

Ill. Niklas Elmehed © Nobel Prize
Outreach

Geoffrey Hinton

Prize share: 1/2

The Nobel Prize in Physics 2024 was awarded jointly to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks"



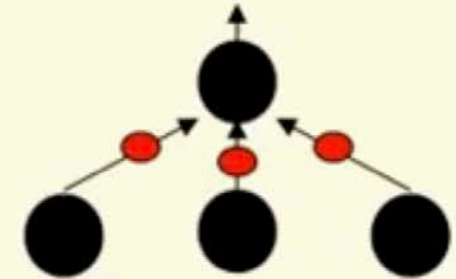
2. Deep Neural Networks

“I have always been convinced that the only way to get artificial intelligence to work is to do the computation in a way similar to the human brain. That is the goal I have been pursuing. We are making progress, though we still have lots to learn about how the brain actually works.” - **Geoffrey Hinton**

Let's start with Geoff Hinton's slide

How the brain works on one slide!

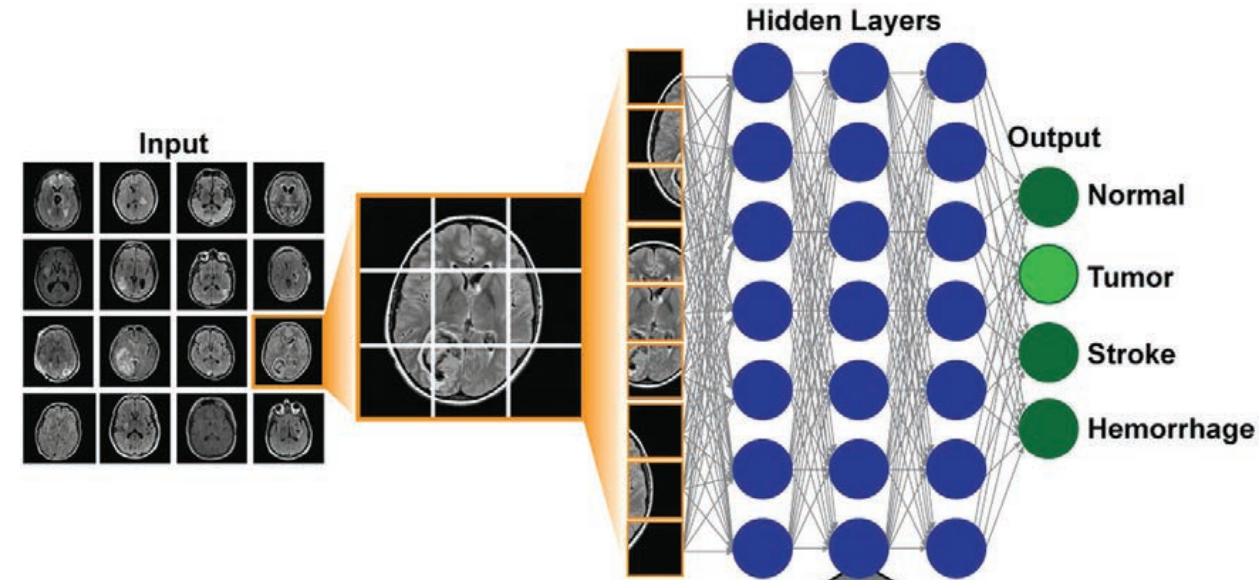
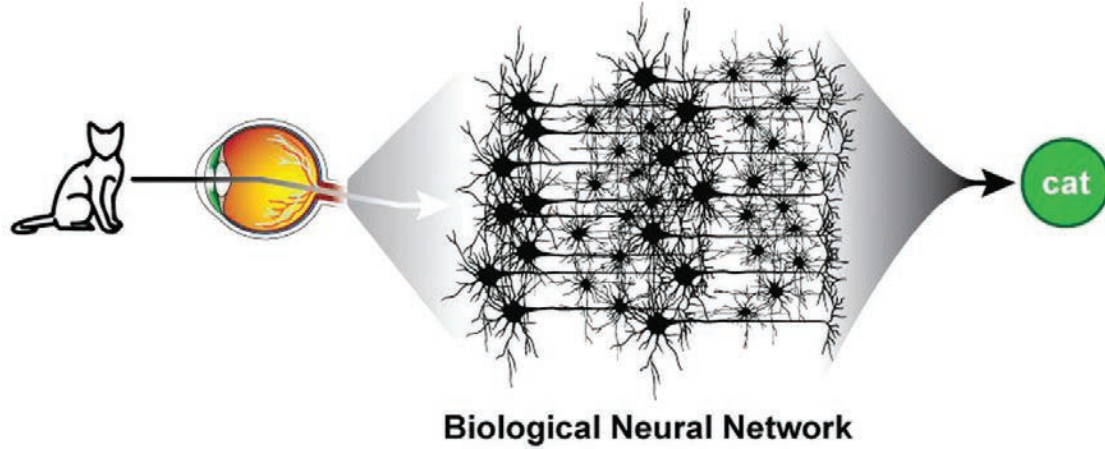
- Each neuron receives inputs from other neurons
 - A few neurons also connect to receptors.
 - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
 - The weights can be positive or negative.
- The synaptic weights **adapt** so that the whole network learns to perform useful computations
 - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^4 weights.
 - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.



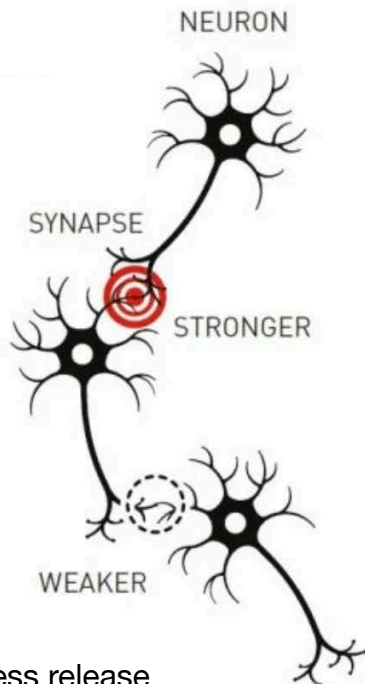
Biological Neurons

Artificial Neurons

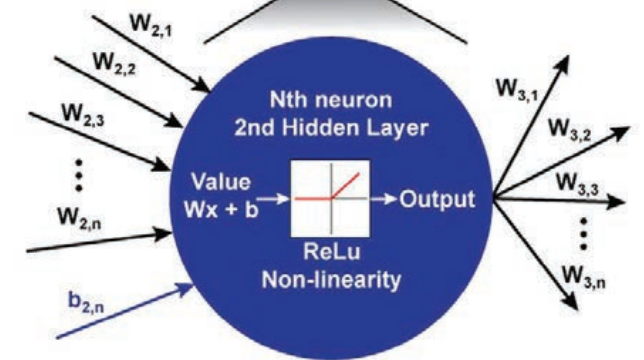
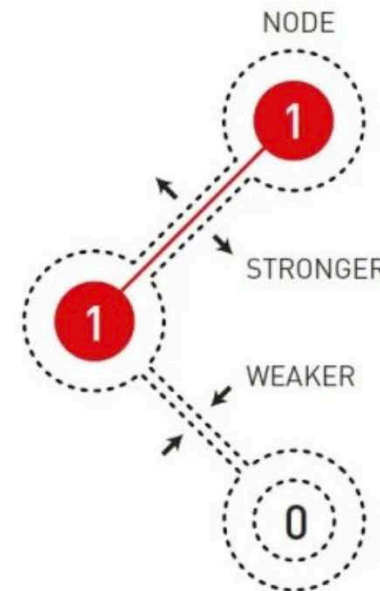
G. Zaharchuk et al. Deep learning in Neuroradiology. AJNR (2018)



The brain's neural network is built from living cells, neurons, with advanced internal machinery. They can send signals to each other through the synapses. When we learn things, the connections between some neurons get stronger, while others get weaker.



Artificial neural networks are built from nodes that are coded with a value. The nodes are connected to each other and, when the network is trained, the connections between nodes that are active at the same time get stronger, otherwise they get weaker.



(lower panel)

Nobel prize in Physics 2024 press release

“Hello, World” of deep learning

Let's try linear regression (fit $y = 3x + 1$ with a neural network)

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

```
# Simple linear model:  $y = 3x + 1$ 
```

```
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
```

```
ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)
```

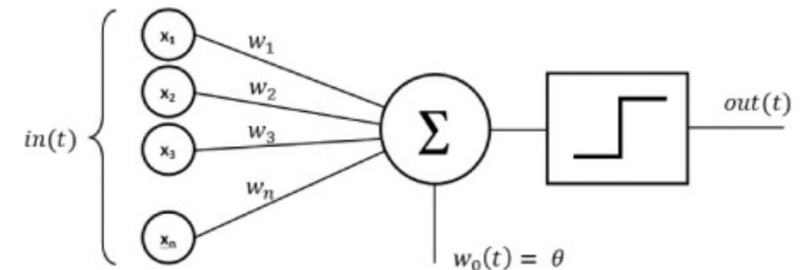
```
model = keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
```

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

```
model.fit(xs, ys, epochs=100)
```

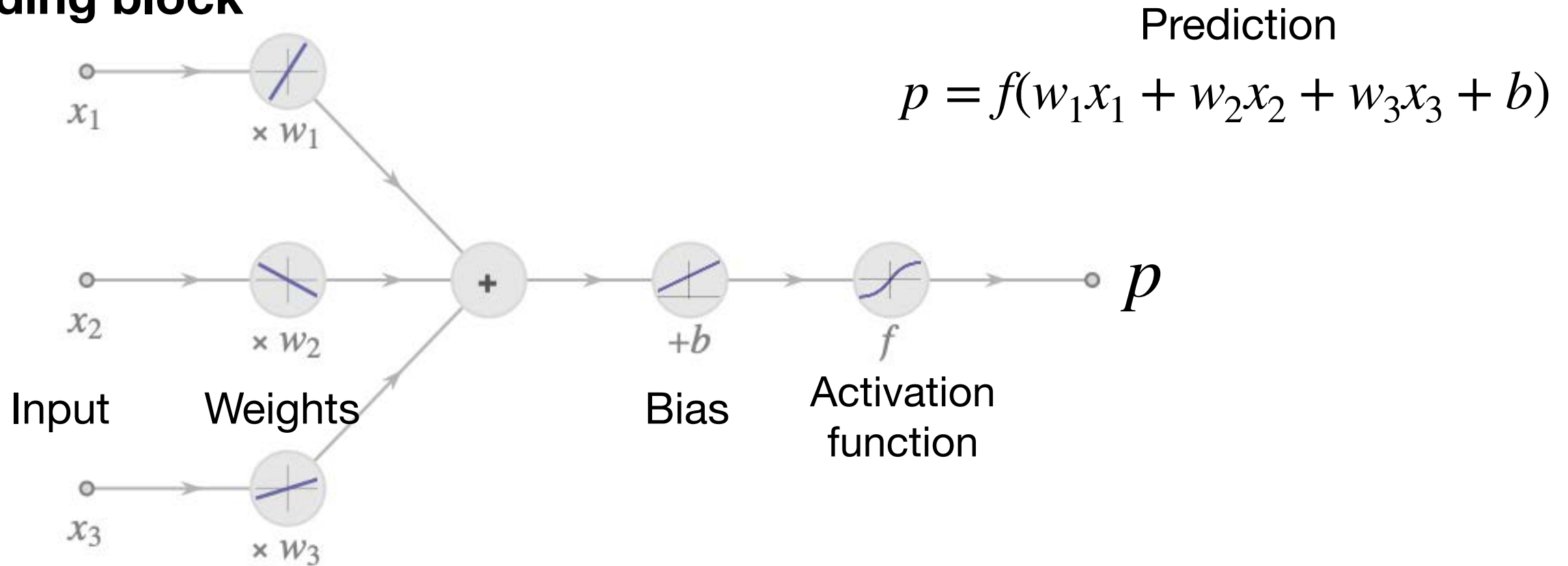
```
print(model.predict(np.array([10.0]))) # Predict  $3 \times 10 + 1$ 
```

Q: Compare this neural net to the perceptron.
What key feature is missing?



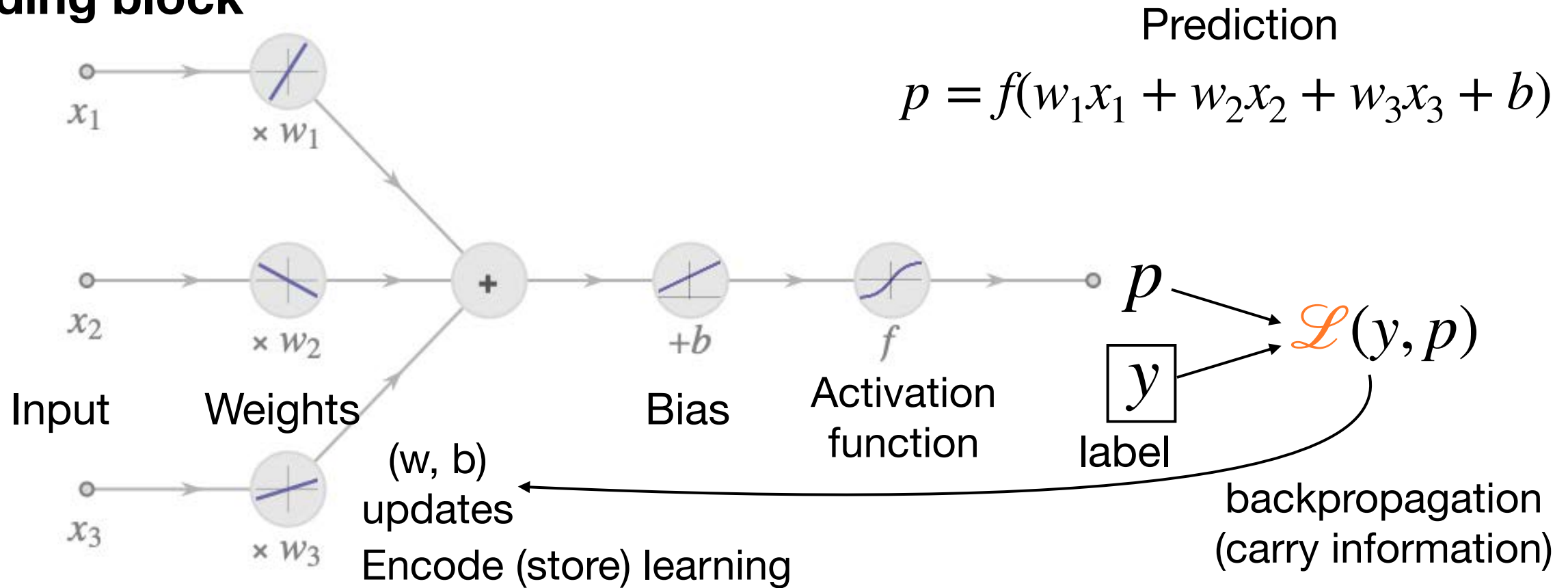
Inside an artificial neuron

Building block



Inside an artificial neuron

Building block

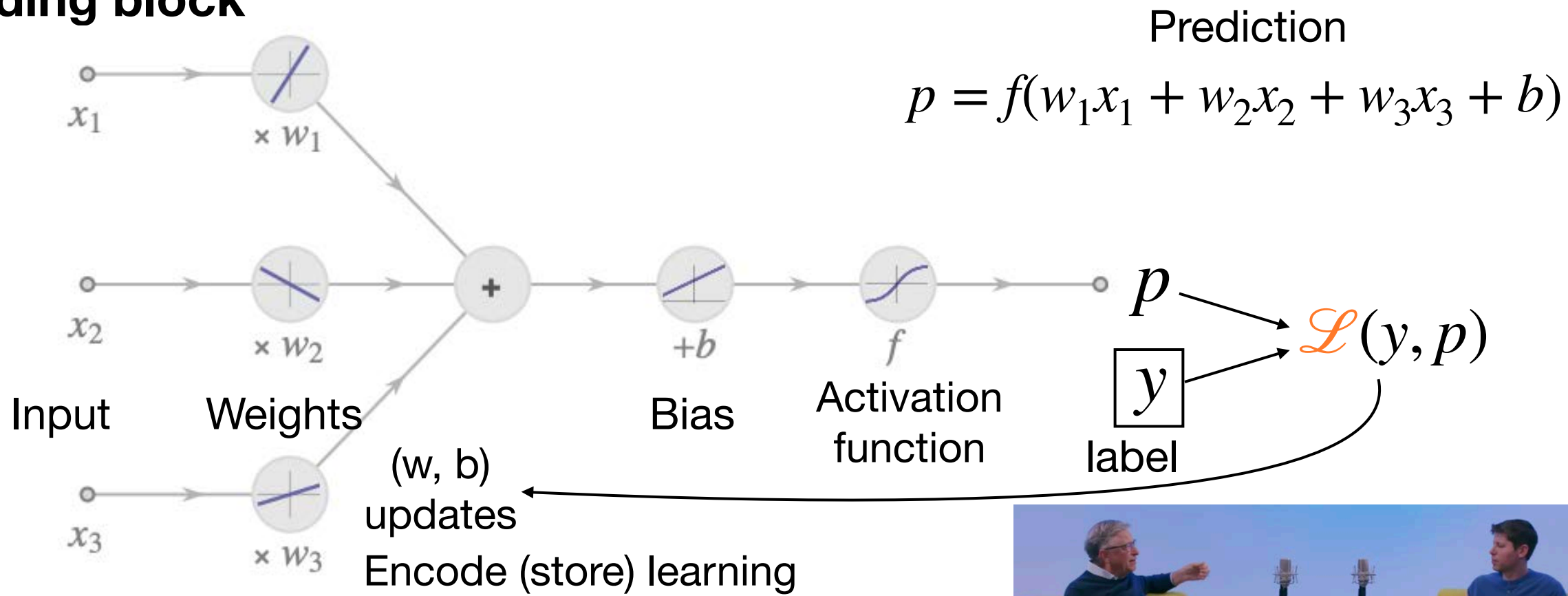


[Input: x_1, x_2, x_3] \rightarrow [Predict: $p = f(w \cdot x + b)$] \rightarrow (compare to y) \rightarrow [compute Loss(p, y)] \rightarrow
 \rightarrow backpropagate gradients: $\partial L / \partial w, \partial L / \partial b$ \rightarrow [Updated w, b] \rightarrow [Next Prediction: p] \rightarrow Loss \rightarrow Backprop \rightarrow inference ...

Quiz: What happens if we remove the bias 'b'? Can the neuron still model all patterns?

Inside an artificial neuron

Building block

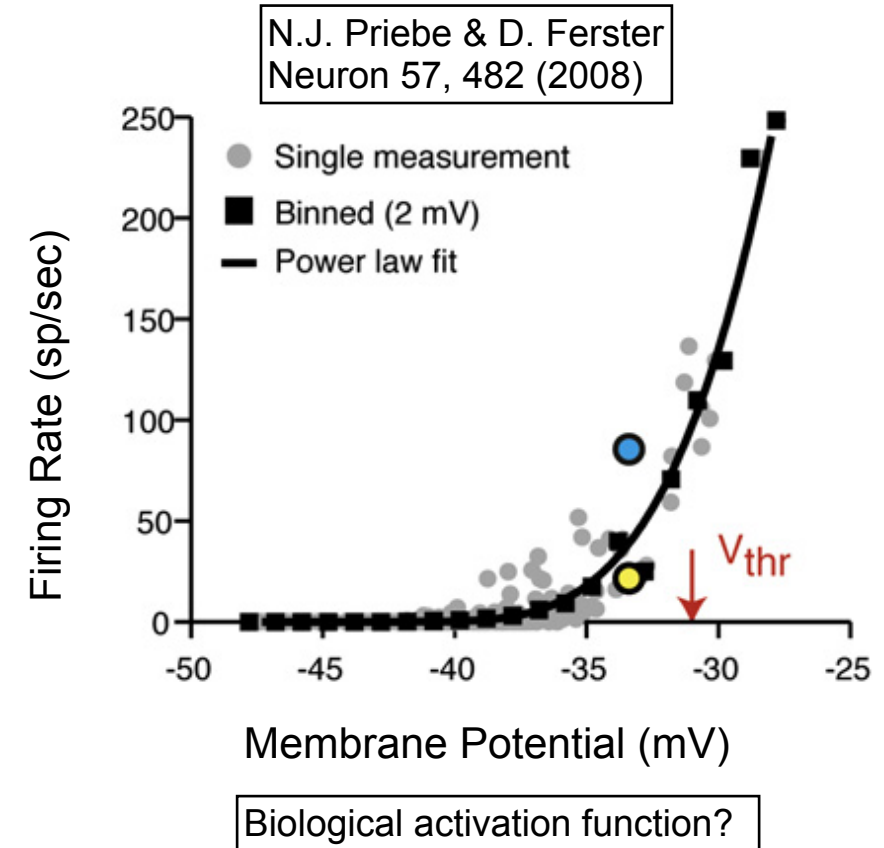
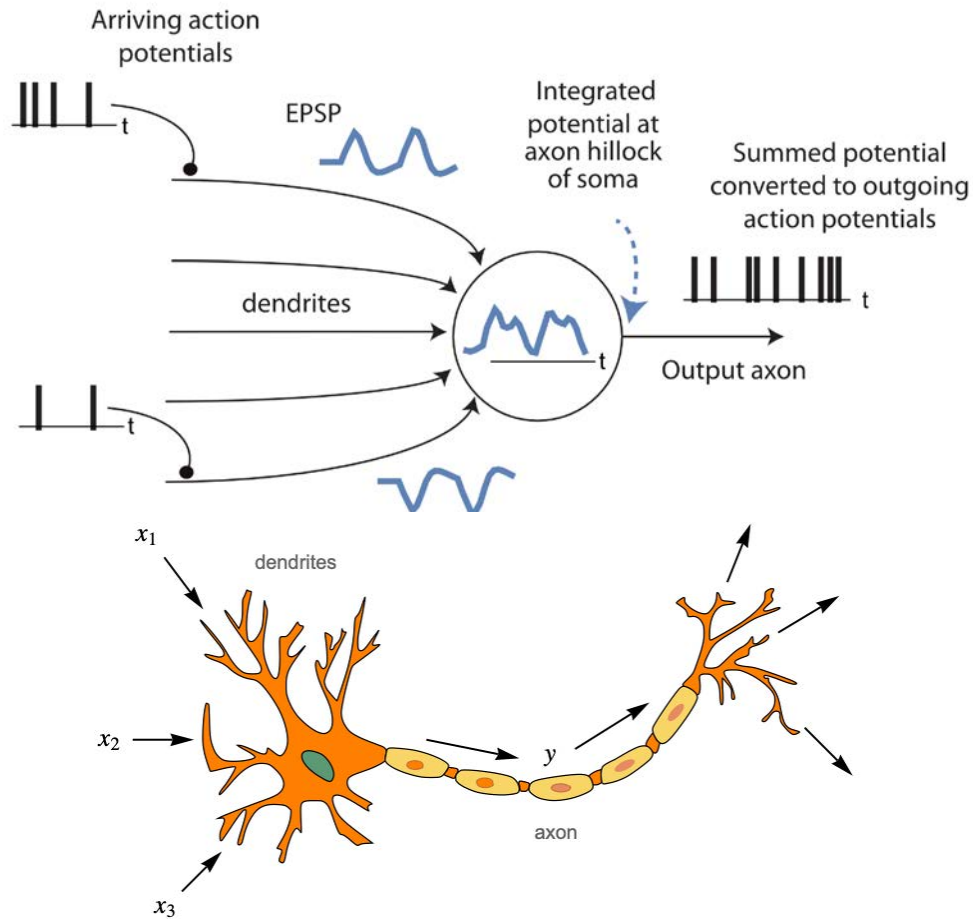


How is Shakespeare encoded in a DNN (like GPT4)?

If weights don't store text verbatim, where does the knowledge reside?
How does the net "remember" patterns without explicitly storing them?

but the idea of where is Shakespearean encoded?

Activation Functions in Biological Neurons

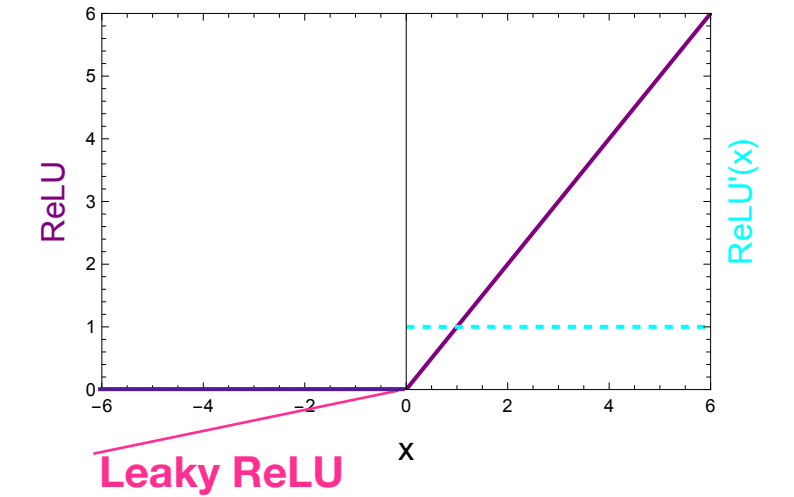
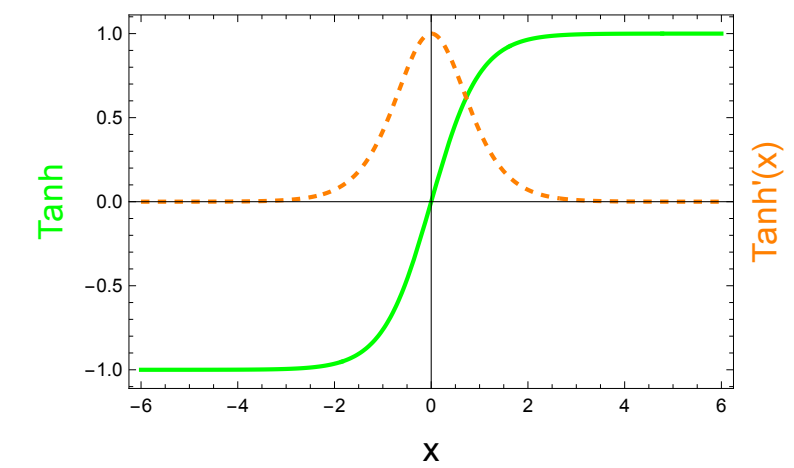
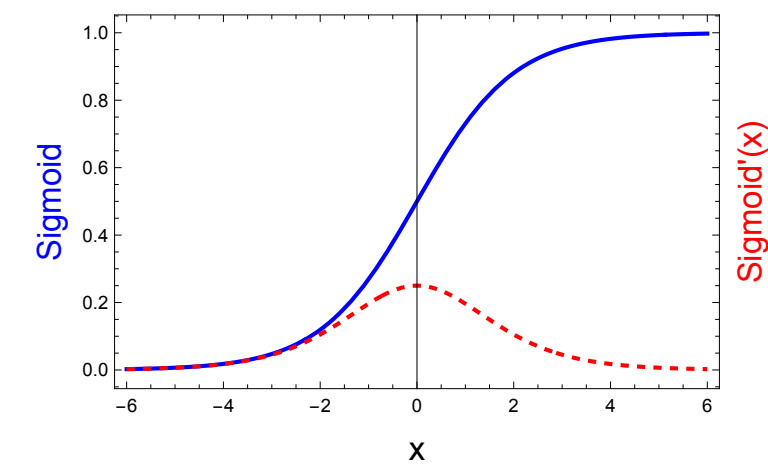


Threshold behavior: Neurons typically do not fire until the membrane potential exceeds a threshold.

The curve (right panel) effectively serves as the **neuron's activation function**, mapping input (membrane potential) to output (firing rate). Does it resemble the ReLU function in DNN?

Activation functions in DNN

Function	Formula	Characteristics
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	Smooth, (0,1) bounded, vanishing gradient problem
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Smooth, (-1, 1), centered
ReLU	$f(x) = \max(0, x)$	Simple, sparse, avoids vanishing gradient
Leaky ReLU	$f(x) = \max(0.01x, x)$	prevent the "dying ReLU" - where neurons stop learning.



Activation functions in Keras vs PyTorch

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=4, activation='relu'), # ReLU
    tf.keras.layers.Dense(units=4, activation='sigmoid'), # Sigmoid
    tf.keras.layers.Dense(units=4, activation='tanh') # Tanh
])
```

In Keras, the layer and activation are tightly coupled (hard to customize)

```
import torch.nn as nn
model = nn.Sequential(
    nn.Linear(4, 4),
    nn.ReLU(),
    nn.Linear(4, 4),
    nn.Sigmoid(),
    nn.Linear(4, 4),
    nn.Tanh()
)
```

PyTorch syntax is more transparent, allows more flexibility to customize models

```
def forward(self, x, attention_score):
    if attention_score > 0.5:
        return F.relu(x)
    else:
        return F.sigmoid(x)
```

Why activation functions matter in DNNs

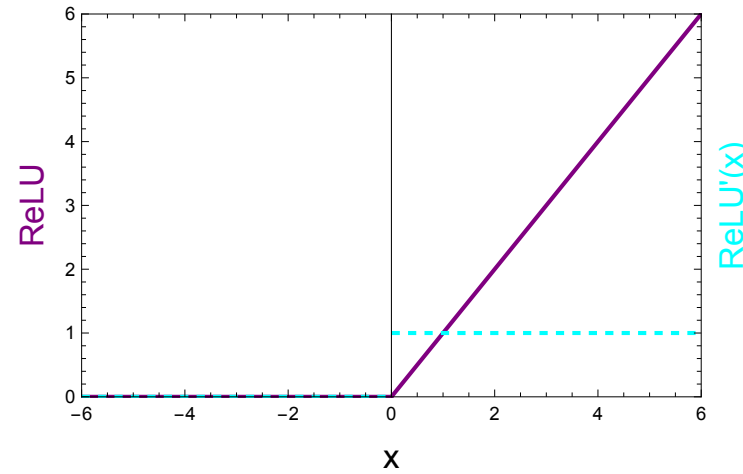
nonlinearity

- Activation functions introduce nonlinearity: DNN can learn complex patterns.
- Nonlinear activations mimic the "thresholding" behavior of biological neurons (see slide #8), enabling networks to respond selectively to patterns. why?
- Nonlinear activations, like the MOSFET's gating mechanism in digital circuits, enable selective neuron activation (responding to specific patterns).
- Abstraction discards irrelevant information; nonlinearity enables this by compressing data into patterns.
- Without nonlinearity, DNN would behave like a single-layer linear net regardless of depth. Nonlinearity allows stacking, hierarchy, abstraction, ...
- But nonlinear models are harder to explain & understand (black box)

ReLU

The simple trick powering DNN

$$\text{ReLU}(x) = \max(0, x)$$




1. How does piecewise-linear ReLU enable DNN to model nonlinear functions?
2. What happens to the gradient of ReLU for inputs where $x < 0$? Why might this cause training challenges?
3. Imagine a network with ReLU in all but the final layer. Can it ever produce negative outputs? How?
4. If we replace ReLU with $f(x) = |x|$, will the network still learn effectively?
5. Does a network with ReLU activation still need a bias term in each layer?

Parameters vs. Hyperparameters


What's the difference? Designing vs. training a model

Parameters: The network's weights and biases, learned during training.

Add this snippet after Hello world in Colab:

```
 for layer in model.layers:  
    print(layer.get_weights()) # Display model parameters (weights, biases)
```



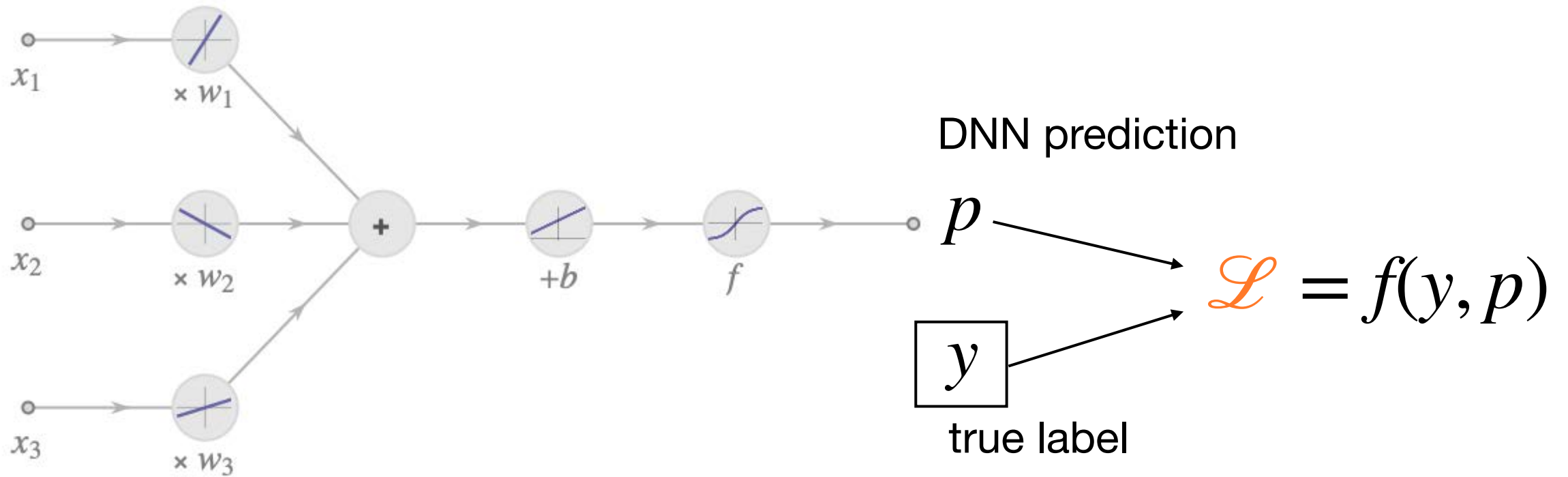
```
 [array([[ -8.7248999e-01, -2.8335035e-01,  8.2182966e-04, -7.5318199e-03]],  
        dtype=float32), array([ 0.10682962,  0.10272265, -0.02109917, -0.04345334], dtype=float32)]  
[array([[ 0.26714262],  
        [ 0.5089375 ],  
        [-0.5574685 ],  
        [-0.6536985 ]], dtype=float32), array([0.09900711], dtype=float32)]
```

Hyper-parameters: Design decisions made before training,

- Number of layers and neurons per layer
- Activation functions (ReLU, tanh, sigmoid, ...)
- Optimization method (Adam vs. SGD) -> will cover later
- Regularization techniques (Dropout, Batch normalization)

Loss Functions (a.k.a. Cost Function)

measures how well a model's prediction (p) matches the labels (y).



Regression losses (e.g., Mean Squared Error (MSE), MAE).

Classification losses (e.g., sigmoid, cross entropy).

Mean Squared Error (MSE) loss function

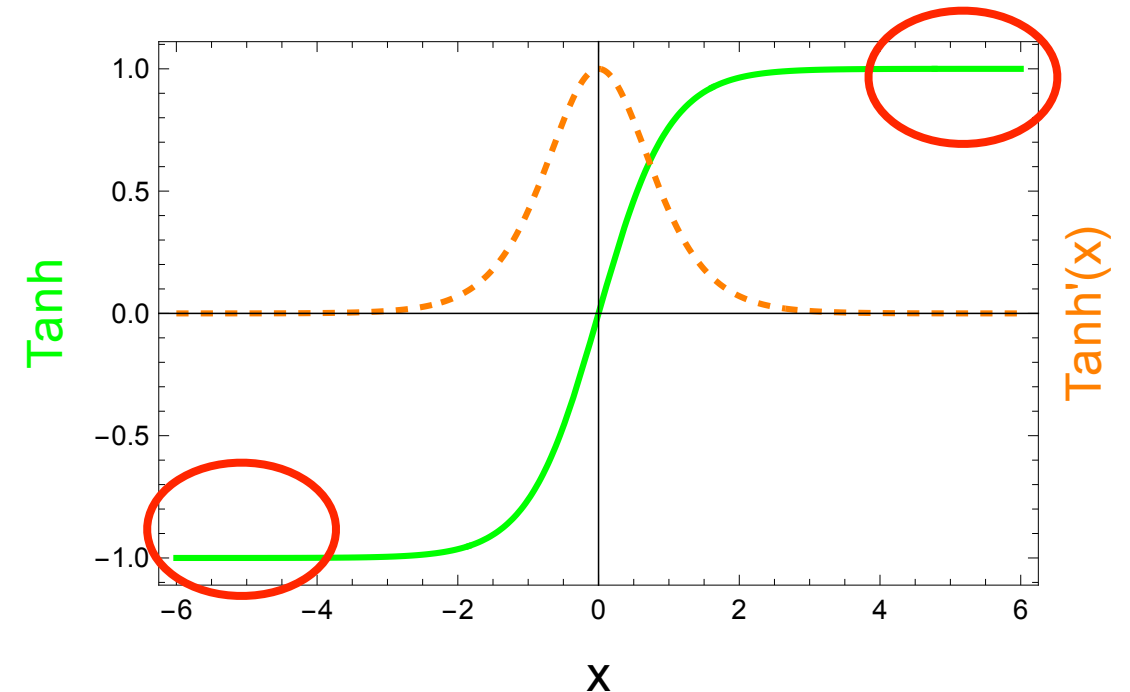
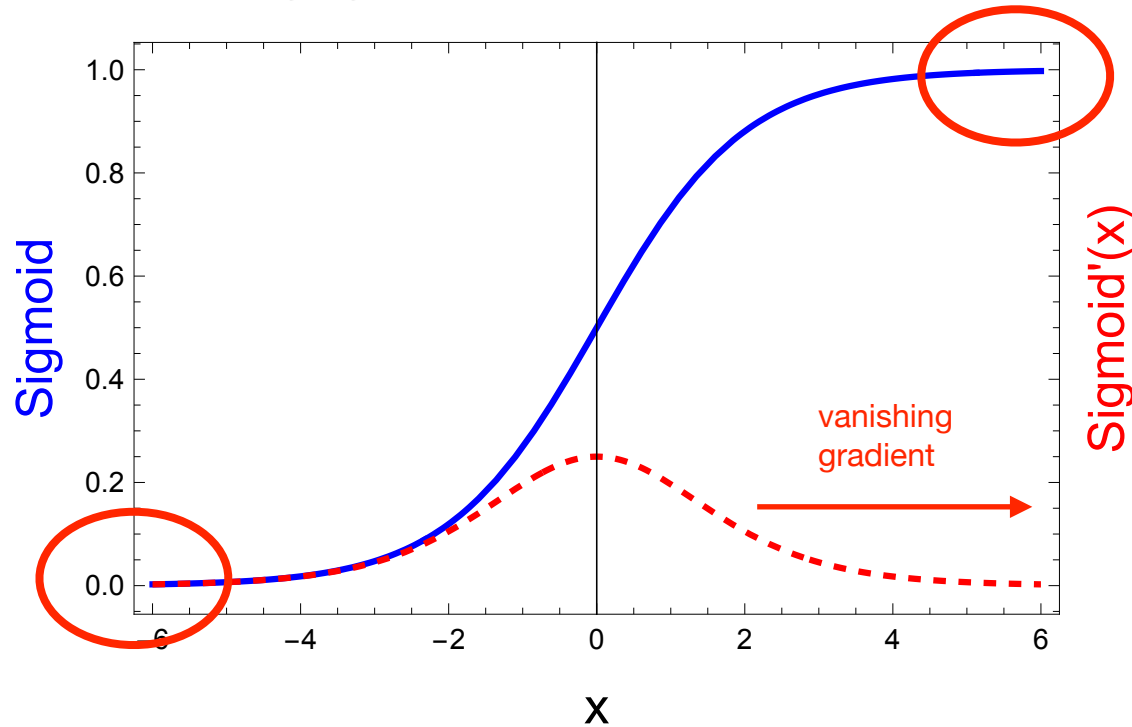
Popular regression loss function

$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - p_i)^2 \qquad \mathcal{L}_{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - p_i|$$

- Quadratic penalization: Penalizes large errors more heavily - sensitive to outliers.
- Derivatives are simple - easy to optimize using gradient methods
- MSE common for regression tasks, but if too sensitive to outliers, use MAE.
- e.g. For predicting house prices: if a few high-priced houses skew results, then using MAE gives better average error by reducing the influence of outliers.

MSE: Neuron saturation

Vanishing gradient: Loss function loses sensitivity to weight parameters



- In saturated regions (red-circle), change in x cause little change in activation: Model loses sensitivity.
- For classification, MSE treats sigmoid output as numbers, not probabilities
- Trick to prevent saturation: In MNIST image classification, large pixel values (0-255) can cause sigmoid activation function to saturate => vanishing gradients & slow training. Normalize pixel intensity to $[0, 1]$ (divide by 255) to ensure stable gradient flow and faster convergence.
- How to fix these issues for classification tasks in general?

Exercise: Handcraft a loss function for classification

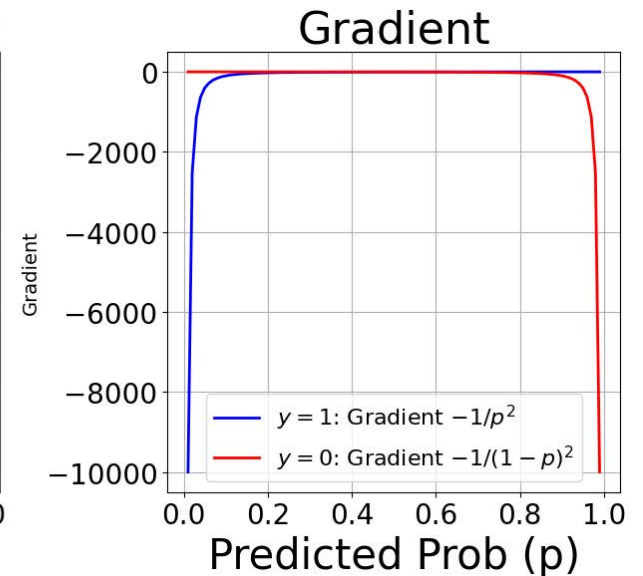
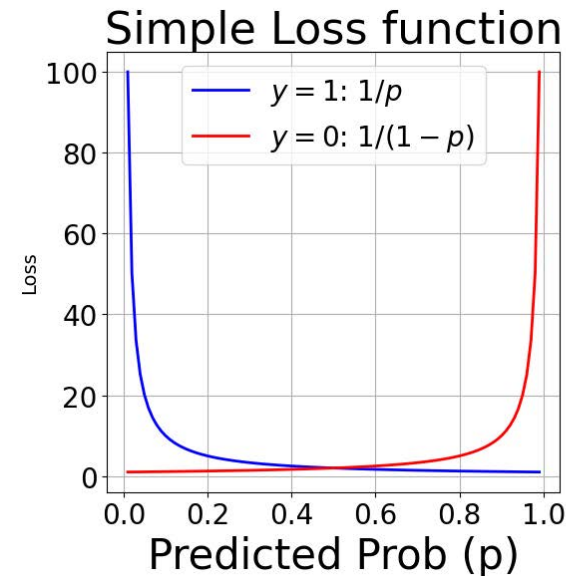
goal: penalize wrong probability predictions (p) for y=0 or y=1

star with an ansatz: $\mathcal{L} = y \cdot f(p) + (1 - y) \cdot g(p)$

- If correct label is y=1, only f(p) matters; f(p) should penalize (increase loss) as $p \rightarrow 0$. How about $f(p) = 1/p$?
- If correct label is y=0, only the right term survives; g(p) should penalize prediction $p \rightarrow 1$. $g(p) = 1/(1-p)$?
- First attempt: $f(p) = 1/p$, $g(p) = 1/(1-p)$

$$\mathcal{L} = \frac{y}{p} + \frac{1-y}{1-p}$$

$$\frac{\partial \mathcal{L}}{\partial p} = -\frac{y}{p^2} + \frac{1-y}{(1-p)^2} \quad \text{gradient explodes!}$$



- To fix exploding gradients, how about using $f(p) = -\log p$, $g(p) = -\log(1-p)$? then

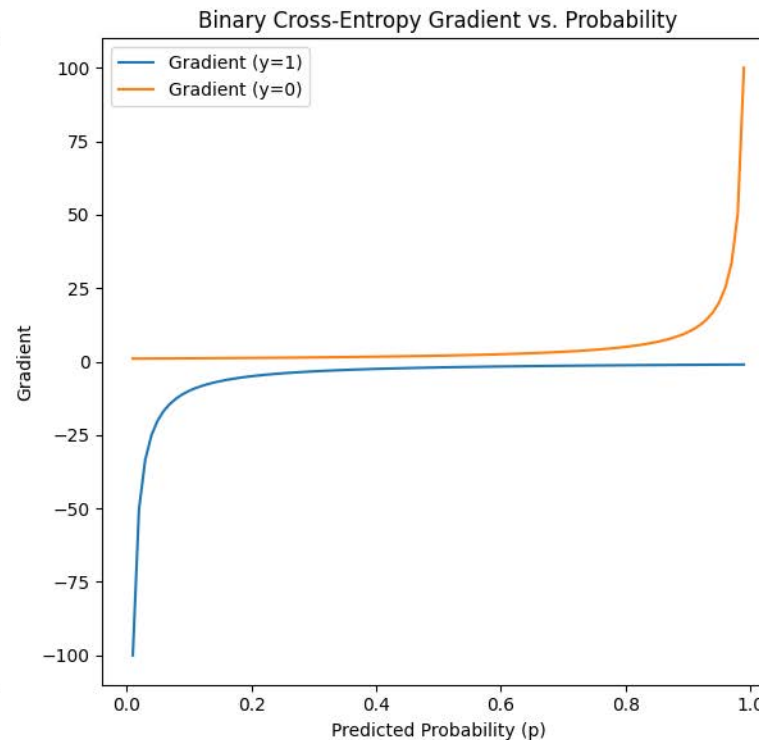
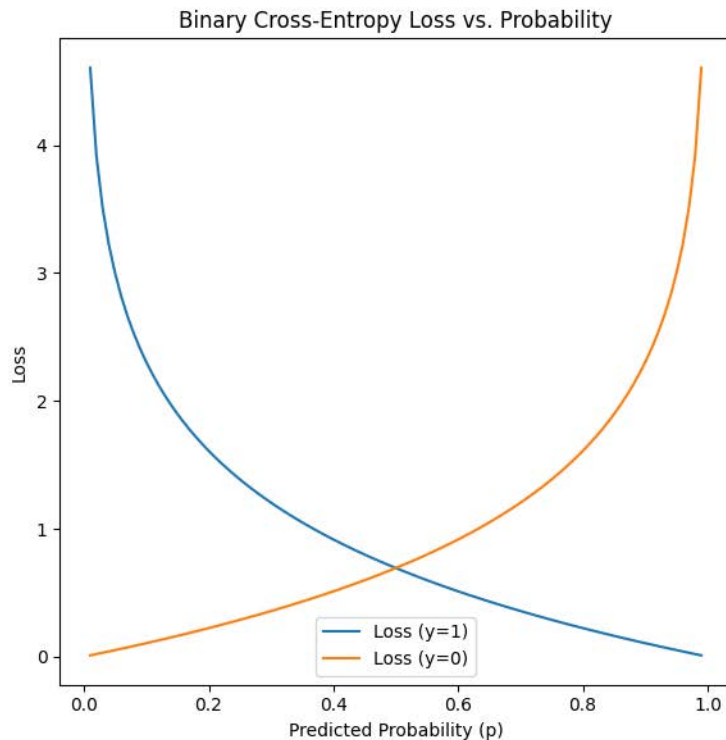
$$\mathcal{L} = -y \cdot \log p - (1-y) \cdot \log(1-p) \quad \text{This is the cross entropy loss!}$$

Cross entropy loss (binary)

$$\mathcal{L} = - [y \log(p) + (1-y) \log(1-p)]$$

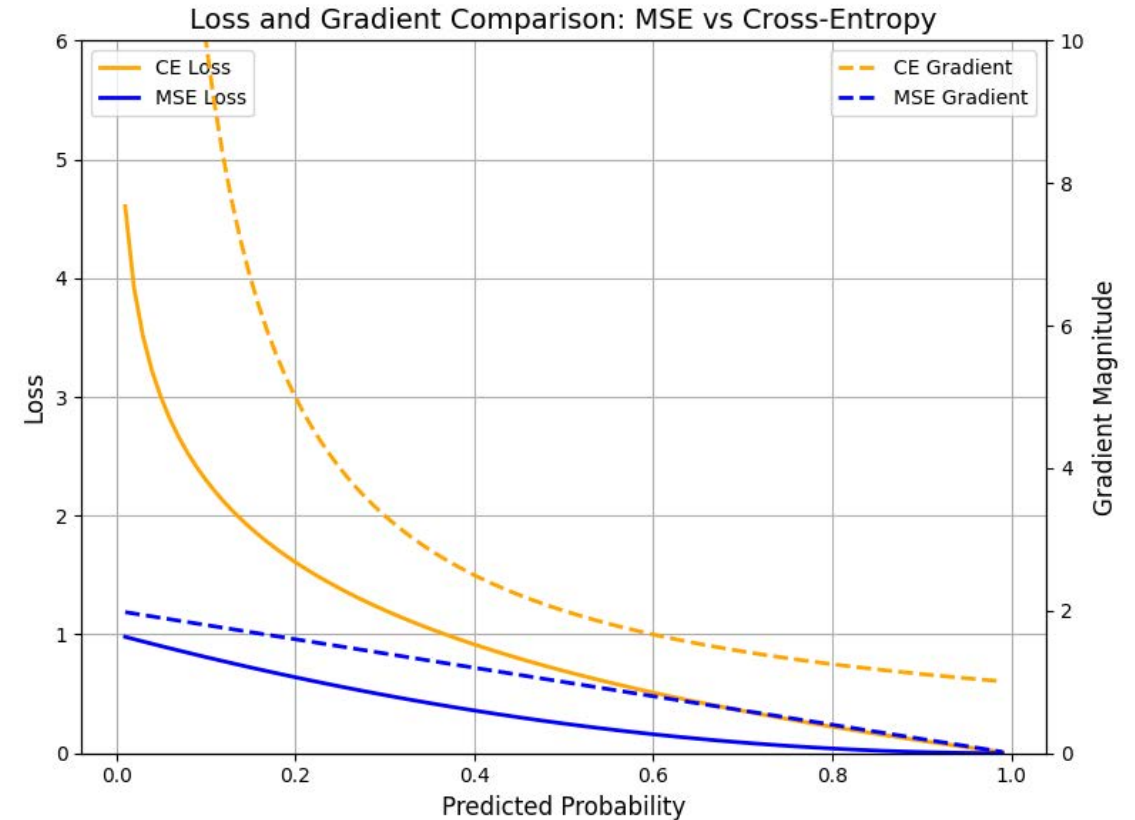
- If $y=1$, only $-\log(p)$ matters, and we penalize predictions as $p \rightarrow 0$
- If $y=0$, only $-\log(1-p)$ matters, and we penalize prediction $p \rightarrow 1$

Cross-entropy treats p as probability (great) and penalizes wrong predictions heavily.



Let's compare MSE Loss vs. C.E. Loss

- For confidently wrong prediction ($y = 1$, yet $p = 0.1$):
 - CE loss: 2.302, CE gradient: -10. (CE penalize wrong 'p' more strongly)
 - MSE loss: 0.81, MSE gradient: -1.8
-
- For confident correct pred ($y = 1$, $p = 0.9$)
 - CE loss 0.105, grad -1.11 (sensitivity)
 - MSE loss 0.01, grad -0.2



Cross entropy loss penalizes confident wrong predictions more heavily and shows sharper gradients => better learning for classification tasks compared to MSE.

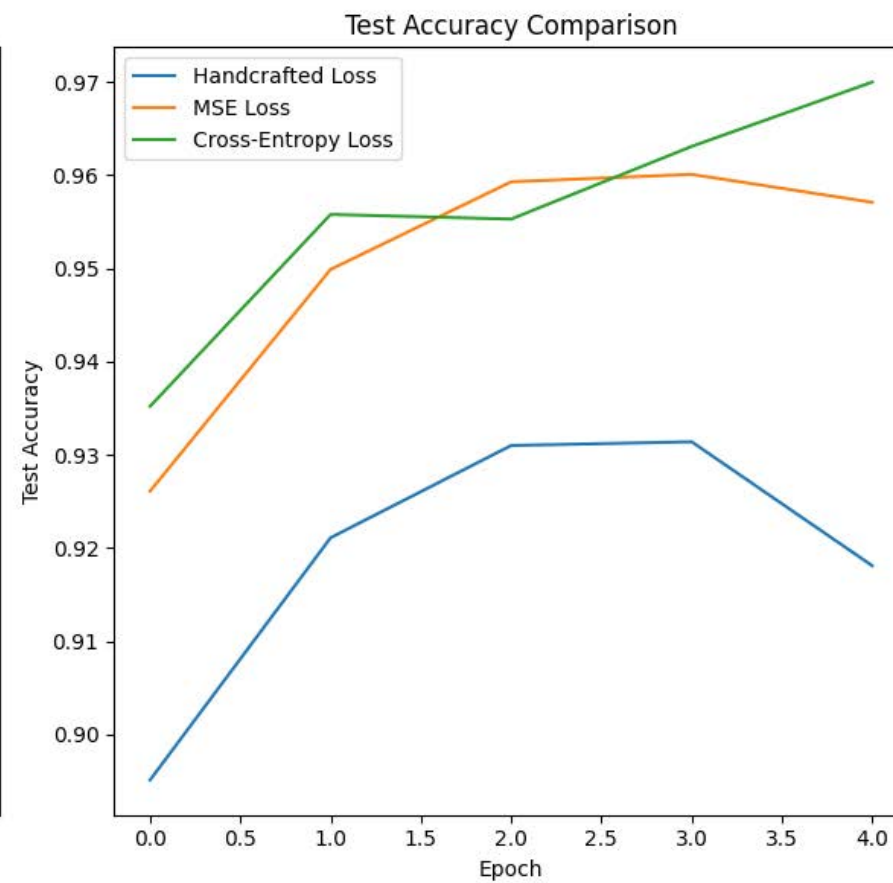
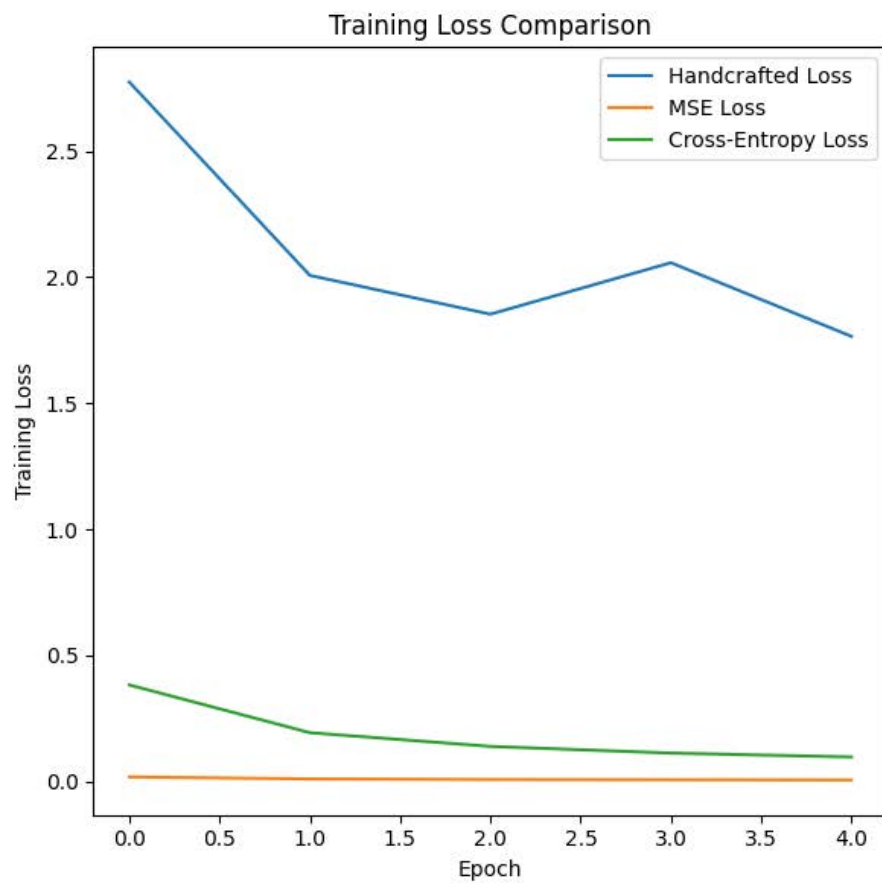
For non-binary classification (multi class)

Categorical cross entropy loss

$$\mathcal{L} = - \sum_i^C y_i \log(p_i)$$

- C: number of classes (e.g. for MNIST, C=10)
- For the correct class ($y_i = 1$), $-\log(p)$ term penalizes small predicted probability ($p \rightarrow 0$) by increasing the loss value.
- Suppose $C = 3$ (dog, cat, rabbit).
- True label: Dog ($y = [0, 1, 0]$); Predicted prob: $p = [0.2, 0.7, 0.1]$; $L = 0.357$
- Instead if $p = [0.2, 0.1, 0.7]$, then $L = 2.3$

Comparison with MNIST dataset



Quiz: choose from MSE, MAE, or Cross Entropy

1. You're training ML model to predict food delivery time. Most orders take 20-40 min, but occasionally there's traffic jam pushing delivery time to ~3 hours. Which loss function would you use to evaluate your ML model performance?
2. You're building a blood glucose level monitoring system. Less than 10 mg/dL error is acceptable, but large errors (± 50 mg/dL) cause dangerous situation. Would you use MSE or MAE to minimize dangerous errors?
3. You're predicting house prices (p , in \$million). $p = [0.5, 0.7, 0.9, 1.2]$ and true house prices are $y=[0.6, 0.6, 0.8, 1.0]$. Compute MSE, MAE, cross entropy.
4. You're predicting if an email is spam ($y=1$) or not ($y=0$). For each email, the predicted $p = [0.99, 0.01, 0.95, 0.05]$, and $y = [0, 1, 1, 0]$. Compare MSE/MAE/CE.
5. You're building a model that outputs (temperature tomorrow, probability rain) as for example (25, 0.7). How would you design a loss function for this model?