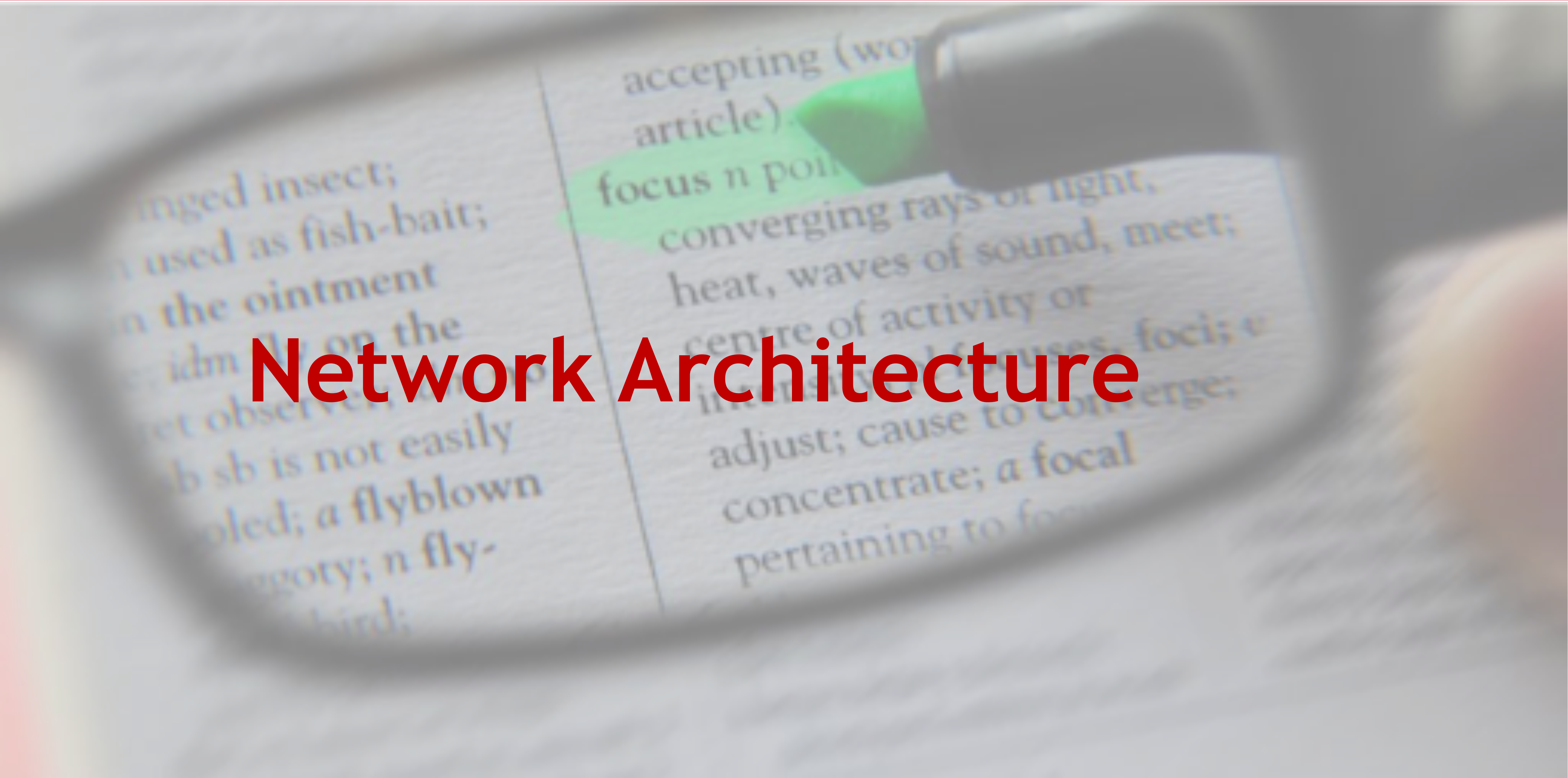
Hewlett Packard Enterprise Data Science Institute
UNIVERSITY OF HOUSTON

HPE DSI 311
Introduction to Machine Learning

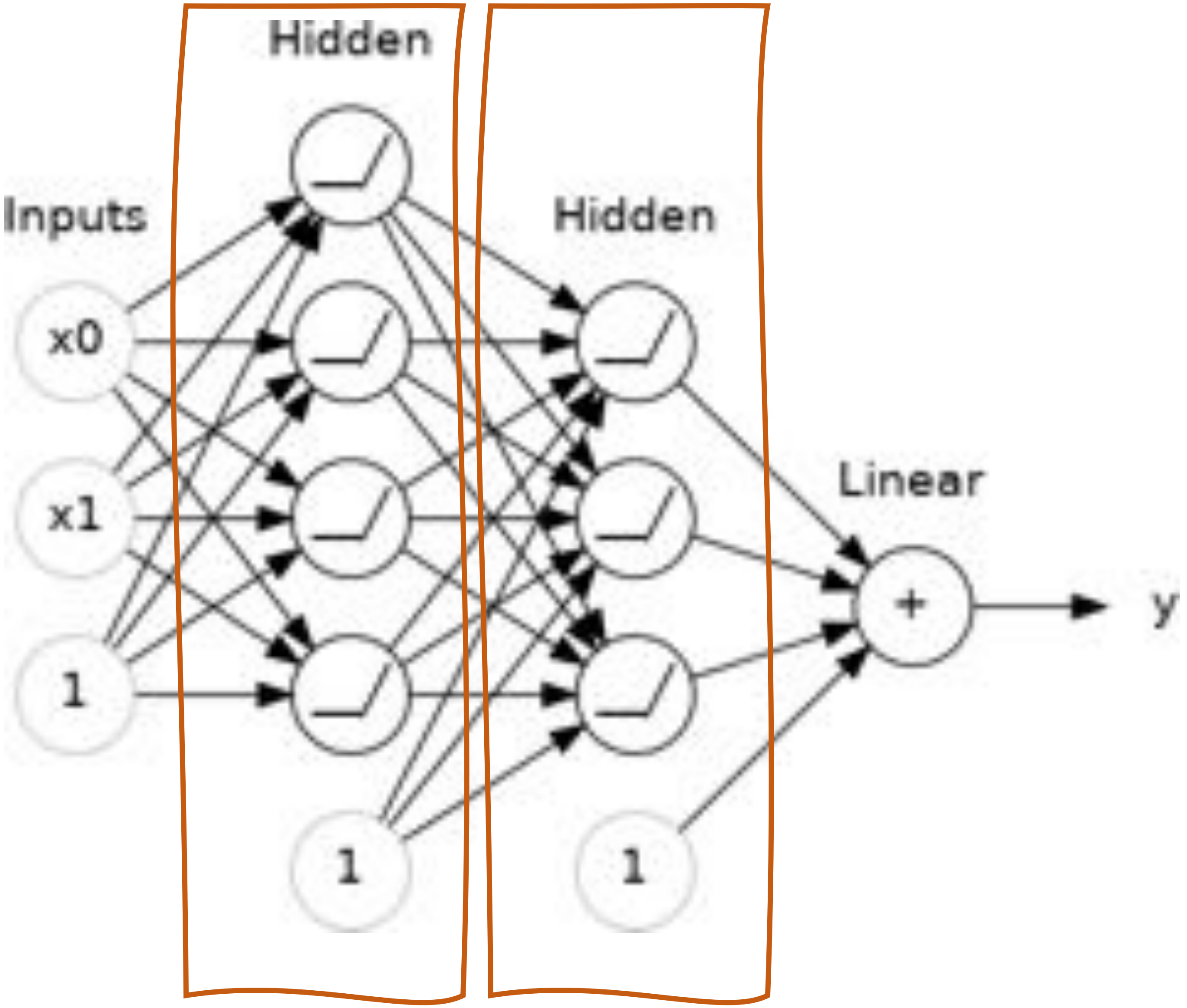
Spring 2023
Instructor: Ioannis Konstantinidis



Network Architecture

Putting it all together

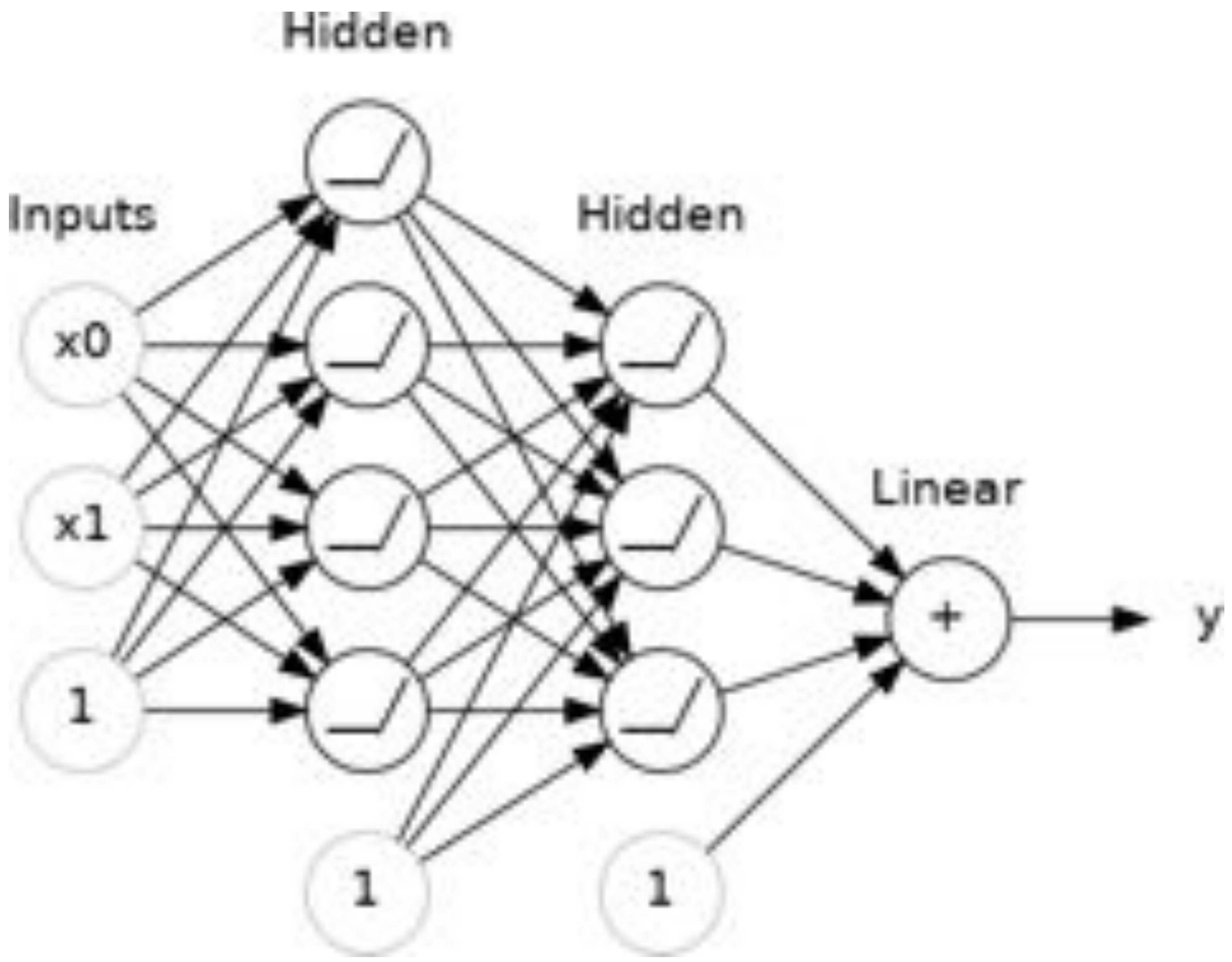
A fully-connected, feed-forward ReLU neural network with two hidden layers



Putting it all together

The architecture of a neural network model is defined by several hyperparameters:

- # of hidden layers
- # of units per layer
- type of activation function

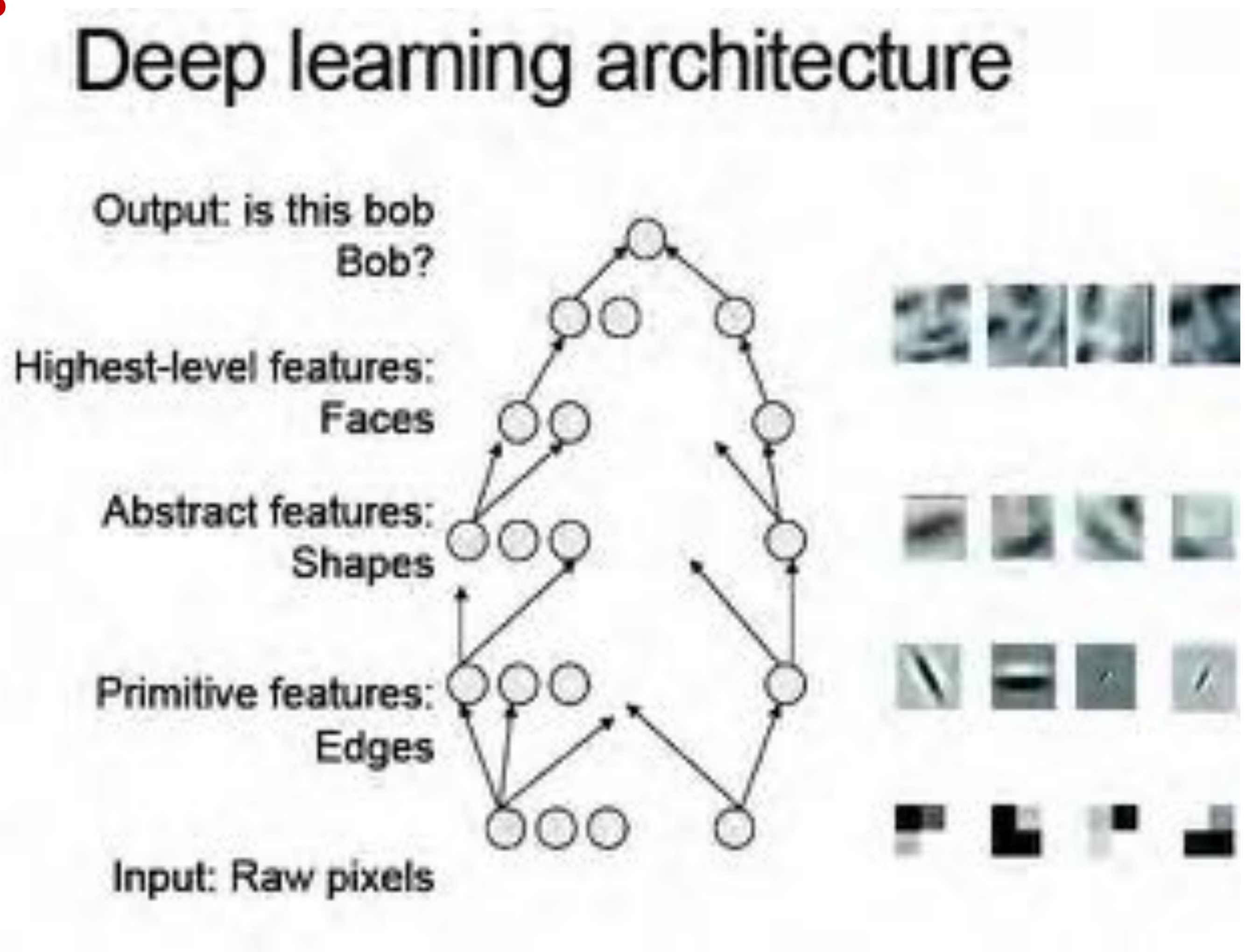


Hierarchical representations

"Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features.

Automatically learning features at multiple levels of abstraction allows a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features."

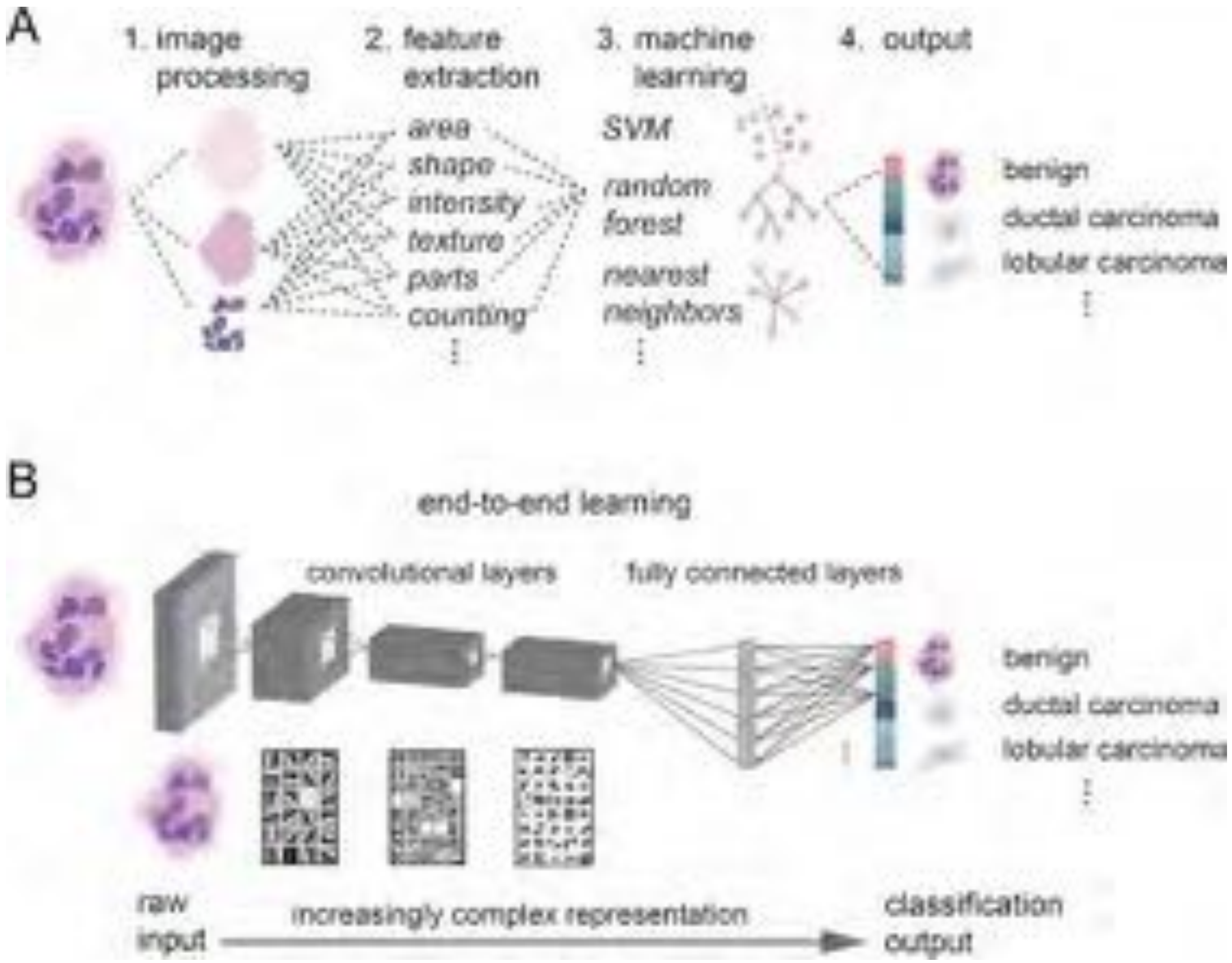
[Bengio, "On the expressive power of deep architectures", Talkat ALT, 2011]
[Bengio, Learning DeepArchitectures for AI,2009]



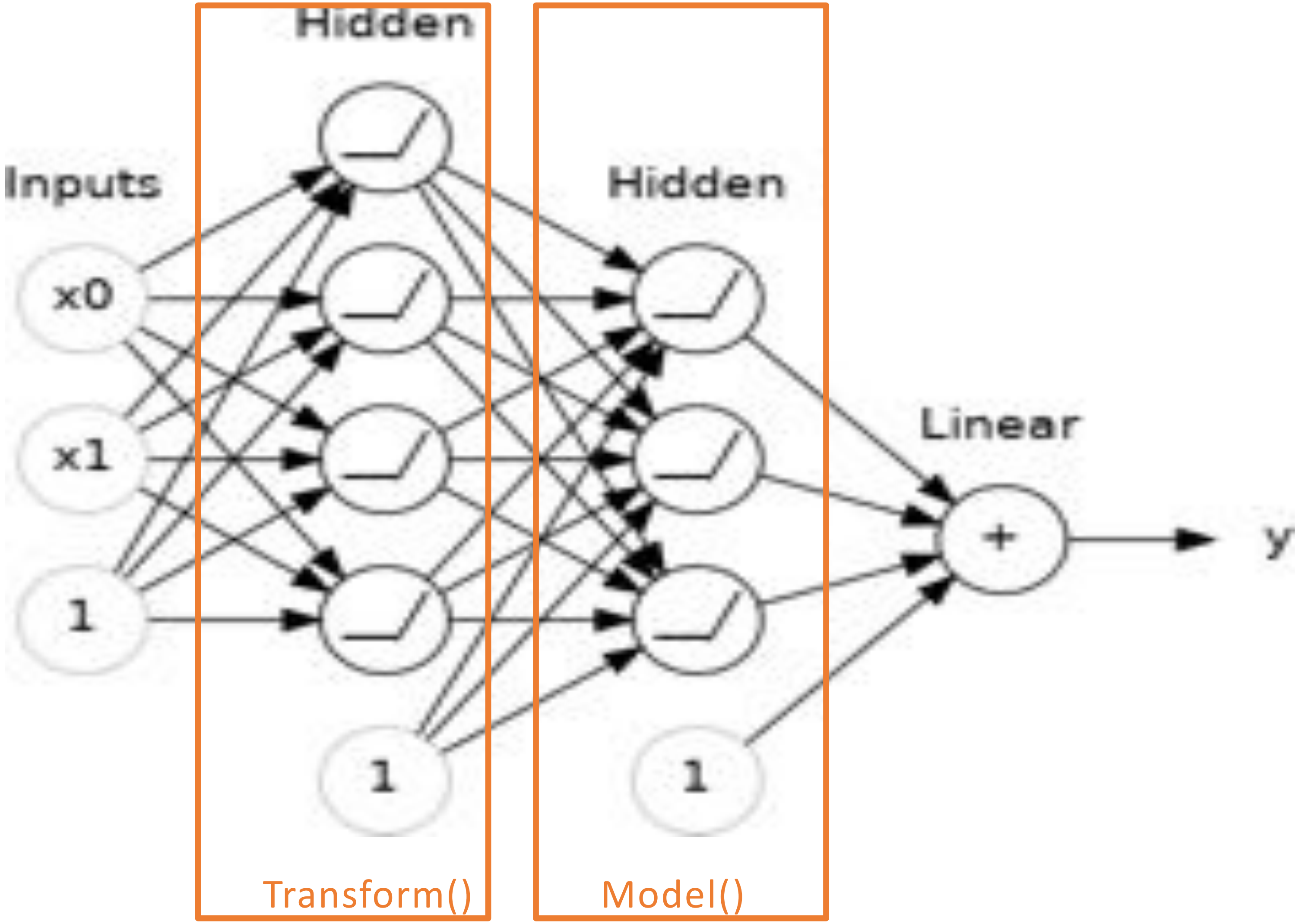
The Deep Learning Revolution

Earlier ML

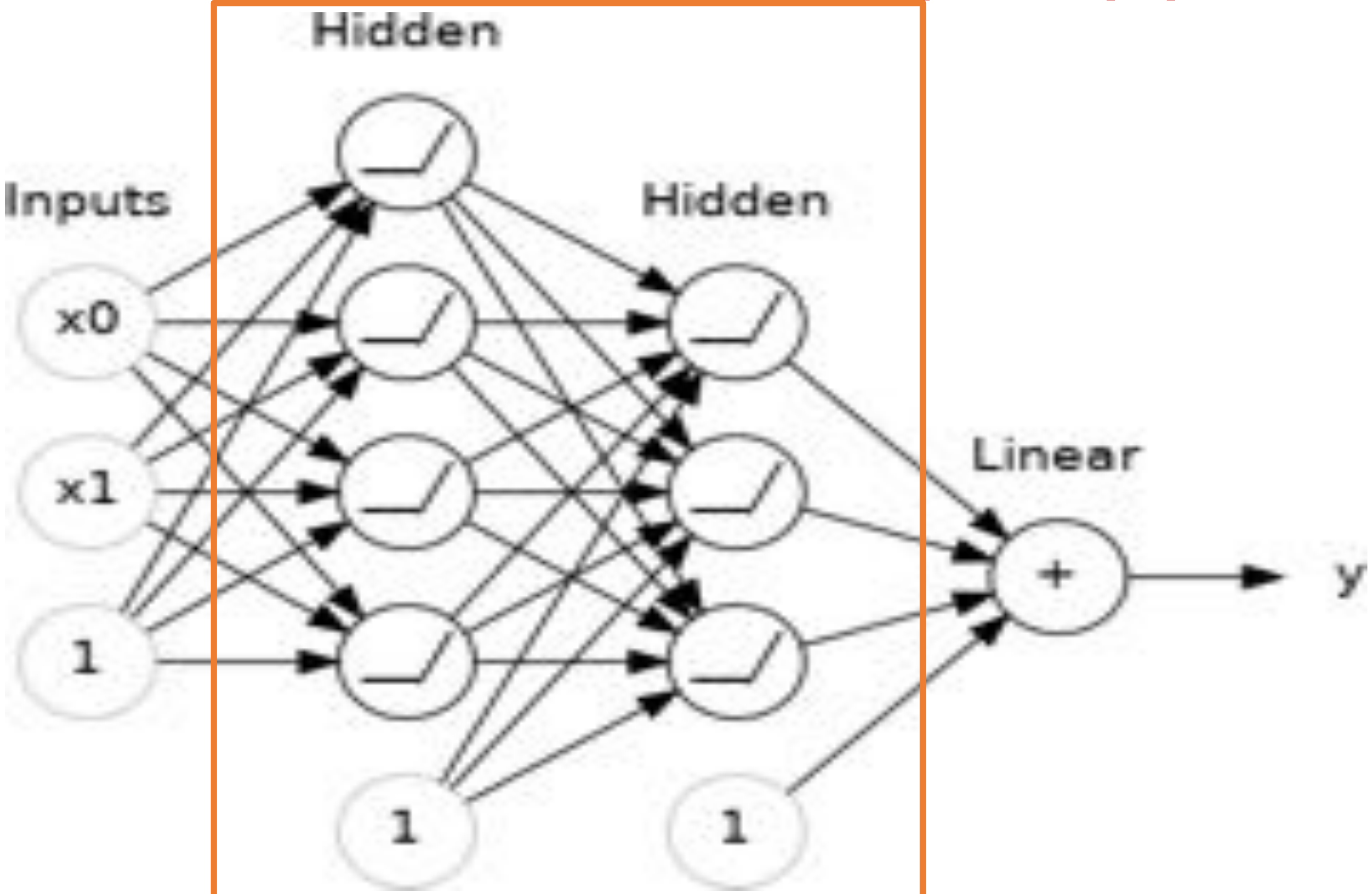
Deep Learning



Multilayer Neural Networks are a complete pipeline



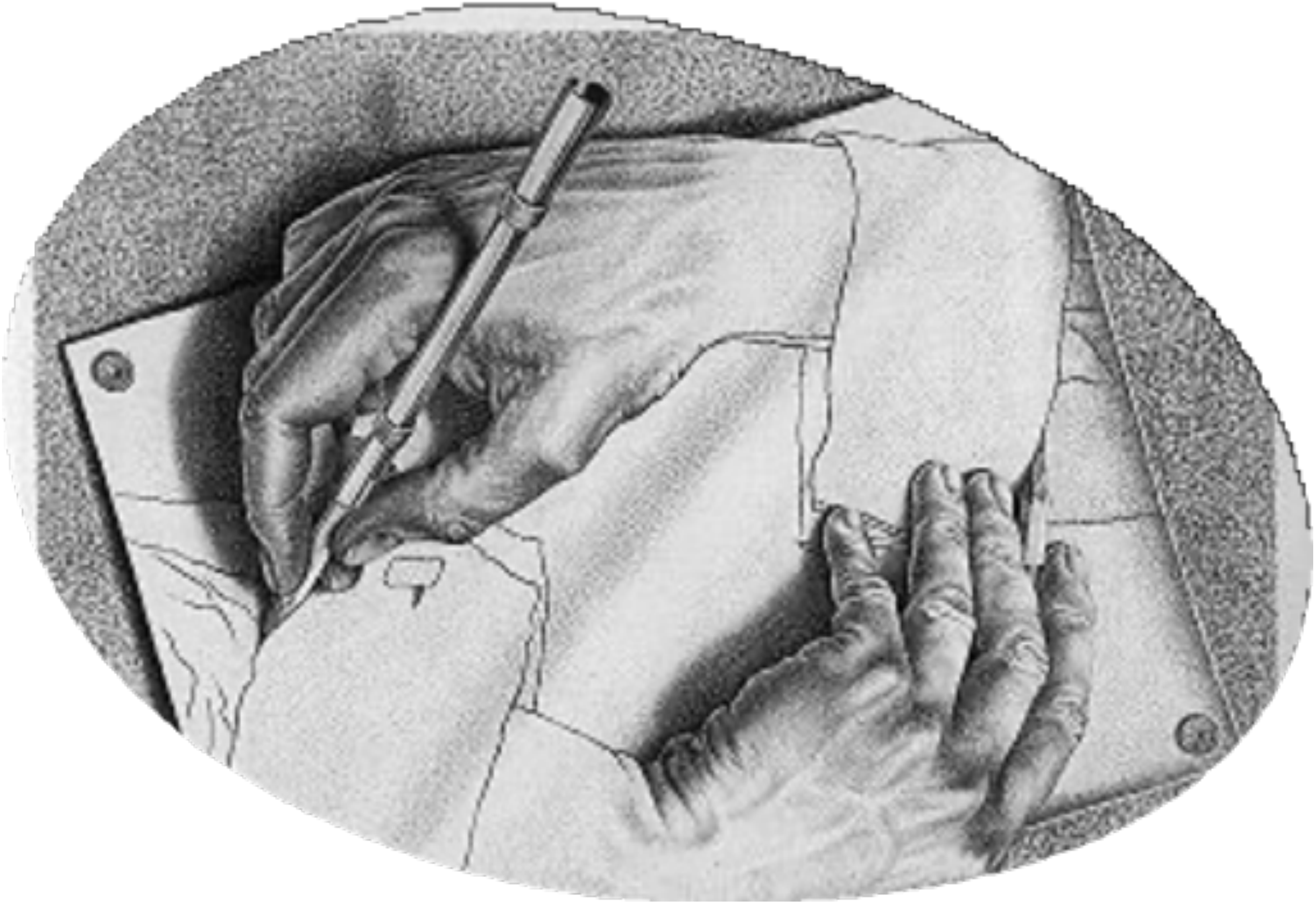
Multilayer Neural Networks are a complete pipeline



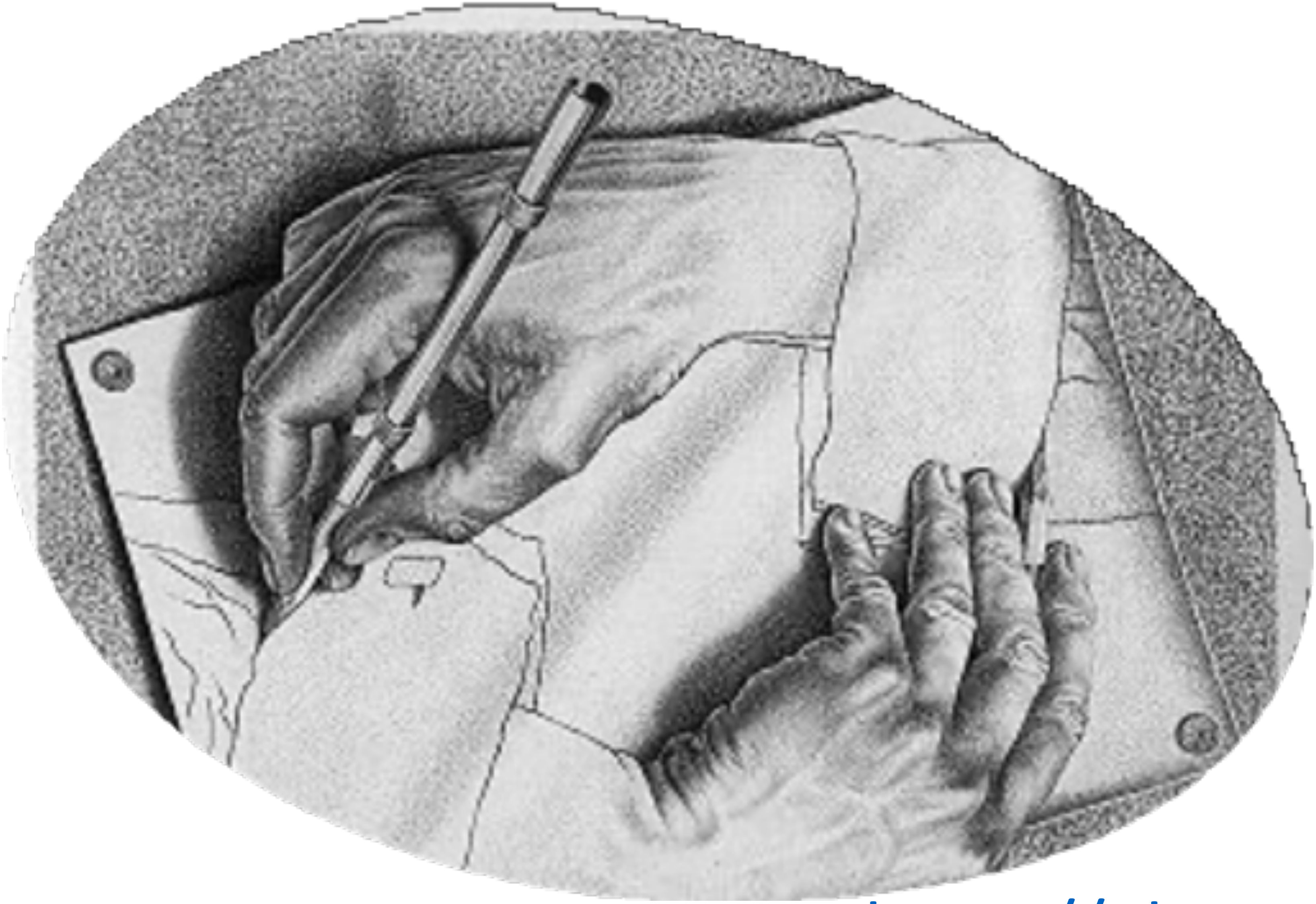
Make_pipeline(Transform,Model)

Multilayer Neural Networks are a complete pipeline

- Feature engineering is automatically “baked into” the process
- Initial layers pick out “low level” features
- Later layers process these transformed data to compute ”higher level” features
- “Top” layers perform classification tasks based on these custom-designed features



Hands-on Example: MLPClassifier



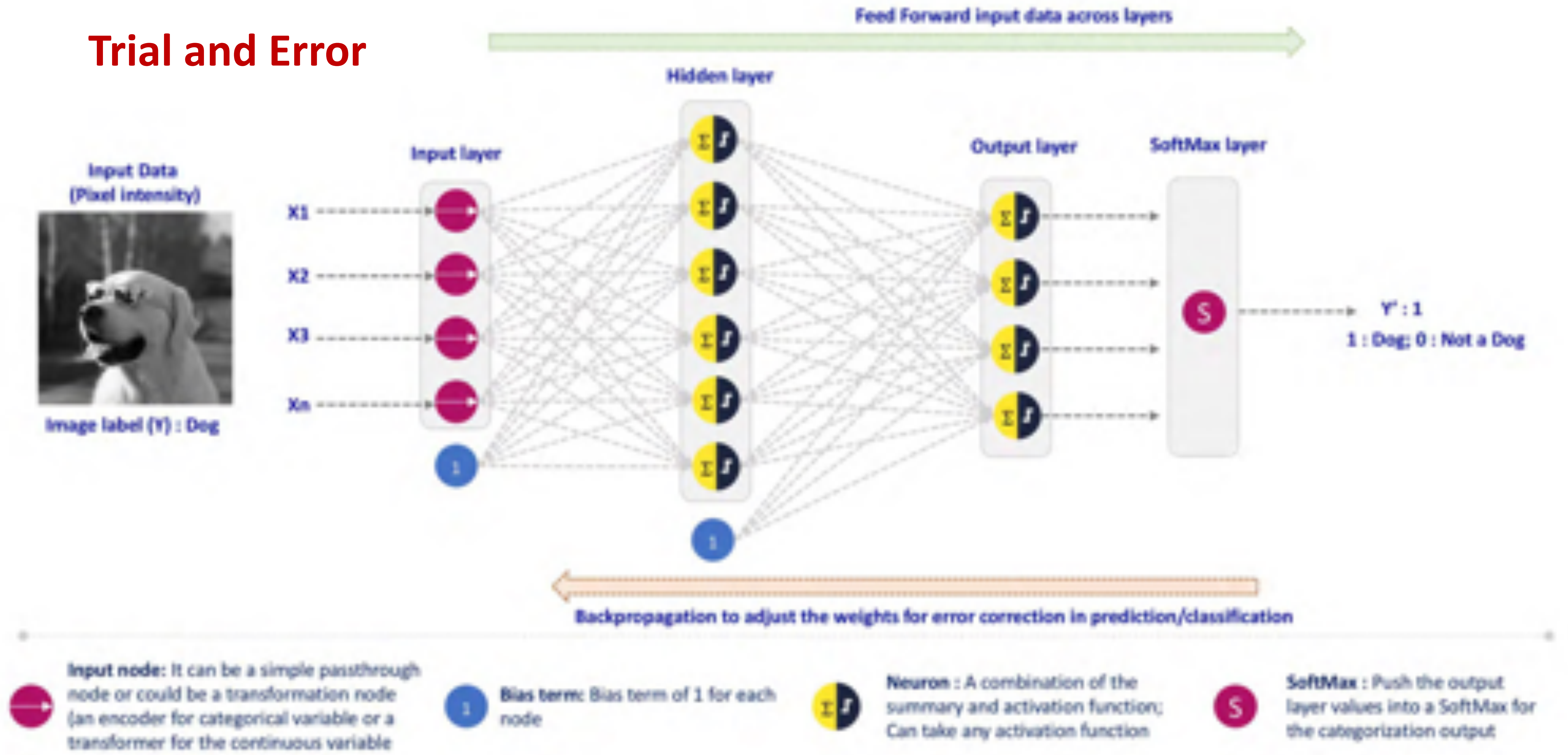
Hands-on Example:

TF playground

https://playground.tensorflow.org/

But how does a neural network learn?

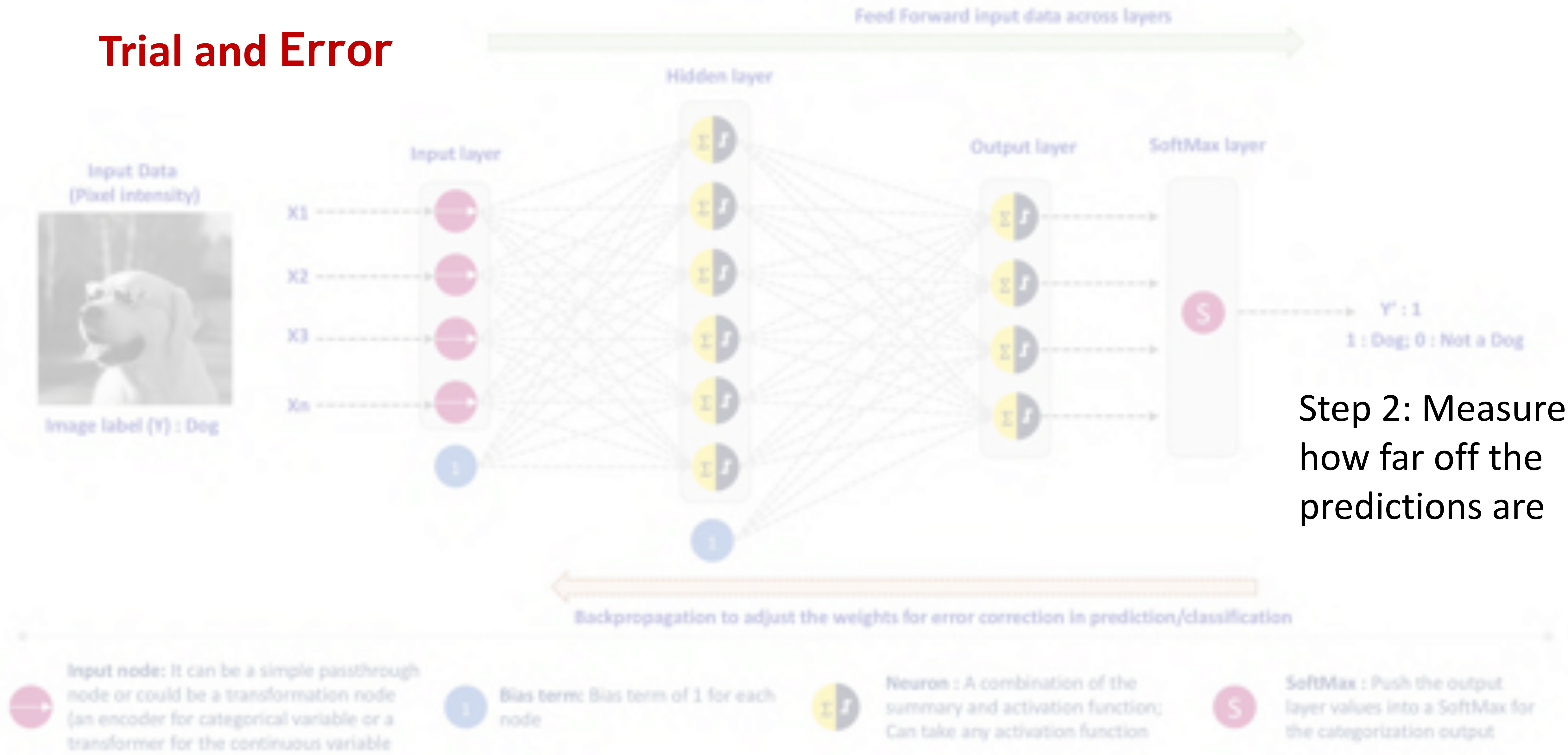
Trial and Error



Trial and Error

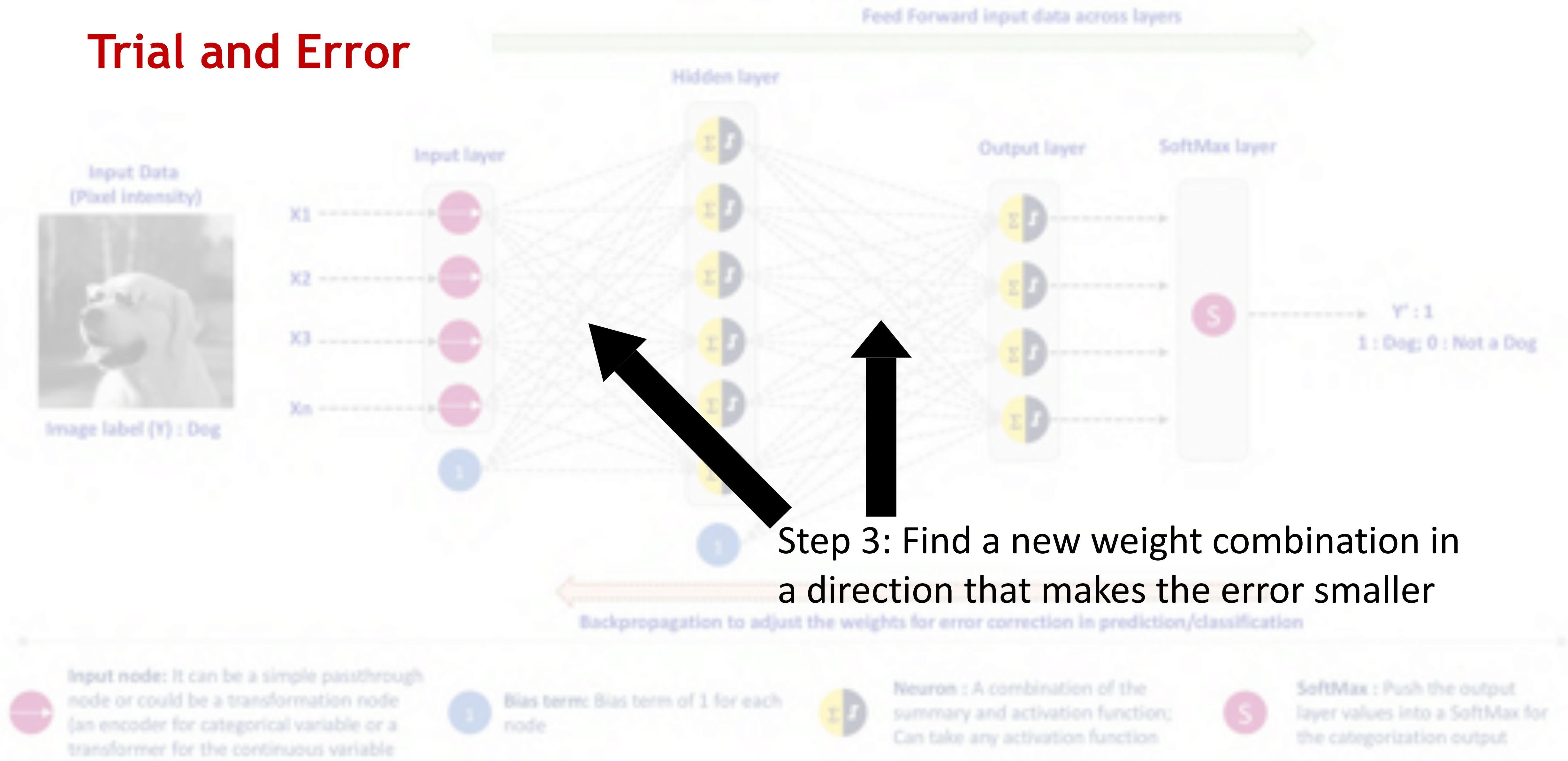
Step 1: Use training data to make predictions

Trial and Error



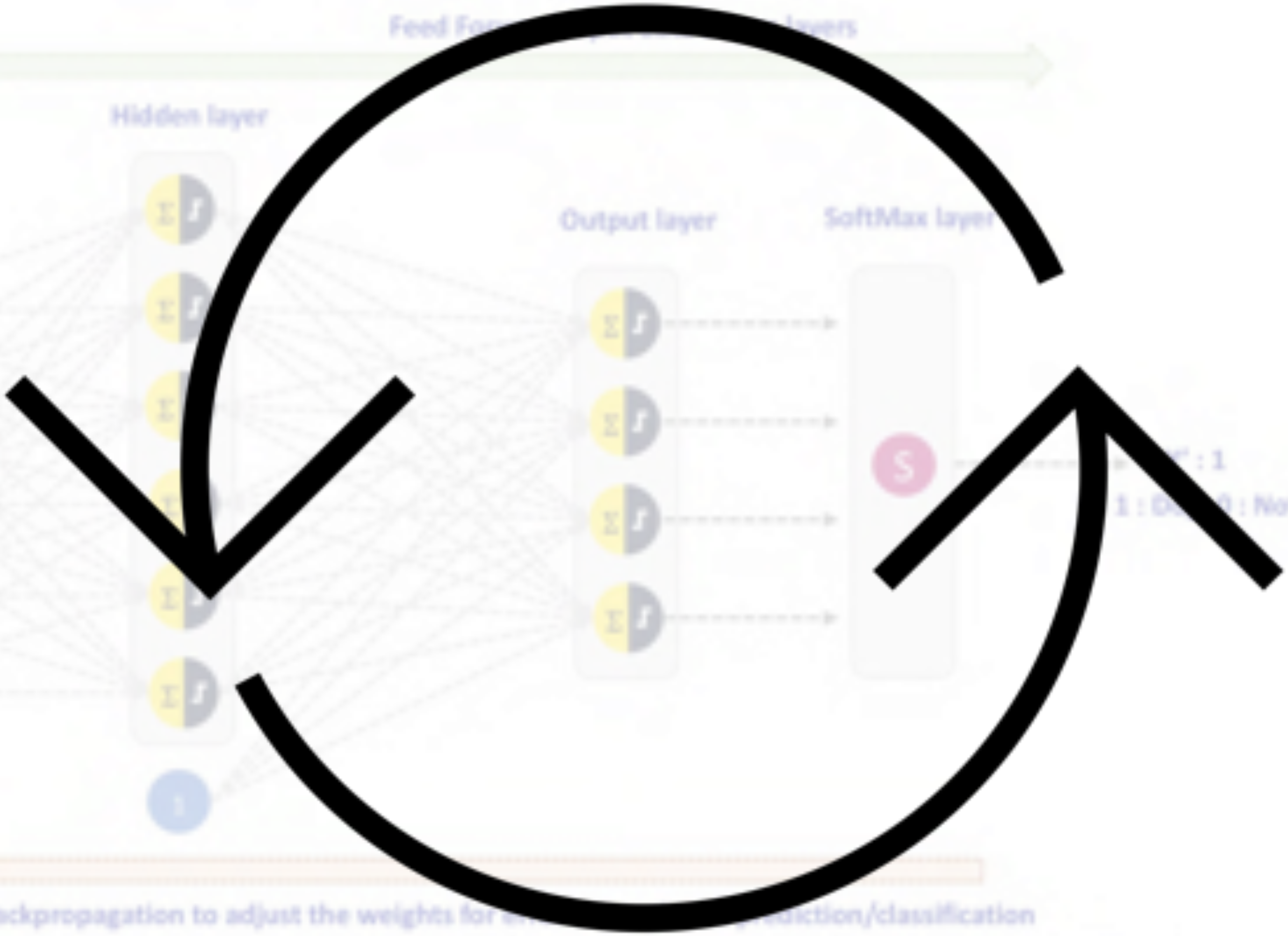
Step 2: Measure how far off the predictions are

Trial and Error



Step 3: Find a new weight combination in a direction that makes the error smaller

Trial and Error



Repeat Steps 1-3 until you reach a point of diminishing returns

Why not exhaustive search?

Curse of dimensionality

- Think of the example with the lost keys; exhaustive search in six million dimensions is prohibitive

If the loss function is differentiable, the gradient acts like a metal detector

- At any location in the yard, the detector will point you in the direction where the signal gain is strongest

Gradient descent allows you to take incremental steps towards the optimal solution

- Walk a step in the direction of the gradient and recalculate

Loss functions and optimizers

Step 2: Measure how far off the predictions are

This is the role of the objective function

- Usually called loss function when applied to Neural Networks

Least squares is a common choice: $C(w,b) = \frac{1}{2N}\sum_n \|y_n - y'_n\|^2$

- All the familiar ones apply (L2, L1, etc.)
- New choice for classification: Cross-Entropy

Regularized versions are also used (hyperparameter alpha)

Step 3: Find a new weight combination that makes smaller errors

The *optimizer*, usually a variation of stochastic gradient descent (SGD), considers each weight in turn (this can be parallelized)

Step 3: Find a new weight combination that makes smaller errors

The *optimizer*, usually a variation of stochastic *gradient* descent (SGD), considers each weight in turn (this can be parallelized)

GRADIENT

- Compute how much the total error (the loss) would be reduced by if that weight is adjusted by a fixed small amount (the *learning rate*)
- The ratio (difference in loss) / (difference in weight) is approximately the *partial derivative* of the loss function with respect to that weight, i.e.,

$$\frac{\partial C(w,b)}{\partial w_i}$$

Step 3: Find a new weight combination that makes smaller errors

The *optimizer*, usually a variation of stochastic gradient *descent* (SGD), considers each weight in turn (this can be parallelized)

DESCENT

- *Reduce* the weight by an amount proportional to $\frac{\partial C(w,b)}{\partial w_i}$
- Weights that contribute a lot (large derivative) get prioritized

Step 3: Find a new weight combination that makes smaller errors

The *optimizer*, usually a variation of *stochastic* gradient descent (SGD), considers each weight in turn (this can be parallelized)

STOCHASTIC

- The partial derivative is only an approximation of the true change in loss
- Calculate based on a *randomly* selected subset of the data

Repeat Steps 1-3 until you reach the point of diminishing returns

Each iteration's sample of training data is called a *minibatch* (or often just "batch")

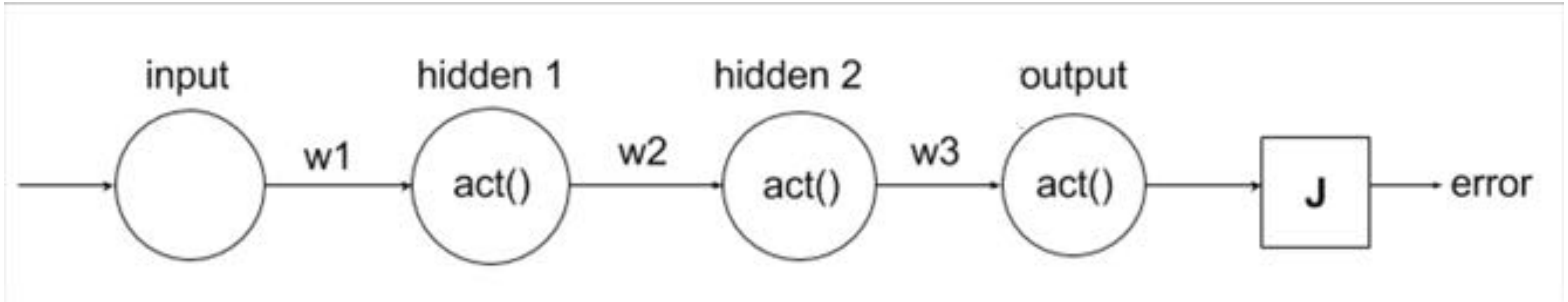
A complete round of the training data is called an *epoch*.

The number of epochs you train for is how many times the network will see each training example.

Need to compute the gradient $\frac{\partial C(w,b)}{\partial w_i}$

Neural network layers are “stacked”:

the input for one is the output of another

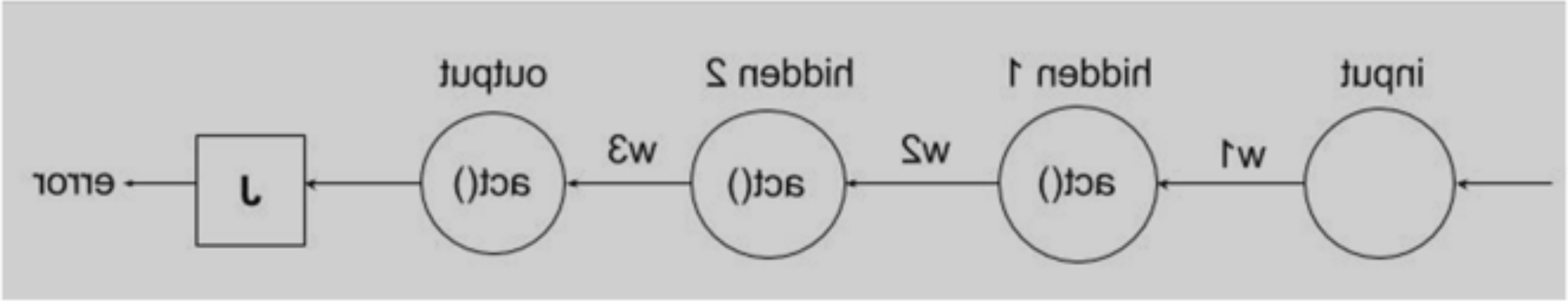


This is a composition of functions

Need to compute the gradient $\frac{\partial C(w,b)}{\partial w_i}$

Neural network layers are “stacked”:

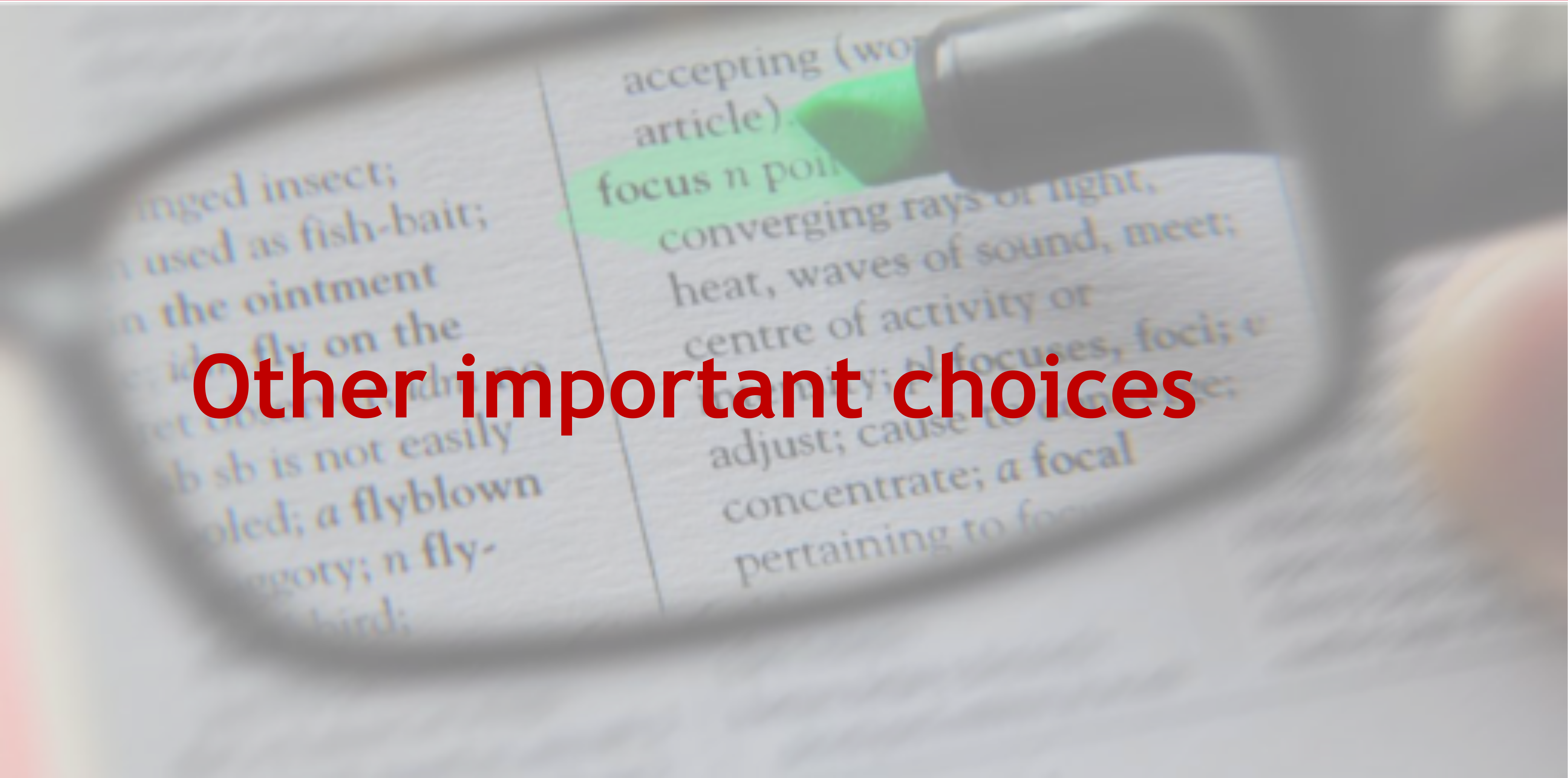
the input for one is the output of another



This is a composition of functions, so we can use the chain rule from calculus; work backwards from the last (top) layer in:

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

Other important choices



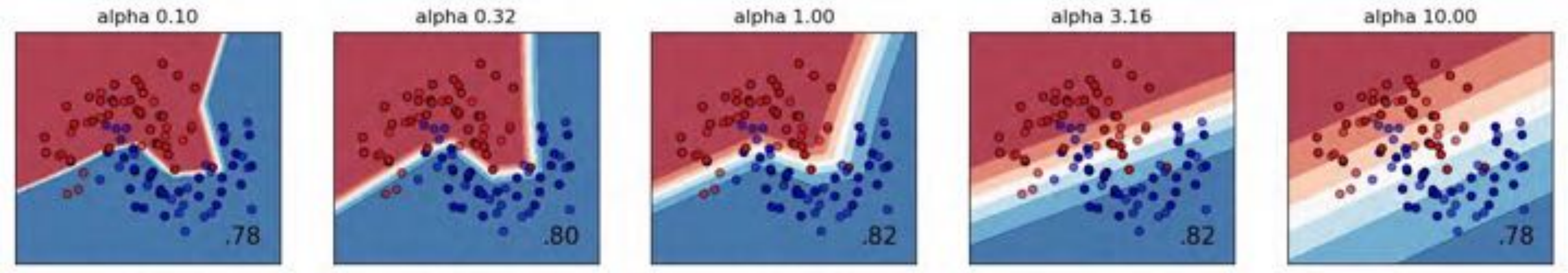
Activation functions

- Sigmoid function and Hyperbolic Tangent function: The gradient is very close to zero over a large portion of its domain which makes it slow and harder for the learning algorithm to learn.
- Rectified Linear Unit (ReLU): $g(z) = \max\{0, z\}$. Since ReLU shares a lot of the properties of linear functions, it tends to work well on most of the problems. However, this means that for $z \leq 0$ the gradient is zero and again can't learn.

Alpha

Default is close to zero, so little regularization

Higher alpha means stronger regularization
(less prone to variance/overfitting,
but more prone to bias/underfitting)



Learning rate

The learning rate determines how far to go in the direction that makes the error smaller.

- For example, the optimizer may estimate that a change of one in the value of the weight connecting input variable X_{143} to the 54th ReLU of the first hidden layer will reduce the error by 3.72 units

But this is only an approximation of the true change.

- How big a step to take? How much to change the weight?

Learning rate

The learning rate has a small positive value, often in the range between 0.0 and 1.0

Smaller learning rates mean more conservative changes in weights, taking smaller steps so the approximation will stay valid

- This requires more training epochs; more steps to travel down the optimization path

Larger learning rates can also cause problems; the approximation may no longer hold, and the error is not actually reduced in an optimal way

- The model might converge too quickly to a suboptimal solution (rushing to make hasty decisions)

Learning rate

Instead of a constant learning rate, it is recommended to use a high learning rate during the start and reduce it during training.

Typically optimizers take care of this automatically.

Adam is a type of SGD algorithm that has an adaptive learning rate that makes it suitable for most problems without any parameter tuning (it is "self tuning", in a sense); it is a great general-purpose optimizer

Batches and epoch

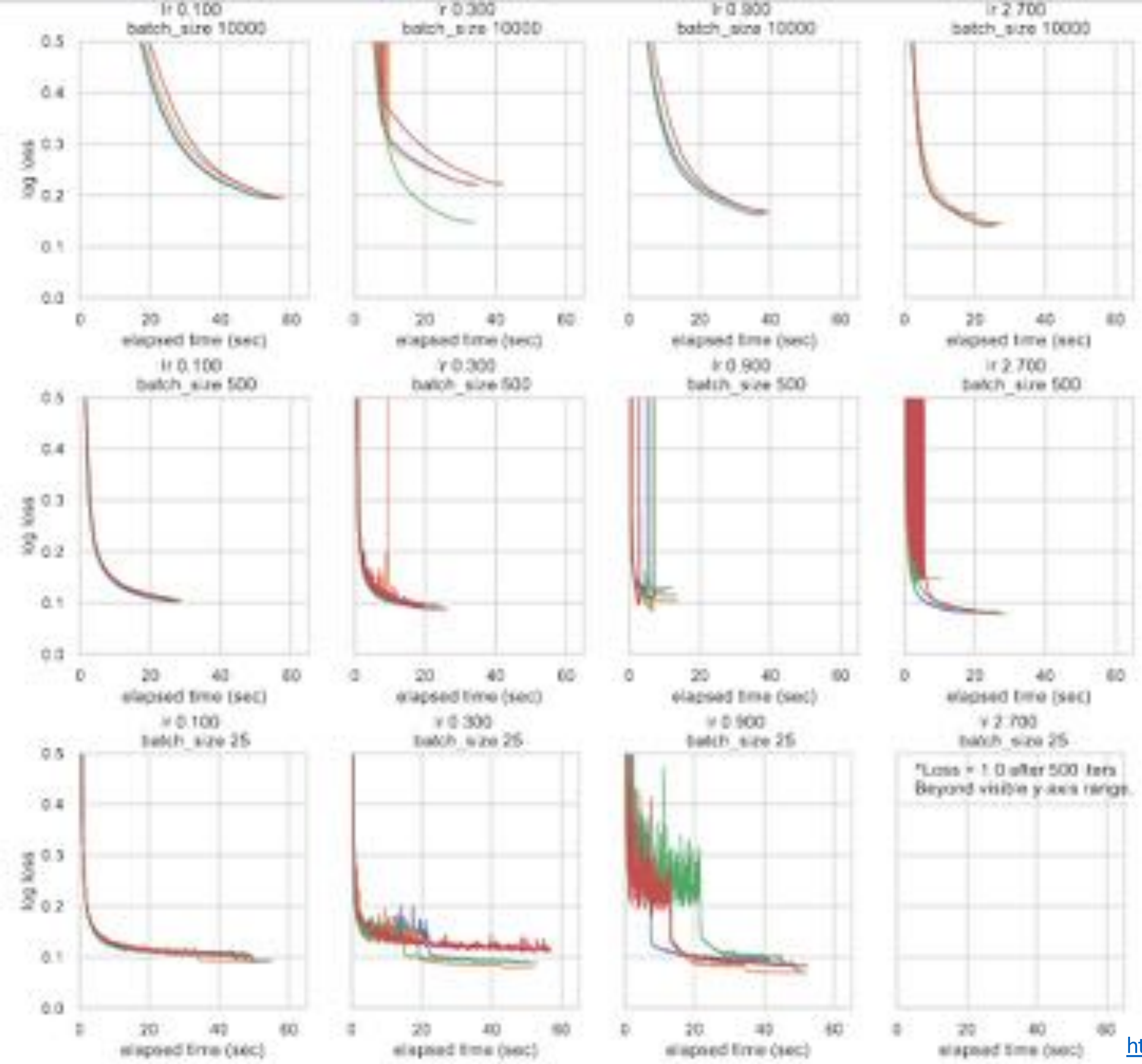
Balance between:

- true stochastic gradient descent (calculate and update separately for each training example)
- true batch gradient descent (calculate and update based on all training examples)

Split the training dataset into small batches of size batch_size

Calculate model error and update model coefficients one batch at a time

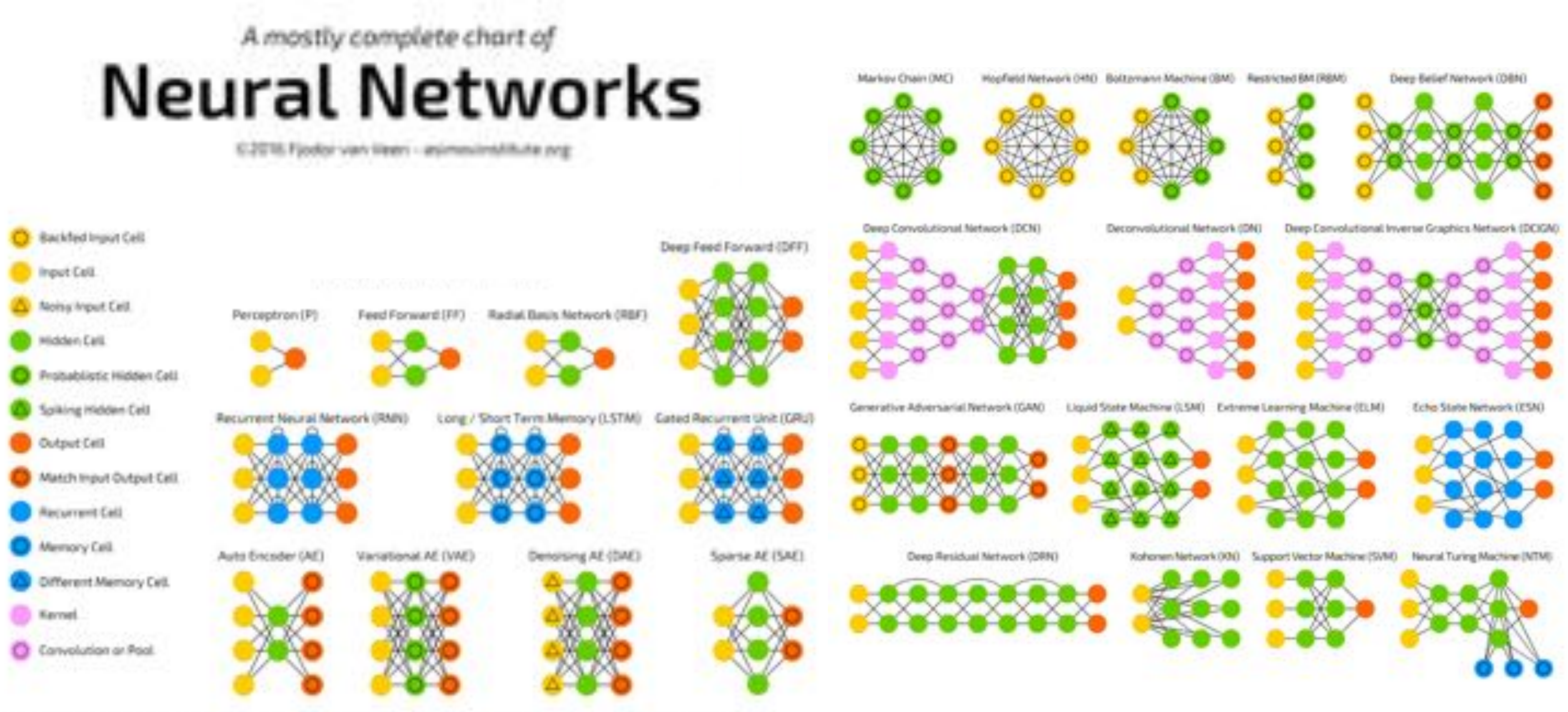
(# of epochs) * (batch_size) = # of training examples



Batch size and learning rate work together

https://www.cs.tufts.edu/comp/135/2020f/images/hw3_fig2_sgd_loss_vs_time.pdf

Homework Assignment #3
Due Sunday (February 26), 11:59 pm (Central)


A mostly complete chart of
Neural Networks
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)