Data Science Initiative for Africa (DSI-A) - Deep Learning Lecture 2

MLPs, Backpropagation, and coding neural networks in Python

Santiago Romero-Brufau
Harvard T.H. Chan School of Public Health
Spring 2

Administrivia

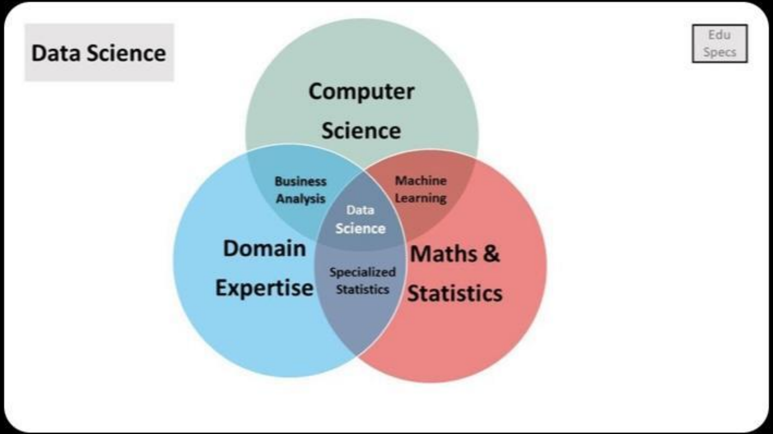
- ChatGPT is a great tool to help debug your code, give you ideas about things to try, etc. https://chat.openai.com/
- It doesn't get everything right, but it's very useful.
- The paid version is useful, but somewhat expensive ($20 (R358))

- Form: https://forms.gle/RZHWxhbBBMes7yPx8

Kareem Carr | Data Scientist @kareem_carr

Is Data Science at risk of disruption by ChatGPT?

Based on a recent report by OpenAI, I conclude the risk is low.



10:11 AM · 3/21/23 · **19.1K** Views

21 Retweets **104** Likes **28** Bookmarks

Kareem Carr | Data Scientist · 12h

The report doesn't directly discuss data science roles but we can draw some conclusions.

Covidiots everywhere @covidiot... · 12h

What about software engineering. I am hearing GPT-4 has the capabilities to upend it

1 1 581

Kareem Carr | Data Scientist · 12h

High risk yes.

1 495

Kareem Carr | Data Scientist @kareem_carr

Data sciences roles which tightly integrate mathematics with critical thinking, scientific analysis and subject matter knowledge are likely to be much more resistant to disruption.

10:11 AM · 3/21/23 · **2,897** Views

Kareem Carr | Data Scientist
@kareem_carr

People find the sentence "my artificial neural network is sentient" vastly more plausible than the sentence "my matrix multiplication algorithm is sentient" even though they are roughly the same claim.

This implies to me that "neural network" is bad terminology.

12:22 PM · 1/5/23 · **140K** Views

Kareem Carr | Data Scientist · 1/5/23

It smuggles in some assumptions that should perhaps be expressed in a much more explicit manner.

If I tell you that sentence is just this 👇 with much bigger matrices, you should probably be extremely skeptical.

$$\begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} = A_1B_1 + A_2B_2 + A_3B_3$$

Bojan Tunguz @tunguz · 1/5/23

"Roughly" is carrying a *LOT* of water in that statement. 😉

1 · 13 · 2,090

Kareem Carr | Data Scientist · 1/5/23

Can't say it better than this: twitter.com/seanluomdphd/s...

> **Sean X. Luo** @seanluomd... · 1/5/23
>
> ReLU = sentience.

1 · 8 · 2,397

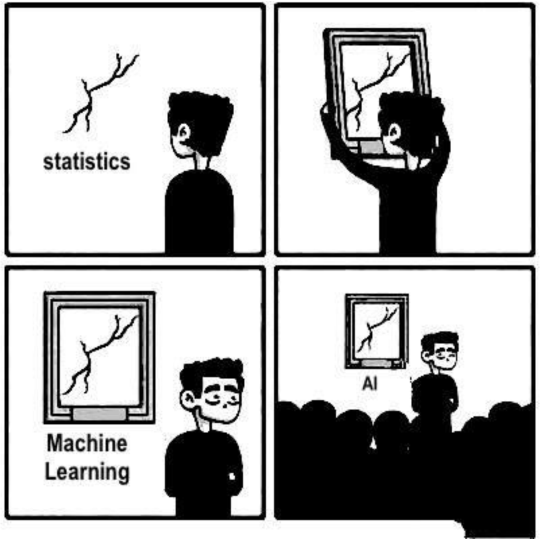
Pabnau @pabnau · 1/5/23

I've mostly agreed with your tweets, but I gotta push back here. You're confusing the part for the whole. Just because ANNs are composed of matrix mults (and other things), doesn't mean they are equivalent. ANNs are non-linear, and recurrent versions can simulate Turing machines

2 · 1 · 32 · 2,440

Deep learning glossaries

1. Google
2. WildML



First, recap from last class

Conceptually, a NN has three components.

In each node we just make simple multiplications and sums, and multiply it by an activation function

First, recap from last class

Conceptually, a NN has three components.

- The network structure
- A loss function
- An optimizer

In each node we just make simple multiplications and sums, and multiply it by an activation function

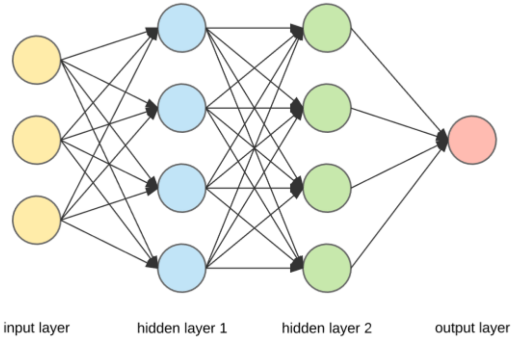
What is a neural net?

A neural net is composed of 3 things:

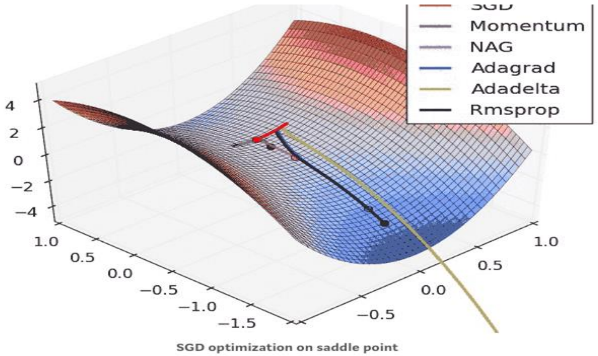
The network structure

The loss function

The optimizer



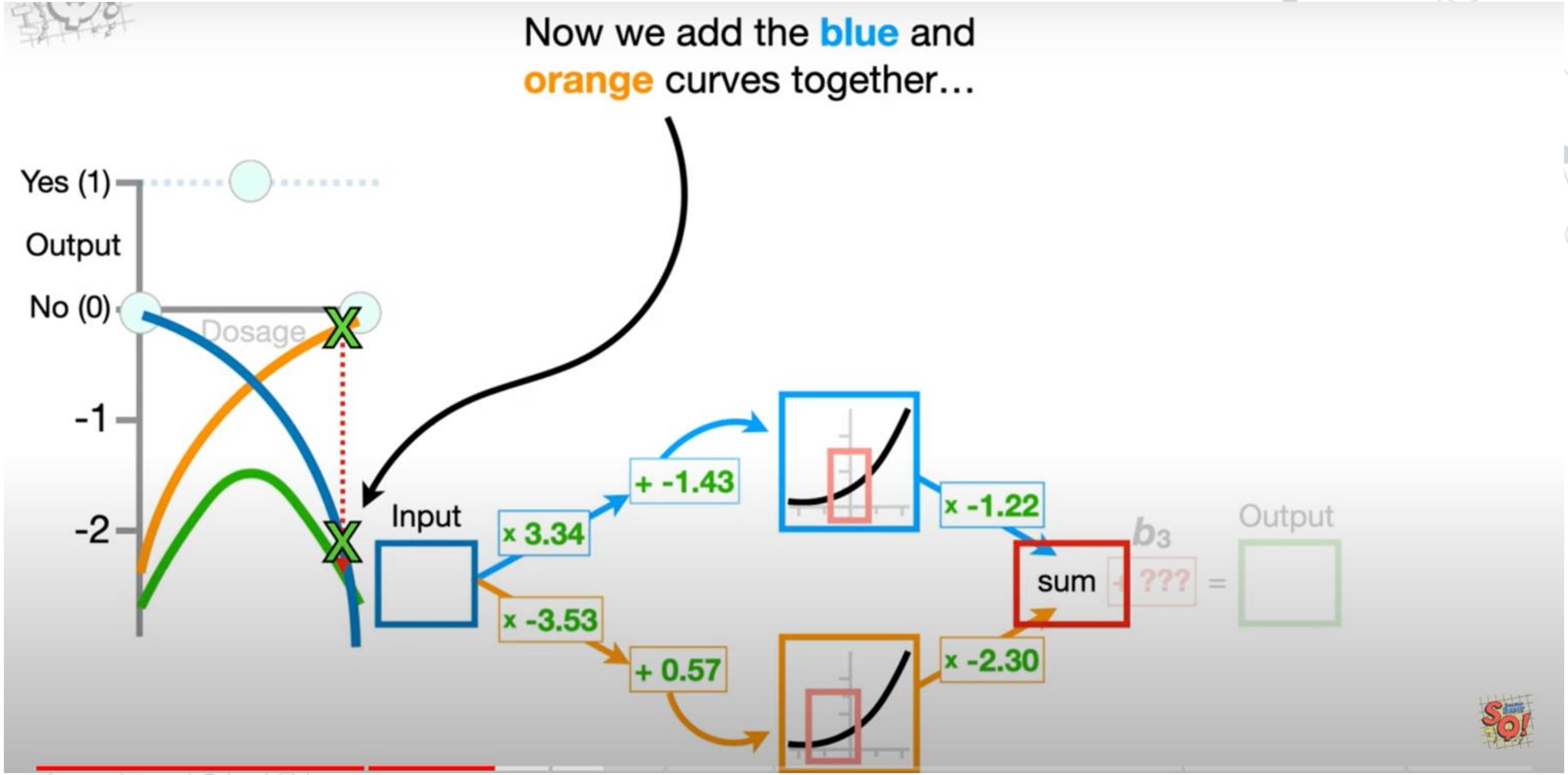
$$-y_i * \log(p_i) - (1 - y_i) * \log(1 - p_i)$$



https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

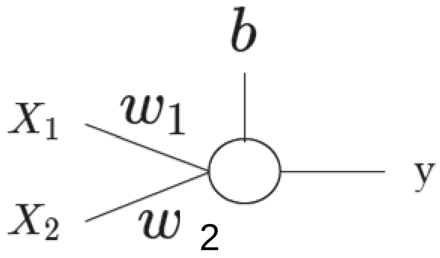
Perceptrons



Now we add the blue and orange curves together...
Yes (1)
Output
No (0)
Dosage
-1
-2
Input
x 3.34
x -3.53
+ -1.43
+ 0.57
x -1.22
x -2.30
sum
b3
+ ???
=
Output

Perceptrons

- Let's put this all together
- Our first network will be a single neuron that will learn a simple function

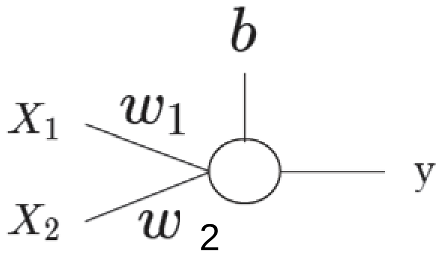


Observatic

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	1

Perceptrons

- How do we make a prediction for each observation?



Assume the following values:

w1	w2	b
1	-1	-0.5

Observatio

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	1

Predictions

- For the first observation, $X_1 = 0, X_2 = 0, y = 0$
- First compute the weighted sum:

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$h = 1 * 0 + -1 * 0 + (-0.5)$$

$$h = -0.5$$

Assume the following values:

w1	w2	b
1	-1	-0.5

Observations

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	1

Predictions

- For the first observation, $X_1 = 0, X_2 = 0, y = 0$
- First compute the weighted sum:

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$h = 1 * 0 + -1 * 0 + (-0.5)$$

$$h = -0.5$$

Transform to a probability:

$$p = \frac{1}{1+\exp(-h)}$$

$$p = \frac{1}{1+\exp(-0.5)}$$

$$p = 0.38$$

Assume the following values:

w1	w2	b
1	-1	-0.5

*Note we are doing binary classification so we use the sigmoid activation function to calculate p

Predictions

- For the first observation, $X_1 = 0, X_2 = 0, y = 0$
- First compute the weighted sum:

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$h = 1 * 0 + -1 * 0 + (-0.5)$$

$$h = -0.5$$

Transform to a probability:

$$p = \frac{1}{1+\exp(-h)}$$

$$p = \frac{1}{1+\exp(-0.5)}$$

$$p = 0.38$$

Round to get prediction:

$$\hat{y} = \text{round}(p)$$

$$\hat{y} = 0$$

Assume the following values:

w1	w2	b
1	-1	-0.5

Predictions

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

$$\hat{y} = \text{round}(p)$$

Assume the following values:

w1	w2	b
1	-1	-0.5

Complete the table:

X1	X2	y	h	p	$\hat{y}$
0	0	0	-0.5	0.38	0
0	1	1			
1	0	1			
1	1	1			

Predictions

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

$$\hat{y} = \text{round}(p)$$

Assume the following values:

w1	w2	b
1	-1	-0.5

Complete the table:

X1	X2	y	h	p	$\hat{y}$
0	0	0	-0.5	0.38	0
0	1	1	-1.5	0.18	0
1	0	1	0.5	0.38	0
1	1	1	-0.5	0.38	0

Performance

- Our network isn't so great
- How do we make it better?
- What does *better* mean?
  - Need to define a measure of performance
  - There are many ways

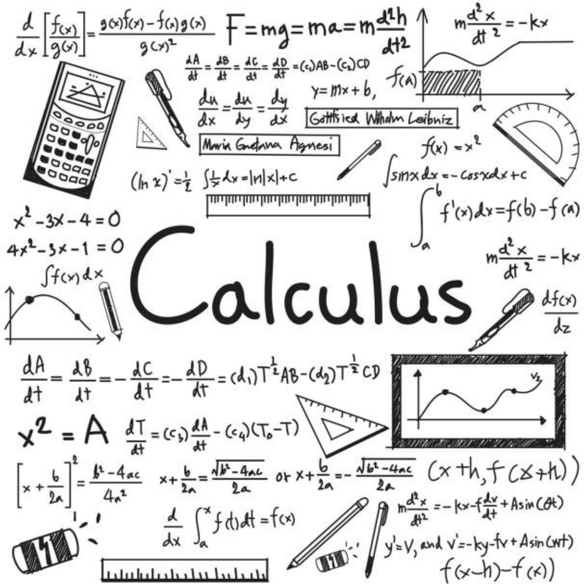
- Let's begin with squared error: $(y - p)^2$
- We need to find values for w_1, w_2, b that make this error as small as possible.
- We need to **learn** values for w_1, w_2, b such that the difference between the predicted and actual values is as small as possible.

Learning From Data

How do we find the best values for w_1, w_2, b ?

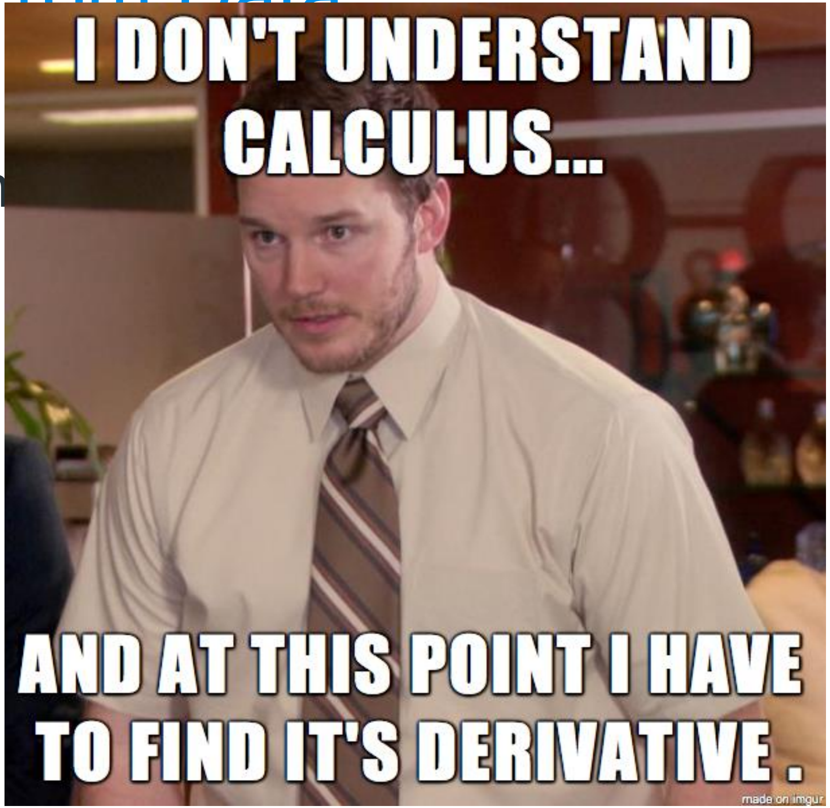
Learning From Data

How do we find the best values for w_1, w_2, b ?



Learning From Data

How do we fin

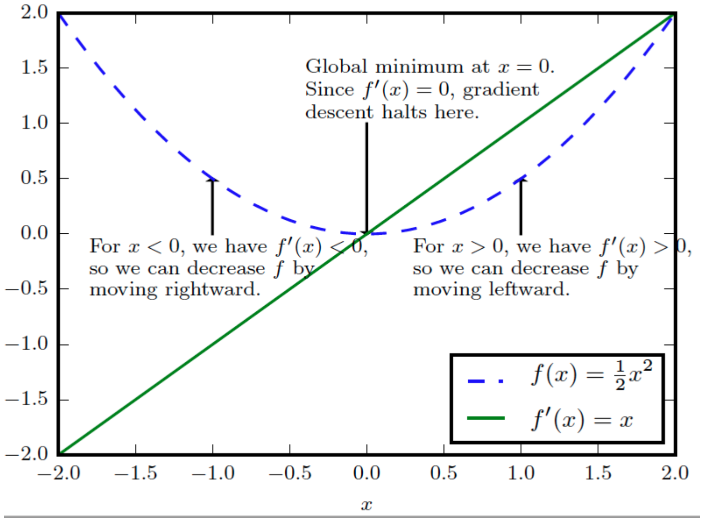


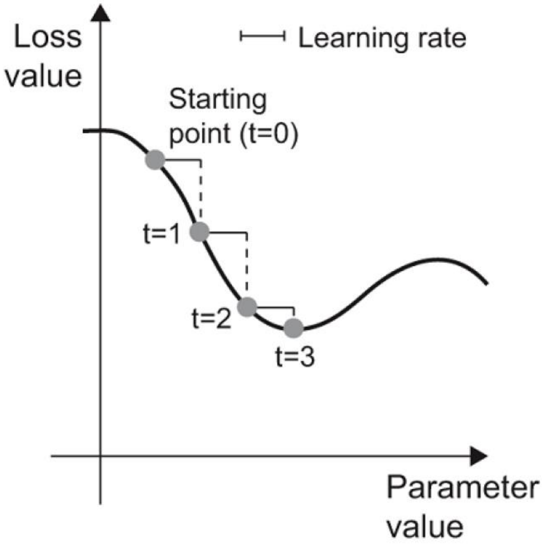
Learning From Data

- Recall that the derivative of a function tells you how it is changing at any given location.
  - If the derivative is positive, it means it’s going up
  - If the derivative is negative, it means it’s going down

- Strategy:
  - Start with initial values for w_1, w_2, b
  - Take partial derivatives of the loss function with respect to w_1, w_2, b
  - Subtract the derivative (also called the **gradient**) from each
  - This is known as **gradient descent**

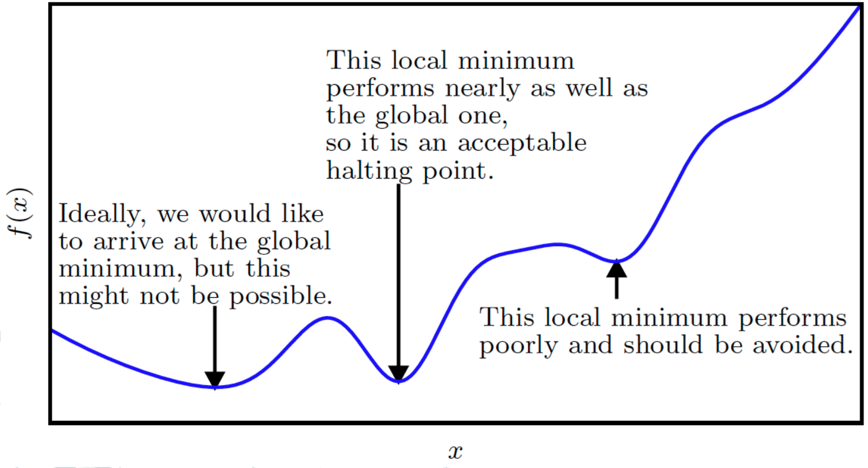
Gradient-Based Optimization

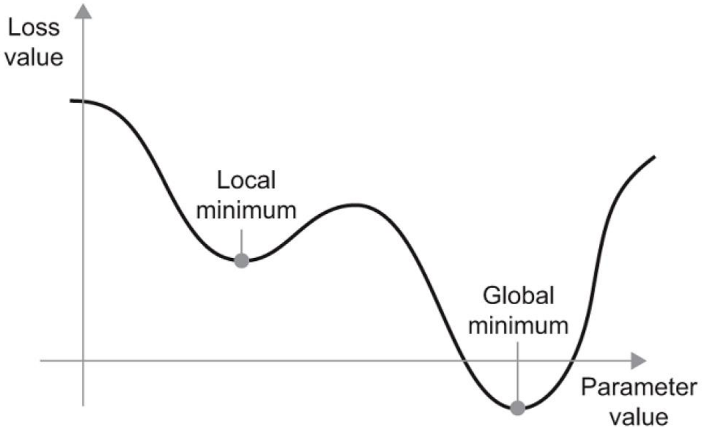




Gradient-Based Optimization

- A point that obtains the absolute lowest value of *f(x)* is a global minimum
- There may be one global minimum or multiple global minima
- It is also possible for there to be local minima that are not globally optimal
- It is common in many settings to settle for a value *f* that is very low but not necessarily minimal





Gradient-Based Optimization

- To minimize *f*, we would like to find the direction in which *f* decreases the fastest
- It can be shown that the gradient points directly uphill and the negative gradient directly downhill
- We can therefore decrease *f* by moving in the direction of the negative gradient
- For example, for a weight w_i

$$w_i^{\text{new}} = w_i^{\text{old}} - \eta g$$

where η is the **learning rate** (how fast you want to move down the gradient), and g is the gradient

The Backpropagation Algorithm

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

The Backpropagation Algorithm

- Our perceptron performs the following computations:

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

- We want to minimize this quantity:

$$l = (y - p)^2$$

- We'll compute the gradients for each parameter by "backpropagating" errors through each component of the network

The Backpropagation Algorithm

For w_1 we need to compute

$$\frac{\partial l}{\partial w_1}$$

To get there we will use the chain rule

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

This is “backprop”

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$l = (y - p)^2$$

The Backpropagation Algorithm

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

$$\boxed{\frac{\partial l}{\partial p}} =$$

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$\boxed{l = (y - p)^2}$$

The Backpropagation Algorithm

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

$$\boxed{\frac{\partial l}{\partial p}} = 2 * (p - y)$$

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$\boxed{l = (y - p)^2}$$

The Backpropagation Algorithm

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

$$\frac{\partial l}{\partial p} = 2 * (p - y)$$

$$\frac{\partial p}{\partial h} =$$

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$l = (y - p)^2$$

The Backpropagation Algorithm

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

$$\frac{\partial l}{\partial p} = 2 * (p - y)$$

$$\frac{\partial p}{\partial h} = p * (1 - p)$$

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$l = (y - p)^2$$

The Backpropagation Algorithm

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

$$\frac{\partial l}{\partial p} = 2 * (p - y)$$

$$\frac{\partial p}{\partial h} = p * (1 - p)$$

$$\frac{\partial h}{\partial w_1} =$$

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$l = (y - p)^2$$

The Backpropagation Algorithm

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial h} * \frac{\partial h}{\partial w_1}$$

$$\frac{\partial l}{\partial p} = 2 * (p - y)$$

$$\frac{\partial p}{\partial h} = p * (1 - p)$$

$$\frac{\partial h}{\partial w_1} = X_1$$

Computations

$$h = w_1 * X_1 + w_2 * X_2 + b$$

$$p = \frac{1}{1+\exp(-h)}$$

Loss

$$l = (y - p)^2$$

Putting it all together:

$$\frac{\partial l}{\partial w_1} = 2 * (p - y) * p * (1 - p) * X_1$$

Gradient Descent with Backprop

For some number of iterations:

1. Compute the gradient for w i
2. Update $w_i^{\text{new}} = w_i^{\text{old}} - \eta g$
3. Repeat until "convergence"

Do this for each weight and bias term.

Multilayer Perceptrons

Perceptron → MLP

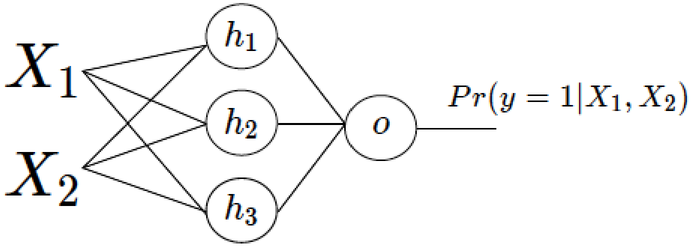
We can turn our perceptron model into a multilayer perceptron

- Instead of just one linear combination, we are going to take several, each with a different set of weights
- Each linear combination will be followed by a nonlinear activation
- Each of these nonlinear features will be fed into the logistic regression classifier (binary classifier)
- All of the weights are learned end-to-end via SGD

MLPs learn a set of nonlinear features directly from data - “feature engineering” is the hallmark of deep learning approaches

Multilayer Perceptrons (MLPs)

Suppose we have the following MLP with 1 hidden layer that has 3 hidden units:



Each neuron in the hidden layer is going to do exactly the same thing as before.

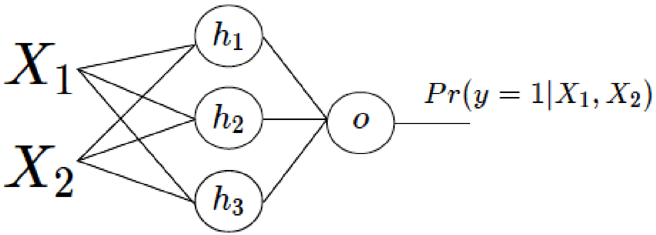
Multilayer Perceptrons (MLPs)

Computations:

$$h_j = \phi(w_{1j} * X_1 + w_{2j} * X_2 + b_j)$$

$$o = b_o + \sum_{j=1}^{3} w_{oj} * h_j$$

$$p = \frac{1}{1+\exp(-o)}$$



*If we use a sigmoid activation function

Output layer weight derivatives

$$\frac{\partial l}{\partial w_{oj}} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial o} * \frac{\partial o}{\partial w_{oj}}$$

$$= (p - y) * p * (1 - p) * h_j$$

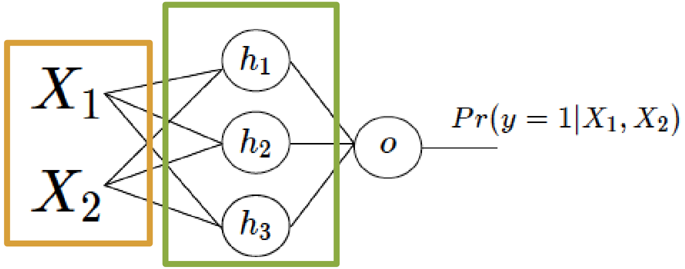
Hidden layer weight derivatives

$$\frac{\partial l}{\partial w_{1j}} = \frac{\partial l}{\partial p} * \frac{\partial p}{\partial o} * \frac{\partial o}{\partial h} * \frac{\partial h}{\partial w_{1j}}$$

$$= (p - y) * p * (1 - p) * h_j * (1 - h_j) * X_1$$

Matrix Notation

Sum notation starts to get unwieldy quickly. We can use matrix notation to represent each calculation in a more concise way.



$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \quad W = \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$Z = W^T X + B \qquad H = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \phi(Z)$$

Notation

As the number of layers grows, the number of matrices grows and we have to add a superscript to denote the layer. We also have to add a superscript to denote which training example we are referencing.

Example notation for 1 weight in 1 hidden layer for 1 training example:

$$w_{j,k}^{[l](i)}$$

Layer *l*

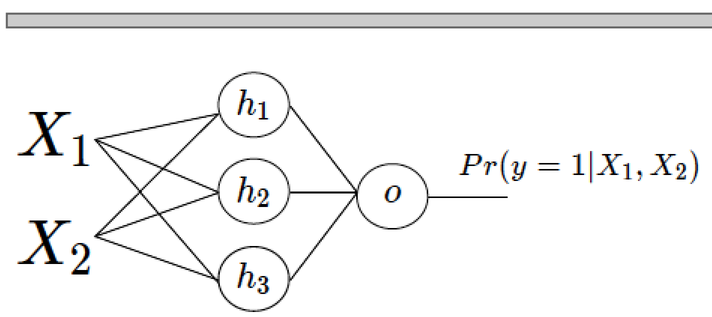
Training example *i*

Hidden neuron *k*

Input feature *j*

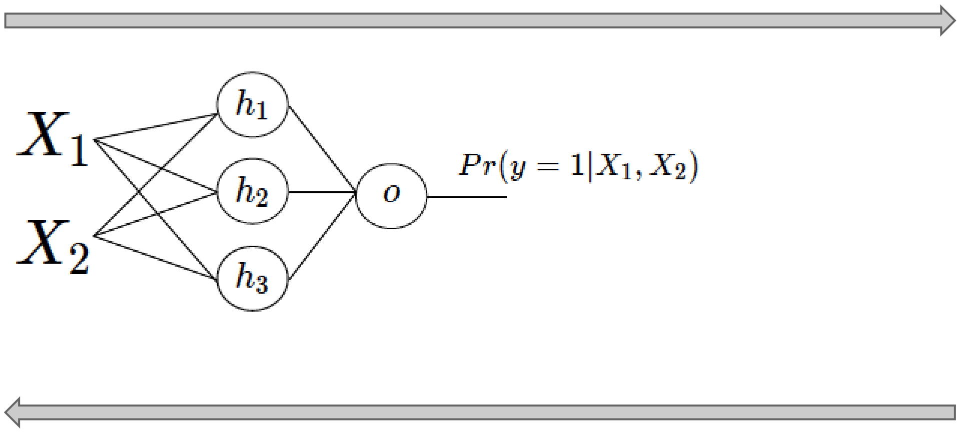
MLP Terminology

Forward pass = computing probability from input



MLP Terminology

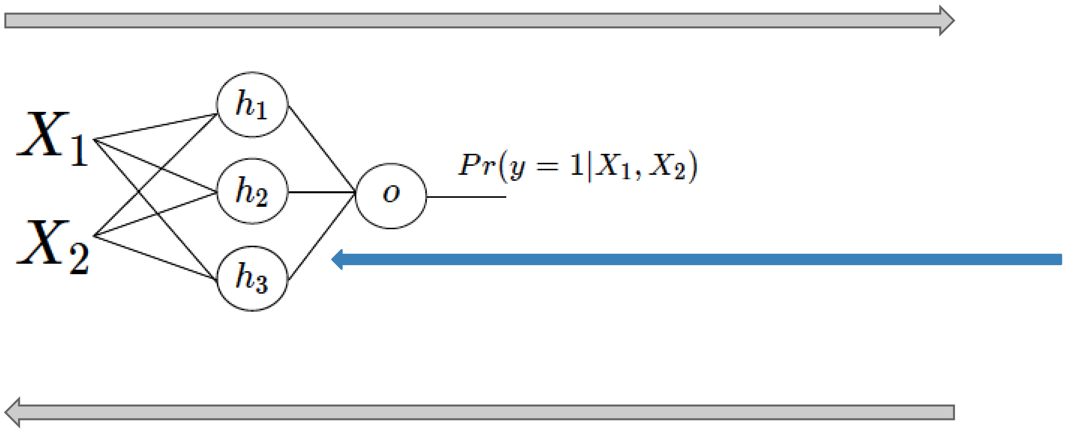
Forward pass = computing probability from input



Backward pass = computing derivatives from the output

MLP Terminology

Forward pass = computing probability from input

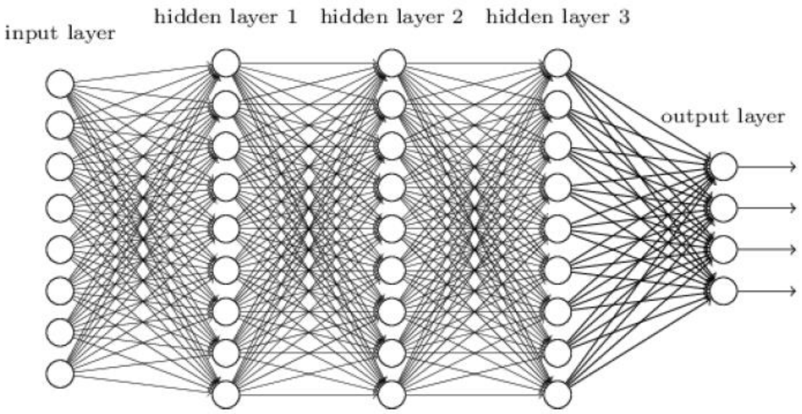


Hidden layers are also called "dense" layers or "fully connected" layers

Backward pass = computing derivatives from the output

MLPs

Increasing the number of layers increases the flexibility of the model - but run the risk of overfitting



Conclusions

- Backprop, perceptrons, and MLPs are the building blocks of neural nets

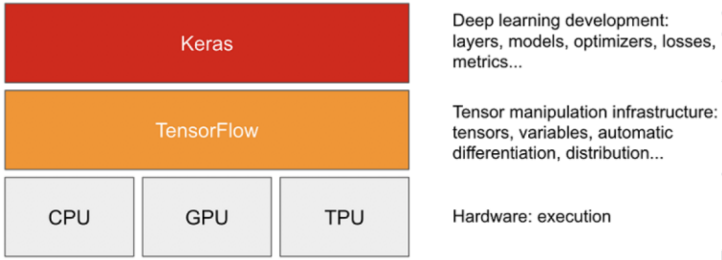
- You'll get a chance to demonstrate your mastery in Problem Set 1

- We will use these concepts for the rest of the semester

This is where we left it last class

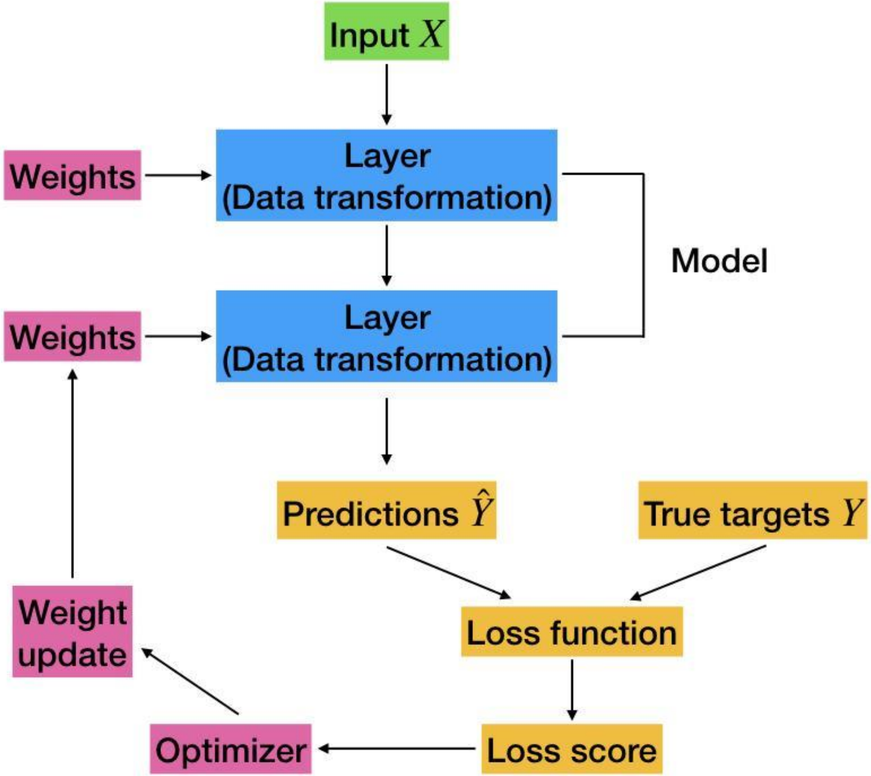
Coding Neural Nets

Keras and Tensorflow



- Keras is a model-level library that provides high-level building blocks for developing deep learning models
- It doesn't handle low-level operations like matrix and tensor (n-dimensional matrix) multiplication and differentiation
  - It uses TensorFlow or Theano or CNTK (Microsoft Cognitive Toolkit) backends for this
  - We will be using TensorFlow
    - It is the most widely adopted, scalable and production ready
- Keras can run on both CPUs and GPUs
  - When running on CPUs, uses Eigen for tensor operations
  - When running on GPUs, uses the NVIDIA CUDA Deep Neural Network library (cuDNN)

Neural Network Workflow



Generic Feedforward Network

Elements needed:

1. Necessary libraries
2. Dataset split into training and test sets (validation as well if you have enough data)
3. `models.Sequential()`: defines a linear, or sequential architecture made up of a set of layers that will stack to create the network
4. `layers.Dense()`: specifies a fully connected layer
5. `model.compile(optimizer, loss, metrics)`: specifies how to execute the training of the network
6. `model.fit(train_data, train_labels, epochs, batch_size)`: fits the neural net using the training data, runs for a specified number of iterations (epochs) using batch_size number of training examples at a time

Generic Feedforward Network

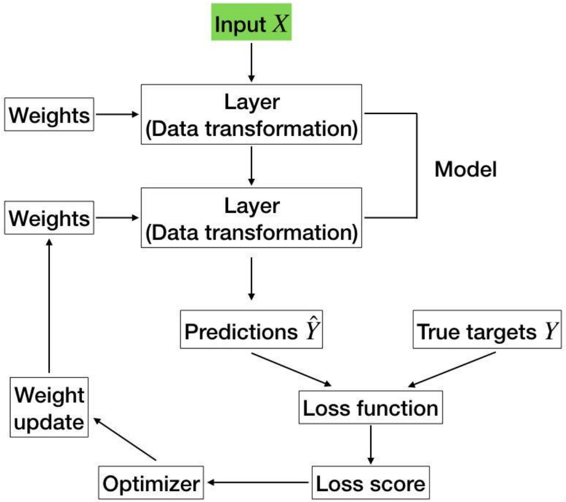
```
# Import needed packages (not an exhaustive list)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load data (will most likely be more complicated)
(x_train, y_train), (x_test, y_test) = load_data()

# Define model architecture
model = keras.Sequential([
  # Layer 1 (Hidden layer, fully connected)
  layers.Dense(c, activation='activation function'),
  # Layer 2 (Output layer, fully connected)
  layers.Dense(d, activation='output activation function')
])

# Define how to execute training
model.compile(optimizer='optimizing algorithm',
              loss='loss function',
              metrics=['performance metric'])

# Train the network
model.fit(x_train, y_train, epochs = e, batch_size = b)
```



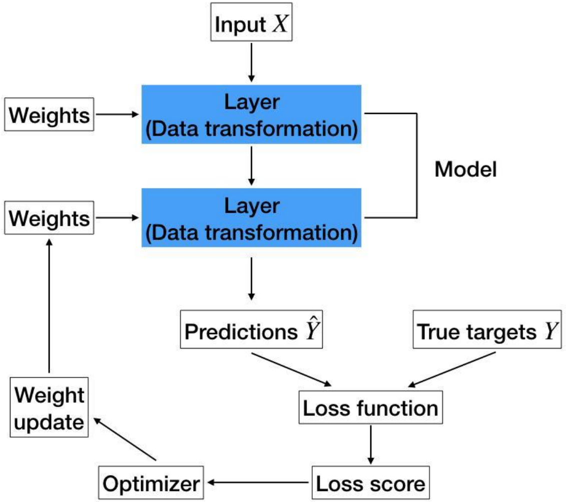

```python
# Import needed packages (not an exhaustive list)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load data (will most likely be more complicated)
(x_train, y_train), (x_test, y_test) = load_data()

# Define model architecture
model = keras.Sequential([
  # Layer 1 (Hidden layer, fully connected)
  layers.Dense(c, activation='activation function'),
  # Layer 2 (Output layer, fully connected)
  layers.Dense(d, activation='output activation function')
])

# Define how to execute training
model.compile(optimizer='optimizing algorithm',
              loss='loss function',
              metrics=['performance metric'])

# Train the network
model.fit(x_train, y_train, epochs = e, batch_size = b)
```



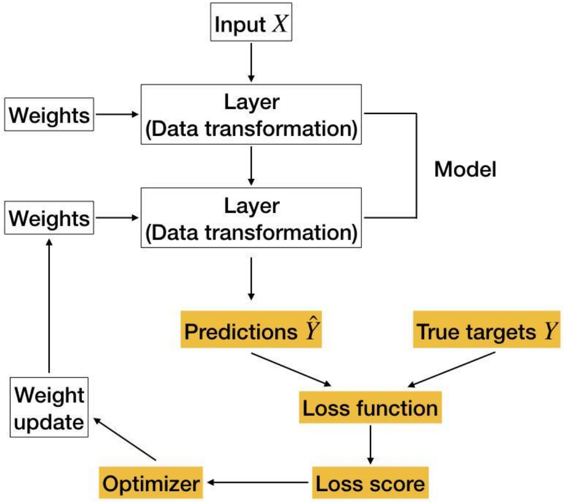

```python
# Import needed packages (not an exhaustive list)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load data (will most likely be more complicated)
(x_train, y_train), (x_test, y_test) = load_data()

# Define model architecture
model = keras.Sequential([
  # Layer 1 (Hidden layer, fully connected)
  layers.Dense(c, activation='activation function'),
  # Layer 2 (Output layer, fully connected)
  layers.Dense(d, activation='output activation function')
])

# Define how to execute training
model.compile(optimizer='optimizing algorithm',
              loss='loss function',
              metrics=['performance metric'])

# Train the network
model.fit(x_train, y_train, epochs = e, batch_size = b)
```



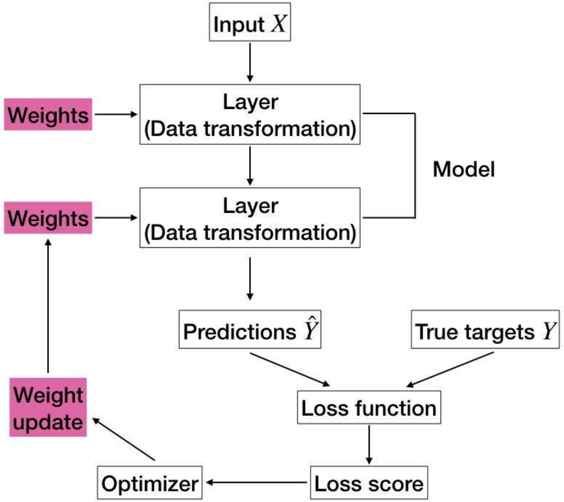

```python
# Import needed packages (not an exhaustive list)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load data (will most likely be more complicated)
(x_train, y_train), (x_test, y_test) = load_data()

# Define model architecture
model = keras.Sequential([
  # Layer 1 (Hidden layer, fully connected)
  layers.Dense(c, activation='activation function'),
  # Layer 2 (Output layer, fully connected)
  layers.Dense(d, activation='output activation function')
])

# Define how to execute training
model.compile(optimizer='optimizing algorithm',
              loss='loss function',
              metrics=['performance metric'])

# Train the network
model.fit(x_train, y_train, epochs = e, batch_size = b)
```




```python
# Import needed packages (not an exhaustive list)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load data (will most likely be more complicated)
(x_train, y_train), (x_test, y_test) = load_data()

# Define model architecture
model = keras.Sequential([
  # Layer 1 (Hidden layer, fully connected)
  layers.Dense(c, activation='activation function'),
  # Layer 2 (Output layer, fully connected)
  layers.Dense(d, activation='output activation function')
])

# Define how to execute training
model.compile(optimizer='optimizing algorithm',
              loss='loss function',
              metrics=['performance metric'])

# Train the network
model.fit(x_train, y_train, epochs = e, batch_size = b)
```

Generic Feedforward Network

train_data: training examples (matrix of feature vectors; $\mathbf{X}_{train}$)

train_labels: training labels ($\mathbf{y}_{train}$)

test_data: test examples used to measure performance of network ($\mathbf{X}_{test}$)

test_labels: test set labels ($\mathbf{y}_{test}$)

Optimizing algorithms: rmsprop, sgd, adagrad, adam, etc.

Loss function options: mse, mae, categorical_crossentropy, etc.

Performance measure options: accuracy, mae, etc.

Here:

c = the number of hidden units (neurons) in a hidden layer

d = the number of units (neurons) in the output layer

e = the number of epochs (iterations) over entire training data set

b = the batch size (how many training examples to optimize at once)

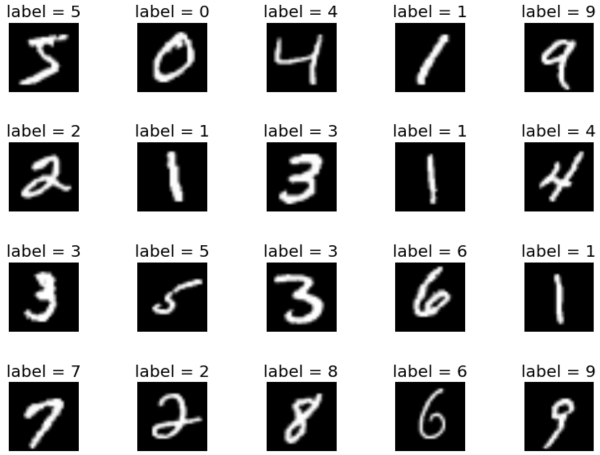
MLPs in Python/Keras

MNIST Data Example

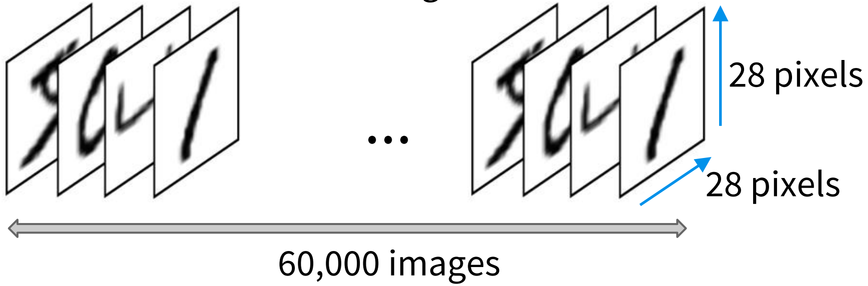
- The [MNIST data set]() includes handwritten digits with corresponding labels

- Training set: 60,000 images of handwritten digits and corresponding labels
  - Each digit is represented as a 28 x 28 matrix of grayscale values 0 - 255
  - The entire training set is stored in a 3D tensor of shape (60000, 28, 28)
  - The corresponding image values are stored as a 1D tensor of values 0 - 9

- Testing set: 10,000 images with the same set up as the training set



Training Data



MNIST Data Example

Data wrangling

- We'll get into RGB images later, but for grayscale images, we need to first transform the matrix of values into a vector of values, and then normalize them to be between 0 and 1. It is not strictly necessary to normalize your inputs, but smaller numbers help speed up training and avoid getting stuck in local minima. This also ensures the gradients don't "explode" or "vanish"
    - Reshape each image from a 28 x 28 matrix of grayscale values 0 - 255 to a vector of length 28*28 = 784 of values 0 - 1 (divide each by 255)

- We now have 10 classes (categories; the digits 0-9)
    - We need to have multiclass labels that tell the network which digit the example is
    - Reshape each corresponding image label to a vector of length 10 of values 0 or 1
    - Example: the digit 3 would be represented as [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
    - You can think of this as "dummy coding" the labels

Activation and Loss Function Choices

Task	Last-layer activation	Loss function
Binary classification	sigmoid	Binary cross-entropy
Multiclass, single-label classification	softmax	Categorical cross-entropy
Multiclass, multilabel classification	sigmoid	Binary cross-entropy
Regression to arbitrary values	None	Mean square error (MSE)
Regression to values between 0 and 1	sigmoid	MSE or binary cross-entropy

Softmax function

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- Softmax units are used as outputs when predicting a discrete variable *y* with *j* possible values
- In this setting, which can be seen as a generalization of the Bernoulli distribution, we need to produce a vector $\hat{\mathbf{y}}$ with $\hat{y}_i = P(y = i|x)$
- We require that each $\hat{y}_i$ lie in the [0, 1] interval and that the entire vector sums to 1
- We first compute $z = w^T x + b$ as usual
- Here, $z_i = log[\tilde{P}(y = i|x)]$ represents an unnormalized log probability for class *i*
- The softmax function then exponentiates and normalizes *z* to obtain $\hat{\mathbf{y}}$

Categorical cross-entropy

- In this case we want to maximize

$$log[P(y=i;z)] = log[\text{softmax}(z)_i] = z_i - log\sum_j exp(z_j)$$

- The first term shows that the input always has a direct contribution to the loss function

- Because $log\sum_j exp(z_j) \approx max_j z_j$, the negative log-likelihood loss function always strongly penalizes the most active incorrect prediction

MNIST Data Example

Network Architecture

- Let’s start with 2 layers:
  - Hidden layer will have 512 hidden units and the **relu activation function**

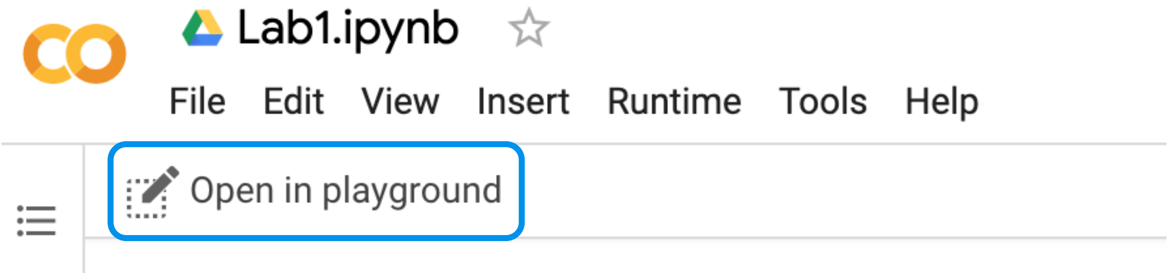
  - Output layer with 10 units (one for each possible digit) and the **softmax activation function** (this produces a vector of length 10, where each element is a probability between 0 and 1 of the image being classified as that digit)
  - Example: [0, 0.3, 0, 0, 0, 0, 0, 0.7, 0, 0] - the highest probability corresponds to a label of 7, so the network would classify this image as a 7

  - **rmsprop optimization algorithm**
  - **categorical_crossentropy loss function**
  - **accuracy performance measure** (the proportion of times the correct class is chosen)

MNIST Data Example

[Colab link]

Step 1



Step 2