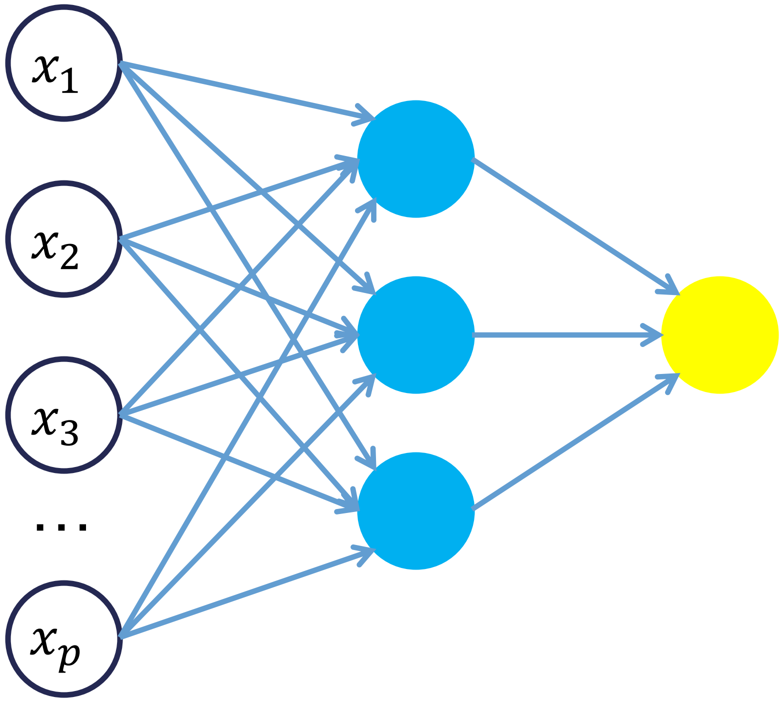
Not So Easy In Practice…

- Multiple variables instead of just one.
  - More than one weight you must now optimize at once.



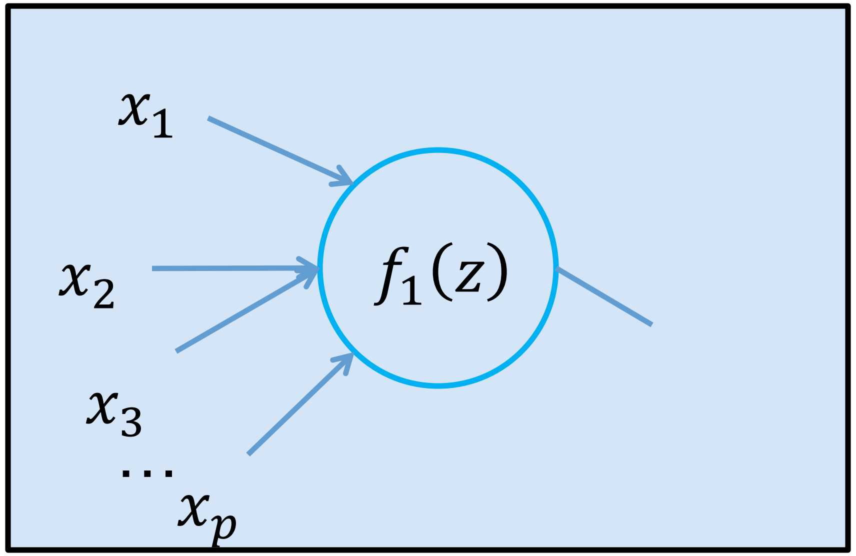
Not So Easy In Practice…

- Hidden layers add complexity to the model since one change in slope impacts many different nodes instead of one.



Not So Easy In Practice…

- Nonlinearities inside the hidden nodes make the calculus harder since the predictor variables are NOT just related to the slope of the SSE function, but the **combined** (think calculus chain rule) slope of the error loss function with the nonlinear (logistic function) relationship…



$$f_1(z) = \frac{1}{1 + e^{-z}}$$

Backpropagation Algorithm

- Forward phase:
  1. Start with some initial weights (often random).
  2. Calculations passed through network.
  3. Predicted value computed.

- Backward phase:
  1. Predicted value compared with actual value (error).
  2. Work backwards through network to adjust weights to make the prediction better.

- **Repeat until some notion of convergence! MATH IS HARD!!!**

?

FITTING NEURAL NETWORK IN R

Ames Data

```
set.seed(4321)

training <- ames %>% sample_frac(0.7)
testing <- anti_join(ames, training, by = 'id')

training <- training %>%
  select(Sale_Price,
         Bedroom_AbvGr,
         Year_Built,
         Mo_Sold,
         Lot_Area,
         Street,
         Central_Air,
         First_Flr_SF,
         Second_Flr_SF,
         Full_Bath,
         Half_Bath,
         Fireplaces,
         Garage_Area,
         Gr_Liv_Area,
         TotRms_AbvGrd)
```

Standardization

- Neural networks **work best when input data are scaled** (but NOT required) to a narrow range around 0.
- For bell-shaped data, statistical z-scores standardization work:

$$z = \frac{x - \bar{x}}{s_x}$$

- For severely asymmetric data, midrange standardization works better:

$$\frac{x - \mathrm{midrange}(x)}{0.5 \times \mathrm{range}(x)} = \frac{x - \dfrac{(\max(x) + \min(x))}{2}}{0.5 \times (\max(x) - \min(x))}$$

Standardization

```
training <- training %>%
  mutate(s_SalePrice = scale(Sale_Price),
         s_Bedroom_AbvGr = scale(Bedroom_AbvGr),
         s_Year_Built = scale(Year_Built),
         s_Mo_Sold = scale(Mo_Sold),
         s_Lot_Area = scale(Lot_Area),
         s_First_Flr_SF = scale(First_Flr_SF),
         s_Second_Flr_SF = scale(Second_Flr_SF),
         s_Garage_Area = scale(Garage_Area),
         s_Gr_Liv_Area = scale(Gr_Liv_Area),
         s_TotRms_AbvGrd = scale(TotRms_AbvGrd))


training$Full_Bath <- as.factor(training$Full_Bath)
training$Half_Bath <- as.factor(training$Half_Bath)
training$Fireplaces <- as.factor(training$Fireplaces)
```

Neural Network

```
set.seed(12345)
nn.ames <- nnet(Sale_Price ~
                  s_Bedroom_AbvGr +
                  s_Year_Built +
                  s_Mo_Sold +
                  s_Lot_Area +
                  s_First_Flr_SF +
                  s_Second_Flr_SF +
                  s_Garage_Area +
                  s_Gr_Liv_Area +
                  s_TotRms_AbvGrd +
                  Street +
                  Central_Air +
                  Full_Bath +
                  Half_Bath +
                  Fireplaces
                  , data = training, size = 5, linout = TRUE)
```

Optimize Number of Hidden Nodes and Decay

```
tune_grid <- expand.grid(
  .size = c(3, 4, 5, 6, 7),
  .decay = c(0, 0.5, 1)
)

set.seed(12345)
nn.ames.caret <- train(Sale_Price ~
                         s_Bedroom_AbvGr + s_Year_Built + s_Mo_Sold + s_Lot_Area + s_First_Flr_SF +
                         s_Second_Flr_SF + s_Garage_Area + s_Gr_Liv_Area + s_TotRms_AbvGrd + Street +
                         Central_Air + Full_Bath + Half_Bath + Fireplaces
                       , data = training,
                        method = "nnet",
                        tuneGrid = tune_grid,
                        trControl = trainControl(method = 'cv', number = 10),
                        trace = FALSE, linout = TRUE)

nn.ames.caret$bestTune

##    size decay
## 12    6     1
```

Optimize Number of Hidden Nodes and Decay

Size = Number of hidden nodes
Decay = Regularization parameter to prevent overfitting

```
tune_grid <- expand.grid(
  .size = c(3, 4, 5, 6, 7),
  .decay = c(0, 0.5, 1)
)

set.seed(12345)
nn.ames.caret <- train(Sale_Price ~
                         s_Bedroom_AbvGr + s_Year_Built + s_Mo_Sold + s_Lot_Area + s_First_Flr_SF +
                         s_Second_Flr_SF + s_Garage_Area + s_Gr_Liv_Area + s_TotRms_AbvGrd + Street +
                         Central_Air + Full_Bath + Half_Bath + Fireplaces
                       , data = training,
                       method = "nnet",
                       tuneGrid = tune_grid,
                       trControl = trainControl(method = 'cv', number = 10),
                       trace = FALSE, linout = TRUE)

nn.ames.caret$bestTune

##    size decay
## 12    6     1
```

Optimize Number of Hidden Nodes and Decay

```
tune_grid <- expand.grid(
  .size = c(3, 4, 5, 6, 7),
  .decay = c(0, 0.5, 1)
)

set.seed(12345)
nn.ames.caret <- train(Sale_Price ~
                         s_Bedroom_AbvGr + s_Year_Built + s_Mo_Sold + s_Lot_Area + s_First_Flr_SF +
                         s_Second_Flr_SF + s_Garage_Area + s_Gr_Liv_Area + s_TotRms_AbvGrd + Street +
                         Central_Air + Full_Bath + Half_Bath + Fireplaces
                       , data = training,
                       method = "nnet",
                       tuneGrid = tune_grid,
                       trControl = trainControl(method = 'cv', number = 10),
                       trace = FALSE, linout = TRUE)

nn.ames.caret$bestTune

##    size decay
## 12    6     1
```

Size = 6 hidden nodes
Decay = 1 for penalty

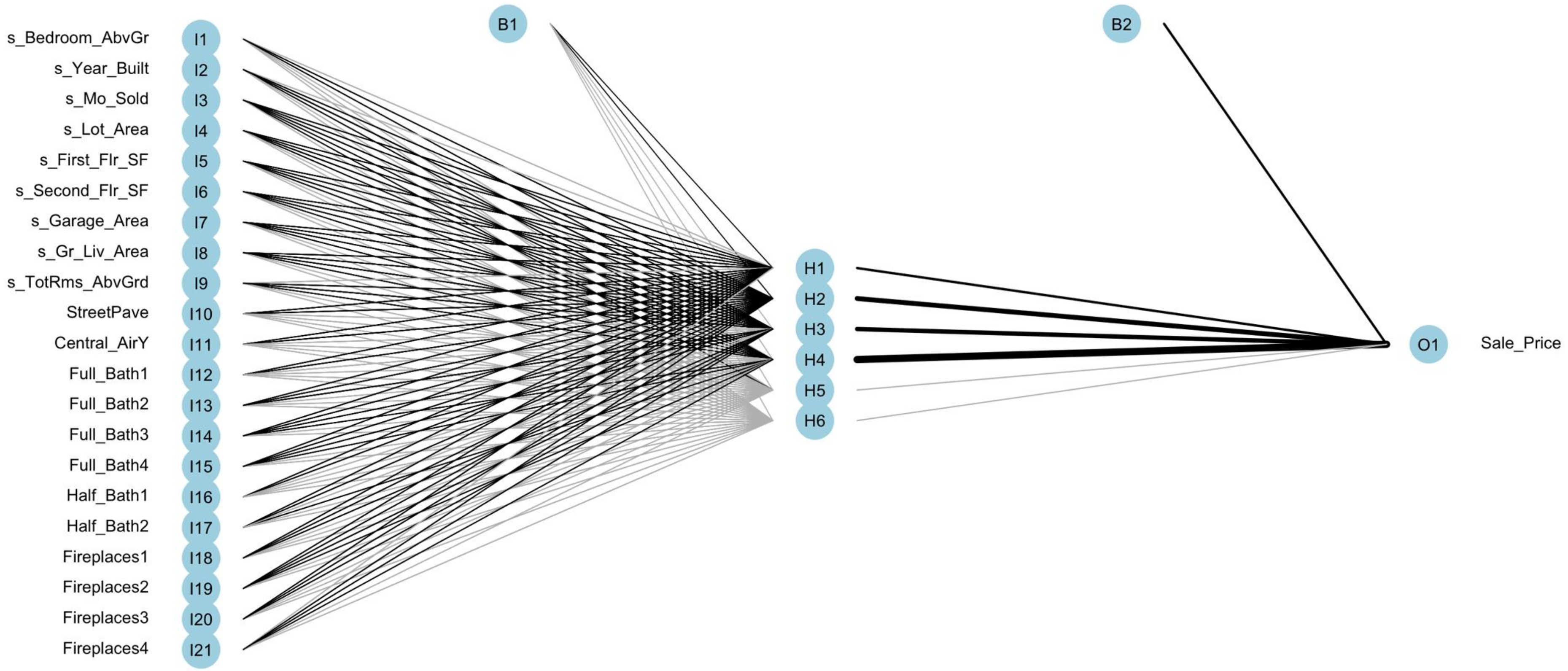
Optimized Neural Network

```
set.seed(12345)
nn.ames <- nnet(Sale_Price ~
                  s_Bedroom_AbvGr +
                  s_Year_Built +
                  s_Mo_Sold +
                  s_Lot_Area +
                  s_First_Flr_SF +
                  s_Second_Flr_SF +
                  s_Garage_Area +
                  s_Gr_Liv_Area +
                  s_TotRms_AbvGrd +
                  Street +
                  Central_Air +
                  Full_Bath +
                  Half_Bath +
                  Fireplaces
                , data = training, size = 6, decay = 1, linout = TRUE)

plotnet(nn.ames)
```

Size = 6 hidden nodes
Decay = 1 for penalty

Neural Network Plot



VARIABLE SELECTION

Variable Selection

- Neural networks typically do NOT care about variable selection.
- All variables are used by default in a complicated and mixed way.
- IF you want to do variable selection, you can examine the weights for each variable → if all weights are low, then MAYBE delete.

- Hinton diagram is one way to visualize these weights IF there are a small number of variables.

Hinton Diagram

```
nn_weights <- matrix(data = nn.ames$wts[1:126], ncol = 6, nrow = 22)

rownames(nn_weights) <- c("bias", nn.ames$coefnames)
colnames(nn_weights) <- c("h1", "h2", "h3", "h4", "h5", "h6")

ggplot(melt(nn_weights), aes(x=Var1, y=Var2, size=abs(value), color=as.factor(sign(value)))) +
  geom_point(shape = 15) +
  scale_size_area(max_size = 40) +
  labs(x = "", y = "", title = "Hinton Diagram of NN Weights") +
  theme_bw()
```

Hinton Diagram

SUMMARY

Neural Network Summary

Advantages

- Used for categorical / numerical target variables.
- Capable of modeling complex nonlinear patterns.
- No assumptions about the data distributions.

Disadvantages

- No insights for variable importance.
- Extremely computationally intensive (VERY SLOW TO TRAIN).
- Tuning of parameters.
- Prone to overfitting training data.

?