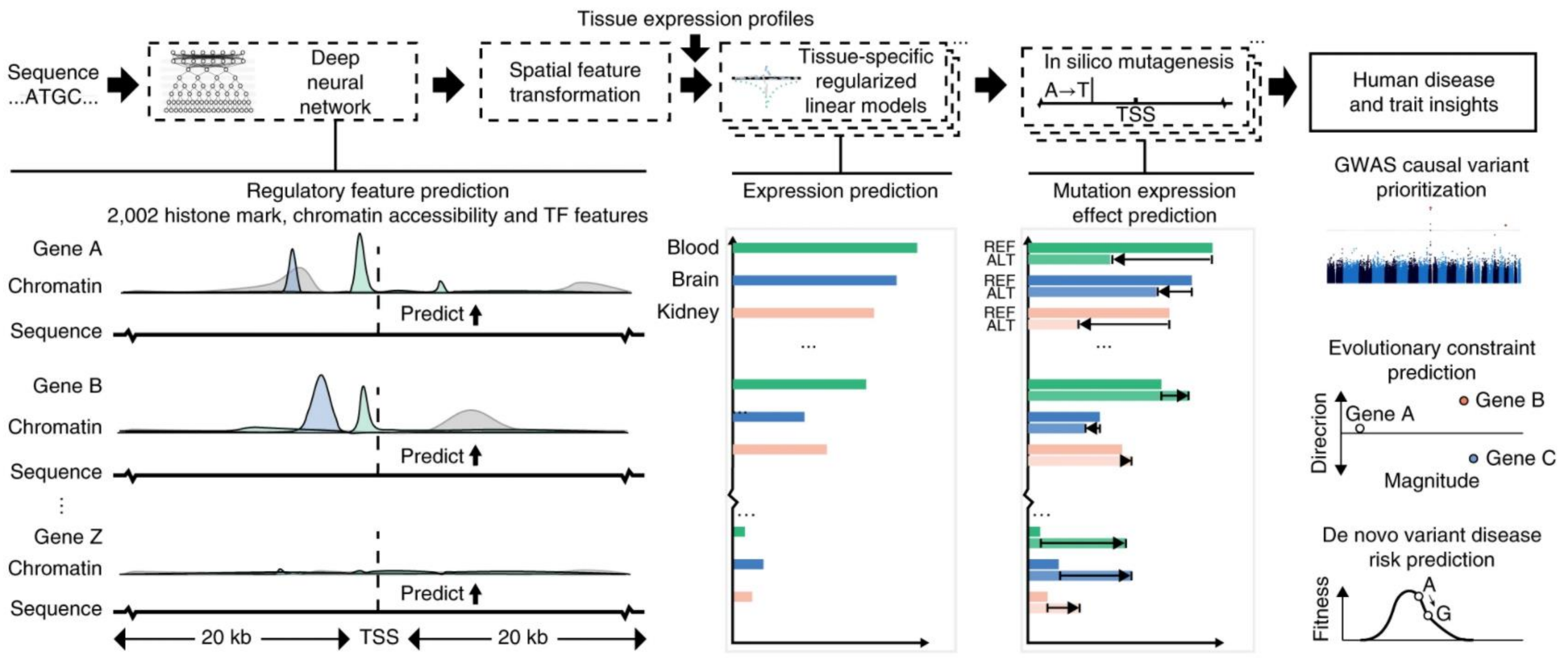
Towards More Realistic Simulated Datasets for Benchmarking Deep Learning Models in Regulatory Genomics

2022/03/02

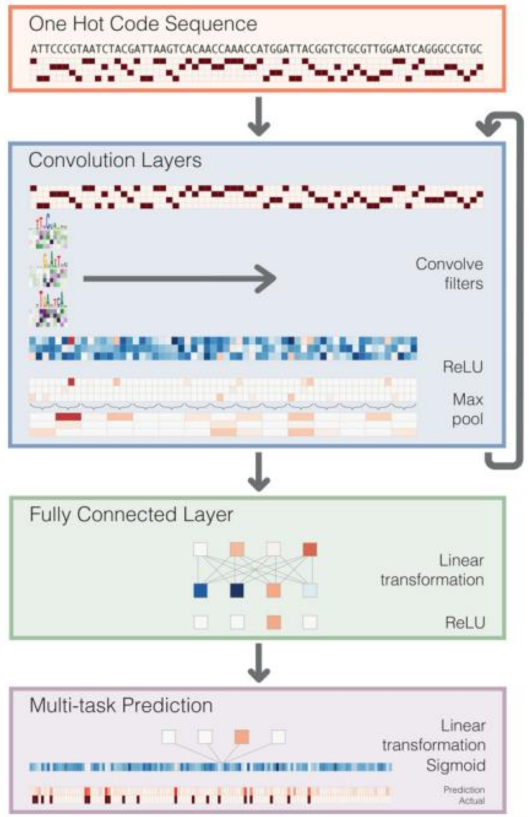
Ping-Han Hsieh

Models

DeepSEA Beluga



Basset



Training Results

		A549	GM12878	H1	HepG2	K562
DeepSEA Beluga	auROC:	0.812	0.800	0.845	0.795	0.824
	auPRC:	0.475	0.403	0.505	0.417	0.535
Basset	auROC:	0.785	0.767	0.806	0.777	0.783
	auPRC:	0.443	0.364	0.462	0.417	0.475
Fraction of Positives		0.134	0.0845	0.0947	0.129	0.156

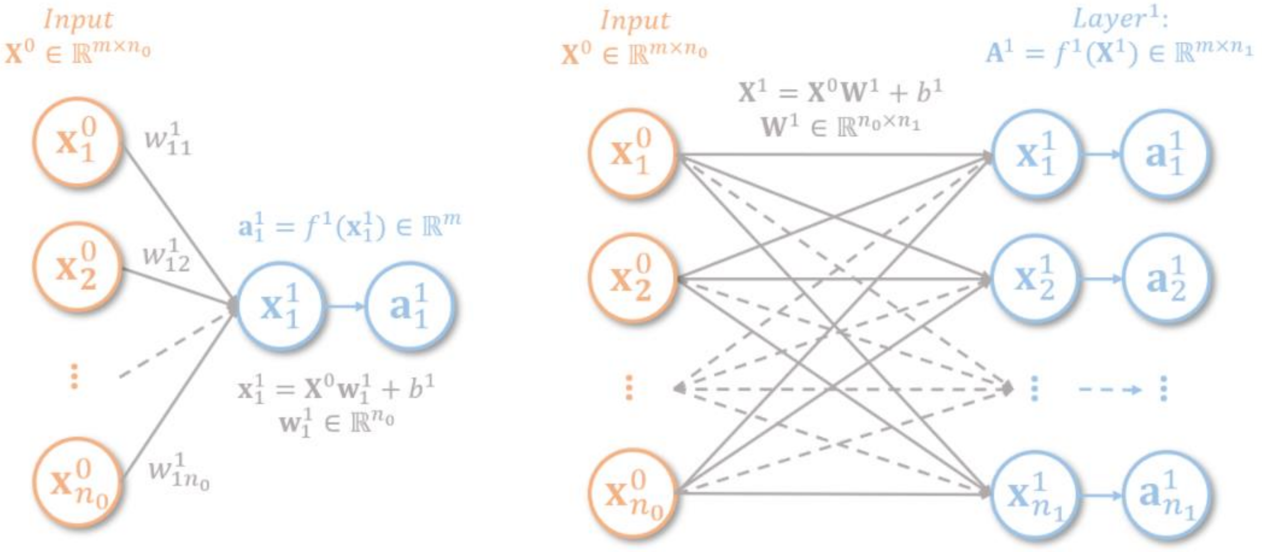
- learning rate = 0.001, batch size = 300, binary cross entropy loss
- Adam optimizer, maximum number of training batches = 30,000 (w/ early stopping)

Deep Neural Network (1)

- The multilayer perceptron consists of several layers of operations. The output of each **neuron** in the layer is the **linear combination** of the input tensor followed by an **activation function**.

$$\mathbf{a}_1^1 = f^1(\mathbf{X}^0\mathbf{w}_1^1 + b^1)$$

- The optimal weight can be approximated using **backpropagation** and **stochastic gradient descent**.



(a) Linear Transformation and Activation (b) Stacked Operations for *Layer*^{1}

Backpropagation (1)

- Suppose we have a very simple neural network with **one hidden layer** followed by a **sigmoid activation**. With **mean squared error**.

$$\hat{\mathbf{y}} = \sigma(\hat{\mathbf{X}}), \quad \hat{\mathbf{x}} = \mathbf{X}\mathbf{w} + b \quad L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- Consider the gradient of the mean squared error with respect to the prediction.

$$\nabla_{\hat{\mathbf{y}}} L(\mathbf{y}, \hat{\mathbf{y}}) = \begin{bmatrix} \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{y}_1} \\ \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{y}_2} \\ \vdots \\ \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{y}_m} \end{bmatrix} = \begin{bmatrix} -\frac{2}{m}(y_1 - \hat{y}_1) \\ -\frac{2}{m}(y_2 - \hat{y}_2) \\ \vdots \\ -\frac{2}{m}(y_1 - \hat{y}_m) \end{bmatrix} = \frac{2}{m}(\mathbf{y} - \hat{\mathbf{y}})$$

Backpropagation (2)

- Suppose we have a very simple neural network with **one hidden layer** followed by a **sigmoid activation**. With **mean squared error**.

$$\hat{\mathbf{y}} = \sigma(\hat{\mathbf{X}}), \quad \hat{\mathbf{x}} = \mathbf{X}\mathbf{w} + b \quad L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- Also the Jacobian matrix of sigmoid function with respect to $\hat{\mathbf{x}}$

$$\frac{\partial \hat{\mathbf{y}}}{\partial \hat{\mathbf{x}}} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial \hat{x}_1} & \frac{\partial \hat{y}_1}{\partial \hat{x}_2} & \cdots & \frac{\partial \hat{y}_1}{\partial \hat{x}_m} \\ \frac{\partial \hat{y}_2}{\partial \hat{x}_1} & \frac{\partial \hat{y}_2}{\partial \hat{x}_2} & \cdots & \frac{\partial \hat{y}_2}{\partial \hat{x}_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_m}{\partial \hat{x}_1} & \frac{\partial \hat{y}_m}{\partial \hat{x}_2} & \cdots & \frac{\partial \hat{y}_m}{\partial \hat{x}_m} \end{bmatrix} = \begin{bmatrix} \sigma(\hat{x}_1)(1-\sigma(\hat{x}_1)) & 0 & \cdots & 0 \\ 0 & \sigma(\hat{x}_2)(1-\sigma(\hat{x}_2)) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma(\hat{x}_m)(1-\sigma(\hat{x}_m)) \end{bmatrix} = \mathbf{1}\sigma(\hat{\mathbf{x}})^T \otimes (\mathbf{I} - \sigma(\hat{\mathbf{x}})\mathbf{1}^T)$$

Backpropagation (3)

- Suppose we have a very simple neural network with **one hidden layer** followed by a **sigmoid activation**. With **mean squared error**.

$$\hat{\mathbf{y}} = \sigma(\hat{\mathbf{X}}), \quad \hat{\mathbf{x}} = \mathbf{X}\mathbf{w} + b \quad L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- Lastly the Jacobian matrix of $\hat{\mathbf{x}}$ with respect to $\mathbf{w}$

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \hat{x}_1}{\partial w_1} & \frac{\partial \hat{x}_1}{\partial w_2} & \cdots & \frac{\partial \hat{x}_1}{\partial w_n} \\ \frac{\partial \hat{x}_2}{\partial w_1} & \frac{\partial \hat{x}_2}{\partial w_2} & \cdots & \frac{\partial \hat{x}_2}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{x}_m}{\partial w_1} & \frac{\partial \hat{x}_m}{\partial w_2} & \cdots & \frac{\partial \hat{x}_m}{\partial w_n} \end{bmatrix} = \mathbf{X}$$

Backpropagation (4)

- Suppose we have a very simple neural network with **one hidden layer** followed by a **sigmoid activation**. With **mean squared error**.

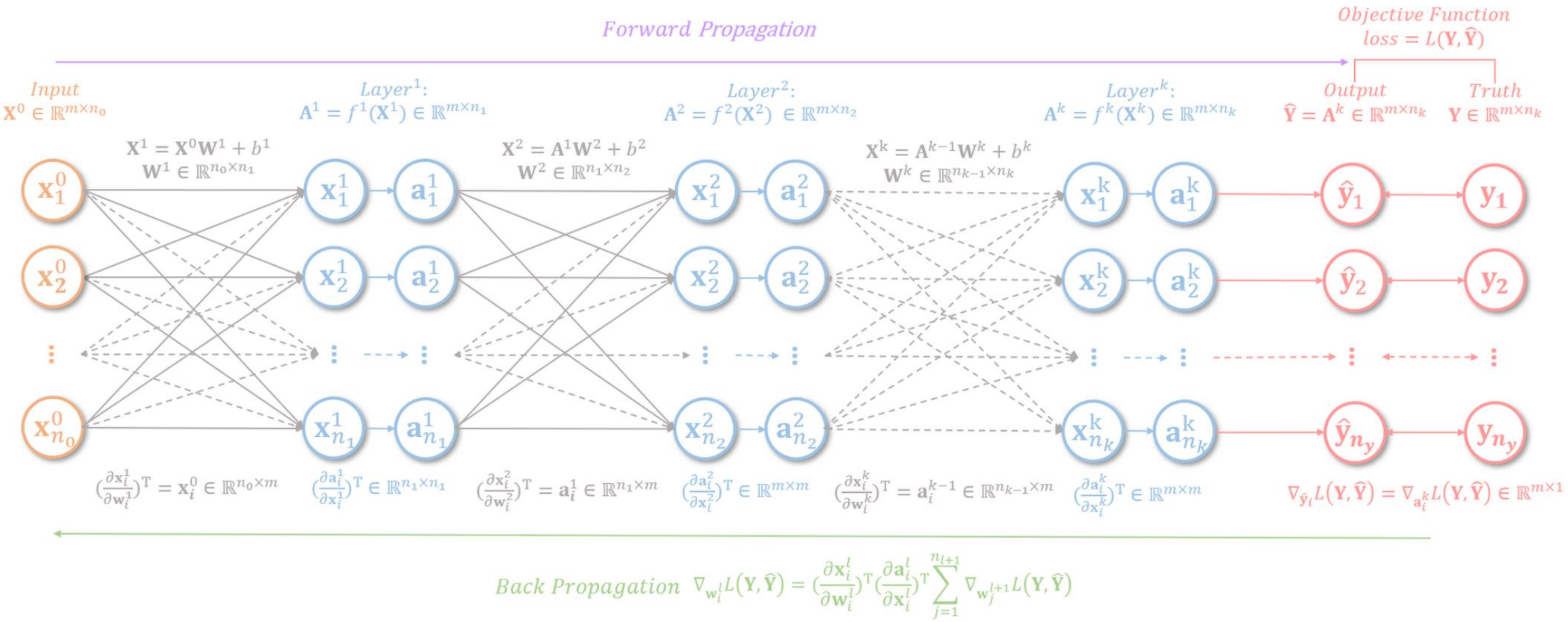
$$\hat{\mathbf{y}} = \sigma(\hat{\mathbf{X}}), \quad \hat{\mathbf{x}} = \mathbf{X}\mathbf{w} + b \quad L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

- Finally, the gradient of the loss function with respect to the weight can be computed using chain rules:

$$\nabla_{\mathbf{w}} L(\mathbf{y}, \hat{\mathbf{y}}) = \begin{bmatrix} \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial w_1} \\ \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial w_2} \\ \vdots \\ \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial w_1} + \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_2}\frac{\partial \hat{x}_2}{\partial w_1} + \cdots + \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_m}\frac{\partial \hat{x}_m}{\partial w_1} \\ \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial w_2} + \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_2}\frac{\partial \hat{x}_2}{\partial w_2} + \cdots + \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_m}\frac{\partial \hat{x}_m}{\partial w_2} \\ \vdots \\ \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial w_n} + \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_2}\frac{\partial \hat{x}_2}{\partial w_n} + \cdots + \frac{\partial L(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{x}_m}\frac{\partial \hat{x}_m}{\partial w_n} \end{bmatrix}$$

$$= \mathbf{X}^T[\mathbf{1}\sigma(\hat{\mathbf{x}})^T \otimes (\mathbf{I} - \sigma(\hat{\mathbf{x}})\mathbf{1}^T)]^T \frac{2}{m}(\mathbf{y} - \hat{\mathbf{y}})$$

Deep Neural Network (2)



In-Silico Mutagenesis (ISM)

- Making *in-silico* perturbations to individual bases in the input and observing the change in the output.

- Steps:
  1. At a given position, mutate into the 3 other possible bases.
  2. Defined the importance of the position as the **negative of the average delta**

- Drawbacks:
  - It only reflects the impact of making a single perturbation.
  - The output has to be recomputed every time there is a perturbation.

Gradient-times-input

(Shrikumar *et al.* 2016)

- Consider a non-linear function:

$$f(x) = y = 2x^2$$

- The gradient is how much the model output will change when the input has changed.

$$\nabla_x f(x) = 4x$$

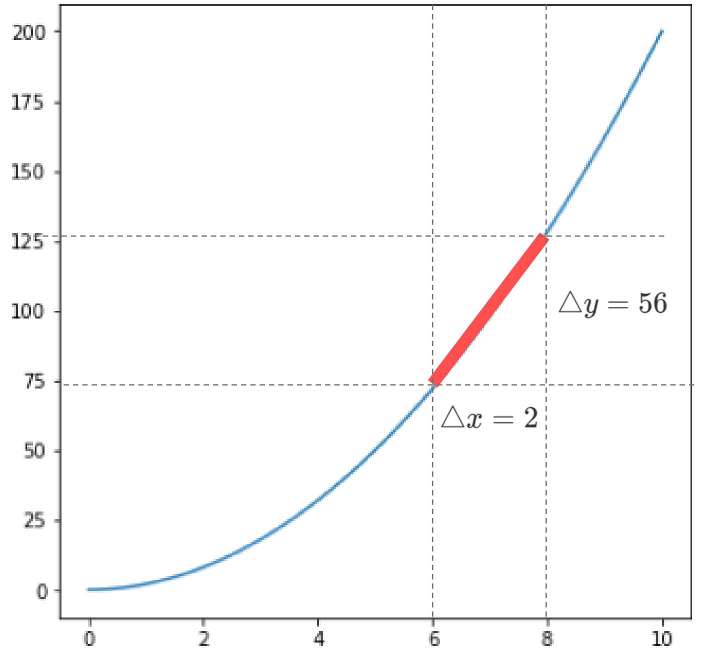
- Generalize to higher dimension

$$f'(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x_i}$$

- Consider the input data, if there is no signal, the contribution should be low.

$$A_i = x_i \frac{\partial f(\mathbf{x})}{\partial x_i}$$ **Gradient-times-input**

- If the gradient is zero, the contribution will be zero. This can lead to saturation problem in neural networks.



$$A_{x|x=6} = x\frac{\nabla f(x)}{\nabla x}|_{x=6} = 6 \times 24 = 144$$

GradCAM (1)

- Compute the gradient of the output with respect to the output of the last CNN layer. For each channel, compute the global max-pooling of the gradient map.

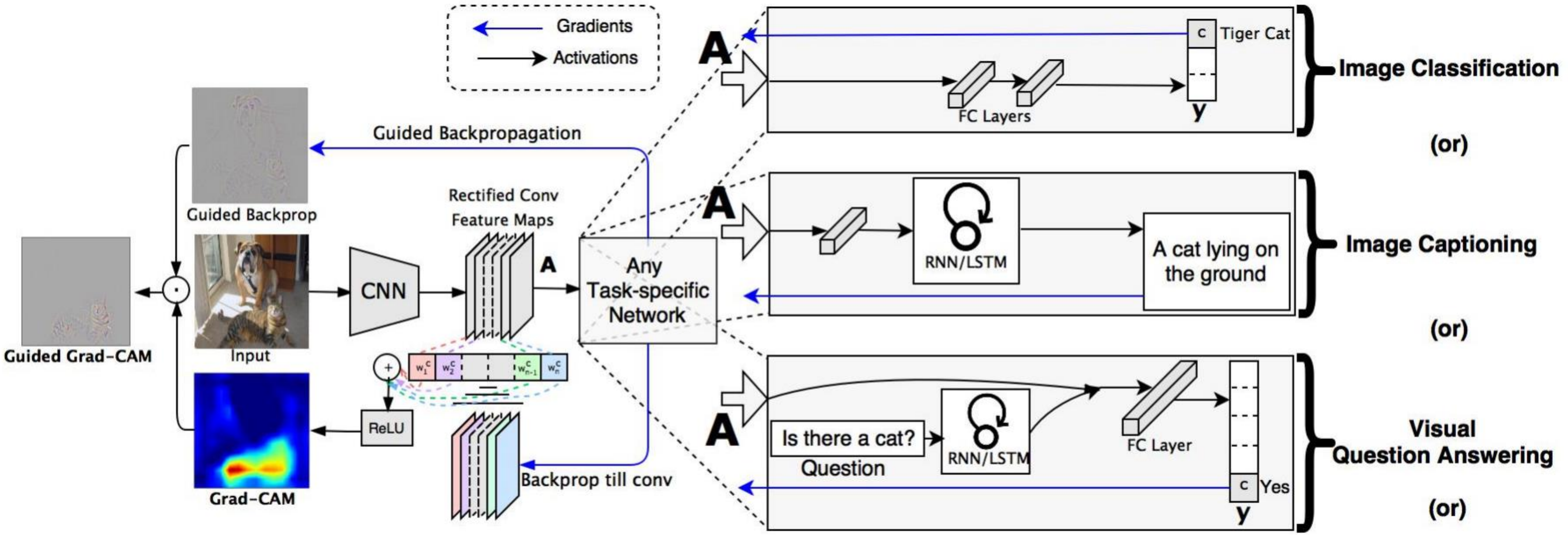
$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i \sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$

- Compute the weighted combination of forward activation map

$$L^c_{\text{Grad-CAM}} = ReLU \underbrace{\left(\sum_k \alpha_k^c A^k \right)}_{\text{linear combination}}$$

- This is very similar to Gradient-times-input, but use the **global max-pooling** of gradient instead and add the **ReLU activation** to remove negative values.

GradCAM (2)



Gradients
Activations
Guided Backpropagation
Guided Backprop
Rectified Conv
Feature Maps
Guided Grad-CAM
Input
CNN
A
Any
Task-specific
Network
ReLU
Grad-CAM
Backprop till conv
A
Tiger Cat
FC Layers
y
Image Classification
(or)
A
RNN/LSTM
A cat lying on the ground
Image Captioning
(or)
A
Is there a cat?
Question
RNN/LSTM
FC Layer
Yes
y
Visual Question Answering
(or)

Integrated Gradient

(Sundararajan *et al.* 2017)

- Define a **reference baseline** (random sequence for DNA, 0 for epigenomic data), and compute the difference between the input data and the reference to make the **interpolated data**, where α controls whether the interpolated data is more similar to the baseline or the original data:

$$\mathbf{x}^{int} = \mathbf{x}^{baseline} + \alpha(\mathbf{x}^{input} - \mathbf{x}^{baseline})$$

- Compute the prediction based on the **interpolated samples** :

$$\frac{\partial f(\mathbf{x}^{int})}{\partial x_i} = \frac{\partial f(\mathbf{x}^{baseline} + \alpha(\mathbf{x}^{input} - \mathbf{x}^{baseline})}{\partial x_i}$$

- Integrate the gradients:

$$\int_{\alpha=0}^{1} \frac{\partial f(\mathbf{x}^{baseline} + \alpha(\mathbf{x}^{input} - \mathbf{x}^{baseline})}{\partial x_i} d\alpha$$ (In practice, discretize α)

- If there is little difference in between the interpolated samples and the reference, the contribution should be low:

$$A_i = (x_i^{input} - x_i^{baseline}) \int_{\alpha=0}^{1} \frac{\partial f(\mathbf{x}^{baseline} + \alpha(\mathbf{x}^{input} - \mathbf{x}^{baseline})}{\partial x_i} d\alpha$$

Image

$\alpha = 0$ $\alpha = 1$

Dinucleotide shuffling frequency

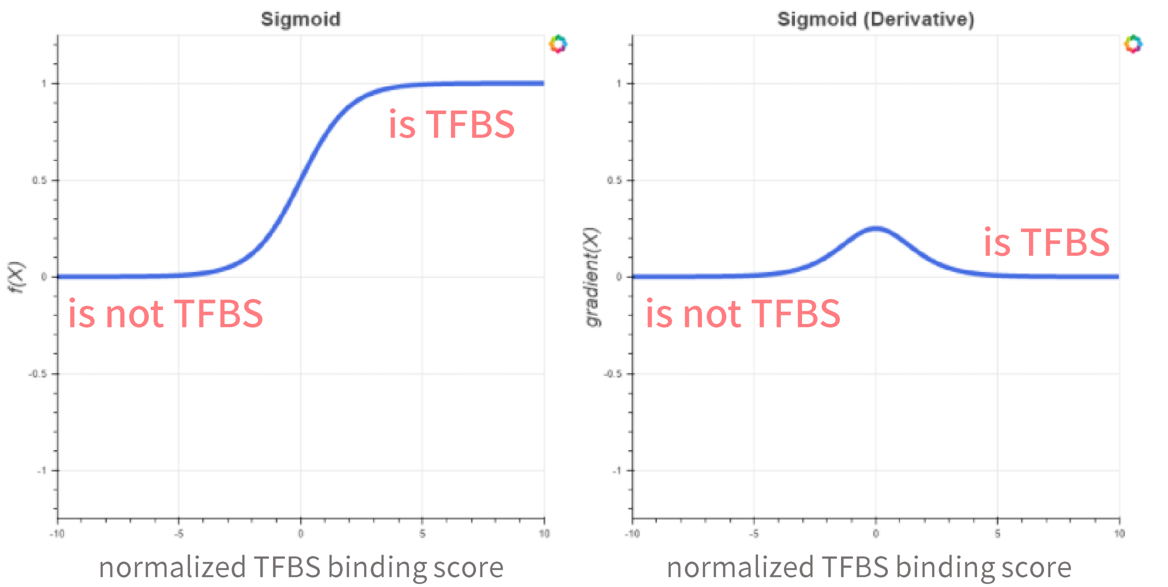
DNA

$P(suffling) = 1.0$ $P(suffling) = 0.0$

$\alpha = 0$ $\alpha = 1$

Why Integrated Gradient Works

- In practice, we could only observe one binding score for a DNA region.
- If the value is very high, or very low, the gradient would be close to zero.
- How do we know if the normalized TFBS binding score is important to predict whether a DNA region is a TFBS in such case.
- We interpolate the input, to create an artificial path from the reference baseline to the observed input (getting many binding scores)
- Finally, the contribution for all the interpolated samples are integrated to infer the attribution score.



The gradient of the original input is zero if the model is very certain

DeepLIFT (1)

- Define a **reference baseline** (random sequence for DNA, 0 for epigenomic data), and compute the difference between the output from the sample and from the reference:

$$t = f(\mathbf{x}),\ t^{baseline} = f(\mathbf{x}^{baseline}) \qquad \Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^{baseline} \quad \Delta t = t - t^{baseline}$$

- Assume the sum of attribution sums to the difference:

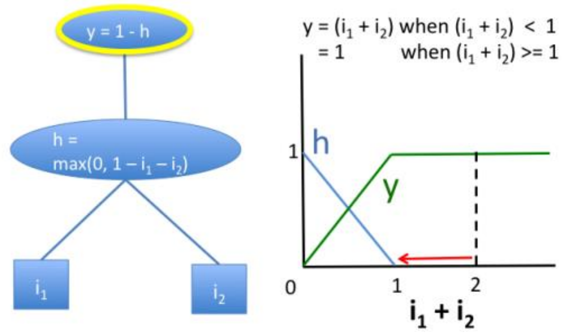
$$\sum_{i=1}^{n} A_{\Delta x_i \Delta t} = \Delta t$$

- Decomposition of positive and negative contribution

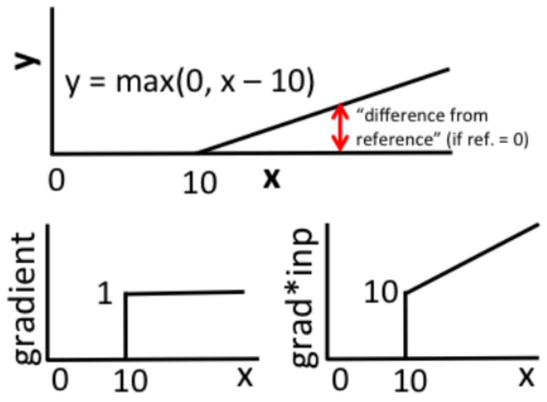
$$A_{\Delta x \Delta t} = A_{\Delta x^+ \Delta t} + A_{\Delta x^- \Delta t}$$

- Define multipliers (similar to gradient):

$$m_{\Delta x \Delta t} = \frac{\sum_{i=1}^{n} A_{\Delta x_i \Delta t}}{\Delta \mathbf{x}} \qquad m_{\Delta x_i \Delta t} = \sum_j (m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta t})$$ (chain rule)



Saturated Gradient



Discontinuous Gradient

DeepLIFT (2)

Linear Rule

- Assign attribution for linear or convolutional layer (w/o activations)

$$y = \sum_{i=1}^{n} x_i w_i + b \Rightarrow \Delta y = \sum_{i=1}^{n} \Delta x_i w_i + b$$

$$\Delta y^+ = \sum_{i=1}^{n} 1_{\{\Delta x_i w_i > 0\}} \Delta x_i w_i + b = \sum_{i=1}^{n} 1_{\{\Delta x_i w_i > 0\}} (\Delta x_i^+ + \Delta x_i^-) w_i + b$$ (linear combination of Δx where Δxw is **positive**)

$$\Delta y^- = \sum_{i=1}^{n} 1_{\{\Delta x_i w_i < 0\}} \Delta x_i w_i + b = \sum_{i=1}^{n} 1_{\{\Delta x_i w_i < 0\}} (\Delta x_i^+ + \Delta x_i^-) w_i + b$$ (linear combination of Δx where Δxw is **negative**)

- Recall that $\sum_{i=1}^{n} A_{\Delta x_i \Delta y} = \Delta y$ and $m_{\Delta x \Delta t} = \frac{\sum_{i=1}^{n} A_{\Delta x_i \Delta t}}{\Delta \mathbf{x}}$

$$\Rightarrow \begin{aligned} A_{\Delta x_i^+ \Delta y_i^+} &= 1_{\{\Delta x_i w_i > 0\}} w_i \Delta x_i^+ \\ A_{\Delta x_i^- \Delta y_i^+} &= 1_{\{\Delta x_i w_i > 0\}} w_i \Delta x_i^- \\ A_{\Delta x_i^+ \Delta y_i^-} &= 1_{\{\Delta x_i w_i < 0\}} w_i \Delta x_i^+ \\ A_{\Delta x_i^- \Delta y_i^-} &= 1_{\{\Delta x_i w_i < 0\}} w_i \Delta x_i^- \end{aligned} \Rightarrow \begin{aligned} m_{\Delta x_i^+ \Delta y_i^+} &= m_{\Delta x_i^- \Delta y_i^+} = 1_{\{\Delta x_i w_i > 0\}} w_i \\ m_{\Delta x_i^+ \Delta y_i^-} &= m_{\Delta x_i^- \Delta y_i^-} = 1_{\{\Delta x_i w_i < 0\}} w_i \end{aligned}$$

yields a valid solution to for the summation-to-delta rules assigning attribution.

DeepLIFT (3)

Rescale and RevealCancel Rule

- Assign attribution for ReLU, sigmoid, hyperbolic tangent (**element-wise**) activations.

  $y = f(x)$

- Since it is elementwise operation, it will have univariate input:

$$A_{\Delta x \Delta y} = \Delta y \Rightarrow m_{\Delta x \Delta y} = \frac{\Delta y}{\Delta x}$$

$$\left.\begin{aligned} \Delta y^{+} &= \frac{\Delta y}{\Delta x}\Delta x^{+} = A_{\Delta x^{+}\Delta y^{+}} \\ \Delta y^{-} &= \frac{\Delta y}{\Delta x}\Delta x^{-} = A_{\Delta x^{-}\Delta y^{-}} \end{aligned}\right. \Rightarrow m_{\Delta x_i^{+}\Delta y_i^{+}} = m_{\Delta x_i^{-}\Delta y_i^{-}} = m_{\Delta x_i \Delta y_i} = \frac{\Delta y}{\Delta x}$$

Rescale Rule

- The rule can be further improved if not assuming $\Delta y^{+} \propto \Delta x^{+}, \quad \Delta y^{-} \propto \Delta x^{-}$

$$\left.\begin{aligned} \Delta y^{+} &= \frac{1}{2}\left(f(x^0 + \Delta x^{+}) - f(x^0)\right) \\ &\quad + \frac{1}{2}\left(f(x^0 + \Delta x^{-} + \Delta x^{+}) - f(x^0 + \Delta x^{-})\right) \\ \Delta y^{-} &= \frac{1}{2}\left(f(x^0 + \Delta x^{-}) - f(x^0)\right) \\ &\quad + \frac{1}{2}\left(f(x^0 + \Delta x^{+} + \Delta x^{-}) - f(x^0 + \Delta x^{+})\right) \end{aligned}\right. \Rightarrow m_{\Delta x_i^{+}\Delta y_i^{+}} = \frac{\Delta y^{+}}{\Delta x^{+}}, \quad m_{\Delta x_i^{-}\Delta y_i^{-}} = \frac{\Delta y^{-}}{\Delta x^{-}}$$

RevealCancel Rule

Discussion

- The benefit of CAM is primarily due to the layer at which the importance scores are calculated, as opposed to the position-invariant channel-weighting approach.
  - The importance scores can become worse when computed closer to the input layer.
- ISM performs slightly worse than DeepLIFT.
  - Mutating a single base may not be enough to substantially disrupt some motif instances.
  - Some motifs identified in the positive set might not be bound due to missing contextual features (might be because of the motif discovery method used during simulation).
- The authors claim that our simulated dataset can be applied not just for benchmarking interpretation, but also for understanding and debugging the learning dynamics of machine learning
  - The authors could give more examples about these applications in the manuscript as a use case.