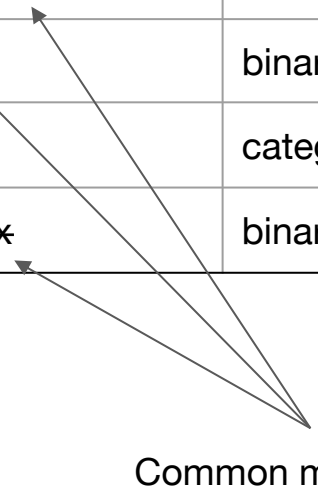
Activations and loss functions

How to choose the last layer's activation and loss in NN?

Problem type	Example	Last-layer activation	Loss
Regression	Price prediction	linear, relu	mae, mse, msle, logcosh
Regression (0 to 1)	Quality prediction	linear, relu, ~~sigmoid~~	mae, mse, ?
Binary classification	Classify cats/dogs	sigmoid, ~~linear~~	binary_crossentropy
Multiclass, single-label	Cats/dogs/horses	softmax	categorical_crossentropy
Multiclass, multilabel	Multiple labels for 1 i/p	sigmoid, ~~softmax~~	binary_crossentropy



Common mistakes

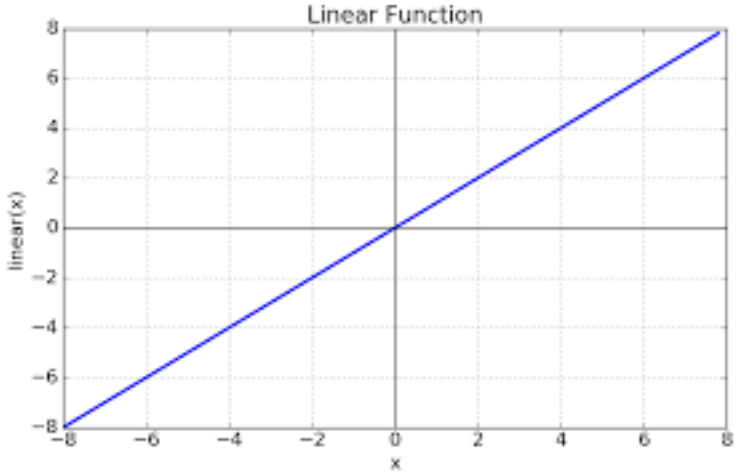
Types of supervised learning

- **Regression problems**
  - The objective is to predict continuous real values
  - Examples: predict housing price, predict image quality
- Classification problems
  - **Binary classification**
    - The objective is to predict the probability of one class
      - automatically implies the probability for the other class
    - Example: predict the probability that a image contains a human face
  - **Multi-class classification (single-label)**
    - The objective is to predict the probability of various classes
    - Example: predict whether an image has cat, dog, horse, or rat
  - **Multi-class classification (multi-label)**
    - The objective is to predict the probability of various classes independently
    - The model is supposed to separately predict the probability for each class
    - Example: predict whether an image has cat, dog, horse, rat or all of them

Activation functions

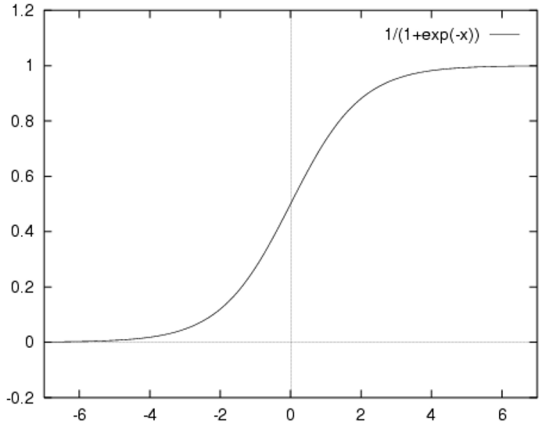
Linear activation = No activation

- No activation function = A simple linear function
  - Easy to solve
  - A linear function is just a polynomial of one degree
  - Limited in complexity
  - Have less power to learn complex functional mappings from data
- A Neural Network without activation function = Linear regression model
  - No matter how many layers you have



Sigmoid

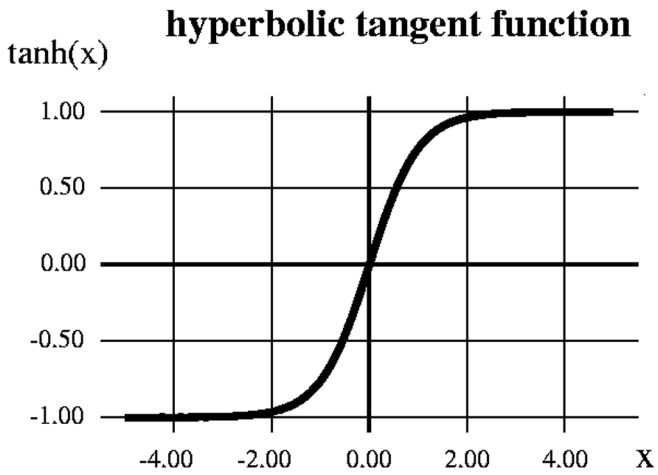
- `f(x) = 1 / 1 + exp(-x)`
  - Range is 0 to 1
  - S—shaped curve
  - Easy to understand and apply
- Problems:
  - Vanishing gradient problem
  - Its output isn't zero centered (0 < output < 1)
    - makes the gradient updates go too far in different directions
    - makes optimization harder
  - Slow convergence



Hyperbolic tangent

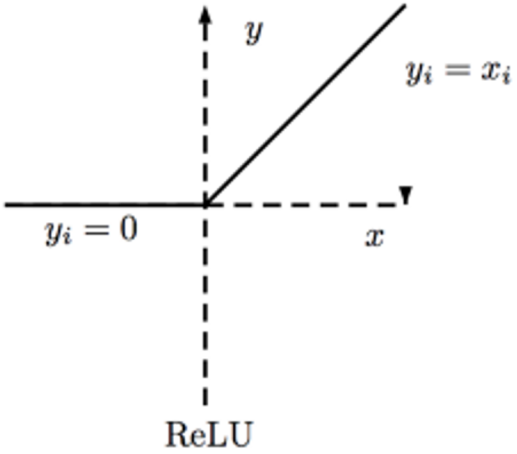
- `f(x) = 1 – exp(-2x) / 1 + exp(-2x)`
  - Range is -1 to 1
  - Output is zero centered
  - Optimization is easier
    - In practice, always preferred over Sigmoid

- Still, it suffers from vanishing gradient problem

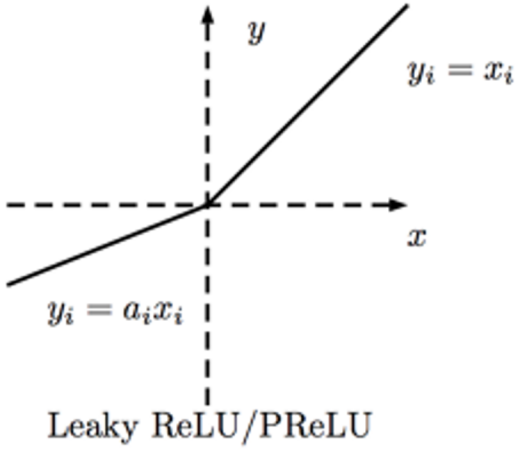


Rectified linear unit (ReLU)

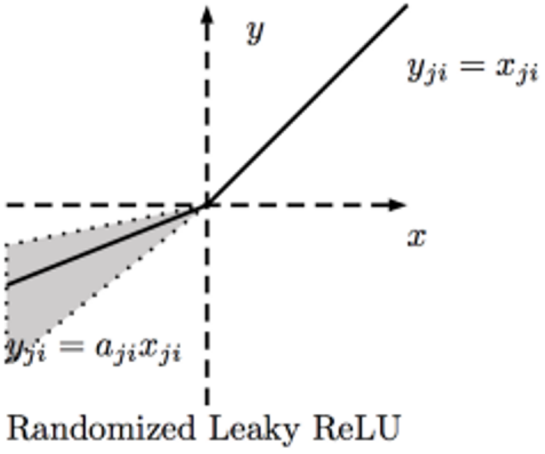
- `R(x) = max(0,x)`
  - It has become very popular in the past few years



ReLU

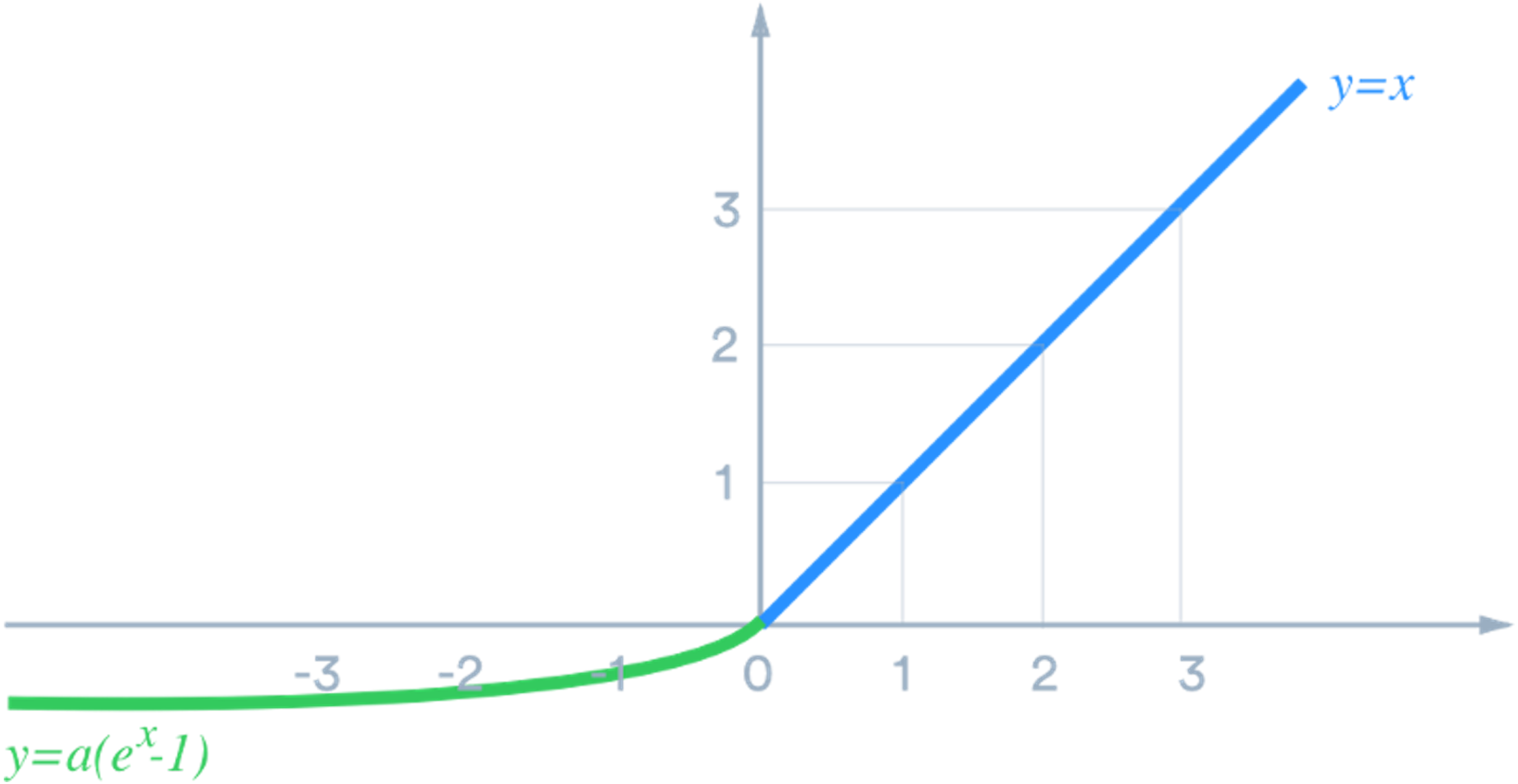


Leaky ReLU/PReLU

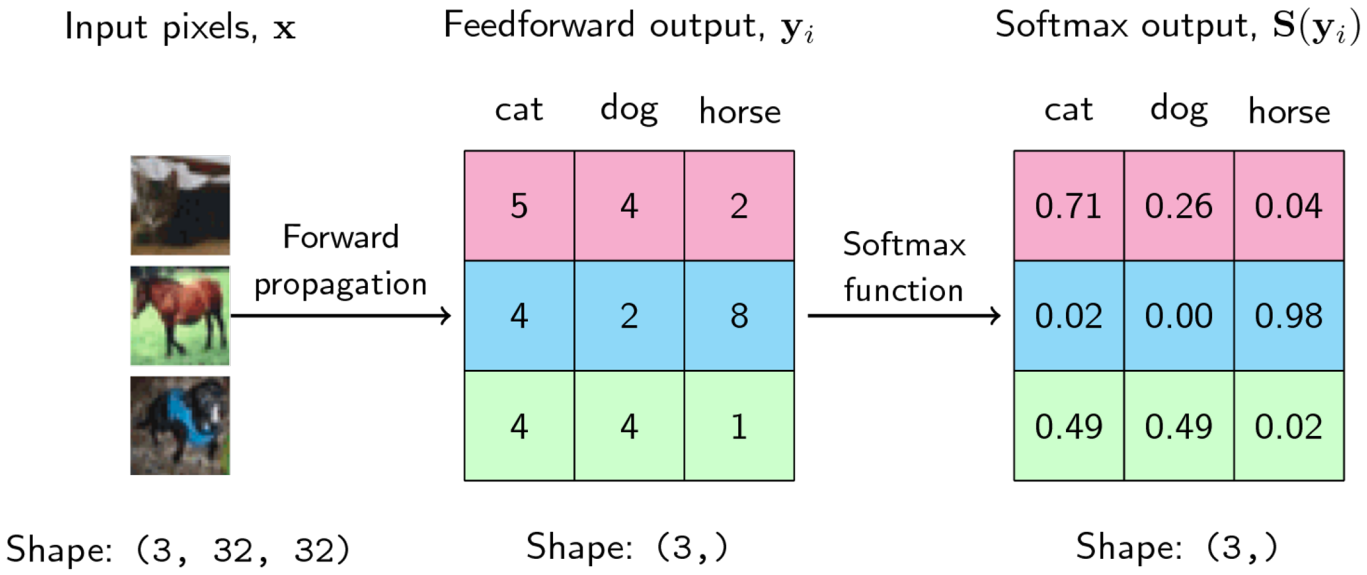


Randomized Leaky ReLU

Exponential linear unit (ELU)



Softmax



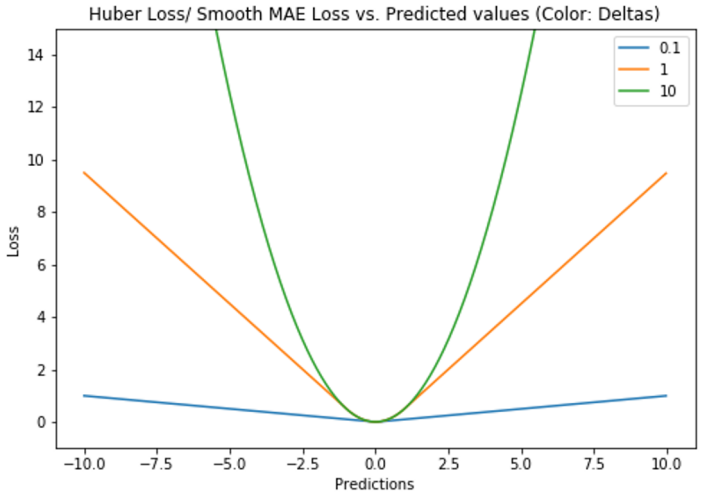
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

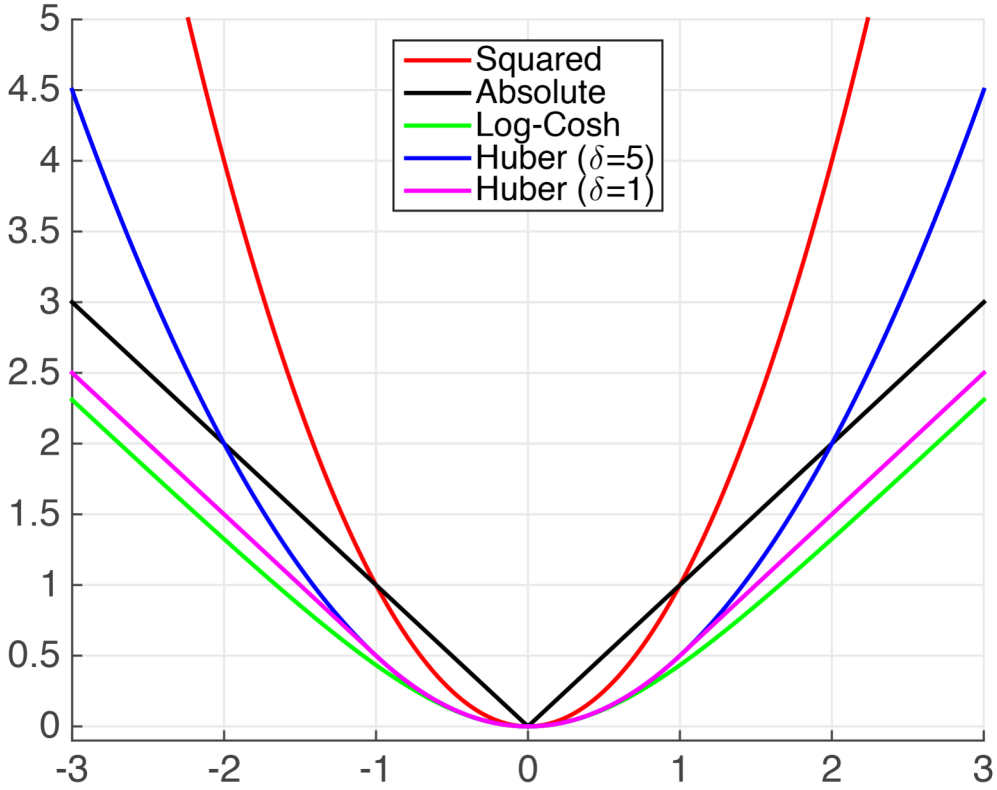
z_j is the output at node j and K is the number of classes

Loss functions & Optimizers

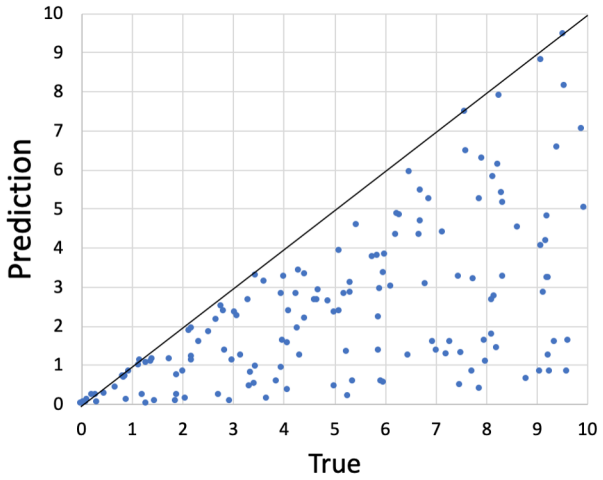
Loss functions for regression

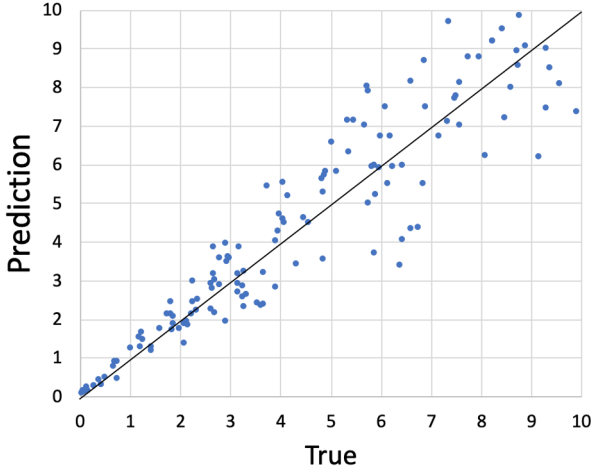
- Find all options at https://keras.io/losses/#available-loss-functions
  - mean_squared_error
  - mean_absolute_error
  - mean_squared_logarithmic_error
  - logcosh
  - huber_loss

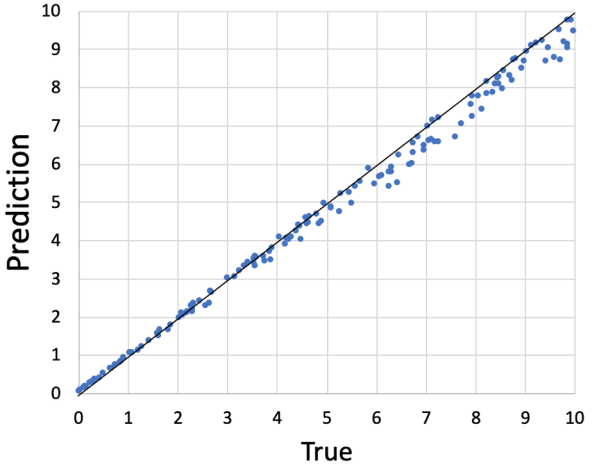




How to choose a loss function for regression?



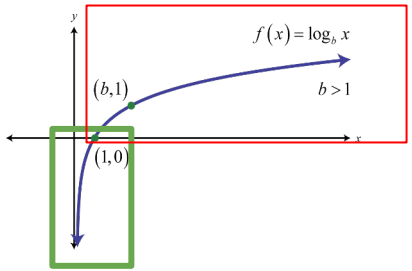


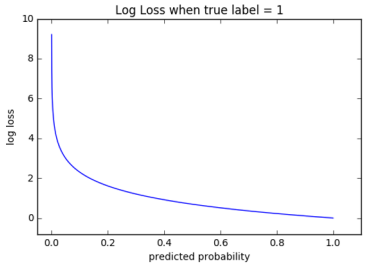


- The three plots above show three different regression models
- Which models are bad? What mistake/s are they making?
  - What loss functions are prone to which kinds of mistakes?
  - How will this change when the numbers are less than 1?
- The choice of loss function usually depends on the “domain” need
  - Do we want the model to focus on the entire range equally, or first/last part of the range?

Cross entropy (log loss) for binary classification

- Measures the performance of a classification model whose output is a ‘probability’ value between 0 and 1 (natural log)
- Log loss penalizes both types of errors, but especially those predictions that are confident and wrong!
- Loss increases as the predicted probability diverges from actual labels (0 or 1)
  - Predicting probability of .012 when the actual observation label is 1 would result in high loss value
  - Similarly predicting 0.97 when the true label is 0 results in high loss
- Negative output when the input is < 1
  - So, add a minus for a positive loss





$$-(y \log(p) + (1 - y) \log(1 - p))$$

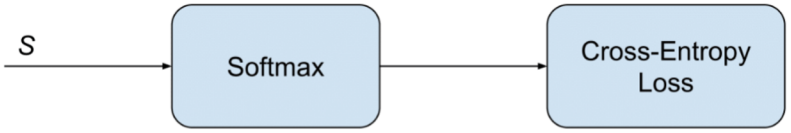
```
def binary_cross_entropy(pred, y):
            if y == 1:
                        return -log(y)
            else:
                        return -log(1-pred)
```

Categorical cross-entropy loss (or softmax loss)

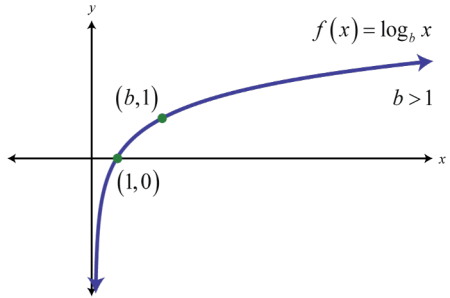
- It is a Softmax activation plus a cross-entropy loss

$$CE = -log\left(\frac{e^{sp}}{\sum_j^C e^{s_j}}\right) =$$

Sp is the positive class



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$



- Example:

True Label: Rabbit

Prediction: Dog = 1, Cat = 4, Rabbit = 8, Squirrel = 2

Softmax : D = e^1/SUM, C = e^4/SUM, R = e^8/SUM, S = e^2/SUM

CE Loss = - (0 * ln(D) + 0 * ln(C) + 1 * ln(R) + 0 * ln(S))

= - (0 + 0 + (-?) + 0)

= + ?

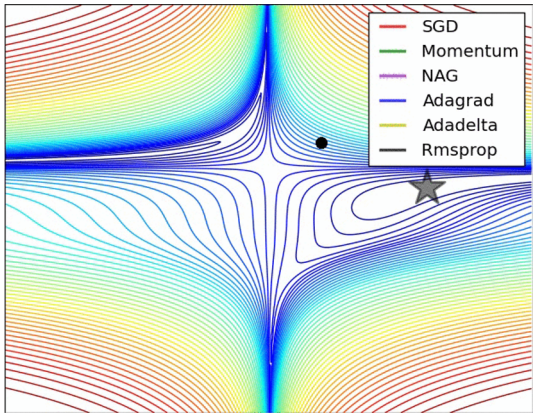
Only the positive class contributes to the CE loss!

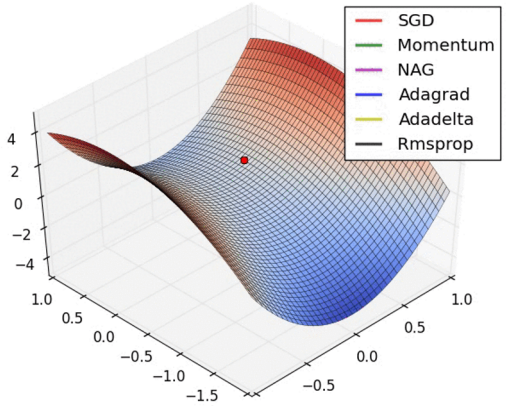
Optimizers

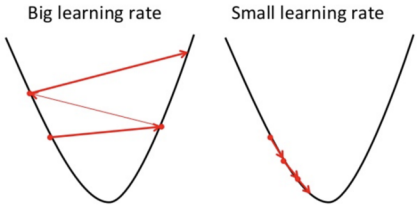
- Options (https://keras.io/optimizers/)
  - Stochastic gradient descent (sgd), RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam
- How to decide?

http://ruder.io/optimizing-gradient-descent/

  - Sparse input data - use adaptive learning-rate methods
  - Try all once with the default learning rates
  - Start with “adam” if time is a big concern







Tip (when developing methods on new datasets): Debug by observing the trends of the training loss and validation loss.