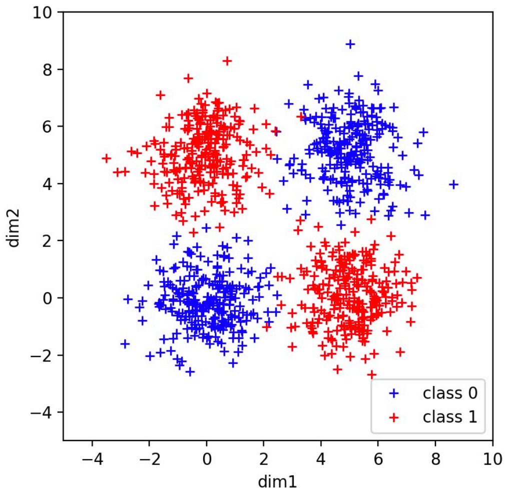
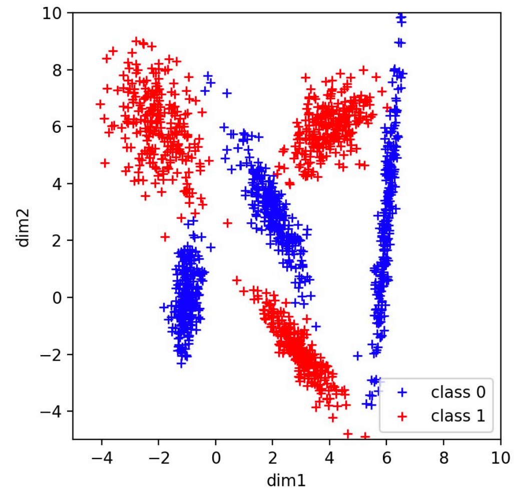
Homework 6
Due Tuesday Oct 25

builds on code from Homework 5



XOR problem



this is an example
do not copy

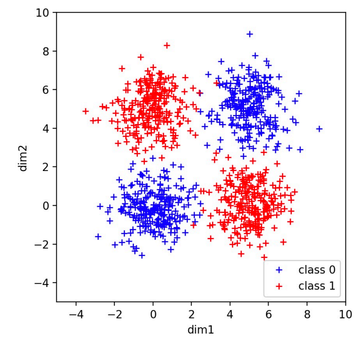
create your
own problem

must be non-linearly separable

create neural networks to learn each
of these classification problems using Keras
(using multi-layer neural networks)

Homework 6
Due Tuesday Oct 25

Q1. create an XOR problem

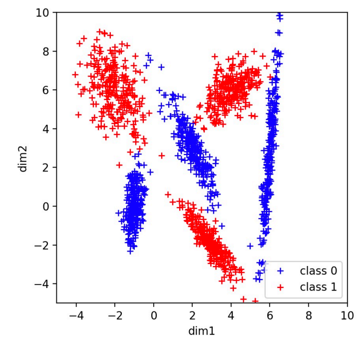


Q2. create a neural network that learns XOR problem
- pick network architecture and its settings
- hold out 20% of the training data for validation
- document your explorations and justifications

Q3. plot training accuracy x epoch, plot `loss` and `val_loss` by epoch, show "3D plot" of testing patterns

Homework 6
Due Tuesday Oct 25

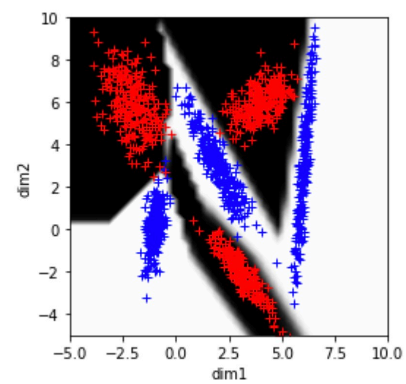
Q4. create your own problem



this is an example do not copy

Q5. create a neural network that learns your problem

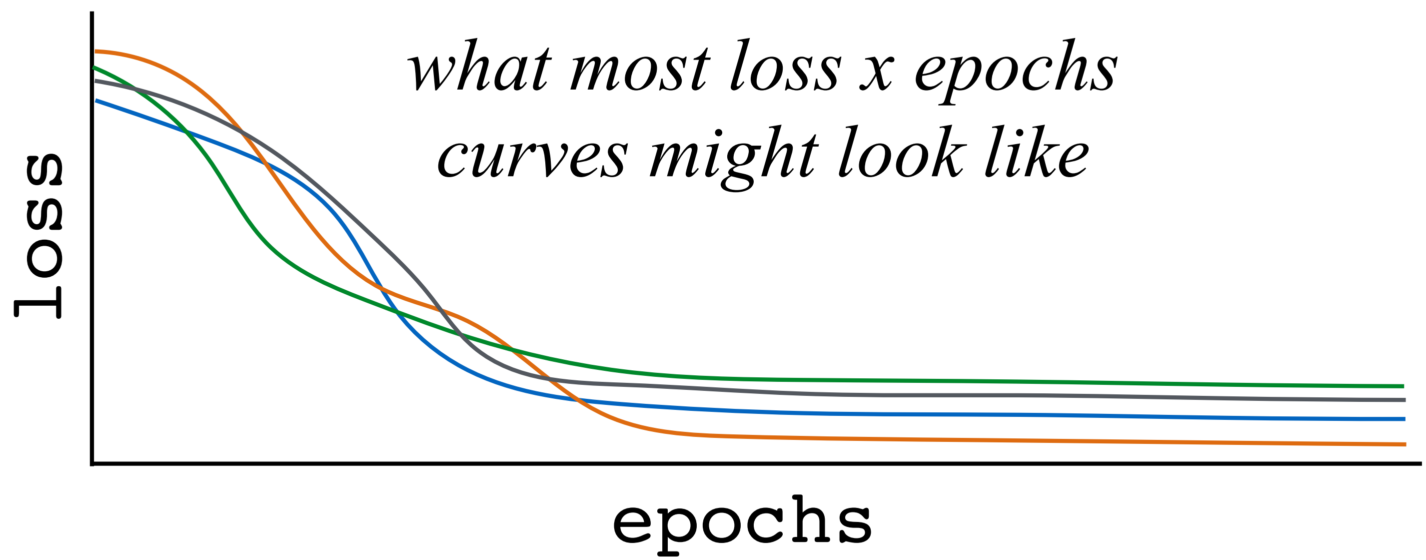
Q6. plot training accuracy x epoch, plot `loss` and `val_loss` by epoch, show "3D plot" of testing patterns



Homework 6
Due Tuesday Oct 25

Note that sometimes you can hit long plateaus in the error surface, depending on the random initialization of the weight and the randomization of the training patterns.

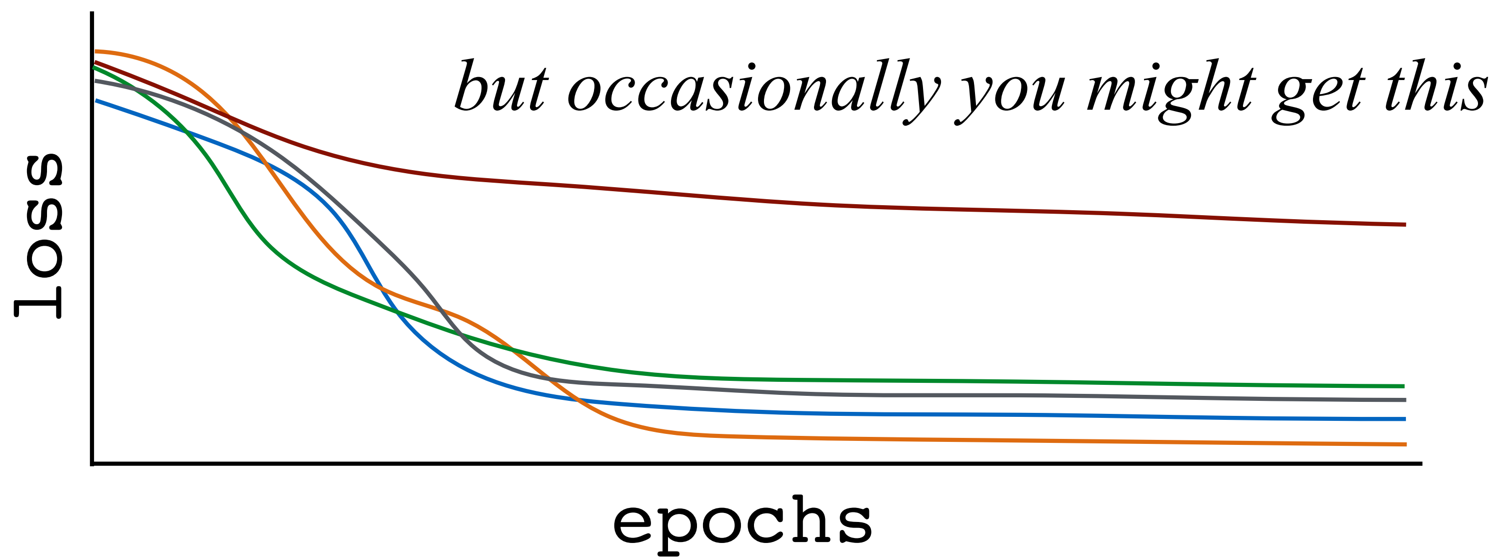
We're looking for your architecture to learn the problems most of the time



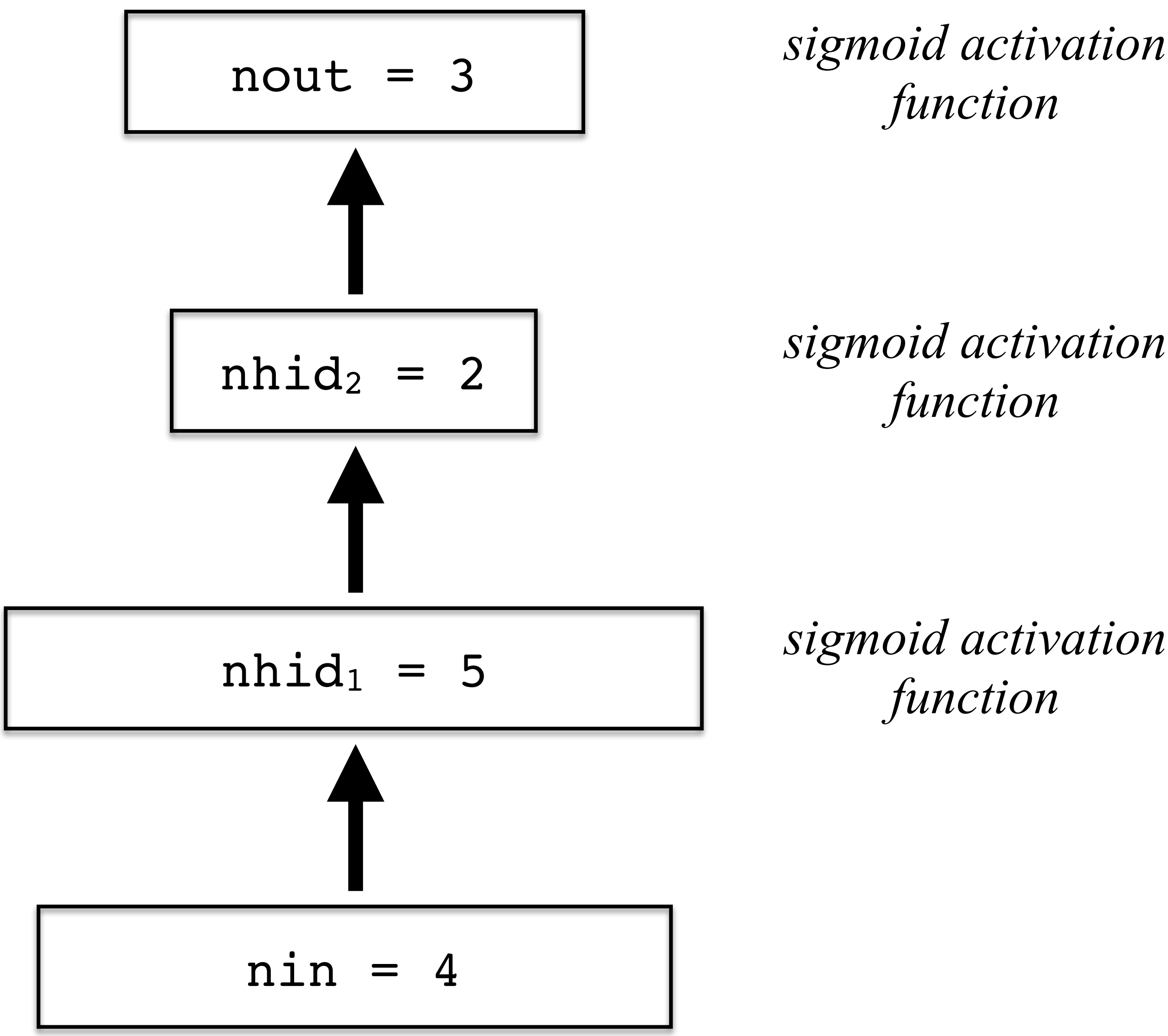
Homework 6
Due Tuesday Oct 25

Note that sometimes you can hit long plateaus in the error surface, depending on the random initialization of the weight and the randomization of the training patterns.

We're looking for your architecture to learn the problems most of the time



Reminder: implementing multi-layer networks in Keras



Reminder: implementing multi-layer networks in Keras

```
from tensorflow.keras import models
from tensorflow.keras import layers

network = models.Sequential()

nin   = 4
nhid1 = 5
nhid2 = 2
nout  = 3
network.add(layers.Dense(nhid1,
                         activation='sigmoid',
                         input_shape=(nin,)))
network.add(layers.Dense(nhid2,
                         activation='sigmoid'))
network.add(layers.Dense(nout,
                         activation='sigmoid'))
```

Reminder: implementing multi-layer networks in Keras

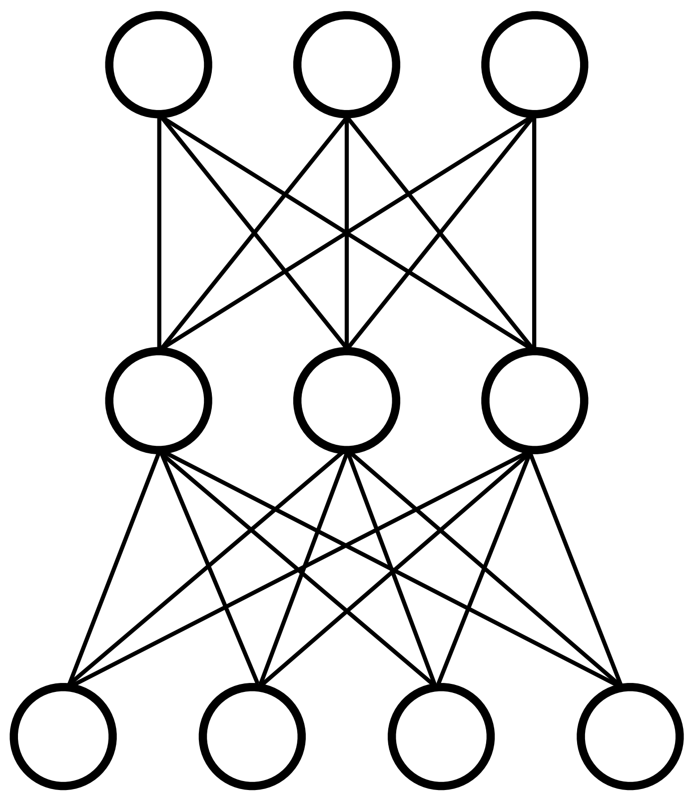
```
network.compile(optimizer='sgd',
                loss='mean_squared_error',
                metrics=['accuracy', 'mse'])

history = network.fit(train_patterns,
                      train_teach,
                      verbose=True,
                      validation_split=.1,
                      epochs=20,
                      batch_size=128)

out_test = network.predict(test_patterns)
```

Softmax Output Nodes

$$o_k = \frac{\exp(net_k)}{\sum_m \exp(net_m)}$$

softmax output **activation function** often used with **classification** networks

guarantees that the output activities sum to one - like probabilities



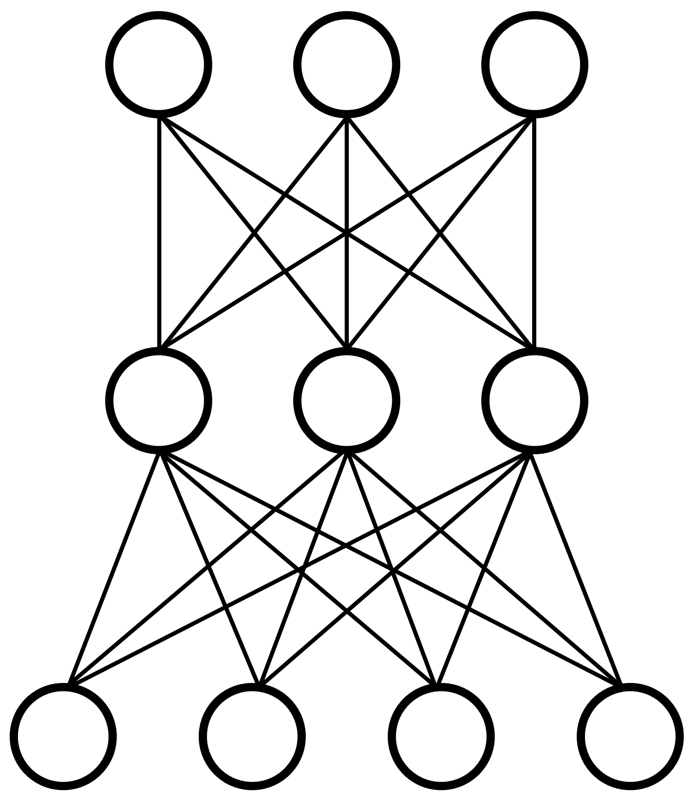
Softmax Output Nodes

$$o_k = \frac{\exp(net_k)}{\sum_m \exp(net_m)}$$

in a real neural model, this would be implemented by "divisive normalization"

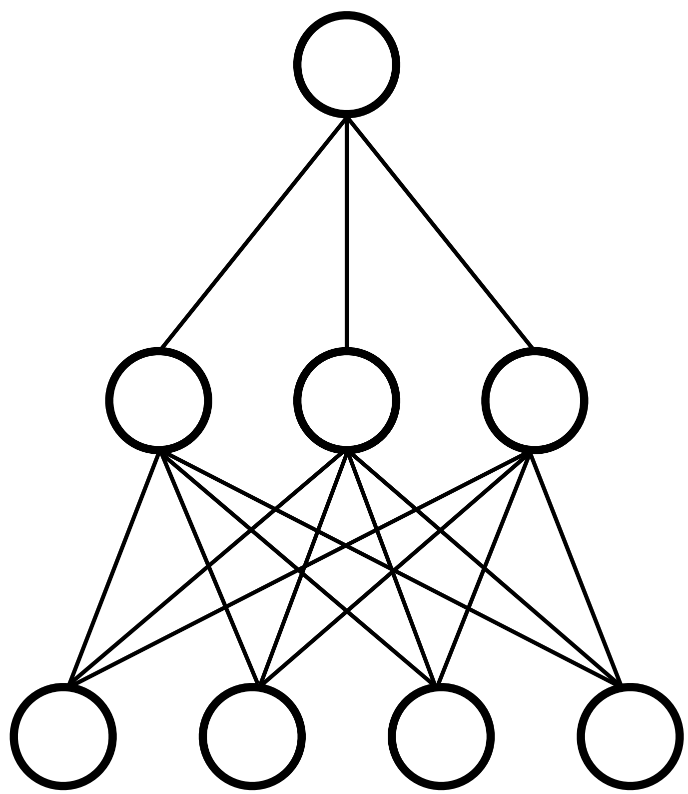
softmax output **activation function** often used with **classification** networks

guarantees that the output activities sum to one - like probabilities



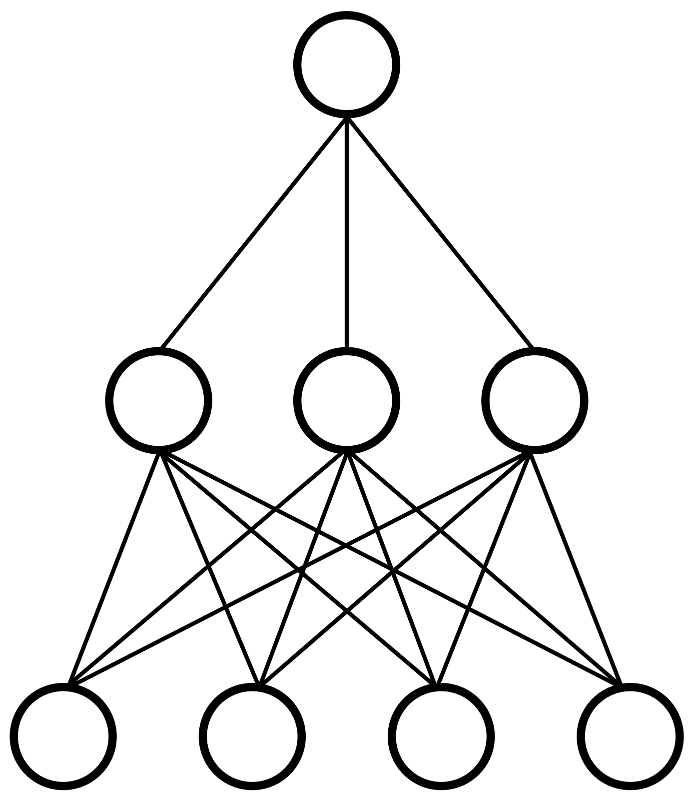
$$o_k = \frac{1}{1 + \exp(-net_k)}$$

imagine a single output node
assuming a sigmoidal activation



$$o_k = \frac{1}{1+\exp(-net_k)} = \frac{\exp(net_k)}{\exp(net_k)+1}$$

imagine a single output node
assuming a sigmoidal activation

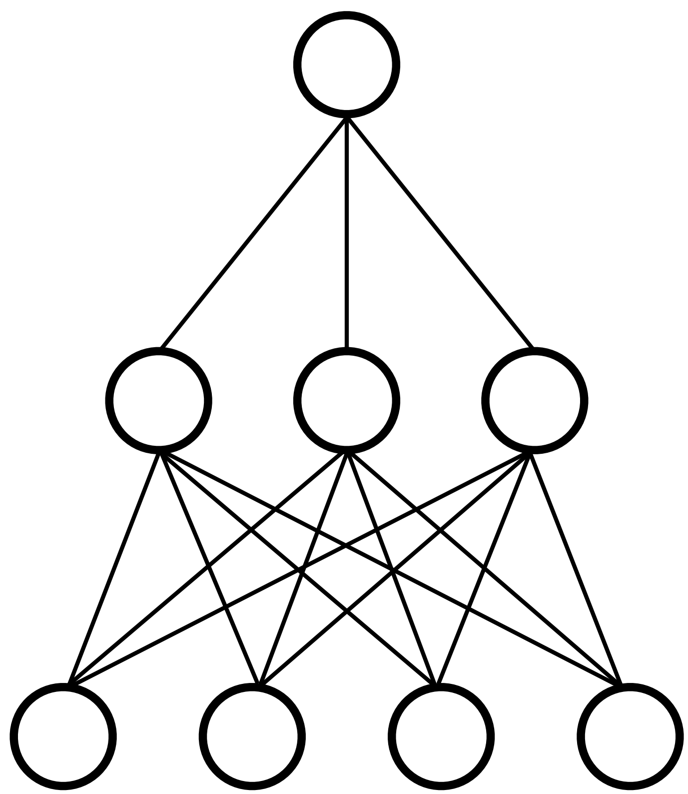


$$o_k = \frac{\exp(net_k)}{\sum_m \exp(net_m)}$$

softmax output activation is in a sense a generalization of the sigmoid

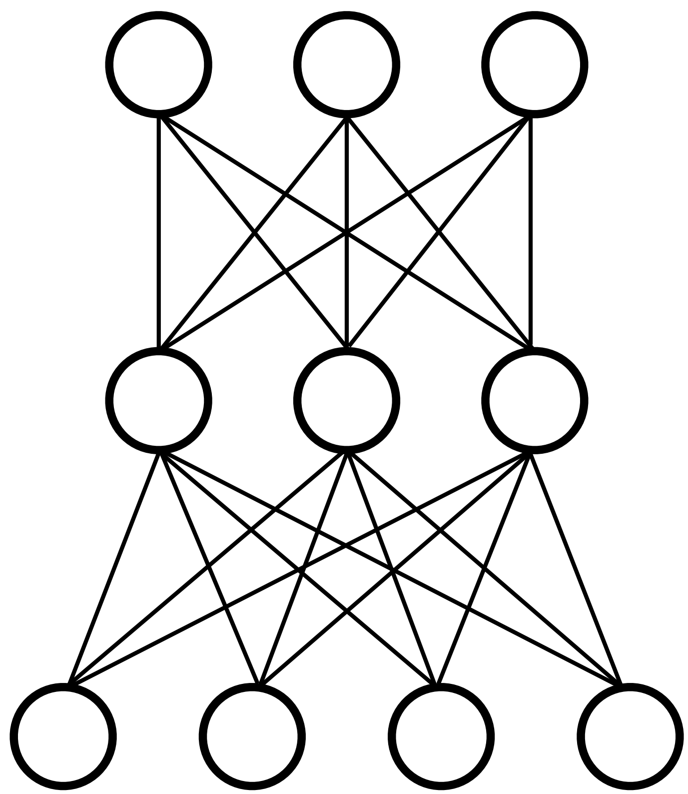
$$o_k = \frac{1}{1+\exp(-net_k)} = \frac{\exp(net_k)}{\exp(net_k)+1}$$

imagine a single output node
assuming a sigmoidal activation



$$o_k = \frac{\exp(net_k)}{\sum_m \exp(net_m)}$$

softmax output activation is in a sense a generalization of the sigmoid, but where outputs <u>activations sum to 1.0 (like probabilities)</u>



Choice of Loss Function

optimizers

'sgd'
'RMSprop'
'adam'
'nadam'
'ftrl'
etc.

```
network.compile(optimizer='sgd',
                loss='mean_squared_error',
                metrics=['accuracy', 'mse'])
```

loss function

'mean_squared_error'
'binary_crossentropy'
'categorical_crossentropy'
etc.

metrics
additional
information
stored

Softmax Output Nodes
& Categorical Cross-Entropy Loss

(instead of mean-squared error)

a different **objective** (**loss**) function

$$\boxed{o_k = \frac{\exp(net_k)}{\sum_m \exp(net_m)}}$$

$$\boxed{Err = C = -\sum_k t_k \ln(o_k)}$$

categorical cross-entropy
(assuming incremental learning)

Softmax Output Nodes
& Categorical Cross-Entropy Loss

(instead of mean-squared error)

a different **objective** (**loss**) function

$$o_k = \frac{\exp(net_k)}{\sum_m \exp(net_m)}$$

$$Err = C = -\sum_k t_k \ln(o_k)$$

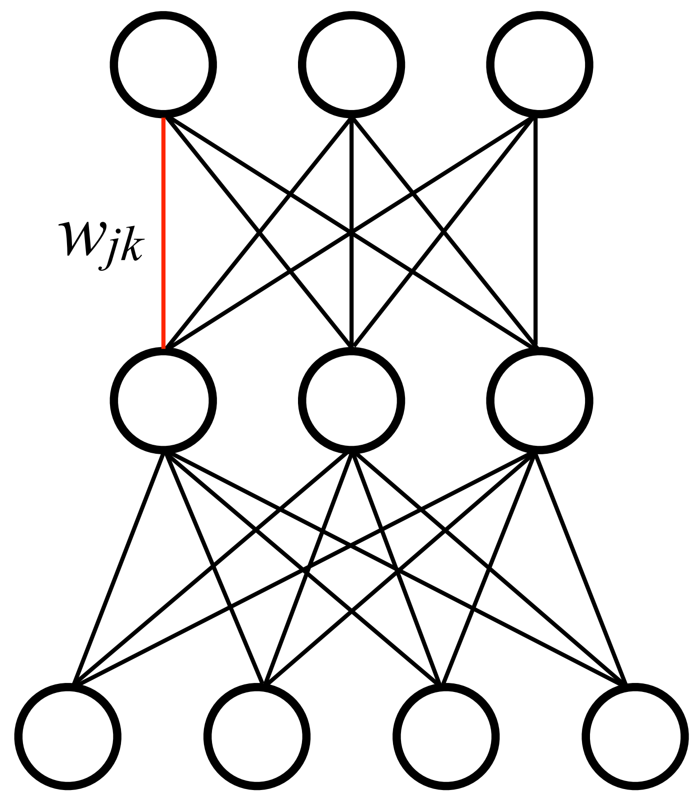
categorical cross-entropy
(assuming incremental learning)

$$-\frac{\partial C}{\partial w_{jk}} = \sum_{k'} \frac{\partial C}{\partial o_k} \frac{\partial o_k}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}}$$

$$-\frac{\partial C}{\partial w_{jk}} = \left(t_k - o_k\right) \frac{\partial net_k}{\partial w_{jk}} = \left(t_k - o_k\right) h_j$$

after a bunch of Calculus
and algebra

note that this also helps with the
vanishing gradient problem at output layer

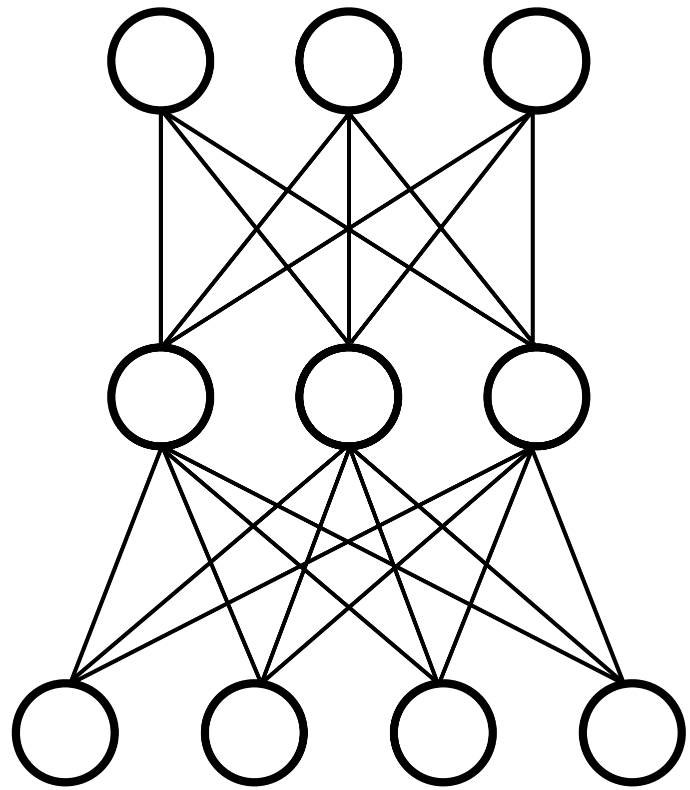


Softmax Output Nodes
& Categorical Cross-Entropy Loss

```
# output layer using softmax
network.add(layers.Dense(nout,
                         activation='softmax'))
.
.
.
# using categorical cross entropy with softmax
network.compile(optimizer='adam',
                loss='categorical_crossentropy')
```

need to use this for any classification networks (including Homework 6)

classification network with multiple output nodes

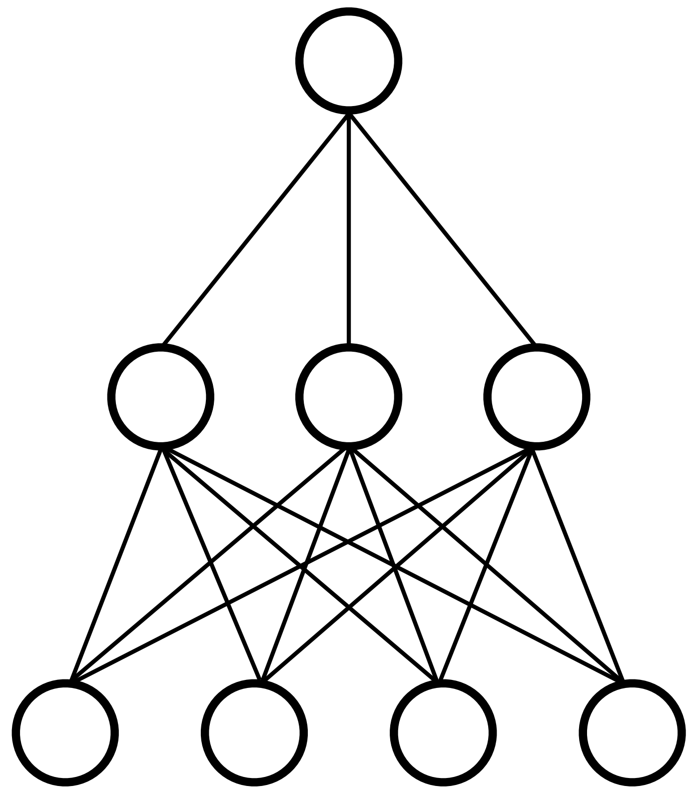


Sigmoid Output Nodes & Binary Cross-Entropy Loss

```
# output layer using softmax
network.add(layers.Dense(nout,
                         activation='sigmoid'))
.
.
.
# using categorical cross entropy with softmax
network.compile(optimizer='adam',
                loss='binary_crossentropy')
```

need to use this for any classification networks (including Homework 6)

classification network with single (binary) output



Choice of Optimizer

optimizers

- `'sgd'`
- `'RMSprop'`
- `'adam'`
- `'nadam'`
- `'ftrl'`
- *etc.*

```
network.compile(optimizer='sgd',
                loss='mean_squared_error',
                metrics=['accuracy', 'mse'])
```

loss function

- `'mean_squared_error'`
- `'binary_crossentropy'`
- `'categorical_crossentropy'`
- *etc.*

metrics
additional information stored

you can try **adam** *for Keras assignments (including Homework 6)*

it might <u>not</u> be better than **sdg**

Choice of Optimizers

```
network.compile(optimizer='sgd', loss='mean_squared_error')

network.compile(optimizer='adagrad', loss='mean_squared_error')

network.compile(optimizer='adam', loss='mean_squared_error')
```

* adam uses things like momentum, parameter-specific learning rates

differences between optimizers is a mathematics/engineering/computation topic, not a computational neuroscience topic

more modern optimizers, like **`adam`**, can fit parameters of a neural network more efficiently than **`sgd`**, but potentially at the cost of poorer generalization - the "right" optimizer depends on the neural network and the goals of modeling

https://keras.io/api/optimizers/