Introduction to Deep Learning

ECE/CS 498 DS U/G

Lecture 21

Ravi K. Iyer

Dept. of Electrical and Computer Engineering

University of Illinois at Urbana Champaign

Announcements

- MP3 Checkpoint 2 due today
  - Remaining Task 3 and Task 4 to be released today
- HW4 on HMM and FGs released today
  - Due on Wednesday, April 24
- ICA 6 on SVM and neural networks next Wednesday, April 24
- **Grad projects**: Mid-project progress report due on Friday, April 19
  - Students are encouraged to discuss with the instruction staff during office hours on Wednesday (Apr 17)
- No discussion section on Friday, April 19
  - Additional office hours in place of it in CSL 141 from 4-5pm

Perceptron Model

- The core of the neural network is perceptron model

Forward

$\boldsymbol{x} = [x_1, x_2, x_3]$

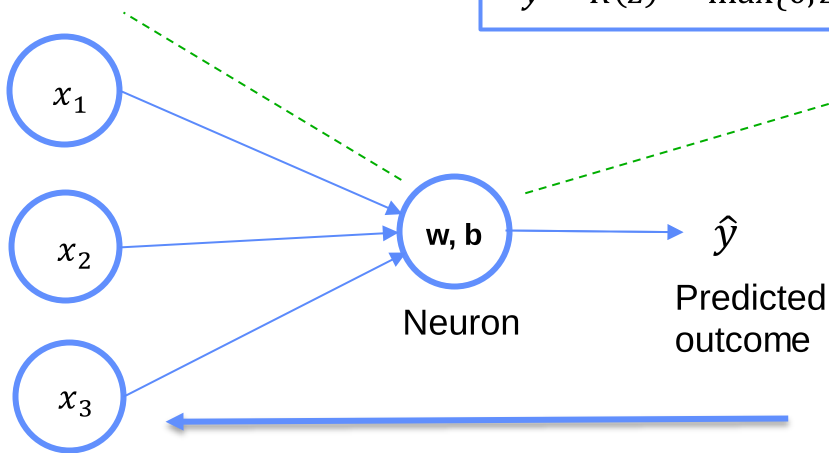
$z = \boldsymbol{w}^T\boldsymbol{x} + \mathrm{b}$

Activation Unit

$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$ (Sigmoid)

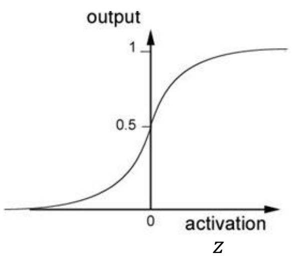
Or

$\hat{y} = R(z) = \max\{0, z\}$ (ReLU)



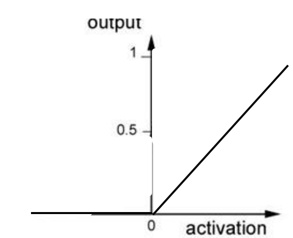
Backward

Sigmoid Function



$\sigma(z) = \frac{1}{1+e^{-z}}$

ReLU Function



$\mathrm{R}(z) = \max\{0, z\}$

Update Rule (Backward):

$$w_{t+1} = w_t - \eta\nabla J(w_t)$$

η : Learning rate

Loss

$$J(w) = \frac{1}{N}\sum_{i=1}^{N} L(w.x^{(i)}, y^{(i)})$$

Computing Gradient

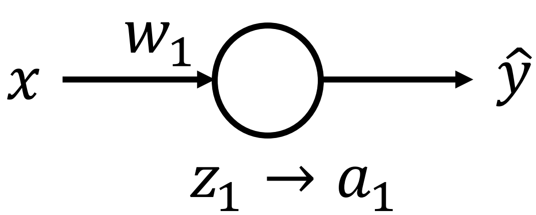
$$\nabla J(\mathbf{w}_0) = \left(\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \cdots, \frac{\partial J(\mathbf{w})}{\partial w_n}\right)_{\mathbf{w}_0}$$

N: number of samples $\quad x^{(i)}$: feature of i^{th} sample

Neural Network: Forward

Following neural networks have sigmoid activation function

Example 1



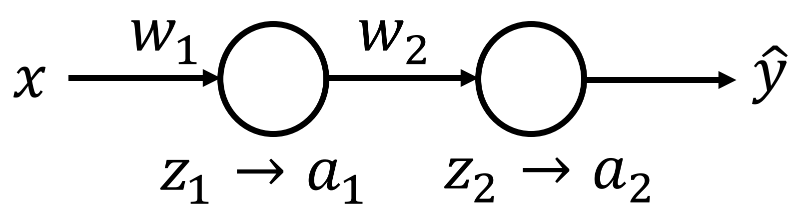
Forward equation:

$$z_1 = x.w_1 + b_1$$

$$a_1 = \sigma(z_1)$$

$$\hat{y} = a_1 = \sigma(x.w_1 + b_1)$$

Example 2



Forward equation:

$$z_1 = x.w_1 + b_1$$

$$a_1 = \sigma(z_1)$$

$$z_2 = a_1.w_2 + b_2$$

$$a_2 = \sigma(z_2)$$

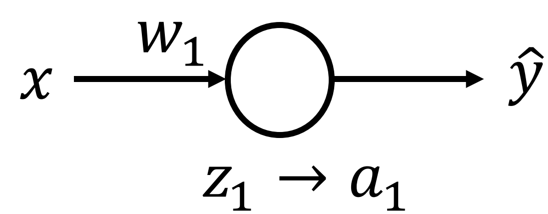
$$\hat{y} = a_2 = \sigma(a_1.w_2 + b_2)$$

$$= \sigma(\sigma(x.w_1 + b_1).w_2 + b_2)$$

Training the model

- The output of the model should be as close to y (ground truth value for input x) as possible
- Mean squared error (error in output layer): $(\hat{y} - y)^2$
- How to train the model i.e., find the weights that minimize the loss?
  - Apply gradient descent
  - Compute gradient using **Backpropagation** (fancy name for chain rule of derivatives)

Example 1



Forward equation:

$$z_1 = x.w_1 + b_1$$

$$a_1 = \sigma(z_1)$$

$$\hat{y} = a_1 = \sigma(x.w_1 + b_1)$$

Refresher on Chain rule for derivatives

$$f(x) = A(B(C(x)))$$

A, B, and C are activation functions at different layers. Using the chain rule we easily calculate the derivative of $f(x)$ with respect to x:

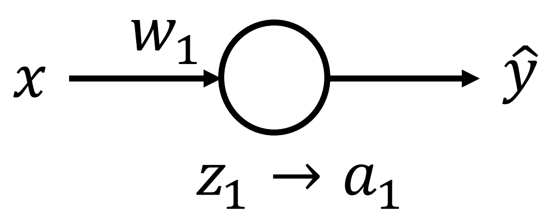
$$f'(x) = f'(A) \cdot A'(B) \cdot B'(C) \cdot C'(x)$$

How about the derivative with respect to B? To find the derivative with respect to B you can pretend $B(C(x))$ is a constant, replace it with a placeholder variable B, and proceed to find the derivative normally with respect to B.

$$f'(B) = f'(A) \cdot A'(B)$$

Backpropagation: Computing Partial Derivatives

Example 1



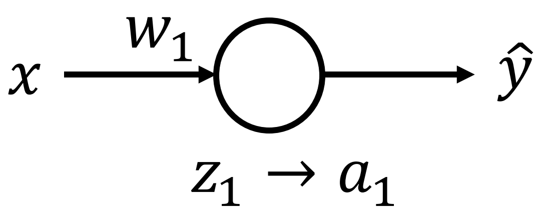
Forward equation:

$$z_1 = x.w_1 + b_1$$
$$a_1 = \sigma(z_1)$$
$$\hat{y} = a_1 = \sigma(x.w_1 + b_1)$$

Function	Formula	Derivatives
Weighted input	$z_1 = x.w_1 + b_1$	$\frac{\partial z_1}{\partial w_1} = x$ $\quad \frac{\partial z_1}{\partial x} = w_1$
Sigmoid	$a_1 = \sigma(z_1) = \frac{1}{1+e^{-z}}$	$\frac{\partial a_1}{\partial z_1} = \sigma(z_1)\sigma(-z_1)$
Cost function (loss)	$L = (\hat{y} - y)^2$	$\frac{\partial L}{\partial a} = 2(\hat{y} - y)$

Backpropagation: Example 1

Example 1



Training the model is equivalent to minimizing the loss $L = (\hat{y} - y)^2$ using gradient descent. Compute $\frac{\partial L}{\partial w_1}$.

Backpropagation: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_1}$

Chain rule – L is directly dependent on a_1

$= \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$

Chain rule - a_1 is directly dependent on z_1

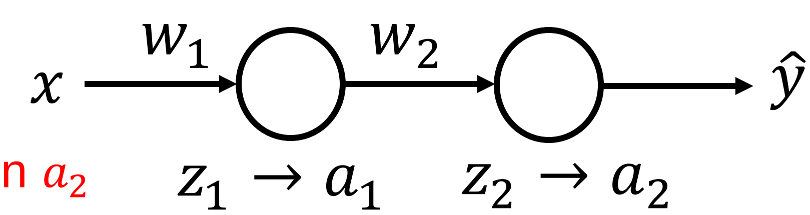
$= 2(\hat{y} - y)\sigma(z_1)\sigma(-z_1)x$

Substituting from table

Backpropagation: Example 2

Compute $\frac{\partial L}{\partial w_1}$ using backpropagation.

Example 2



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial w_1}$$

L is directly dependent on a_2

$$= \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_1}$$

a_2 is directly dependent on z_2

$$= \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

z_2 is directly dependent on a_1

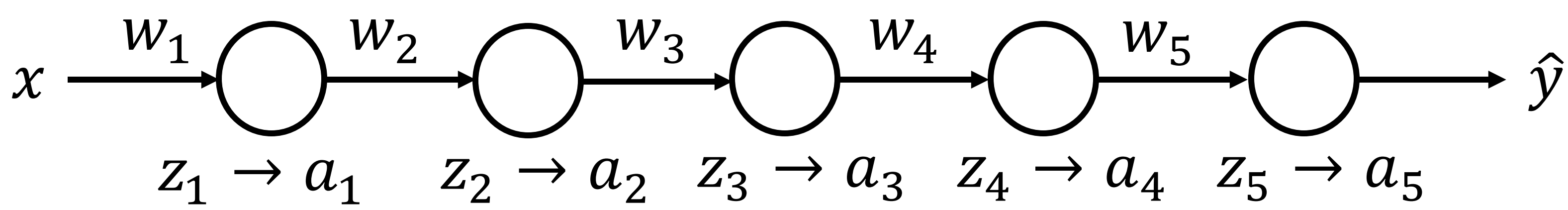
$$= \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

a_1 is directly dependent on z_1

$$= \overbrace{2(\hat{y} - y)}\ \overbrace{\sigma(z_2)\sigma(-z_2)}w_2\overbrace{\sigma(z_1)\sigma(-z_1)}x$$

Each of the above derivates can be easily computed

Computational cost of backpropagation

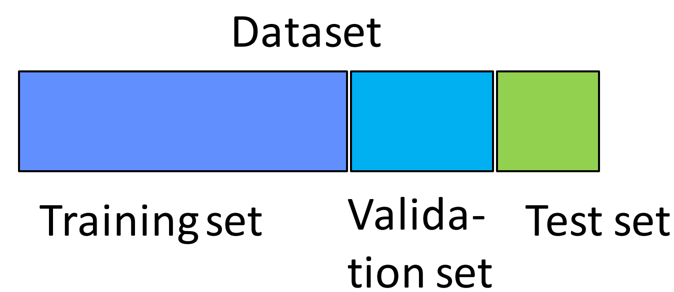


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_5}\frac{\partial a_5}{\partial z_5}\frac{\partial z_5}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

- The cost of computation increases as the network goes deeper and deeper
- Take advantage of redundancy in the calculation to reduce the overall computational cost

Training duration of the model

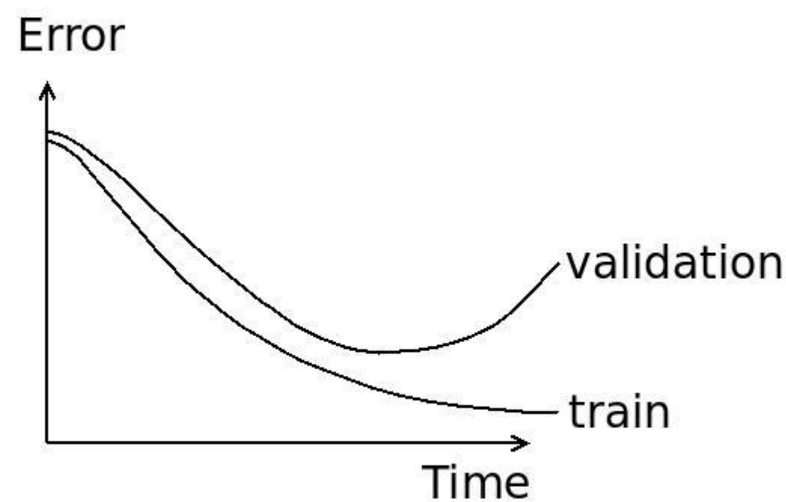
- Data is divided into 3 parts: Training set, validation set, test set
- A large enough neural network will completely fit the training data if trained for long enough
  - Overfitting; will not be generalizable



How long should the model be trained for?

One solution -

- Also use validation set for evaluating loss
  - Note: Training is done only with training set (not validation set)
- Train the model till the point the validation error starts increasing



Universal approximation theorem

- Universal approximation theorem states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any function, provided that the network is given enough hidden units
  - Squashing function example: sigmoid function, step function

- The theorem also holds for activation functions like Rectified Linear Unit (ReLU)
  - Note: ReLU is not a squashing function since positive values do not get "squashed"

Advantage of using more hidden layers

- A feedforward network with a single layer is sufficient to represent any function, but layers may be infeasibly large and may fail to generalize correctly
  - Using deeper models can reduce the number of units and amount of generalization error

- Deep network encodes a general belief that the function we want to learn should involve composition of several simpler functions
  - For example, we can a non-linear function can be approximated by several linear functions (linear functions are simpler)
- (Empirically) greater depth seem to result in better generalization

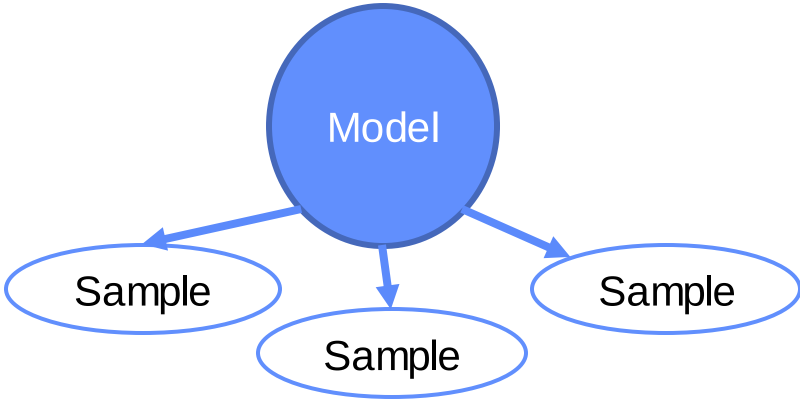
- Commonly used neural network architecture:
  - Convolutional Neural Networks
  - Recurrent Neural Networks

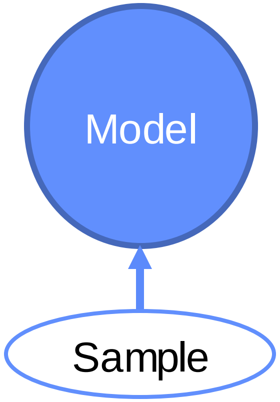
Limitations of Deep Learning

- Interpretability
- Need large amount of data
- Need computational power

Frequentist vs Bayesian View

- Frequentist: Assume repetitive sampling from population to find single value of parameter in model
- Bayesian: Assume a probability distribution of parameter and its reliability is increased by sampling data





Frequentist View of Linear Regression

- Linear model of *y* from *X*
- Optimize root mean square error for training data x_i and y_i

$$y = \beta^T X + \varepsilon$$

- Implicit assumption that β can be any value in domain

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \hat{y})^2 = \sum_{i=1}^{N} (y_i - \beta^T x_i)^2$$

Bayesian View of Linear Regression

- What if y described a probability distribution?

$$y \sim N(\beta^T X, \sigma^2 I)$$

- By Bayes theorem β describes a distribution

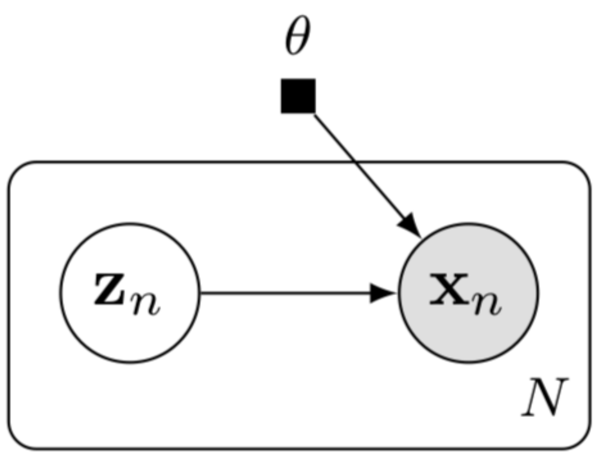
$$P(\beta|y, X) = \frac{P(y|\beta, X) * P(\beta|X)}{P(y|X)}$$

- Can encode initial guess of β with $P(\beta|X)$
- Can bias model by calculating $P(\beta|y, X)$ which considers X, Y
- Given all possible value for β in the posterior, we try those values one by one to predict the new data.
- The result is averaged proportionality to the probability of those values, hence we are taking expectation.

Bayesian Deep Learning

Deep Learning is nothing more than … or each parameter.

How does this look in PGM terms?

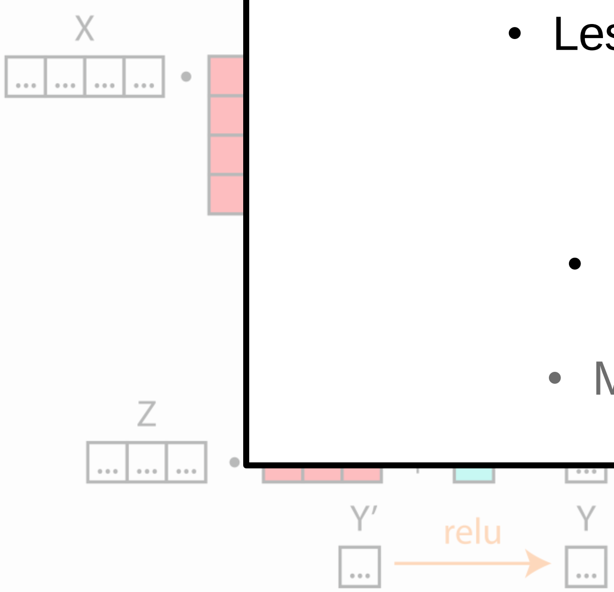


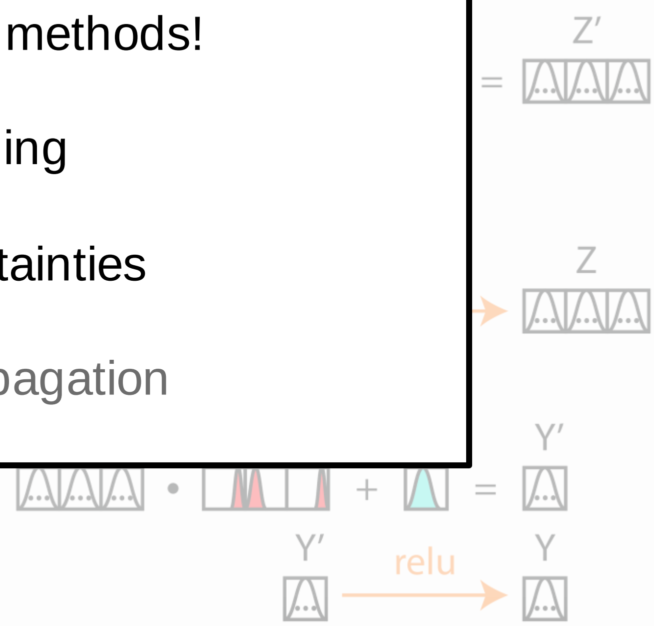
Bayesian Deep Learning

Deep Learning is nothing more than ... r each parameter.

So why would anyone want to do this?

- Less data hungry DL methods!
- Transfer Learning
- Quantifying Uncertainties
- Modeling error propagation





Transfer Learning

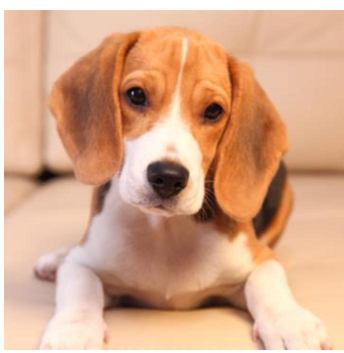
Prior information give initial “guess” to start training a dog detection network.

Cat Detection Neural Network

Cat

Not Cat

Prior Information

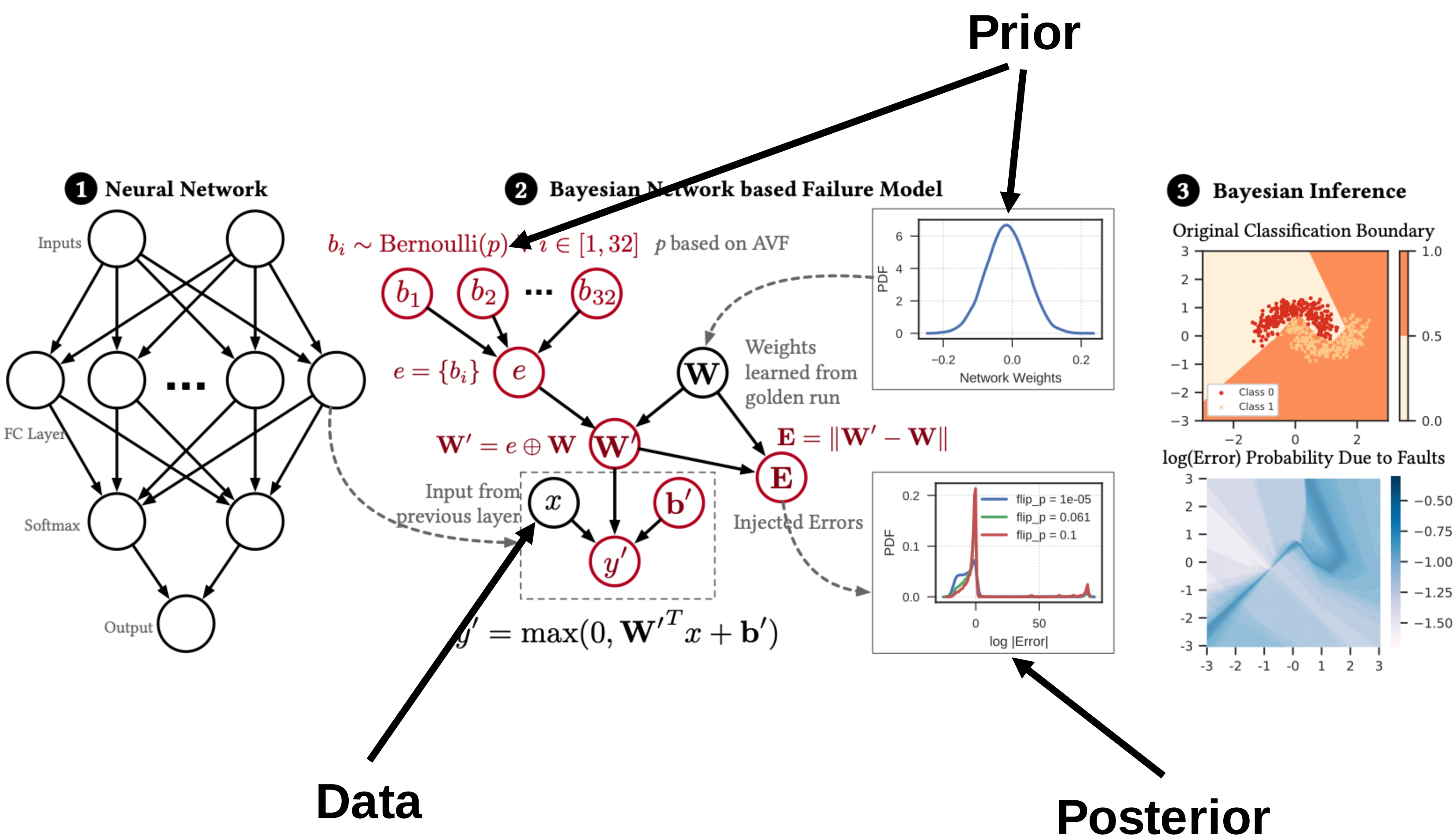


Dog Detection Neural Network

Dog

Not Dog

Fault Injection



Validation: Binary Classification Performance

You are solving a Binary Classification Problem, Predict if person has cancer (1) or not (0). You have learnt a model using your training dataset.

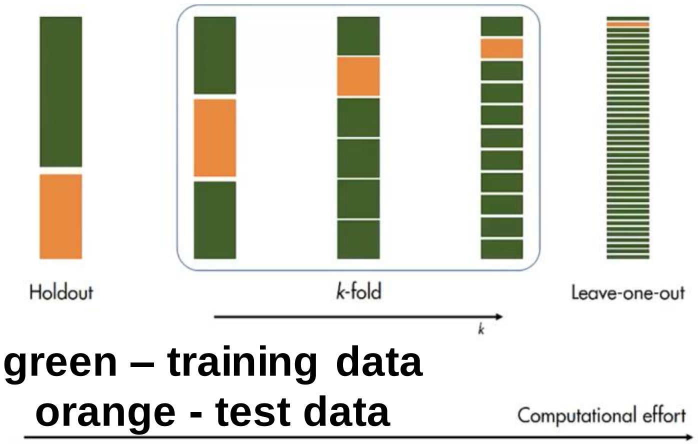
How do you test your model?

Explanation:

1) **Holdout**: Divide dataset into 2 parts, learn model on training data and validate on the test data
2) **K-fold**: Divide data into K equal parts, run the following K-times: (hold out one part of the dataset, train on the remaining, test on the holdout). Every data-sample is part of the training data K-1 times and part of the test data 1 time.
3) **Leave-one-out**: For every data sample, leave it out, train the model on the remaining data and predict for the left out data sample.

Cross Validation: For a given training iteration, divide the dataset into 2 parts:
- training set (for training the model)
- test set (for validating the model)

Different ways of doing this:
(1) Holdout
(2) K-fold
(3) Leave-one-out

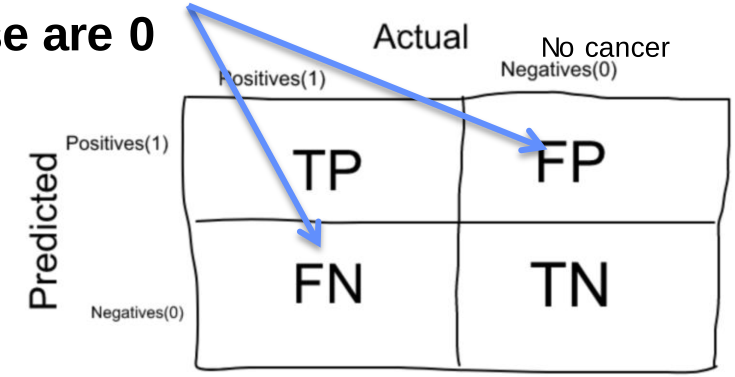


green – training data
orange - test data

What does it mean to **validate** on the test data?

Ideal if these are 0

Confusion Matrix



Depending on the **cost of making a mistake**, minimize the number of FN or FP

Common issues

Precision **=**1 if FP=0, but FN may be high
Sensitivity = 1 if predict everything as positive
Specificity = 1 if predict everything as negative
How to strike a **balance**? Use F1-score!

Metrics

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Sensitivity (Recall)} = \frac{TP}{TP + FN}$$

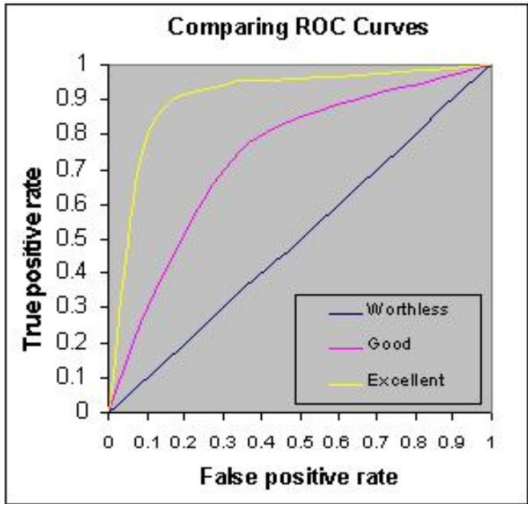
$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{F1-score} = \frac{2 * Precision * Recall}{Precision + Recall}$$

Harmonic mean of Precision and Recall (when large disparity between the 2 values, the score is closer to the smaller value)

ROC and AUC

Receiver Operating Characteristic curve: A plot of **False Positive Rate** Vs **True Positive Rate**



$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{TN + FP}$$

An ML model has parameters and for different values of the parameters, the model gives different FPR and TPR.

An **ROC (Receiver Operating Characteristic) curve** can be plotted for these different settings. A **good setting** of the parameters i.e. a good classifier, would have a **high TPR and low FPR**.

Area Under Curve (AUC):
A good classifier would have the largest area under the ROC curve.
The random predictor would have an ROC of 0.5 (the curve of the 'worthless' model in fig.)

AUC value **interpretation** –
If a pair of data samples are drawn independently, one each from the positive and negative sets, AUC gives the probability that the classifier will predict a lower score for the negative sample as compared to the positive sample

References

- Deep Learning textbook by Goodfellow et al
- https://theneural.wordpress.com/2011/07/04/hello-world/
- https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html