

PCSE 595

Special Topics in Machine Learning

Dr. Sam Henry

samuel.henry@cnu.edu

Luter 325

Office Hours

- Please stop by!
- This is an [Open Question Answering Time](#)

Monday, Wednesday, Friday
11:00-12:00, 1:00-1:30
Or by appointment

Office hours are in-person (just come by my office, LUTR 325)



*I expect to see
everyone at office
hours at some point*

Pizza My Mind

- You can get extra credit
 - Up to two extra points on your final grade for one PCSE course
- Attend them! They're fun, informative, and employers present
 - Don't wait until you need a job or internship, go now!
- Thursdays at 12:20

... and you get free pizza!!



Students in PCSE classes can get extra credit if they attend at least 10 events. 10-11 events: 1 extra point; 12-13 events: 2 extra points.

Programming Neural Networks

- We want to program neural networks that can use your GPU
- Today a Nvidia RTX 3090 has 35 Tflops of single precision (32bit) performance which is the same as the fastest supercomputer in the world in 2002



=



Programming Neural Networks

- **Keras** is a Python API for deep learning
 - Allows you to program deep neural networks at a high level
 - Keras is built on **TensorFlow** – a lower level GPU programming Python
 - Theano is an alternative backend (but rarely used)
- **PyTorch** is an alternative to Keras and is similar in all respects
- Both Keras and PyTorch have Sequential and Functional APIs
 - Sequential can only implement sequential ANNs
 - Functional is more flexible (and not really anymore difficult)
 - **We will use the functional API**



Demo: Breast Cancer Dataset Neural Network

- Given 9 features extracted from a digitized image, classify a sample as benign or malignant
- 699 Total Samples
- 458 benign and 241 Malignant

Binary Classification Problem:
sample is either Malignant (1) or not (0)

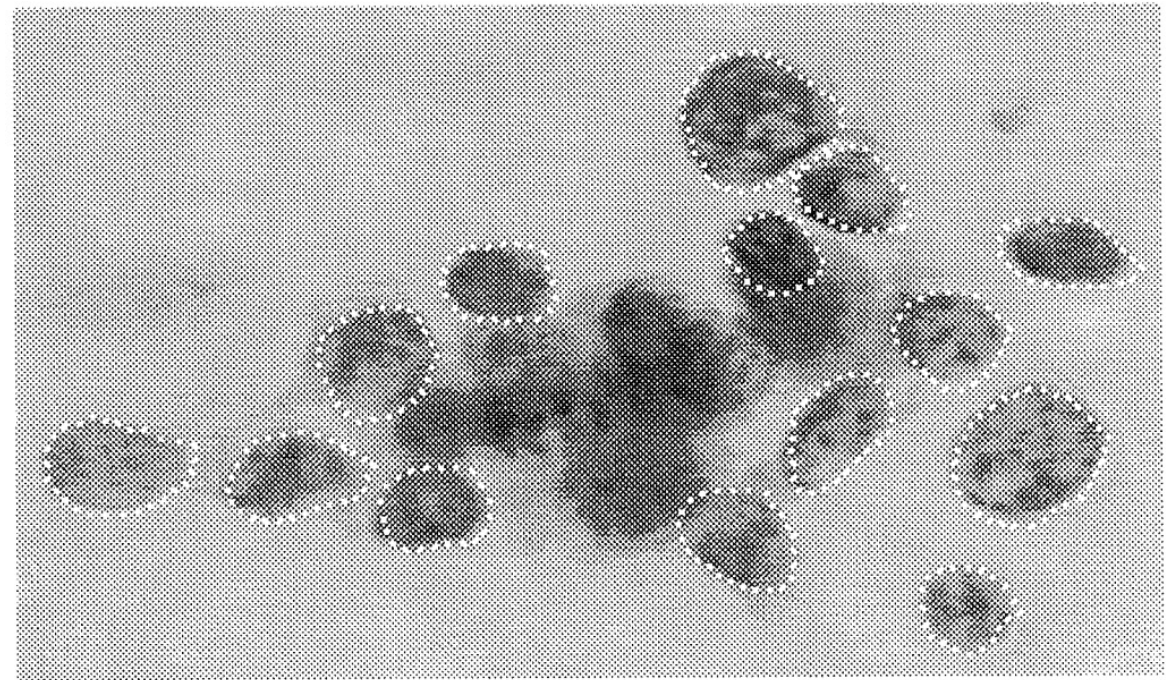
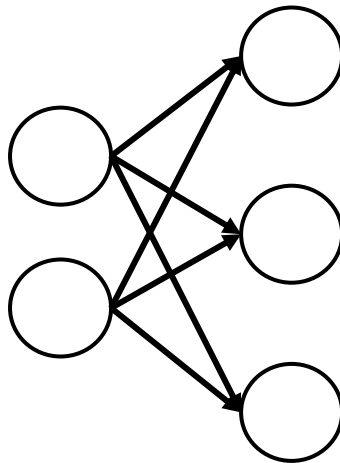


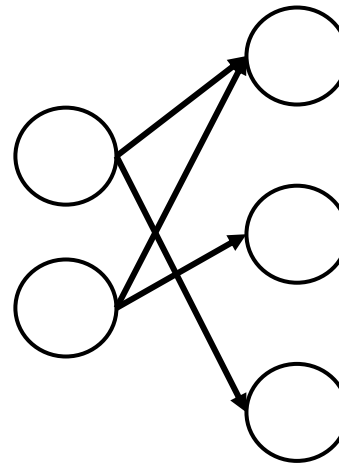
Figure 1: Initial Approximate Boundaries of Cell Nuclei

What Does Densely Connected Mean?

- A “Dense Layer” in Keras is densely connected to its input layer, meaning every input neuron is connected to each neuron in the layer.
- This is pretty standard for neural network



Dense



Not Dense

Questions

- What does “None” mean in the model summary?
- Performance increases, but then gets worse at certain points – Why?
- Final accuracy doesn't match the last reported accuracy – Why?

Equivalent using the Sequential API

Create a model defined by the sequential API

```
model = Sequential()
```

```
model.add(Dense(25, input_shape=(9,), activation=sigmoid'))
```

```
model.add(Dense(10, activation= sigmoid'))
```

```
model.add(Dense(1, activation= sigmoid'))
```

```
model.compile(optimizer='sgd',loss='binary_crossentropy',metrics=['accuracy'])
```

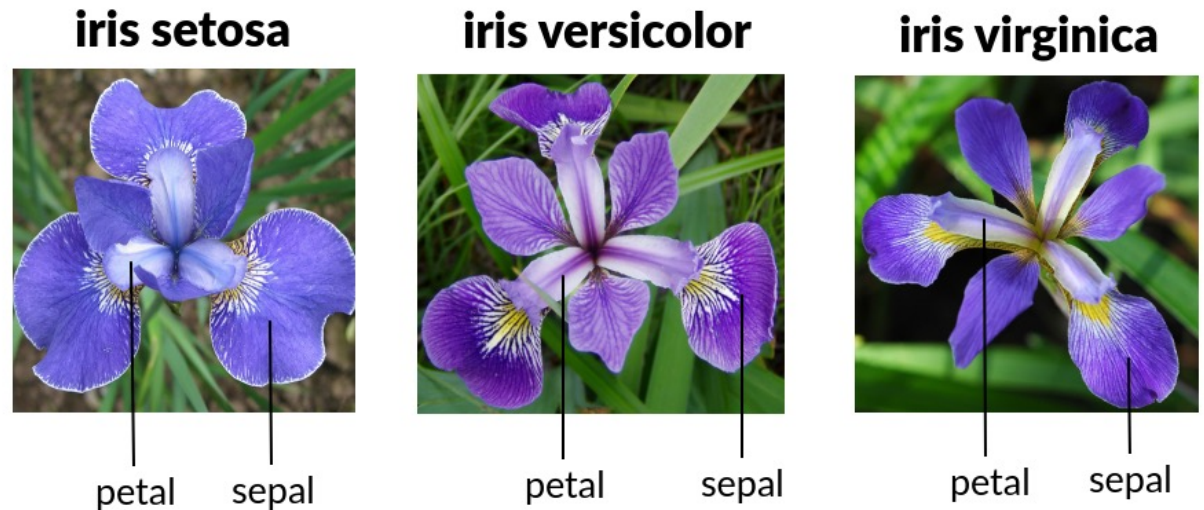
Add the input layer,
hidden layer and output
layer

Put all the pieces together, and define the optimizer,
loss, and evaluation metrics

Iris Dataset

- Given 4 features measured from flowers, classify a sample as being iris-setosa, iris-versicolor, or iris-virginica
- 150 Total Samples
- 50 of each flower type

Multi Class Classification Problem:
sample is **one** of 3 flower types
Perfectly balanced dataset




<https://archive.ics.uci.edu/dataset/53/iris>

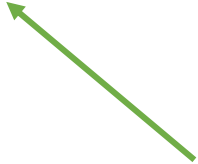
Classification Problem Types

- **Binary** – each sample is a member of exactly one class
 - Final output is a 0 or 1 indicating class membership
- **Multiclass** – each sample is a member of exactly one class
 - Final output contains exactly one “1”
- **Multilabel** – each sample is a member of zero or more labels
 - Final output contains zero or more “1’s”

Label probabilities are
dependent on each other



Label probabilities are
independent from one another



Example:

- Binary outputs a $\langle 0 \rangle$ or $\langle 1 \rangle$
- Multiclass outputs a vector, say: $\langle 0, 1, 0 \rangle$, where each index represents class membership
- Multilabel outputs a vector, say: $\langle 1, 1, 0 \rangle$, where each index represents label membership

How do we perform multiclass or multilabel classification?

- K Nearest Neighbors (KNN)
- Nearest Centroid
- Decision Tree
- Artificial Neural Network (ANN)

Discuss

Options:

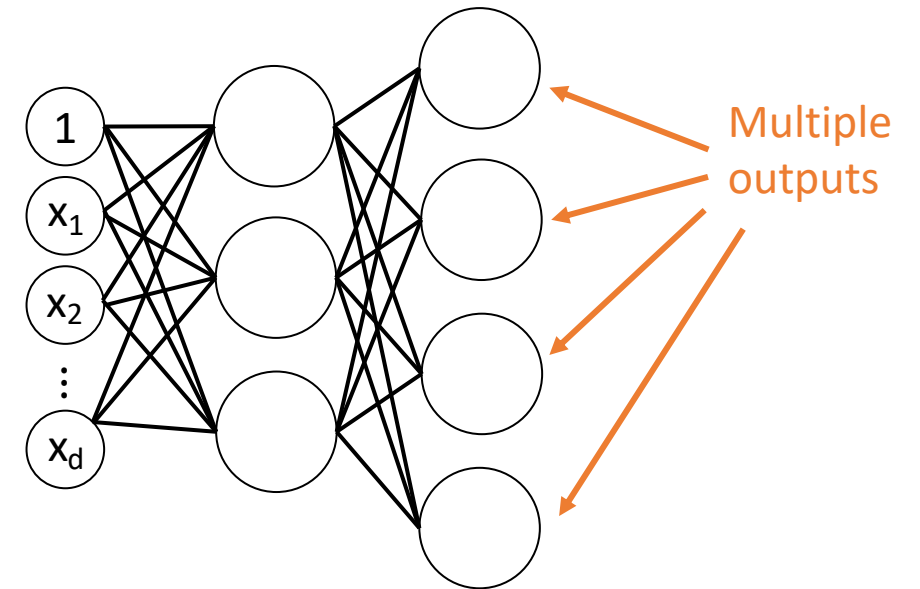
Formulate as multiple binary classifiers

Formulate directly as a multiclass classifier

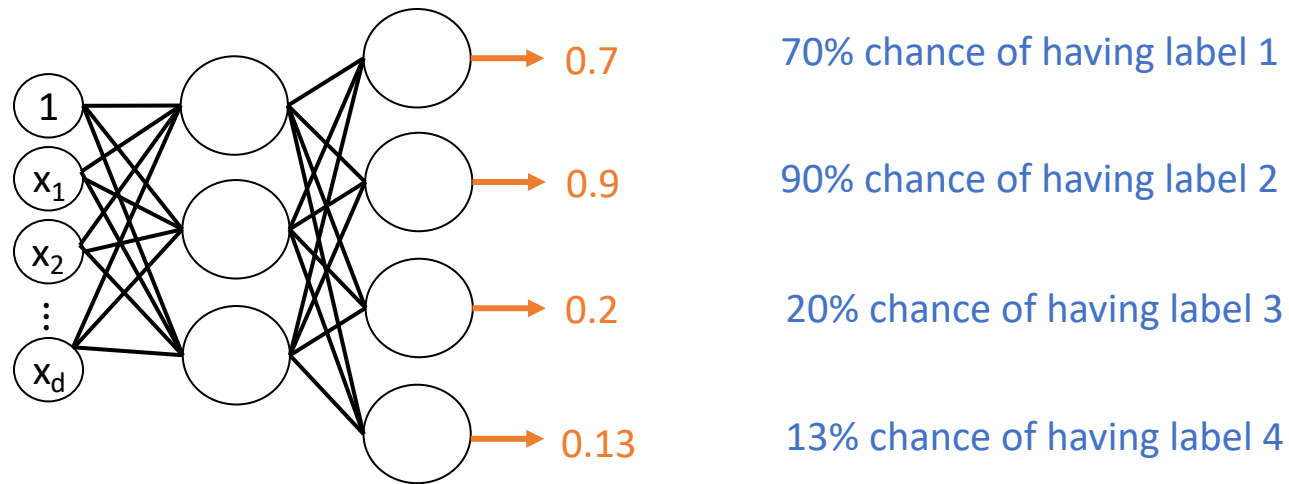
Formulate directly as a multilabel classifier

Multiclass and Multilabel ANNs

- Neural networks are flexible and can have multiple outputs
1. How do we interpret the output for multiclass and multilabel problems?
 2. How do we determine the loss for multiclass and multilabel problems?



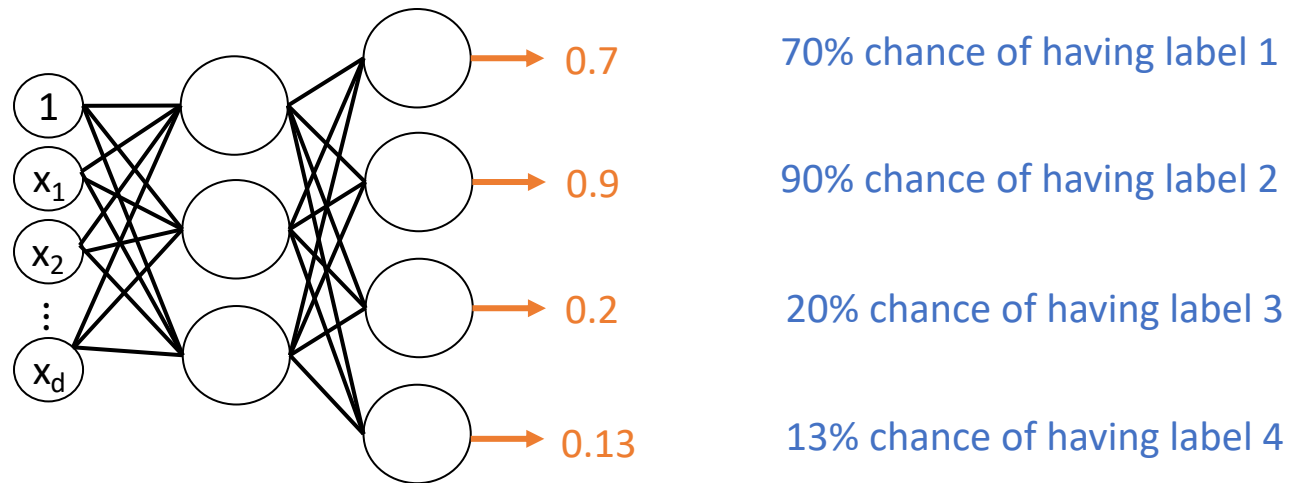
Interpreting Multilabel Output



- The output layer must have the same number of neurons as the number of labels
- Each label is assumed to be **independent**
- Each output is the probability of having that one label
- **Sigmoid (logistic) activation function** at each output node to ensure this interpretation

Round each predicted value to convert it to a label

Loss for Multilabel Problems



- Since the output is just multiple binary outputs, we use binary cross entropy (BCE) as the loss function
- The error of each output neuron (needed for back-propagation) is the BCE for that label
- We can interpret the total loss of the network as the summed BCE

Interpreting Multiclass output



- The output layer must have the same number of neurons as the number of classes
- Output is a probability distribution over all classes
- For a set of numbers to be a probability distribution:
 1. Each number must be between 0 and 1
 2. All numbers must sum to 1 (100%)
- **Softmax activation function** for each output node to ensure the output can be interpreted like this.

Take **argmax** over all predicted values to convert to a label

Softmax Activation Function

- The **softmax function** is a generalization of the logistic function to multiple classes/dimensions.
- We use the softmax activation function on the last layer of multiclass classification problems to ensure the output is a probability distribution over the number of classes

$$\text{softmax}(\vec{\hat{y}})_i = \frac{\hat{y}_i}{\sum_{j=1}^c \hat{y}_j}$$

where \hat{y} is a vector of outputs
 c is the number of classes

This is just the output divided by the sum of all outputs

Loss for Multiclass Problems



- The class label is no longer binary, so don't use binary cross entropy
- Instead, use **categorical cross entropy** as the loss function

Deriving Categorical Cross Entropy

Loss for a single sample:

actual class (y_i) times the log probability of actual class ($\log(\hat{y})$ or $\log(1 - \hat{y})$),
depending on if actual class is 0 or 1)

Binary Cross Entropy

$$y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})$$

formulated for binary problems (0 or 1 turns part of equation on or off)

Generalizing for non-binary classes, where y_i is a one-hot vector of class labels

Cross Entropy

$$-\sum_{i=1}^c y_i \ln(\hat{y}_i)$$

$y = [0,1]$ (or similar) for binary classification, indicating not class 0, but is class 1

$y = [0,0,1,0]$ (or similar) for $[0,0,1,0]$ for multiclass

In either case, result is the just the probability of it's true class (so, equivalent to BCE)

Assuming a one-hot encoding of output (so, $y_i = 0$ for all but one class)


The one nonzero class, $y_p = 1$, y_p can be removed

$$-\sum_{i=1}^c y_i \ln(\hat{y}_i) = \ln(\hat{y}_p)$$

And assuming a softmax output layer, meaning:

$$\hat{y}_p = \frac{e^{y_p}}{\sum_{i=1}^c e^{\hat{y}_i}}$$

Categorical Cross Entropy


$$-\ln \left(\frac{e^{y_p}}{\sum_{i=1}^c e^{\hat{y}_i}} \right)$$

Take Away: Cross Entropy

Cross Entropy (CE)

$$-\sum_{i=1}^c y_i \ln(\hat{y}_i)$$

Cross Entropy is sometimes called “**Log Loss**”, and is the general form for both BCE and CCE

Binary Cross Entropy (BCE)

$$y_i \ln(\hat{y}) + (1 - y_i) \ln(1 - \hat{y})$$

binary cross entropy is the specific case for binary classification

Categorical Cross Entropy (CCE)

$$-\ln \left(\frac{e^{y_p}}{\sum_{i=1}^c e^{\hat{y}_i}} \right)$$

categorical cross entropy is the specific case for one-hot encoded multiclass labels with a softmax activation function

These equations are for a single sample. The loss function is calculated over all samples. For example:

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^n CCE(\hat{y}_i, y_i)$$

Categorical Cross Entropy

Final Formulation
of Categorical Cross
Entropy

- We changed our loss function, so now we have to change how we compute the derivative

Categorical Cross Entropy

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^n \ln \left(\frac{e^{y_p}}{\sum_{i=1}^c e^{\widehat{y}_i}} \right)$$

...but, we aren't going to cover this. It doesn't change backpropagation much, just the error portion of the derivative on the last layer. Just remember, **if you change your loss function, you are changing the surface you are performing gradient descent on.**

Therefore you must correctly calculate the derivative on that surface.

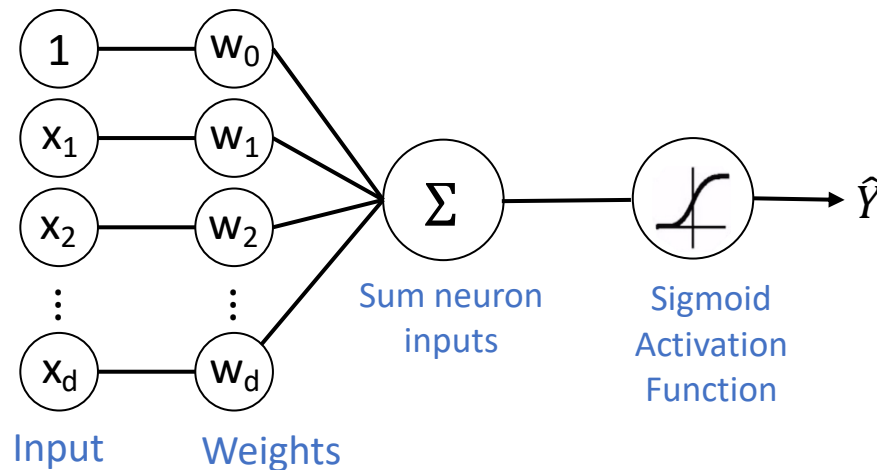
Recap: Problem Types for ANNs

- Binary Classification:
 - Output a scalar, the probability of true class
 - Use **sigmoid activation function** in output layer with **binary cross entropy** loss
 - **Round** the probability to convert to a label
- Multilabel Classification:
 - Output is a vector, the independent probabilities of each class
 - Use **sigmoid activation function** in output layer with **binary cross entropy** loss
 - **Round** each probability to convert to a vector of labels
- Multiclass Classification:
 - Output is a vector, the joint probability over all classes
 - Use **softmax activation function** in output layer with **categorical cross entropy** loss
 - **Argmax** over all probabilities to convert to a single class

Demo Basic_Iris.py

Different Activation functions

- You can use different activation functions **for hidden layers** with minimal changes to the back propagation algorithm



- Activation Functions should be differentiable for the chain rule to work
- Activation Functions should have derivatives that are easy to calculate for efficiency

Check them out: <https://keras.io/api/layers/activations/>

The Vanishing Gradient Problem

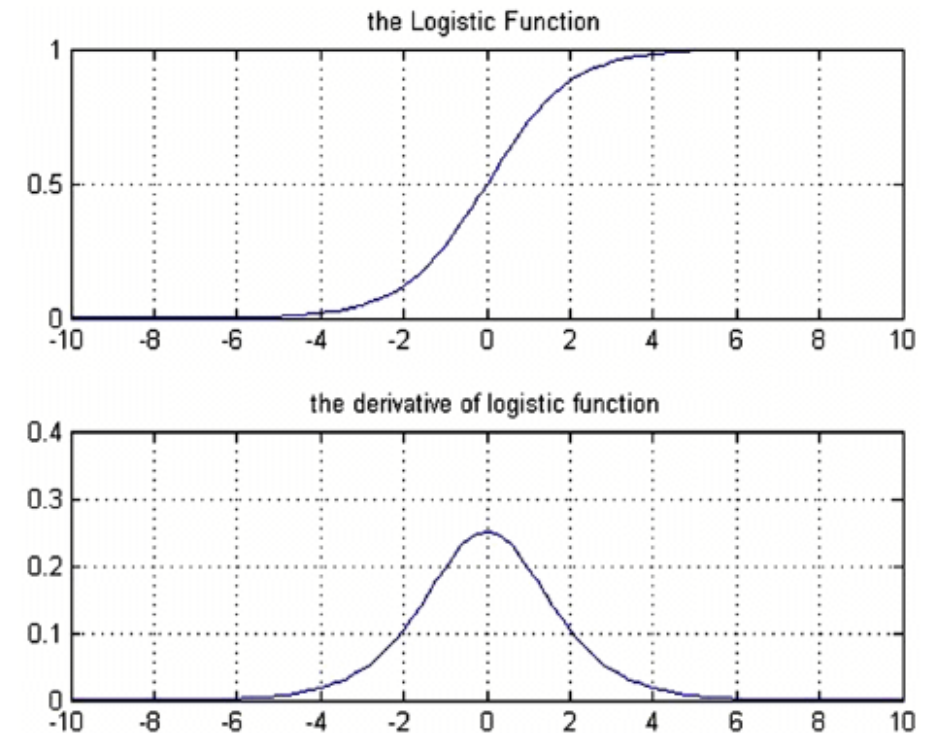
Vanishing Gradient Problem:

$$\text{gradient}_j = z_i * h_j(1 - h_j) * e_j$$

Derivative of the input Gradient of the error

For neurons in deep networks, due to the chain rule, this becomes related directly to a product of weighted derivatives of individual neurons

This can cause the gradient to approach 0, and for learning to slow close to stopping

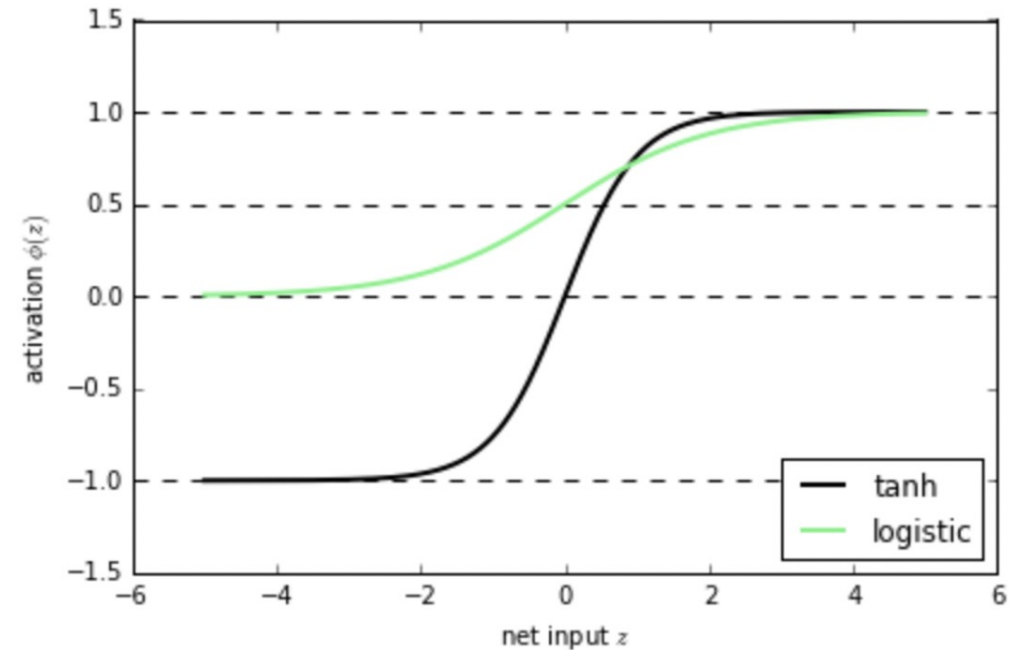


Tanh function

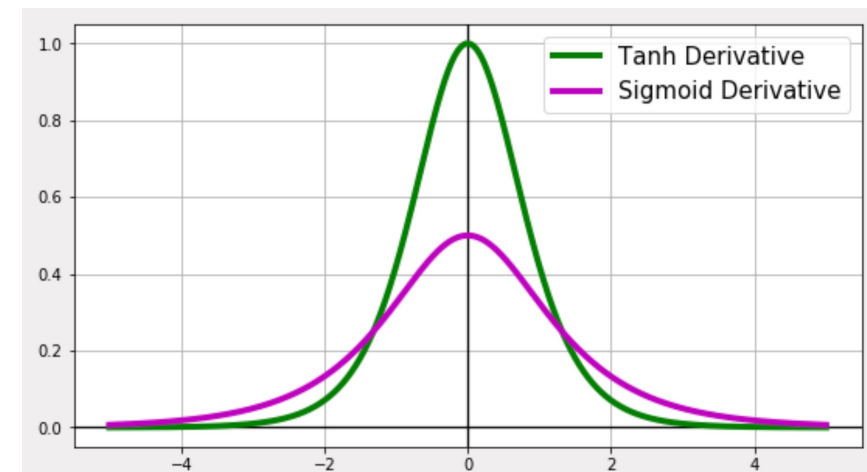
$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- tanh function is often used instead of logistic
- Unlike logistic, its range is -1 to 1. This is a greater range meaning that it is often steeper.
 - This means it has a higher gradient which can reduce the vanishing gradient problem

derivatives of tanh vs. logistic



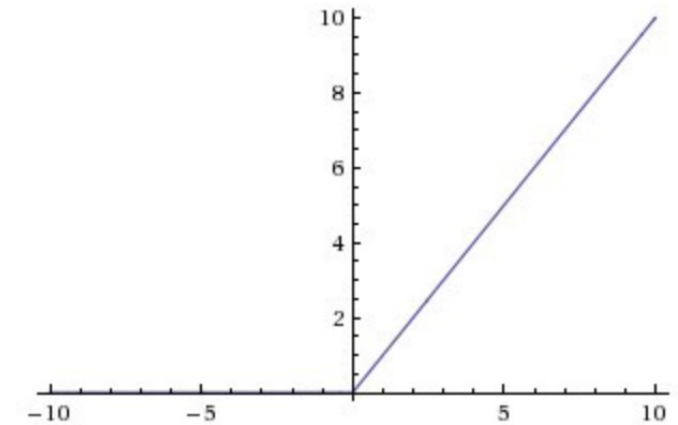
tanh Function vs. logistic



$$A(x) = \max(0, x)$$

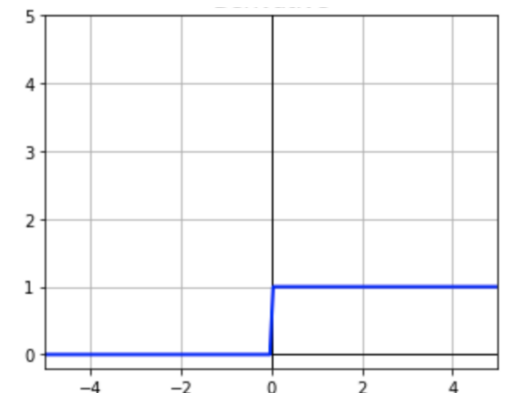
ReLu Function

- Just like tanh and logistic, ReLu is nonlinear, so it can be used in nonlinear regression problems
- Not truly differentiable, but differentiable in practice
 - If input ≤ 0 , derivative is 0, else derivative is 1
- Benefits:
 - Vanishing gradient problem is solved
 - Derivative is really fast to calculate
 - Sparsity of activations (many neurons go to 0)
 - This means that less calculations are made (commonly up to 50% of a network)
 - This means deep neural networks are faster to predict and train
- Problems:
 - Instead of vanishing, gradients can explode (exploding gradient problem)
 - “Dying ReLu problem” - Because of the sparsity, whole portions of the network stop responding to input during training and training effectively stops



ReLu function

Derivative of ReLu



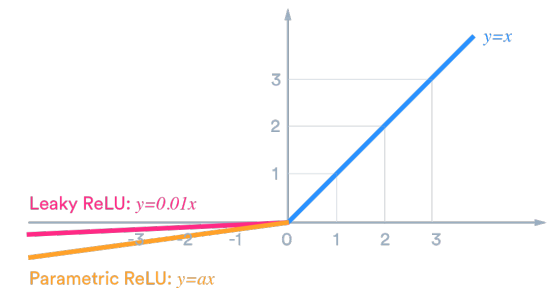
Lots of Options – Still a research area

- Many activations functions have been developed that address:
 1. The vanishing gradient problem
 2. The exploding gradient problem
 3. The dying ReLu problem
- ReLU solved the vanishing gradient problem with a computationally very simple method,
 - but it introduced the the exploding gradient problem and the dying ReLu problem.

- To solve the dying ReLu problem, slight modifications of ReLu were introduced

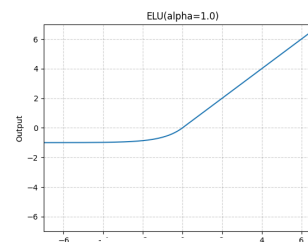
- Leaky ReLu
- Parametric ReLu

These are not continuously differentiable

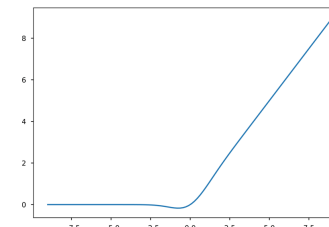


- Some studies show being continuously differentiable leads to faster convergence and these methods exist:

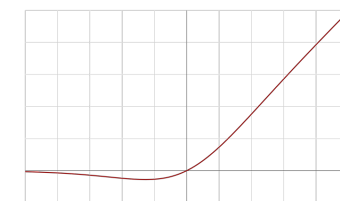
- ELU
- GELU – state of the art for NLP tasks
- SiLU = Swish – state of the art for many tasks
- Mish



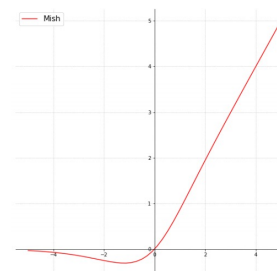
ELU



ReLU



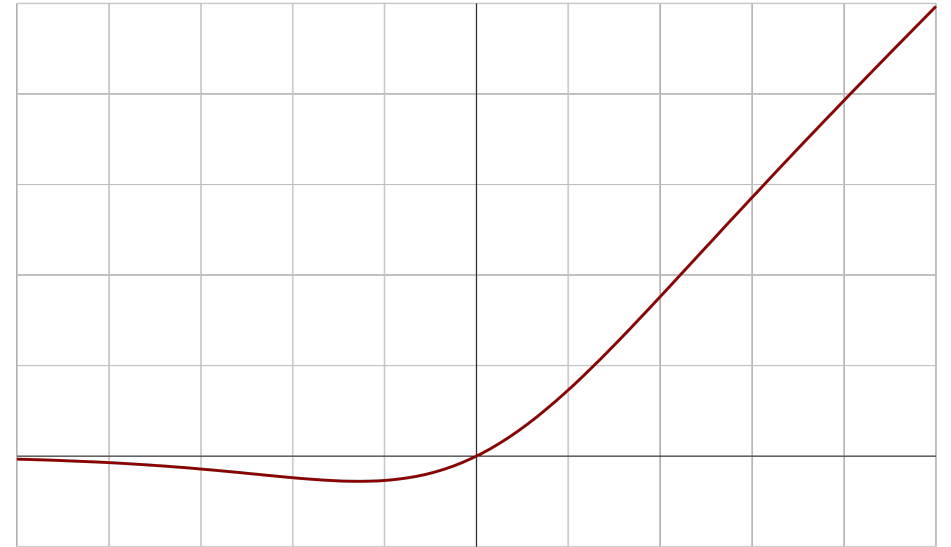
SiLU



Mish

Interesting History of SiLU

- Proposed alongside GELU in 2016 as Sigmoid Linear Unit (SiLU)
- Later, “re-proposed” in 2017 by Elfwing et al. calling it “SIL”
 - Paper quickly updated it to SiLU and credit given
- Later, “re-proposed” again, by Quoc Le (Google Brain) and named “Swish”
 - Didn’t cite either previous papers
 - Was contacted, and rather than give credit, added the β variable to make it “novel”
 - In practice, β was set to 1 by researchers
 - Implemented as part of Google’s tensor packages as Swift and SiLU was more or less overlooked
- Later, it GLU was used in BERT and GPT and people saw SiLU alongside it



$$\text{swish}(x) = x \text{ sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}}$$

Activation Functions: Take Aways

- All of these activation functions are OK, and choice of a function typically has a small impact on performance
 - but using a logistic function is generally a bad choice for hidden layers
- You can mix and match activation functions in your network
- I recommend using SiLU or GeLU as a default for hidden layers
- The activation function of the output layer is determined by the problem type and loss function
 - Logistic for Binary Classification and Multilabel classification
 - Softmax for Multiclass classification
 - Linear (No activation function) for regression



This is the first time I've mentioned this!

Optimization Algorithms

- Optimization algorithms define how you update at each step
- All optimization algorithms work “like gradient descent”
 - You are searching a space defined by weights and loss
 - You move in the direction of the derivative
- Each algorithm is a slight variation on this
 - Different claims on how they overcome challenges in the optimization function

Check them out: <https://keras.io/api/optimizers/>

Common Optimization Algorithms

These algorithms use batch-based updates

The differences between them are technical and aren't important

- Gradient Descent
- Gradient Descent with Momentum
- RMS Prop
- AdaGrad
- Adam

'Adam' optimizer – name is derived from “adaptive moment” optimizer

This is a good default choice for an optimizer – it tends to perform well. Intuitively it is still descending a gradient, but generally has faster training time than gradient descent

Want to know more? This article gives a GREAT explanation of their development:

<https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>