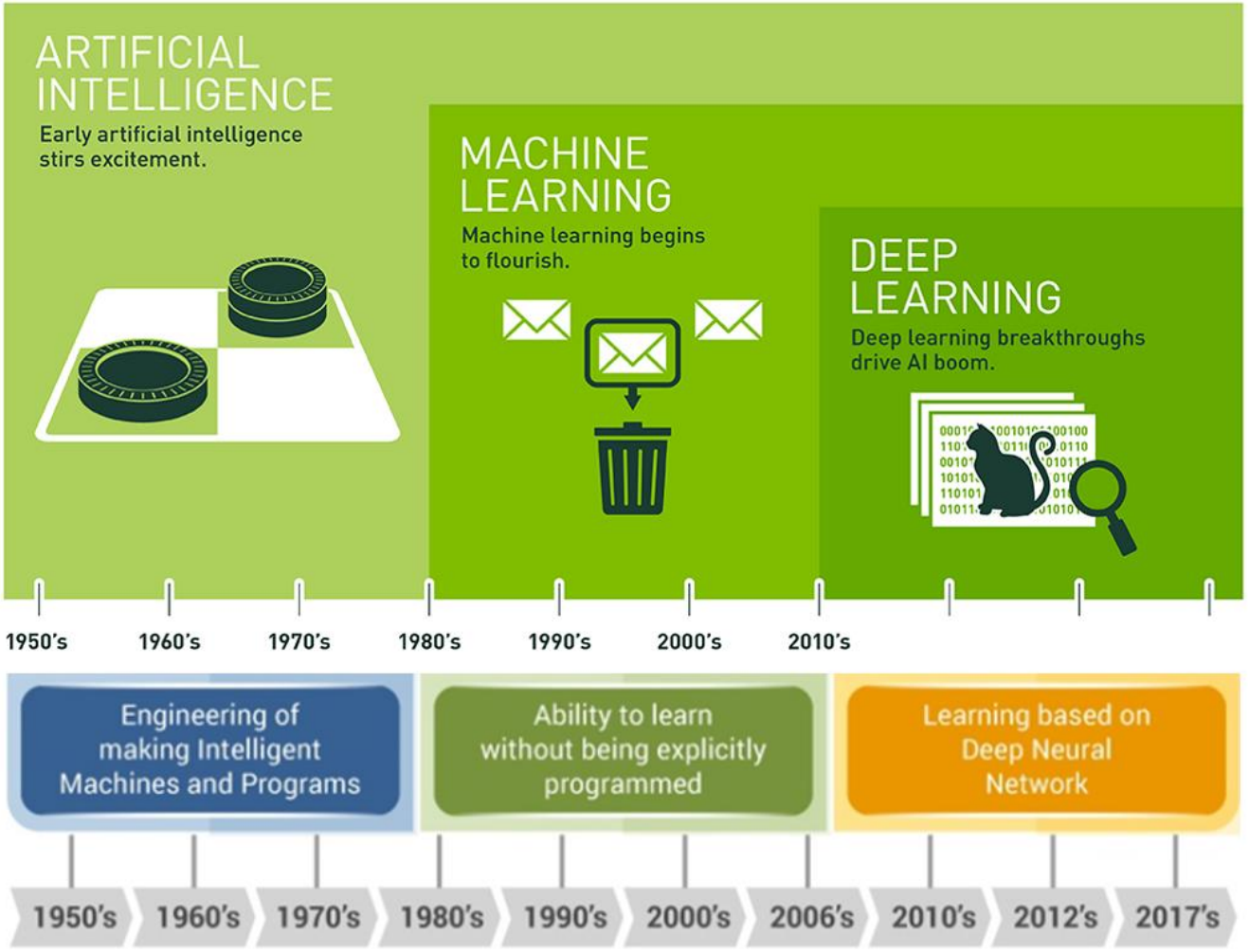
IFA-305 Sistem Cerdas (Intelligent System)
Lecture 9-12

Introduction to Deep Learning

Nur Uddin, PhD.

Program Studi Informatika
Universitas Pembangunan Jaya
Tangerang Selatan

AI Evolution (#1)



Scope

- You're about to learn all you need to get started building your own deep neural networks.

- Using Keras and Tensorflow you'll learn how to:
  - create a fully-connected neural network architecture
  - apply neural nets to two classic ML problems: regression and classification
  - train neural nets with stochastic gradient descent, and
  - improve performance with dropout, batch normalization, and other techniques

What is Deep Learning?

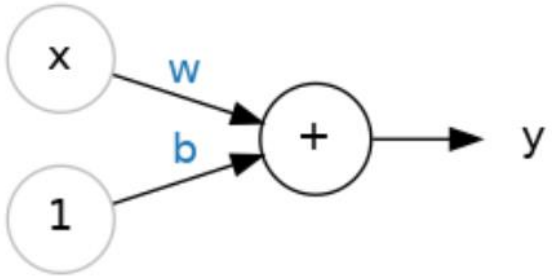
- Some of the most impressive advances in artificial intelligence in recent years have been in the field of deep learning.
- Natural language translation, image recognition, and game playing are all tasks where deep learning models have neared or even exceeded human-level performance.
- Deep learning is an approach to machine learning characterized by deep stacks of computations. This depth of computation is what has enabled deep learning models to disentangle the kinds of complex and hierarchical patterns found in the most challenging real-world datasets.

Neural Networks and Deep Learning

- Through their power and scalability neural networks have become the defining model of deep learning.
- Neural networks are composed of neurons, where each neuron individually performs only a simple computation.
- The power of a neural network comes instead from the complexity of the connections these neurons can form.

The Linear Unit

- So let's begin with the fundamental component of a neural network: the individual neuron. As a diagram, a neuron (or unit) with one input looks like:

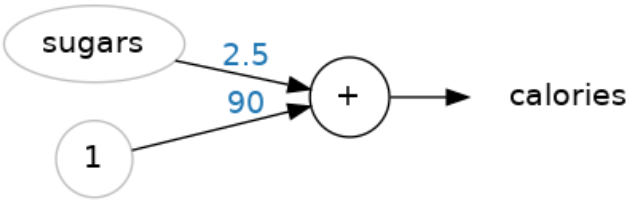


The Linear Unit: $y = wx + b$

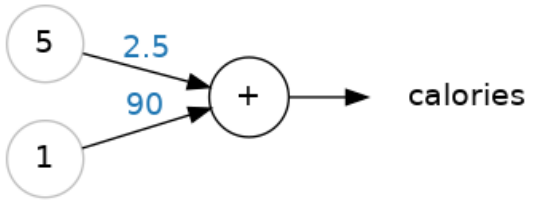
- Though individual neurons will usually only function as part of a larger network, it's often useful to start with a single neuron model as a baseline. Single neuron models are linear models.

Example: The Linear Unit as a Model

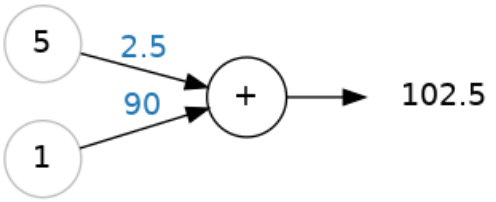
- Let's think about how this might work on a dataset like 80 Cereals. Training a model with 'sugars' (grams of sugars per serving) as input and 'calories' (calories per serving) as output, we might find the bias is b=90 and the weight is w=2.5. We could estimate the calorie content of a cereal with 5 grams of sugar per serving like this:



w=2.5, b=90



Input sugars=5

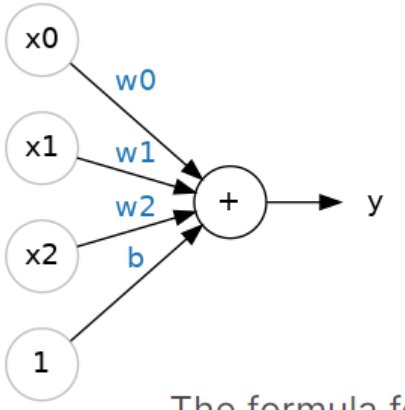


Multiply inputs by weights and sum.

calories=2.5×5+90=102.5

Multiple Inputs

- The 80 Cereals dataset has many more features than just 'sugars'. What if we wanted to expand our model to include things like fiber or protein content? That's easy enough. We can just add more input connections to the neuron, one for each additional feature. To find the output, we would multiply each input to its connection weight and then add them all together.

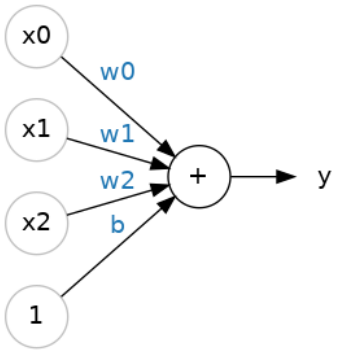


The formula for this neuron would be $y = w_0x_0 + w_1x_1 + w_2x_2 + b$. A linear unit with two inputs will fit a plane, and a unit with more inputs than that will fit a hyperplane.

Linear Unit in Keras (1)

- The easiest way to create a model in Keras is through keras.Sequential, which creates a neural network as a stack of layers. We can create models like those above using a dense layer (which we'll learn more about in the next lesson).

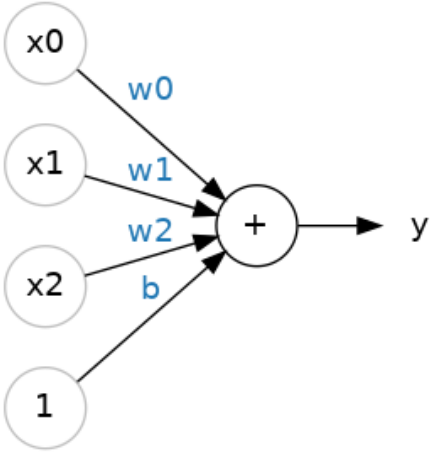
- We could define a linear model accepting three input features ('sugars', 'fiber', and 'protein') and producing a single output ('calories') like so:




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

# Create a network with 1 linear unit
model = keras.Sequential([
    layers.Dense(units=1, input_shape=[3])
])
```

Linear Unit in Keras (2)




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

# Create a network with 1 linear unit
model = keras.Sequential([
    layers.Dense(units=1, input_shape=[3])
])
```

With the first argument, `units`, we define how many outputs we want. In this case we are just predicting `'calories'`, so we'll use `units=1`.

With the second argument, `input_shape`, we tell Keras the dimensions of the inputs. Setting `input_shape=[3]` ensures the model will accept three features as input (`'sugars'`, `'fiber'`, and `'protein'`).

Part 2

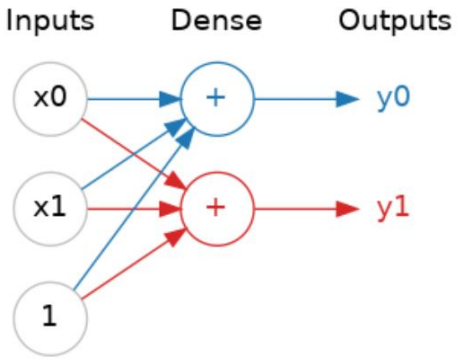
Build Neural Networks

- In this lesson we're going to see how we can build neural networks capable of learning the complex kinds of relationships deep neural nets are famous for.

- The key idea here is modularity, building up a complex network from simpler functional units. We've seen how a linear unit computes a linear function -- now we'll see how to combine and modify these single units to model more complex relationships.

Layers

- Neural networks typically organize their neurons into layers. When we collect together linear units having a common set of inputs we get a dense layer.

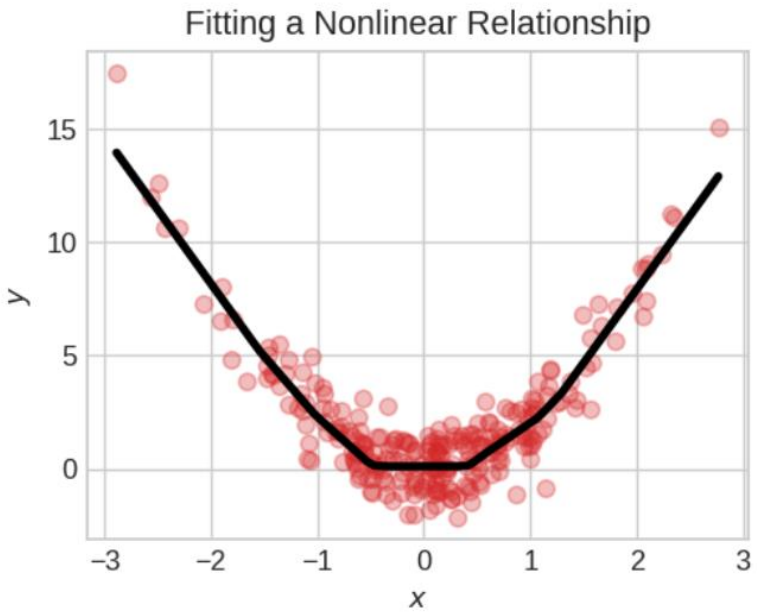


A dense layer of two linear units receiving two inputs and a bias.

- You could think of each layer in a neural network as performing some kind of relatively simple transformation. Through a deep stack of layers, a neural network can transform its inputs in more and more complex ways. In a well-trained neural network, each layer is a transformation getting us a little bit closer to a solution.

Activation Functions (1)

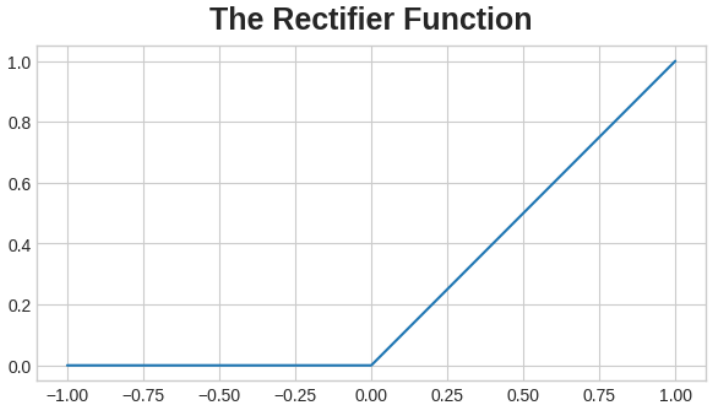
- Without activation functions, neural networks (of linear units) can only learn linear relationships. In order to fit curves, we'll need to use activation functions.



Without activation functions, neural networks can only learn linear relationships. In order to fit curves, we'll need to use activation functions.

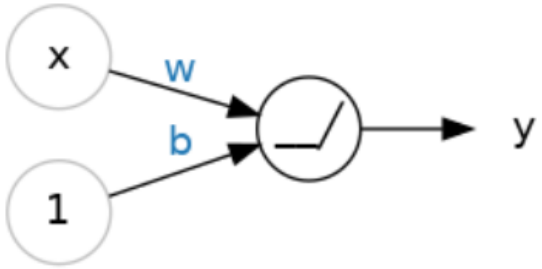
Activation Functions (2)

- An activation function is simply some function we apply to each of a layer's outputs (its activations). An example is the rectifier function max(0,x) .



ReLU

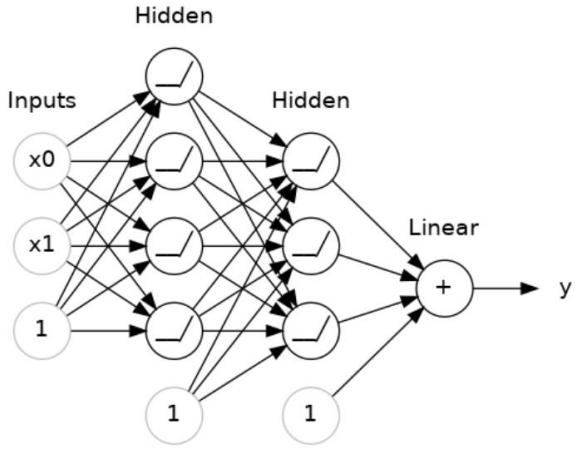
- When we attach the rectifier to a linear unit, we get a rectified linear unit or ReLU. (For this reason, it's common to call the rectifier function the "ReLU function".)
- Applying a ReLU activation to a linear unit means the output becomes max(0, w * x + b), which we might draw in a diagram like:



A rectified linear unit.

Stacking Dense Layers

- Now that we have some nonlinearity, let's see how we can stack layers to get complex data transformations.

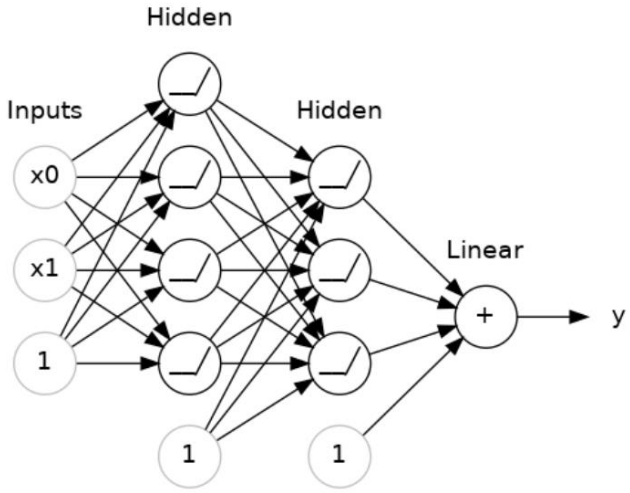


A stack of dense layers makes a "fully-connected" network.

- The layers before the output layer are sometimes called hidden since we never see their outputs directly.
- Now, notice that the final (output) layer is a linear unit (meaning, no activation function). That makes this network appropriate to a regression task, where we are trying to predict some arbitrary numeric value. Other tasks (like classification) might require an activation function on the output.

Building Sequential Models

- The Sequential model we've been using will connect together a list of layers in order from first to last: the first layer gets the input, the last layer produces the output. This creates the model in the figure above:



A stack of dense layers makes a "fully-connected" network.

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

model = keras.Sequential([
    # the hidden ReLU layers
    layers.Dense(units=4, activation='relu', input_shape=[2]),
    layers.Dense(units=3, activation='relu'),
    # the linear output layer
    layers.Dense(units=1),
])
```