Comp 551: Machine Learning

Allan Wang

Last updated: March 26, 2019

Contents

1 Linear Regression

- Classification - discrete set output
- Regression - continuous output
- Supervised learning - given training examples with labels, find model to predict labels given input
- i.i.d assumption - training set is assumed to be *independently* and *identically distributed*
- Linear hypothesis - find weights to minimize

$$Err() = \Sigma_{i=1:n}(y_i - w^T x_i)^2 \tag{1}$$

Least-Squares Error

$$\begin{aligned} f_w &= argmin\Sigma_{i=1:n}(y_i - w^T x_i)^2 \\ \hat{w} &= (X^T X)^{-1} X^T Y \end{aligned} \tag{2}$$

- Note that both w and x vectors have an extra dimension (m + 1) for a dummy/intercept term (all 1s for x)
- Computational cost is $O(m^3 + nm^2)$ for n inputs and m features
- Only works if $X^T X$ is nonsingular (no correlation between features)
  - If a feature is a linear function of others, you can drop it
  - If the number of features exceeds number of training examples, you can reduce the number of features

Gradient Descent

$$\begin{aligned} w_{k+1} &= w_k - \alpha_k \frac{\partial Err(w_k)}{\partial w_k} \\ \frac{\partial Err(w)}{\partial w} &= 2(X^T X w - X^T Y) \end{aligned} \tag{3}$$

- Less expensive approach; weights calculated through iterations
- Goal is to reduce the weight errors from the previous iteration
- Repeat until $|w_{k+1} - w_k| < \epsilon$
- $a_k > 0$ is the learning rate for iteration k
  - If too large, oscillates forever

  - If too small, takes longer to reach local minimum

- Robbins-Monroe conditions prove convergence:

$$\begin{aligned} \Sigma_{k=0:\infty}\alpha_k &= \infty \\ \Sigma_{k=0:\infty}\alpha_k^2 &< \infty \end{aligned} \tag{4}$$

- Not that convergence is to local minimum only, not always global

---

- Feature design - if features cannot fully represent a model, we can transform them using non linear functions (eg powers, binary thresholds, etc) into new features. Note that the weights are still linear combinations.
- Overfitting - hypothesis explains training data well, but does not generalize to new data; high variance, low bias
- Underfitting - does not capture trend; low variance, high bias
- Simple model - high training error, high test error
- Complex model - low training error, high test error
- Training error always goes down with complexity, but at some point in between, test error will be at its lowest

Validation

- Validation set should be separate from training data
- K-Fold cross validation
  - Create k partitions for available data
  - For each iteration, train with $k-1$ subsets, then validate on remaining subset. Repeat k times
  - Return average prediction error
- Leave-One-Out Cross Validation
  - K-fold where $k = n$

2 Linear Classification

Binary Classification

- Probabilistic - estimate conditional probability $P(y|x)$ given feature data
- Decision boundaries - partition feature spaces into different regions

Discriminative Models

- Directly estimate $P(y|x)$
  - Answers what what the features tell us about the class
  - Difficult to estimate
- Log-odds ratio

$$a = ln\frac{P(y=1|x)}{P(y=0|x)} \tag{5}$$

  - Outputs likelihood of $y = 1$ vs $y = 0$
  - Decision boundary is set of points for which $a = 0$
- Logistic function - predicted probability for $y = 1$

$$\sigma(w^Tx) = \frac{1}{1+e^{-w^Tx}} \tag{6}$$

- Likelihood

$$\begin{aligned} L(D) &= P(y_1, y_2, ..., y_n|x_1, x_2, ..., x_n, w) \\ &= \prod_{i=1}^{n} \sigma(w^Tx_i)^{y_i}(1-\sigma(w^Tx_i))^{1-y_i} \end{aligned} \tag{7}$$

  - Numerically unstable for lots of small numbers
- Log-Likelihood

$$\begin{aligned} l(D) &= ln(L(D)) \\ &= \Sigma_{i=1}^{n} y_i ln(\sigma(w^Tx_i)) + (1-y_i)ln(1-\sigma(w^Tx_i)) \end{aligned} \tag{8}$$

  - Easier to optimize
  - Negative log-likelihood = cross-entropy loss
    Maximizing log-likelihood = minimizing cross-entropy loss

- Many kinds of losses exist, eg absolute error, or binary; however, these losses are not always easy to optimize (not differentiable)
- Gradient descent update rule

$$w_{k+1} = w_k + \alpha_k \Sigma_{i=1:n} x_i (y_i - \sigma(w_k^T x_k)) \tag{9}$$

Generative Models

- Use Bayes' rule to estimate

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x)} \tag{10}$$

  - Finds the marginal probability of a class
  - Easy to estimate

- Linear Discriminant Analysis (LDA)

$$P(x|y) = \frac{e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}}{(2\pi)^{\frac{1}{2}} |\Sigma|^{\frac{1}{2}}} \tag{11}$$

  - Every class assumed to be Gaussian/normally distributed
  - Both classes have same covariance matrix Σ, but different means μ
  - Estimations

$$P(y=0) = \frac{N_0}{N_0 + N_1}$$
$$P(y=1) = \frac{N_1}{N_0 + N_1}$$

$$\mu_0 = \frac{\Sigma_{i=1:n} I(y_i = 0) x_i}{N_0}$$
$$\mu_1 = \frac{\Sigma_{i=1:n} I(y_i = 1) x_i}{N_1} \tag{12}$$

$$\Sigma = \frac{\Sigma_{k=0:1} \Sigma_{i=1:n} I(y_i = k)(x_i - \mu_i)(x_i - \mu_k)^T}{N_0 + N_1 - 2}$$

    * N_0 and N_1 are the # of training samples from classes 1 and 0 respectively
    * $I(x)$ is an indicator function: $I(x) = 0$ if $x = 0$, $I(x) = 1$ if $x = 1$

- Cost is $O(m^2)$ for m classes

- Quadratic Discriminant Analysis (QDA)
  - Allows different covariance matrices, Σ_y for each class y
  - Cost is $O(nm^2)$ for m classes and n features

- Naïve Bayes
  - Assumes x_j is conditionally independent given y

$$P(x_j|y) = P(x_j|y, x_k)\forall j, k \tag{13}$$

  - Still one Σ_y per class, but they are diagonal
  - Cost is $O(nm)$ for m classes and n features

- Laplace smoothing
  - Replace maximum likelihood estimator:

$$\begin{aligned}
&\text{Before:}\\
&Pr(x_j|y=1) = \frac{(\text{number of instances with } x_j = 1 \text{ and } y = 1)}{(\text{number of examples with } y = 1)}\\
&\text{After:}\\
&Pr(x_j|y=1) = \frac{(\text{number of instances with } x_j = 1 \text{ and } y = 1) + 1}{(\text{number of examples with } y = 1) + 2}
\end{aligned} \tag{14}$$

  - Allows data we previously did not encounter to have a probability greater than 0

3 Evaluation

Evaluating Classification

- True positive (m11) - expect 1, predict 1
- False positive (m01) - expect 0, predict 1; type I error
- True negative (m00) - expect 0, predict 0
- False negative (m10) - expect 1, predict 0; type II error

- $$\begin{array}{ll} \text{Error rate} & \frac{m01 + m10}{m} \\ \text{Accuracy} & \frac{TP + TN}{ALL} \\ \text{Precision} & \frac{TP}{TP + FP} \\ \text{Recall/Sensitivity} & \frac{TP}{TP + FN} \\ \text{Specificity} & \frac{TN}{FP + TN} \\ \text{False positive rate} & \frac{FP}{FP + TN} \\ \text{F1 measure} & F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \end{array} \tag{15}$$

- Receiver operating characteristic (ROC)
  - Plot true negative and true positive prediction rates based on a moving boundary
  - Comparison done by looking at area under ROC curve (AUC)
  - In a perfect algorithm, $AUC = 1$; for random algorithms, $AUC = 0.5$

Evaluating Regression

- Mean Square Error
$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{16}$$
- Root Mean Square Error
$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{17}$$
- Mean Absolute Error
$$MAE = \frac{1}{n}\Sigma_{i=1}^{n}|\hat{y}_i - y_i| \tag{18}$$

4 Regularization

- High bias - simple model
- High variance - overly complex model
- Regularization

  - Reduce overfitting
  - Reduce variance at the cost of bias

L2 Regularization

- Aka ridge regression

$$\begin{aligned}\hat{w}^{ridge} &= argmin_w(\Sigma_{i=1:n}(y_i - w^T x_i)^2 + \lambda\Sigma_{j=0:m} w_j^2) \\ &= (X^T X + \lambda I)^{-1} X^T Y\end{aligned} \tag{19}$$

  - λ can be selected manually or by cross validation
  - Not equivariant under scaling of data; typically need to normalize inputs first
  - Can also modify penalty by adding penalty term $2\lambda w$

$$\frac{\partial Err(w)}{\partial w} = 2(X^T X w - X^T Y) + \underline{2\lambda w} \tag{20}$$

  - Tends to lower all weights

L1 Regularization

- Aka lasso regression

$$\hat{w}^{lasso} = argmin_w(\Sigma_{i=1:n}(y_i - w^T x_i)^2 + \lambda\Sigma_{j=1:m} |w_j|) \tag{21}$$

- More computationally expensive than L2
- Sets the weights of less relevant input features to 0
- Can also modify penalty by adding penalty term $\lambda sign(w)$ where
  - $sign(x) = 1$ if $x > 0$
  - $sign(x) = 0$ if $x = 0$
  - $sign(x) = -1$ if $x < 0$

- Elastic net combines L2 and L1

5 Decision Trees

- Nodes represent partitions
- Internal nodes are tests based on different features
  - Typically branch on all possibilities for discrete features
  - Typically branch on threshold values for continuous features
- Leaf nodes include set of training examples satisfying all tests along the branch
- Can express every boolean function; goal is to encode functions compactly
- Typically constructed sing recursive top-down procedure + pruning to avoid overfitting
- Advantages: fast, easy to interpret, accurate
- Disadvantages: sensitive to outliers, bad for learning functions with smooth, curvilinear boundaries
- Approach
  1. If training instances have same class, create leaf with that class label and exit
  2. Else, pick best test to split on
  3. Split training data
  4. Recurse on 1-3 for each subset

Finding Best Test

- If event E with probability $P(E)$ has occurred with certainty, we received $I(E)$ bits of information, where
$$I(E) = log_2 \frac{1}{P(E)} \tag{22}$$
- Events with smaller probabilities produce more bits of information
- Entropy (H(S))- average amount of information from source S, which emits symbols $\{s_1, ..., s_k\}$ with probabilities $\{p_1, ..., p_k\}$
$$H(S) = \Sigma_i p_i I(s_i) = -\Sigma_i p_i log_2 p_i \tag{23}$$
- Conditional entropy
$$H(y|x) = \Sigma_v P(x = v) H(y|x = v) \tag{24}$$

- Information gain - reduction in entropy obtained by knowing x

$$IG(x) = H(D) - H(D|x) \tag{25}$$

- Binary Classification
  - Given p positive samples and n negative samples:

$$H(D) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n} \tag{26}$$

  - Example: Given 40 examples with 30 positive and 10 negative, as well as a test T that divides to [15+, 7-] for true and [15+, 3-] for false, we obtain entropy

$$H(D|T) = \frac{22}{40}\left[-\frac{15}{22} log_2 \frac{15}{22} - \frac{7}{22} log_2 \frac{7}{22}\right] + \frac{18}{40}\left[-\frac{15}{18} log_2 \frac{15}{18} - \frac{3}{18} log_2 \frac{3}{18}\right] = 0.788 \tag{27}$$

- For classification, choose test with highest information gain
- For regression, choose test with lowest MSE
- For real-valued features, can select possible thresholds based on midpoints of data values

Avoiding Overfitting

- Remove some nodes for better generalization
- Early stopping - stop growing tree if splits do not improve information gain of validation step
- Post pruning - grow full tree then remove lower nodes with low information gain on validation set (generally better)
  - For each node:
  - Evaluate validation set accuracy if subtree is pruned
  - Greedily remove nodes that most improve validation set accuracy
  - Replace removed node by leaf with majority class of corresponding examples
  - Stop when pruning hurts validation set accuracy

6 Features

- Strategies
  - Domain knowledge to construct "add hoc" features
  - Normalization
  - Non-linear expansions
  - Regularization
- NLP Features
  - Words (binary, absolute frequency, relative frequency)
  - TF-IDF
  - N-grams
  - Syntactic features
  - Word embeddings
  - Stopwords (common words that may not provide much information)
  - Lemmatization (inflectional morphology)
- Approaches
  - Wrapper & filter - applied during preprocessing
  - Embedded - integrated in optimization method (eg regularization)

Principal Component Analysis (PCA)

- Project data into lower-dimensional sub-spaces
- Solves
$$argmin_{W,U}\Sigma_{i=1:n}\left\|X - XWU^T\right\|^2 \tag{28}$$
- Solution W given by eigen-decomposition of X^TX; columns are orthogonal

Variable Ranking

- Rank features using scoring features; select the highest ranked features
- Simple & fast, but requires a scoring function
- Scoring functions

- Correlation
- Mutual information (finds nonlinear relationships)

7 Instance Learning

- Parametric supervised learning
  - Assumes we have labelled examples
  - Learns parameter vector of fixed size
- Non-parametric learning
  - Store all training examples $\langle x_i, y_i \rangle$
  - With new query, compute value based on most similar points
  - Requires distance function (eg euclidian distance)
    - Often domain specific
    - May require feature preprocessing
    - Can sometimes be learned

One-Nearest Neighbour

- Lazy learning - generalization upon query
- Requires no learning (just store data)
- For new point x_{new}, find nearest sample $x_{i^*} = argmin_i d(x_i, x)$ and predict $y_{new} = y_{i^*}$
- Essentially results in a decision boundary where each boundary is equidistance between two points of opposite classes

K-Nearest Neighbour (KNN)

- Requires no learning (just store data)
- For new point x_{new}, find k nearest training samples
- For regression, predict mean/median of neighbouring y values
- For classification, predict majority or empirical probability of each class
- Low k results in low bias and high variance
- High k results in high bias and low variance
- Sensitive to small variations

Distance-Weighted (kernel-based) NN

- For new point x_{new}, compute $w_i = w(d(x_i, x_{new}))$ for all x_i in training data, where d is a weighting function and predict $y = \frac{\Sigma_i w_i y_i}{\Sigma_i w_i}$
- Example of a weighting function is the inverse euclidean distance
- For gaussian weighting, bigger σ results in more variance

- Instance based learning is useful when
  - A good distance metric exists
  - There aren't too many attributes per instance (otherwise distances will be similar, leading to noise)

8 Support Vector Machines (SVM)

Perceptron

$$h_w(x) = sign(w^T x) \tag{29}$$

- Outputs +1 if $w^T x \geq 0$, -1 otherwise
- Decision boundary is $w^T x = 0$
- Gradient-descent learning

$$Err(w) = \Sigma_{i=1:n} \left\{0 \text{ if } y_i w^T x_i \geq 0; -y_i w^T x \text{ otherwise}\right\} \tag{30}$$

  - Error is 0 for correctly classified example
  - Error shows how much to shift for incorrectly classified example
  - Error is 0 if all examples are classified correctly

- Learning converges within finite number of updates for linearly separable datasets; otherwise, there will be oscillation
- Dual representation

$$w = \Sigma_{i=1:n} \alpha_i y_i x_i \tag{31}$$

  - α_i is sum of step sizes used for all updates applied to example i

- Solution often non-unique; depends on order of updates

Linear SVM

- Choose w such that margin is maximized
- Margin is twice the Euclidean distance from hyper-plane to nearest training example
- Let γ_i be the distance from x_i to decision boundary, connected at point x_i^0. Let w be the normal to the decision boundary. We can define:

$$x_i^0 = x_i - \frac{\gamma_i w}{\|w\|} \tag{32}$$

  - $\frac{w}{\|w\|}$ is the unit normal
  - γ_i is the scalar distance from x_i to x_i^0

- Goal is to minimize $\frac{1}{2}\|w\|^2$ with respect to w such that $y_i w^T x_i \geq 1$
- \\TODO Lagrangian optimization; 439
- Support vectors - points lying on edge of margin, affecting the decision boundary
- \\TODO Hinge loss; 461
- Multiple classes
  - One-vs-all - learn K separate binary classifiers
    - Can be inconsistent
    - Training sets are imbalanced
  - Multi-class SVM

9 Ensembles

Run base algorithms multiple times, then combine predictions for a final prediction

Bagging

- Create K independent models by training same base algorithm on different subsets of training data
- Reduces variance, increases bias
- Good for "reasonable" classifiers
- Bootstrapping - cross-validation but using random sampling with replacement

  - For new dataset, randomly select value from full dataset with replacement until size is sufficient

- Incremental construction
  - Hypotheses should not be constructed independently
  - New hypotheses should focus on problematic instances
- Random forest
  - Use K bootstrap replicates to train K different trees
  - For each node, pick m variables at random (where $m < M$, M being the total number of features)
  - Determine best test using normalized information gain
  - Recurse until depth is reached without pruning
  - Resulting trees have high variance, but ensemble uses average, which reduces variance
  - Fast & can handle lots of data
  - Can circumvent difficulties in optimization
  - Low interpretability
  - New predictions may be more expensive (need to go through all trees)

Boosting

- Incrementally train K models, where each successive model tries to fix mistakes of previous models
- Reduces bias & increases classification margin
- Tends not to overfit
- Good for very simple classifiers
- Problematic if lots of data is mislabelled
- Algorithm
  - Train simple predictor
  - Re-weight training examples, putting more weight on those that were not properly classified
  - Repeat n times
- Adaboost adapts error rate to reduce training error exponentially fast

Stacking

- Stacking - train K different models and combine their output
- Predictions from base models become features of new meta-model
- Works well when base models have complimentary strengths and weaknesses

10 Neural Networks

- A single perceptron can represent linear boundaries, but cannot represent non-linearly separated functions like XOR. However, stacking perceptrons to create new datasets allows more functions to become linearly separable
- Sigmoid function provides "soft threshold", whereas perceptron provides "hard threshold"
- Feed-forward neural network
  - Connections from one layer to the next in one direction; no connections between units of the same layer
  - Hidden units are $h_i = \sigma(w_i^T x + b_i), \forall i$
  - Unlike stacking, we jointly train the entire network
  - Can have multiple output units
  - Depth - number of layers
  - Fully-connected - if all units in layer j provide input to all units in layer $j + 1$
- Neural network with two hidden layers can approximate any function to arbitrary accuracy
- Activation function can be an arbitrary non-linear function, not just limited to sigmoid

Gradient Descent

- Used to minimize errors, as hypothesis is differentiable and too complex to compute optimal weights directly
- Stochastic gradient descent
  1. Initialize weights to small random numbers
  2. Pick single training example x
  3. Feed example then compute correction

4. Compute correction for output unit

$$\frac{\partial J}{\partial w_{out}} = \delta_{out} x \tag{33}$$

5. Compute correction for each hidden unit j

$$\frac{\partial J}{\partial w_j} = \delta_{out} w_{out,j} \sigma(w_j^T x + b)(1 - \sigma(w_j^T x + b))x \tag{34}$$

6. Update each network weight

$$\begin{aligned} w_j &= w_j - \alpha \frac{\partial J}{\partial w_j} \forall j \\ w_{out} &= w_{out} - \alpha \frac{\partial J}{\partial w_{out}} \end{aligned} \tag{35}$$

7. Repeat from step 2 until convergence

- Batch gradient descent
  - Compute error on all examples
  - Loop through training data, accumulating weight changes
  - Update all weights and repeat
- Mini-batch gradient descent
  - Compute error on randomly selected subset

11 Backpropagation

Activation Functions

- Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{36}$$

  - Easy to differentiate
  - Easily saturates; inputs outside of [-4, 4] remain essentially constant, making it hard to train

- Tanh

$$\phi(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{37}$$

  - Easy to differentiate $(1 - tanh^2(z))$
  - Slightly less prone to saturation than sigmoid

- ReLU

$$ReLU(z) = max(z, 0) \tag{38}$$

  - Unbounded range; never saturates
  - De facto standard

- Softplus

$$softplus(z) = ln(1 + e^x) \tag{39}$$

  - Similar to ReLU but smoother
  - Expensive derivative compared to ReLU

Derivatives

- Tell us how much we impact another node if we change one node by one unit
- For non-neighbouring nodes, sum over all possible paths from one node to the other, multiplying derivatives on each edge of the path
- Computationally intensive, so we can instead factor paths by summing derivatives of all inputs to a node first
- Forward mode - start from source, and at each node, sum all incoming edges/derivatives
- Reverse mode - start from sink, and at each node, sum all outgoing edges/derivatives
- Both only touch each edge once, but since we want derivatives of the output/loss with respect to all previous layers, reverse mode is better
- Reverse-mode automatic differentiation (RV-AD) = backpropagation

---

- Guaranteed convergence to local minimum
- Use random restarts to find better minimum
- Overtraining occurs if weights take on large magnitudes; # of training updates is also a hyper-parameter
- Adaptive optimization algorithms change learning rates for each parameter based on history of gradient updates

Momentum

On i^{th} gradient descent, add momentum (second term)

$$\Delta_i w = \alpha \frac{\partial J}{\partial w} + \beta \Delta_{i-1} w \tag{40}$$

- Allows us to pass small local minima
- Keeps weights moving if error is flat
- However, can get out of a global maximum
- Is tunable, increasing chance of divergence

12 Convolutional Neural Network (CNN)

- Often used in computer vision
- Input usually 3D tensor for 2D RGB image
- Local receptive fields
  - Hidden unit connected to patch of input image
  - Unit connected to all 3 colours channels
- Share matrix of parameters across units
  - Units within depth slice at all positions have same weights
  - Feature map computed via discrete convolution with kernel matrix
- Pooling/subsampling of hidden units in same neighbourhood
  - Dimensionality reduced, making features more high-level & abstract
- Convolutional "kernels" aim to extract useful information (eg blurring/edge detection)
- Can have multiple layers, vary width/size of receptive fields, and vary stride (spacing between receptive fields)
- Pooling layers often added between convolutional layers; takes max of inputs in receptive fields
- Dropout regularization
  - Helps generalize better & reduce overfitting
  - Independently set each hidden unit to zero with probability p (often $p = 0.5$)

- Batch normalization
  - Compute mean & variance independently for each dimension and normalize each input
  - Usually done already at input layer, but this extends it to other layers
  - Results in more stable gradient estimates
- Can use softmax instead of sigmoid for multi-class classification (not the same as multi-output classification)

Recurrent Neural Network (RNN)

- Often used in text & speech
- Allows us to retain temporal information in sequences
- Add cycles with time delay (affects next step, to avoid infinite loops); see recurrence below [12]
- Output types
  - One at the end of the sequence (eg sentiment classification)
  - One at each time step (eg language generation)
    - Use softmax
    - Usually add an "end of sentence" token for stopping
- One-hot vectors
  - Map each value to column of a weight matrix
  - Removes bias associated with categorical value
- Train using backpropagation through time (BPTT)
  - Same as regular backpropagation, but we unroll the computation graphto find the flow through all layers
  - Can truncate flow by using only last k time steps

Recurrence

Elman RNN

$$\begin{aligned} h_t &= \sigma(Wh_{t-1} + Ux_t + b) \\ o_t &= \phi(Vh_t + c) \end{aligned} \tag{41}$$

Jordan RNN

$$\begin{aligned} h_t &= \sigma(W o_{t-1} + U x_t + b) \\ o_t &= \phi(V h_t + c) \end{aligned} \tag{42}$$

- Elman usually better, as output is often constrained, resulting in information loss

Long Short-Term Memory (LSTM)

- Long-term dependencies are hard to learn. Given that the transition matrix W is the same for each step, a small offset from 1 will cause gradients to explode or vanish
- Can avoid exploding gradients through gradient clipping. If gradient magnitude is better than threshold, set it to $threshold * sign(gradient)$
- Can also use non-multiplicative interactions like LSTM (additive)
- LSTM has a hidden state (past info for next prediction) and cell state (past info for future predictions)
- Uses gates to control information flow
- Input gate - what info from current input & previous hidden state do we want to transfer to cell state?

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{43}$$

- Cell update - additive linear combination of old cell state & processed input

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{44}$$

- Output gate - what information from cell state do we need for next prediction?

$$\begin{aligned} o_t &= \sigma(W_o \, [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{45}$$

- Use bidirectional LSTMs to increase expressiveness; apply both left-ro-right and right-to-left
- Can also add attention by adding connections from encoder hidden states to context vector

- To produce output of a different length from input, use two RNNs. 'Encoder' RNN transforms input to vector. 'Decoder' RNN generates sequence from vector. Vector is often referred to as the "context" vector, and as used as input to the decoder at every timestep, along with the output at the previous timestep.
- Teacher forcing - train model by inputting ground-truth instead of input from previous iteration. No change done for testing as we don't have labels.