AI for Genomics: from CNNs and LSTMs to Transformers

Nikolay Oskolkov, Group Leader (PI) at LIOS, Riga, Latvia

Physalia course, 09.09.2025

Session 2c: Bayesian neural networks principles and applications to clinical diagnostics

@NikolayOskolkov

@oskolkov.bsky.social

Personal homepage:
https://nikolay-oskolkov.com

Image generated at www.canva.com

Session content

Topics we'll cover in this session:

1) When traditional Deep Learning is not good enough

2) Differences between Frequentist and Bayesian fitting

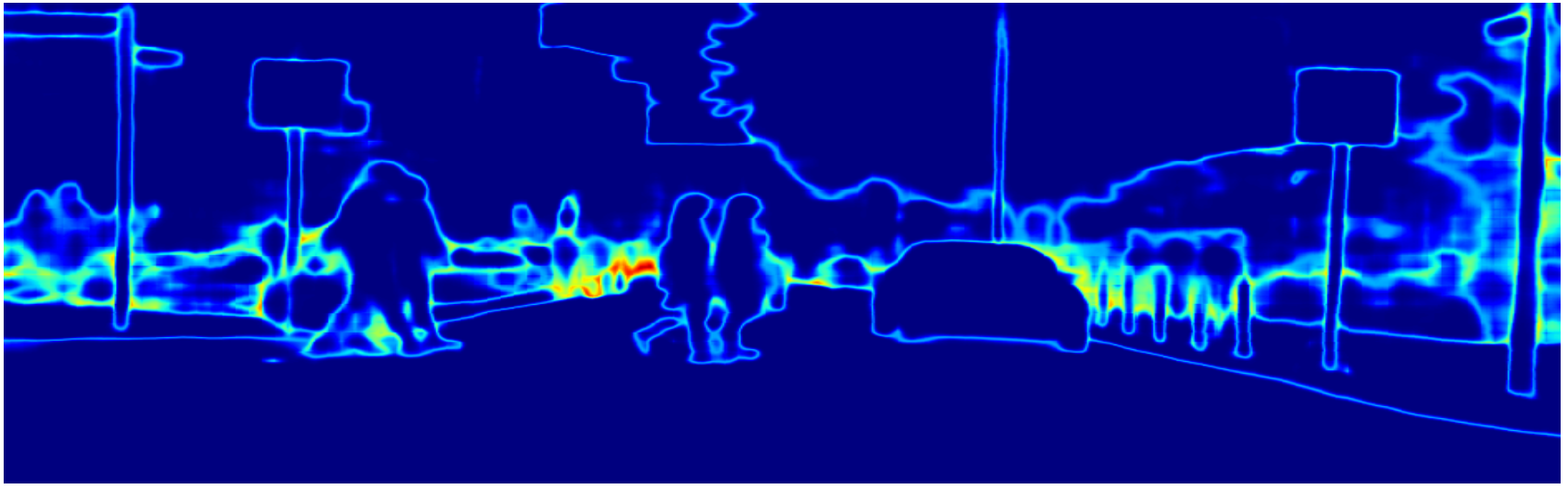
3) Markov Chain Monte Carlo (MCMC) from scratch in R

4) Bayesian Artificial Neural Networks

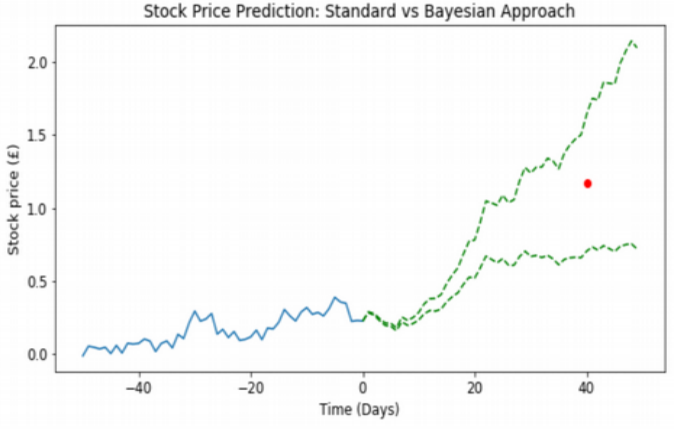
5) Applications of Bayesian Deep Learning in Life Sciences

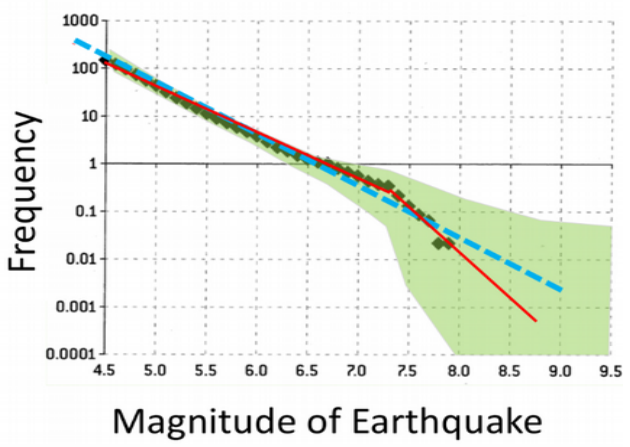
Fundamentals of Bayesian learning

When Deep Learning is not good enough

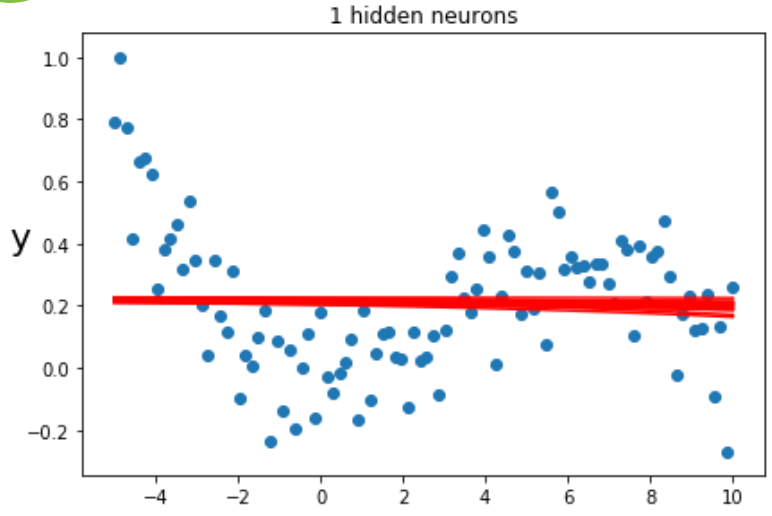


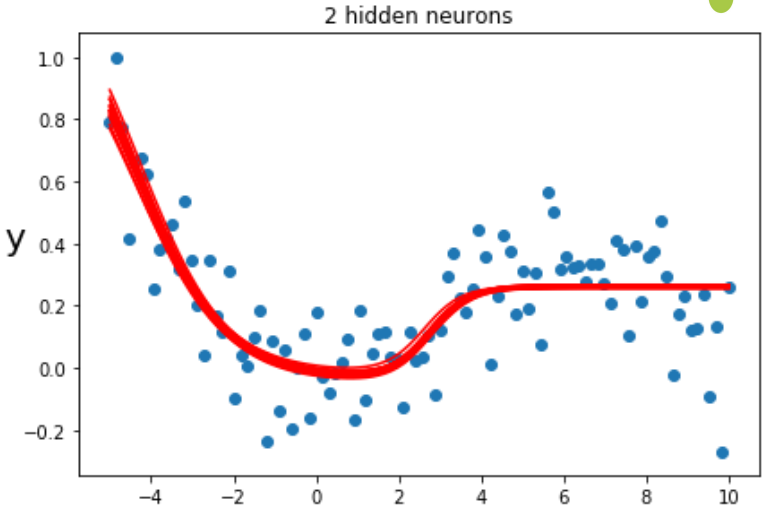
Intelligence is to know how much you do not know

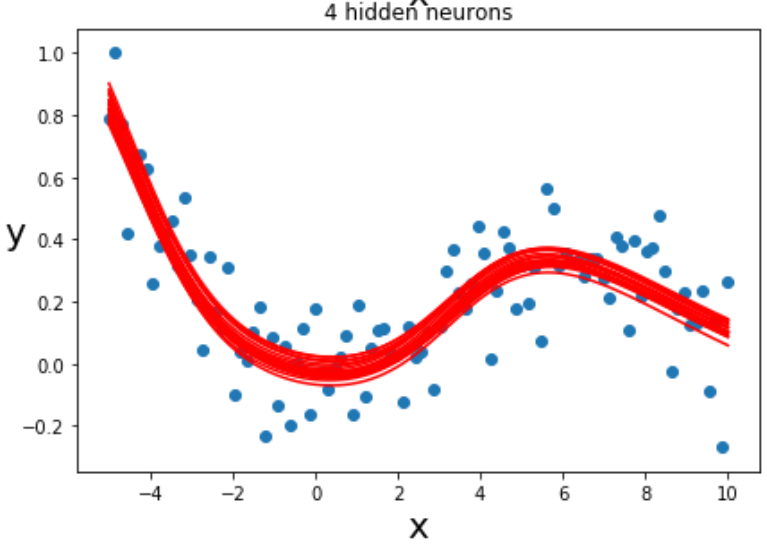


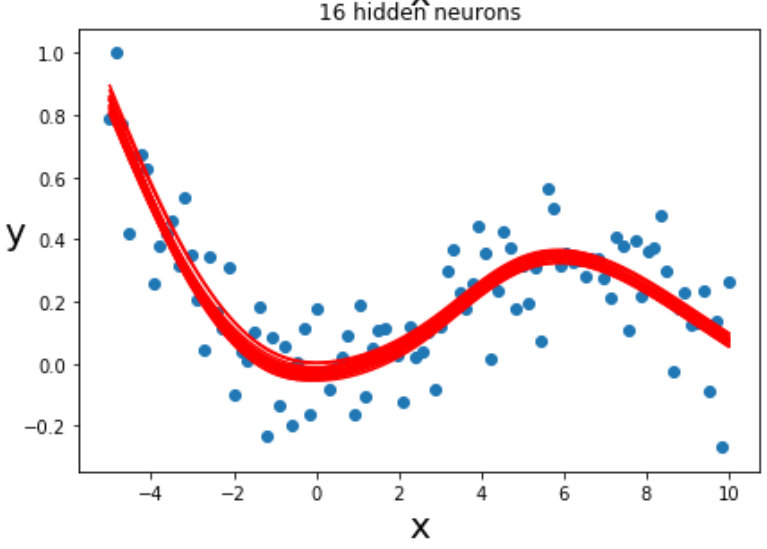


Bayesian fitting with uncertainties

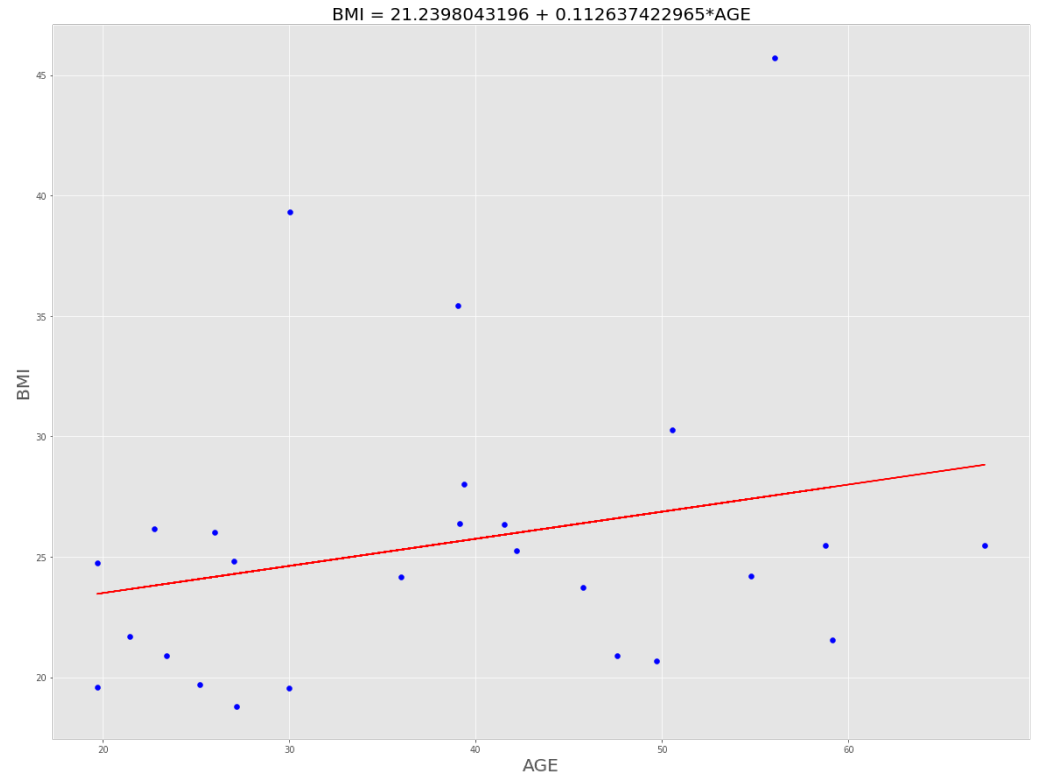




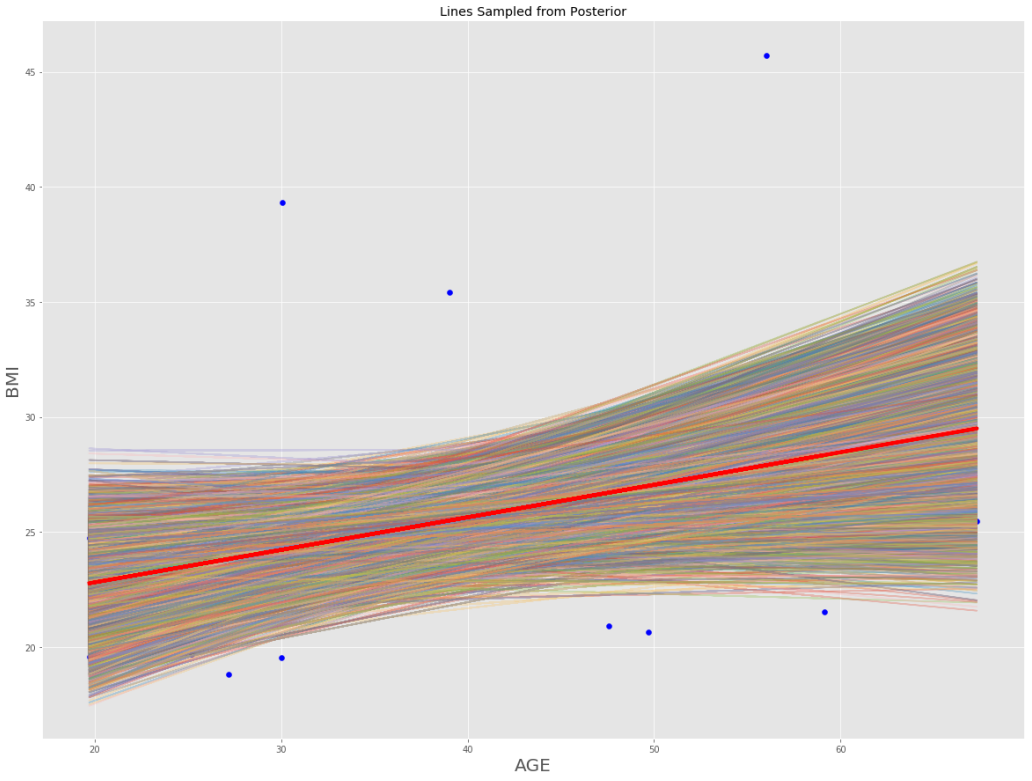




Frequentist vs. Bayesian linear fitting

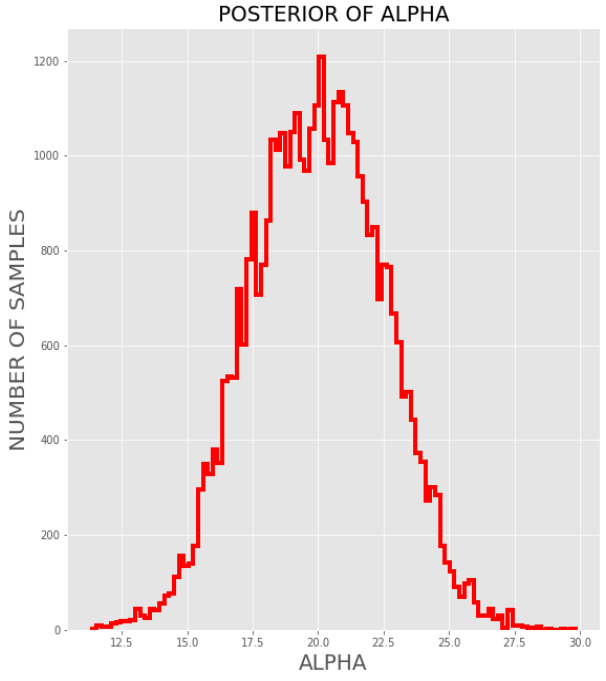


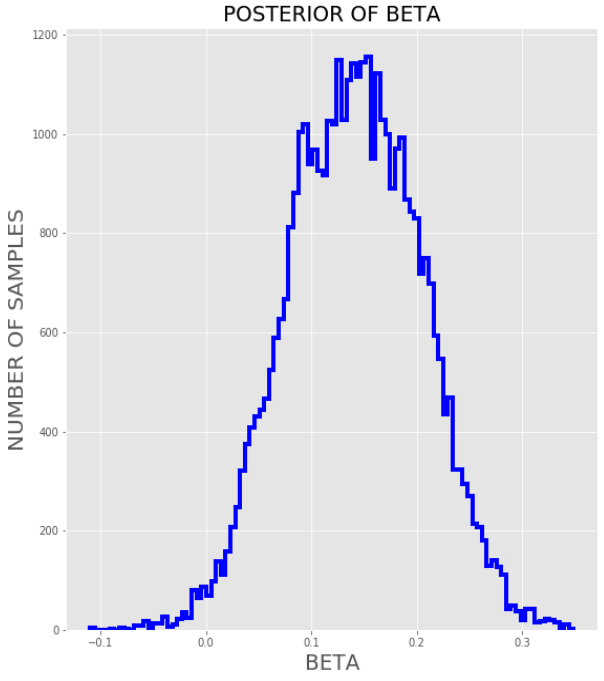
Fitting Frequentist linear model

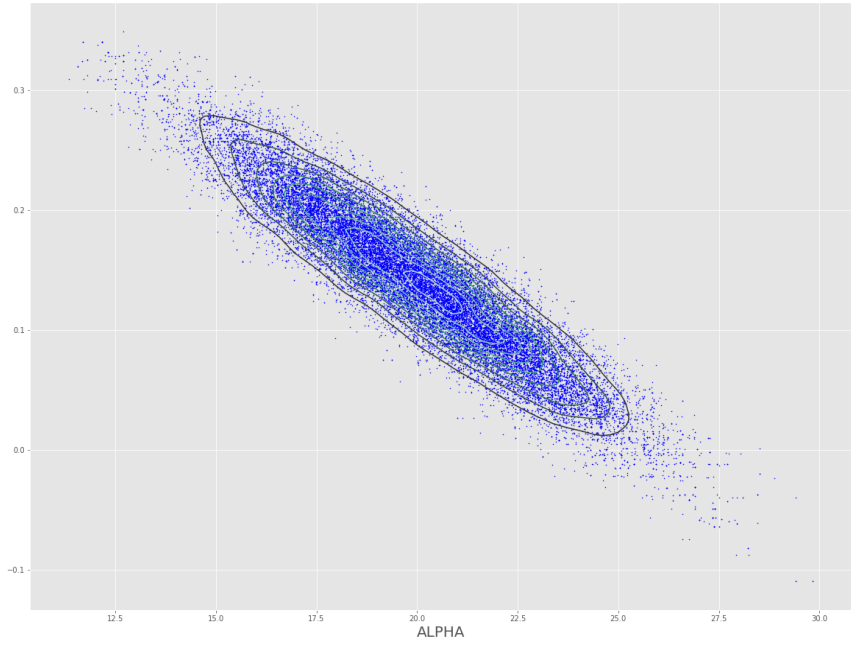


Fitting Bayesian linear model

Drawing possible fits from Posterior







Slope and intersect are not fixed values any more

MCMC can draw infinite number of fits from proximity of optimal slope and intercept parameters

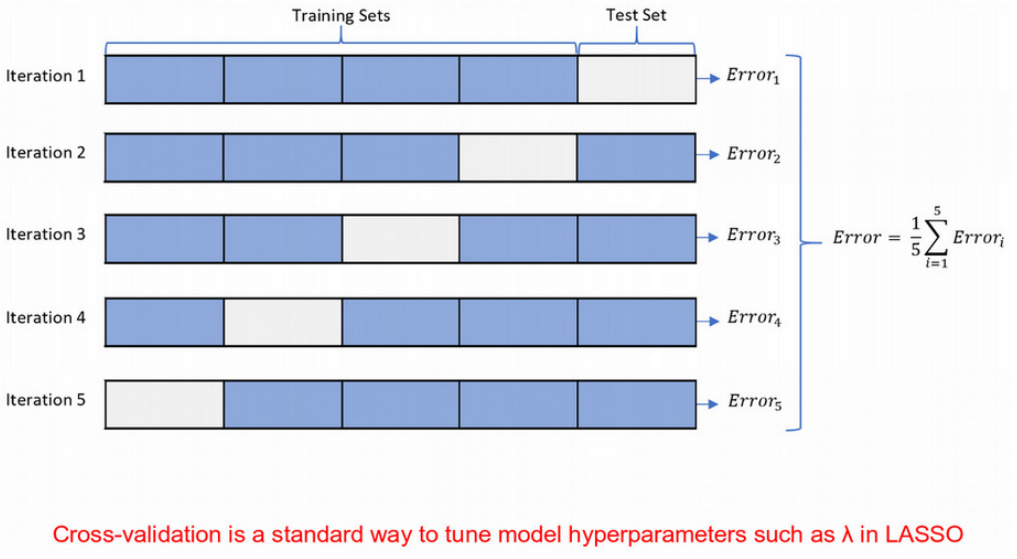
Relation between Frequentist and Bayesian

Regularizations: LASSO

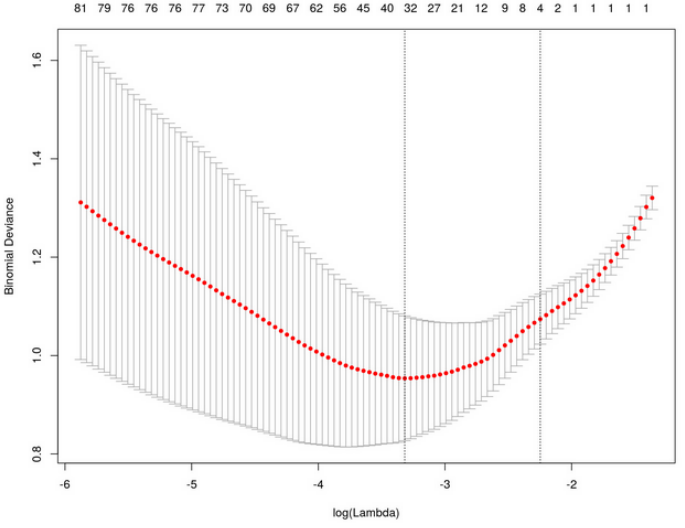
$$Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

$$\text{OLS} = (Y - \beta_1 X_1 - \beta_2 X_2)^2$$

$$\text{Penalized OLS} = (Y - \beta_1 X_1 - \beta_2 X_2)^2 + \lambda(|\beta_1| + |\beta_2|)$$



Cross-validation is a standard way to tune model hyperparameters such as λ in LASSO



Regularizations are priors in Bayesian statistics

$$Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon; \quad Y \sim N(\, \beta_1 X_1 + \beta_2 X_2, \sigma^2\,) \equiv \mathrm{L}\,(\,\mathrm{Y} \mid \beta_1, \beta_2\,)$$

- **Maximum Likelihood** principle: maximize probability to observe data given parameters:

$$\mathrm{L}\,(\,\mathrm{Y} \mid \beta_1, \beta_2\,) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(\mathrm{Y} - \beta_1 \mathrm{X}_1 - \beta_2 \mathrm{X}_2)^2}{2\sigma^2}}$$

- **Bayes theorem**: maximize posterior probability of observing parameters given data:

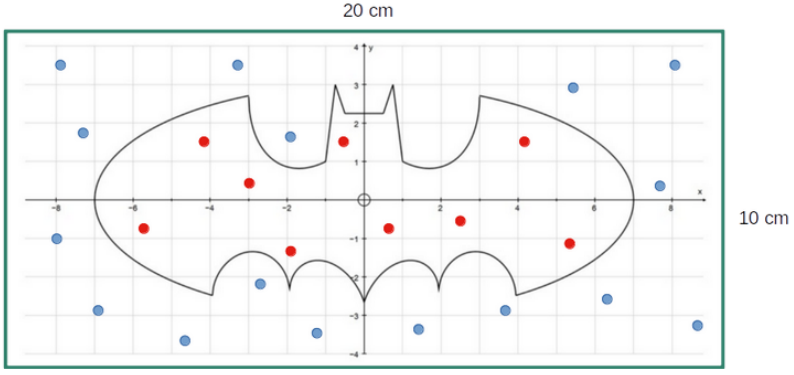
$$\mathrm{Posterior}(\mathrm{params} \mid \mathrm{data}) = \frac{\mathrm{L}(\mathrm{data} \mid \mathrm{params}) * \mathrm{Prior}(\mathrm{params})}{\int \mathrm{L}(\mathrm{data} \mid \mathrm{params}) * \mathrm{Prior}(\mathrm{params})\, \mathrm{d}(\mathrm{params})}$$

$$\mathrm{Posterior}(\,\beta_1, \beta_2 \mid \mathrm{Y}\,) \sim \mathrm{L}\,(\,\mathrm{Y} \mid \beta_1, \beta_2\,) * \mathrm{Prior}(\beta_1, \beta_2) \sim \exp^{-\frac{(\mathrm{Y} - \beta_1 \mathrm{X}_1 - \beta_2 \mathrm{X}_2)^2}{2\sigma^2}} * \exp^{-\lambda(|\beta_1| + |\beta_2|)}$$

$$-\log\left[\mathrm{Posterior}(\,\beta_1, \beta_2 \mid \mathrm{Y}\,)\right] \sim (\mathrm{Y} - \beta_1 \mathrm{X}_1 - \beta_2 \mathrm{X}_2)^2 + \lambda(|\beta_1| + |\beta_2|)$$

Markov Chain Monte Carlo (MCMC): introduction

- Integration via Monte Carlo sampling

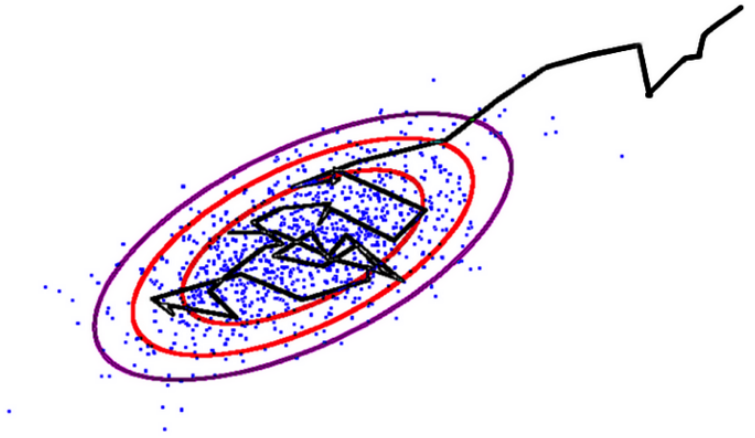


$$I = 2\int_{2}^{4} x dx = 2\frac{x^2}{2}\Big|_{2}^{4} = 16 - 4 = 12$$

```
f <- function(x){return(2*x)}; a <- 2; b <- 4; N <- 10000; count <- 0
x <- seq(from = a, to = b, by = (b-a) / N); y_max <- max(f(x))
for(i in 1:N)
{
  x_sample <- runif(1, a, b); y_sample <- runif(1, 0, y_max)
  if(y_sample <= f(x_sample)){count <- count + 1}
}
paste0("Integral by Monte Carlo: I = ", (count / N) * (b - a) * y_max)
```

```
[1] "Integral by Monte Carlo: I = 11.9248"
```

- Markov Chain Monte Carlo (MCMC)

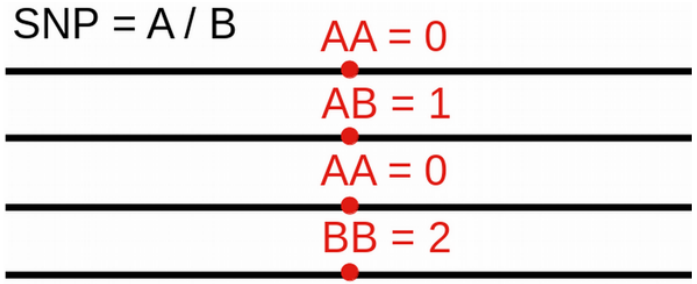


$$\text{Hastings ratio} = \frac{\text{Posterior}\left(\text{params}_{\text{next}} \mid \text{data}\right)}{\text{Posterior}\left(\text{params}_{\text{previous}} \mid \text{data}\right)}$$

- If Hastings ratio > *u* [0, 1], then **accept**, else **reject**
- Hastings ratio does not contain the intractable integral from Bayes theorem

Markov Chain Monte Carlo (MCMC) from scratch in R

- Example from population genetics

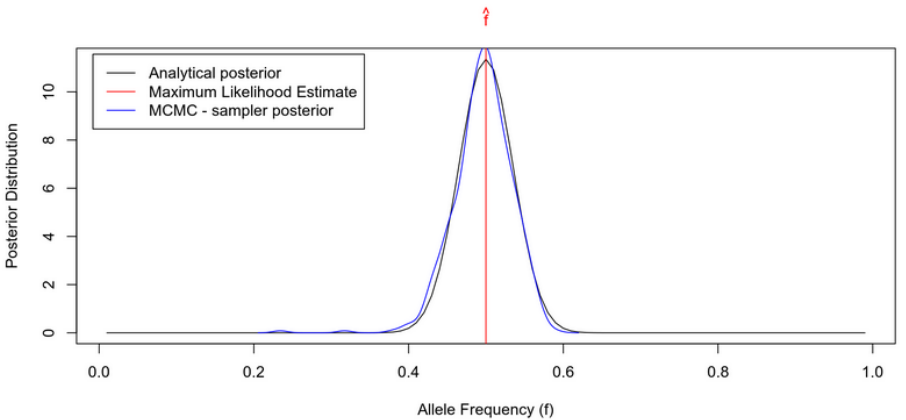



```r
N <- 100; n <- c(25, 50, 25) # Observed genotype data for N individuals
f_MLE <- (n[2] + 2*n[3]) / (2 * sum(n)) # MLE of allele frequency

# Define log-likelihood function (log-binomial distribution)
LL <- function(n, f){return((n[2] + 2*n[3])*log(f) + (n[2] + 2*n[1])*log(1-f))}
# Define log-prior function (log-beta distribution)
LP <- function(f, alpha, beta){return(dbeta(f, alpha, beta, log = TRUE))}

# Run MCMC Metropolis - Hastings sampler
f_poster <- vector(); alpha <- 0.5; beta <- 0.5; f_cur <- 0.1 # initialization
for(i in 1:1000)
{
  f_next <- abs(rnorm(1, f_cur, 0.1)) # make random step for allele frequency

  LL_cur <- LL(n, f_cur); LL_next <- LL(n, f_next)
  LP_cur <- LP(f_cur, alpha, beta); LP_next <- LP(f_next, alpha, beta)
  hastings_ratio <- LL_next + LP_next - LL_cur - LP_cur

  if(hastings_ratio > log(runif(1))){f_cur <- f_next}; f_poster[i] <- f_cur
}
```

$$L(n \mid f) = \prod_g \left[\binom{2}{g} f^g (1-f)^{2-g} \right]^{n_g}$$

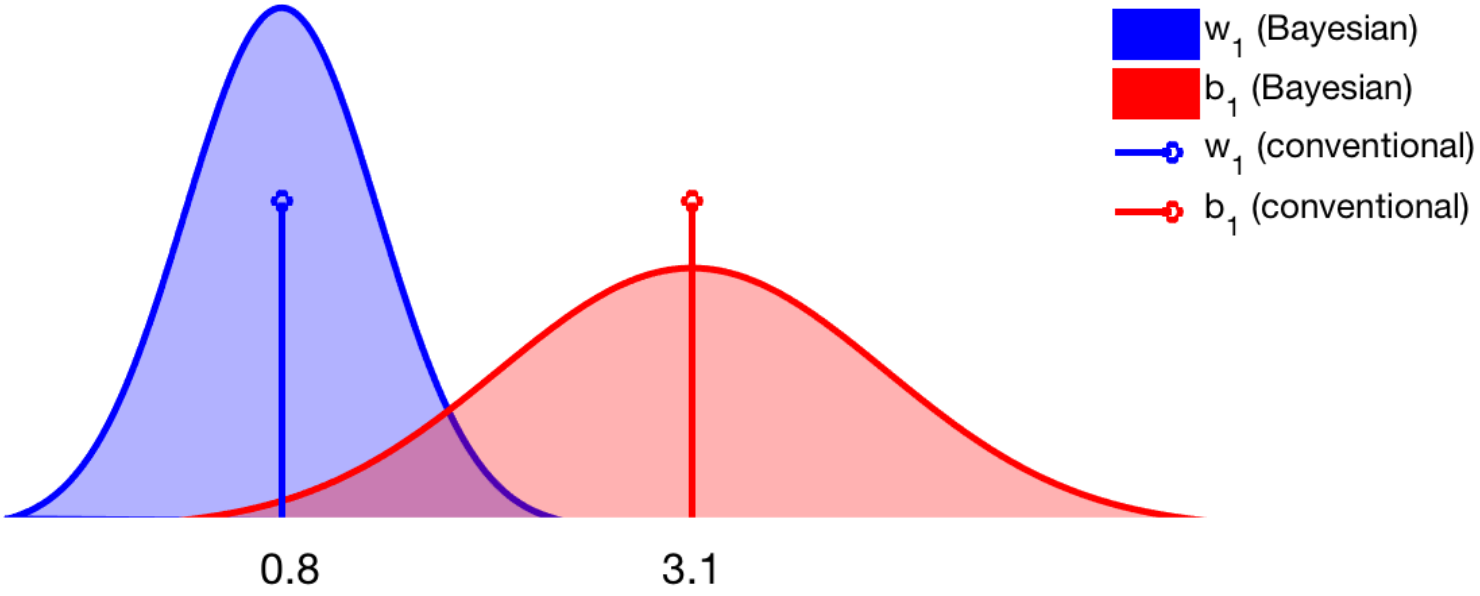
$$\frac{\partial \log[L(n|f)]}{\partial f} = 0 \Rightarrow \hat{f} = \frac{n_1 + 2n_2}{2(n_0 + n_1 + n_2)}$$

$$\text{Prior}(\text{f}, \alpha, \beta) = \frac{1}{\text{B}(\alpha, \beta)} \text{f}^{\alpha-1} (1-\text{f})^{\beta-1}$$



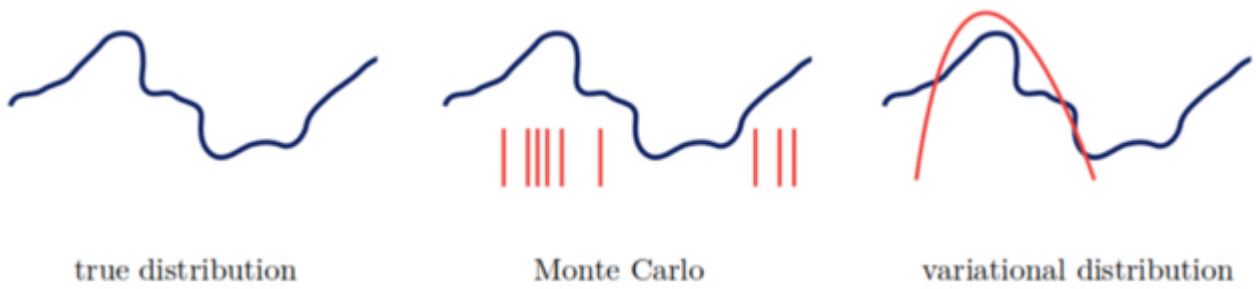
Bayesian deep learning

Bayesian artificial neural network



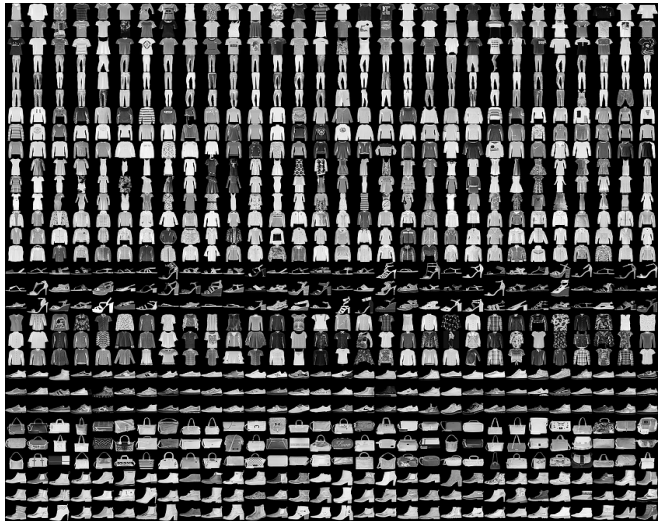
Adding priors
to weights of
neural networks

Possible ways
for computing
posterior distribution:



Frequentist image recognition

Fashion MNIST



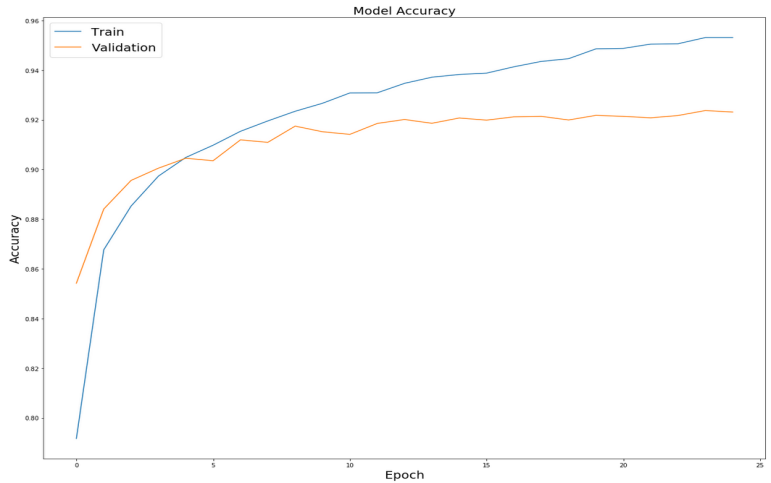
```
In [24]: # normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0

In [25]: # one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
print(num_classes)
10

In [27]: # Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(1, 28, 28), padding='same', activation='relu',
                 kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
                 kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
```

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 32, 28, 28)	320
dropout_7 (Dropout)	(None, 32, 28, 28)	0
conv2d_9 (Conv2D)	(None, 32, 28, 28)	9248
max_pooling2d_4 (MaxPooling2	(None, 32, 14, 14)	0
flatten_4 (Flatten)	(None, 6272)	0
dense_7 (Dense)	(None, 512)	3211776
dropout_8 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 10)	5130

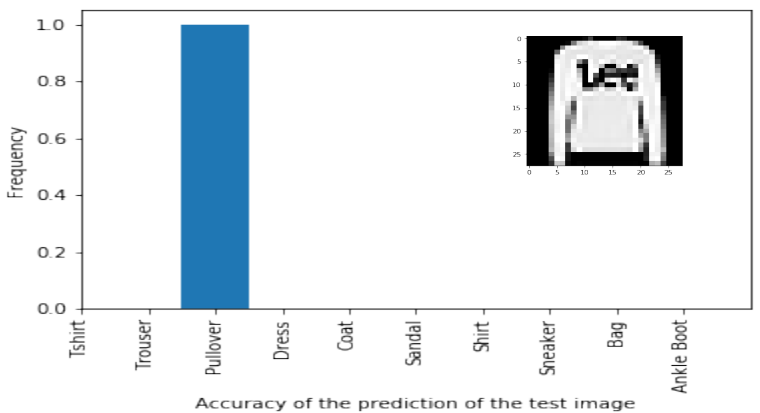
Total params: 3,226,474
Trainable params: 3,226,474
Non-trainable params: 0

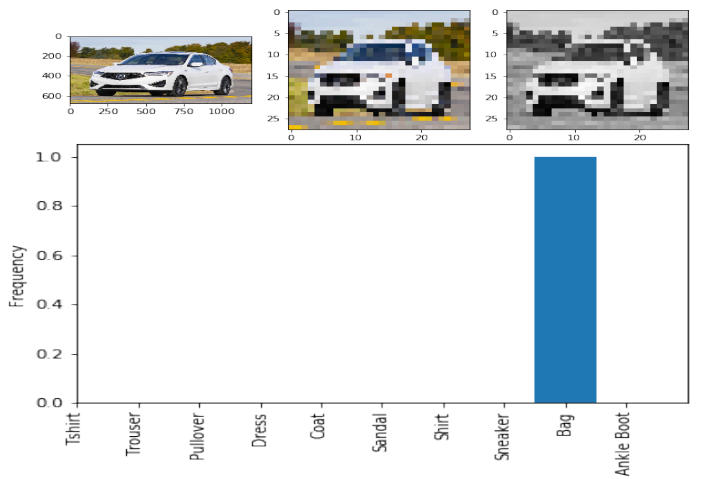
None

```
In [28]: # Fit the model
#model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
history = model.fit(X_train, y_train, epochs = epochs, verbose = 1, validation_split = 0.25,
                    batch_size = 32, shuffle = True)

Train on 45000 samples, validate on 15000 samples
Epoch 1/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.5762 - acc: 0.7917 - val_loss: 0.39
73 - val_acc: 0.8542
Epoch 2/25
45000/45000 [==============================] - 1124s 25ms/step - loss: 0.3643 - acc: 0.8676 - val_loss: 0.31
27 - val_acc: 0.8841
Epoch 3/25
45000/45000 [==============================] - 1158s 26ms/step - loss: 0.3129 - acc: 0.8853 - val_loss: 0.28
25 - val_acc: 0.8956
Epoch 4/25
45000/45000 [==============================] - 1609s 36ms/step - loss: 0.2813 - acc: 0.8973 - val_loss: 0.27
27 - val_acc: 0.9005
Epoch 5/25
45000/45000 [==============================] - 902s 20ms/step - loss: 0.2618 - acc: 0.9048 - val_loss: 0.258
8 - val_acc: 0.9045
Epoch 6/25
45000/45000 [==============================] - 936s 21ms/step - loss: 0.2451 - acc: 0.9098 - val_loss: 0.256
4 - val_acc: 0.9035
Epoch 7/25
```



Prediction



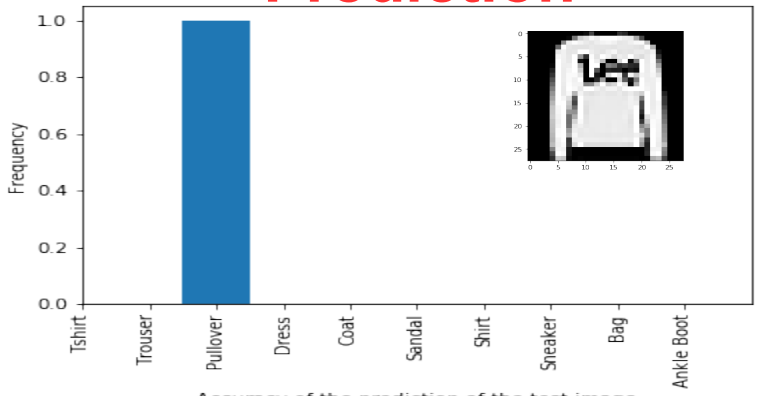


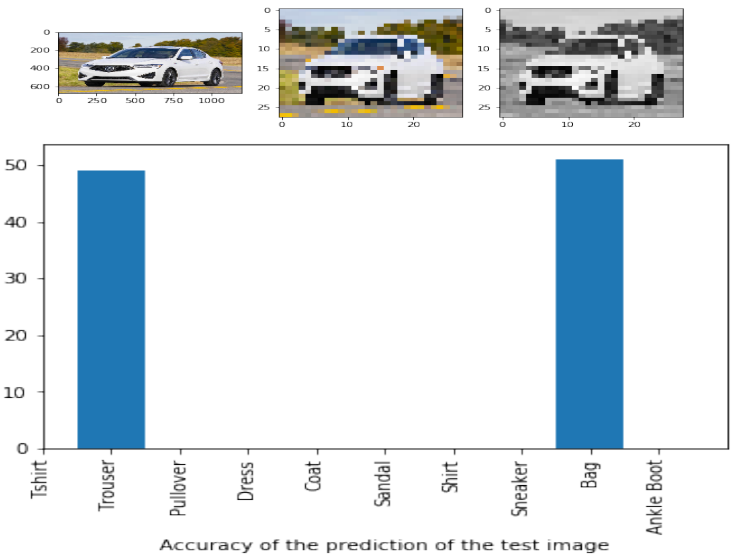
Bayesian image recognition

PyMC3, Edward, TensorFlow Probability

```
In [8]: x_train = x_train.reshape((x_train.shape[0],D))
        x_test = x_test.reshape((x_test.shape[0],D))
        print(x_train.shape)
        print(x_test.shape)

        (60000, 784)
        (10000, 784)

In [9]: from keras.utils import to_categorical
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)
        print(y_train.shape)
        print(y_test.shape)

        (60000, 10)
        (10000, 10)

In [10]: ed.set_seed(314159)
         N = 100      # number of images in a minibatch.
         D = D        # number of features.
         K = 10       # number of classes.

         # Create a placeholder to hold the data (in minibatches) in a TensorFlow graph.
         x = tf.placeholder(tf.float32, [None, D])
         # Normal(0,1) priors for the variables. Note that the syntax assumes TensorFlow 1.1.
         w = Normal(loc=tf.zeros([D, K]), scale=tf.ones([D, K]))
         b = Normal(loc=tf.zeros(K), scale=tf.ones(K))
         # Categorical likelihood for classication.
         y = Categorical(tf.matmul(x, w) + b)

In [11]: # Contruct the q(w) and q(b). in this case we assume Normal distributions.
         qw = Normal(loc=tf.Variable(tf.random_normal([D, K])),
                     scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, K]))))
         qb = Normal(loc=tf.Variable(tf.random_normal([K])),
                     scale=tf.nn.softplus(tf.Variable(tf.random_normal([K]))))

In [12]: def generator(arrays, batch_size = N):
             starts = [0] * len(arrays)  # pointers to where we are in iteration
             while True:
                 batches = []
                 for i, array in enumerate(arrays):
                     start = starts[i]
                     stop = start + batch_size
                     diff = stop - array.shape[0]
                     if diff <= 0:
                         batch = array[start:stop]
                         starts[i] += batch_size
                     else:
                         batch = np.concatenate((array[start:], array[:diff]))
                         starts[i] = diff
                     batches.append(batch)
                 yield batches
         cifar10 = generator([x_train, y_train], N)

In [13]: # We use a placeholder for the labels in anticipation of the traning data.
         y_ph = tf.placeholder(tf.int32, [N])
         # Define the VI inference technique, ie. minimise the KL divergence between q and p.
         inference = ed.KLqp({w: qw, b: qb}, data={y: y_ph})
         # Initialse the infernce variables
         inference.initialize(n_iter=50000, n_print=100, scale={y: float(x_train.shape[0]) / N})
         # We will use an interactive session.
         sess = tf.InteractiveSession()
         # Initialise all the vairables in the session.
         tf.global_variables_initializer().run()
         # Let the training begin. We load the data in minibatches and update the VI infernce using each new batch.
         for _ in range(inference.n_iter):
             X_batch, Y_batch = next(cifar10)
             #X_batch = X_batch.reshape(N,-1)
             # TensorFlow method gives the label data in a one hot vetor format. We convert that into a single label.
             Y_batch = np.argmax(Y_batch, axis=1)
             info_dict = inference.update(feed_dict={x: X_batch, y_ph: Y_batch})
             inference.print_progress(info_dict)
         50000/50000 [100%] ██████████████████████████████ Elapsed: 221s | Loss: 85453.266

In [14]: # Generate samples the posterior and store them.
         n_samples = 100
         prob_lst = []
         samples = []
         w_samples = []
         b_samples = []
         for _ in range(n_samples):
             w_samp = qw.sample()
             b_samp = qb.sample()
             w_samples.append(w_samp)
             b_samples.append(b_samp)
             # Also compute the probabiliy of each class for each (w,b) sample.
             prob = tf.nn.softmax(tf.matmul(x_test, w_samp) + b_samp)
             prob_lst.append(prob.eval())
             sample = tf.concat([tf.reshape(w_samp,[-1]),b_samp],0)
             samples.append(sample.eval())

In [15]: # Compute the accuracy of the model.
         # For each sample we compute the predicted class and compare with the test labels.
         # Predicted class is defined as the one which as maximum probability.
         # We perform this test for each (w,b) in the posterior giving us a set of accuracies
         # Finally we make a histogram of accuracies for the test data.
         accy_test = []
         for prob in prob_lst:
             y_trn_prd = np.argmax(prob, axis=1).astype(np.float32)
             acc = (y_trn_prd == np.argmax(y_test, axis=1)).mean()*100
             accy_test.append(acc)

         plt.hist(accy_test)
         plt.title("Histogram of prediction accuracies in the CIFAR10 test data")
         plt.xlabel("Accuracy")
         plt.ylabel("Frequency")
```

Prediction





Bayesian deep learning applications for single cell biology

scRNAseq Cancer-Associated Fibroblasts

nature communications

Article | Open access | Published: 04 December 2018

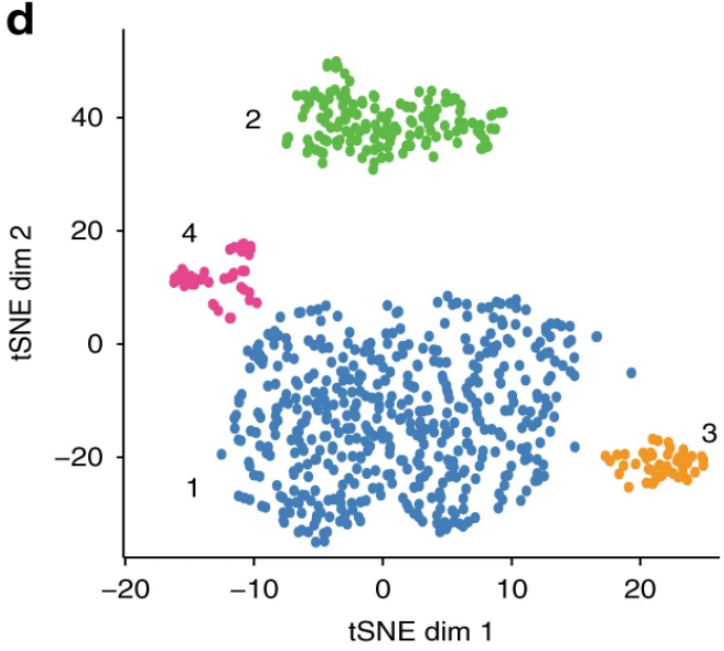
Spatially and functionally distinct subclasses of brea cancer-associated fibroblasts revealed by single cell RNA sequencing

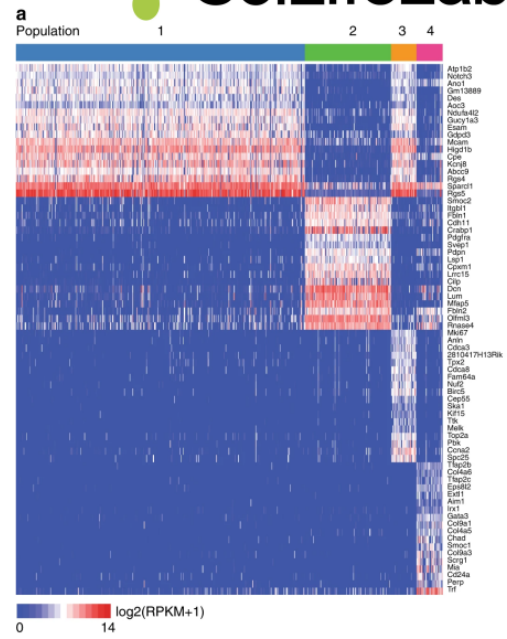
Michael Bartoschek, Nikolay Oskolkov, Matteo Bocci, John Lövrot, Christer Larsson, Mikael Sommar Chris D. Madsen, David Lindgren, Gyula Pekar, Göran Karlsson, Markus Ringnér, Jonas Bergh, Åsa Björklund & Kristian Pietras ✉

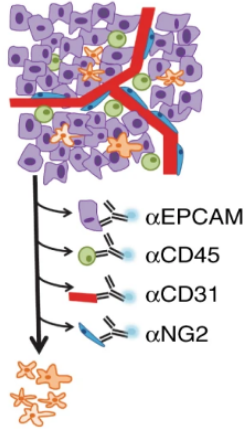
Nature Communications **9**, Article number: 5150 (2018) | Cite this article

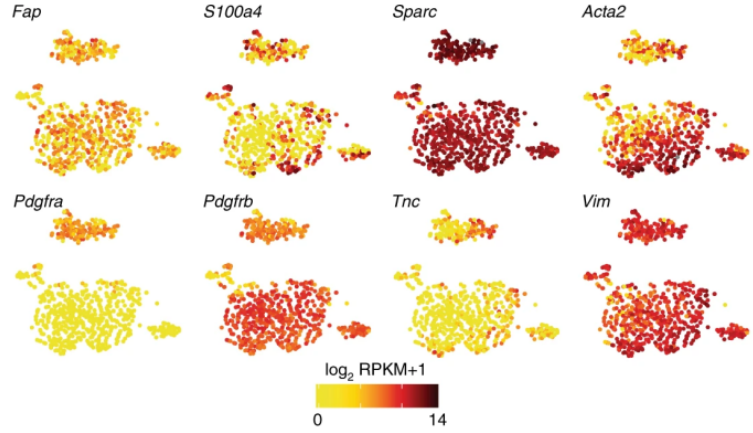
Abstract

Cancer-associated fibroblasts (CAFs) are a major constituent of the tumor microenvironment, although their origin and roles in shaping disease initiation, progression and treatment response remain unclear due to significant heterogeneity. Here, following a negative selection strategy combined with single-cell RNA sequencing of 768 transcriptomes of mesenchymal cells from a genetically engineered mouse model of breast cancer, we define three distinct subpopulations of CAFs. Validation at the transcriptional and protein level in several experimental models of cancer and human tumors reveal spatial separation of the CAF subclasses attributable to different origins, including the peri-vascular niche, the mammary fat pad and the transformed epithelium. Gene profiles for each CAF subtype correlate to distinctive functional programs and hold independent prognostic capability in

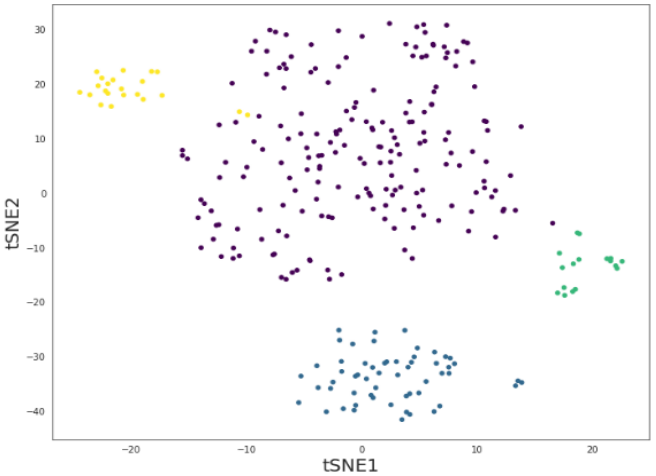


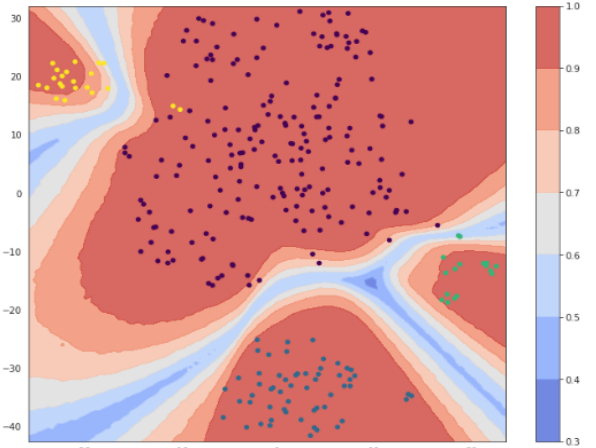


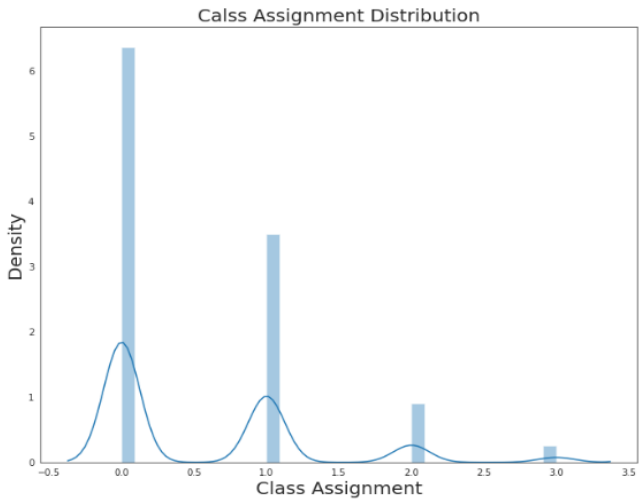


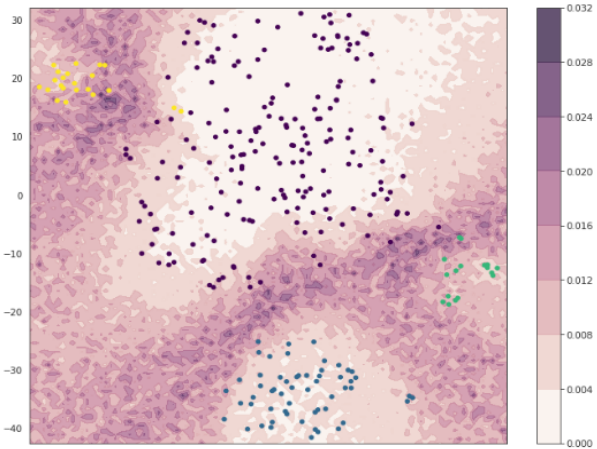


Bayesian deep learning for single cell









Bartoschek, Oskolkov et al., Nature Communications 2018

1) handy for unseen (during training) data, 2) uncertainties are crucial for clinical diagnostics

Session summary

Take home messages of the session:

1) Bayesian learning is important for life-death predictions

2) Bayesian fitting generates a family (not single) of curves

3) Adding priors to weights converts a ANN to a Bayesian ANN

4) Bayesian deep learning used for predictions with uncertainty

National Bioinformatics Infrastructure Sweden (NBIS)

Knut och Alice Wallenbergs Stiftelse

Vetenskapsrådet

Lunds Universitet